Red Hat Integration 2023.Q4

Debezium 用户指南

用于 Red Hat Integration 2.3.4

Last Updated: 2024-04-11

Red Hat Integration 2023.Q4 Debezium 用户指南

用于 Red Hat Integration 2.3.4

法律通告

摘要

本指南论述了如何使用 Red Hat Integration 提供的连接器。

目录

前言

Debezium 是一组分布式服务，用于捕获数据库中的行级更改，以便您的应用程序能够查看和响应这些更改。Debezium 记录提交到每个数据库表的所有行级更改。每个应用程序都会读取感兴趣的事务日志，以按照发生的顺序查看所有操作。

本指南提供有关使用以下 Debezium 主题的详细信息：

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

要改进，创建一个 JIRA 问题并描述您推荐的更改。提供尽可能多的详细信息，以便我们快速解决您的请求。

前提条件

- 您有一个红帽客户门户网站帐户。此帐户可让您登录到 Red Hat Jira Software 实例。
  如果您没有帐户，系统会提示您创建一个帐户。

流程

1. 单击以下链接： 创建问题。

2. 在 **Summary** 文本框中输入问题的简短描述。
3. 在 **Description** 文本框中提供以下信息：
   - 找到此问题的页面的 URL。
   - 有关此问题的详细描述。
     您可以将信息保留在任何其他字段中的默认值。
4. 点 **Create** 将 JIRA 问题提交到文档团队。

感谢您花时间来提供反馈。

第 1 章 DEBEZIUM 的高级别概述

Debezium 是一组分布式服务，用于捕获数据库中的更改。您的应用程序可以使用并响应这些更改。Debezium 会在更改事件记录中的每个数据库表中捕获每行级别的更改，并将这些记录流传输到 Kafka 主题。应用程序读取这些流，其以与生成的顺序提供更改事件记录。

更多详情位于以下部分中：

- 第 1.1 节 “Debezium 功能”
- 第 1.2 节 “Debezium 架构的描述”

1.1. DEBEZIUM 功能

Debezium 是 Apache Kafka Connect 的一组源连接器。每个连接器都通过使用数据库功能更改数据捕获(CDC)，从而从不同的数据库间更改。与其他方法（如轮询或双写）不同，基于日志的 CDC 由 Debezium 实施：

- 确保 **捕获所有数据更改。**
- 生成具有 **非常低延迟的** 更改事件，同时避免增加频繁轮询所需的 CPU 使用量。例如，对于 MySQL 或 PostgreSQL，延迟时间为 millisecond 范围内。
- **不需要对数据模型的更改**，如"Last Updated"列。
- 可以捕获 **删除**。
- **可以捕获旧的记录状态和其他元数据**，如事务 ID 并导致查询，具体取决于数据库的功能和配置。

基于日志更改数据捕获的五个优点 是一个博客文章，它提供了更详细的信息。

Debezium 连接器使用各种相关功能和选项捕获数据更改：

- **快照：** （可选）如果连接器启动，且并非所有日志仍然存在，则可以进行数据库当前状态的初始快照。通常，当数据库有一段时间运行时，会出现这种情况，并丢弃事务日志，不再需要事务恢复或复制。执行快照有不同的模式，包括支持 *增量* 快照，可以在连接器运行时触发。如需了解更多详细信息，请参阅您使用的连接器的文档。
- filters ：您可以使用 include/exclude 列表过滤器配置捕获的模式、表和列的集合。
- **masking** ：可以屏蔽特定列中的值，例如，当它们包含敏感数据时。
- **监控：** 大多数连接器都可使用 JMX 监控。
- 随时可用的 **单个消息转换(SMT)** 用于消息路由、过滤、事件扁平化等等。有关 Debezium 提供的 SMT 的更多信息，请参阅 应用转换来修改使用 Apache Kafka 交换的消息 。

每个连接器的文档提供了有关连接器功能和配置选项的详细信息。

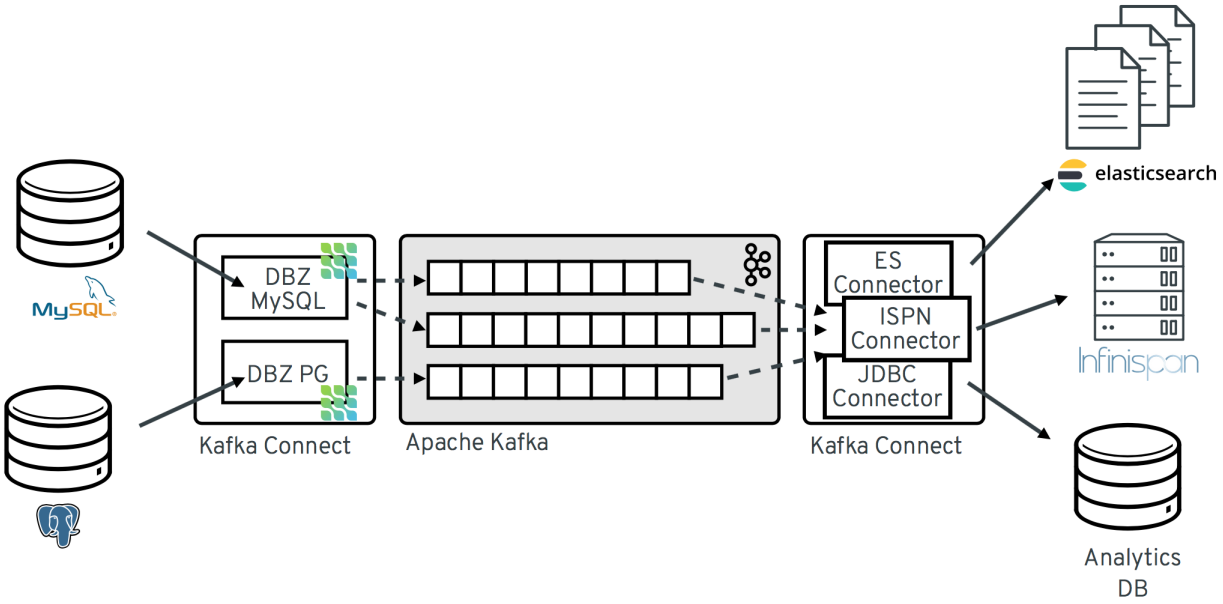
1.2. DEBEZIUM 架构的描述

您可以使用 Apache Kafka Connect 部署 Debezium。Kafka Connect 是一个用于实施和操作的框架和运行时：

- 源连接器，如将记录发送到 Kafka 的 Debezium

- 将记录从 Kafka 主题传播到其他系统的接收器连接器

下图显示了基于 Debezium 的更改数据捕获管道的架构：



如镜像中所示，部署了 MySQL 和 PostgresSQL 的 Debezium 连接器，以捕获对这两种类型的数据库的更改。每个 Debezium 连接器建立与源数据库的连接：

- MySQL 连接器使用客户端库来访问 **binlog**。
- PostgreSQL 连接器从逻辑复制流读取。

Kafka Connect 作为 Kafka 代理之外的独立服务运行。

默认情况下，从一个数据库表的更改写入 Kafka 主题，其名称对应于表名称。如果需要，您可以通过配置 Debezium 的主题 路由转换来调整目标主题名称。例如，您可以：

- 将记录路由到名称与表名称不同的主题
- 流将多个表的事件记录改为单个主题

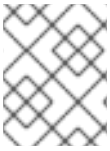
在 Apache Kafka 中更改了事件记录后，Kafka Connect eco-system 的不同连接器可以将记录流传输到其他系统和数据库，如 Elasticsearch、数据仓库和分析系统或缓存，如 Infinispan。根据所选的 sink 连接器，您可能需要配置 Debezium 的新记录状态提取 转换。此 Kafka Connect SMT 将 **after** 结构从 Debezium 的更改事件传播到接收器连接器。这是默认传播的详细更改事件记录。

第 2 章 所需的自定义资源升级

Debezium 是一个 Kafka 连接器插件，部署到在 OpenShift 上的 AMQ Streams 上运行的 Apache Kafka 集群。要准备 OpenShift CRD **v1**，在 AMQ Streams 的当前版本中，所需的自定义资源定义(CRD) API 版本现在设置为 **v1beta2**。API 的 **v1beta2** 版本替换了之前支持的 **v1beta1** 和 **v1alpha1** API 版本。对 **v1alpha1** 和 **v1beta1** API 版本的支持已在 AMQ Streams 中弃用。现在，这些早期版本已从大多数 AMQ Streams 自定义资源中删除，包括用来配置 Debezium 连接器的 KafkaConnect 和 KafkaConnector 资源。

基于 **v1beta2** API 版本的 CRD 使用 OpenAPI structural 模式。基于取代的 v1alpha1 或 v1beta1 API 的自定义资源不支持 structural 模式，并与当前版本 AMQ Streams 不兼容。在升级到 AMQ Streams2.5 之前，您必须升级现有自定义资源以使用 API 版本 **kafka.strimzi.io/v1beta2**。在升级到 AMQ Streams 1.7 后，您可以随时升级自定义资源。在升级到 AMQ Streams2.5 或更新版本前，您必须完成升级到 v1beta2 API。

为了便于升级 CRD 和自定义资源，AMQ Streams 提供了一个 API 转换工具，它会自动将其升级到与 **v1beta2** 兼容格式。有关该工具以及如何升级 AMQ Streams 的完整说明， 请参阅在 OpenShift 中部署和升级 AMQ Streams。



注意

更新自定义资源的要求只适用于在 OpenShift 上的 AMQ Streams 上运行的 Debezium 部署。要求不适用于 Red Hat Enterprise Linux 上的 Debezium

第 3 章 DB2 的 DEBEZIUM 连接器

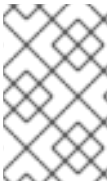
Debezium 的 Db2 连接器可以在 Db2 数据库的表中捕获行级更改。有关与此连接器兼容的 Db2 数据库版本的详情，请查看 Debezium 支持的配置页面。

这个连接器主要由 SQL Server 的 Debezium 实现所发，它使用基于 SQL 的轮询模型将表置于"capture 模式"。当表处于捕获模式时，Debezium Db2 连接器会为该表的每个行级更新生成更改事件。

处于捕获模式的表有一个关联的 change-data 表，Db2 创建它。对于处于捕获模式的表的每个更改，Db2 会将有关该更改的数据添加到表的相关 change-data 表中。change-data 表包含一行的每个状态的条目。它还具有要删除的特殊条目。Debezium Db2 连接器从 change-data 表中读取更改事件，并将事件发送到 Kafka 主题。

当 Debezium Db2 连接器第一次连接到 Db2 数据库时，连接器会读取连接器配置为捕获更改的表的一致性快照。默认情况下，这是所有非系统表。有连接器配置属性，允许您指定将哪些表放入捕获模式，或者从捕获模式中排除哪些表。

快照完成后，连接器开始向捕获模式的表发送更改事件。默认情况下，更改特定表的事件会进入与表名称相同的 Kafka 主题。应用程序和服务会消耗这些主题的更改事件。



注意

连接器需要使用抽象语法表示法(ASN)库，该库作为 Linux 的 Db2 标准部分提供。要使用 ASN 库，您必须有用于 IBM InfoSphere 数据复制(IIDR)的许可证。您不必安装 IIDR 来使用 ASN 库。

使用 Debezium Db2 连接器的信息和步骤进行组织，如下所示：

3.1. DEBEZIUM DB2 连接器概述

Debezium Db2 连接器基于在 Db2 中启用 SQL Replication 的 ASN Capture/Apply 代理。捕获代理：

- 为处于捕获模式的表生成 change-data 表。
- 以捕获模式监控表，并在对应的 change-data 表中存储这些表的更改事件。

Debezium 连接器使用 SQL 接口来查询 change-data 表以更改事件。

数据库管理员必须将更改捕获为捕获模式的表。为方便起见，在 C 中提供了 Debezium 管理用户定义的功能(UDF)，您可以编译，然后使用以下管理任务：

- 启动、停止和重新初始化 ASN 代理
- 将表置于捕获模式
- 创建复制(ASN)模式和 change-data 表
- 从捕获模式中删除表

或者，您可以使用 Db2 控制命令来完成这些任务。

在感兴趣的表采用捕获模式后，连接器会读取对应的 change-data 表，以获取表更新的更改事件。连接器会向与更改表的名称相同的 Kafka 主题，为每个行级插入、更新和删除操作发送更改事件。这是您可以修改的默认行为。客户端应用程序读取与感兴趣的数据库表对应的 Kafka 主题，并可对每行更改事件做出反应。

通常，数据库管理员会将表置于捕获模式，并在表的生命周期内出现。这意味着连接器没有对表进行的所有更改的完整历史记录。因此，当 Db2 连接器首先连接到特定的 Db2 数据库时，它首先为捕获模式的每个表执行 *一致的快照*。连接器完成快照后，连接器流会从进行快照的时间点更改事件。这样，连接器以捕获模式的表的一致性视图开始，且不会丢弃执行快照时所做的任何更改。

Debezium 连接器可以接受故障。当连接器读取和生成更改事件时，它会记录 change-data 表条目的日志序列号(LSN)。LSN 是数据库日志中更改事件的位置。如果连接器因任何原因停止，包括通信故障、网络问题或崩溃，在重启时继续读取其关闭的 change-data 表。这包括快照。也就是说，如果在连接器停止时快照没有完成，重启连接器会开始新的快照。

3.2. DEBEZIUM DB2 连接器如何工作

为了优化配置和运行 Debezium Db2 连接器，了解连接器如何执行快照、流更改事件、决定 Kafka 主题名称并处理模式更改。

详情包括在以下主题中：

- 第 3.2.1 节 “Debezium Db2 连接器如何执行数据库快照”
- 第 3.2.2 节 “临时快照”
- 第 3.2.3 节 “增量快照”
- 第 3.2.4 节 “Debezium Db2 连接器如何读取 change-data 表”
- 第 3.2.5 节 “接收 Debezium Db2 更改事件记录的默认 Kafka 主题名称”
- 第 3.2.7 节 “关于 Debezium Db2 连接器模式更改主题”
- 第 3.2.8 节 “Debezium Db2 连接器生成的事件代表事务边界”

3.2.1. Debezium Db2 连接器如何执行数据库快照

Db2 的复制功能不旨在存储数据库更改的完整历史记录。因此，Debezium Db2 连接器无法从日志检索数据库的完整历史记录。要让连接器为数据库的当前状态建立基线，连接器首次启动时，它会执行 捕获模式 中的表的初始 *一致快照*。对于快照捕获的每个更改，连接器会向捕获的表的 Kafka 主题发送一个 **读取** 事件。

您可以在以下部分找到有关快照的更多信息：

- 第 3.2.2 节 “临时快照”
- 第 3.2.3 节 “增量快照”

Debezium Db2 连接器用来执行初始快照的默认工作流

以下工作流列出了 Debezium 创建快照所采取的步骤。这些步骤描述了当 **snapshot.mode** 配置属性设置为其默认值时（即 **的初始** ）时快照的流程。您可以通过更改 **snapshot.mode** 属性的值来自定义连接器创建快照的方式。如果您配置不同的快照模式，连接器使用这个工作流的修改版本完成快照。

1. 建立与数据库的连接。
2. 确定哪个表处于捕获模式，并应包含在快照中。默认情况下，连接器捕获所有非系统表的数据。快照完成后，连接器将继续流传输指定表的数据。如果您希望连接器只从特定表捕获数据，您可以通过设置 **table.include.list** 或 **table.exclude.list** 等属性来只捕获表或表元素子集的数据。
3. 在捕获模式下的每个表上获取锁定。这个锁定可确保在快照完成前，这些表中不会发生模式更改。锁定的级别由 **snapshot.isolation.mode** 连接器配置属性决定。
4. 在服务器的事务日志中读取最高（最新）LSN 位置。
5. 捕获所有表的模式，或指定为捕获的所有表。连接器在其内部数据库模式历史记录主题中保留模式信息。架构历史记录提供有关发生更改事件时生效的结构的信息。

注意

默认情况下，连接器捕获数据库中每个表的模式，这些模式处于捕获模式，包括没有配置为捕获的表。如果没有为捕获配置表，则初始快照只捕获其结构；它不会捕获任何表数据。

有关为什么没有包括在初始快照中的表的快照保留模式信息，请参阅 了解为什么初始快照捕获所有表的 schema。

6. 释放在第 3 步中获得的任何锁定。其他数据库客户端现在可以写入任何之前锁定的表。
7. 在 LSN 分步读取时，连接器会扫描为捕获而指定的表。在扫描过程中，连接器完成以下任务：
   a. 确认表已在快照开始前创建。如果表是在快照启动后创建的，连接器会跳过表。快照完成后，连接器过渡到 streaming，它会发出快照开始后创建的任何表的更改事件。
   b. 为从表获取的每行生成 **读取** 事件。所有 **读取** 事件都包含相同的 LSN 位置，这是在第 4 步中获取的 LSN 位置。
   c. 将每个 **读取** 事件发送到源表的 Kafka 主题。
   d. 释放数据表锁定（如果适用）。
8. 在连接器偏移中记录快照成功完成。

生成的初始快照捕获捕获捕获的表中每行的当前状态。在这个基准状态中，连接器会捕获后续更改。

在快照进程开始后，如果进程因为连接器失败、重新平衡或其他原因而中断，则进程会在连接器重启后重启。

连接器完成初始快照后，它会继续从在第 4 步中读取的位置进行流，使其不会错过任何更新。

如果连接器因为任何原因而再次停止，它会在重启后从之前关闭的位置恢复流更改。

3.2.1.1. 初始快照捕获所有表的 schema 历史记录的描述

连接器运行的初始快照捕获两种类型的信息：

表数据

在连接器的 **table.include.list** 属性中命名的表中的 **INSERT**、**UPDATE** 和 **DELETE** 操作的信息。

模式数据

描述应用到表的结构更改的 DDL 语句。模式数据会保留给内部模式历史记录主题，以及连接器的 schema 更改主题（如果配置了）。

运行初始快照后，您可能会注意到快照捕获没有指定用于捕获的表的模式信息。默认情况下，初始快照旨在捕获数据库中存在的每个表的模式信息，而不仅仅是从指定为捕获的表的表。连接器要求表的模式存在于架构历史记录主题中，然后才能捕获表。通过启用初始快照来捕获不是原始捕获集一部分的表的 schema 数据，Debebe 准备好连接器，以便稍后需要捕获这些表中的事件数据。如果初始快照没有捕获表的 schema，您必须将模式添加到历史记录主题，然后才能从表中捕获数据。

在某些情况下，您可能想要限制初始快照中的模式捕获。当您要减少完成快照所需的时间时，这非常有用。或者，当 Debezium 通过可访问多个逻辑数据库的用户帐户连接到数据库实例时，但您希望连接器只从特定逻辑数据库中的表捕获更改。

附加信息

- 从不是由初始快照捕获的表捕获数据（没有模式更改）
- 从不是由初始快照捕获的表捕获数据（应用程序更改）
- 设置 **schema.history.internal.store.only.captured.tables.ddl** 属性，以指定从中捕获模式信息的表。
- 设置 **schema.history.internal.store.only.captured.databases.ddl** 属性，以指定从中捕获模式更改的逻辑数据库。

3.2.1.2. 从不是由初始快照捕获的表捕获数据（没有模式更改）

在某些情况下，您可能希望连接器从其模式未被初始快照捕获的表中捕获数据。根据连接器配置，初始快照只能捕获数据库中特定表的表模式。如果历史记录主题中没有表模式，连接器将无法捕获表，并报告缺少的 schema 错误。

您可能仍然能够从表中捕获数据，但您必须执行额外的步骤来添加表模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。
- 没有模式更改应用于连接器读取的 LSN 和最新更改表条目之间的表。有关从具有结构性更改的新表中捕获数据的详情，请参考 第 3.2.1.3 节 “从不是由初始快照捕获的表捕获数据（应用程序更改）”。

流程

1. 停止连接器。

2. 删除由 schema.history.internal. **kafka.topic 属性指定的内部数据库架构历史记录** 主题。

3. 清除配置的 Kafka Connect **offset.storage.topic** 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 对连接器配置应用以下更改：

   a. （可选）将 **schema.history.internal.captured.tables.ddl** 的值设置为 **false**。此设置会导致快照捕获所有表的 schema，并保证以后可以重建所有表的 schema 历史记录。

   注意

   捕获所有表的架构的快照需要更多时间来完成。

   b. 添加您希望连接器捕获至 **table.include.list** 的表。

   c. 将 **snapshot.mode** 设置为以下值之一：

   初始

   重启连接器时，它会获取捕获表数据和表结构的数据库的完整快照。
   如果您选择这个选项，请考虑将 **schema.history.internal.captured.tables.ddl** 属性的值设置为 **false**，以便连接器捕获所有表的 schema。

   schema_only

   重启连接器时，它会获取仅捕获表模式的快照。与完整数据快照不同，这个选项不会捕获任何表数据。如果 ou 想要更快地重启连接器，则使用此选项，而不是完整快照。

5. 重启连接器。连接器完成 **snapshot.mode** 指定的快照类型。

6. （可选）如果连接器执行了 **schema_only** 快照，在快照完成后，启动一个增量快照来 从您添加的表中捕获数据。连接器在继续从表中实时更改时运行快照。运行增量快照可捕获以下数据更改：

   - 对于之前捕获的连接器的表，增量 snapsot 捕获连接器停机时所发生的变化，即在连接器停止和当前重启之间的时间间隔。
   - 对于新添加的表，增量快照会捕获所有现有表行。

3.2.1.3. 从不是由初始快照捕获的表捕获数据（应用程序更改）

如果架构更改应用到表，则在架构更改前提交的记录与更改后提交的不同结构不同。当 Debezium 从表中捕获数据时，它会读取 schema 历史记录，以确保它为每个事件应用正确的模式。如果 schema 历史记录主题中没有 schema，则连接器无法捕获表，并出现错误结果。

如果要从初始快照捕获的表中捕获数据，并且修改了表的 schema，则必须将模式添加到历史记录主题中（如果它还没有可用）。您可以通过运行新的模式快照或运行表的初始快照来添加模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。
- 架构更改应用于表，以便捕获的记录没有统一结构。

流程

初始快照捕获了所有表的模式(storage.only.captured.tables.ddl 设置为 false)

1. 编辑 **table.include.list** 属性，以指定您要捕获的表。
2. 重启连接器。
3. 如果要从新添加的表中捕获现有数据，则启动 增量快照。

初始快照没有捕获所有表的模式(storage.only.captured.tables.ddl 设置为 true)

如果初始快照没有保存您要捕获的表的模式，请完成以下步骤之一：

流程 1：架构快照，后跟增量快照

在此过程中，连接器首先执行 schema 快照。然后，您可以启动增量快照，使连接器能够同步数据。

1. 停止连接器。
2. 删除由 schema.history.internal.**kafka.topic 属性指定的内部数据库架构历史记录** 主题。
3. 清除配置的 Kafka Connect **offset.storage.topic** 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

> **警告**
>
> 删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 为连接器配置中的属性设置值，如以下步骤所述：
   a. 将 **snapshot.mode** 属性的值设置为 **schema_only**。
   b. 编辑 **table.include.list** 以添加您要捕获的表。
5. 重启连接器。
6. 等待 Debezium 捕获新表和现有表的模式。在连接器停止后发生任何表的数据更改不会被捕获。
7. 为确保没有丢失数据，请启动 增量快照。

步骤 2：初始快照，后跟可选的增量快照

在此过程中，连接器执行数据库的完整初始快照。与任何初始快照一样，在具有多个大型表的数据库中，运行初始快照可能会非常耗时。快照完成后，您可以选择触发增量快照来捕获连接器离线时发生的任何更改。

1. 停止连接器。
2. 删除由 schema.history.internal.**kafka.topic 属性指定的内部数据库架构历史记录** 主题。
3. 清除配置的 Kafka Connect **offset.storage.topic** 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 编辑 **table.include.list** 以添加您要捕获的表。
5. 为连接器配置中的属性设置值，如以下步骤所述：
   a. 将 **snapshot.mode** 属性的值设置为 **initial**。
   b. （可选）将 **schema.history.internal.store.only.captured.tables.ddl** 设置为 **false**。
6. 重启连接器。连接器获取完整的数据库快照。快照完成后，连接器会过渡到 streaming。
7. （可选）要捕获连接器离线时更改的任何数据，请启动 增量快照。

3.2.2. 临时快照

默认情况下，连接器仅在首次启动后运行初始快照操作。在正常情况下，在这个初始快照后，连接器不会重复快照过程。连接器捕获的任何更改事件数据都只通过流处理。

然而，在某些情况下，连接器在初始快照期间获得的数据可能会过时、丢失或不完整。为了提供总结表数据的机制，Debezium 包含一个执行临时快照的选项。数据库中的以下更改可能会导致执行临时快照：

- 连接器配置会被修改为捕获不同的表集合。
- Kafka 主题已删除，必须重建。
- 由于配置错误或某些其他问题导致数据损坏。

您可以通过启动所谓的 *临时快照来为之前捕获的表重新运行快照*。临时快照需要使用 信号表。您可以通过向 Debezium 信号表发送信号请求来发起临时快照。

当您启动现有表的临时快照时，连接器会将内容附加到表已存在的主题中。如果删除了之前存在的主题，如果启用了 自动主题创建，Debezium 可以自动创建主题。

临时快照信号指定要包含在快照中的表。快照可以捕获整个数据库的内容，或者仅捕获数据库中表的子集。另外，快照也可以捕获数据库中表的内容子集。

您可以通过将 **execute-snapshot** 消息发送到信号表来指定要捕获的表。将 **execute-snapshot** 信号类型设置为 **增量**，并提供快照中包含的表名称，如下表所述：

表 3.1. 临时 execute-snapshot 信号记录的示例

字段	默认	值
type	**incremental**	指定您要运行的快照类型。 设置类型是可选的。目前，您只能请求 **增量** 快照。
data-collections	*不适用*	包含与要快照的表的完全限定域名匹配的正则表达式的数组。 名称的格式与 **signal.data.collection** 配置选项的格式相同。
additional-condition	*不适用*	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。
surrogate-key	*N/A*	可选字符串，指定连接器在快照过程中用作表的主键的列名称。

触发临时快照

您可以通过向信号表中添加 **execute-snapshot** 信号类型的条目来发起临时快照。连接器处理消息后，它会开始快照操作。快照进程读取第一个和最后一个主密钥值，并使用这些值作为每个表的开头和结束点。根据表中的条目数量以及配置的块大小，Debezium 会将表划分为块，并一次性执行每个块的快照。

目前，**execute-snapshot** 操作类型仅触发 增量快照。如需更多信息，请参阅 增加快照。

3.2.3. 增量快照

为了提供管理快照的灵活性，Debezium 包含附加快照机制，称为 *增量快照*。增量快照依赖于 Debezium 机制 向 Debezium 连接器发送信号。

在增量快照中，除了一次捕获数据库的完整状态，就像初始快照一样，Debebe 会在一系列可配置的块中捕获每个表。您可以指定您希望快照捕获的表 以及每个块的大小。块大小决定了快照在数据库的每个获取操作期间收集的行数。增量快照的默认块大小为 1024 行。

当增量快照进行时，Debebe 使用 watermarks 跟踪其进度，维护它捕获的每个表行的记录。与标准初始快照过程相比，捕获数据的阶段方法具有以下优点：

- 您可以使用流化数据捕获并行运行增量快照，而不是在快照完成前进行后流。连接器会在快照过程中从更改日志中捕获接近实时事件，且操作都不会阻止其他操作。
- 如果增量快照的进度中断，您可以在不丢失任何数据的情况下恢复它。在进程恢复后，快照从停止的点开始，而不是从开始计算表。
- 您可以随时根据需要运行增量快照，并根据需要重复该过程以适应数据库更新。例如，您可以在修改连接器配置后重新运行快照，以将表添加到其 **table.include.list** 属性中。

增量快照过程

当您运行增量快照时，Debezium 会按主键对每个表进行排序，然后根据 配置的块大小 将表分成块。然后，按块的工作块会捕获块中的每个表行。对于它捕获的每行，快照会发出 **READ** 事件。该事件代表块的快照开始时的行值。

当快照继续进行时，其他进程可能会继续访问数据库，可能会修改表记录。为了反映此类更改，**INSERT**、**UPDATE** 或 **DELETE** 操作会按照常常提交到事务日志。同样，持续 Debezium 流进程将继续检测这些更改事件，并将相应的更改事件记录发送到 Kafka。

Debezium 如何使用相同的主密钥在记录间解决冲突

在某些情况下，streaming 进程发出的 **UPDATE** 或 **DELETE** 事件会停止序列。也就是说，流流过程可能会发出一个修改表行的事件，该事件捕获包含该行的 **READ** 事件的块。当快照最终为行发出对应的 **READ** 事件时，其值已被替换。为确保以正确的逻辑顺序处理到达序列的增量快照事件，Debebe 使用缓冲方案来解析冲突。仅在快照事件和流化事件之间发生冲突后，De Debezium 会将事件记录发送到 Kafka。

快照窗口

为了帮助解决修改同一表行的后期事件和流化事件之间的冲突，Debebe 会使用一个所谓的 *快照窗口*。**快照窗口分解了增量快照捕获指定表块数据的间隔。在块的快照窗口打开前，Debebe 会使用其常见行为，并将事件从事务日志直接下游发送到目标 Kafka 主题。但从特定块的快照打开后，直到关闭为止，De-duplication 步骤会在具有相同主密钥的事件之间解决冲突。**

对于每个数据收集，**Debezium** 会发出两种类型的事件，并将其存储在单个目标 **Kafka** 主题中。从表直接捕获的快照记录作为 **READ** 操作发送。同时，当用户继续更新数据收集中的记录，并且会更新事务日志来反映每个提交，**Debezium** 会为每个更改发出 **UPDATE** 或 **DELETE** 操作。

当快照窗口打开时，**Debezium** 开始处理快照块，它会向内存缓冲区提供快照记录。在快照窗口期间，缓冲区中 **READ** 事件的主密钥与传入流事件的主键进行比较。如果没有找到匹配项，则流化事件记录将直接发送到 **Kafka**。如果 **Debezium** 检测到匹配项，它会丢弃缓冲的 **READ** 事件，并将流化记录写入目标主题，因为流的事件逻辑地取代静态快照事件。在块关闭的快照窗口后，缓冲区仅包含 **READ** 事件，这些事件不存在相关的事务日志事件。**Debezium** 将这些剩余的 **READ** 事件发送到表的 **Kafka** 主题。

连接器为每个快照块重复这个过程。

警告

Db2 的 Debezium 连接器不支持增量快照运行时的模式更改。

3.2.3.1. 触发增量快照

目前，启动增量快照的唯一方法是向源数据库上的 信号表发送临时快照 信号。

作为 SQL INSERT 查询，您将向信号提交信号。

在 Debezium 检测到信号表中的更改后，它会读取信号并运行请求的快照操作。

您提交的查询指定要包含在快照中的表，并可以选择指定快照操作的类型。目前，快照操作的唯一有效选项是默认值 incremental。

要指定快照中包含的表，请提供列出表或用于匹配表的正则表达式数组的 数据集合，例如：

```
{"data-collections": ["public.MyFirstTable", "public.MySecondTable"]}
```

增量快照信号的 data-collections 数组没有默认值。如果 data-collections 数组为空，Debezium 会检测到不需要任何操作，且不会执行快照。

注意

如果要包含在快照中的表的名称在数据库、模式或表的名称中包含句点(.)，以将表添加到 data-collections 数组中，您必须使用双引号转义名称的每个部分。

例如，要包含一个存在于 公共 模式的表，其名称为 My.Table，请使用以下格式：`"public"."My.Table "`。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道来触发增量快照

1. 发送 SQL 查询，将临时增量快照请求添加到信号表中：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) 1
values ('ad-hoc-1', 2
  'execute-snapshot', 3
  '{"data-collections": ["schema1.table1", "schema2.table2"], 4
  "type":"incremental"}, 5
  "additional-condition":"color=blue"}'); 6
```

命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 3.2. SQL 命令中字段的描述，用于将增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。 使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。相反，Debebe 会在快照期间生成自己的 **id** 字符串作为水位线信号。
3	**execute-snapshot**	**type** 参数指定信号旨在触发的操作。
4	**data-collections**	信号的 **data** 字段所需的组件，用于指定表名称或正则表达式数组，以匹配快照中包含的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。
5	**incremental**	信号的 **data** 字段的可选 **类型** 组件，用于指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 如果没有指定值，连接器将运行增量快照。

项	值	描述
6	**additional-condition**	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。有关 **additional-condition** 参数的更多信息，请参阅 带有额外条件的临时增量快照。

带有额外条件的临时增量快照

如果您希望快照只包含表中的内容子集，您可以通过向快照信号附加 additional-condition 参数来修改信号请求。

典型的快照的 SQL 查询采用以下格式：

```
SELECT * FROM <tableName> ....
```

通过添加 additional-condition 参数，您可以将 WHERE 条件附加到 SQL 查询中，如下例所示：

```
SELECT * FROM <tableName> WHERE <additional-condition> ....
```

以下示例显示了向信号表发送带有额外条件的临时增量快照请求的 SQL 查询：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，假设您有一个包含以下列的 products 表：

- **ID （主键）**
- **color**
- **quantity**

如果您需要 product 表的增量快照，其中只包含 color=blue 的数据项，您可以使用以下 SQL 语句来触发快照：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue"}');
```

additional-condition 参数还允许您传递基于多个列的条件。例如，使用上例中的 product 表，您可以提交查询来触发增量快照，该快照仅包含 color=blue 和 quantity>10 的项数据：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue AND quantity>10"}');
```

以下示例显示了连接器捕获的增量快照事件的 JSON。

示例：增加快照事件消息

```
{
  "before":null,
  "after": {
    "pk":"1",
    "value":"New data"
  },
  "source": {
    ...
    "snapshot":"incremental" 1
  },
  "op":"r", 2
  "ts_ms":"1620393591654",
  "transaction":null
}
```

项	字段名称	描述
1	**snapshot**	指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 在 SQL 查询中指定 **type** 值，您提交到信号表是可选的。 如果没有指定值，连接器将运行增量快照。
2	**op**	指定事件类型。 快照事件的值是 **r**，表示 **READ** 操作。

3.2.3.2. 使用 Kafka 信号频道来触发增量快照

您可以向 配置的 Kafka 主题 发送消息，以请求连接器来运行临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 execute-snapshot，data 字段必须具有以下字段：

表 3.3. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与快照中包含的表的完全限定域名匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的列子集的条件。

execute-snapshot Kafka 消息示例：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

带有额外条件的临时增量快照

Debezium 使用 additional-condition 字段来选择表内容的子集。

通常，当 Debezium 运行快照时，它会运行 SQL 查询，例如：

SELECT * FROM *<tableName>* ....

当快照请求包含 additional-condition 时，extra-condition 会附加到 SQL 查询中，例如：

SELECT * FROM *<tableName>* WHERE *<additional-condition>* ....

例如，如果一个 product table with the column id （主键）、color 和 brand，如果您希望快照只包含 color='blue' 的内容，当您请求快照时，您可以附加一个 additional-condition 语句来过滤内容：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue'"}}`
```

您可以使用 additional-condition 语句根据多个列传递条件。例如，如果您希望快照只包含 color='blue' 的 products 表中，以及 brand='MyBrand'，则您可以发送以下请求：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}`
```

3.2.3.3. 停止增量快照

您还可以通过向源数据库上的表发送信号来停止增量快照。您可以通过发送 SQL INSERT 查询向表提交停止快照信号。

在 Debezium 检测到信号表中的更改后，它会读取信号，并在正在进行时停止增量快照操作。

您提交的查询指定 增量 的快照操作，以及要删除的当前运行快照的表。

先决条件

- **启用了信号。**
  - **源数据库中存在信号数据收集**

源数据库中存在信号数据收集。

- 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道停止增量快照

1. 发送 SQL 查询以停止临时增量快照到信号表：

```
INSERT INTO <signalTable> (id, type, data) values ('<id>', 'stop-snapshot', '{"data-collections": ["<tableName>","<tableName>"],"type":"incremental"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) 1
values ('ad-hoc-1', 2
  'stop-snapshot', 3
  '{"data-collections": ["schema1.table1", "schema2.table2"], 4
  "type":"incremental"}'); 5
```

signal 命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 3.4. SQL 命令中字段的描述，用于将停止增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。
3	**stop-snapshot**	指定 **type** 参数指定信号要触发的操作。
4	**data-collections**	信号的 **data** 字段的可选组件，用于指定表名称或正则表达式数组，以匹配要从快照中删除的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。如果省略了 **data** 字段的这一组件，信号将停止正在进行的整个增量快照。

项	值	描述
5	**incremental**	信号的 **data** 字段所需的组件，用于指定要停止的快照操作类型。 目前，唯一有效的选项是 **增量的**。 如果没有指定 **类型** 值，信号将无法停止增量快照。

3.2.3.4. 使用 Kafka 信号频道停止增量快照

您可以将信号消息发送到 配置的 Kafka 信号主题，以停止临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 stop-snapshot，data 字段必须具有以下字段：

表 3.5. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	可选数组，以逗号分隔的正则表达式，与表的完全限定域名匹配，以包含在快照中。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

以下示例显示了典型的 stop-snapshot Kafka 信息：

```
Key = `test_connector`

Value = `{"type":"stop-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

3.2.4. Debezium Db2 连接器如何读取 change-data 表

在完成快照后，当 Debezium Db2 连接器首次启动时，连接器会标识每个处于捕获模式的源表的 change-data 表。连接器对每个 change-data 表执行以下操作：

1. 读取在上一次存储、最高 LSN 和当前最高 LSN 中创建的更改事件。

2. 根据提交 LSN 和每个事件的更改 LSN，对更改事件进行排序。这样可确保连接器按表更改的顺序发出更改事件。

3. 将提交并更改 LSN 作为偏移到 Kafka Connect。

4. 存储传递给 Kafka Connect 的连接器的最高 LSN。

重启后，连接器会从离开的偏移（提交并更改 LSN）发出更改事件。当连接器正在运行并发出更改事件时，如果您从捕获模式中删除表或向捕获模式添加表，连接器会检测到更改并相应地修改其行为。

3.2.5. 接收 Debezium Db2 更改事件记录的默认 Kafka 主题名称

默认情况下，Db2 连接器会将表中的所有 INSERT、UPDATE 和 DELETE 操作的更改事件写入特定于该表的单一 Apache Kafka 主题。连接器使用以下惯例来命名更改事件主题：

topicPrefix.schemaName.tableName

以下列表为默认名称的组件提供定义：

topicPrefix

由 topic.prefix 连接器配置属性指定的主题前缀。

schemaName

操作所在的模式的名称。

tableName

操作所在的表的名称。

例如，一个使用 mydatabase 数据库的 Db2 安装，其中包含四个表：PRODUCTS,

PRODUCTS_ON_HAND, CUSTOMERS, 和 ORDERS，它们包括在 MYSCHEMA schema 中。连接器会将事件发送到这四个 Kafka 主题：

- mydatabase.MYSCHEMA.PRODUCTS
- mydatabase.MYSCHEMA.PRODUCTS_ON_HAND
- mydatabase.MYSCHEMA.CUSTOMERS
- mydatabase.MYSCHEMA.ORDERS

连接器应用类似的命名约定，以标记其内部数据库架构历史记录主题、架构更改主题 和事务元数据主题。

如果默认主题名称不满足您的要求，您可以配置自定义主题名称。要配置自定义主题名称，您可以在逻辑主题路由 SMT 中指定正则表达式。有关使用逻辑主题路由 SMT 来自定义主题命名的更多信息，请参阅 主题路由。

3.2.6. Debezium Db2 连接器如何处理数据库架构更改

当数据库客户端查询数据库时，客户端将使用数据库的当前架构。但是，数据库模式可以随时更改，这意味着连接器必须能够识别每个插入、更新或删除操作被记录的时间。另外，连接器不一定将当前的模式应用到每个事件。如果事件相对旧，则应用当前模式之前可能会记录该事件。

为确保在 schema 更改后正确处理事件，Debezium Db2 连接器根据 Db2 更改数据表的结构存储新模式的快照，它反映了其相关数据表的结构。连接器在数据库 schema 历史记录 Kafka 主题中存储表 schema 信息，以及结果更改 LSN。连接器使用存储的 schema 表示来生成更改事件，这些事件在每次插入、更新或删除操作时正确镜像表结构。

当连接器在崩溃或安全停止后重启时，它会从它读取的最后一个位置恢复读取 Db2 中的条目。根据连接器从数据库架构历史记录主题读取的 schema 信息，连接器应用存在于连接器重启的位置上的表结构。

如果您更新处于捕获模式的 Db2 表的 schema，您也务必要更新对应更改表的模式。您必须是一个具有升级权限的 Db2 数据库管理员，才能更新数据库架构。有关如何在 Debezium 环境中更新 Db2 数据库模式的更多信息，请参阅 架构历史记录 eveolution。

数据库架构历史记录主题仅用于内部连接器。另外，连接器也可以将 模式更改事件发送到用于消费者应用程序的不同主题。

其他资源

- 接收 Debezium 事件记录 的主题的默认名称。

3.2.7. 关于 Debezium Db2 连接器模式更改主题

您可以配置 Debezium Db2 连接器来生成模式更改事件，该事件描述了应用到数据库中表的架构更改。

Debezium 在以下情况下向 schema 更改主题发送一条消息：

- 新表进入捕获模式。
- 从捕获模式中删除表。
- 在 数据库架构 更新过程中，以捕获模式的表有变化。

连接器将模式更改事件写入 Kafka 模式更改主题，其名称为 < *topicPrefix&* gt;，其中 &lt *;topicPrefix* > 是 topic.prefix 连接器配置属性中指定的主题前缀。连接器发送到 schema 更改主题的消息包含一个包含以下元素的有效负载：

databaseName

将语句应用到的数据库的名称。databaseName 的值充当 message 键。

pos

语句出现在事务日志中的位置。

tableChanges

架构更改后整个表模式的结构化表示。tableChanges 字段包含一个数组，其中包含表的每个列的条目。由于结构化表示以 JSON 或 Avro 格式呈现数据，因此用户可轻松读取消息，而不必先通过 DDL 解析器处理它们。

重要

对于处于捕获模式的表，连接器不仅将模式更改的历史记录存储在 schema 更改主题中，也存储在内部数据库架构历史记录主题中。内部数据库架构历史记录主题仅用于连接器，它不适用于消耗应用程序直接使用。确保需要通知架构更改的应用程序只消耗来自 schema 更改主题的信息。

重要

切勿对数据库架构历史记录主题进行分区。要使数据库架构历史记录主题正常工作，它必须维护连接器发出的事件记录的全局顺序。

要确保主题没有在分区间分割，请使用以下方法之一为主题设置分区计数：

- 如果您手动创建数据库架构历史记录主题，请指定分区计数 1。
- 如果您使用 Apache Kafka 代理自动创建数据库 schema 历史记录主题，则会创建该主题，将 Kafka num.partitions配置选项 的值设置为 1。

警告

连接器向其架构更改主题发出的消息格式处于异常状态，并在不通知的情况下进行更改。

示例：消息发送到 Db2 连接器模式更改主题

以下示例显示了 schema 更改主题中的消息。该消息包含表模式的逻辑表示。

```
{
 "schema": {
 ...
 },
 "payload": {
  "source": {
   "version": "2.3.4.Final",
   "connector": "db2",
   "name": "db2",
```

```
  "ts_ms": 0,
  "snapshot": "true",
  "db": "testdb",
  "schema": "DB2INST1",
  "table": "CUSTOMERS",
  "change_lsn": null,
  "commit_lsn": "00000025:00000d98:00a2",
  "event_serial_no": null
},
"ts_ms": 1588252618953, 1
"databaseName": "TESTDB", 2
"schemaName": "DB2INST1",
"ddl": null, 3
"tableChanges": [ 4
 {
  "type": "CREATE", 5
  "id": "\"DB2INST1\".\"CUSTOMERS\"", 6
  "table": { 7
   "defaultCharsetName": null,
   "primaryKeyColumnNames": [ 8
    "ID"
   ],
   "columns": [ 9
    {
     "name": "ID",
     "jdbcType": 4,
     "nativeType": null,
     "typeName": "int identity",
     "typeExpression": "int identity",
     "charsetName": null,
     "length": 10,
     "scale": 0,
     "position": 1,
     "optional": false,
     "autoIncremented": false,
     "generated": false
    },
    {
     "name": "FIRST_NAME",
     "jdbcType": 12,
     "nativeType": null,
     "typeName": "varchar",
     "typeExpression": "varchar",
     "charsetName": null,
     "length": 255,
     "scale": null,
     "position": 2,
     "optional": false,
     "autoIncremented": false,
     "generated": false
    },
    {
     "name": "LAST_NAME",
     "jdbcType": 12,
     "nativeType": null,
```

```
            "typeName": "varchar",
            "typeExpression": "varchar",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 3,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "EMAIL",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "varchar",
            "typeExpression": "varchar",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 4,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          }
        ],
        "attributes": [ 10
          {
            "customAttribute": "attributeValue"
          }
        ]
      }
    }
  ]
 }
}
```

表 3.6. 向 schema 更改主题发送的消息中字段的描述

项	字段名称	描述
1	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 在源对象中，ts_ms 表示数据库中进行更改的时间。通过将 payload.source.ts_ms 的值与 payload.ts_ms 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
2	**databaseName** **schemaName**	标识包含更改的数据库和架构。
3	**ddl**	对于 Db2 连接器，始终为 **null**。对于其他连接器，此字段包含负责架构更改的 DDL。此 DDL 不适用于 Db2 连接器。
4	**tableChanges**	包含 DDL 命令生成的模式更改的一个或多个项目的数组。

项	字段名称	描述
5	**type**	描述更改的类型。该值如下之一： • **CREATE** - 已创建表 • **ALTER** - 表被修改 • **DROP** - 表被删除
6	**id**	创建、更改或丢弃的表的完整标识符。
7	**table**	代表应用更改后的表元数据。
8	**primaryKeyColumnNames**	组成表主密钥的列的列表。
9	**columns**	更改表中每个列的元数据。
10	**属性**	每个表更改的自定义属性元数据。

在连接器发送到 schema 更改主题的消息中，message 键是包含 schema 更改的数据库的名称。在以下示例中，payload 字段包含键：

```
{
  "schema": {
    "type": "struct",
    "fields": [
      {
        "type": "string",
        "optional": false,
        "field": "databaseName"
      }
    ],
    "optional": false,
    "name": "io.debezium.connector.db2.SchemaChangeKey"
  },
  "payload": {
    "databaseName": "TESTDB"
  }
}
```

3.2.8. Debezium Db2 连接器生成的事件代表事务边界

Debezium 可以生成代表事务边界的事件，并增强更改数据事件消息。

DEBEZIUM 接收事务元数据时的限制

Debezium 注册并只针对部署连接器后发生的事务接收元数据。部署连接器前发生的事务元数据不可用。

Debezium 为每个事务中的 BEGIN 和 END 分隔符生成事务边界事件。事务边界事件包含以下字段：

status

BEGIN 或 END.

id

唯一事务标识符的字符串。

ts_ms

数据源的事务边界事件(BEGIN 或 END 事件)的时间。如果数据源没有向事件时间提供 Debezium，则该字段代表 Debezium 处理事件的时间。

event_count （用于 END 事件）

事务提供的事件总数。

data_collections （用于 END 事件）

data_collection 和 event_count 元素的数组，用于指示连接器发出来自数据收集的更改的事件数量。

示例

```
{
  "status": "BEGIN",
  "id": "00000025:00000d08:0025",
  "ts_ms": 1486500577125,
  "event_count": null,
  "data_collections": null
}

{
  "status": "END",
  "id": "00000025:00000d08:0025",
  "ts_ms": 1486500577691,
  "event_count": 2,
  "data_collections": [
```

```
    {
      "data_collection": "testDB.dbo.tablea",
      "event_count": 1
    },
    {
      "data_collection": "testDB.dbo.tableb",
      "event_count": 1
    }
  ]
}
```

除非通过 topic.transaction 选项覆盖，否则连接器会将事务事件发送到 < *topic.prefix*>.transaction 主题。

数据更改事件增强

当启用事务元数据时，连接器会通过新的 transaction 字段增强更改事件 Envelope。此字段以字段复合的形式提供有关每个事件的信息：

id

唯一事务标识符的字符串。

total_order

事件在事务生成的所有事件中绝对位置。

data_collection_order

在事务发出的所有事件间，按数据收集位置。

以下是消息的示例：

```
{
  "before": null,
  "after": {
    "pk": "2",
    "aa": "1"
  },
  "source": {
...
  },
  "op": "c",
  "ts_ms": "1580390884335",
```

```
  "transaction": {
    "id": "00000025:00000d08:0025",
    "total_order": "1",
    "data_collection_order": "1"
  }
}
```

3.3. DEBEZIUM DB2 连接器数据更改事件的描述

Debezium Db2 连接器为每个行级 INSERT、UPDATE 和 DELETE 操作生成数据更改事件。每个事件包含一个键和值。键的结构和值取决于已更改的表。

Debezium 和 Kafka Connect 围绕 *事件消息的持续流* 设计。但是，这些事件的结构可能会随时间推移而改变，而用户很难处理这些事件。要解决这个问题，每个事件都包含其内容的 schema，或者如果您正在使用 schema registry，用户可以使用该模式 ID 从 registry 获取 schema。这使得每个事件都自包含。

以下框架 JSON 显示更改事件的基本四部分。但是，如何配置您选择在应用程序中使用的 Kafka Connect converter，决定更改事件中的这四个部分的表示。只有在将转换器配置为生成它时，schema 字段才会处于更改事件中。同样，只有在您配置转换器来生成它时，事件密钥和事件有效负载才会处于更改事件中。如果您使用 JSON 转换程序，并将其配置为生成所有四个基本更改事件部分，更改事件具有此结构：

```
{
 "schema": { 1
   ...
 },
 "payload": { 2
   ...
 },
 "schema": { 3
   ...
 },
 "payload": { 4
   ...
 },
}
```

表 3.7. 更改事件基本内容概述

项	字段名称	描述

项	字段名称	描述
1	**schema**	第一个 **schema** 字段是事件键的一部分。它指定一个 Kafka Connect 模式，用于描述事件键的 **payload** 部分的内容。换句话说，第一个 **schema** 字段描述了主密钥的结构，如果表没有主键，则描述主键的结构。 可以通过设置 **message.key.columns** 连接器配置属性 来覆盖表的主键。在这种情况下，第一个 schema 字段描述了该属性标识的键的结构。
2	**payload**	第一个 **payload** 字段是 event 键的一部分。它具有前面的**schema** 字段描述的结构，它包含已更改的行的密钥。
3	**schema**	第二个 **schema** 字段是事件值的一部分。它指定 Kafka Connect 模式，用于描述事件值 **有效负载部分的内容**。换句话说，第二个 **模式** 描述了已更改的行的结构。通常，此模式包含嵌套模式。
4	**payload**	第二个 **payload** 字段是事件值的一部分。它具有上一个 **schema** 字段描述的结构，它包含已更改的行的实际数据。

默认情况下，连接器流将事件记录改为与事件原始表相同的主题。如需更多信息，请参阅 主题名称。

警告

Debezium Db2 连接器确保所有 Kafka Connect 模式名称都遵循 Avro 模式名称格式。这意味着逻辑服务器名称必须以拉丁字母或下划线开头，即 a-z、A-Z 或 _。逻辑服务器名称和数据库和表名称中的每个字符都必须是拉丁字母、数字或下划线，即 a-z、A-Z、0-9 或 _。如果存在无效字符，它将使用下划线字符替换。

如果逻辑服务器名称、数据库名称或表名称包含无效字符，且唯一与另一个名称区分名称的字符无效，这可能会导致意外冲突冲突，从而被下划线替换。

另外，数据库、模式和表的 Db2 名称可能区分大小写。这意味着连接器可将多个表的事件记录发送到同一 Kafka 主题。

详情包括在以下主题中：

- 第 3.3.1 节 “关于 Debezium db2 中的键更改事件”
- 第 3.3.2 节 “关于 Debezium Db2 中的值更改事件”

3.3.1. 关于 Debezium db2 中的键更改事件

更改事件的密钥包含更改表的密钥和更改行的实际键的 schema。当连接器创建事件时，schema 及其对应有效负载都会包含更改表的 PRIMARY KEY （或唯一约束）中每个列的字段。

考虑以下 客户 表，后跟此表的更改事件键的示例。

表示例

```
CREATE TABLE customers (
 ID INTEGER IDENTITY(1001,1) NOT NULL PRIMARY KEY,
 FIRST_NAME VARCHAR(255) NOT NULL,
 LAST_NAME VARCHAR(255) NOT NULL,
 EMAIL VARCHAR(255) NOT NULL UNIQUE
);
```

更改事件键示例

每次捕获 customer 表的更改事件都有相同的事件关键模式。只要 customers 表有以前的定义，可以捕获 customer 表更改的事件都有以下关键结构：在 JSON 中，它类似如下：

```
{
  "schema": { 1
    "type": "struct",
    "fields": [ 2
      {
        "type": "int32",
        "optional": false,
        "field": "ID"
      }
    ],
    "optional": false, 3
    "name": "mydatabase.MYSCHEMA.CUSTOMERS.Key" 4
  },
```

```
    "payload": {  5
      "ID": 1004
    }
}
```

表 3.8. 更改事件键的描述

项	字段名称	描述
1	**schema**	键的 schema 部分指定一个 Kafka Connect 模式，它描述了键的 **payload** 部分的内容。
2	**fields**	指定 **有效负载中** 预期的每个字段，包括每个字段的名称、类型以及是否需要。
3	**optional**	指明 event 键是否必须在其 **payload** 字段中包含一个值。在本例中，键有效负载中的值是必需的。当表没有主键时，键的 payload 字段中的值是可选的。
4	**mydatabase.MYSCHEMA.CUSTOMERS.Key**	定义密钥有效负载结构的模式名称。这个 schema 描述了已更改的表的主键的结构。键模式名称的格式是 *connector-name.database-name.table-name*.**Key**.在本例中： • **mydatabase** 是生成此事件的连接器的名称。 • **MYSCHEMA** 是包含已更改的表的数据库模式。 • **CUSTOMERS** 是更新的表。
5	**payload**	包含生成此更改事件的行的密钥。在本例中，键 包含一个 **ID** 字段，其值为 **1004**。

3.3.2. 关于 Debezium Db2 中的值更改事件

更改事件中的值比键复杂一些。与键一样，该值有一个 **schema** 部分和 **payload** 部分。**schema** 部分包含描述 **payload** 部分的 **Envelope** 结构的 **schema**，包括其嵌套字段。为创建、更新或删除数据的操作更改事件，它们都有一个带有 **envelope** 结构的值有效负载。

考虑用于显示更改事件键示例的相同示例表：

表示例

```
CREATE TABLE customers (
 ID INTEGER IDENTITY(1001,1) NOT NULL PRIMARY KEY,
```

```
 FIRST_NAME VARCHAR(255) NOT NULL,
 LAST_NAME VARCHAR(255) NOT NULL,
 EMAIL VARCHAR(255) NOT NULL UNIQUE
);
```

customer 表的每个更改事件的事件值部分都指定了相同的模式。事件值的有效负载因事件类型而异：

- *创建* 事件
- *更新* 事件
- *删除* 事件

创建 事件

以下示例显示了一个更改事件的值部分，连接器为在 customer 表中创建数据的操作生成的更改事件的值部分：

```
{
 "schema": {  1
  "type": "struct",
  "fields": [
    {
     "type": "struct",
     "fields": [
       {
         "type": "int32",
         "optional": false,
         "field": "ID"
       },
       {
         "type": "string",
         "optional": false,
         "field": "FIRST_NAME"
       },
       {
         "type": "string",
         "optional": false,
         "field": "LAST_NAME"
       },
       {
         "type": "string",
```

```
        "optional": false,
        "field": "EMAIL"
      }
    ],
    "optional": true,
    "name": "mydatabase.MYSCHEMA.CUSTOMERS.Value", (2)
    "field": "before"
  },
  {
    "type": "struct",
    "fields": [
      {
        "type": "int32",
        "optional": false,
        "field": "ID"
      },
      {
        "type": "string",
        "optional": false,
        "field": "FIRST_NAME"
      },
      {
        "type": "string",
        "optional": false,
        "field": "LAST_NAME"
      },
      {
        "type": "string",
        "optional": false,
        "field": "EMAIL"
      }
    ],
    "optional": true,
    "name": "mydatabase.MYSCHEMA.CUSTOMERS.Value",
    "field": "after"
  },
  {
    "type": "struct",
    "fields": [
      {
        "type": "string",
        "optional": false,
        "field": "version"
      },
      {
        "type": "string",
        "optional": false,
        "field": "connector"
      },
      {
        "type": "string",
        "optional": false,
        "field": "name"
      },
      {
        "type": "int64",
```

```
          "optional": false,
          "field": "ts_ms"
        },
        {
          "type": "boolean",
          "optional": true,
          "default": false,
          "field": "snapshot"
        },
        {
          "type": "string",
          "optional": false,
          "field": "db"
        },
        {
          "type": "string",
          "optional": false,
          "field": "schema"
        },
        {
          "type": "string",
          "optional": false,
          "field": "table"
        },
        {
          "type": "string",
          "optional": true,
          "field": "change_lsn"
        },
        {
          "type": "string",
          "optional": true,
          "field": "commit_lsn"
        },
      ],
      "optional": false,
      "name": "io.debezium.connector.db2.Source", 3
      "field": "source"
    },
    {
      "type": "string",
      "optional": false,
      "field": "op"
    },
    {
      "type": "int64",
      "optional": true,
      "field": "ts_ms"
    }
  ],
  "optional": false,
  "name": "mydatabase.MYSCHEMA.CUSTOMERS.Envelope" 4
},
"payload": { 5
  "before": null, 6
  "after": { 7
```

```
    "ID": 1005,
    "FIRST_NAME": "john",
    "LAST_NAME": "doe",
    "EMAIL": "john.doe@example.org"
   },
   "source": {  8
    "version": "2.3.4.Final",
    "connector": "db2",
    "name": "myconnector",
    "ts_ms": 1559729468470,
    "snapshot": false,
    "db": "mydatabase",
    "schema": "MYSCHEMA",
    "table": "CUSTOMERS",
    "change_lsn": "00000027:00000758:0003",
    "commit_lsn": "00000027:00000758:0005",
   },
   "op": "c",  9
   "ts_ms": 1559729471739  10
  }
 }
```

表 3.9. *创建* 事件值字段的描述

项	字段名称	描述
1	**schema**	值的 schema，用于描述值有效负载的结构。当连接器为特定表生成的每次更改事件中，更改事件的值模式都是相同的。
2	**name**	在 **schema** 部分中，每个 **name** 字段为值的有效负载中的字段指定 schema。 **mydatabase.MYSCHEMA.CUSTOMERS.Value** 是有效负载 **before** 和 **after** 字段的 schema。这个模式特定于 **customers** 表。连接器将此模式用于 **MYSCHEMA.CUSTOMERS** 表中的所有行。 **before** 和 **after** 字段的 schema 的名称格式为 ***logicalName.schemaName.tableName*.Value**，这样可确保 schema 名称在数据库中是唯一的。这意味着，在使用 Avro converter 时，每个逻辑源中的每个表生成的 Avro 模式都有自己的 evolution 和 history。
3	**name**	**io.debezium.connector.db2.Source** 是有效负载的 **source** 字段的 schema。这个模式特定于 Db2 连接器。连接器将其用于它生成的所有事件。
4	**name**	**mydatabase.MYSCHEMA.CUSTOMERS.Envelope** 是载荷总体结构的 schema，其中 **mydatabase** 是数据库，**MYSCHEMA** 是架构，**CUSTOMERS** 是表。
5	**payload**	值的实际数据。这是更改事件提供的信息。 可能会出现事件的 JSON 表示比它们描述的行大得多。这是因为 JSON 表示必须包含消息的 schema 部分和有效负载部分。但是，通过使用 Avro converter，您可以显著减少连接器流到 Kafka 主题的信息大小。

项	字段名称	描述
6	**before**	指定事件发生前行状态的可选字段。当 **op** 字段是 **c** 用于创建（如本例所示），**before** 字段为 **null**，因为此更改事件用于新内容。
7	**after**	指定事件发生后行状态的可选字段。在本例中，**after** 字段包含新行的 **ID**、**FIRST_NAME**、**LAST_NAME** 和 **EMAIL** 列的值。
8	**source**	描述事件源元数据的必需字段。**源** 结构显示有关此更改的 Db2 信息，它提供了可追溯性。它还有可与同一主题中的其他事件或其它主题进行比较的信息，以了解此事件在之前、之后还是与其他事件相同提交的一部分。源元数据包括： ● Debezium 版本 ● 连接器类型和名称 ● 在数据库中进行更改时的时间戳 ● 事件是持续快照的一部分 ● 包含新行的数据库、模式和表的名称 ● 更改 LSN ● 提交 LSN （如果此事件是快照的一部分）
9	**op**	描述导致连接器生成事件的操作类型的强制字符串。在本例中，**c** 表示操作创建了行。有效值为： ● **c** = create ● **u** = update ● **d** = delete ● **r** = 读取（仅适用于快照）
10	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

***更新* 事件**

示例 customers 表中一个更新的改变事件的值有与那个表的 *create* 事件相同的模式。同样，*update* 事件值有效负载具有相同的结构。但是，事件值有效负载在 *update* 事件中包含不同的值。以下是连接器为 customer 表中更新生成的更改事件值 的示例 ：

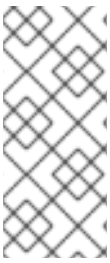
```
{
```

```
  "schema": { ... },
  "payload": {
    "before": { 1
      "ID": 1005,
      "FIRST_NAME": "john",
      "LAST_NAME": "doe",
      "EMAIL": "john.doe@example.org"
    },
    "after": { 2
      "ID": 1005,
      "FIRST_NAME": "john",
      "LAST_NAME": "doe",
      "EMAIL": "noreply@example.org"
    },
    "source": { 3
      "version": "2.3.4.Final",
      "connector": "db2",
      "name": "myconnector",
      "ts_ms": 1559729995937,
      "snapshot": false,
      "db": "mydatabase",
      "schema": "MYSCHEMA",
      "table": "CUSTOMERS",
      "change_lsn": "00000027:00000ac0:0002",
      "commit_lsn": "00000027:00000ac0:0007",
    },
    "op": "u", 4
    "ts_ms": 1559729998706 5
  }
}
```

表 3.10. *更新* 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在 *update* 事件值中，**before** 字段包含每个表列的字段，以及数据库提交前该列中的值。在本例中，请注意 **EMAIL** 值为 **john.doe@example.com**。
2	**after**	指定事件发生后行状态的可选字段。您可以比较 **before** 和 **after** 结构，以确定这个行的更新是什么。在这个示例中，**EMAIL** 值是 **noreply@example.com**。

项	字段名称	描述
3	**source**	描述事件源元数据的必需字段。**source** 字段结构包含与 *create* 事件中的相同字段，但某些值有所不同，例如，示例 *update* 事件具有不同的 LSN。您可以使用此信息将此事件与其他事件进行比较，以了解此事件在之前、之后还是与其他事件相同的提交的一部分。源元数据包括： • Debezium 版本 • 连接器类型和名称 • 在数据库中进行更改时的时间戳 • 事件是持续快照的一部分 • 包含新行的数据库、模式和表的名称 • 更改 LSN • 提交 LSN （如果此事件是快照的一部分）
4	**op**	描述操作类型的强制字符串。在 *update* 事件值中，**op** 字段值为 **u**，表示此行因为更新而改变。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。



注意

更新行 primary/unique 键的列会更改行的键值。当键更改时，Debezium 会输出 三个事件：一个 DELETE 事件，以及一个带有行的旧键的 tombstone 事件，后跟一个带有行的新键的事件。

***删除* 事件**

***delete* 更改事件中的值与为同一表的 *create* 和 *update* 事件相同的 schema 部分。示例 customer 表的 *delete* 事件中的事件值 payload 类似如下：**

```
{
  "schema": { ... },
  },
  "payload": {
    "before": { 1
      "ID": 1005,
      "FIRST_NAME": "john",
      "LAST_NAME": "doe",
```

```
    "EMAIL": "noreply@example.org"
  },
  "after": null, (2)
  "source": { (3)
    "version": "2.3.4.Final",
    "connector": "db2",
    "name": "myconnector",
    "ts_ms": 1559730445243,
    "snapshot": false,
    "db": "mydatabase",
    "schema": "MYSCHEMA",
    "table": "CUSTOMERS",
    "change_lsn": "00000027:00000db0:0005",
    "commit_lsn": "00000027:00000db0:0007"
  },
  "op": "d", (4)
  "ts_ms": 1559730450205 (5)
 }
}
```

表 3.11. *删除* 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在一个 *delete* 事件值中，**before** 字段包含在使用数据库提交删除行前的值。
2	**after**	指定事件发生后行状态的可选字段。在 *delete* 事件值中，**after** 字段为 **null**，表示行不再存在。
3	**source**	描述事件源元数据的必需字段。在一个 *delete* 事件值中，**source** 字段结构与同一表的 *create* 和 *update* 事件相同。许多 **source** 字段值也相同。在 *delete* 事件值中，**ts_ms** 和 LSN 字段值以及其它值可能会改变。但是 *delete* 事件值中的 **source** 字段提供相同的元数据： • Debezium 版本 • 连接器类型和名称 • 在数据库中进行更改时的时间戳 • 事件是持续快照的一部分 • 包含新行的数据库、模式和表的名称 • 更改 LSN • 提交 LSN （如果此事件是快照的一部分）
4	**op**	描述操作类型的强制字符串。**op** 字段值为 **d**，表示此行已被删除。

项	字段名称	描述
5	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

删除 **更改事件记录为消费者提供处理此行删除所需的信息。包含旧值，因为有些用户可能需要它们才能正确处理删除。**

Db2 连接器事件旨在使用 Kafka 日志压缩。只要保留每个密钥的最新消息，日志压缩就会启用删除一些旧的消息。这可让 Kafka 回收存储空间，同时确保主题包含完整的数据集，并可用于重新载入基于密钥的状态。

删除行时，*delete* 事件值仍可用于日志压缩，因为 Kafka 您可以删除具有相同键的所有之前信息。但是，要让 Kafka 删除具有相同键的所有消息，消息值必须为 null。为了实现此目的，在 Debezium 的 Db2 连接器发出 *delete* 事件后，连接器会发出一个特殊的 tombstone 事件，它具有相同的键有一个 null 值 。

3.4. DEBEZIUM DB2 连接器如何映射数据类型

有关 Db2 支持的数据类型的完整描述，请参阅 Db2 文档中的 数据类型。

Db2 连接器代表对包含结构的事件的更改，这些事件与行存在的表类似。事件包含每个列值的一个字段。在事件中如何代表该值取决于列的 Db2 数据类型。本节描述了这些映射。如果默认数据类型转换不满足您的需要，您可以为连接器 创建自定义转换器。

以下部分详情：

- 基本类型
- 时序类型

- 时间戳类型
- 表 3.15 “十进制类型”

基本类型

下表描述了连接器如何将每个 Db2 数据类型映射到 *字面类型以及* 事件字段中 *的语义类型*。

- *literal type* 描述了值如何表示，使用 Kafka Connect schema 类型：INT8,INT16,INT32,INT64,FLOAT32,FLOAT64,BOOLEAN,STRING,BYTES, ARRAY ,ARRAY,MAP, STRUCT.
- *语义类型* 描述了 Kafka Connect 模式如何使用字段名称来捕获字段 *的含义*。

表 3.12. Db2 基本数据类型的映射

Db2 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
布尔值	布尔值	只有快照可以从带有 BOOLEAN 类型列的表中获取。目前，Db2 上的 SQL 复制不支持 BOOLEAN，因此 Debezium 无法在这些表上执行 CDC。考虑使用其他类型的类型。
BIGINT	INT64	不适用
二进制	BYTES	不适用
BLOB	BYTES	不适用
CHAR[(N)]	字符串	不适用
CLOB	字符串	不适用
DATE	INT32	io.debezium.time.Date 代表没有时区信息的时间戳
DECFLOAT	BYTES	org.apache.kafka.connect.data.Decimal
十进制	BYTES	org.apache.kafka.connect.data.Decimal
DBCLOB	字符串	不适用

Db2 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DOUBLE	**FLOAT64**	不适用
整数	**INT32**	不适用
REAL	**FLOAT32**	不适用
SMALLINT	**INT16**	不适用
时间	**INT32**	**io.debezium.time.Time** 字符串表示没有时区信息
TIMESTAMP	**INT64**	**io.debezium.time.MicroTimestamp** 字符串表示没有时区信息的时间戳
VARBINARY	**BYTES**	不适用
VARCHAR[(N)]	**字符串**	不适用
VARGRAPHIC	**字符串**	不适用
XML	**字符串**	**io.debezium.data.Xml** 字符串表示 XML 文档

如果存在，则列的默认值会被传播到对应字段的 Kafka Connect 模式。更改事件包含字段的默认值，除非给出了显式列值。因此，很少需要从 schema 获取默认值。

时序类型

除了包含时区信息的 DATETIMEOFFSET 数据类型外，Db2 根据 time.precision.mode 连接器配置属性值来映射临时类型。以下小节描述了这些映射：

- time.precision.mode=adaptive
- time.precision.mode=connect

time.precision.mode=adaptive

当将 time.precision.mode 配置属性设置为 adaptive 时，连接器会根据列的数据类型定义决定字面 type 和 semantic 类型。这样可确保事件 *完全* 代表数据库中的值。

表 3.13. time.precision.mode 为 adaptive时的映射

Db2 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**io.debezium.time.Date** 代表自时期起的天数。
TIME (0), **TIME (1)**, **TIME (2)**, **TIME (3)**	**INT32**	**io.debezium.time.Time** 代表过去的毫秒数，不包括时区信息。
TIME (4), **TIME (5)**, **TIME (6)**	**INT64**	**io.debezium.time.MicroTime** 代表过去的微秒数，不包括时区信息。
TIME (7)	**INT64**	**io.debezium.time.NanoTime** 代表过去午夜的纳秒数量，不包括时区信息。
DATETIME	**INT64**	**io.debezium.time.Timestamp** 代表自 epoch 后的毫秒数，不包括时区信息。

time.precision.mode=connect

当将 time.precision.mode 配置属性设为 connect 时，连接器会使用 Kafka Connect 逻辑类型。当消费者只能处理内置的 Kafka Connect 逻辑类型，且无法处理变量-precision 时间值时，这非常有用。但是，因为 Db2 支持十分之一微秒的精度，使用 connect 时间精度的连接器会在数据库列带有 *fractional second precision* 值大于 3 时，导致精度下降。

表 3.14. 当 time.precision.mode 为 connect时映射

Db2 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**org.apache.kafka.connect.data.Date** 代表自 epoch 后的天数。
TIME([P])	**INT64**	**org.apache.kafka.connect.data.Time** 代表自午夜起的毫秒数，不包括时区信息。Db2 允许 **P** 在 0-7 范围内存储最多 10 个微秒的精度，但当 **P** 大于 3 时，这个模式会导致精度丢失。

Db2 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATETIME	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 epoch 起的毫秒数，不包括时区信息。

时间戳类型

DATETIME 类型代表一个没有时区信息的时间戳。此类列根据 UTC 转换为对等的 Kafka Connect 值。例如，DATETIME 值 "2018-06-20 15:13:16.945104" 由一个带有值 "1529507596000" 的 io.debezium.time.Timestamp 代表。

运行 Kafka Connect 和 Debezium 的 JVM 时区不会影响此转换。

表 3.15. 十进制类型

Db2 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
NUMERIC[(P[,S])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** schema 参数包含一个整数，代表十进制点被转换了多少位。**connect.decimal.precision** schema 参数包含一个整数，代表给定十进制值的精度。
DECIMAL[(P[,S])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** schema 参数包含一个整数，代表十进制点被转换了多少位。**connect.decimal.precision** schema 参数包含一个整数，代表给定十进制值的精度。

3.5. 设置 DB2 以运行 DEBEZIUM 连接器

要使 Debezium 捕获提交到 Db2 表的更改事件，具有所需特权的 Db2 数据库管理员必须在数据库中配置表以更改数据捕获。开始运行 Debezium 后，您可以调整捕获代理的配置来优化性能。

有关设置用于 Debezium 连接器的 Db2 的详情，请查看以下部分：

- **第 3.5.1 节 “配置 Db2 表以更改数据捕获”**

- 第 3.5.2 节 “Db2 捕获代理配置对服务器负载和延迟的影响”
- 第 3.5.3 节 “Db2 捕获代理配置参数”

3.5.1. 配置 Db2 表以更改数据捕获

要将表置于捕获模式中，Debebe 提供了一组用户定义的功能(UDF)，供您使用。此处的步骤演示了如何安装和运行这些管理 UDF。或者，您可以运行 Db2 控制命令将表置于捕获模式。然后，管理员必须为您要 Debezium 捕获的每个表启用 CDC。

先决条件

- 以 db2instl 用户身份登录到 Db2。
- 在 Db2 主机上，Debezium 管理 UDF 位于 $HOME/asncdctools/src 目录中。UDF 可从 Debezium 示例存储库 获取。
- Db2 命令 bldrtn 位于 PATH 中，例如，使用 Db2 11.5 运行 导出 PATH=$PATH:/opt/ibm/db2/V11.5.0.0/samples/c/

流程

1. 使用 Db2 提供的 bldrtn 命令，编译 Db2 服务器主机上的 Debezium 管理 UDF：

```
cd $HOME/asncdctools/src
```

```
bldrtn asncdc
```

2. 启动数据库（如果尚未运行）。将 DB_NAME 替换为您要 Debezium 连接到的数据库的名称。

```
db2 start db DB_NAME
```

3. 确保 JDBC 可以读取 Db2 元数据目录：

```
cd $HOME/sqllib/bnd
```

```
db2 connect to DB_NAME
db2 bind db2schema.bnd blocking all grant public sqlerror continue
```

4. 确保数据库最近备份。ASN 代理必须具有从中读取的最新起点。如果您需要执行备份，请运行以下命令来修剪数据，以便只有最新版本可用。如果您不需要保留旧版本的数据，请为备份位置指定 dev/null。

   a. 备份数据库。将 DB_NAME 和 BACK_UP_LOCATION 替换为适当的值：

   ```
   db2 backup db DB_NAME to BACK_UP_LOCATION
   ```

   b. 重启数据库：

   ```
   db2 restart db DB_NAME
   ```

5. 连接到数据库，以安装 Debezium 管理 UDF。假设您以 db2instl 用户身份登录，因此 UDF 应在 db2inst1 用户上安装。

   ```
   db2 connect to DB_NAME
   ```

6. 复制 Debezium 管理 UDF，并为它们设置权限：

   ```
   cp $HOME/asncdctools/src/asncdc $HOME/sqllib/function
   ```

   ```
   chmod 777 $HOME/sqllib/function
   ```

7. 启用用于启动和停止 ASN 捕获代理的 Debezium UDF：

   ```
   db2 -tvmf $HOME/asncdctools/src/asncdc_UDF.sql
   ```

8. 创建 ASN 控制表：

   ```
   $ db2 -tvmf $HOME/asncdctools/src/asncdctables.sql
   ```

9. 启用 Debezium UDF，添加表以捕获模式并从捕获模式中删除表：

   ```
   $ db2 -tvmf $HOME/asncdctools/src/asncdcaddremove.sql
   ```

- 设置 Db2 服务器后，使用 UDF 使用 SQL 命令控制 Db2 复制(ASN)。有些 UDF 期望返回值，在这种情况下，您使用 SQL VALUE 语句调用它们。对于其他 UDF，请使用 SQL CALL 语句。

10. 从 SQL 客户端启动 ASN 代理：

```
VALUES ASNCDC.ASNCDCSERVICES('start','asncdc');
```

或者在 shell 中：

```
db2 "VALUES ASNCDC.ASNCDCSERVICES('start','asncdc');"
```

前面的语句返回以下结果之一：

- asncap 已在运行
- start --> *<COMMAND>*

在这种情况下，在终端窗口中输入指定的 *<COMMAND>*，如下例所示：

```
/database/config/db2inst1/sqllib/bin/asncap capture_schema=asncdc
capture_server=SAMPLE &
```

11. 将表置于捕获模式。对您要放入捕获的每个表调用以下语句：将 MYSCHEMA 替换为包含您要放入捕获模式的模式的名称。同样，将 MYTABLE 替换为要放入捕获模式的表名称：

```
CALL ASNCDC.ADDTABLE('MYSCHEMA', 'MYTABLE');
```

12. 重新初始化 ASN 服务：

```
VALUES ASNCDC.ASNCDCSERVICES('reinit','asncdc');
```

其他资源

Debezium Db2 管理 UDF 的参考表

3.5.2. Db2 捕获代理配置对服务器负载和延迟的影响

当数据库管理员为源表启用更改数据捕获时，捕获代理开始运行。代理从事务日志中读取新的更改事件记录，并将事件记录复制到捕获表中。在源表中提交更改的时间以及更改出现在对应更改表中的时间，总有较小的延迟间隔。这个延迟间隔代表在源表中发生更改时以及 Debezium 可用于 Apache Kafka 的更改时之间的差距。

理想情况下，对于必须快速响应数据变化的应用程序，您希望在源和捕获表之间保持关闭同步。您可能想，运行捕获代理以尽可能快地持续处理事件更改事件，可能会导致吞吐量增加，并减少 latency netobserv-wagoning 更改表，以便在事件发生后马上使用新事件记录（在最近实时发生）。但是，这不一定如此。在寻求更多即时同步时，需要支付性能损失。每次更改代理查询数据库以获取新事件记录时，它会增加数据库主机上的 CPU 负载。服务器上的额外的负载可能会对整个数据库性能造成负面影响，并可能会降低事务效率，特别是在高峰数据库使用时。

监控数据库指标非常重要，以便您知道数据库是否达到服务器无法支持捕获代理的活动级别。如果您在运行捕获代理时遇到问题，请调整捕获代理设置来减少 CPU 负载。

3.5.3. Db2 捕获代理配置参数

在 Db2 上，IBMSNAP_CAPPARMS 表包含控制捕获代理行为的参数。您可以调整这些参数的值，以平衡捕获进程的配置，以减少 CPU 负载，并且仍然保持可接受的延迟级别。

注意

有关如何配置 Db2 捕获代理参数的具体指导超出了本文档的范围。

在 IBMSNAP_CAPPARMS 表中，以下参数对减少 CPU 负载有最大影响：

COMMIT_INTERVAL

- 指定捕获代理等待将数据提交到更改数据的秒数。
- 较高的值可减少数据库主机上的负载并增加延迟。

- 默认值为 30。

SLEEP_INTERVAL

- 指定捕获代理在达到活跃事务日志结束后等待启动新的提交周期的秒数。
- 较高的值可减少服务器上的负载，并增加延迟。
- 默认值为 5。

其他资源

- 有关捕获代理参数的更多信息，请参阅 Db2 文档。

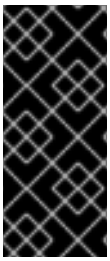
3.6. 部署 DEBEZIUM DB2 连接器

您可以使用以下任一方法部署 Debezium Db2 连接器：

- 使用 AMQ Streams 自动创建包含连接器插件的镜像。

  这是首选的方法。

- 从 Dockerfile 构建自定义 Kafka Connect 容器镜像。



重要

由于许可证要求，Debezium Db2 连接器存档不包括 Debezium 连接到 Db2 数据库的 Db2 JDBC 驱动程序。要启用连接器访问数据库，您必须将驱动程序添加到连接器环境中。有关如何获取驱动程序的详情，请参考 获取 Db2 JDBC 驱动程序。

其他资源

- 第 3.6.6 节 “Debezium Db2 连接器配置属性的描述”

3.6.1. 获取 Db2 JDBC 驱动程序

由于许可证的要求，Debezium 连接到一个 Db2 数据库所需的 Db2 JDBC 驱动程序文件没有包括在 Debezium Db2 连接器存档中。驱动程序可从 Maven Central 下载。根据您使用的部署方法，您可以通过向 Kafka Connect 自定义资源添加命令或用于构建连接器镜像的 Dockerfile 来检索驱动程序。

- 如果您使用 AMQ Streams 将连接器添加到 Kafka Connect 镜像，请将驱动程序的 Maven Central 位置添加到 KafkaConnect 自定义资源中的 builds.plugins.artifact.url 中，如第 3.6.3 节 “使用 AMQ Streams 部署 Debezium Db2 连接器”所示。
- 如果您使用 Dockerfile 为连接器构建容器镜像，请在 Dockerfile 中插入 curl 命令，以指定从 Maven Central 下载所需驱动程序文件的 URL。更多信息请参阅第 3.6.4 节 “通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium Db2 连接器”。

3.6.2. 使用 AMQ Streams 进行 Db2 连接器部署

从 Debezium 1.7 开始，部署 Debezium 连接器的首选方法是使用 AMQ Streams 构建包含连接器插件的 Kafka Connect 容器镜像。

在部署过程中，您可以创建并使用以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR，并包含有关镜像中需要包含连接器工件的信息。
- KafkaConnector CR，提供包括连接器用来访问源数据库的信息。在 AMQ Streams 启动 Kafka Connect pod 后，您可以通过应用 KafkaConnector CR 来启动连接器。

在 Kafka Connect 镜像的构建规格中，您可以指定可用于部署的连接器。对于每个连接器插件，您还可以指定您的部署可以使用的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。当 AMQ Streams 构建 Kafka Connect 镜像时，它会下载指定的工件，并将其合并到镜像中。

KafkaConnect CR 中的 spec.build.output 参数指定存储生成的 Kafka Connect 容器镜像的位置。容器镜像可以存储在 Docker registry 中，也可以存储在 OpenShift ImageStream 中。要将镜像存储在 ImageStream 中，您必须在部署 Kafka Connect 前创建 ImageStream。镜像流不会被自动创建。

注意

如果使用 KafkaConnect 资源来创建集群，之后无法使用 Kafka Connect REST API 创建或更新连接器。您仍然可以使用 REST API 来检索信息。

其他资源

- 在 OpenShift 中使用 AMQ Streams 配置 Kafka 连接。
- 在 OpenShift 中部署和管理 AMQ Streams 中，使用 AMQ Streams 自动创建新容器镜像。

3.6.3. 使用 AMQ Streams 部署 Debezium Db2 连接器

使用早期版本的 AMQ Streams 时，要在 OpenShift 上部署 Debezium 连接器，您需要首先为连接器构建 Kafka Connect 镜像。在 OpenShift 上部署连接器的当前首选方法是使用 AMQ Streams 中的构建配置来构建 Kafka Connect 容器镜像，其中包含您要使用的 Debezium 连接器插件。

在构建过程中，AMQ Streams Operator 将 KafkaConnect 自定义资源（包括 Debezium 连接器定义）中的输入参数转换为 Kafka Connect 容器镜像。构建会从 Red Hat Maven 存储库或其他配置的 HTTP 服务器下载必要的工件。

新创建的容器被推送到在 .spec.build.output 中指定的容器 registry，用于部署 Kafka Connect 集群。在 AMQ Streams 构建 Kafka Connect 镜像后，您可以创建 KafkaConnector 自定义资源来启动构建中包含的连接器。

先决条件

- 您可以访问安装了集群 Operator 的 OpenShift 集群。
- AMQ Streams Operator 正在运行。
- 在 OpenShift 中部署和升级 AMQ Streams 所述，会部署 Apache Kafka 集群。
- Kafka Connect 在 AMQ Streams 上部署

- 您有一个 Red Hat Integration 许可证。
- 已安装 OpenShift oc CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。
- 根据您要存储 Kafka Connect 构建镜像的方式，您需要 registry 权限，或者您必须创建 ImageStream 资源：

  将构建镜像存储在镜像 registry 中，如 Red Hat Quay.io 或 Docker Hub

  - 在 registry 中创建和管理镜像的帐户和权限。

  将构建镜像存储为原生 OpenShift ImageStream

  - ImageStream 资源已部署到集群中，以存储新的容器镜像。您必须为集群显式创建 ImageStream。默认无法使用镜像流。如需有关 ImageStreams 的更多信息，请参阅在 OpenShift Container Platform 中管理镜像流。

流程

1. 登录 OpenShift 集群。
2. 为连接器创建 Debezium KafkaConnect 自定义资源(CR)，或修改现有的资源。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 metadata.annotations 和 spec.build 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

> **例 3.1. 定义包含 Debezium 连接器的 KafkaConnect 自定义资源的 dbz-connect.yaml 文件**
>
> 在以下示例中，自定义资源被配置为下载以下工件：
>
> - Debezium Db2 连接器存档。
> - Service Registry 归档。Service Registry 是一个可选组件。只有在打算将 Avro 序列化与连接器搭配使用时，才添加 Service Registry 组件。

- Debezium 脚本 SMT 归档以及与 Debezium 连接器一起使用的相关语言依赖项。SMT 归档和语言依赖项是可选组件。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时，才添加这些组件。

- Db2 JDBC 驱动程序，需要连接到 Db2 数据库，但不包含在连接器存档中。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: debezium-kafka-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" (1)
spec:
  version: 3.5.0
  build: (2)
    output: (3)
      type: imagestream (4)
      image: debezium-streams-connect:latest
    plugins: (5)
      - name: debezium-connector-db2
        artifacts:
          - type: zip (6)
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-db2/2.3.4.Final-redhat-00001/debezium-connector-db2-2.3.4.Final-redhat-00001-plugin.zip (7)
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-distro-connect-converter/2.4.4.Final-redhat-<build-number>/apicurio-registry-distro-connect-converter-2.4.4.Final-redhat-<build-number>.zip (8)
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-scripting/2.3.4.Final-redhat-00001/debezium-scripting-2.3.4.Final-redhat-00001.zip
(9)
          - type: jar
            url:
https://repo1.maven.org/maven2/org/codehaus/groovy/groovy/3.0.11/groovy-3.0.11.jar (10)
          - type: jar
            url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-jsr223/3.0.11/groovy-jsr223-3.0.11.jar
          - type: jar
            url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-json3.0.11/groovy-json-3.0.11.jar
          - type: jar (11)
            url: https://repo1.maven.org/maven2/com/ibm/db2/jcc/11.5.0.0/jcc-11.5.0.0.jar

  bootstrapServers: debezium-kafka-cluster-kafka-bootstrap:9093

  ...
```

表 3.16. Kafka Connect 配置设置的描述

项	描述
1	将 **strimzi.io/use-connector-resources** 注解设置为 **"true"**，使 Cluster Operator 使用 **KafkaConnector** 资源在此 Kafka Connect 集群中配置连接器。
2	**spec.build** 配置指定在镜像中存储构建镜像的位置，并列出要在镜像中包含的插件，以及插件工件的位置。
3	**build.output** 指定存储新构建镜像的 registry。
4	指定镜像输出的名称和镜像名称。**output.type** 的有效值是 **要推送到** 容器 registry（如 Docker Hub 或 Quay）或 **镜像流** 的有效值，以将镜像推送到内部 OpenShift ImageStream。要使用 ImageStream，必须将 ImageStream 资源部署到集群中。有关在 KafkaConnect 配置中指定 **build.output** 的更多信息，请参阅在 OpenShift 中配置 AMQ Streams 中的 AMQ Streams Build schema 参考。
5	**plugins** 配置列出了您要包含在 Kafka Connect 镜像中的所有连接器。对于列表中的每个条目，指定一个插件**名称**，以及有关构建连接器所需的工件的信息。另外，对于每个连接器插件，您还可以包含可用于连接器的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。
6	**artifacts.type** 的值指定在 **artifacts.url** 中指定的工件类型。有效类型为 **zip**、**tgz** 或 **jar**。Debezium 连接器存档以 **.zip** 文件格式提供。JDBC 驱动程序文件采用 **.jar** 格式。**类型** 值必须与 **url** 字段中引用的文件类型匹配。
7	**artifacts.url** 的值指定 HTTP 服务器的地址，如 Maven 存储库，用于存储连接器工件的文件。OpenShift 集群必须有权访问指定的服务器。
8	（可选）指定用于下载 Service Registry 组件的工件 **类型和 url**。包含 Service Registry 工件，只有在您希望连接器使用 Apache Avro 来序列化带有 Service Registry 的事件键和值时，而不是使用默认的 JSON 转换程序。
9	（可选）指定 Debezium 脚本 SMT 归档的工件 **类型和 url**，以用于 Debezium 连接器。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时才包括脚本 SMT。要使用脚本 SMT，您必须部署 JSR 223 兼容脚本实现，如 groovy。

项	描述
10	（可选）指定 JSR 223 兼容脚本实施的 JAR 文件的工件 **类型和 url**，这是 Debezium 脚本 SMT 所需的。  **重要** 如果使用 AMQ Streams 将连接器插件合并到 Kafka Connect 镜像中，每个所需的脚本语言组件，artifact **.url** 必须指定 JAR 文件的位置，并且 **artifacts.type** 的值也必须设置为 **jar**。无效的值会导致连接器在运行时失败。 要启用带有脚本 SMT 的 Apache Groovy 语言，示例中的自定义资源会为以下库检索 JAR 文件： • **groovy** • **Groovy-jsr223**（指定代理） • **groovy-json**（解析 JSON 字符串的模块） Debezium 脚本 SMT 还支持使用 JSR 223 实现 GraalVM JavaScript。
11	在 Maven Central 中指定 Db2 JDBC 驱动程序的位置。Debezium Db2 连接器存档中没有包括所需的驱动程序。

3. **输入以下命令将 KafkaConnect 构建规格应用到 OpenShift 集群：**

```
oc create -f dbz-connect.yaml
```

根据自定义资源中指定的配置，Streams Operator 准备要部署的 Kafka Connect 镜像。构建完成后，Operator 将镜像推送到指定的 registry 或 ImageStream，并启动 Kafka Connect 集群。集群中提供了您在配置中列出的连接器工件。

4. **创建一个 KafkaConnector 资源来定义您要部署的每个连接器的实例。**
例如，创建以下 KafkaConnector CR，并将它保存为 db2-inventory-connector.yaml

例 3.2. 为 Debezium 连接器定义 KafkaConnector 自定义资源的 db2-inventory-connector.yaml 文件

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: debezium-kafka-connect-cluster
  name: inventory-connector-db2 (1)
spec:
```

```
    class: io.debezium.connector.db2.Db2ConnectorConnector (2)
    tasksMax: 1 (3)
    config: (4)
      schema.history.internal.kafka.bootstrap.servers: debezium-kafka-cluster-kafka-
bootstrap.debezium.svc.cluster.local:9092
      schema.history.internal.kafka.topic: schema-changes.inventory
      database.hostname: db2.debezium-db2.svc.cluster.local (5)
      database.port: 50000 (6)
      database.user: debezium (7)
      database.password: dbz (8)
      database.dbname: mydatabase (9)
      topic.prefix: inventory-connector-db2 (10)
      table.include.list: public.inventory (11)

      ...
```

表 3.17. 连接器配置设置的描述

项	描述
1	使用 Kafka Connect 集群注册的连接器名称。
2	连接器类的名称。
3	可以同时操作的任务数量。
4	连接器的配置。
5	主机数据库实例的地址。
6	数据库实例的端口号。
7	Debezium 用于连接到数据库的帐户名称。
8	Debezium 用于连接到数据库用户帐户的密码。
9	要从中捕获更改的数据库名称。
10	数据库实例或集群的主题前缀。 指定的名称只能由字母数字字符或下划线组成。 因为主题前缀被用作从这个连接器接收更改事件的任何 Kafka 主题的前缀，所以该名称在集群中的连接器之间必须是唯一的。 如果连接器与 Avro 连接器集成，则此命名空间也用于相关 Kafka Connect 模式的名称，以及相应 Avro 模式的命名空间。
11	连接器捕获更改事件的表列表。

5. 运行以下命令来创建连接器资源：

```
oc create -n <namespace> -f <kafkaConnector>.yaml
```

例如，

```
oc create -n debezium -f {context}-inventory-connector.yaml
```

连接器注册到 Kafka Connect 集群，并开始针对 KafkaConnector CR 中的 spec.config.database.dbname 指定的数据库运行。连接器 pod 就绪后，Debebe 正在运行。

现在，您已准备好 验证 Debezium Db2 部署。

3.6.4. 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium Db2 连接器

要部署 Debezium Db2 连接器，您必须构建包含 Debezium 连接器存档的自定义 Kafka Connect 容器镜像，然后将此容器镜像推送到容器 registry。然后，您需要创建以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR。CR 中的 image 属性指定您创建的容器镜像的名称，以运行 Debezium 连接器。您可以将此 CR 应用到部署 Red Hat AMQ Streams 的 OpenShift 实例。AMQ Streams 提供将 Apache Kafka 带到 OpenShift 的 operator 和镜像。
- 定义 Debezium Db2 连接器的 KafkaConnector CR。将此 CR 应用到应用 KafkaConnect CR 的同一 OpenShift 实例。

先决条件

- Db2 正在运行，您完成了 设置 Db2 以使用 Debezium 连接器 的步骤。
- AMQ Streams 部署在 OpenShift 中，并运行 Apache Kafka 和 Kafka Connect。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams。
- podman 或 Docker 已安装。
-

Kafka Connect 服务器有权访问 Maven Central，以下载 Db2 所需的 JDBC 驱动程序。您还可以使用驱动程序的本地副本，或者从本地 Maven 存储库或其他 HTTP 服务器可用的本地副本。

- 您有一个在容器 registry 中创建和管理容器（如 quay.io 或 docker.io）的帐户和权限，您要添加将运行 Debezium 连接器的容器。

流程

1. 为 Kafka Connect 创建 Debezium Db2 容器：

   a. 创建一个使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 的 Dockerfile 作为基础镜像。例如，在终端窗口中输入以下命令：

```
cat <<EOF >debezium-container-for-db2.yaml (1)
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/debezium (2)
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-db2/2.3.4.Final-redhat-00001/debezium-connector-db2-2.3.4.Final-redhat-00001-plugin.zip \
&& unzip debezium-connector-db2-2.3.4.Final-redhat-00001-plugin.zip \
&& rm debezium-connector-db2-2.3.4.Final-redhat-00001-plugin.zip
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://repo1.maven.org/maven2/com/ibm/db2/jcc/11.5.0.0/jcc-11.5.0.0.jar
USER 1001
EOF
```

项	描述
1	您可以指定您想要的任何文件名。
2	指定 Kafka Connect 插件目录的路径。如果您的 Kafka Connect 插件目录位于不同的位置，请将此路径替换为目录的实际路径。

   该命令在当前目录中创建一个名为 debezium-container-for-db2.yaml 的 Dockerfile。

   b. 从您在上一步中创建的 debezium-container-for-db2.yaml Docker 文件中构建容器镜像。在包含文件的目录中，打开终端窗口并输入以下命令之一：

```
podman build -t debezium-container-for-db2:latest .
```

```
docker build -t debezium-container-for-db2:latest .
```

前面的命令使用名称 debezium-container-for-db2 构建容器镜像。

c. 将自定义镜像推送到容器 registry，如 quay.io 或内部容器 registry。容器 registry 必须可供您要部署镜像的 OpenShift 实例使用。输入以下命令之一：

```
podman push <myregistry.io>/debezium-container-for-db2:latest
```

```
docker push <myregistry.io>/debezium-container-for-db2:latest
```

d. 创建新的 Debezium Db2 KafkaConnect 自定义资源(CR)。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 注解和 镜像 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  #...
  image: debezium-container-for-db2 2

  ...
```

项	描述
1	**metadata.annotations** 表示 **KafkaConnector** 资源用于配置在这个 Kafka Connect 集群中使用的 Cluster Operator。
2	**spec.image** 指定您创建的镜像的名称，以运行 Debezium 连接器。此属性覆盖 Cluster Operator 中的 **STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE** 变量。

e. 输入以下命令将 KafkaConnect CR 应用到 OpenShift Kafka Connect 环境：

```
oc create -f dbz-connect.yaml
```

该命令添加了一个 Kafka Connect 实例，用于指定您为运行 Debezium 连接器而创建的镜像的名称。

2. 创建一个 KafkaConnector 自定义资源，以配置 Debezium Db2 连接器实例。

您可以在 .yaml 文件中配置 Debezium Db2 连接器，该文件指定连接器的配置属性。连接器配置可能指示 Debezium 为 schema 和表的子集生成事件，或者可能会设置属性，以便 Debezium 忽略、掩码或截断敏感、太大或不需要的指定列中的值。

以下示例配置了一个 Debezium 连接器，它连接到端口 50000 上的 Db2 服务器主机 192.168.99.100。此主机有一个名为 mydatabase 的数据库，名为 inventory 的表，inventory-connector-db2 是服务器的逻辑名称。

Db2 inventory-connector.yaml

```
apiVersion: kafka.strimzi.io/v1beta2
 kind: KafkaConnector
 metadata:
  name: inventory-connector-db2 (1)
  labels:
   strimzi.io/cluster: my-connect-cluster
  annotations:
   strimzi.io/use-connector-resources: 'true'
 spec:
  class: io.debezium.connector.db2.Db2Connector (2)
  tasksMax: 1 (3)
  config: (4)
   database.hostname: 192.168.99.100 (5)
   database.port: 50000 (6)
   database.user: db2inst1 (7)
   database.password: Password! (8)
   database.dbname: mydatabase (9)
   topic.prefix: inventory-connector-db2 (10)
   table.include.list: public.inventory (11)

   ...
```

表 3.18. 连接器配置设置的描述

项	描述
1	将连接器的名称注册到 Kafka Connect 集群时。
2	此 Db2 连接器类的名称。
3	任何时候都只能运行一个任务。
4	连接器的配置。
5	数据库主机，即 Db2 实例的地址。
6	Db2 实例的端口号。
7	Db2 用户的名称。
8	Db2 用户的密码。
9	要从中捕获更改的数据库名称。
10	Db2 实例/集群的逻辑名称，它组成一个命名空间，并在使用 Avro Connector 时用于连接器写入的 Kafka 主题的名称、Kafka Connect 模式的名称以及相应 Avro 模式的命名空间。
11	连接器只捕获 **public.inventory** 表中的更改。

3. **使用 Kafka Connect 创建连接器实例。例如，如果您将 KafkaConnector 资源保存在 inventory-connector.yaml 文件中，您将运行以下命令：**

```
oc apply -f inventory-connector.yaml
```

前面的命令注册 inventory-connector，连接器开始针对 KafkaConnector CR 中定义的 mydatabase 数据库运行。

有关您可以为 Debezium Db2 连接器设置的配置属性的完整列表，请参阅 Db2 连接器属性。

结果

连接器启动后，它会为连接器配置 为捕获更改，执行 Db2 数据库表的一致性快照。然后，连接器开始为行级操作生成数据更改事件，并将事件记录流传输到 Kafka 主题。

3.6.5. 验证 Debezium Db2 连接器正在运行

如果连接器正确启动且没有错误，它会为每个连接器配置为捕获的表创建一个主题。下游应用程序可以订阅这些主题，以检索源数据库中发生的信息事件。

要验证连接器是否正在运行，您可以从 OpenShift Container Platform Web 控制台或 OpenShift CLI 工具(oc)执行以下操作：

- 验证连接器状态。
- 验证连接器是否生成主题。
- 验证主题是否填充了读取操作("op":"r")的事件，连接器在每个表的初始快照中生成。

先决条件

- Debezium 连接器部署到 OpenShift 上的 AMQ Streams。
- 已安装 OpenShift oc CLI 客户端。
- 访问 OpenShift Container Platform web 控制台。

流程

1. 使用以下方法之一检查 KafkaConnector 资源的状态：
   - 在 OpenShift Container Platform Web 控制台中：
     a. 导航到 Home → Search。
     b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaConnector。
     c. 在 KafkaConnectors 列表中，点您要检查的连接器的名称，如 inventory-

connector-db2。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- 在终端窗口中：

a. 使用以下命令：

```
oc describe KafkaConnector <connector-name> -n <project>
```

例如，

```
oc describe KafkaConnector inventory-connector-db2 -n debezium
```

该命令返回类似以下示例的状态信息：

例 3.3. KafkaConnector 资源状态

```
Name:         inventory-connector-db2
Namespace:    debezium
Labels:       strimzi.io/cluster=debezium-kafka-connect-cluster
Annotations:  <none>
API Version:  kafka.strimzi.io/v1beta2
Kind:         KafkaConnector

...

Status:
  Conditions:
    Last Transition Time:  2021-12-08T17:41:34.897153Z
    Status:                True
    Type:                  Ready
  Connector Status:
    Connector:
      State:      RUNNING
      worker_id:  10.131.1.124:8083
    Name:         inventory-connector-db2
    Tasks:
      Id:           0
      State:        RUNNING
      worker_id:    10.131.1.124:8083
    Type:           source
  Observed Generation:  1
```

```
  Tasks Max:          1
  Topics:
    inventory-connector-db2.inventory
    inventory-connector-db2.inventory.addresses
    inventory-connector-db2.inventory.customers
    inventory-connector-db2.inventory.geom
    inventory-connector-db2.inventory.orders
    inventory-connector-db2.inventory.products
    inventory-connector-db2.inventory.products_on_hand
Events:  <none>
```

2. 验证连接器是否创建了 Kafka 主题：

- 通过 OpenShift Container Platform Web 控制台。

a. 导航到 Home → Search。

b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaTopic。

c. 在 KafkaTopics 列表中，点您要检查的主题名称，例如 inventory-connector-db2.inventory.orders---ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- 在终端窗口中：

a. 使用以下命令：

```
oc get kafkatopics
```

该命令返回类似以下示例的状态信息：

例 3.4. KafkaTopic 资源状态

```
NAME                                                 CLUSTER
```

```
PARTITIONS   REPLICATION FACTOR   READY
connect-cluster-configs                                    debezium-kafka-cluster   1
1                  True
connect-cluster-offsets                                    debezium-kafka-cluster   25
1                  True
connect-cluster-status                                     debezium-kafka-cluster   5
1                  True
consumer-offsets---84e7a678d08f4bd226872e5cdd4eb527fadc1c6a
debezium-kafka-cluster   50          1                  True
inventory-connector-db2--a96f69b23d6118ff415f772679da623fbbb99421
debezium-kafka-cluster   1           1                  True
inventory-connector-db2.inventory.addresses---
1b6beaf7b2eb57d177d92be90ca2b210c9a56480          debezium-kafka-cluster
1          1                  True
inventory-connector-db2.inventory.customers---
9931e04ec92ecc0924f4406af3fdace7545c483b          debezium-kafka-cluster   1
1                  True
inventory-connector-db2.inventory.geom---
9f7e136091f071bf49ca59bf99e86c713ee58dd5              debezium-kafka-cluster
1          1                  True
inventory-connector-db2.inventory.orders---
ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d              debezium-kafka-cluster
1          1                  True
inventory-connector-db2.inventory.products---
df0746db116844cee2297fab611c21b56f82dcef          debezium-kafka-cluster   1
1                  True
inventory-connector-db2.inventory.products_on_hand---
8649e0f17ffcc9212e266e31a7aeea4585e5c6b5   debezium-kafka-cluster   1
1                  True
schema-changes.inventory                                   debezium-kafka-cluster
1          1                  True
strimzi-store-topic---effb8e3e057afce1ecf67c3f5d8e4e3ff177fc55          debezium-
kafka-cluster   1          1                  True
strimzi-topic-operator-kstreams-topic-store-changelog---
b75e702040b99be8a9263134de3507fc0cc4017b  debezium-kafka-cluster  1   1
True
```

3. **检查主题内容。**

- **在终端窗口中输入以下命令：**

```
oc exec -n <project>  -it <kafka-cluster> -- /opt/kafka/bin/kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=<topic-name>
```

例如，

```
oc exec -n debezium  -it debezium-kafka-cluster-kafka-0 -- /opt/kafka/bin/kafka-
console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=inventory-connector-db2.inventory.products_on_hand
```

指定主题名称的格式与 oc describe 命令返回的格式与第 1 步中返回，例如 inventory-connector-db2.inventory.addresses。

对于主题中的每个事件，命令会返回类似以下示例的信息：

例 3.5. Debezium 更改事件的内容


```
{"schema":{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"inventory-
connector-db2.inventory.products_on_hand.Key"},"payload":{"product_id":101}}
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-db2.inventory.products_on_hand.Value","field":"before"},{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-db2.inventory.products_on_hand.Value","field":"after"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":
{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connecto
r.db2.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transacti
on"}],"optional":false,"name":"inventory-connector-
db2.inventory.products_on_hand.Envelope"},"payload":{"before":null,"after":
{"product_id":101,"quantity":3},"source":{"version":"2.3.4.Final-redhat-
00001","connector":"db2","name":"inventory-connector-
db2","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"table":"pro
ducts_on_hand","server_id":0,"gtid":null,"file":"db2-
bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":1638985247805,"t
ransaction":null}}
```

在前面的示例中，有效负载 值显示连接器快照从表 inventory.products_on_hand 生成读取 (op" ="r")事件。product_id 记录的 "before" 状态为 null，表示该记录不存在之前的值。"after" 状态对于 product_id 为 101 的项目的 quantity 显示为 3。

3.6.6. Debezium Db2 连接器配置属性的描述

Debezium Db2 连接器具有大量配置属性，可用于实现应用程序的正确连接器行为。许多属性都有默认值。有关属性的信息组织如下：

- 所需的配置属性
- 高级配置属性
- 数据库模式历史记录连接器配置属性，用于控制 Debezium 如何处理从数据库 schema 历史记录主题读取的事件。
  - 透传数据库架构历史记录属性
- 控制 数据库驱动程序行为的直通数据库驱动程序属性。

所需的 Debezium Db2 连接器配置属性

除非默认值可用， 否则需要以下配置属性。

属性	默认	描述
name	没有默认值	连接器的唯一名称。尝试使用相同的名称再次注册将失败。所有 Kafka Connect 连接器都需要此属性。
connector.class	没有默认值	连接器的 Java 类的名称。始终对 Db2 连接器使用 io.debezium.connector.db2.Db2Connector 的值。

属性	默认	描述
tasks.max	**1**	应该为此连接器创建的最大任务数量。Db2 连接器始终使用单个任务，因此不使用这个值，因此默认值始终可以接受。
database.hostname	没有默认值	Db2 数据库服务器的 IP 地址或主机名。
database.port	**50000**	Db2 数据库服务器的整数端口号。
database.user	没有默认值	用于连接到 Db2 数据库服务器的 Db2 数据库用户的名称。
database.password	没有默认值	连接到 Db2 数据库服务器时要使用的密码。
database.dbname	没有默认值	从中流传输更改的 Db2 数据库的名称
topic.prefix	没有默认值	为特定的 Db2 数据库服务器提供命名空间的主题前缀，用于托管 Debezium 正在捕获更改的数据库。在主题前缀名称中只能使用字母数字字符、连字符、句点和下划线。主题前缀应该在所有其他连接器中唯一，因为此主题前缀用于从这个连接器接收记录的所有 Kafka 主题。  **警告** 不要更改此属性的值。如果您重启后更改了 name 值，而不是继续向原始主题发出事件，连接器会将后续事件发送到名称基于新值的主题。连接器也无法恢复其数据库架构历史记录主题。
table.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您希望连接器要捕获的表的完全限定表标识符匹配。当设置此属性时，连接器只捕获指定表中的更改。每个标识符都是 *schemaName.tableName*。默认情况下，连接器捕获每个非系统表中的更改。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配，它与表名称中的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.exclude.list** 属性。

属性	默认	描述
table.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，它与您不希望连接器捕获的表的完全限定表标识符匹配。连接器捕获不包括在 exclude 列表中的每个非系统表中的更改。每个标识符都是 *schemaName.tableName*。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配，它与表名称中的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.include.list** 属性。
column.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与列的完全限定域名匹配，以在更改事件记录值中包含。列的完全限定域名格式为 *schemaName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；它与列中可能出现的子字符串匹配。如果您在配置中包含此属性，不要设置 **column.exclude.list** 属性。
column.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与列的完全限定域名匹配，以便从更改事件值中排除。列的完全限定域名格式为 *schemaName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；它与列中可能出现的子字符串匹配。主键列始终包含在事件的键中，即使它们从值中排除。如果您在配置中包含此属性，请不要设置 **column.include.list** 属性。

属性	默认	描述
column.mask.hash.*hashAlgorithm*.with.salt.*salt*	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。列的完全限定域名格式为 *schemaName.tableName.columnName*。要匹配一个列的名称，Debezium 应用正则表达式，它由您指定为 *anchored* 正则表达式。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。在生成的更改事件记录中，指定列的值将被 pseudonyms 替换。 一个 pseudonym，它包括了通过应用指定的 *hashAlgorithm* 和 *salt* 的结果的哈希值。根据所使用的哈希函数，会维护引用完整性，而列值则替换为 pseudonyms。支持的哈希功能在 Java Cryptography 架构标准 Algorithm Name 文档的 MessageDigest 部分中 进行了描述。 在以下示例中，**CzQMA0cB5K** 是一个随机选择的 salt。 `column.mask.hash.SHA-256.with.salt.CzQMA0cB5K = inventory.orders.customerName, inventory.shipment.customerName` 如有必要，pseudonym 会自动缩短为列的长度。连接器配置可以包含多个属性，用于指定不同的哈希算法和 salt。 根据所用的 *hashAlgorithm* （选择 *salt* ）和实际数据集，生成的数据集可能无法完全屏蔽。
time.precision.mode	**adaptive**	时间、日期和时间戳可以以不同的精度类型代表： **adaptive** 基于数据库栏的类型，使用 millisecond, microsecond, 或 nanosecond 精度值，捕获数据库中的时间和时间戳。 **connect** 始终使用 Kafka Connect 的内置的 **Time**, **Date**, 和 **Timestamp** 的代表（无论数据库栏的精度，始终使用 millisecond 精度）来表示时间和时间戳的值。如需更多信息，请参阅临时类型。

属性	默认	描述
tombstones.on.delete	**true**	控制 *delete* 事件是否后跟一个 tombstone 事件。 **true** - 一个 delete 操作由 *delete* 事件和后续 tombstone 事件表示。 **false** - 仅有一个 *delete* 事件被抛出。 删除源记录后，发出 tombstone 事件（默认行为）可让 Kafka 在为主题启用了 日志 压缩时完全删除与已删除行键相关的所有事件。
include.schema.changes	**true**	布尔值，指定连接器是否应该将数据库模式中的更改发布到与数据库服务器 ID 的名称相同的 Kafka 主题。每个架构更改都使用包含数据库名称和一个 JSON 结构的键记录，该键描述了 schema 更新。这独立于连接器内部记录数据库架构历史记录。
column.truncate.to.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果在列中的数据超过了在属性名中的 *length* 指定的字符长度时删节数据，设置此属性。将 **length** 设置为正整数值，如 **column.truncate.to.20.chars**。 列的完全限定域名观察以下格式：*schemaName.tableName.columnName*。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。
column.mask.with.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果您希望连接器屏蔽一组列的值，例如，如果它们包含敏感数据，则设置此属性。将 ***length*** 设置为一个正整数，替换在属性名称中的 *length* 指定的星号（*）的数量列中的数据。将 *length* 设置为 **0**（零）将指定列中的数据替换为空字符串。 列的完全限定域名观察以下格式：*schemaName.tableName.columnName*。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。

属性	默认	描述
column.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，与您希望连接器发出代表列元数据的额外参数的完全限定名称匹配。当设置此属性时，连接器会将以下字段添加到事件记录的 schema 中： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*databaseName.tableName.columnName*, 或 *databaseName.schemaName.tableName.columnName*. 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。

属性	默认	描述
datatype.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，用于指定为数据库中列定义的数据类型的完全限定名称。当设置此属性时，对于具有匹配数据类型的列，连接器会发出在 schema 中包含以下额外字段的事件记录： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*databaseName.tableName.typeName*, 或 *databaseName.schemaName.tableName.typeName*. 要匹配数据类型的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据类型的整个名称字符串匹配；表达式与类型名称中可能存在的子字符串不匹配。 有关 Db2 特定数据类型名称的列表，请参阅 Db2 数据类型映射。

属性	默认	描述
message.key.columns	*空字符串*	指定连接器用来组成自定义消息键的表达式列表，用于更改它发布到指定表的 Kafka 主题的事件记录。 默认情况下，Debezium 使用表的主键列作为它发出的记录的消息键。在默认位置，或者为缺少主密钥的表指定一个键，您可以根据一个或多个列配置自定义消息密钥。 要为表建立自定义消息键，请列出表，后跟要用作消息键的列。每个列表条目都采用以下格式： ***<fully-qualified_tableName> : <keyColumn> , <keyColumn>*** To base a table key on multiple column name, 在列名称间插入逗号。 每个完全限定表名称是以下格式的正则表达式： ***<schemaName>.<tableName>*** 属性可以列出多个表的条目。使用分号分隔列表中不同表的条目。 以下示例为表 **inventory.customers** 和 **purchaseorders** 设置了消息键： **inventory.customers:pk1,pk2; (.*).purchaseorders:pk3,pk4** 在前面的示例中，列 **pk1** 和 **pk2** 作为表 **inventory.customer** 的消息键指定。对于任何模式中的购买顺序表，列 **pk3** 和 **pk4** 充当消息键。
schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列

属性	默认	描述
field.name.adjustment.mode	none	指定应如何调整字段名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列 如需了解更多详细信息，请参阅 Avro 命名。

高级连接器配置属性

以下 *高级配置* 属性在大多数情况下可以正常工作，因此很少需要在连接器的配置中指定。

属性	默认	描述

converters	没有默认值	枚举连接器可以使用的 自定义转换器 实例的符号链接列表。例如，
		isbn
		您必须设置 **converters** 属性，使连接器能够使用自定义转换器。
		对于您为连接器配置的每个转换器，您还必须添加一个 **.type** 属性，它指定了实现转换器接口的类的完整名称。**.type** 属性使用以下格式：
		***<converterSymbolicName>*.type**
		例如，
		`isbn.type: io.debezium.test.IsbnConverter`
		如果要进一步控制配置的转换器的行为，您可以添加一个或多个配置参数将值传递给转换器。要将任何其他配置参数与转换器关联，请为参数名称加上转换器的符号名作为前缀。
		例如，
		`isbn.schema.name: io.debezium.db2.type.Isbn`

属性	默认	描述
snapshot.mode	初始	指定在连接器启动时执行快照的条件： **initial** - 对于捕获模式的表，连接器为表和表中的数据生成模式快照。这可用于填充带有数据的完整表示的 Kafka 主题。 **initial_only** - 获取诸如 initial 的结构和数据的快照，而是在快照完成后不会过渡到流更改。 **模式_only** - 对于捕获模式的表，连接器仅获取表的快照。这只在从现在发生的更改需要发送到 Kafka 主题时很有用。快照完成后，连接器将通过从数据库的 redo 日志中读取更改事件来继续。
snapshot.isolation.mode	repeatable_read	在快照期间，控制事务隔离级别以及连接器锁定在捕获模式的表的时长。可能的值有： **read_uncommitted** - 不阻止其他事务在初始快照期间更新表行。这个模式没有数据一致性保证；有些数据可能会丢失或损坏。 **read_committed** - 不阻止其他事务在初始快照期间更新表行。新记录可能出现两次： 在初始快照中一次，一次在流传输阶段。但是，这个一致性级别适用于数据镜像。 **repeatable_read** - 防止在初始快照期间更新表行的其他事务。新记录可能出现两次： 在初始快照中一次，一次在流传输阶段。但是，这个一致性级别适用于数据镜像。 **exclusive** - 使用可重复的读取隔离级别，但对要读取的所有表采用专用锁定。这个模式可防止其他事务在初始快照期间更新表行。只有 **exclusive** 模式可以保证完全一致性；初始快照和流日志构成了线性历史记录。
event.processing.failure.handling.mode	fail	指定连接器在处理事件期间如何处理异常。可能的值有： **fail** - 连接器记录有问题的事件的偏移并停止处理。 **warn** - 连接器会记录有问题的事件的偏移，并使用下一个事件继续处理。 **跳过** - 连接器跳过有问题的事件，并继续处理下一个事件。

属性	默认	描述
poll.interval.ms	500	正整数值，指定连接器在开始处理批处理事件前应等待新更改事件数的毫秒数。默认值为 500 毫秒，或 0.5 秒。
max.batch.size	2048	正整数值，用于指定连接器处理的每个批处理的最大大小。
max.queue.size	8192	正整数值，用于指定阻塞队列可以保存的最大记录数。当 Debezium 从数据库读取事件时，它会将事件放置在阻塞队列中，然后再将它们写入 Kafka。阻塞队列可以提供从数据库读取更改事件时，连接器最快于将其写入 Kafka 的信息，或者在 Kafka 不可用时从数据库读取更改事件。当连接器定期记录偏移时，队列中保存的事件会被忽略。始终将 **max.queue.size** 的值设置为大于 **max.batch.size** 的值。
max.queue.size.in.bytes	0	一个长的整数值，用于指定阻塞队列的最大卷（以字节为单位）。默认情况下，不会为阻塞队列指定卷限制。要指定队列可以消耗的字节数，请将此属性设置为正长值。 如果还设置了 **max.queue.size**，当队列的大小达到任一属性指定的限制时，写入队列将被阻止。例如，如果您设置了 **max.queue.size=1000**、和 **max.queue.size.in.bytes=5000**，在队列包含 1000 个记录后，或者队列中记录的卷达到 5000 字节后，写入队列会被阻止。
heartbeat.interval.ms	0	控制连接器将心跳信息发送到 Kafka 主题的频率。默认行为是连接器不会发送心跳信息。 心跳消息可用于监控连接器是否从数据库接收更改事件。心跳消息有助于减少在连接器重启时需要重新更改事件的数量。要发送心跳消息，请将此属性设置为正整数，这代表心跳消息之间的毫秒数。 当数据库中有多个更新被跟踪但只有少量更新处于捕获模式时，心跳消息很有用。在这种情况下，连接器照常从数据库事务日志中读取，但很少向 Kafka 发出更改记录。这意味着连接器有几个向 Kafka 发送最新偏移的机会。发送心跳消息可让连接器将最新的偏移发送到 Kafka。
snapshot.delay.ms	没有默认值	连接器在连接器启动时执行快照前应等待的时间（以毫秒为单位）。如果您在集群中启动多个连接器，则此属性可用于避免快照中断，这可能会导致连接器的重新平衡。

属性	默认	描述
snapshot.include.collection.list	**table.include.list**中指定的所有表	可选的、以逗号分隔的正则表达式列表，与要包含在快照中的表的完全限定名称(<***schemaName***>.<***tableName***>)匹配。指定的项目必须在连接器的 **table.include.list** 属性中命名。只有在连接器的 **snapshot.mode** 属性设置为除 **never** 的值时，此属性才会生效。此属性不会影响增量快照的行为。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。
snapshot.fetch.size	**2000**	在快照期间，连接器以每行的批处理读取表内容。此属性指定批处理中的最大行数。
snapshot.lock.timeout.ms	**10000**	正整数值，指定执行快照时要等待的最大时间（以毫秒为单位）。如果连接器无法获取这个间隔的表锁定，则快照会失败。连接器如何执行快照 提供详细信息。其他可能的设置是： **0** - 当无法获得锁定时连接器会立即失败。 **-1** - 连接器会无限期等待。

属性	默认	描述
snapshot.select.statement.overrides	没有默认值	指定要包含在快照中的表行。如果您希望快照只包含表中的行的子集，请使用属性。此属性仅影响快照。它不适用于连接器从日志中读取的事件。 该属性包含以逗号分隔的、完全限定表名称列表，格式为 < ***schemaName>.<tableName>***。例如， **"snapshot.select.statement.overrides": "inventory.products,customers.orders"** 对于列表中的每个表，添加一个进一步的配置属性，用于指定连接器在获取快照时要在表上运行的 **SELECT** 语句。指定的 **SELECT** 语句决定了快照中包含的表行的子集。使用以下格式指定此 **SELECT** 语句属性的名称： **snapshot.select.statement.overrides.*<schemaName> . <tableName>***。例如，**snapshot.select.statement.overrides.customers.orders**。 Example: 在包含 soft-delete 列 **delete_flag** 的 **customers.orders** 表中，如果您希望快照只包含没有软删除的记录，请添加以下属性： "snapshot.select.statement.overrides": "customer.orders", "snapshot.select.statement.overrides.customer.orders": "SELECT * FROM [customers].[orders] WHERE delete_flag = 0 ORDER BY id DESC" 在生成的快照中，连接器只包括 **delete_flag = 0** 的记录。
provide.transaction.metadata	**false**	确定连接器是否生成带有事务边界的事件，并使用事务元数据增强更改事件信。如果您希望连接器进行此操作，请指定 **true**。详情请参阅 Transaction metadata。

属性	默认	描述
skipped.operations	**t**	在流过程中将跳过的操作类型的逗号分隔列表。操作包括：用于 inserts/create、**u** 表示更新、**d** 表示 delete、**t** 表示 truncates，**none** 不跳过任何操作。**默认情况下，会跳过 truncate 操作（不会由此连接器发出）。**
signal.data.collection	没有默认值	用于向连接器发送信号的数据收集的完全限定名称。https://access.redhat.com/documentation/zh-cn/red_hat_integration/2023.q4/html-single/debezium_user_guide/index#debezium-signaling-enabling-source-signaling-channel使用以下格式指定集合名称：<***schemaName> . < tableName>***
signal.enabled.channels	source	为连接器启用的信号频道名称列表。默认情况下，以下频道可用： • **source** • **kafka** • **file** • **jmx**
notification.enabled.channels	没有默认值	为连接器启用的通知频道名称列表。默认情况下，以下频道可用： • **sink** • **log** • **jmx**
incremental.snapshot.chunk.size	**1024**	连接器在增量快照块期间获取并读取内存的最大行数。增加块大小可提高效率，因为快照会运行更多大小的快照查询。但是，较大的块大小还需要更多内存来缓冲快照数据。将块大小调整为提供环境中最佳性能的值。
topic.naming.strategy	**io.debezium.schema.SchemaTopicNamingStrategy**	应该用来确定数据更改、模式更改、事务、心跳事件等的 TopicNamingStrategy 类的名称，默认为 **SchemaTopicNamingStrategy**。
topic.delimiter	**.**	指定主题名称的分隔符，默认为 **.**。

属性	默认	描述
topic.cache.size	10000	在绑定的并发哈希映射中用于保存主题名称的大小。此缓存将有助于确定与给定数据收集对应的主题名称。
topic.heartbeat.prefix	__debezium-heartbeat	控制连接器向其发送心跳信息的主题名称。主题名称具有此模式： *topic.heartbeat.prefix.topic.prefix* 例如，如果主题前缀是 **fulfillment**，则默认主题名称为 **__debezium-heartbeat.fulfillment**。
topic.transaction	Transactions	控制连接器向其发送事务元数据消息的主题名称。主题名称具有此模式： *topic.prefix.topic.transaction* 例如，如果主题前缀是 **fulfillment**，默认的主题名称为 **fulfillment.transaction**。
snapshot.max.threads	1	指定连接器执行初始快照时使用的线程数量。要启用并行初始快照，请将属性设置为大于 1 的值。在并行初始快照中，连接器会同时处理多个表。  **重要** 并行初始快照只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。
errors.max.retries	-1	在失败前，retriable 错误（如连接错误）的最大重试次数(-1 = no limit, 0 = disabled, > 0 = num of retries)。

Debezium 连接器数据库架构历史记录配置属性

Debezium 提供了一组 schema.history.internal.* 属性，用于控制连接器如何与 schema 历史记录主题进行交互。

下表描述了用于配置 Debezium 连接器的 schema.history.internal 属性。

表 3.19. 连接器数据库架构历史记录配置属性

属性	默认	描述
schema.history.internal.kafka.topic	没有默认值	连接器存储数据库 schema 历史记录的 Kafka 主题的全名。
schema.history.internal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。此连接用于检索之前由连接器存储的数据库架构历史记录，以及用于从源数据库读取的每个 DDL 语句。每个对都应指向 Kafka Connect 进程使用的相同 Kafka 集群。
schema.history.internal.kafka.recovery.poll.interval.ms	**100**	整数值，用于指定连接器在启动/恢复期间应等待的最大毫秒数，同时轮询持久数据。默认值为 100ms。
schema.history.internal.kafka.query.timeout.ms	**3000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端获取集群信息时应等待的最大毫秒数。
schema.history.internal.kafka.create.timeout.ms	**30000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端创建 kafka 历史记录主题时应等待的最大毫秒数。
schema.history.internal.kafka.recovery.attempts	**100**	连接器在连接器恢复失败前应尝试读取持久性历史记录数据的次数上限，并显示错误。接收数据后等待的最大时间为 restore. **attempts recovery. poll.interval.ms**。
schema.history.internal.skip.unparseable.ddl	**false**	指定连接器是否应忽略格式或未知数据库语句的布尔值，或者停止处理，以便人可以解决这个问题。安全默认值为 **false**。跳过应只用于小心，因为在处理 binlog 时可能会导致数据丢失或中断。

属性	默认	描述
schema.history.internal.store.only.captured.tables.ddl	**false**	一个布尔值，用于指定连接器是否记录来自 schema 或数据库中的所有表的模式结构，还是仅从为捕获的表中指定的表。 指定以下值之一： **false （默认）** 在数据库快照过程中，连接器会记录数据库中所有非系统表的 schema 数据，包括没有指定用于捕获的表。最好保留默认设置。如果您稍后决定捕获您最初未指定用于捕获的表的更改，则连接器可以轻松地开始从这些表中捕获数据，因为它们的模式结构已经存储在 schema 历史记录主题中。Debezium 需要表的 schema 历史记录，以便它可以识别发生更改事件时存在的结构。 **true** 在数据库快照过程中，连接器只记录 Debezium 捕获更改事件的表模式。如果您更改了默认值，稍后将连接器配置为从数据库中其他表捕获数据，则连接器缺少从表中捕获更改事件所需的 schema 信息。
schema.history.internal.store.only.captured.databases.ddl	**false**	一个布尔值，用于指定连接器是否记录来自数据库实例中的所有逻辑数据库的架构结构。 指定以下值之一： **true** 连接器只记录 Debezium 捕获更改事件的逻辑数据库和模式中的表的架构结构。 **false** 连接器记录所有逻辑数据库的模式结构。 **注意** MySQL Connector 的默认值为 **true**

配置制作者和消费者客户端的直通数据库架构历史记录属性

Debezium 依赖于 Kafka producer 将模式更改写入数据库架构历史记录主题。同样，它依赖于 Kafka 使用者在连接器启动时从数据库 schema 历史记录主题中读取。您可以通过将值分配给以 schema.history.internal.producer 和 schema.history.internal.consumer ruby 前缀开头的 pass-through 配置属性来定义 Kafka producer 和 消费者 客户端的配置。直通生成者和消费者数据库模式历史记录属性控制一系列行为，如这些客户端与 Kafka 代理的连接的方式，如下例所示：

```
schema.history.internal.producer.security.protocol=SSL
schema.history.internal.producer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.producer.ssl.keystore.password=test1234
schema.history.internal.producer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.producer.ssl.truststore.password=test1234
```

```
schema.history.internal.producer.ssl.key.password=test1234

schema.history.internal.consumer.security.protocol=SSL
schema.history.internal.consumer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.consumer.ssl.keystore.password=test1234
schema.history.internal.consumer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.consumer.ssl.truststore.password=test1234
schema.history.internal.consumer.ssl.key.password=test1234
```

Debezium 从属性名称中剥离前缀，然后再将属性传递给 Kafka 客户端。

如需有关 Kafka producer 配置属性和 Kafka 使用者配置属性的更多详情，请参阅 Kafka 文档。

Debezium 连接器 Kafka 信号配置属性

Debezium 提供了一组 signal.* 属性，用于控制连接器如何与 Kafka 信号主题进行交互。

下表描述了 Kafka 信号 属性。

表 3.20. Kafka 信号配置属性

属性	默认	描述
signal.kafka.topic	<topic.prefix>-signal	连接器监控用于临时信号的 Kafka 主题的名称。 **注意** 如果禁用了 自动主题创建，您必须手动创建所需的信号主题。需要信号主题来保留信号排序。信号主题必须具有单个分区。
signal.kafka.groupId	kafka-signal	Kafka 用户使用的组 ID 的名称。
signal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。每个对都引用 Debezium Kafka Connect 进程使用的 Kafka 集群。
signal.kafka.poll.timeout.ms	100	一个整数值，用于指定连接器在轮询信号时等待的最大毫秒数。

Debezium 连接器传递信号 Kafka 使用者客户端配置属性

Debezium 连接器为信号 Kafka 使用者提供直通配置。透传信号属性以 signals.consumer.* 前缀开始。例如，连接器将 signal.consumer.security.protocol=SSL 等属性传递给 Kafka 消费者。

Debezium 从属性中剥离前缀，然后再将属性传递给 Kafka 信号消费者。

Debezium 连接器接收器通知配置属性

下表描述了 通知 属性。

表 3.21. sink 通知配置属性

属性	默认	描述
notification.sink.topic.name	没有默认值	从 Debezium 接收通知的主题名称。当您将 notification.enabled.channels 属性配置为将 sink 作为启用的通知频道之一时，需要此属性。

Debezium 连接器透传数据库驱动程序配置属性

Debezium 连接器为数据库驱动程序的直通配置提供。直通数据库属性以前缀 driver metric 开头。例如，连接器将 driver.foobar=false 等属性传递给 JDBC URL。

与 数据库架构历史记录客户端通过直通属性 一样，Debebe 会在将前缀传递给数据库驱动程序之前从属性中剥离前缀。

3.7. 监控 DEBEZIUM DB2 连接器性能

Debezium Db2 连接器提供了三种指标类型，除了 Apache ZooKeeper、Apache Kafka 和 Kafka Connect 提供的 JMX 指标的内置支持之外。

- 快照指标 提供有关执行快照时连接器操作的信息。
- 流指标 在连接器捕获更改和流更改事件记录时提供有关连接器操作的信息。
- 模式历史记录指标 提供有关连接器模式历史记录状态的信息。

Debezium 监控文档 提供了如何使用 JMX 公开这些指标的详细信息。

3.7.1. 在 Db2 数据库的快照过程中监控 Debezium

MBean 是 debezium.db2:type=connector-metrics,context=snapshot,server= <*topic.prefix*>。

快照指标不会公开，除非快照操作处于活跃状态，或者快照自上次连接器启动以来发生。

下表列出了可用的 shapshot 指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个快照事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上次启动或重置后看到的事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在快照和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	队列的空闲容量，用于在快照和主 Kafka Connect 循环之间传递事件。
TotalTableCount	int	包括在快照中的表的总数。
RemainingTableCount	int	快照必须复制的表数。
SnapshotRunning	布尔值	快照是否已启动。
SnapshotPaused	布尔值	快照是否已暂停。
SnapshotAborted	布尔值	快照是否中止。
SnapshotCompleted	布尔值	快照是否完成。

属性	类型	描述
SnapshotDurationInSeconds	long	快照为止所花费的秒数，即使未完成也是如此。也包括快照暂停的时间。
SnapshotPausedDurationInSeconds	long	快照暂停的秒数。如果快照暂停几次，暂停的时间会添加。
RowsScanned	Map<String, Long>	包含快照中每个表的行数的映射。表会在处理过程中逐步添加到映射中。更新每个 10,000 行扫描并在完成表后。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

连接器还在执行增量快照时提供以下额外快照指标：

属性	类型	描述
ChunkId	字符串	当前快照块的标识符。
ChunkFrom	字符串	定义当前块的主密钥集的下限。
ChunkTo	字符串	定义当前块的主密钥集的上限。
TableFrom	字符串	当前快照表的主密钥集的下限。
TableTo	字符串	当前快照表的主密钥集的上限。

3.7.2. 监控 Debezium Db2 连接器记录流

MBean 是 debezium.db2:type=connector-metrics,context=streaming,server= <*topic.prefix*>。

下表列出了可用的流指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个流事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上一次启动或指标重置以来看到的事件总数。
TotalNumberOfCreateEventsSeen	long	此连接器自上次启动或指标重置以来看到的创建事件总数。
TotalNumberOfUpdateEventsSeen	long	此连接器自上次启动或指标重置以来看到的更新事件总数。
TotalNumberOfDeleteEventsSeen	long	此连接器自上次启动或指标重置以来看到的删除事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的队列的可用容量。
Connected	布尔值	表示连接器目前是否连接到数据库服务器的标记。
MilliSecondsBehindSource	long	最后一次更改事件时间戳和连接器处理它之间的毫秒数。这些值将讨论运行数据库服务器和连接器的计算机上时钟之间的任何区别。
NumberOfCommittedTransactions	long	已提交的已处理事务的数量。
SourceEventPosition	Map<String, String>	最后收到的事件的协调。

属性	类型	描述
LastTransactionId	字符串	最后处理事务的事务的事务标识符。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

3.7.3. 监控 Debezium Db2 连接器模式历史记录

MBean 是 debezium.db2:type=connector-metrics,context=schema-history,server=<*topic.prefix*>。

下表列出了可用的模式历史记录指标。

属性	类型	描述
Status	字符串	STOPPED 之一，RECOVERING （从存储恢复历史记录），RUNNING 描述数据库架构历史记录的状态。
RecoveryStartTime	long	恢复启动时的 epoch 秒的时间（以秒为单位）。
ChangesRecovered	long	在恢复阶段读取的更改数量。
ChangesApplied	long	恢复和运行时期间应用的模式更改总数。
MilliSecondsSinceLast RecoveredChange	long	从历史记录存储中恢复自上次更改以来经过的毫秒数。
MilliSecondsSinceLastAppliedChange	long	从上次更改被应用后经过的毫秒数。
LastRecoveredChange	字符串	从历史记录存储中恢复的最后一个更改的字符串表示。

属性	类型	描述
LastAppliedChange	字符串	最后应用的更改的字符串表示。

3.8. 管理 DEBEZIUM DB2 连接器

部署 Debezium Db2 连接器后，使用 Debezium 管理 UDF 使用 SQL 命令控制 Db2 复制(ASN)。有些 UDF 期望返回值，在这种情况下，您使用 SQL VALUE 语句调用它们。对于其他 UDF，请使用 SQL CALL 语句。

表 3.22. Debezium 管理 UDF 的描述

任务	命令和备注
启动 ASN 代理	VALUES ASNCDC.ASNCDCSERVICES('start','asncdc');
Stop the ASN 代理	VALUES ASNCDC.ASNCDCSERVICES('stop','asncdc');
检查 ASN 代理的状态	VALUES ASNCDC.ASNCDCSERVICES('status','asncdc');
Put a table into capture mode	CALL ASNCDC.ADDTABLE ('MYSCHEMA', 'MYTABLE'); 将 MYSCHEMA 替换为包含您要放入捕获模式的 schema 的名称。同样，将 MYTABLE 替换为要放入捕获模式的表的名称。
从捕获模式中删除表	CALL ASNCDC.REMOVETABLE('MYSCHEMA', 'MYTABLE');
重新初始化 ASN 服务	VALUES ASNCDC.ASNCDCSERVICES ('reinit','asncdc'); 在将表置于捕获模式后或从捕获模式中删除表后执行这个操作。

3.9. 在 DEBEZIUM 连接器的捕获模式中更新 DB2 表的模式

虽然 Debezium Db2 连接器可以捕获模式更改，以更新模式，但您必须与数据库管理员合作，以确保连接器继续生成更改事件。这是 Db2 实施复制的方式所需要的。

对于捕获模式中的每个表，Db2 中的复制功能会创建一个 change-data 表，其中包含该源表的所有更改。但是 change-data 表模式是静态的。如果以捕获模式为表更新模式，那么您还必须更新其对应 change-data 表的模式。Debezium Db2 连接器无法做到这一点。具有升级特权的数据库管理员必须更新处于捕获模式的表的模式。

警告

在同一表中有新的模式更新前，完全执行模式更新非常重要。因此，建议只在单一批处理中执行所有 DDL，因此仅执行 schema 更新过程。

更新表模式通常有两个步骤：

- 离线 - 在 Debezium 停止时执行
- 在线 - 在 Debezium 运行时执行

每种方法都有优缺点。

3.9.1. 为 Debezium Db2 连接器执行离线 schema 更新

在执行离线 schema 更新前，您可以停止 Debezium Db2 连接器。虽然这是安全的模式更新过程，但可能不适用于具有高可用性要求的应用程序。

先决条件

- 处于捕获模式的一个或多个表需要 schema 更新。

流程

1. 暂停更新数据库的应用程序。
2. 等待 Debezium 连接器流传输所有未流更改事件记录。
3. 停止 Debezium 连接器。

4. 对源表 schema 应用所有更改。

5. 在 ASN 注册表中，将更新的模式标记为 INACTIVE。

6. 重新初始化 ASN 捕获服务。

7. 通过运行 Debezium UDF 从捕获模式中删除表，将带有旧模式的源表从捕获模式中删除。

8. 通过运行 Debezium UDF 将源表添加到捕获模式，方法是运行 Debezium UDF 以将表添加到捕获模式。

9. 在 ASN 注册表中，将更新的源表标记为 ACTIVE。

10. 重新初始化 ASN 捕获服务。

11. 恢复更新数据库的应用程序。

12. 重启 Debezium 连接器。

3.9.2. 为 Debezium Db2 连接器执行在线 schema 更新

在线 schema 更新不会导致应用程序和数据处理的停机时间。也就是说，在执行在线 schema 更新前，不会停止 Debezium Db2 连接器。另外，在线模式更新步骤比离线 schema 更新的步骤简单。

但是，当表处于捕获模式时，在更改为列名称后，Db2 复制功能将继续使用旧列名称。新的列名称不会出现在 Debezium 更改事件中。您必须重启连接器来查看更改事件中的新列名称。

先决条件

- 处于捕获模式的一个或多个表需要 schema 更新。

在表末尾添加列时的步骤

1. 锁定您要更改其模式的源表。
2. 在 ASN 注册表中，将锁定的表标记为 INACTIVE。
3. 重新初始化 ASN 捕获服务。
4. 将所有更改应用到源表的 schema。
5. 将所有更改应用到对应的 change-data 表的 schema。
6. 在 ASN 注册表中，将源表标记为 ACTIVE。
7. 重新初始化 ASN 捕获服务。
8. 可选。重启连接器，查看更改事件中更新的列名称。

在表的中间添加列时的步骤

1. 锁定要更改的源表。
2. 在 ASN 注册表中，将锁定的表标记为 INACTIVE。
3. 重新初始化 ASN 捕获服务。
4. 对于要更改的每个源表：
   a. 在 source 表中导出数据。
   b. 截断源表。

c. 更改源表并添加列。

d. 将导出的数据加载到更改的源表中。

e. 在源表对应的 change-data 表中导出数据。

f. 截断 change-data 表。

g. 更改 change-data 表并添加列。

h. 将导出的数据加载到更改的 change-data 表中。

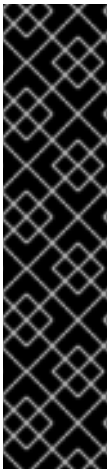
5. 在 ASN 注册表中，将表标记为 INACTIVE。这会将旧的 change-data 表标记为不活动状态，允许其中的数据保留，但不再更新它们。

6. 重新初始化 ASN 捕获服务。

7. 可选。重启连接器，查看更改事件中更新的列名称。

第 4 章 JDBC 的 DEBEZIUM 连接器（开发者预览）

Debezium JDBC 连接器是一个 Kafka Connect sink 连接器实现，它可以使用多个源主题的事件，然后使用 JDBC 驱动程序将这些事件写入关系数据库。这个连接器支持各种数据库划分，包括 Db2、MySQL、Oracle、PostgreSQL 和 SQL Server。



重要

Debezium JDBC 连接器是开发者预览软件。红帽以任何方式支持开发人员预览软件，且功能不完整或生产就绪。对于生产环境或关键业务工作负载，不要使用开发人员预览软件。开发人员预览软件可提前访问即将发布的产品软件。客户可以使用此软件测试功能并在开发过程中提供反馈。此软件可能会随时更改或删除，并收到有限的测试。

有关 Red Hat Developer Preview 软件支持范围的更多信息，请参阅 开发人员预览支持范围。

4.1. DEBEZIUM JDBC 连接器的工作方式

Debezium JDBC 连接器是一个 Kafka Connect sink 连接器，因此需要 Kafka Connect 运行时。连接器会定期轮询订阅的 Kafka 主题，使用来自这些主题的事件，然后将事件写入配置的关联数据库。连接器使用 upsert 语义和基本模式演进来支持幂等写入操作。

Debezium JDBC 连接器提供以下功能：

- 第 4.1.1 节 “Debezium JDBC 连接器如何使用复杂更改事件的描述”
- 第 4.1.2 节 “Debezium JDBC 连接器在交付时描述”
- 第 4.1.3 节 “Debezium JDBC 使用多个任务的描述”
- 第 4.1.4 节 “Debezium JDBC 连接器数据和列类型映射的描述”
- 第 4.1.5 节 “有关 Debezium JDBC 连接器如何处理源事件中主密钥的描述”
-

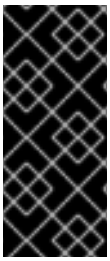
第 4.1.6 节 “将 Debezium JDBC 连接器配置为在消耗 DELETE 或 *tombstone* 事件时删除行”

- 第 4.1.7 节 “启用连接器来执行幂等写入”
- 第 4.1.8 节 “Debezium JDBC 连接器的 schema evolution 模式”
- 第 4.1.9 节 “指定选项来定义目标表和列名称的字母大小写”

4.1.1. Debezium JDBC 连接器如何使用复杂更改事件的描述

默认情况下，Debezium 源连接器会生成复杂的分层更改事件。当 Debezium 连接器与其他 JDBC sink 连接器实现一起使用时，您可能需要应用 ExtractNewRecordState 单一消息转换(SMT)来扁平化更改事件的有效负载，以便 sink 实现可以消耗它们。如果您运行 Debezium JDBC sink 连接器，则不需要部署 SMT，因为 Debezium sink 连接器可以直接使用原生 Debezium 更改事件，而无需使用转换。

当 JDBC sink 连接器使用来自 Debezium 源连接器的复杂更改事件时，它会从原始 插入 或更新 事件的 after 部分中提取值。当接收器(sink)连接器消耗 delete 事件时，不会查询事件有效负载的一部分。



重要

Debezium JDBC sink 连接器没有设计为从架构更改主题中读取。如果您的源连接器被配置为捕获模式更改，在 JDBC 连接器配置中，设置 topics 或 topics.regex 属性，以便连接器不会使用来自架构更改主题。

4.1.2. Debezium JDBC 连接器在交付时描述

Debezium JDBC sink 连接器保证了从 Kafka 主题使用的事件至少被处理一次。

4.1.3. Debezium JDBC 使用多个任务的描述

您可以在多个 Kafka Connect 任务中运行 Debezium JDBC sink 连接器。要在多个任务中运行连接器，请将 tasks.max 配置属性设置为您希望连接器使用的任务数量。Kafka Connect 运行时启动指定任务数量，并为每个任务运行一个连接器实例。多个任务可以通过从并行多个源主题读取和处理更改来提高性能。

4.1.4. Debezium JDBC 连接器数据和列类型映射的描述

要启用 Debezium JDBC sink 连接器，可以正确地将数据类型从入站消息字段映射到出站消息字段，连接器需要源事件中存在的每种字段的数据类型信息。连接器在不同的数据库间支持广泛的列类型映射。要正确从事件字段中的 类型 元数据转换 destination 列类型，连接器会应用为源数据库定义的数据类型映射。您可以通过在源连接器配置中设置 column.propagate.source.type 或 datatype.propagate.source.type 选项来提高连接器为列解析数据类型的方式。当您启用这些选项时，Debezium 包含额外的参数元数据，它可帮助 JDBC sink 连接器更准确地解析目的地列的数据类型。

要使 Debezium JDBC sink 连接器处理 Kafka 主题的事件，当存在时 Kafka 主题消息密钥必须是原语数据类型或 Struct。此外，源消息的有效负载必须是 Struct，它具有没有嵌套结构类型的扁平化结构，或者符合 Debezium 复杂分层结构的嵌套结构布局。

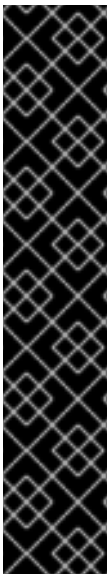
如果 Kafka 主题中的事件结构没有遵循这些规则，您必须实现自定义单个消息转换，将源事件的结构转换为可用格式。

4.1.5. 有关 Debezium JDBC 连接器如何处理源事件中主密钥的描述

默认情况下，Debezium JDBC sink 连接器不会将源事件中的任何字段转换为事件的主键。不幸的是，缺少 stable 主键可能会使事件处理复杂，具体取决于您的业务需求，或者 sink 连接器使用 upsert 语义。要定义一致的主密钥，您可以将连接器配置为使用下表中描述的主要密钥模式之一：

模式	描述
none	创建表时没有指定主键字段。
kafka	主键由以下三列组成： • **__connect_topic** • **__connect_partition** • **__connect_offset** 这些列的值来自 Kafka 事件的协调。
record_key	主键由 Kafka 事件的密钥组成。 如果 primary 键是一个原语类型，请通过设置 **primary.key.fields** 属性指定要使用的列的名称。如果主键是 struct 类型，则 struct 中的字段将映射为主密钥的列。您可以使用 **primary.key.fields** 属性将主键限制为列的子集。

模式	描述
record_value	主键由 Kafka 事件的值组成。 因为 Kafka 事件的值始终是一个 **Struct**，所以默认情况下，值中的所有字段将变为主键的列。要使用主键中的字段子集，请设置 **primary.key.fields** 属性，以指定您要从中派生主键列的值中以逗号分隔的字段列表。



重要

如果将 primary.key.mode 设置为 kafka，并将 schema.evolution 设置为 basic，则一些数据库中断可能会抛出异常。当 dialect 将 STRING 数据类型映射到变量长度字符串数据类型时（如 TEXT 或 CLOB），而 dialect 不允许主键列具有未绑定长度。要避免这个问题，请在您的环境中应用以下设置：

- **不要将 schema.evolution 设置为 basic。**
- **事先创建数据库表和主密钥映射。**

4.1.6. 将 Debezium JDBC 连接器配置为在消耗 DELETE 或 *tombstone* 事件时删除行

当消耗 DELETE 或 *tombstone* 事件时，Debezium JDBC sink 连接器可以删除目标数据库中的行。默认情况下，JDBC sink 连接器不会启用删除模式。

如果要连接器删除行，则必须在连接器配置中明确设置 delete.enabled=true。要使用此模式，还必须将 primary.key.fields 设置为 none 以外的值。上述配置是必要的，因为根据主密钥映射执行删除，因此如果目标表没有主键映射，则连接器无法删除行。

4.1.7. 启用连接器来执行幂等写入

Debezium JDBC sink 连接器可以执行幂等写入，使其可以重复重新执行相同的记录，而不更改最终数据库状态。

要启用连接器来执行幂等写入，您必须将连接器的 insert.mode 明确设置为 upsert。根据指定的 主密钥是否已存在，upsert 操作将作为更新或 插入 来应用。

如果主键值已存在，则操作会更新行中的值。如果指定的主键值不存在，则 插入 会添加一个新行。

每个数据库划分都以不同的方式处理幂等写入，因为 *upsert* 操作没有 SQL 标准。下表显示了 Debezium 支持的 数据库的 upsert DML 语法：

dialect	UPSERT 语法
Db2	**MERGE ...**
MySQL	**在重复的密钥更新 ... 上插入 ...**
Oracle	**MERGE ...**
PostgreSQL	**在冲突的 ... DO UPDATE SET ...**
SQL Server	**MERGE ...**

4.1.8. Debezium JDBC 连接器的 schema evolution 模式

您可以将以下模式演进模式与 Debezium JDBC sink 连接器一起使用：

模式	描述
none	连接器不执行任何 DDL 模式演进。
基本的	连接器自动检测事件有效负载中的字段，但目标表中不存在。连接器更改目标表以添加新字段。

当 schema.evolution 设置为 basic 时，连接器会根据传入事件的结构自动创建或修改目标数据库表。

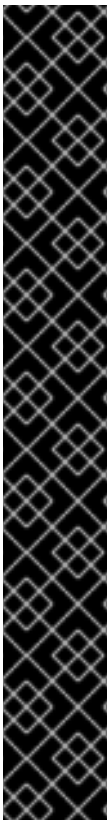
当第一次从主题收到事件时，目标表尚不存在，Debezium JDBC sink 连接器使用事件的密钥，或记录的 schema 结构来解决表的列结构。如果启用了 schema evolution，连接器会在将 DML 事件应用到目标表前准备并执行 CREATE TABLE SQL 语句。

当 Debezium JDBC 连接器从主题接收事件时，如果记录的模式结构与目标表的 schema 结构不同，则连接器使用事件的密钥或其架构结构来识别哪些列是新的列，且必须添加到数据库表中。如果启用了

schema evolution，连接器会在将 DML 事件应用到目标表前准备并执行 ALTER TABLE SQL 语句。由于更改列数据类型、丢弃列和调整主键可以被视为危险的操作，因此禁止连接器执行这些操作。

每个字段的 schema 确定列是否为 NULL 还是 NOT NULL。架构还定义每个列的默认值。如果连接器试图创建带有 nullability 设置的表或不需要的默认值，则必须在接收器连接器处理事件前手动创建表，或者调整相关字段的 schema。要调整 nullability 设置或默认值，您可以引入一个自定义单一消息转换来应用管道中的更改，或修改源数据库中定义的列状态。

字段的数据类型根据预定义的映射集合来解决。如需更多信息，请参阅 第 4.2 节 “Debezium JDBC 连接器如何映射数据类型”。



重要

当您向目标数据库中已存在的表的事件结构引入新字段时，您必须将新字段定义为可选，或者字段必须在数据库 schema 中指定默认值。如果要从目标表中删除字段，请使用以下选项之一：

- 手动删除字段。
- 丢弃列。
- 为字段分配默认值。
- 将字段定义为可为空。

4.1.9. 指定选项来定义目标表和列名称的字母大小写

Debezium JDBC sink 连接器通过构建 DDL (schema 更改)或 DML （数据更改）在目标数据库上执行的 SQL 语句来消耗 Kafka 信息。默认情况下，连接器使用源主题的名称和事件字段作为目标表中的表和列名称的基础。构建的 SQL 不会自动使用引号限定标识符，以保留原始字符串的大小写情况。因此，默认情况下，目标数据库中表或列名称的文本大小完全取决于在未指定问题单时数据库如何处理名称字符串。

例如，如果目标数据库划分是 Oracle，且事件的主题为，则目标 表将创建为 ORDERS，因为 Oracle 在名称没有加引号时默认为大写名称。同样，如果目标数据库划分是 PostgreSQL，且事件的主题为 ORDERS，则目标表将以 订购 的形式创建，因为 PostgreSQL 在名称没有加引号时默认为小写名称。

要明确保留 Kafka 事件中存在的表和字段名称的大小写，在连接器配置中，将 quote.identifiers 属性的值设置为 true。当设定此选项时，当传入的事件用于名为 orders 的主题，目标数据库划分是 Oracle 时，连接器会创建一个名称 orders 的表，因为构建的 SQL 将表的名称定义为 "orders "。当连接器创建列名称时，启用引用会导致行为相同。

4.2. DEBEZIUM JDBC 连接器如何映射数据类型

Debezium JDBC sink 连接器使用逻辑或原语类型解析列的数据类型。原语类型包括整数、浮点、布尔值、字符串和字节数等值。通常，这些类型只通过特定的 Kafka Connect Schema 类型代码表示。逻辑数据类型更为复杂的类型，包括 Struct （基于 Struct）类型，其具有固定的字段名称和模式，或者以特定编码表示的值，如自 epoch 起的天数。

以下示例演示了原语和逻辑数据类型的代表结构：

Primitive 字段 schema

```
{
  "schema": {
    "type": "INT64"
  }
}
```

逻辑字段模式

```
[
  "schema": {
    "type": "INT64",
    "name": "org.apache.kafka.connect.data.Date"
  }
]
```

Kafka Connect 不是这些复杂逻辑类型的唯一源。实际上，Debezium 源连接器生成更改事件，它们具有类似逻辑类型的字段，以表示各种不同的数据类型，包括但不限于、时间戳、日期甚至 JSON 数据。

Debezium JDBC sink 连接器使用这些原语和逻辑类型，将列的类型解析为 JDBC SQL 代码，后者代表列的类型。然后，底层 Hibernate 持久性框架使用这些 JDBC SQL 代码，将列的类型解析为使用中的逻辑数据类型。下表说明了 Kafka Connect 和 JDBC SQL 类型之间以及 Debezium 和 JDBC SQL 类型之间的原语和逻辑映射。实际的最后一列类型因每种数据库类型而异。

1. 表 4.1 “Kafka Connect Primitives 和 Column 数据类型之间的映射”
2. 表 4.2 “Kafka Connect 逻辑类型和列数据类型之间的映射”
3. 表 4.3 “Debezium 逻辑类型和列数据类型之间的映射”
4. 表 4.4 “Debezium 特定逻辑类型和 Column 数据类型之间的映射”

表 4.1. Kafka Connect Primitives 和 Column 数据类型之间的映射

原语类型	JDBC SQL 类型
INT8	Type.TINYINT
INT16	Type.SMALLINT
INT32	Type.INTEGER
INT64	Type.BIGINT
FLOAT32	Type.FLOAT
FLOAT64	Type.DOUBLE
布尔值	Type.BOOLEAN
字符串	Type.CHAR, Types.NCHAR, Types.VARCHAR, Types.NVARCHAR
BYTES	类型.VARBINARY

表 4.2. Kafka Connect 逻辑类型和列数据类型之间的映射

逻辑类型	JDBC SQL 类型
org.apache.kafka.connect.data.Decimal	Type.DECIMAL

逻辑类型	JDBC SQL 类型
org.apache.kafka.connect.data.Date	type.DATE
org.apache.kafka.connect.data.Time	Type.TIMESTAMP
org.apache.kafka.connect.data.Timestamp	Type.TIMESTAMP

表 4.3. Debezium 逻辑类型和列数据类型之间的映射

逻辑类型	JDBC SQL 类型
io.debezium.time.Date	type.DATE
io.debezium.time.Time	Type.TIMESTAMP
io.debezium.time.MicroTime	Type.TIMESTAMP
io.debezium.time.NanoTime	Type.TIMESTAMP
io.debezium.time.ZonedTime	Types.TIME_WITH_TIMEZONE
io.debezium.time.Timestamp	Type.TIMESTAMP
io.debezium.time.MicroTimestamp	Type.TIMESTAMP
io.debezium.time.NanoTimestamp	Type.TIMESTAMP
io.debezium.time.ZonedTimestamp	Types.TIMESTAMP_WITH_TIMEZONE
io.debezium.data.VariableScaleDecimal	Type.DOUBLE

重要

如果数据库不支持带有时区的时间或时间戳，则映射会解析到没有时区的等效时间。

表 4.4. Debezium 特定逻辑类型和 Column 数据类型之间的映射

逻辑类型	MySQL SQL 类型	PostgreSQL SQL 类型	SQL Server SQL 类型
io.debezium.data.Bits	**bit(n)**	**位(n) 或 位的不同**	**varbinary(n)**
io.debezium.data.Enum	**Enum**	Type.VARCHAR	不适用

逻辑类型	MySQL SQL 类型	PostgreSQL SQL 类型	SQL Server SQL 类型
io.debezium.data.Json	**json**	**json**	不适用
io.debezium.data.EnumSet	**set**	不适用	不适用
io.debezium.time.Year	**year (n)**	不适用	不适用
io.debezium.time.MicroDuration	不适用	**interval**	不适用
io.debezium.data.Ltree	不适用	**ltree**	不适用
io.debezium.data.Uuid	不适用	**uuid**	不适用
io.debezium.data.Xml	不适用	**xml**	**xml**

除了上述原语和逻辑映射外，如果更改事件的来源是 Debezium 源连接器、列类型的解析及其长度、精度和规模，可以通过启用列或数据类型传播来进一步影响。要强制传播，必须在源连接器配置中设置以下属性之一：

- **column.propagate.source.type**
- **datatype.propagate.source.type**

Debezium JDBC sink 连接器应用具有较高优先级的值。

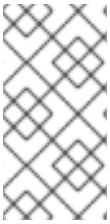
例如，假设更改事件中包含以下字段模式：

Debezium 更改事件字段 schema，启用了列或数据类型传播

```
{
  "schema": {
    "type": "INT8",
    "parameters": {
      "__debezium.source.column.type": "TINYINT",
      "__debezium.source.column.length": "1"
```

```
    }
  }
}
```

在上例中，如果没有设置 schema 参数，Debezium JDBC sink 连接器会将此字段映射到 type .SMALLINT。Type.SMALLINT 可以有不同的逻辑数据库类型，具体取决于数据库问题。对于 MySQL，示例中的列类型转换为没有指定长度的 TINYINT 列类型。如果为源连接器启用了列或数据类型传播，Debezium JDBC sink 连接器会使用映射信息来优化数据类型映射过程，并使用类型 TINYINT (1) 创建列。



注意

通常，当源和 sink 数据库使用相同的数据库类型时，使用列或数据类型传播的影响会大得多。

4.3. 部署 DEBEZIUM JDBC 连接器

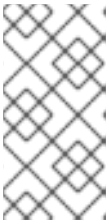
要部署 Debezium JDBC 连接器，请安装 Debezium JDBC 连接器存档，配置连接器，并通过将其配置添加到 Kafka Connect 来启动连接器。

前提条件

- 已安装 Apache ZooKeeper、Apache Kafka 和 Kafka Connect。
- 已安装目标数据库并配置为接受 JDBC 连接。

流程

1. 下载 Debezium JDBC 连接器插件存档。
2. 将文件提取到 Kafka Connect 环境中。
3. （可选）从 Maven Central 下载 JDBC 驱动程序，并将下载的驱动程序文件提取到包含 JDBC sink 连接器 JAR 文件的目录。



注意

JDBC sink 连接器不包括 Oracle 和 Db2 的驱动程序。您必须下载驱动程序并手动安装它们。

4. 将驱动程序 JAR 文件添加到安装了 JDBC sink 连接器的路径中。

5. 确保安装 JDBC sink 连接器的路径是 Kafka Connect plugin.path 的一部分。

6. 重启 Kafka Connect 进程以获取新的 JAR 文件。

4.3.1. Debezium JDBC 连接器配置

通常，您可以通过提交指定连接器的配置属性来注册 Debezium JDBC 连接器。以下示例显示了注册 Debezium JDBC sink 连接器实例的 JSON 请求，该连接器使用来自名为 orders 的事件，以及最常见的配置设置：

示例： Debezium JDBC 连接器配置

```
{
  "name": "jdbc-connector", 1
  "config": {
    "connector.class": "io.debezium.connector.jdbc.JdbcSinkConnector", 2
    "tasks.max": "1", 3
    "connection.url": "jdbc:postgresql://localhost/db", 4
    "connection.username": "pguser", 5
    "connection.password": "pgpassword", 6
    "insert.mode": "upsert", 7
    "delete.enabled": "true", 8
    "primary.key.mode": "record_key", 9
    "schema.evolution": "basic", 10
    "database.time_zone": "UTC" 11
  }
}
```

1 1 1 1 1 1 1 1 1 1 1 1 1 1

使用 Kafka Connect 服务注册时分配给连接器的名称。

2 2 2 2 2 2 2 2 2 2 2 2 2 2

JDBC sink 连接器类的名称。

3 3 3 3 3 3 3 3 3 3 3 3

为此连接器创建的最大任务数量。

4 4 4 4 4 4 4 4 4 4 4 4

连接器用来连接到它写入的 sink 数据库的 JDBC URL。

5 5 5 5 5 5 5 5 5 5 5

用于身份验证的数据库用户的名称。

6 6 6 6 6 6 6

用于身份验证的数据库用户的密码。

7 7 7 7 7 7

连接器使用的 insert.mode。

8 8 8 8 8 8

启用删除数据库中的记录。如需更多信息，请参阅 delete.enabled 配置属性。

9 9 9 9 9 9

指定用于解析主键列的方法。如需更多信息，请参阅 primary.key.mode 配置属性。

10 10 10 10 10 10

启用连接器来演进目标数据库的模式。如需更多信息，请参阅 schema.evolution 配置属性。

11 11 11 11

指定编写临时字段类型时使用的时区。

有关您可以为 Debezium JDBC 连接器设置的配置属性的完整列表，请参阅 JDBC 连接器属性。

您可以使用 POST 命令将此配置发送到正在运行的 Kafka Connect 服务。服务记录配置并启动执行以下操作的接收器连接器任务：

- 连接到数据库。
- 使用订阅的 Kafka 主题的事件。
- 将事件写入配置的数据库。

4.4. DEBEZIUM JDBC 连接器配置属性的描述

Debezium JDBC sink 连接器有几个配置属性，可用于实现满足您的需求的连接器行为。许多属性都有默认值。有关属性的信息组织如下：

- JDBC 连接器通用属性
- JDBC 连接器连接属性
- JDBC 连接器运行时属性
- JDBC 连接器可扩展属性

表 4.5. 通用属性

属性	默认	描述
name	没有默认值	连接器的唯一名称。如果您在注册连接器时尝试重复使用此名称，则失败结果。所有 Kafka Connect 连接器都需要此属性。

属性	默认	描述
connector.class	没有默认值	连接器的 Java 类的名称。对于 Debezium JDBC 连接器，请指定值 **io.debezium.connector.jdbc.JdbcSinkConnector**。
tasks.max	1	此连接器使用的最大任务数量。
topics	没有默认值	要使用的主题列表，用逗号分开。不要将此属性与 topics.regex 属性结合使用。
topics.regex	没有默认值	指定要使用的主题的正则表达式。在内部，正则表达式编译到 **java.util.regex.Pattern**。不要将此属性与 topics 属性结合使用。

表 4.6. JDBC 连接器连接属性

属性	默认	描述
connection.url	没有默认值	用于连接到数据库的 JDBC 连接 URL。
connection.username	没有默认值	连接器用来连接到数据库的数据库用户帐户的名称。
connection.password	没有默认值	连接器用来连接到数据库的密码。
connection.pool.min_size	**5**	指定池中最小连接数。
connection.pool.min_size	**32**	指定池维护的最大并发连接数。
connection.pool.acquire_increment	**32**	指定当连接池超过其最大大小时，连接器尝试获取的连接数量。
connection.pool.timeout	**1800**	指定在丢弃未使用的连接前保留的秒数。

表 4.7. JDBC 连接器运行时属性

属性	默认	描述
database.time_zone	**UTC**	指定插入 JDBC 临时值时使用的时区。
delete.enabled	**false**	指定连接器是否处理 **DELETE** 或 *tombstone* 事件，并从数据库中删除对应的行。使用此选项要求您将 primary.key.mode 设置为 **record.key**。

属性	默认	描述
insert.mode	insert	指定用于将事件插入到数据库中的策略。可用的选项如下： **insert** 指定所有事件都应构建 **INSERT**- 基于 SQL 语句。仅在没有使用主键时使用这个选项，或者当您可以确定没有更新与现有主键值的行时使用这个选项。 **update** 指定所有事件都应构建 **UPDATE**- 基于 SQL 语句。只有在您可以确定连接器只接收应用到现有行的事件时才使用这个选项。 **upsert** 指定连接器使用 **upsert** 语义向表添加事件。也就是说，如果主密钥不存在，连接器将执行 **INSERT** 操作，如果键存在，则连接器会执行 **UPDATE** 操作。当需要幂等写入时，应该将连接器配置为使用这个选项。
primary.key.mode	none	指定连接器如何从事件解析主键列。 **none** 指定没有创建主键列。 **kafka** 指定连接器使用 Kafka 协调作为主键列。密钥协调从事件的主题名称、分区和偏移中定义，并使用以下名称映射到列： • **__connect_topic** • **__connect_partition** • **__connect_offset** **record_key** 指定从事件的 record 键中提供的主键列。如果记录键是一个原语类型，则需要 **primary.key.fields** 属性来指定主键栏的名称。如果记录键是 struct 类型，则 **primary.key.fields** 属性是可选的，且可用于将事件键中的列指定为表的主键。 **record_value** 指定从事件值中获取主键列。您可以设置 **primary.key.fields** 属性，将主键定义为事件值中的字段子集；否则，所有字段都会默认使用。

属性	默认	描述
primary.key.fields	没有默认值	主键列的名称或以逗号分隔的字段列表，以便从中派生主密钥。 当 primary.key.mode 设置为 record_key 且事件的键是原语类型时，此属性指定要用于键的列名称。 当 primary.key.mode 设置为带有非原始键的 record_key 或 record_value 时，此属性指定来自 key 或 value 的以逗号分隔的字段名称列表。如果 primary.key.mode 设置为带有非原始键的 record_key，或 record_value，且此属性没有指定，则连接器会从记录键或记录值的所有字段生成主密钥，具体取决于指定的模式。
quote.identifiers	false	指定生成的 SQL 语句是否使用引号来分离表和列名称。详情请查看 第 4.1.9 节 “指定选项来定义目标表和列名称的字母大小写” 部分。
schema.evolution	none	指定连接器如何演变目标表模式。如需更多信息，请参阅 第 4.1.8 节 “Debezium JDBC 连接器的 schema evolution 模式”。可用的选项如下： **none** 指定连接器没有演进目的地模式。 **基本的** 指定发生基本演变。连接器通过将传入事件的记录模式与数据库表结构进行比较，在表中添加缺少的列。
table.name.format	${topic}	根据事件的主题名称，指定确定目标表名称如何格式化的字符串。占位符 ${topic} 替换为主题名称。

表 4.8. JDBC 连接器可扩展属性

属性	默认	描述
column.naming.strategy	i.d.c.j.n.DefaultColumnNamingStrategy	指定连接器用来从事件字段名称解析列名称的 ColumnNamingStrategy 实现的完全限定类名称。 默认情况下，连接器使用字段名称作为列名称。

属性	默认	描述
table.naming.strategy	i.d.c.j.n.DefaultTableNamingStrategy	指定 TableNamingStrategy 实现的完全限定类名称，连接器用来从传入的事件主题名称解析表名称。 默认行为是： • 将 table.name.format 配置属性中的 ${topic} 占位符替换为事件的主题。 • 通过将点(.)替换为下划线(_)来清理表名称。

4.5. JDBC 连接器常见问题

ExtractNewRecordState 单个消息转换需要吗？

否，实际上是 Debezium JDBC 连接器与其他实现的不同因素之一。虽然连接器能够像竞争者一样获取扁平化的事件，但它还可以原生处理 Debezium 的复杂更改事件结构，而无需任何特定类型的转换。

如果更改了列的类型，或者列被重命名或丢弃，这是否由 schema evolution 处理？

不，Debezium JDBC 连接器不会对现有列进行任何更改。连接器支持的模式演进非常基本。它只是将事件结构中的字段与表的列列表进行比较，然后添加表中尚未定义为列的任何字段。如果列的类型或默认值更改，连接器不会在目标数据库中调整它们。如果重命名了列，则旧列将按原样保留，连接器会将带有新名称的列附加到表中；但是，在旧列中带有数据的现有行保持不变。这些类型的架构更改应该手动处理。

如果列的类型没有解析为我希望的类型，那么我如何强制映射到不同的数据类型？

Debezium JDBC 连接器使用复杂的类型系统来解析列的数据类型。有关此类型系统如何将特定字段的 schema 定义解析为 JDBC 类型的详情，请查看 第 4.1.4 节 “Debezium JDBC 连接器数据和列类型映射的描述” 部分。如果要应用不同的数据类型映射，请手动定义表来显式获取首选的列类型。

如何在不更改 Kafka 主题名称的情况下为表名称指定前缀或后缀？

要在目标表名称中添加前缀或后缀，请调整 table.name.format connector 配置属性以应用您想要的前缀或后缀。例如，若要使用 jdbc_ 前缀所有表名称，请使用值 jdbc_${topic} 指定 table.name.format 配置属性。如果连接器订阅了名为 orders 的主题，则生成的表将创建为 jdbc_orders。

为什么有些列会自动加引号，即使未启用标识符引用？

在某些情况下，可能会明确引用特定列或表名称，即使未启用 quote.identifiers。当列或表名称以 开头或使用通常被视为非法语法的特定惯例时，这通常是必需的。例如，当将 primary.key.mode 设置为 kafka 时，如果列的名称被引号，一些数据库只允许列的名称以下划线开头。引用行为是特定于临时的，不同的数据库类型会有所不同。

第 5 章 MONGODB 的 DEBEZIUM 连接器

Debezium 的 MongoDB 连接器跟踪 MongoDB 副本集或 MongoDB 分片集群，以记录数据库和集合中的更改，以记录 Kafka 主题中的事件。连接器会自动处理分片集群中添加或删除分片，更改每个副本集的成员资格、每个副本集中的选举，以及等待通信问题的解决。

有关与此连接器兼容的 MongoDB 版本的详情，请查看 Debezium 支持的配置页面。

使用 Debezium MongoDB 连接器的信息和步骤进行组织，如下所示：

5.1. DEBEZIUM MONGODB 连接器概述

MongoDB 的复制机制提供冗余和高可用性，也是在生产环境中运行 MongoDB 的首选方法。MongoDB 连接器捕获副本集或分片集群中的更改。

MongoDB *副本集* 由一组服务器组成，它们都有相同数据的副本，而复制可确保客户端对副本集 *的主* 文档进行的所有更改都正确应用到其他副本集的服务器，称为 ***secondaries***。MongoDB 复制的工作原理是，其 *oplog* 中的更改（或操作日志）可以正常工作，然后每个 secondaries 都会读取主的 oplog，并将所有操作都应用到自己的文档。当新的服务器添加到副本集时，该服务器首先对主设备执行所有数据库

和集合 的快照，然后读取主的 oplog 以应用自开始快照后可能所做的所有更改。当新的服务器捕获到主 oplog 的尾部时，这个新的服务器成为辅助服务器（并且能够处理查询）。

5.1.1. MongoDB 连接器如何使用更改流捕获事件记录的描述

虽然 Debezium MongoDB 连接器没有成为副本集的一部分，但它使用类似的复制机制来获取 oplog 数据。主要区别在于连接器不会直接读取 oplog。相反，它会将 oplog 数据的捕获和解码委派给 MongoDB 更改流 功能。通过更改流，MongoDB 服务器会公开集合中发生的变化作为事件流。Debezium 连接器监控流，然后提供下游更改。连接器第一次检测到副本集，它会检查 oplog 以获取最后记录的事务，然后执行主数据库和集合的快照。在连接器完成复制数据后，它会从之前读取的 oplog 位置创建一个更改流。

当 MongoDB 连接器处理更改时，它会定期记录事件源自于 oplog 流的位置。当连接器停止时，它会记录它处理的最后 oplog 流位置，以便重启后它可以从该位置恢复流。换句话说，可以停止、升级或维护连接器，并在以后重启一段时间，并且始终在不丢失单一事件的情况下准确获取关闭的位置。当然，MongoDB oplogs 通常以最大大小设置，因此如果连接器长时间停止，则 oplog 中的操作可能会在连接器有机会读取它们前清除。在这种情况下，重启连接器检测到缺少的 oplog 操作，执行快照，然后继续流更改。

MongoDB 连接器还接受在副本集的成员资格和领导、分片集群中的分片以及可能导致通信失败的网络问题。连接器始终使用副本集的主节点流更改，因此当选举和不同的节点变为主要节点时，连接器将立即停止流更改，连接到新的主节点，并使用新的主节点开始流更改。同样，如果连接器无法与副本集主通信，它会尝试重新连接（使用 exponential backoff，以便不会大量网络或副本集）。重新建立连接后，连接器会从捕获的最后一个事件继续流更改。这样，连接器会动态调整副本集成员资格中的更改，并自动处理通信中断。

其他资源

- 复制机制
- 副本集
- 副本集选举
- 分片集群
- 分片添加

- 分片删除
- 更改流

5.2. DEBEZIUM MONGODB 连接器的工作方式

连接器支持的 MongoDB 拓扑概述可用于规划应用程序。

配置和部署 MongoDB 连接器时，它首先连接到 seed 地址中的 MongoDB 服务器，并确定每个可用副本集的详情。由于每个副本集都有自己的独立 oplog，因此连接器将尝试为每个副本集使用单独的任务。连接器可以限制其使用的最大任务数量，如果没有足够的任务可用，则连接器会将多个副本集分配给每个任务，尽管该任务仍会为每个副本集使用单独的线程。

注意

当针对分片集群运行连接器时，请使用大于副本集的 tasks.max 值。这将允许连接器为每个副本集创建一个任务，并可让 Kafka Connect 协调、分发和管理所有可用 worker 进程中的任务。

以下主题详细介绍了 Debezium MongoDB 连接器的工作方式：

- 第 5.2.1 节 “Debezium 连接器支持的 MongoDB 拓扑”
- 第 5.2.2 节 “Debezium MongoDB 连接器如何为副本集和分片集群使用逻辑名称”
- 第 5.2.3 节 “Debezium MongoDB 连接器如何执行快照”
- 第 5.2.4 节 “临时快照”
- 第 5.2.5 节 “增量快照”

- 第 5.2.6 节 “Debezium MongoDB 连接器流更改事件记录”
- 第 5.2.8 节 “接收 Debezium MongoDB 更改事件记录的默认 Kafka 主题名称”
- 第 5.2.9 节 “事件密钥控制 Debezium MongoDB 连接器的主题分区”
- 第 5.2.10 节 “Debezium MongoDB 连接器生成的事件代表事务边界”

5.2.1. Debezium 连接器支持的 MongoDB 拓扑

MongoDB 连接器支持以下 MongoDB 拓扑：

MongoDB 副本集

Debezium MongoDB 连接器可以从单个 MongoDB 副本集 捕获更改。生产副本集 至少需要三个成员。

要将 MongoDB 连接器与副本集搭配使用，您必须将连接器配置中的 mongodb.connection.string 属性的值设置为 副本集连接字符串。当连接器准备好从 MongoDB 更改流开始捕获更改时，它会启动一个连接任务。然后，连接任务使用指定的连接字符串来建立与可用副本集成员的连接。

警告

由于连接器管理数据库连接的方式的变化，此 Debezium 发行版本不再支持使用 mongodb.members.auto.discover 属性，以防止连接器执行成员资格发现。

MongoDB 分片集群

MongoDB 分片的集群 包括：

- 一个或多个 *分片*，每个分片都部署为副本集；

- 充当 *集群配置服务器的*单独副本集

- 客户端需要连接到的一个或多个 *routers* (也称为 mongos)。它们会将请求路由到相关的分片。

要将 MongoDB 连接器与分片集群搭配使用，在连接器配置中，将 mongodb.connection.string 属性的值设置为 分片集群连接字符串。

警告

mongodb.connection.string 属性替换了已弃用的 mongodb.hosts 属性，用于为连接器提供 *配置服务器副本的主机地址*。在当前发行版本中，使用 mongodb.connection.string 为连接器提供 MongoDB 路由器的地址，也称为 mongos。

注意

当连接器连接到分片集群时，它会发现有关代表集群中分片的每个副本集的信息。连接器使用单独的任务来捕获每个分片的更改。当从集群中添加或删除分片时，连接器会动态调整任务数量，以补偿更改。

MongoDB 独立服务器

MongoDB 连接器无法监控独立 MongoDB 服务器的更改，因为独立服务器没有 oplog。如果单机服务器转换为一个带有成员的副本集，则连接器可以正常工作。

注意

MongoDB 不建议在生产环境中运行独立服务器。如需更多信息，请参阅 MongoDB 文档。

5.2.2. Debezium MongoDB 连接器如何为副本集和分片集群使用逻辑名称

连接器配置属性 topic.prefix 充当 MongoDB 副本集或分片集群的*逻辑名称*。连接器以多种方式使用逻辑名称：作为所有主题名称的前缀，并在记录每个副本集的更改流位置时作为唯一标识符。

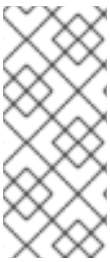
您应该为每个 MongoDB 连接器分配一个唯一的逻辑名称，以有意义的描述源 MongoDB 系统。我们建议逻辑名称以字母或下划线字符开头，以及字母数字字符或下划线的剩余字符。

5.2.3. Debezium MongoDB 连接器如何执行快照

当 Debezium 任务开始使用副本集时，它使用连接器的逻辑名称和副本集名称来查找一个 *偏移* 信息，该偏移之前停止读取更改的位置。如果找到偏移并且仍然存在于 oplog 中，则任务会立即进行 流传输更改，从记录的偏移位置开始。

但是，如果没有找到偏移，或者 oplog 不再包含该位置，则任务必须首先通过执行快照来获取副本集内容的当前状态。*这个过程首先记录 oplog 的当前位置，并记录为偏移（以及表示快照已启动的标记）。然后，该任务会继续复制每个集合，尽可能生成多个线程（最多 snapshot.max.threads 配置属性值）来并行执行此功能。连接器为其看到的每个文档记录一个单独的 读取事件。每个读取事件都包含对象的标识符、对象的完整状态和有关找到对象的 MongoDB 副本集 的源 信息。源信息还包括一个标记，表示该事件是在快照期间生成的。*

此快照将继续，直到它复制了与连接器的过滤器匹配的所有集合。如果在任务快照完成前停止连接器，重启连接器会在重启连接器再次开始快照。



注意

在连接器执行任何副本集的快照时，尝试避免任务重新分配和重新配置。连接器生成日志消息来报告快照的进度。要提供最大的控制，请为每个连接器运行单独的 Kafka Connect 集群。

您可以在以下部分找到有关快照的更多信息：

- *第 8.2.3 节 “临时快照”*
- *第 8.2.4 节 “增量快照”*

5.2.4. 临时快照

默认情况下，连接器仅在首次启动后运行初始快照操作。在正常情况下，在这个初始快照后，连接器不会重复快照过程。连接器捕获的任何更改事件数据都只通过流处理。

然而，在某些情况下，连接器在初始快照期间获得的数据可能会过时、丢失或不完整。为了提供重新捕获集合数据的机制，Debezium 包含一个执行临时快照的选项。数据库中的以下更改可能会导致执行临时快照：

- 连接器配置会被修改来捕获不同的集合集。
- Kafka 主题已删除，必须重建。
- 由于配置错误或某些其他问题导致数据损坏。

您可以通过启动所谓的 临时快照来为之前捕获的集合重新运行快照。临时快照需要使用 信号集合。您可以通过向 Debezium 信号集合发送信号请求来发起临时快照。

当您启动现有集合的临时快照时，连接器会将内容附加到已用于集合的主题。如果删除了之前存在的主题，如果启用了 自动主题创建，Debezium 可以自动创建主题。

临时快照信号指定要包含在快照中的集合。快照可以捕获整个数据库的内容，或者仅捕获数据库中集合的子集。另外，快照也可以捕获数据库中集合内容的子集。

您可以通过将 execute-snapshot 消息发送到信号集合来指定要捕获的集合。将 execute-snapshot 信号类型设置为 增量，并提供快照中包含的集合名称，如下表所述：

表 5.1. 临时 execute-snapshot 信号记录的示例

字段	默认	值
type	**incremental**	指定您要运行的快照类型。 设置类型是可选的。目前，您只能请求 **增量** 快照。
data-collections	*N/A*	包含与要快照的集合的完全限定域名匹配的正则表达式的数组。 名称的格式与 **signal.data.collection** 配置选项的格式相同。
additional-condition	*N/A*	可选字符串，根据集合的列指定条件，用于捕获集合内容的子集。
surrogate-key	*N/A*	可选字符串，指定连接器在快照过程中用作集合的主键的列名称。

触发临时快照

您可以通过向信号集合添加 execute-snapshot 信号类型的条目来发起临时快照。连接器处理消息后，它会开始快照操作。快照进程读取第一个和最后一个主密钥值，并使用这些值作为每个集合的开始和结束点。根据集合中的条目数量以及配置的块大小，Debezium 会将集合划分为块，并一次性执行每个块的快照。

目前，execute-snapshot 操作类型仅触发 增量快照。如需更多信息，请参阅 增加快照。

5.2.5. 增量快照

为了提供管理快照的灵活性，Debezium 包含附加快照机制，称为 增量快照。增量快照依赖于 Debezium 机制 向 Debezium 连接器发送信号。

在增量快照中，除了一次捕获数据库的完整状态，就像初始快照一样，Debebe 会在一系列可配置的块中捕获每个集合。您可以指定您希望快照捕获 的集合以及每个块的大小。块大小决定了快照在数据库的每个获取操作期间收集的行数。增量快照的默认块大小为 1024 行。

当增量快照进行时，Debebe 使用 watermarks 跟踪其进度，维护它捕获的每个集合行的记录。与标准初始快照过程相比，捕获数据的阶段方法具有以下优点：

- 您可以使用流化数据捕获并行运行增量快照，而不是在快照完成前进行后流。连接器会在快照过程中从更改日志中捕获接近实时事件，且操作都不会阻止其他操作。
- 如果增量快照的进度中断，您可以在不丢失任何数据的情况下恢复它。在进程恢复后，快照从停止的时间点开始，而不是从开始获取集合。
- 您可以随时根据需要运行增量快照，并根据需要重复该过程以适应数据库更新。例如，您可以在修改连接器配置后重新运行快照，以将集合添加到其 collection.include.list 属性中。

增量快照过程

当您运行增量快照时，Debezium 会按主密钥对每个集合进行排序，然后根据 配置的块大小 将集合分成块。然后，根据块工作块，它会捕获块中的每个集合行。对于它捕获的每行，快照会发出 READ 事件。该事件代表块的快照开始时的行值。

当快照继续进行时，其他进程可能会继续访问数据库，可能会修改集合记录。为了反映此类更改，INSERT、UPDATE 或 DELETE 操作会按照常常提交到事务日志。同样，持续 Debezium 流进程将

继续检测这些更改事件，并将相应的更改事件记录发送到 Kafka。

Debezium 如何使用相同的主密钥在记录间解决冲突

在某些情况下，streaming 进程发出的 UPDATE 或 DELETE 事件会停止序列。也就是说，流流过程可能会发出一个修改集合行的事件，该事件捕获包含该行的 READ 事件的块。当快照最终为行发出对应的 READ 事件时，其值已被替换。为确保以正确的逻辑顺序处理到达序列的增量快照事件，Debebe 使用缓冲方案来解析冲突。仅在快照事件和流化事件之间发生冲突后，De Debezium 会将事件记录发送到 Kafka。

快照窗口

为了帮助解决修改同一集合行的过期事件和流化事件之间的冲突，Debebe 会使用一个所谓的 快照窗口。快照窗口分解了增量快照捕获指定集合块数据的间隔。在块的快照窗口打开前，Debebe 会使用其常见行为，并将事件从事务日志直接下游发送到目标 Kafka 主题。但从特定块的快照打开后，直到关闭为止，De-duplication 步骤会在具有相同主密钥的事件之间解决冲突。

对于每个数据收集，Debezium 会发出两种类型的事件，并将其存储在单个目标 Kafka 主题中。从表直接捕获的快照记录作为 READ 操作发送。同时，当用户继续更新数据收集中的记录，并且会更新事务日志来反映每个提交，Debezium 会为每个更改发出 UPDATE 或 DELETE 操作。

当快照窗口打开时，Debezium 开始处理快照块，它会向内存缓冲区提供快照记录。在快照窗口期间，缓冲区中 READ 事件的主密钥与传入流事件的主键进行比较。如果没有找到匹配项，则流化事件记录将直接发送到 Kafka。如果 Debezium 检测到匹配项，它会丢弃缓冲的 READ 事件，并将流化记录写入目标主题，因为流的事件逻辑地取代静态快照事件。在块关闭的快照窗口后，缓冲区仅包含 READ 事件，这些事件不存在相关的事务日志事件。Debezium 将这些剩余的 READ 事件发送到集合的 Kafka 主题。

连接器为每个快照块重复这个过程。

警告

增量快照需要预先排序主密钥。但是，字符串 不能保证稳定的排序作为编码，特殊字符可能会导致意外行为(Mongo sort String)。在执行增量快照时，请考虑将其他类型的用于主键。

分片集群的增量快照

分片集群的增量快照是 Debezium MongoDB 连接器的技术预览功能。技术预览功能不被红帽产品服务级别协议(SLA)支持，且可能无法完成。因此，红帽不推荐在生产环境中实施任何技术预览功能。此技术预览功能为您提供对即将推出的产品创新的早期访问，允许您在开发过程中测试并提供反馈。如需有关支持范围的更多信息，请参阅 技术预览功能支持范围。

要将增量快照与分片 MongoDB 集群搭配使用，您必须为以下属性设置特定值：

- 将 mongodb.connection.mode 设置为 sharded。
- 将 incremental.snapshot.chunk.size 设置为一个足够大的值，以便满足更改流管道 的复杂性。

5.2.5.1. 触发增量快照

目前，启动增量快照的唯一方法是向源数据库上的 信号集合发送临时快照 信号。

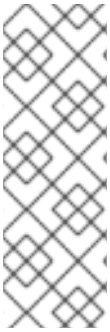
您可以使用 MongoDB insert （） 方法向信号提交信号。

在 Debezium 检测到信号集合中的更改后，它会读取信号并运行请求的快照操作。

您提交的查询指定要包含在快照中的集合，并可以选择指定快照操作的类型。目前，快照操作的唯一有效选项是默认值 incremental。

要指定快照中包含的集合，提供一个 data-collections 数组，它列出用于匹配集合的集合或用于匹配集合的正则表达式数组，例如
{"data-collections": ["public.Collection1", "public.Collection2"]}

增量快照信号的 data-collections 数组没有默认值。如果 data-collections 数组为空，Debezium 会检测到不需要任何操作，且不会执行快照。



注意

如果要包含在快照中的集合名称在数据库、模式或表的名称中包含句点(.)，以将集合添加到 data-collections 数组中，您必须使用双引号转义名称的每个部分。

例如，要包含存在于 公共 数据库中的数据收集，其名称为 My.Collection，请使用以下格式："public"."My.Collection"。

前提条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道来触发增量快照

1. 在信号集合中插入快照信号文档：

```
<signalDataCollection>.insert({"id" : _<idNumber>,"type" : <snapshotType>, "data" :
{"data-collections" ["<collectionName>", "<collectionName>"],"type":
<snapshotType>}});
```

例如，

```
db.debeziumSignal.insert({ (1)
"type" : "execute-snapshot", (2) (3)
"data" : {
"data-collections" ["\"public\".\"Collection1\"", "\"public\".\"Collection2\""], (4)
"type": "incremental"} (5)
});
```

命令中的 id、type 和 data 参数的值对应于 信号集合的字段。

下表描述了示例中的参数：

表 5.2. MongoDB insert （）命令中的字段的描述，用于将增量快照信号发送到信号集合

项	值	描述
1	**db.debeziumSignal**	指定源数据库上信号集合的完全限定名称。
2	null	**_id** 参数指定作为信号请求的 **id** 标识符分配的任意字符串。 上例中的 insert 方法省略了可选的 **_id** 参数。由于文档没有明确为该参数分配值，因此 MongoDB 自动分配给文档的任意 id 将成为信号请求的 **id** 标识符。 使用此字符串识别信号集合中条目的日志记录消息。Debezium 不使用此标识符字符串。相反，Debebe 会在快照期间生成自己的 **id** 字符串作为水位线信号。
3	**execute-snapshot**	指定 **type** 参数指定信号要触发的操作。
4	**data-collections**	信号的 **data** 字段所需的组件，用于指定集合名称或正则表达式数组，以匹配快照中包含的集合名称。 数组列出了按照完全限定名称匹配集合的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号集合的名称相同。
5	**incremental**	信号的 **data** 字段的可选 **类型** 组件，用于指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 如果没有指定值，连接器将运行增量快照。

以下示例显示了连接器捕获的增量快照事件的 JSON。

示例：增加快照事件消息

```
{
   "before":null,
   "after": {
      "pk":"1",
      "value":"New data"
   },
   "source": {
      ...
      "snapshot":"incremental" 1
   },
   "op":"r", 2
   "ts_ms":"1620393591654",
   "transaction":null
}
```

项	字段名称	描述
1	**snapshot**	指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 在 SQL 查询中指定 **type** 值，您提交到信号集合是可选的。 如果没有指定值，连接器将运行增量快照。
2	**op**	指定事件类型。 快照事件的值是 **r**，表示 **READ** 操作。

5.2.5.2. 使用 Kafka 信号频道来触发增量快照

您可以向 配置的 Kafka 主题 发送消息，以请求连接器来运行临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 execute-snapshot，data 字段必须具有以下字段：

表 5.3. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与快照中包含的表的完全限定域名匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的列子集的条件。

execute-snapshot Kafka 消息示例：

```
Key = `test_connector`
```

```
Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"],
"type": "INCREMENTAL"}}`
```

带有额外条件的临时增量快照

Debezium 使用 additional-condition 字段来选择集合内容的子集。

通常，当 Debezium 运行快照时，它会运行 SQL 查询，例如：

*SELECT * FROM <tableName> ....*

当快照请求包含 additional-condition 时，extra-condition 会附加到 SQL 查询中，例如：

*SELECT * FROM <tableName> WHERE <additional-condition> ....*

例如，如果一个带有列的 id （主键）、颜色 和品牌 的产品 集合，如果您希望快照只包含 color='blue' 的内容，当您请求快照时，您可以附加一个 additional-condition 语句来过滤内容：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue'"}}`
```

您可以使用 additional-condition 语句根据多个列传递条件。例如，使用与上例中的相同 产品 集合，如果您希望快照只包含来自用于 color='blue'、和 brand='MyBrand' 的产品集合中的内容，您可以发送以下请求：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}`
```

5.2.5.3. 停止增量快照

您还可以通过向源数据库上的集合发送信号来停止增量快照。您可以通过将文档插入到信号集合中提交停止快照信号。在 Debezium 检测到信号集合中的更改后，它会读取信号，并在正在进行时停止增量快照操作。

您提交的查询指定 增量 的快照操作，以及要删除的当前运行快照的集合。

先决条件

- *启用了信号。*
  - *源数据库中存在信号数据收集。*
  - *信号数据收集在 signal.data.collection 属性中指定。*

使用源信号频道停止增量快照

1. *在信号集合中插入停止快照信号文档：*

```
<signalDataCollection>.insert({"id" : _<idNumber>,"type" : "stop-snapshot", "data" :
{"data-collections" ["<collectionName>", "<collectionName>"],"type":
"incremental"}});
```

例如，

```
db.debeziumSignal.insert({ 1
"type" : "stop-snapshot", 2 3
"data" : {
"data-collections" ["\"public\".\"Collection1\"", "\"public\".\"Collection2\""], 4
"type": "incremental"} 5
});
```

signal 命令中的 id、type 和 data 参数的值对应于 信号集合的字段。

下表描述了示例中的参数：

表 5.4. 插入命令中的字段描述，用于将停止增量快照文档发送到信号集合

项	值	描述
1	**db.debeziumSignal**	指定源数据库上信号集合的完全限定名称。

项	值	描述
2	null	上例中的 insert 方法省略了可选的 **_id** 参数。由于文档没有明确为该参数分配值，因此 MongoDB 自动分配给文档的任意 id 将成为信号请求的 **id** 标识符。 使用此字符串识别信号集合中条目的日志记录消息。Debezium 不使用此标识符字符串。
3	**stop-snapshot**	**type** 参数指定信号旨在触发的操作。
4	**data-collections**	信号的 **data** 字段的可选组件，用于指定集合名称或正则表达式数组，以匹配要从快照中删除的集合名称。 数组列出了按照完全限定名称匹配集合的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号集合的名称相同。如果省略了 **data** 字段的这一组件，信号将停止正在进行的整个增量快照。
5	**incremental**	信号的 **data** 字段所需的组件，用于指定要停止的快照操作类型。 目前，唯一有效的选项是 **增量的**。 如果没有指定 **类型** 值，信号将无法停止增量快照。

5.2.5.4. 使用 Kafka 信号频道停止增量快照

您可以将信号消息发送到 配置的 Kafka 信号主题，以停止临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 stop-snapshot，data 字段必须具有以下字段：

表 5.5. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	可选数组，以逗号分隔的正则表达式，与表的完全限定域名匹配，以包含在快照中。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

以下示例显示了典型的 stop-snapshot Kafka 信息：

```
Key = `test_connector`

Value = `{"type":"stop-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"],
"type": "INCREMENTAL"}}`
```

5.2.6. Debezium MongoDB 连接器流更改事件记录

在副本集的连接器任务记录偏移后，它使用偏移来确定应该开始流更改的 oplog 中的位置。然后，任务（取决于配置）连接到副本集的主节点，或连接到副本集范围更改流，并开始从该位置流更改。它处理所有创建、插入和删除操作，并将其转换为 Debezium 更改事件。每个更改事件都包含找到操作的 oplog 中的位置，连接器会定期将其记录为最新的偏移。记录偏移的时间间隔由 offset.flush.interval.ms 进行管理，它是一个 Kafka Connect worker 配置属性。

当连接器被安全停止时，处理最后一个偏移会被记录，以便在重启后，连接器将继续保持关闭的位置。如果连接器的任务意外终止，则任务可能会在最后一次记录偏移后处理和生成事件，但在记录最后一个偏移前；重启时，连接器从最后 记录的 偏移开始，可能会生成之前在崩溃前生成的一些相同事件。

注意

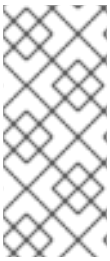
当 Kafka 管道中的所有组件都正常运行时，Kafka 用户会 准确接收每个消息一次。但是，当出现问题时，Kafka 只能保证消费者 至少接收每个消息一次。为避免意外结果，使用者必须能够处理重复的消息。

如前文所述，连接器任务始终使用副本集的主节点从 oplog 中流更改，确保连接器尽可能看到最新的操作，并可以捕获比使用第二个aries 更低的延迟的变化。当副本集选择新主节点时，连接器会立即停止流更改，连接到新主节点，并在同一位置开始从新主节点流更改。同样，如果连接器遇到与副本集成员通信的问题，它会尝试使用 exponential backoff 来重新连接，因此不会大量副本集，并在连接后继续从最后一个离开的位置继续流更改。这样，连接器可以动态地调整副本集成员资格中的更改，并自动处理通信失败。

总之，MongoDB 连接器在大多数情况下继续运行。通信问题可能会导致连接器等待问题解决。

5.2.7. MongoDB 支持填充 Debezium 更改事件中的 before 字段

在 MongoDB 6.0 及更高版本中，您可以配置更改流来发出文档的预镜像状态，以填充 MongoDB 更改事件的 before 字段。要在 MongoDB 中启用预镜像，您必须使用 db.createCollection(), create, 或 collMod 为集合设置 changeStreamPreAndPostImages。要启用 Debezium MongoDB 在更改事件中包含预镜像，请将连接器的 capture.mode 设置为其中一个 *_with_pre_image 选项。



MONGODB 更改流事件的大小限制

MongoDB 更改流事件的大小限制为 16MB。因此，使用预镜像会增加超过这个阈值的可能性，这可能会导致失败。有关如何避免超过更改流限制的详情，请参考 MongoDB 文档。

5.2.8. 接收 Debezium MongoDB 更改事件记录的默认 Kafka 主题名称

MongoDB 连接器将所有插入、更新和删除操作的事件写入每个集合中的文档到单个 Kafka 主题。Kafka 主题的名称始终使用 logicalName.databaseName.collectionName 格式，其中 logicalName 是连接器的逻辑名称（使用 topic.prefix 配置属性指定），databaseName 是创建发生在的数据库的名称，collectionName 是受影响的文档所在的 MongoDB 集合的名称。

例如，假设一个 MongoDB 副本集有一个 inventory 数据库，其中包含四个集合：products, products_on_hand, customers, 和 orders。如果监控这个数据库的连接器有一个逻辑名称 fulfillment，则这个连接器会在这四个 Kafka 主题上生成事件：

- *fulfillment.inventory.products*
- *fulfillment.inventory.products_on_hand*
- *fulfillment.inventory.customers*
- *fulfillment.inventory.orders*

请注意，主题名称不包含副本集名称或分片名称。因此，对分片集合（每个分片都包含集合文档的子集）的所有更改都会进入相同的 Kafka 主题。

您可以根据需要将 Kafka 设置为自动创建主题。如果没有，则必须在启动连接器前使用 Kafka 管理工具创建主题。

5.2.9. 事件密钥控制 Debezium MongoDB 连接器的主题分区

MongoDB 连接器不会明确确定如何为事件分区主题。相反，它允许 Kafka 如何根据事件密钥确定分区主题。您可以通过在 Kafka Connect worker 配置中定义分区 器 实现的名称来更改 Kafka 的分区逻辑。

Kafka 只为写入单个主题分区的事件维护总顺序。按键对事件进行分区意味着，具有相同键的所有事件始终都到达同一分区。这样可确保特定文档的所有事件始终被完全排序。

5.2.10. Debezium MongoDB 连接器生成的事件代表事务边界

Debezium 可以生成代表事务元数据边界的事件，并增强更改数据事件消息。

DEBEZIUM 接收事务元数据时的限制

Debezium 注册并只针对部署连接器后发生的事务接收元数据。部署连接器前发生的事务元数据不可用。

对于每个事务 BEGIN 和 END，Debezium 会生成一个包含以下字段的事件：

status

BEGIN 或 END

id

唯一事务标识符的字符串。

event_count （用于 END 事件）

事务发出的事件总数。

data_collections （用于 END 事件）

data_collection 和 event_count 的数组，它通过源自给定数据收集的更改来提供事件数量。

以下示例显示了一个典型的信息：

```
{
  "status": "BEGIN",
  "id": "1462833718356672513",
  "event_count": null,
  "data_collections": null
}

{
  "status": "END",
```

```
  "id": "1462833718356672513",
  "event_count": 2,
  "data_collections": [
    {
      "data_collection": "rs0.testDB.collectiona",
      "event_count": 1
    },
    {
      "data_collection": "rs0.testDB.collectionb",
      "event_count": 1
    }
  ]
}
```

除非通过 topic.transaction 选项覆盖，否则事务事件将写入名为 <topic. prefix>.transaction 的主题。

更改数据事件增强

如果启用了事务元数据，数据消息 Envelope 会增加一个新的 transaction 字段。此字段以字段复合的形式提供有关每个事件的信息：

id

唯一事务标识符的字符串。

total_order

事件在事务生成的所有事件中绝对位置。

data_collection_order

在事务发出的所有事件间，按数据收集位置。

下面是一个信息示例：

```
{
  "after": "{\"_id\" : {\"$numberLong\" : \"1004\"},\"first_name\" : \"Anne\",\"last_name\" :
\"Kretchmar\",\"email\" : \"annek@noanswer.org\"}",
  "source": {
...
  },
  "op": "c",
  "ts_ms": "1580390884335",
  "transaction": {
    "id": "1462833718356672513",
    "total_order": "1",
```

```
    "data_collection_order": "1"
  }
 }
```

5.3. DEBEZIUM MONGODB 连接器数据更改事件的描述

Debezium MongoDB 连接器为每个插入、更新或删除数据的文档级操作生成数据更改事件。每个事件包含一个键和值。密钥的结构和值取决于更改的集合。

Debezium 和 Kafka Connect 围绕 事件消息的持续流 设计。但是，这些事件的结构可能会随时间推移而改变，而用户很难处理这些事件。要解决这个问题，每个事件都包含其内容的 schema，或者如果您正在使用 schema registry，用户可以使用该模式 ID 从 registry 获取 schema。这使得每个事件都自包含。

以下框架 JSON 显示更改事件的基本四部分。但是，如何配置您选择在应用程序中使用的 Kafka Connect converter，决定更改事件中的这四个部分的表示。只有在将转换器配置为生成它时，schema 字段才会处于更改事件中。同样，只有在您配置转换器来生成它时，事件密钥和事件有效负载才会处于更改事件中。如果您使用 JSON 转换程序，并将其配置为生成所有四个基本更改事件部分，更改事件具有此结构：

```
{
 "schema": { 1
  ...
 },
 "payload": { 2
  ...
 },
 "schema": { 3
  ...
 },
 "payload": { 4
  ...
 },
}
```

表 5.6. 更改事件基本内容概述

项	字段名称	描述
1	**schema**	第一个 **schema** 字段是事件键的一部分。它指定一个 Kafka Connect 模式，用于描述事件键的 **payload** 部分的内容。换句话说，第一个 **schema** 字段描述了更改的文档的密钥结构。
2	**payload**	第一个 **payload** 字段是 event 键的一部分。它具有上一个**schema** 字段描述的结构，它包含已更改的文档的密钥。

项	字段名称	描述
3	**schema**	第二个 **schema** 字段是事件值的一部分。它指定 Kafka Connect 模式，用于描述事件值 **有效负载部分的内容**。换句话说，第二个 **模式** 描述了更改的文档的结构。通常，此模式包含嵌套模式。
4	**payload**	第二个 **payload** 字段是事件值的一部分。它具有上一个 **schema** 字段描述的结构，它包含已更改的文档的实际数据。

默认情况下，连接器流将事件记录改为名称与事件原始集合相同的主题。请参阅 主题名称。

警告

MongoDB 连接器确保所有 Kafka Connect 模式名称都遵循 Avro 模式名称格式。这意味着逻辑服务器名称必须以拉丁字母或下划线开头，即 a-z、A-Z 或 _。逻辑服务器名称和名称和集合名称中的每个字符都必须是一个拉丁字母、数字或下划线，即 a-z、A-Z、0-9 或 _。如果存在无效字符，它将使用下划线字符替换。

如果逻辑服务器名称、数据库名称或集合名称包含无效字符，且唯一与另一个名称区分名称的字符无效，这可能会导致意外冲突冲突，从而被下划线替换。

如需更多信息，请参阅以下主题：

- *第 5.3.1 节 “关于 Debezium MongoDB 中的键更改事件”*
- *第 5.3.2 节 “关于 Debezium MongoDB 更改事件中的值”*

5.3.1. 关于 Debezium MongoDB 中的键更改事件

更改事件的密钥包含更改的文档的密钥和更改的文档的实际密钥的 schema。对于给定的集合，schema 及其对应有效负载都包含一个 id 字段。此字段的值是文档的标识符，表示为来自 MongoDB 扩展 JSON 序列化严格模式 的字符串。

考虑一个连接器，其逻辑名称为 fulfillment，包括一个 inventory 数据库的副本集，以及包含如下文档的 customers 集合。

文档示例

```
{
  "_id": 1004,
  "first_name": "Anne",
  "last_name": "Kretchmar",
  "email": "annek@noanswer.org"
}
```

更改事件键示例

每次捕获 客户 集合更改的事件都有相同的事件关键模式。只要 customers 集合有以前的定义，捕获 customer 集合更改的每个更改事件都有以下关键结构：在 JSON 中，它类似如下：

```
{
  "schema": { 1
    "type": "struct",
    "name": "fulfillment.inventory.customers.Key", 2
    "optional": false, 3
    "fields": [ 4
      {
        "field": "id",
        "type": "string",
        "optional": false
      }
    ]
  },
  "payload": { 5
    "id": "1004"
  }
}
```

表 5.7. 更改事件键的描述

项	字段名称	描述
1	**schema**	键的 schema 部分指定一个 Kafka Connect 模式，它描述了键的 **payload** 部分的内容。

项	字段名称	描述
2	**fulfillment.inventory.customers.Key**	定义密钥有效负载结构的模式名称。这个模式描述了已更改的文档的密钥结构。键模式名称的格式是 *connector-name.database-name.collection-name*.**Key**。在本例中： ● **fulfillment** 是生成此事件的连接器的名称。 ● **inventory** 是包含已更改的集合的数据库。 ● **客户** 是包含已更新文档的集合。
3	**optional**	指明 event 键是否必须在其 **payload** 字段中包含一个值。在本例中，键有效负载中的值是必需的。当文档没有键时，键的 payload 字段中的值是可选的。
4	**fields**	指定 **有效负载中** 预期的每个字段，包括每个字段的名称、类型以及是否需要。
5	**payload**	包含生成此更改事件的文档的密钥。在本例中，键包含类型为 **字符串的 id** 字段，其值为 **1004**。

这个示例使用带有整数标识符的文档，但任何有效的 MongoDB 文档标识符的工作方式相同，包括文档标识符。对于文档标识符，事件键的 payload.id 值是字符串，它表示更新的文档的原始 _id 字段作为使用 strict 模式的 MongoDB 扩展 JSON 序列化。下表提供了如何表示不同类型的 _id 字段的示例。

表 5.8. 在事件键有效负载中代表文档 _id 字段的示例

类型	MongoDB _id Value	密钥的有效负载
整数	1234	**{ "id" : "1234" }**
浮点值	12.34	**{ "id" : "12.34" }**
字符串	"1234"	**{ "id" : "\"1234\"" }**
文档	**{ "hi" : "kafka", "nums" : [10.0, 100.0, 1000.0] }**	**{ "id" : "{\"hi\" : \"kafka\", \"nums\" : [10.0, 100.0, 1000.0]}" }**
ObjectId	**ObjectId("596e275826f08b2730779e1f")**	**{ "id" : "{\"$oid\" : \"596e275826f08b2730779e1f\"}" }**

类型	MongoDB _id Value	密钥的有效负载
二进制	BinData("a2Fma2E=",0)	{ "id" : "{\"$binary\" : \"a2Fma2E=\", \"$type\" : \"00\"}" }

5.3.2. 关于 Debezium MongoDB 更改事件中的值

更改事件中的值比键复杂一些。与键一样，该值有一个 schema 部分和 payload 部分。schema 部分包含描述 payload 部分的 Envelope 结构的 schema，包括其嵌套字段。为创建、更新或删除数据的操作更改事件，它们都有一个带有 envelope 结构的值有效负载。

考虑用于显示更改事件键示例的相同示例文档：

文档示例

```
{
  "_id": 1004,
  "first_name": "Anne",
  "last_name": "Kretchmar",
  "email": "annek@noanswer.org"
}
```

每个事件类型都描述了更改此文档的更改事件的值部分：

- 创建 事件
- 更新 事件
- 删除 事件
- tombstone 事件

创建 事件

以下示例显示了连接器为在 customers 集合中创建数据的操作生成的更改事件的值部分：

```
{
  "schema": { 1
    "type": "struct",
    "fields": [
      {
        "type": "string",
        "optional": true,
        "name": "io.debezium.data.Json", 2
        "version": 1,
        "field": "after"
      },
      {
        "type": "string",
        "optional": true,
        "name": "io.debezium.data.Json",
        "version": 1,
        "field": "patch"
      },
      {
        "type": "struct",
        "fields": [
          {
            "type": "string",
            "optional": false,
            "field": "version"
          },
          {
            "type": "string",
            "optional": false,
            "field": "connector"
          },
          {
            "type": "string",
            "optional": false,
            "field": "name"
          },
          {
            "type": "int64",
            "optional": false,
            "field": "ts_ms"
          },
          {
            "type": "boolean",
            "optional": true,
            "default": false,
            "field": "snapshot"
          },
          {
            "type": "string",
            "optional": false,
```

```
            "field": "db"
          },
          {
            "type": "string",
            "optional": false,
            "field": "rs"
          },
          {
            "type": "string",
            "optional": false,
            "field": "collection"
          },
          {
            "type": "int32",
            "optional": false,
            "field": "ord"
          },
          {
            "type": "int64",
            "optional": true,
            "field": "h"
          }
        ],
        "optional": false,
        "name": "io.debezium.connector.mongo.Source", 3
        "field": "source"
      },
      {
        "type": "string",
        "optional": true,
        "field": "op"
      },
      {
        "type": "int64",
        "optional": true,
        "field": "ts_ms"
      }
    ],
    "optional": false,
    "name": "dbserver1.inventory.customers.Envelope" 4
    },
  "payload": { 5
    "after": "{\"_id\" : {\"$numberLong\" : \"1004\"},\"first_name\" : \"Anne\",\"last_name\" :
\"Kretchmar\",\"email\" : \"annek@noanswer.org\"}", 6
    "source": { 7
      "version": "2.3.4.Final",
      "connector": "mongodb",
      "name": "fulfillment",
      "ts_ms": 1558965508000,
      "snapshot": false,
      "db": "inventory",
      "rs": "rs0",
      "collection": "customers",
      "ord": 31,
      "h": 1546547425148721999
```

```
    },
    "op": "c", 8
    "ts_ms": 1558965515240 9
  }
}
```

表 5.9. 创建 事件值字段的描述

项	字段名称	描述
1	**schema**	值的 schema，用于描述值有效负载的结构。当连接器为特定集合生成的每次更改事件中，更改事件的值模式都是相同的。
2	**name**	在 **schema** 部分中，每个 **name** 字段为值有效负载中的字段指定 schema。 **io.debezium.data.Json** 是、**patch** 和 **filter** 字段 **后** 有效负载的 schema。这个模式是针对 **customers** 集合的。*create* 事件是包含 **after** 字段的唯一事件类型。*update* 事件包含一个 **filter** 字段和一个 **patch** 字段。*delete* 事件包含一个 **过滤器** 字段，但不是 **after** 字段或 **patch** 字段。
3	**name**	**io.debezium.connector.mongo.Source** 是有效负载的 **source** 字段的 schema。这个模式特定于 MongoDB 连接器。连接器将其用于它生成的所有事件。
4	**name**	**dbserver1.inventory.customers.Envelope** 是负载总体结构的模式，其中 **dbserver1** 是连接器名称，**inventory** 是数据库，**customers** 是集合。这个模式特定于集合。
5	**payload**	值的实际数据。这是更改事件提供的信息。 可能会出现事件的 JSON 表示比它们描述的文档大得多。这是因为 JSON 表示必须包含消息的 schema 和 payload 部分。但是，通过使用 Avro converter，您可以显著减少连接器流到 Kafka 主题的信息大小。
6	**after**	指定事件发生后文档状态的可选字段。在本例中，post **字段包含** 新文档的 **_id**、**first_name**、**last_name** 和 **email** 字段的值。**after** 值始终是一个字符串。按照惯例，它包含文档的 JSON 表示。MongoDB oplog 条目仅包含 _create_ 事件的完整文档，当 **capture.mode** 选项被设置为 **change_streams_update_full** 时还包括 **update** 事件的文档; 换句话说，无论 **capture.mode** 选项是什么，*create* 事件是唯一包含 *after* 字段的事件类型。

项	字段名称	描述
7	**source**	描述事件源元数据的必需字段。此字段包含可用于将此事件与其他事件进行比较的信息，以及事件的来源、事件发生的顺序以及事件是否为同一事务的一部分。源元数据包括： ● Debezium 版本。 ● 生成事件的连接器的名称。 ● MongoDB 副本集的逻辑名称，它组成了生成事件的命名空间，并在连接器写入的 Kafka 主题名称中使用。 ● 包含新文档的集合和数据库名称。 ● 如果事件是快照的一部分。 ● 在数据库中进行更改时的时间戳，并在时间戳内发生事件。 ● MongoDB 操作的唯一标识符(oplog 事件中的 **h** 字段)。 ● MongoDB 会话 **lsid** 和事务号 **txnNumber** 的唯一标识符，以防更改是在事务中执行（仅更改流捕获模式）。
8	**op**	描述导致连接器生成事件的操作类型的强制字符串。在本例中，**c** 表示操作创建了文档。有效值为： ● **c** = create ● **u** = update ● **d** = delete ● **r** = 读取（仅适用于快照）
9	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

更改流捕获模式

示例 customers 集合中一个更新的改变事件的值有与那个集合的 create 事件相同的模式。同样，事件值有效负载具有相同的结构。但是，事件值有效负载在 update 事件中包含不同的值。只有在 capture.mode 选项被设置为 change_streams_update_full 时，update 事件才会包括一个 after 值。如果 capture.mode 选择被设置为 *_with_pre_image 选项之一，会提供一个 before 值。存在一个新的 structured 字段 updateDescription，本例中为几个额外的字段：

- ***updatedFields 是一个字符串字段，其中包含更新的文档字段的 JSON 表示及其值***

- *removedFields 是从文档中删除的字段名称列表*
- *truncatedArrays 是文档中的数组列表，被截断*

以下是连接器为 customer 集合中更新生成的更改事件值 的示例 ：

```
{
  "schema": { ... },
  "payload": {
    "op": "u", (1)
    "ts_ms": 1465491461815, (2)
    "before":"{\"_id\": {\"$numberLong\": \"1004\"},\"first_name\": \"unknown\",\"last_name\":
\"Kretchmar\",\"email\": \"annek@noanswer.org\"}", (3)
    "after":"{\"_id\": {\"$numberLong\": \"1004\"},\"first_name\": \"Anne Marie\",\"last_name\":
\"Kretchmar\",\"email\": \"annek@noanswer.org\"}", (4)
    "updateDescription": {
      "removedFields": null,
      "updatedFields": "{\"first_name\": \"Anne Marie\"}", (5)
      "truncatedArrays": null
    },
    "source": { (6)
      "version": "2.3.4.Final",
      "connector": "mongodb",
      "name": "fulfillment",
      "ts_ms": 1558965508000,
      "snapshot": false,
      "db": "inventory",
      "rs": "rs0",
      "collection": "customers",
      "ord": 1,
      "h": null,
      "tord": null,
      "stxnid": null,
      "lsid":"{\"id\": {\"$binary\": \"FA7YEzXgQXSX9OxmzlIH2w==\",\"$type\": \"04\"},\"uid\":
{\"$binary\": \"47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=\",\"$type\": \"00\"}}",
      "txnNumber":1
    }
  }
}
```

表 5.10. 更新 事件值字段的描述

项	字段名称	描述
1	**op**	描述导致连接器生成事件的操作类型的强制字符串。在本例中，**u** 表示操作更新了文档。

项	字段名称	描述
2	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
3	**before**	在更改前包含实际 MongoDB 文档的 JSON 字符串表示。如果捕获模式没有设置为 ***_with_preimage** 选项之一，则 *update* 事件值不包含 **before** 字段。
4	**after**	包含实际 MongoDB 文档的 JSON 字符串表示。 如果捕获模式没有设置为 **change_streams_update_full**，则 *update* 事件值不会包含 **after** 字段
5	**updatedFields**	包含文档更新字段值的 JSON 字符串表示。在本例中，更新将 **first_name** 字段改为新值。
6	**source**	描述事件源元数据的必需字段。此字段包含与同一集合的 *create* 事件相同的信息，但它们的值不同，因为此事件来自 oplog 中的不同位置。源元数据包括： ● Debezium 版本。 ● 生成事件的连接器的名称。 ● MongoDB 副本集的逻辑名称，它组成了生成事件的命名空间，并在连接器写入的 Kafka 主题名称中使用。 ● 包含更新文档的集合和数据库名称。 ● 如果事件是快照的一部分。 ● 在数据库中进行更改时的时间戳，并在时间戳内发生事件。 ● MongoDB 会话 **lsid** 和事务号 **txnNumber** 的唯一标识符，以防更改是在事务中执行的。

警告

事件中的 after 值应作为文档的 at-point-of-time 值进行处理。该值不会动态计算，但是从集合中获取的。因此，如果多个更新一个紧随另一个发生，则所有 update 事件都会包含在文档中存储的代表最后的值相同的 after 值。

如果您的应用程序依赖于逐步更改演进，则应该只依赖 updateDescription。

删除 事件

delete 更改事件中的值与为同一集合的 create 和 update 事件相同的 schema 部分。delete 事件中的 payload 部分包含与为同一集合的 create 和 update 事件不同的值。特别是，delete 事件不包含 after 值和 updateDescription 值。以下是 customers 集合中文档的 delete 事件示例：

```
{
  "schema": { ... },
  "payload": {
    "op": "d", (1)
    "ts_ms": 1465495462115, (2)
    "before":"{\"_id\": {\"$numberLong\": \"1004\"},\"first_name\": \"Anne
Marie\",\"last_name\": \"Kretchmar\",\"email\": \"annek@noanswer.org\"}", (3)
    "source": { (4)
      "version": "2.3.4.Final",
      "connector": "mongodb",
      "name": "fulfillment",
      "ts_ms": 1558965508000,
      "snapshot": true,
      "db": "inventory",
      "rs": "rs0",
      "collection": "customers",
      "ord": 6,
      "h": 1546547425148721999
    }
  }
}
```

表 5.11. 删除 事件值字段的描述

项	字段名称	描述
1	**op**	描述操作类型的强制字符串。**op** 字段值为 **d**，表示此文档已被删除。

项	字段名称	描述
2	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
3	**before**	在更改前包含实际 MongoDB 文档的 JSON 字符串表示。如果捕获模式没有设置为 ***_with_preimage** 选项之一，则 *update* 事件值不包含 **before** 字段。
4	**source**	描述事件源元数据的必需字段。此字段包含与同一集合的 *create* 或 *update* 事件相同的信息，但它们的值不同，因为此事件来自 oplog 中的不同位置。源元数据包括： ● Debezium 版本。 ● 生成事件的连接器的名称。 ● MongoDB 副本集的逻辑名称，它组成了生成事件的命名空间，并在连接器写入的 Kafka 主题名称中使用。 ● 包含已删除文档的集合和数据库名称。 ● 如果事件是快照的一部分。 ● 在数据库中进行更改时的时间戳，并在时间戳内发生事件。 ● MongoDB 操作的唯一标识符(oplog 事件中的 **h** 字段)。 ● MongoDB 会话 **lsid** 和事务号 **txnNumber** 的唯一标识符，以防更改是在事务中执行（仅更改流捕获模式）。

MongoDB 连接器事件被设计为使用 Kafka 日志压缩。只要保留每个密钥的最新消息，日志压缩就会启用删除一些旧的消息。这可让 Kafka 回收存储空间，同时确保主题包含完整的数据集，并可用于重新载入基于密钥的状态。

tombstone 事件

唯一标识的文档的所有 MongoDB 连接器事件都有完全相同的密钥。删除文档时，delete 事件值仍可用于日志压缩，因为 Kafka 您可以删除具有相同键的所有之前信息。但是，要让 Kafka 删除具有该键的所有消息，消息值必须为 null。为了实现此目的，在 Debezium 的 MongoDB 连接器发出一个 delete 事件后，连接器会发出一个特殊的 tombstone 事件，它具有相同的键但有一个 null 值。tombstone 事件告知 Kafka，可以删除具有相同键的所有消息。

5.4. 设置 MONGODB 以使用 DEBEZIUM 连接器

MongoDB 连接器使用 MongoDB 的更改流来捕获更改，因此连接器只适用于 MongoDB 副本集，或者每个分片都是一个单独的副本集的分片集群。有关设置 副本集或 分片集群，请参阅 MongoDB 文档。另外，请务必了解如何使用副本集启用 访问控制和身份验证。

您还必须有一个 MongoDB 用户，该用户具有适当的角色才能读取 oplog 的 admin 数据库。此外，用户还必须能够在分片集群的配置服务器中读取配置数据库，并且必须具有 listDatabases 特权操作。当使用更改流（默认）时，用户还必须具有集群范围的特权操作 查找和 changeStream。

当您打算使用 pre-image 并填充 before 字段时，您需要首先为一个集合启用 changeStreamPreAndPostImages，使用 db.createCollection(), create, 或 collMod。

5.5. 部署 DEBEZIUM MONGODB 连接器

您可以使用以下任一方法部署 Debezium MongoDB 连接器：

- 使用 AMQ Streams 自动创建包含连接器插件的镜像。

  这是首选的方法。

- 从 Dockerfile 构建自定义 Kafka Connect 容器镜像。

其他资源

- 第 5.5.5 节 “Debezium MongoDB 连接器配置属性的描述”

5.5.1. 使用 AMQ Streams 部署 MongoDB 连接器

从 Debezium 1.7 开始，部署 Debezium 连接器的首选方法是使用 AMQ Streams 构建包含连接器插件的 Kafka Connect 容器镜像。

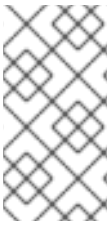
在部署过程中，您可以创建并使用以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR，并包含有关镜像中需要包含连接器工件的信息。

- KafkaConnector CR，提供包括连接器用来访问源数据库的信息。在 AMQ Streams 启动 Kafka Connect pod 后，您可以通过应用 KafkaConnector CR 来启动连接器。

在 Kafka Connect 镜像的构建规格中，您可以指定可用于部署的连接器。对于每个连接器插件，您还可以指定您的部署可以使用的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。当 AMQ Streams 构建 Kafka Connect 镜像时，它会下载指定的工件，并将其合并到镜像中。

KafkaConnect CR 中的 spec.build.output 参数指定存储生成的 Kafka Connect 容器镜像的位置。容器镜像可以存储在 Docker registry 中，也可以存储在 OpenShift ImageStream 中。要将镜像存储在 ImageStream 中，您必须在部署 Kafka Connect 前创建 ImageStream。镜像流不会被自动创建。



注意

如果使用 KafkaConnect 资源来创建集群，之后无法使用 Kafka Connect REST API 创建或更新连接器。您仍然可以使用 REST API 来检索信息。

其他资源

- 在 OpenShift 中使用 AMQ Streams 配置 Kafka 连接。
- 在 OpenShift 中部署和管理 AMQ Streams 中，使用 AMQ Streams 自动创建新容器镜像。

5.5.2. 使用 AMQ Streams 部署 Debezium MongoDB 连接器

使用早期版本的 AMQ Streams 时，要在 OpenShift 上部署 Debezium 连接器，您需要首先为连接器构建 Kafka Connect 镜像。在 OpenShift 上部署连接器的当前首选方法是使用 AMQ Streams 中的构建配置来构建 Kafka Connect 容器镜像，其中包含您要使用的 Debezium 连接器插件。

在构建过程中，AMQ Streams Operator 将 KafkaConnect 自定义资源（包括 Debezium 连接器定义）中的输入参数转换为 Kafka Connect 容器镜像。构建会从 Red Hat Maven 存储库或其他配置的 HTTP 服务器下载必要的工件。

新创建的容器被推送到在 .spec.build.output 中指定的容器 registry，用于部署 Kafka Connect 集群。在 AMQ Streams 构建 Kafka Connect 镜像后，您可以创建 KafkaConnector 自定义资源来启动构建中包含的连接器。

先决条件

- 您可以访问安装了集群 Operator 的 OpenShift 集群。
- AMQ Streams Operator 正在运行。
- 在 OpenShift 中部署和升级 AMQ Streams 所述，会部署 Apache Kafka 集群。
- Kafka Connect 在 AMQ Streams 上部署
- 您有一个 Red Hat Integration 许可证。
- 已安装 OpenShift oc CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。
- 根据您要存储 Kafka Connect 构建镜像的方式，您需要 registry 权限，或者您必须创建 ImageStream 资源：

  将构建镜像存储在镜像 registry 中，如 Red Hat Quay.io 或 Docker Hub

  - 在 registry 中创建和管理镜像的帐户和权限。

  将构建镜像存储为原生 OpenShift ImageStream

  - ImageStream 资源已部署到集群中，以存储新的容器镜像。您必须为集群显式创建 ImageStream。默认无法使用镜像流。如需有关 ImageStreams 的更多信息，请参阅在 OpenShift Container Platform 中管理镜像流。

流程

1. 登录 OpenShift 集群。
2. 为连接器创建 Debezium KafkaConnect 自定义资源(CR)，或修改现有的资源。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 metadata.annotations 和 spec.build 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了

KafkaConnect 自定义资源。

例 5.1. 定义包含 Debezium 连接器的 KafkaConnect 自定义资源的 dbz-connect.yaml 文件

在以下示例中，自定义资源被配置为下载以下工件：

- *Debezium MongoDB 连接器存档。*
- *Service Registry 归档。Service Registry 是一个可选组件。只有在打算将 Avro 序列化与连接器搭配使用时，才添加 Service Registry 组件。*
- *Debezium 脚本 SMT 归档以及您要与 Debezium 连接器一起使用的关联脚本引擎。SMT 归档和脚本语言依赖项是可选组件。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时，才添加这些组件。*

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: debezium-kafka-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  version: 3.5.0
  build: 2
    output: 3
      type: imagestream 4
      image: debezium-streams-connect:latest
    plugins: 5
      - name: debezium-connector-mongodb
        artifacts:
          - type: zip 6
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-mongodb/2.3.4.Final-redhat-00001/debezium-connector-mongodb-2.3.4.Final-redhat-00001-plugin.zip 7
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-distro-connect-converter/2.4.4.Final-redhat-<build-number>/apicurio-registry-distro-connect-converter-2.4.4.Final-redhat-<build-number>.zip 8
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-scripting/2.3.4.Final-redhat-00001/debezium-scripting-2.3.4.Final-redhat-00001.zip
9
          - type: jar
            url:
https://repo1.maven.org/maven2/org/codehaus/groovy/groovy/3.0.11/groovy-
```

```
3.0.11.jar (10)
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-jsr223/3.0.11/groovy-jsr223-3.0.11.jar
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-json3.0.11/groovy-json-3.0.11.jar

    bootstrapServers: debezium-kafka-cluster-kafka-bootstrap:9093

    ...
```

表 5.12. Kafka Connect 配置设置的描述

项	描述
1	将 **strimzi.io/use-connector-resources** 注解设置为 **"true"**，使 Cluster Operator 使用 **KafkaConnector** 资源在此 Kafka Connect 集群中配置连接器。
2	**spec.build** 配置指定在镜像中存储构建镜像的位置，并列出要在镜像中包含的插件，以及插件工件的位置。
3	**build.output** 指定存储新构建镜像的 registry。
4	指定镜像输出的名称和镜像名称。**output.type** 的有效值是 **要推送到** 容器 registry（如 Docker Hub 或 Quay）或 **镜像流** 的有效值，以将镜像推送到内部 OpenShift ImageStream。要使用 ImageStream，必须将 ImageStream 资源部署到集群中。有关在 KafkaConnect 配置中指定 **build.output** 的更多信息，请参阅在 OpenShift 中配置 AMQ Streams 中的 AMQ Streams Build schema 参考。
5	**plugins** 配置列出了您要包含在 Kafka Connect 镜像中的所有连接器。对于列表中的每个条目，指定一个插件**名称**，以及有关构建连接器所需的工件的信息。另外，对于每个连接器插件，您还可以包含可用于连接器的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。
6	**artifacts.type** 的值指定在 **artifacts.url** 中指定的工件类型。有效类型为 **zip**、**tgz** 或 **jar**。Debezium 连接器存档以 **.zip** 文件格式提供。**类型** 值必须与 **url** 字段中引用的文件类型匹配。
7	**artifacts.url** 的值指定 HTTP 服务器的地址，如 Maven 存储库，用于存储连接器工件的文件。Debezium 连接器工件在 Red Hat Maven 存储库中提供。OpenShift 集群必须有权访问指定的服务器。
8	（可选）指定用于下载 Service Registry 组件的工件 **类型和 url**。包含 Service Registry 工件，只有在您希望连接器使用 Apache Avro 来序列化带有 Service Registry 的事件键和值时，而不是使用默认的 JSON 转换程序。
9	（可选）指定 Debezium 脚本 SMT 归档的工件 **类型和 url**，以用于 Debezium 连接器。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时才包括脚本 SMT。要使用脚本 SMT，您必须部署 JSR 223 兼容脚本实现，如 groovy。

项	描述
10	（可选）指定 JSR 223 兼容脚本实施的 JAR 文件的工件 **类型和 url**，这是 Debezium 脚本 SMT 所需的。  **重要** 如果使用 AMQ Streams 将连接器插件合并到 Kafka Connect 镜像中，每个所需的脚本语言 **工件。url** 必须指定 JAR 文件的位置，并且 **artifacts.type** 的值也必须设置为 **jar**。无效的值会导致连接器在运行时失败。 要启用带有脚本 SMT 的 Apache Groovy 语言，示例中的自定义资源会为以下库检索 JAR 文件： • **groovy** • **Groovy-jsr223**（指定代理） • **groovy-json**（解析 JSON 字符串的模块） 作为替代方案，Debebe Debezium 脚本 SMT 也支持使用 JSR 223 实现 GraalVM JavaScript。

3. *输入以下命令将 KafkaConnect 构建规格应用到 OpenShift 集群：*

```
oc create -f dbz-connect.yaml
```

根据自定义资源中指定的配置，Streams Operator 准备要部署的 Kafka Connect 镜像。构建完成后，Operator 将镜像推送到指定的 registry 或 ImageStream，并启动 Kafka Connect 集群。集群中提供了您在配置中列出的连接器工件。

4. *创建一个 KafkaConnector 资源来定义您要部署的每个连接器的实例。*
例如，创建以下 KafkaConnector CR，并将它保存为 mongodb-inventory-connector.yaml

例 5.2. mongodb-inventory-connector.yaml 文件，该文件为 Debezium 连接器定义 KafkaConnector 自定义资源

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: debezium-kafka-connect-cluster
  name: inventory-connector-mongodb (1)
spec:
  class: io.debezium.connector.mongodb.MongoDbConnector (2)
```

```
    tasksMax: 1 3
    config: 4
      mongodb.hosts: rs0/192.168.99.100:27017 5
      mongodb.user: debezium 6
      mongodb.password: dbz 7
      topic.prefix: inventory-connector-mongodb 8
      collection.include.list: inventory[.]* 9
```

表 5.13. 连接器配置设置的描述

项	描述
1	使用 Kafka Connect 集群注册的连接器名称。
2	连接器类的名称。
3	可以同时操作的任务数量。
4	连接器的配置。
5	主机数据库实例的地址和端口号。
7	Debezium 用于连接到数据库的帐户名称。
8	Debezium 用于连接到数据库用户帐户的密码。
8	数据库实例或集群的主题前缀。 指定的名称只能由字母数字字符或下划线组成。 因为主题前缀被用作从这个连接器接收更改事件的任何 Kafka 主题的前缀，所以该名称在集群中的连接器之间必须是唯一的。 如果连接器与 Avro 连接器集成，则此命名空间也用于相关 Kafka Connect 模式的名称，以及相应 Avro 模式的命名空间。
9	连接器捕获更改的集合名称。

5. *运行以下命令来创建连接器资源：*

```
oc create -n <namespace> -f <kafkaConnector>.yaml
```

例如，

```
oc create -n debezium -f {context}-inventory-connector.yaml
```

连接器注册到 Kafka Connect 集群，并开始针对 KafkaConnector CR 中的

spec.config.database.dbname 指定的数据库运行。连接器 pod 就绪后，Debebe 正在运行。

现在，您可以验证 Debezium MongoDB 部署。

5.5.3. 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium MongoDB 连接器

要部署 Debezium MongoDB 连接器，您必须构建包含 Debezium 连接器归档的自定义 Kafka Connect 容器镜像，然后将此容器镜像推送到容器 registry。然后，您创建两个自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR。CR 中的 image 属性指定您创建的容器镜像的名称，以运行 Debezium 连接器。您可以将此 CR 应用到部署 Red Hat AMQ Streams 的 OpenShift 实例。AMQ Streams 提供将 Apache Kafka 带到 OpenShift 的 operator 和镜像。
- 定义 Debezium MongoDB 连接器的 KafkaConnector CR。将此 CR 应用到应用 KafkaConnect CR 的同一 OpenShift 实例。

先决条件

- MongoDB 正在运行，您完成了 设置 MongoDB 的步骤，以便使用 Debezium 连接器。
- AMQ Streams 部署在 OpenShift 中，并运行 Apache Kafka 和 Kafka Connect。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams。
- podman 或 Docker 已安装。
- 您有一个在容器 registry 中创建和管理容器（如 quay.io 或 docker.io）的帐户和权限，您要添加将运行 Debezium 连接器的容器。

流程

1. 为 Kafka Connect 创建 Debezium MongoDB 容器：
   a. 创建一个使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 的 Dockerfile 作为基础镜像。例如，在终端窗口中输入以下命令：

```
cat <<EOF >debezium-container-for-mongodb.yaml (1)
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/debezium (2)
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-mongodb/2.3.4.Final-redhat-00001/debezium-connector-mongodb-2.3.4.Final-redhat-00001-plugin.zip \
&& unzip debezium-connector-mongodb-2.3.4.Final-redhat-00001-plugin.zip \
&& rm debezium-connector-mongodb-2.3.4.Final-redhat-00001-plugin.zip
RUN cd /opt/kafka/plugins/debezium/
USER 1001
EOF
```

项	描述
1	您可以指定您想要的任何文件名。
2	指定 Kafka Connect 插件目录的路径。如果您的 Kafka Connect 插件目录位于不同的位置，请将此路径替换为目录的实际路径。

该命令在当前目录中创建一个名为 debezium-container-for-mongodb.yaml 的 Dockerfile。

b.

从您在上一步中创建的 debezium-container-for-mongodb.yaml Docker 文件中构建容器镜像。在包含文件的目录中，打开终端窗口并输入以下命令之一：

```
podman build -t debezium-container-for-mongodb:latest .
```

```
docker build -t debezium-container-for-mongodb:latest .
```

前面的命令使用名称 debezium-container-for-mongodb 构建容器镜像。

c.

将自定义镜像推送到容器 registry，如 quay.io 或内部容器 registry。容器 registry 必须可供您要部署镜像的 OpenShift 实例使用。输入以下命令之一：

```
podman push <myregistry.io>/debezium-container-for-mongodb:latest
```

```
docker push <myregistry.io>/debezium-container-for-mongodb:latest
```

d.

创建新的 Debezium MongoDB KafkaConnect 自定义资源(CR)。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 注解和 镜像 属性。以下示例显示了一

个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" (1)
spec:
  #...
  image: debezium-container-for-mongodb (2)

  ...
```

项	描述
1	**metadata.annotations** 表示 **KafkaConnector** 资源用于配置在这个 Kafka Connect 集群中使用的 Cluster Operator。
2	**spec.image** 指定您创建的镜像的名称，以运行 Debezium 连接器。此属性覆盖 Cluster Operator 中的 **STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE** 变量。

e. *输入以下命令将 KafkaConnect CR 应用到 OpenShift Kafka Connect 环境：*

```
oc create -f dbz-connect.yaml
```

该命令添加了一个 Kafka Connect 实例，用于指定您为运行 Debezium 连接器而创建的镜像的名称。

2. *创建一个 KafkaConnector 自定义资源来配置 Debezium MongoDB 连接器实例。*

您可以在 .yaml 文件中配置 Debezium MongoDB 连接器，该文件指定连接器的配置属性。连接器配置可能指示 Debezium 为 MongoDB 副本集或分片集群的子集生成更改事件。另外，您可以设置过滤不需要的集合的属性。

以下示例配置了一个 Debezium 连接器，它在 192.168.99.100 上的端口 27017 连接到 MongoDB 副本集 rs0，并捕获 清单 集合中发生的更改。inventory-connector-mongodb 是副本集的逻辑名称。

MongoDB inventory-connector.yaml

```
apiVersion: kafka.strimzi.io/v1beta2
 kind: KafkaConnector
 metadata:
  name: inventory-connector-mongodb (1)
  labels: strimzi.io/cluster: my-connect-cluster
 spec:
  class: io.debezium.connector.mongodb.MongoDbConnector (2)
  config:
   mongodb.connection.string: mongodb://192.168.99.100:27017/?replicaSet=rs0 (3)
   topic.prefix: inventory-connector-mongodb (4)
   collection.include.list: inventory[.]* (5)
```

(1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)

用于使用 Kafka Connect 注册连接器的名称。

(2) (2) (2) (2) (2) (2) (2) (2) (2) (2) (2) (2)

MongoDB 连接器类的名称。

(3) (3) (3) (3) (3) (3) (3) (3) (3) (3)

用于连接到 MongoDB 副本集的主机地址。

(4) (4) (4) (4) (4) (4) (4) (4) (4) (4)

MongoDB 副本集的逻辑名称，它组成了生成事件的命名空间，并在使用 Avro converter 时，用来写入的 Kafka 主题、Kafka Connect 模式名称和相应 Avro 模式的命名空间中使用。

(5) (5) (5) (5) (5) (5) (5) (5)

与要监控的所有集合的命名空间（如 <dbName>.<collectionName>）匹配的正则表达式列表。

3. 使用 Kafka Connect 创建连接器实例。例如，如果您将 KafkaConnector 资源保存在 inventory-connector.yaml 文件中，您将运行以下命令：

```
oc apply -f inventory-connector.yaml
```

前面的命令注册 inventory-connector，连接器开始针对 KafkaConnector CR 中定义的 清单 集合运行。

有关您可以为 Debezium MongoDB 连接器设置的配置属性的完整列表，请参阅 MongoDB 连接器配置属性。

结果

连接器启动后，它会完成以下操作：

- 在 MongoDB 副本集中执行集合的一致性快照。
- 读取副本集的更改流。
- 为每个插入、更新和删除文档生成更改事件。
- Streams 将事件记录改为 Kafka 主题。

5.5.4. 验证 Debezium MongoDB 连接器是否正在运行

如果连接器正确启动且没有错误，它会为每个连接器配置为捕获的表创建一个主题。下游应用程序可以订阅这些主题，以检索源数据库中发生的信息事件。

要验证连接器是否正在运行，您可以从 OpenShift Container Platform Web 控制台或 OpenShift CLI 工具(oc)执行以下操作：

- 验证连接器状态。
- 验证连接器是否生成主题。
- 验证主题是否填充了读取操作("op":"r")的事件，连接器在每个表的初始快照中生成。

先决条件

- *Debezium 连接器部署到 OpenShift 上的 AMQ Streams。*
- *已安装 OpenShift oc CLI 客户端。*
- *访问 OpenShift Container Platform web 控制台。*

流程

1. *使用以下方法之一检查 KafkaConnector 资源的状态：*
   - *在 OpenShift Container Platform Web 控制台中：*
     a. *导航到 Home → Search。*
     b. *在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaConnector。*
     c. *在 KafkaConnectors 列表中，点您要检查的连接器的名称，如 inventory-connector-mongodb。*
     d. *在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。*
   - *在终端窗口中：*
     a. *使用以下命令：*

```
oc describe KafkaConnector <connector-name> -n <project>
```

例如，

```
oc describe KafkaConnector inventory-connector-mongodb -n debezium
```

该命令返回类似以下示例的状态信息：

例 5.3. KafkaConnector 资源状态

```
Name:         inventory-connector-mongodb
Namespace:    debezium
Labels:       strimzi.io/cluster=debezium-kafka-connect-cluster
Annotations:  <none>
API Version:  kafka.strimzi.io/v1beta2
Kind:         KafkaConnector

...

Status:
  Conditions:
    Last Transition Time:  2021-12-08T17:41:34.897153Z
    Status:                True
    Type:                  Ready
  Connector Status:
    Connector:
      State:      RUNNING
      worker_id:  10.131.1.124:8083
    Name:         inventory-connector-mongodb
    Tasks:
      Id:           0
      State:        RUNNING
      worker_id:    10.131.1.124:8083
    Type:           source
  Observed Generation:  1
  Tasks Max:            1
  Topics:
    inventory-connector-mongodb.inventory
    inventory-connector-mongodb.inventory.addresses
    inventory-connector-mongodb.inventory.customers
    inventory-connector-mongodb.inventory.geom
    inventory-connector-mongodb.inventory.orders
    inventory-connector-mongodb.inventory.products
    inventory-connector-mongodb.inventory.products_on_hand
Events:  <none>
```

2. 验证连接器是否创建了 Kafka 主题：

- 通过 OpenShift Container Platform Web 控制台。

a. *导航到 Home → Search。*

b. *在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaTopic。*

c. *在 KafkaTopics 列表中，点您要检查的主题名称，例如 inventory-connector-mongodb.inventory.orders---ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d。*

d. *在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。*

- *在终端窗口中：*

a. *使用以下命令：*

```
oc get kafkatopics
```

该命令返回类似以下示例的状态信息：

例 5.4. KafkaTopic 资源状态

```
NAME                                                CLUSTER
PARTITIONS   REPLICATION FACTOR   READY
connect-cluster-configs                              debezium-kafka-cluster   1
1                 True
connect-cluster-offsets                              debezium-kafka-cluster   25
1                 True
connect-cluster-status                               debezium-kafka-cluster   5
1                 True
consumer-offsets---84e7a678d08f4bd226872e5cdd4eb527fadc1c6a
debezium-kafka-cluster   50          1                 True
inventory-connector-mongodb--a96f69b23d6118ff415f772679da623fbbb99421
debezium-kafka-cluster   1           1                 True
inventory-connector-mongodb.inventory.addresses---
1b6beaf7b2eb57d177d92be90ca2b210c9a56480          debezium-kafka-cluster
1           1                 True
inventory-connector-mongodb.inventory.customers---
9931e04ec92ecc0924f4406af3fdace7545c483b          debezium-kafka-cluster   1
1                 True
inventory-connector-mongodb.inventory.geom---
9f7e136091f071bf49ca59bf99e86c713ee58dd5            debezium-kafka-cluster
1           1                 True
```

```
inventory-connector-mongodb.inventory.orders---
ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d          debezium-kafka-cluster
1          1          True
inventory-connector-mongodb.inventory.products---
df0746db116844cee2297fab611c21b56f82dcef          debezium-kafka-cluster   1
1          True
inventory-connector-mongodb.inventory.products_on_hand---
8649e0f17ffcc9212e266e31a7aeea4585e5c6b5   debezium-kafka-cluster   1
1          True
schema-changes.inventory                          debezium-kafka-cluster
1          1          True
strimzi-store-topic---effb8e3e057afce1ecf67c3f5d8e4e3ff177fc55          debezium-
kafka-cluster   1          1          True
strimzi-topic-operator-kstreams-topic-store-changelog---
b75e702040b99be8a9263134de3507fc0cc4017b  debezium-kafka-cluster  1   1
True
```

3. *检查主题内容。*

- *在终端窗口中输入以下命令：*

```
oc exec -n <project>  -it <kafka-cluster> -- /opt/kafka/bin/kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=<topic-name>
```

例如，

```
oc exec -n debezium  -it debezium-kafka-cluster-kafka-0 -- /opt/kafka/bin/kafka-
console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=inventory-connector-mongodb.inventory.products_on_hand
```

指定主题名称的格式与 oc describe 命令返回的格式与第 1 步中返回，例如 inventory-connector-mongodb.inventory.addresses。

对于主题中的每个事件，命令会返回类似以下示例的信息：

例 5.5. Debezium 更改事件的内容

```
{"schema":{"type":"struct","fields":
```

[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"inventory-connector-mongodb.inventory.products_on_hand.Key"},"payload":{"product_id":101}}
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-connector-mongodb.inventory.products_on_hand.Value","field":"before"},
{"type":"struct","fields":[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-connector-mongodb.inventory.products_on_hand.Value","field":"after"},
{"type":"struct","fields":[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":
{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connector.mongodb.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transaction"}],"optional":false,"name":"inventory-connector-mongodb.inventory.products_on_hand.Envelope"},"payload":{"before":null,"after":
{"product_id":101,"quantity":3},"source":{"version":"2.3.4.Final-redhat-00001","connector":"mongodb","name":"inventory-connector-mongodb","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"table":"products_on_hand","server_id":0,"gtid":null,"file":"mongodb-bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":1638985247805,"transaction":null}}

*在前面的示例中，有效负载 值显示连接器快照从表 **inventory.products_on_hand** 生成读取 **(op" ="r")事件。product_id** 记录的 **"before" 状态为 null，表示该记录不存在之前的值。"after" 状态对于 product_id 为 101 的项目的 quantity** 显示为 **3。***

5.5.5. Debezium MongoDB 连接器配置属性的描述

***Debezium MongoDB** 连接器具有大量配置属性，可用于实现应用程序的正确连接器行为。许多属性都有默认值。有关属性的信息组织如下：*

- *所需的 Debezium MongoDB 连接器配置属性*

- *高级 Debezium MongoDB 连接器配置属性*

除非默认值可用， 否则需要以下配置属性。

表 5.14. 所需的 Debezium MongoDB 连接器配置属性

属性	默认	描述
name	没有默认值	连接器的唯一名称。尝试使用相同的名称再次注册将失败。（所有 Kafka Connect 连接器都需要此属性。）
connector.class	没有默认值	连接器的 Java 类的名称。始终为 MongoDB 连接器使用 **io.debezium.connector.mongodb.MongoDbConnector** 的值。
mongodb.connection.string	没有默认值	指定连接器用来连接到 MongoDB 副本集 的连接字符串。此属性替换了之前在 MongoDB 连接器版本中提供的 **mongodb.hosts** 属性。 **注意** 捕获分片 MongoDB 集群更改的连接器仅在 **mongodb.connection.mode** 设置为 **replica_set** 时在初始分片发现过程中使用此连接字符串。在初始发现过程后，会为每个分片生成连接字符串。

属性	默认	描述
mongodb.connection.mode	replica_set	指定连接器连接到 **分片** MongoDB 集群时所使用的策略。将此属性设置为以下值之一： **replica_set** 连接器为每个分片建立到副本集的独立连接。 **分片** 连接器根据 mongodb-connection-string 的值建立与数据库的单一连接。+  **注意** **replica_set** 选项允许连接器在多个连接器任务之间分发分片处理。但是，在这个配置中，连接器会在连接到单个分片时绕过 MongoDB 路由器，而 MongoDB 不建议这样做。  **警告** 在连接模式间切换无效偏移时，这会触发新快照。
topic.prefix	没有默认值	标识此连接器监控的连接器和/或 MongoDB 副本集或分片集群的唯一名称。每台服务器应由大多数 Debezium 连接器监控，因为此服务器名称会添加所有持久的 Kafka 主题，从 MongoDB 副本集或集群中重复。仅使用字母数字字符、连字符、句点和下划线来组成名称。逻辑名称在所有其他连接器之间应是唯一的，因为该名称在命名此连接器的 Kafka 主题中用作前缀。  **警告** 不要更改此属性的值。如果您重启后更改了 name 值，而不是继续向原始主题发出事件，连接器会将后续事件发送到名称基于新值的主题。

属性	默认	描述
mongodb.user	没有默认值	连接到 MongoDB 时使用的数据库用户的名称。只有在 MongoDB 被配置为使用身份验证时才需要。
mongodb.password	没有默认值	连接到 MongoDB 时使用的密码。只有在 MongoDB 被配置为使用身份验证时才需要。
mongodb.authsource	**admin**	包含 MongoDB 凭证的数据库（身份验证源）。只有在 MongoDB 配置为将 MongoDB 与另一个身份验证数据库而不是 **admin** 进行身份验证时才需要。
mongodb.ssl.enabled	**false**	连接器将使用 SSL 连接到 MongoDB 实例。
mongodb.ssl.invalid.hostname.allowed	**false**	启用 SSL 时，此设置控制连接阶段是否禁用了严格的主机名检查。如果为 **true**，连接不会阻止中间人攻击。
database.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与要监控的数据库名称匹配。默认情况下，会监控所有数据库。 当设置 **database.include.list** 时，连接器只监控属性指定的数据库。其他数据库不包括在监控中。 要匹配数据库的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据库的整个名称字符串匹配；它与数据库名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **database.exclude.list** 属性。
database.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与数据库名称匹配，以便在监控中排除。当设置 **database.exclude.list** 时，连接器会监控每个数据库，但属性指定的数据库除外。 要匹配数据库的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据库的整个名称字符串匹配；它与数据库名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，请不要设置 **database.include.list** 属性。

属性	默认	描述
collection.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与要监控 MongoDB 集合的完全限定命名空间匹配。默认情况下，连接器会监控除 **本地和 admin** 数据库中除它们以外的所有集合。当设置了 **collection.include.list** 时，连接器只监控属性指定的集合。其他集合不包括在监控中。集合标识符的格式是 *databaseName.collectionName*。 要匹配命名空间的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与命名空间的整个名称字符串匹配，它与名称中的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **collection.exclude.list** 属性。
collection.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与要从监控中排除的 MongoDB 集合的完全限定命名空间匹配。当设置了 **collection.exclude.list** 时，连接器会监控每个集合，但属性指定的集合除外。集合标识符的格式是 *databaseName.collectionName*。 要匹配命名空间的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与命名空间的整个名称字符串匹配，它不与数据库名称中存在的子字符串匹配。 如果您在配置中包含此属性，请不要设置 **collection.include.list** 属性。
snapshot.mode	**初始**	指定连接器启动时执行快照的条件。将属性设置为以下值之一： **初始** 当连接器启动时，如果没有检测到偏移主题中的值，它会执行数据库的快照。 **never** 当连接器启动时，它会跳过快照过程，并立即开始将数据库记录的操作更改事件流传输到 oplog。

属性	默认	描述
capture.mode	change_streams_update_full	指定连接器用来捕获 MongoDB 服务器的 **update** 事件更改的方法。将此属性设置为以下值之一： **change_streams** **update** 事件消息不包括完整文档。消息不包含代表更改**前**文档状态的字段。 **change_streams_update_full** **update** 事件消息包括完整文档。消息不包含代表更新前文档状态的 **before** 字段。事件消息返回 **after** 字段中文档的完整状态。  **注意** 在某些情况下，当将 **capture.mode** 配置为返回完整文档时，更新事件消息的 **updateDescription** 和 **after** 字段可能会报告不一致的值。在将多个更新应用到快速成功的文档后，这些差异可能会导致。连接器仅在收到事件的 **updateDescription** 字段中描述的更新后，从 MongoDB 数据库请求完整的文档。如果后续更新在连接器可以从数据库检索源文档前修改它，则连接器会收到稍后更新修改的文档。 **change_streams_update_full_with_pre_image** **update** 事件消息包括完整文档，并包含一个代表 **更改前** 文档状态的字段。 **change_streams_with_pre_image** **更新** 事件不包括完整文档，而是包含一个代表 **更改前** 文档状态的字段。

属性	默认	描述
snapshot.include.collection.list	**collection.include.list**中指定的所有集合	一个可选的、以逗号分隔的正则表达式列表，与您要包含在快照中的模式的完全限定域名(<***databaseName>*** **.<*collectionName*>**)匹配。指定的项目必须在连接器的 **collection.include.list** 属性中命名。只有在连接器的 **snapshot.mode** 属性设置为除 **never** 的值时，此属性才会生效。 此属性不会影响增量快照的行为。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个名称字符串匹配；它与 schema 名称中可能存在的子字符串匹配。
field.exclude.list	*空字符串*	可选的、以逗号分隔的字段名称列表，这些字段应排除在更改事件消息值中。字段的完全限定域名格式为 *databaseName.collectionName.fieldName.nestedFieldName*，其中 *databaseName* 和 *collectionName* 可能包含与任何字符匹配的通配符 jpeg。
field.renames	*空字符串*	可选的、以逗号分隔的字段替换列表，用于重命名更改事件消息值中的字段。字段的完全限定替换格式为 *databaseName.collectionName.fieldName.nestedFieldName:newNestedFieldName*，其中 *databaseName* 和 *collectionName* 可以包含与任何字符匹配的通配符 packagemanifests，用于确定字段重命名映射。下一个字段替换应用于列表中之前字段替换的结果，因此在重命名同一路径中的多个字段时请注意这一点。

属性	默认	描述
tasks.max	**1**	指定连接器用来连接到分片集群的最大任务数量。当您将连接器与单个 MongoDB 副本集搭配使用时，默认值为可以接受。但是，当集群包含多个分片时，要启用 Kafka Connect 来分发每个副本集的工作，请指定等于或大于集群中的分片数量的值。然后，MongoDB 连接器可以使用单独的任务连接到集群中每个分片的副本集。  **注意** 只有在连接器连接到分片 MongoDB 集群，并且 **mongodb.connection.mode** 属性设置为 **replica_set** 时，此属性才会生效。当 **mongodb.connection.mode** 设置为 **sharded** 时，或者连接器连接到未分片的 MongoDB 副本集部署时，连接器会忽略此设置，并默认使用单个任务。
snapshot.max.threads	**1**	正整数值，用于指定用于在副本集中执行集合空间同步的最大线程数。默认为 1。
tombstones.on.delete	**true**	控制 *delete* 事件是否后跟一个 tombstone 事件。 **true** - 一个 delete 操作由 *delete* 事件和后续 tombstone 事件表示。 **false** - 仅有一个 *delete* 事件被抛出。 删除源记录后，发出 tombstone 事件（默认行为）可让 Kafka 在为主题启用了 日志 压缩时完全删除与已删除行键相关的所有事件。
snapshot.delay.ms	没有默认值	在启动后，连接器在进行快照前应等待的时间（以毫秒为单位）。 可用于在集群中启动多个连接器时避免快照中断，这可能会导致连接器的重新平衡。
snapshot.fetch.size	**0**	指定在拍摄快照时每个集合中应一次读取的最大文档数。连接器将在这个大小的多个批处理中读取集合内容。 默认值为 0，这表示服务器选择合适的获取大小。

属性	默认	描述
schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列
field.name.adjustment.mode	none	指定应如何调整字段名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列 如需了解更多详细信息，请参阅 Avro 命名。
mongodb.hosts	没有默认值	以逗号分隔的主机名和端口对列表（格式为 'host' 或 'host:port'）在副本集中的 MongoDB 服务器。列表中可以包含单个主机名和端口对。  **注意** 此属性已弃用，应该被 **+mongodb.connection.string** 替代。

以下高级配置属性具有很好的默认值，这些默认值在大多数情况下将可以正常工作，因此很少需要在连接器的配置中指定。

表 5.15. Debezium MongoDB 连接器高级配置属性

属性	默认	描述
max.batch.size	**2048**	正整数值，指定每个应在此连接器迭代过程中处理的事件的最大大小。默认值为 2048。

属性	默认	描述
max.queue.size	8192	正整数值，用于指定阻塞队列可以保存的最大记录数。当 Debezium 从数据库读取事件时，它会将事件放置在阻塞队列中，然后再将它们写入 Kafka。阻塞队列可以提供从数据库读取更改事件时，连接器最快于将其写入 Kafka 的信息，或者在 Kafka 不可用时从数据库读取更改事件。当连接器定期记录偏移时，队列中保存的事件会被忽略。始终将 **max.queue.size** 的值设置为大于 **max.batch.size** 的值。
max.queue.size.in.bytes	0	一个长的整数值，用于指定阻塞队列的最大卷（以字节为单位）。默认情况下，不会为阻塞队列指定卷限制。要指定队列可以消耗的字节数，请将此属性设置为正长值。 如果还设置了 **max.queue.size**，当队列的大小达到任一属性指定的限制时，写入队列将被阻止。例如，如果您设置了 **max.queue.size=1000**、和 **max.queue.size.in.bytes=5000**，在队列包含 1000 个记录后，或者队列中记录的卷达到 5000 字节后，写入队列会被阻止。
poll.interval.ms	1000	正整数值，指定连接器在每个迭代过程中应等待的毫秒数，以便出现新更改事件。默认值为 500 毫秒，或 0.5 秒。
connect.backoff.initial.delay.ms	1000	正整数值，指定在第一次连接尝试或没有主可用后尝试重新连接到主时的初始延迟。默认为 1 秒 (1000 ms)。
connect.backoff.max.delay.ms	1000	正整数值，指定在重复失败连接尝试或没有主可用后尝试重新连接到主时的最大延迟。默认为 120 秒(120,000 ms)。
connect.max.attempts	16	正整数值，指定发生异常和任务中止前尝试到副本集的主连接的最大失败数。默认为 16，在失败时，**connect.backoff.initial.delay.ms** 和 **connect.backoff.max.delay.ms** 的默认值会超过 20 分钟的尝试。

属性	默认	描述
heartbeat.interval.ms	0	控制发送心跳消息的频率。 此属性包含一个间隔，以毫秒为单位定义连接器将信息发送到 heartbeat 主题的频率。这可用于监控连接器是否仍然从数据库接收更改事件。在较长的时间段内，您还应利用心跳消息，以防在非捕获的集合中只更改了心跳消息。在这种情况下，连接器将继续从数据库读取 oplog/change 流，但永远不会将任何更改信息发送到 Kafka，这意味着没有将偏移更新提交到 Kafka。这将导致 oplog 文件被轮转，但连接器不会注意到它，因此在重启一些事件时，这个事件将不再需要重新执行初始快照。 将此参数设置为 **0** 以不发送心跳信息。 默认禁用此选项。
skipped.operations	t	在流过程中将跳过的操作类型的逗号分隔列表。操作包括：用于 inserts/create、**u** 表示 updates/replace、**d** 表示删除、**t** 表示截断，**none** 不跳过上述操作。**默认情况下，为了与其他 Debezium 连接器保持一致，会跳过 truncate 操作（不会由此连接器发出）。但是，由于 MongoDB 不支持 截断更改事件，这实际上与指定 none 相同。**
snapshot.collection.filter.overrides	没有默认值	控制快照中包含的集合项目。此属性仅影响快照。以 *databaseName.collectionName* 格式指定以逗号分隔的集合名称列表。 对于您指定的每个集合，还要指定另一个配置属性：**snapshot.collection.filter.overrides.*databaseName.collectionName***。例如，其他配置属性的名称可能是：**snapshot.collection.filter.overrides.customers.orders**。将此属性设置为有效的过滤器表达式，它只检索您在快照中所需的项目。当连接器执行快照时，它只检索与过滤器表达式匹配的项目。
provide.transaction.metadata	false	当设置为 **true** Debezium 时，使用事务边界生成事件，并使用事务元数据增强数据事件信封。 如需了解更多详细信息，请参阅 事务元数据。
retriable.restart.connector.wait.ms	10000 (10 秒)	在发生可检索错误后重启连接器前等待的毫秒数。

属性	默认	描述
mongodb.poll.interval.ms	**30000**	连接器轮询新的、删除或更改的副本集的时间间隔。
mongodb.connect.timeout.ms	10000 (10 秒)	驱动程序在新连接尝试中止前等待的毫秒数。
mongodb.heartbeat.frequency.ms	10000 (10 秒)	集群监控器尝试访问每台服务器的频率。
mongodb.socket.timeout.ms	0	套接字上的发送/接收在超时发生前可能需要的毫秒数。**0** 代表禁用此行为。
mongodb.server.selection.timeout.ms	30000 (30 秒)	驱动程序在超时前等待选择服务器的毫秒数，并抛出错误。
cursor.pipeline	没有默认值	当流更改时，此设置应用处理来更改流事件，作为标准 MongoDB 聚合流管道的一部分。Pipeline 是一个 MongoDB 聚合管道，由数据库的说明组成，用于过滤或转换数据。这可用于自定义连接器使用的数据。此属性的值必须是 JSON 格式的允许 聚合管道阶段 的数组。请注意，这会在用于支持连接器的内部管道后附加（例如，过滤操作类型、数据库名称、集合名称等）。
cursor.pipeline.order	internal_first	用于构建有效 MongoDB 聚合流管道的顺序。将属性设置为以下值之一： **internal_first** 连接器定义的内部阶段会首先应用。这意味着，只有由连接器捕获的事件才会被连接器捕获到用户定义的阶段（通过设置 **cursor.pipeline**进行配置）。 **user_first** 'cursor.pipeline' 属性定义的阶段会首先应用。在这个模式中，所有事件（包括未由连接器捕获的事件）被反馈到用户定义的管道阶段。如果 **cursor.pipeline** 的值包含复杂操作，则此模式可能会对性能造成负面影响。
cursor.max.await.time.ms	**0**	指定 oplog/change 流光标将在导致执行超时异常前等待服务器生成结果的最大毫秒数。值 **0** 表示使用 server/driver 默认等待超时。

属性	默认	描述
signal.data.collection	没有默认值	用于向连接器发送信号的数据收集的完全限定名称。https://access.redhat.com/documentation/zh-cn/red_hat_integration/2023.q4/html-single/debezium_user_guide/index#debezium-signaling-enabling-source-signaling-channel使用以下格式指定集合名称：<*databaseName*> . < *collectionName*>
signal.enabled.channels	source	为连接器启用的信号频道名称列表。默认情况下，以下频道可用： • **source** • **kafka** • **file** • **jmx**
notification.enabled.channels	没有默认值	为连接器启用的通知频道名称列表。默认情况下，以下频道可用： • **sink** • **log** • **jmx**
incremental.snapshot.chunk.size	**1024**	连接器在增量快照块期间获取并读取内存的最大文档数。增加块大小可提高效率，因为快照会运行更多大小的快照查询。但是，较大的块大小还需要更多内存来缓冲快照数据。将块大小调整为提供环境中最佳性能的值。 增量快照是 Debezium MongoDB 连接器的技术预览功能。
topic.naming.strategy	**io.debezium.schema.DefaultTopicNamingStrategy**	应该用来确定数据更改、模式更改、事务、心跳事件等的主题名称，默认为 **DefaultTopicNamingStrategy**。
topic.delimiter	**.**	指定主题名称的分隔符，默认为 **.**。
topic.cache.size	**10000**	在绑定的并发哈希映射中用于保存主题名称的大小。此缓存将有助于确定与给定数据收集对应的主题名称。

属性	默认	描述
topic.heartbeat.prefix	__debezium-heartbeat	控制连接器向其发送心跳信息的主题名称。主题名称具有此模式： *topic.heartbeat.prefix.topic.prefix* 例如，如果主题前缀是 **fulfillment**，则默认主题名称为 **__debezium-heartbeat.fulfillment**。
topic.transaction	Transactions	控制连接器向其发送事务元数据消息的主题名称。主题名称具有此模式： *topic.prefix.topic.transaction* 例如，如果主题前缀是 **fulfillment**，默认的主题名称为 **fulfillment.transaction**。
errors.max.retries	-1	在失败前，retriable 错误（如连接错误）的最大重试次数(-1 = no limit, 0 = disabled, > 0 = num of retries)。

Debezium 连接器 Kafka 信号配置属性

Debezium 提供了一组 signal.* 属性，用于控制连接器如何与 Kafka 信号主题进行交互。

下表描述了 Kafka 信号 属性。

表 5.16. Kafka 信号配置属性

属性	默认	描述
signal.kafka.topic	<topic.prefix>-signal	连接器监控用于临时信号的 Kafka 主题的名称。 **注意** 如果禁用了 自动主题创建，您必须手动创建所需的信号主题。需要信号主题来保留信号排序。信号主题必须具有单个分区。
signal.kafka.groupId	kafka-signal	Kafka 用户使用的组 ID 的名称。
signal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。每个对都引用 Debezium Kafka Connect 进程使用的 Kafka 集群。

属性	默认	描述
signal.kafka.poll.timeout.ms	100	一个整数值，用于指定连接器在轮询信号时等待的最大毫秒数。

Debezium 连接器传递信号 Kafka 使用者客户端配置属性

Debezium 连接器为信号 Kafka 使用者提供直通配置。透传信号属性以 signals.consumer. 前缀开始。例如，连接器将 signal.consumer.security.protocol=SSL 等属性传递给 Kafka 消费者。*

Debezium 从属性中剥离前缀，然后再将属性传递给 Kafka 信号消费者。

Debezium 连接器接收器通知配置属性

下表描述了 通知 属性。

表 5.17. sink 通知配置属性

属性	默认	描述
notification.sink.topic.name	没有默认值	从 Debezium 接收通知的主题名称。当您将 notification.enabled.channels 属性配置为将 sink 作为启用的通知频道之一时，需要此属性。

5.6. 监控 DEBEZIUM MONGODB 连接器性能

Debezium MongoDB 连接器除了支持 Zookeeper、Kafka 和 Kafka Connect 的内置支持外，还有两个指标类型。

- *快照指标 提供有关执行快照时连接器操作的信息。*
- *流指标 在连接器捕获更改和流更改事件记录时提供有关连接器操作的信息。*

Debezium 监控文档 提供了有关如何使用 JMX 公开这些指标的详细信息。

5.6.1. 在 MongoDB 快照过程中监控 Debezium

MBean 是 debezium.mongodb:type=connector-metrics,context=snapshot,server=

<topic.prefix> , task= <task.id>.

快照指标不会公开，除非快照操作处于活跃状态，或者快照自上次连接器启动以来发生。

下表列出了可用的 shapshot 指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个快照事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上次启动或重置后看到的事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在快照和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	队列的空闲容量，用于在快照和主 Kafka Connect 循环之间传递事件。
TotalTableCount	int	包括在快照中的表的总数。
RemainingTableCount	int	快照必须复制的表数。
SnapshotRunning	布尔值	快照是否已启动。
SnapshotPaused	布尔值	快照是否已暂停。
SnapshotAborted	布尔值	快照是否中止。
SnapshotCompleted	布尔值	快照是否完成。

属性	类型	描述
SnapshotDurationInSeconds	long	快照为止所花费的秒数，即使未完成也是如此。也包括快照暂停的时间。
SnapshotPausedDurationInSeconds	long	快照暂停的秒数。如果快照暂停几次，暂停的时间会添加。
RowsScanned	Map<String, Long>	包含快照中每个表的行数的映射。表会在处理过程中逐步添加到映射中。更新每个 10,000 行扫描并在完成表后。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

Debezium MongoDB 连接器还提供以下自定义快照指标：

属性	类型	描述
NumberOfDisconnects	long	数据库断开连接数。

5.6.2. 监控 Debezium MongoDB 连接器记录流

MBean 是 debezium.mongodb:type=connector-metrics,context=streaming,server=<topic.prefix> , task= <task.id>.

下表列出了可用的流指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个流事件。

属性	类型	描述
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上一次启动或指标重置以来看到的事件总数。
TotalNumberOfCreateEventsSeen	long	此连接器自上次启动或指标重置以来看到的创建事件总数。
TotalNumberOfUpdateEventsSeen	long	此连接器自上次启动或指标重置以来看到的更新事件总数。
TotalNumberOfDeleteEventsSeen	long	此连接器自上次启动或指标重置以来看到的删除事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的队列的可用容量。
Connected	布尔值	表示连接器目前是否连接到数据库服务器的标记。
MilliSecondsBehindSource	long	最后一次更改事件时间戳和连接器处理它之间的毫秒数。这些值将讨论运行数据库服务器和连接器的计算机上时钟之间的任何区别。
NumberOfCommittedTransactions	long	已提交的已处理事务的数量。
SourceEventPosition	Map<String, String>	最后收到的事件的协调。
LastTransactionId	字符串	最后处理事务的事务的事务标识符。

属性	类型	描述
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

Debezium MongoDB 连接器还提供以下自定义流指标：

属性	类型	描述
NumberOfDisconnects	long	数据库断开连接数。
NumberOfPrimaryElections	long	主节点选举数量。

5.7. DEBEZIUM MONGODB 连接器如何处理错误和问题

Debezium 是一个分布式系统，用于捕获多个上游数据库中的所有更改，永远不会丢失或丢失事件。当系统正常运行并谨慎管理时，Debezium 会在每次更改事件时发送一次。

如果出现错误，系统不会丢失任何事件。但是，当它从错误中恢复时，可能会重复一些更改事件。在这种情况下，Debezium （如 Kafka） 至少 提供更改事件。

以下主题详细介绍了 Debezium MongoDB 连接器如何处理各种错误和问题。

- *配置和启动错误*
- *MongoDB 变得不可用*
- *Kafka Connect 进程正常停止*

- Kafka Connect 进程崩溃
- Kafka 变得不可用
- 如果 snapshot.mode 设置为 initial，则连接器会在停止很长时间后失败
- MongoDB 丢失写入

配置和启动错误

在以下情况下，连接器在尝试启动时失败，在日志中报告错误或异常，并停止运行：

- 连接器的配置无效。
- 连接器无法使用指定的连接参数成功连接到 MongoDB。

失败后，连接器会尝试使用 exponential backoff 进行重新连接。您可以配置重新连接尝试的最大数量。

在这些情况下，这个错误将了解更多有关此问题的详细信息，并可能会有推荐的临时解决方案。当配置已被修正或 MongoDB 问题已被解决时，可以重启连接器。

MongoDB 变得不可用

当连接器运行后，如果任何 MongoDB 副本集的主节点不可用或无法访问，则连接器将尝试重新连接到主节点，使用 exponential backoff 来防止网络或服务器饱和。如果在可配置的连接尝试次数后主仍然不可用，则连接器将失败。

尝试重新连接由三个属性控制：

- connect.backoff.initial.delay.ms - 第一次尝试重新连接前的延迟，默认值为 1 秒(1000 毫秒)。
-

connect.backoff.max.delay.ms - 尝试重新连接前的最大延迟，默认值为 120 秒(120,000 毫秒)。

- *connect.max.attempts - 生成错误前的最大尝试次数，默认值为 16。*

每个延迟都是之前的延迟，最多为最大延迟。根据默认值，下表显示每个失败连接尝试的延迟，以及失败前的总累计时间。

重新连接尝试号	尝试前延迟，以秒为单位	尝试前的总延迟，以分钟和秒为单位
1	1	00:01
2	2	00:03
3	4	00:07
4	8	00:15
5	16	00:31
6	32	01:03
7	64	02:07
8	120	04:07
9	120	06:07
10	120	08:07
11	120	10:07
12	120	12:07
13	120	14:07
14	120	16:07
15	120	18:07
16	120	20:07

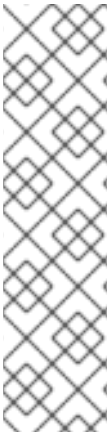
Kafka Connect 进程正常停止

如果 Kafka Connect 以分布式模式运行，并且 Kafka Connect 进程被正常停止，则在关闭该进程 Kafka Connect 之前，会将所有进程的连接器任务迁移到该组中的另一个 Kafka Connect 进程，新的连接器任务将准确获取之前的任务。在处理连接器任务时，在新进程中安全停止并重新启动时会有一个短暂的延迟。

如果组只包含一个进程，且该进程被安全停止，则 Kafka Connect 将停止连接器，并记录每个副本集的最后偏移量。重启后，副本集任务将持续保持关闭的位置。

Kafka Connect 进程崩溃

如果 Kafka Connector 进程意外停止，则运行的任何连接器任务都将终止，而不记录其最近处理的偏移。当 Kafka Connect 以分布式模式运行时，它会在其他进程中重启这些连接器任务。但是，MongoDB 连接器将从之前进程 记录 的最后偏移中恢复，这意味着新的替换任务可能会生成在崩溃前处理的一些相同更改事件。重复事件的数量取决于偏移刷新周期和数据卷在崩溃前更改。



注意

因为在从故障恢复过程中可能会重复一些事件，因此消费者应始终预测某些事件可能会重复。Debezium 更改是幂等的，因此一系列事件始终产生相同的状态。

Debezium 还包括每个更改事件消息，提供有关事件来源的源特定信息，包括 MongoDB 事件的唯一事务标识符(h)和时间戳(sec 和 ord)。消费者可以跟踪这些值，以知道它是否已看到特定的事件。

Kafka 变得不可用

当连接器生成更改事件时，Kafka Connect 框架会使用 Kafka producer API 在 Kafka 中记录这些事件。Kafka Connect 还会定期记录您在 Kafka Connect worker 配置中指定的频率，这些事件中显示的最新的偏移量。如果 Kafka 代理不可用，运行连接器的 Kafka Connect worker 进程只会重复尝试重新连接到 Kafka 代理。换句话说，连接器任务将直接暂停，直到可以重新建立连接，此时连接器将完全恢复它们关闭的位置。

如果 snapshot.mode 设置为 initial，则连接器会在停止很长时间后失败

如果连接器被安全停止，用户可能会继续对副本设置成员执行操作。连接器离线时发生的更改将继续记录在 MongoDB 的 oplog 中。在大多数情况下，在连接器重启后，它会读取 oplog 中的偏移值，以确定每个副本集传递的最后一个操作，然后从该点恢复流更改。重启后，当连接器停止时发生的数据库操作会正常发送到 Kafka，在一段时间后，连接器会捕获数据库。连接器捕获所需的时间取决于 Kafka 的功能和性能以及数据库中发生的更改卷。

但是，如果连接器长时间停止，则 MongoDB 会在连接器不活跃时清除 oplog，从而导致连接器的最

后位置丢失信息。连接器重启后，它无法恢复流，因为 oplog 不再包含前面的偏移值，用于标记连接器处理的最后一个操作。连接器还无法执行快照，因为它通常会在 snapshot.mode 属性设置为 initial 时，且没有偏移值。在这种情况下，存在不匹配，因为 oplog 不包含之前偏移的值，但连接器的内部 Kafka 偏移主题中存在偏移值。错误结果，连接器失败。

要从失败中恢复，请删除失败的连接器，并使用同一配置创建新连接器，但使用不同的连接器名称。当您启动新连接器时，它会执行快照以达到数据库的状态，然后恢复流。

MongoDB 丢失写入

在某些情况下，MongoDB 可能会丢失提交，这会导致 MongoDB 连接器无法捕获丢失的更改。例如，如果在应用更改后的主要崩溃并记录其 oplog 的更改，则 oplog 可能会在次要节点读取其内容前不可用。因此，被选为新主节点的辅助节点可能会缺少其 oplog 中的最新更改。

目前，在 MongoDB 中无法防止这个副作用。

第 6 章 MYSQL 的 DEBEZIUM 连接器

MySQL 有一个二进制日志(binlog)，它按照它们提交到数据库的顺序记录所有操作。这包括对表模式的更改，以及对表中的数据的更改。MySQL 使用 binlog 进行复制和恢复。

Debezium MySQL 连接器读取 binlog，为行级 INSERT、UPDATE 和 DELETE 操作生成更改事件，并将更改事件发送到 Kafka 主题。客户端应用程序读取这些 Kafka 主题。

因为 MySQL 通常会在指定时间段内清除 binlogs，因此 MySQL 连接器会针对每个数据库执行初始 一致的快照。MySQL 连接器从创建快照的时间点读取 binlog。

有关与此连接器兼容的 MySQL 数据库版本的详情，请查看 Debezium 支持的配置页面。

使用 Debezium MySQL 连接器的信息和步骤进行组织，如下所示：

6.1. DEBEZIUM MYSQL 连接器的工作方式

连接器支持的 MySQL 拓扑概述可用于规划应用程序。为了优化配置和运行 Debezium MySQL 连接器，了解连接器如何跟踪表结构、公开模式更改、执行快照以及确定 Kafka 主题名称。

详情包括在以下主题中：

6.1.1. Debezium 连接器支持的 MySQL 拓扑

Debezium MySQL 连接器支持以下 MySQL 拓扑：

Standalone

当使用单个 MySQL 服务器时，服务器必须启用 binlog (并选择性地启用 GTIDs)，以便 Debezium MySQL 连接器可以监控服务器。这通常可以接受，因为二进制日志也可以用作 增量备份。在这种情况下，MySQL 连接器总是连接到并遵循这个独立 MySQL 服务器实例。

主和副本

Debezium MySQL 连接器可以遵循其中一个主服务器或其中一个副本(如果该副本启用了 binlog)，但连接器只会看到对该服务器可见的集群的更改。通常，除了多主拓扑外，这不是问题。

连接器在服务器的 binlog 中记录其位置，这在集群中的每个服务器都有所不同。因此，连接器必

须只遵循一个 MySQL 服务器实例。如果该服务器失败，必须在连接器继续之前重启或恢复该服务器。

高可用性集群

MySQL 存在各种 高可用性解决方案，它们可以更容易容许，并且几乎立即从问题和故障中恢复。大多数 HA MySQL 集群使用 GTID，以便副本可以在任何主服务器上跟踪所有更改。

Multi-primary

网络数据库(NDB)集群复制 使用一个或多个 MySQL 副本节点，它们各自从多个主服务器复制。这是聚合多个 MySQL 集群复制的强大方法。这个拓扑需要使用 GTID。

Debezium MySQL 连接器可以使用这些多主 MySQL 副本作为源，只要新副本被发现到旧副本，就可以切换到不同的多主 MySQL 副本。也就是说，新副本具有在第一个副本中看到的所有事务。即使连接器只使用数据库和/或表的子集，当尝试重新连接到新的多主 MySQL 副本时，也可以将连接器配置为包含或排除特定的 GTID 源，并在 binlog 中找到正确的位置。

托管

支持 Debezium MySQL 连接器以使用托管选项，如 Amazon RDS 和 Amazon Aurora。

由于这些托管选项不允许全局读取锁定，因此表级锁定用于创建 一致的快照。

6.1.2. Debezium MySQL 连接器如何处理数据库架构更改

当数据库客户端查询数据库时，客户端将使用数据库的当前架构。但是，数据库模式可以随时更改，这意味着连接器必须能够识别每个插入、更新或删除操作被记录的时间。另外，连接器不一定将当前的模式应用到每个事件。如果事件相对旧，则应用当前模式之前可能会记录该事件。

为确保在架构更改后正确处理事件，MySQL 仅包含在事务日志中，不仅影响数据的行级更改，还应用于数据库的 DDL 语句。当连接器在 binlog 中遇到这些 DDL 语句时，它会解析它们并更新每个表模式的内存表示。连接器使用此模式表示来识别每个插入、更新或删除操作时表的结构，并生成适当的更改事件。在单独的数据库架构历史记录 Kafka 主题中，连接器记录所有 DDL 语句，以及 binlog 中出现每个 DDL 语句的位置。

当连接器在崩溃或安全停止后重启时，它从特定位置（即时间点）开始读取 binlog。连接器通过读取数据库模式历史记录 Kafka 主题并将所有 DDL 语句解析为连接器启动的 binlog 中，以此重建此时存在的表结构。

此数据库架构历史记录主题仅用于内部连接器。另外，连接器也可以将 模式更改事件发送到用于消费

者应用程序的不同主题。

当 MySQL 连接器捕获表中的更改时，会应用 gh-ost 或 pt-online-schema-change 等模式更改，在迁移过程中会创建帮助程序表。您必须配置连接器来捕获这些帮助程序表中的更改。如果消费者不需要为帮助程序表生成的记录，请配置 单个消息转换(SMT) 从连接器发出的消息中删除这些记录。

其他资源

- 接收 Debezium 事件记录 的主题的默认名称。

6.1.3. Debezium MySQL 连接器如何公开数据库架构更改

您可以配置 Debezium MySQL 连接器来生成模式更改事件，该事件描述了应用到数据库中表的架构更改。连接器将模式更改事件写入名为 < topicPrefix> 的 Kafka 主题，其中 topicPrefix 是 topic.prefix 连接器配置属性中指定的命名空间。连接器发送到 schema 更改主题的消息包含一个有效负载，以及可选的包含更改事件消息的 schema。

模式更改事件消息的有效负载包括以下元素：

ddl

提供会导致架构更改的 SQL CREATE、ALTER 或 DROP 语句。

databaseName

将 DDL 语句应用到的数据库的名称。databaseName 的值充当 message 键。

pos

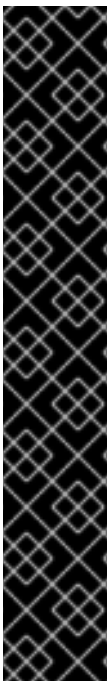
语句出现在 binlog 中的位置。

tableChanges

架构更改后整个表模式的结构化表示。tableChanges 字段包含一个数组，其中包含表的每个列的条目。由于结构化表示以 JSON 或 Avro 格式呈现数据，因此用户可轻松读取消息，而不必先通过 DDL 解析器处理它们。

重要

对于处于捕获模式的表，连接器不仅将模式更改的历史记录存储在 schema 更改主题中，也存储在内部数据库架构历史记录主题中。内部数据库架构历史记录主题仅用于连接器，它不适用于消耗应用程序直接使用。确保需要通知架构更改的应用程序只消耗来自 schema 更改主题的信息。



重要

切勿对数据库架构历史记录主题进行分区。要使数据库架构历史记录主题正常工作，它必须维护连接器发出的事件记录的全局顺序。

要确保主题没有在分区间分割，请使用以下方法之一为主题设置分区计数：

- 如果您手动创建数据库架构历史记录主题，请指定分区计数 1。
- 如果您使用 Apache Kafka 代理自动创建数据库 schema 历史记录主题，则会创建该主题，将 Kafka num.partitions配置选项 的值设置为 1。

警告

连接器发出到其 schema 更改主题的消息格式处于 incubating 状态，并可能在没有通知的情况下改变。

示例：消息发送到 MySQL 连接器模式更改主题

以下示例显示了 JSON 格式的典型的模式更改消息。该消息包含表模式的逻辑表示。

```
{
  "schema": { },
  "payload": {
    "source": { 1
      "version": "2.3.4.Final",
      "connector": "mysql",
      "name": "mysql",
      "ts_ms": 1651535750218, 2
```

```
    "snapshot": "false",
    "db": "inventory",
    "sequence": null,
    "table": "customers",
    "server_id": 223344,
    "gtid": null,
    "file": "mysql-bin.000003",
    "pos": 570,
    "row": 0,
    "thread": null,
    "query": null
  },
  "databaseName": "inventory", 3
  "schemaName": null,
  "ddl": "ALTER TABLE customers ADD middle_name varchar(255) AFTER first_name", 4
  "tableChanges": [ 5
    {
      "type": "ALTER", 6
      "id": "\"inventory\".\"customers\"", 7
      "table": { 8
        "defaultCharsetName": "utf8mb4",
        "primaryKeyColumnNames": [ 9
          "id"
        ],
        "columns": [ 10
          {
            "name": "id",
            "jdbcType": 4,
            "nativeType": null,
            "typeName": "INT",
            "typeExpression": "INT",
            "charsetName": null,
            "length": null,
            "scale": null,
            "position": 1,
            "optional": false,
            "autoIncremented": true,
            "generated": true
          },
          {
            "name": "first_name",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "VARCHAR",
            "typeExpression": "VARCHAR",
            "charsetName": "utf8mb4",
            "length": 255,
            "scale": null,
            "position": 2,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "middle_name",
```

```
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "VARCHAR",
            "typeExpression": "VARCHAR",
            "charsetName": "utf8mb4",
            "length": 255,
            "scale": null,
            "position": 3,
            "optional": true,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "last_name",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "VARCHAR",
            "typeExpression": "VARCHAR",
            "charsetName": "utf8mb4",
            "length": 255,
            "scale": null,
            "position": 4,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "email",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "VARCHAR",
            "typeExpression": "VARCHAR",
            "charsetName": "utf8mb4",
            "length": 255,
            "scale": null,
            "position": 5,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          }
        ],
        "attributes": [ 11
          {
            "customAttribute": "attributeValue"
          }
        ]
      }
    }
  ]
 }
}
```

表 6.1. 向 schema 更改主题发送的消息中字段的描述

项	字段名称	描述
1	**source**	**source** 字段与连接器写入表特定主题的标准数据更改事件完全相同。此字段对于在不同主题上关联事件非常有用。
2	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 在源对象中，ts_ms 表示数据库中进行更改的时间。通过将 payload.source.ts_ms 的值与 payload.ts_ms 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
3	**databaseName** **schemaName**	标识包含更改的数据库和架构。**databaseName** 字段的值被用作记录的 message 键。
4	**ddl**	此字段包含负责架构更改的 DDL。**ddl** 字段可以包含多个 DDL 语句。每个语句都应用于 **databaseName** 字段中的数据库。多个 DDL 语句按照它们应用到数据库的顺序出现。 客户端可以提交应用到多个数据库的 DDL 语句。如果 MySQL 以原子方式应用它们，连接器会按顺序获取 DDL 语句，按数据库对它们进行分组，并为每个组创建一个 schema 更改事件。如果 MySQL 单独应用它们，则连接器会为每个语句创建单独的 schema 更改事件。
5	**tableChanges**	包含 DDL 命令生成的模式更改的一个或多个项目的数组。
6	**type**	描述更改的类型。该值如下之一： **创建** 已创建的表。 **更改** 修改表。 **DROP** 表已删除。
7	**id**	创建、更改或丢弃的表的完整标识符。如果是表重命名，这个标识符是 < ***old>、 <new>***; 表名称的串联。
8	**table**	代表应用更改后的表元数据。
9	**primaryKeyColumnNames**	组成表主密钥的列的列表。
10	**columns**	更改表中每个列的元数据。

项	字段名称	描述
11	**属性**	每个表更改的自定义属性元数据。

如需更多信息，请参阅 schema 历史记录主题。

6.1.4. Debezium MySQL 连接器如何执行数据库快照

当 Debezium MySQL 连接器首次启动时，它会执行数据库的初始一致快照。这个快照可让连接器为数据库的当前状态建立基准。

Debezium 可以在运行快照时使用不同的模式。快照模式由 snapshot.mode 配置属性决定。属性的默认值为 初始。您可以通过更改 snapshot.mode 属性的值来自定义连接器创建快照的方式。

您可以在以下部分找到有关快照的更多信息：

- *第 6.1.5 节 “临时快照”*
- *第 6.1.6 节 “增量快照”*

连接器在执行快照时完成一系列任务。快照模式以及对数据库有效的表锁定策略的具体步骤会有所不同。当 Debezium MySQL 连接器执行 使用全局读取锁 或 表级 锁定的初始快照时，Debezium MySQL 连接器可以完成不同的步骤。

6.1.4.1. 使用全局读锁的初始快照

您可以通过更改 snapshot.mode 属性的值来自定义连接器创建快照的方式。如果您配置不同的快照模式，连接器使用这个工作流的修改版本完成快照。有关不允许全局读取锁定的环境中快照进程的详情，请查看 表级锁定的快照工作流。

Debezium MySQL 连接器用来执行带有全局读锁的初始快照的默认工作流

下表显示了 Debezium 遵循的工作流中的步骤，以使用全局读取锁定创建快照。

步骤	操作
1	建立与数据库的连接。
2	确定要捕获的表。默认情况下，连接器捕获所有非系统表的数据。快照完成后，连接器将继续流传输指定表的数据。如果您希望连接器只从特定表捕获数据，您可以通过设置 **table.include.list** 或 **table.exclude.list** 等属性来只捕获表或表元素子集的数据。
3	获取表上的全局读取锁定，以捕获给其他数据库客户端阻止 *写入*。 快照本身不会阻止其他客户端应用 DDL，这可能会影响连接器的尝试读取 binlog 位置和表模式。连接器在读取 binlog 位置时保留全局读取锁定，并在以后的步骤中释放锁定。
4	使用 可重复的读取语义 启动事务，以确保事务中的所有后续读取都针对 *一致的快照* 完成。  **注意** 使用这些隔离语义可能会减慢快照的进度。如果快照完成用时过长，请考虑使用不同的隔离配置，或者跳过初始快照并运行 增量快照。
5	读取当前的 binlog 位置。
6	捕获数据库中所有表的结构，或者为捕获指定的所有表。连接器在其内部数据库模式历史记录主题中保留模式信息，包括所有必要的 **DROP...** 和 **CREATE...** DDL 语句。 架构历史记录提供有关发生更改事件时生效的结构的信息。  **注意** 默认情况下，连接器捕获数据库中每个表的 schema，包括没有配置为捕获的表。如果没有为捕获配置表，则初始快照只捕获其结构；它不会捕获任何表数据。 有关为什么没有包括在初始快照中的表的快照保留模式信息，请参阅 了解为什么初始快照捕获所有表的 schema。
7	释放在第 3 步中获得的全局读取锁定。其他数据库客户端现在可以写入数据库。
8	在连接器在 Step 5 中读取的 binlog 位置，连接器开始扫描为捕获的表。在扫描过程中，连接器完成以下任务： 1. 确认表已在快照开始前创建。如果表是在快照启动后创建的，连接器会跳过表。快照完成后，连接器过渡到 streaming，它会发出快照开始后创建的任何表的更改事件。 2. 为从表获取的每行生成 **读取** 事件。所有 **读取** 事件都包含相同的 binlog 位置，这是在第 5 步中获取的位置。 3. 将每个 **读取** 事件发送到源表的 Kafka 主题。 4. 释放数据表锁定（如果适用）。

步骤	操作
9	提交事务。
10	在连接器偏移中记录快照成功完成。

生成的初始快照捕获捕获捕获的表中每行的当前状态。在这个基准状态中，连接器会捕获后续更改。

在快照进程开始后，如果进程因为连接器失败、重新平衡或其他原因而中断，则进程会在连接器重启后重启。

连接器完成初始快照后，它会继续从在第 5 步中读取的位置进行流，使其不会错过任何更新。

如果连接器因为任何原因而再次停止，它会在重启后从之前关闭的位置恢复流更改。

连接器重启后，如果删除了日志，则日志中连接器的位置可能不再可用。然后，连接器会失败，并返回一个错误，表示需要新的快照。要将连接器配置为在这种情况下自动启动快照，请将 snapshot.mode 属性的值设置为 when_needed。有关 Debezium MySQL 连接器故障排除的更多信息，请参阅 当出现问题时的行为。

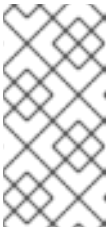
6.1.4.2. 使用表级锁定的初始快照

在某些数据库环境中，管理员不允许全局读取锁定。如果 Debezium MySQL 连接器检测到不允许全局读取锁定，连接器会在执行快照时使用表级锁定。要使连接器执行使用表级锁的快照，Debezium 连接器用来连接到 MySQL 的数据库帐户必须具有 LOCK TABLES 权限。

Debezium MySQL 连接器用来执行带有表级别锁定的初始快照的默认工作流

以下工作流列出了 Debezium 使用表级读取锁定创建快照所采取的步骤。有关不允许全局读取锁定的环境中快照进程的详情，请查看 全局读取锁定的快照工作流。

步骤	操作
1	建立与数据库的连接。
2	确定要捕获的表。默认情况下，连接器捕获所有非系统表。要让连接器捕获表或表元素的子集，您可以设置多个 **include** 和 **exclude** 属性来过滤数据，如 **table.include.list** 或 **table.exclude.list**。

步骤	操作
3	获取表级锁定。
4	使用 可重复的读取语义 启动事务，以确保事务中的所有后续读取都针对 *一致的快照* 完成。
5	读取当前的 binlog 位置。
6	读取连接器配置为捕获更改的数据库和表的 schema。连接器在其内部数据库模式历史记录主题中保留模式信息，包括所有必要的 **DROP...** 和 **CREATE...** DDL 语句。 架构历史记录提供有关发生更改事件时生效的结构的信息。  **注意** 默认情况下，连接器捕获数据库中每个表的 schema，包括没有配置为捕获的表。如果没有为捕获配置表，则初始快照只捕获其结构；它不会捕获任何表数据。 有关为什么没有包括在初始快照中的表的快照保留模式信息，请参阅 了解为什么初始快照捕获所有表的 schema。
7	在连接器在 Step 5 中读取的 binlog 位置，连接器开始扫描为捕获的表。在扫描过程中，连接器完成以下任务： 1. 确认表已在快照开始前创建。如果表是在快照启动后创建的，连接器会跳过表。快照完成后，连接器过渡到 streaming，它会发出快照开始后创建的任何表的更改事件。 2. 为从表获取的每行生成 **读取** 事件。所有 **读取** 事件都包含相同的 binlog 位置，这是在第 5 步中获取的位置。 3. 将每个 **读取** 事件发送到源表的 Kafka 主题。 4. 释放数据表锁定（如果适用）。
8	提交事务。
9	释放表级锁定。其他数据库客户端现在可以写入任何之前锁定的表。
10	在连接器偏移中记录快照成功完成。

6.1.4.3. 初始快照捕获所有表的 schema 历史记录的描述

连接器运行的初始快照捕获两种类型的信息：

表数据

在连接器的 table.include.list 属性中命名的表中的 INSERT、UPDATE 和 DELETE 操作的信息。

模式数据

描述应用到表的结构更改的 DDL 语句。模式数据会保留给内部模式历史记录主题，以及连接器的 schema 更改主题（如果配置了）。

运行初始快照后，您可能会注意到快照捕获没有指定用于捕获的表的模式信息。默认情况下，初始快照旨在捕获数据库中存在的每个表的模式信息，而不仅仅是从指定为捕获的表的表。连接器要求表的模式存在于架构历史记录主题中，然后才能捕获表。通过启用初始快照来捕获不是原始捕获集一部分的表的 schema 数据，Debebe 准备好连接器，以便稍后需要捕获这些表中的事件数据。如果初始快照没有捕获表的 schema，您必须将模式添加到历史记录主题，然后才能从表中捕获数据。

在某些情况下，您可能想要限制初始快照中的模式捕获。当您要减少完成快照所需的时间时，这非常有用。或者，当 Debezium 通过可访问多个逻辑数据库的用户帐户连接到数据库实例时，但您希望连接器只从特定逻辑数据库中的表捕获更改。

附加信息

- 从不是由初始快照捕获的表捕获数据（没有模式更改）
- 从不是由初始快照捕获的表捕获数据（应用程序更改）
- 设置 schema.history.internal.store.only.captured.tables.ddl 属性，以指定从中捕获模式信息的表。
- 设置 schema.history.internal.store.only.captured.databases.ddl 属性，以指定从中捕获模式更改的逻辑数据库。

6.1.4.4. 从不是由初始快照捕获的表捕获数据（没有模式更改）

在某些情况下，您可能希望连接器从其模式未被初始快照捕获的表中捕获数据。根据连接器配置，初始快照只能捕获数据库中特定表的表模式。如果历史记录主题中没有表模式，连接器将无法捕获表，并报告缺少的 schema 错误。

您可能仍然能够从表中捕获数据，但您必须执行额外的步骤来添加表模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。

- 在事务日志中，表的所有条目都使用相同的模式。有关从具有存结构更改的新表中捕获数据的详情，请参考从 *未由初始快照(schema 更改)捕获的表中的捕获数据*。

流程

1. 停止连接器。
2. 删除由 schema.history.internal. *kafka.topic 属性指定的内部数据库架构历史记录* 主题。
3. 对连接器配置应用以下更改：
   a. 将 *snapshot.mode* 设置为 schema_only_recovery。
   b. 将 *schema.history.internal.store.only.captured.tables.ddl* 的值设置为 false。
   c. 添加您希望连接器捕获至 table.include.list 的表。这样可保证将来，连接器可以重建所有表的 schema 历史记录。
4. 重启连接器。快照恢复过程根据表的当前结构重建模式历史记录。
5. （可选）在快照完成后，启动一个 *增量快照* 来捕获新添加的表的现有数据，以及该连接器关闭时发生的其他表的更改。
6. （可选）将 snapshot.mode 重置为 schema_only，以防止连接器在以后的重启后启动恢复。

6.1.4.5. 从不是由初始快照捕获的表捕获数据（应用程序更改）

如果架构更改应用到表，则在架构更改前提交的记录与更改后提交的不同结构不同。当 Debezium 从表中捕获数据时，它会读取 schema 历史记录，以确保它为每个事件应用正确的模式。如果 schema 历史记录主题中没有 schema，则连接器无法捕获表，并出现错误结果。

如果要从初始快照捕获的表中捕获数据，并且修改了表的 schema，则必须将模式添加到历史记录主

题中（如果它还没有可用）。您可以通过运行新的模式快照或运行表的初始快照来添加模式。

前提条件

- *您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。*
- *架构更改应用于表，以便捕获的记录没有统一结构。*

流程

初始快照捕获了所有表的模式(storage.only.captured.tables.ddl 设置为 false)

1. *编辑 table.include.list 属性，以指定您要捕获的表。*
2. *重启连接器。*
3. *如果要从新添加的表中捕获现有数据，则启动 增量快照。*

初始快照没有捕获所有表的模式(storage.only.captured.tables.ddl 设置为 true)

如果初始快照没有保存您要捕获的表的模式，请完成以下步骤之一：

流程 1：架构快照，后跟增量快照

在此过程中，连接器首先执行 schema 快照。然后，您可以启动增量快照，使连接器能够同步数据。

1. *停止连接器。*
2. *删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。*
3. *清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。*

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. *为连接器配置中的属性设置值，如以下步骤所述：*

   a. *将 snapshot.mode 属性的值设置为 schema_only。*

   b. *编辑 table.include.list 以添加您要捕获的表。*

5. *重启连接器。*

6. *等待 Debezium 捕获新表和现有表的模式。在连接器停止后发生任何表的数据更改不会被捕获。*

7. *为确保没有丢失数据，请启动 增量快照。*

步骤 2：初始快照，后跟可选的增量快照

在此过程中，连接器执行数据库的完整初始快照。与任何初始快照一样，在具有多个大型表的数据库中，运行初始快照可能会非常耗时。快照完成后，您可以选择触发增量快照来捕获连接器离线时发生的任何更改。

1. *停止连接器。*

2. *删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。*

3. *清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。*

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 编辑 table.include.list 以添加您要捕获的表。

5. 为连接器配置中的属性设置值，如以下步骤所述：

   a. 将 snapshot.mode 属性的值设置为 initial。

   b. （可选）将 schema.history.internal.store.only.captured.tables.ddl 设置为 false。

6. 重启连接器。连接器获取完整的数据库快照。快照完成后，连接器会过渡到 streaming。

7. （可选）要捕获连接器离线时更改的任何数据，请启动 增量快照。

6.1.5. 临时快照

默认情况下，连接器仅在首次启动后运行初始快照操作。在正常情况下，在这个初始快照后，连接器不会重复快照过程。连接器捕获的任何更改事件数据都只通过流处理。

然而，在某些情况下，连接器在初始快照期间获得的数据可能会过时、丢失或不完整。为了提供总结表数据的机制，Debezium 包含一个执行临时快照的选项。数据库中的以下更改可能会导致执行临时快照：

- 连接器配置会被修改为捕获不同的表集合。
- Kafka 主题已删除，必须重建。
- 由于配置错误或某些其他问题导致数据损坏。

您可以通过启动所谓的 临时快照来为之前捕获的表重新运行快照。临时快照需要使用 信号表。您可以通过向 Debezium 信号表发送信号请求来发起临时快照。

当您启动现有表的临时快照时，连接器会将内容附加到表已存在的主题中。如果删除了之前存在的主题，如果启用了 自动主题创建，Debezium 可以自动创建主题。

临时快照信号指定要包含在快照中的表。快照可以捕获整个数据库的内容，或者仅捕获数据库中表的子集。另外，快照也可以捕获数据库中表的内容子集。

您可以通过将 execute-snapshot 消息发送到信号表来指定要捕获的表。将 execute-snapshot 信号类型设置为 增量，并提供快照中包含的表名称，如下表所述：

表 6.2. 临时 execute-snapshot 信号记录的示例

字段	默认	值
type	**incremental**	指定您要运行的快照类型。 设置类型是可选的。目前，您只能请求 **增量** 快照。
data-collections	*N/A*	包含与要快照的表的完全限定域名匹配的正则表达式的数组。 名称的格式与 **signal.data.collection** 配置选项的格式相同。
additional-condition	*N/A*	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。
surrogate-key	*N/A*	可选字符串，指定连接器在快照过程中用作表的主键的列名称。

触发临时快照

您可以通过向信号表中添加 execute-snapshot 信号类型的条目来发起临时快照。连接器处理消息后，它会开始快照操作。快照进程读取第一个和最后一个主密钥值，并使用这些值作为每个表的开头和结束点。根据表中的条目数量以及配置的块大小，Debezium 会将表划分为块，并一次性执行每个块的快照。

目前，execute-snapshot 操作类型仅触发 增量快照。如需更多信息，请参阅 增加快照。

6.1.6. 增量快照

为了提供管理快照的灵活性，Debezium 包含附加快照机制，称为 增量快照。增量快照依赖于 Debezium 机制 向 Debezium 连接器发送信号。

在增量快照中，除了一次捕获数据库的完整状态，就像初始快照一样，Debebe 会在一系列可配置的块中捕获每个表。您可以指定您希望快照捕获的表 以及每个块的大小。块大小决定了快照在数据库的每个获取操作期间收集的行数。增量快照的默认块大小为 1024 行。

当增量快照进行时，Debebe 使用 watermarks 跟踪其进度，维护它捕获的每个表行的记录。与标准初始快照过程相比，捕获数据的阶段方法具有以下优点：

- 您可以使用流化数据捕获并行运行增量快照，而不是在快照完成前进行后流。连接器会在快照过程中从更改日志中捕获接近实时事件，且操作都不会阻止其他操作。
- 如果增量快照的进度中断，您可以在不丢失任何数据的情况下恢复它。在进程恢复后，快照从停止的点开始，而不是从开始计算表。
- 您可以随时根据需要运行增量快照，并根据需要重复该过程以适应数据库更新。例如，您可以在修改连接器配置后重新运行快照，以将表添加到其 table.include.list 属性中。

增量快照过程

当您运行增量快照时，Debezium 会按主键对每个表进行排序，然后根据 配置的块大小 将表分成块。然后，按块的工作块会捕获块中的每个表行。对于它捕获的每行，快照会发出 READ 事件。该事件代表块的快照开始时的行值。

当快照继续进行时，其他进程可能会继续访问数据库，可能会修改表记录。为了反映此类更改，INSERT、UPDATE 或 DELETE 操作会按照常常提交到事务日志。同样，持续 Debezium 流进程将继续检测这些更改事件，并将相应的更改事件记录发送到 Kafka。

Debezium 如何使用相同的主密钥在记录间解决冲突

在某些情况下，streaming 进程发出的 UPDATE 或 DELETE 事件会停止序列。也就是说，流流过程可能会发出一个修改表行的事件，该事件捕获包含该行的 READ 事件的块。当快照最终为行发出对应的 READ 事件时，其值已被替换。为确保以正确的逻辑顺序处理到达序列的增量快照事件，Debebe 使用缓

冲方案来解析冲突。仅在快照事件和流化事件之间发生冲突后，De Debezium 会将事件记录发送到 Kafka。

快照窗口

为了帮助解决修改同一表行的后期事件和流化事件之间的冲突，Debebe 会使用一个所谓的 快照窗口。快照窗口分解了增量快照捕获指定表块数据的间隔。在块的快照窗口打开前，Debebe 会使用其常见行为，并将事件从事务日志直接下游发送到目标 Kafka 主题。但从特定块的快照打开后，直到关闭为止，De-duplication 步骤会在具有相同主密钥的事件之间解决冲突。

对于每个数据收集，Debezium 会发出两种类型的事件，并将其存储在单个目标 Kafka 主题中。从表直接捕获的快照记录作为 READ 操作发送。同时，当用户继续更新数据收集中的记录，并且会更新事务日志来反映每个提交，Debezium 会为每个更改发出 UPDATE 或 DELETE 操作。

当快照窗口打开时，Debezium 开始处理快照块，它会向内存缓冲区提供快照记录。在快照窗口期间，缓冲区中 READ 事件的主密钥与传入流事件的主键进行比较。如果没有找到匹配项，则流化事件记录将直接发送到 Kafka。如果 Debezium 检测到匹配项，它会丢弃缓冲的 READ 事件，并将流化记录写入目标主题，因为流的事件逻辑地取代静态快照事件。在块关闭的快照窗口后，缓冲区仅包含 READ 事件，这些事件不存在相关的事务日志事件。Debezium 将这些剩余的 READ 事件发送到表的 Kafka 主题。

连接器为每个快照块重复这个过程。

6.1.6.1. 触发增量快照

目前，启动增量快照的唯一方法是向源数据库上的 信号表发送临时快照 信号。

作为 SQL INSERT 查询，您将向信号提交信号。

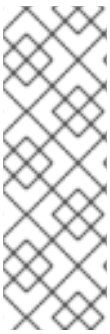
在 Debezium 检测到信号表中的更改后，它会读取信号并运行请求的快照操作。

您提交的查询指定要包含在快照中的表，并可以选择指定快照操作的类型。目前，快照操作的唯一有效选项是默认值 incremental。

要指定快照中包含的表，请提供列出表或用于匹配表的正则表达式数组的 数据集合，例如：

{"data-collections": ["public.MyFirstTable", "public.MySecondTable"]}

增量快照信号的 data-collections 数组没有默认值。如果 data-collections 数组为空，Debezium 会检测到不需要任何操作，且不会执行快照。



注意

如果要包含在快照中的表的名称在数据库、模式或表的名称中包含句点(.)，以将表添加到 data-collections 数组中，您必须使用双引号转义名称的每个部分。

例如，要包含一个存在于 公共 模式的表，其名称为 My.Table，请使用以下格式：**"public"."My.Table "**。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道来触发增量快照

1. 发送 SQL 查询，将临时增量快照请求添加到信号表中：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) 1
values ('ad-hoc-1',  2
  'execute-snapshot',  3
  '{"data-collections": ["schema1.table1", "schema2.table2"], 4
  "type":"incremental"}, 5
  "additional-condition":"color=blue"}'); 6
```

命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 6.3. SQL 命令中字段的描述，用于将增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。 使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。相反，Debebe 会在快照期间生成自己的 **id** 字符串作为水位线信号。
3	**execute-snapshot**	**type** 参数指定信号旨在触发的操作。
4	**data-collections**	信号的 **data** 字段所需的组件，用于指定表名称或正则表达式数组，以匹配快照中包含的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。
5	**incremental**	信号的 **data** 字段的可选 **类型** 组件，用于指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 如果没有指定值，连接器将运行增量快照。
6	**additional-condition**	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。有关 **additional-condition** 参数的更多信息，请参阅 **带有额外条件的临时增量快照**。

带有额外条件的临时增量快照

如果您希望快照只包含表中的内容子集，您可以通过向快照信号附加 additional-condition 参数来修改信号请求。

典型的快照的 SQL 查询采用以下格式：

```
SELECT * FROM <tableName> ....
```

通过添加 additional-condition 参数，您可以将 WHERE 条件附加到 SQL 查询中，如下例所示：

```
SELECT * FROM <tableName> WHERE <additional-condition> ....
```

以下示例显示了向信号表发送带有额外条件的临时增量快照请求的 SQL 查询：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，假设您有一个包含以下列的 products 表：

- ID （主键）
- color
- quantity

如果您需要 product 表的增量快照，其中只包含 color=blue 的数据项，您可以使用以下 SQL 语句来触发快照：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue"}');
```

additional-condition 参数还允许您传递基于多个列的条件。例如，使用上例中的 product 表，您可以提交查询来触发增量快照，该快照仅包含 color=blue 和 quantity>10 的项数据：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue AND quantity>10"}');
```

以下示例显示了连接器捕获的增量快照事件的 JSON。

示例：增加快照事件消息

```
{
  "before":null,
  "after": {
    "pk":"1",
    "value":"New data"
  },
  "source": {
    ...
    "snapshot":"incremental" 1
  },
  "op":"r", 2
  "ts_ms":"1620393591654",
  "transaction":null
}
```

项	字段名称	描述
1	**snapshot**	指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 在 SQL 查询中指定 **type** 值，您提交到信号表是可选的。 如果没有指定值，连接器将运行增量快照。
2	**op**	指定事件类型。 快照事件的值是 **r**，表示 **READ** 操作。

6.1.6.2. 使用 Kafka 信号频道来触发增量快照

您可以向 配置的 Kafka 主题 发送消息，以请求连接器来运行临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 execute-snapshot，data 字段必须具有以下字段：

表 6.4. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与快照中包含的表的完全限定域名匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的列子集的条件。

execute-snapshot Kafka 消息示例：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

带有额外条件的临时增量快照

Debezium 使用 additional-condition 字段来选择表内容的子集。

通常，当 Debezium 运行快照时，它会运行 SQL 查询，例如：

SELECT * FROM <tableName> ....

当快照请求包含 additional-condition 时，extra-condition 会附加到 SQL 查询中，例如：

SELECT * FROM <tableName> WHERE <additional-condition> ....

例如，如果一个 product table with the column id （主键）、color 和 brand，如果您希望快照只包含 color='blue' 的内容，当您请求快照时，您可以附加一个 additional-condition 语句来过滤内容：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type": "INCREMENTAL", "additional-condition":"color='blue'"}}`
```

您可以使用 additional-condition 语句根据多个列传递条件。例如，如果您希望快照只包含 color='blue' 的 products 表中，以及 brand='MyBrand'，则您可以发送以下请求：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}`
```

6.1.6.3. 停止增量快照

您还可以通过向源数据库上的表发送信号来停止增量快照。您可以通过发送 SQL INSERT 查询向表提交停止快照信号。

在 Debezium 检测到信号表中的更改后，它会读取信号，并在正在进行时停止增量快照操作。

您提交的查询指定 增量 的快照操作，以及要删除的当前运行快照的表。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道停止增量快照

1. 发送 SQL 查询以停止临时增量快照到信号表：

```
INSERT INTO <signalTable> (id, type, data) values ('<id>', 'stop-snapshot', '{"data-
collections": ["<tableName>","<tableName>"],"type":"incremental"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) 1
```

```
values ('ad-hoc-1', 2
  'stop-snapshot', 3
  '{"data-collections": ["schema1.table1", "schema2.table2"], 4
  "type":"incremental"}'); 5
```

signal 命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 6.5. SQL 命令中字段的描述，用于将停止增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。
3	**stop-snapshot**	指定 **type** 参数指定信号要触发的操作。
4	**data-collections**	信号的 **data** 字段的可选组件，用于指定表名称或正则表达式数组，以匹配要从快照中删除的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。如果省略了 **data** 字段的这一组件，信号将停止正在进行的整个增量快照。
5	**incremental**	信号的 **data** 字段所需的组件，用于指定要停止的快照操作类型。目前，唯一有效的选项是 **增量的**。 如果没有指定 **类型** 值，信号将无法停止增量快照。

6.1.6.4. 使用 Kafka 信号频道停止增量快照

您可以将信号消息发送到 配置的 Kafka 信号主题，以停止临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

*信号类型是 **stop-snapshot**，**data** 字段必须具有以下字段：*

表 6.6. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	可选数组，以逗号分隔的正则表达式，与表的完全限定域名匹配，以包含在快照中。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

*以下示例显示了典型的 **stop-snapshot Kafka** 信息：*

```
Key = `test_connector`

Value = `{"type":"stop-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"],
"type": "INCREMENTAL"}}`
```

6.1.7. 接收 Debezium MySQL 更改事件记录的默认 Kafka 主题名称

*默认情况下，**MySQL** 连接器会将表中的所有 **INSERT**、**UPDATE** 和 **DELETE** 操作的更改事件写入特定于该表的单一 **Apache Kafka** 主题。*

连接器使用以下惯例来命名更改事件主题：

topicPrefix.databaseName.tableName

*假设 **fulfillment** 是主题前缀，**inventory** 是数据库名称，数据库包含名为 **orders, customers,** 和 **products** 的表。**Debezium MySQL** 连接器将事件发送到三个 **Kafka** 主题，每个表对应一个数据库：*

```
fulfillment.inventory.orders
fulfillment.inventory.customers
fulfillment.inventory.products
```

以下列表为默认名称的组件提供定义：

topicPrefix

由 topic.prefix 连接器配置属性指定的主题前缀。

schemaName

操作所在的模式的名称。

tableName

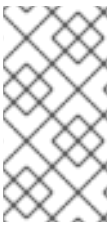
操作所在的表的名称。

连接器应用类似的命名约定，以标记其内部数据库架构历史记录主题、架构更改主题 和事务元数据主题。

如果默认主题名称不满足您的要求，您可以配置自定义主题名称。要配置自定义主题名称，您可以在逻辑主题路由 SMT 中指定正则表达式。有关使用逻辑主题路由 SMT 来自定义主题命名的更多信息，请参阅 主题路由。

事务元数据

Debezium 可以生成代表事务边界的事件，以及丰富的数据更改事件消息。



DEBEZIUM 接收事务元数据时的限制

Debezium 注册并只针对部署连接器后发生的事务接收元数据。部署连接器前发生的事务元数据不可用。

Debezium 为每个事务中的 BEGIN 和 END 分隔符生成事务边界事件。事务边界事件包含以下字段：

status

BEGIN 或 END.

id

唯一事务标识符的字符串。

ts_ms

数据源的事务边界事件(BEGIN 或 END 事件)的时间。如果数据源没有向事件时间提供

Debezium，则该字段代表 Debezium 处理事件的时间。

event_count （用于 END 事件）

事务发出的事件总数。

data_collections （用于 END 事件）

data_collection 和 event_count 元素的数组，用于指示连接器发出来自数据收集的更改的事件数量。

示例

```
{
  "status": "BEGIN",
  "id": "0e4d5dcd-a33b-11ea-80f1-02010a22a99e:10",
  "ts_ms": 1486500577125,
  "event_count": null,
  "data_collections": null
}

{
  "status": "END",
  "id": "0e4d5dcd-a33b-11ea-80f1-02010a22a99e:10",
  "ts_ms": 1486500577691,
  "event_count": 2,
  "data_collections": [
    {
      "data_collection": "s1.a",
      "event_count": 1
    },
    {
      "data_collection": "s2.a",
      "event_count": 1
    }
  ]
}
```

除非通过 topic.transaction 选项覆盖，否则连接器会将事务事件发送到 < topic.prefix>.transaction 主题。

更改数据事件增强

启用事务元数据后，数据消息 Envelope 通过新的 transaction 字段进行了增强。此字段以字段复合

的形式提供有关每个事件的信息：

id

唯一事务标识符的字符串。

total_order

事件在事务生成的所有事件中绝对位置。

data_collection_order

在事务发出的所有事件间，按数据收集位置。

以下是消息的示例：

```
{
  "before": null,
  "after": {
    "pk": "2",
    "aa": "1"
  },
  "source": {
...
  },
  "op": "c",
  "ts_ms": "1580390884335",
  "transaction": {
    "id": "0e4d5dcd-a33b-11ea-80f1-02010a22a99e:10",
    "total_order": "1",
    "data_collection_order": "1"
  }
}
```

对于没有启用 GTID 的系统，事务标识符是使用 binlog filename 和 binlog 位置的组合构建的。例如，如果与事务 BEGIN 事件对应的 binlog 文件名和位置分别是 mysql-bin.000002 和 1913，则 Debezium 构建的事务标识符为 file=mysql-bin.000002,pos=1913。

6.2. DEBEZIUM MYSQL 连接器数据更改事件的描述

Debezium MySQL 连接器为每个行级 INSERT、UPDATE 和 DELETE 操作生成数据更改事件。每个事件包含一个键和值。键的结构和值取决于已更改的表。

Debezium 和 Kafka Connect 围绕 事件消息的持续流 设计。但是，这些事件的结构可能会随时间推移

而改变，而用户很难处理这些事件。要解决这个问题，每个事件都包含其内容的 schema，或者如果您正在使用 schema registry，用户可以使用该模式 ID 从 registry 获取 schema。这使得每个事件都自包含。

以下框架 JSON 显示更改事件的基本四部分。但是，如何配置您选择在应用程序中使用的 Kafka Connect converter，决定更改事件中的这四个部分的表示。只有在将转换器配置为生成它时，schema 字段才会处于更改事件中。同样，只有在您配置转换器来生成它时，事件密钥和事件有效负载才会处于更改事件中。如果您使用 JSON 转换程序，并将其配置为生成所有四个基本更改事件部分，更改事件具有此结构：

```
{
 "schema": { 1
   ...
 },
 "payload": { 2
   ...
 },
 "schema": { 3
   ...
 },
 "payload": { 4
   ...
 },
}
```

表 6.7. 更改事件基本内容概述

项	字段名称	描述
1	**schema**	第一个 **schema** 字段是事件键的一部分。它指定一个 Kafka Connect 模式，用于描述事件键的 **payload** 部分的内容。换句话说，第一个 **schema** 字段描述了主密钥的结构，如果表没有主键，则描述主键的结构。 可以通过设置 **message.key.columns** 连接器配置属性 来覆盖表的主键。在这种情况下，第一个 schema 字段描述了该属性标识的键的结构。
2	**payload**	第一个 **payload** 字段是 event 键的一部分。它具有前面的**schema** 字段描述的结构，它包含已更改的行的密钥。
3	**schema**	第二个 **schema** 字段是事件值的一部分。它指定 Kafka Connect 模式，用于描述事件值 **有效负载部分的内容**。换句话说，第二个 **模式** 描述了已更改的行的结构。通常，此模式包含嵌套模式。
4	**payload**	第二个 **payload** 字段是事件值的一部分。它具有上一个 **schema** 字段描述的结构，它包含已更改的行的实际数据。

默认情况下，连接器流将事件记录改为与事件原始表相同的主题。请参阅 主题名称。

警告

MySQL 连接器确保所有 Kafka Connect 模式名称都遵循 Avro 模式名称格式。这意味着逻辑服务器名称必须以拉丁字母或下划线开头，即 a-z、A-Z 或 _。逻辑服务器名称和数据库和表名称中的每个字符都必须是拉丁字母、数字或下划线，即 a-z、A-Z、0-9 或 _。如果存在无效字符，它将使用下划线字符替换。

如果逻辑服务器名称、数据库名称或表名称包含无效字符，且唯一与另一个名称区分名称的字符无效，这可能会导致意外冲突冲突，从而被下划线替换。

详情包括在以下主题中：

- 第 6.2.1 节 “关于 Debezium MySQL 中的键更改事件”
- 第 6.2.2 节 “关于 Debezium MySQL 更改事件中的值”

6.2.1. 关于 Debezium MySQL 中的键更改事件

更改事件的密钥包含更改表的密钥和更改行的实际键的 schema。当连接器创建事件时，schema 及其对应有效负载都会包含更改表的 PRIMARY KEY （或唯一约束）中每个列的字段。

考虑以下 客户 表，后跟此表的更改事件键的示例。

```
CREATE TABLE customers (
  id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE KEY
) AUTO_INCREMENT=1001;
```

每次捕获 customer 表的更改事件都有相同的事件关键模式。只要 customers 表有以前的定义，可以捕获 customer 表更改的事件都有以下关键结构：在 JSON 中，它类似如下：

```
{
```

```
"schema": { 1
  "type": "struct",
  "name": "mysql-server-1.inventory.customers.Key", 2
  "optional": false, 3
  "fields": [ 4
    {
      "field": "id",
      "type": "int32",
      "optional": false
    }
  ]
},
"payload": { 5
  "id": 1001
}
}
```

表 6.8. 更改事件键的描述

项	字段名称	描述
1	**schema**	键的 schema 部分指定一个 Kafka Connect 模式，它描述了键的 **payload** 部分的内容。
2	**mysql-server-1.inventory.customers.Key**	定义密钥有效负载结构的模式名称。这个 schema 描述了已更改的表的主键的结构。键模式名称的格式是 *connector-name.database-name.table-name*.**Key**.在本例中： • **MySQL-server-1** 是生成此事件的连接器的名称。 • **inventory** 是包含已更改表的数据库。 • **客户** 是更新的表。
3	**optional**	指明 event 键是否必须在其 **payload** 字段中包含一个值。在本例中，键有效负载中的值是必需的。当表没有主键时，键的 payload 字段中的值是可选的。
4	**fields**	指定 **有效负载中** 预期的每个字段，包括每个字段的名称、类型以及是否需要。
5	**payload**	包含生成此更改事件的行的密钥。在本例中，键 包含一个 **id** 字段，其值为 **1001**。

6.2.2. 关于 Debezium MySQL 更改事件中的值

更改事件中的值比键复杂一些。与键一样，该值有一个 schema 部分和 payload 部分。schema 部分包含描述 payload 部分的 Envelope 结构的 schema，包括其嵌套字段。为创建、更新或删除数据的操作更改事件，它们都有一个带有 envelope 结构的值有效负载。

考虑用于显示更改事件键示例的相同示例表：

```
CREATE TABLE customers (
  id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE KEY
) AUTO_INCREMENT=1001;
```

对这个表的更改事件的值部分描述：

- 创建 事件
- 更新 事件
- 主密钥更新
- 删除 事件
- tombstone 事件
- Truncate 事件

创建 事件

以下示例显示了一个更改事件的值部分，连接器为在 customer 表中创建数据的操作生成的更改事件的值部分：

```
{
  "schema": { 1
    "type": "struct",
    "fields": [
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
```

```
        "field": "id"
      },
      {
        "type": "string",
        "optional": false,
        "field": "first_name"
      },
      {
        "type": "string",
        "optional": false,
        "field": "last_name"
      },
      {
        "type": "string",
        "optional": false,
        "field": "email"
      }
    ],
    "optional": true,
    "name": "mysql-server-1.inventory.customers.Value", 2
    "field": "before"
  },
  {
    "type": "struct",
    "fields": [
      {
        "type": "int32",
        "optional": false,
        "field": "id"
      },
      {
        "type": "string",
        "optional": false,
        "field": "first_name"
      },
      {
        "type": "string",
        "optional": false,
        "field": "last_name"
      },
      {
        "type": "string",
        "optional": false,
        "field": "email"
      }
    ],
    "optional": true,
    "name": "mysql-server-1.inventory.customers.Value",
    "field": "after"
  },
  {
    "type": "struct",
    "fields": [
      {
        "type": "string",
        "optional": false,
```

```
      "field": "version"
    },
    {
      "type": "string",
      "optional": false,
      "field": "connector"
    },
    {
      "type": "string",
      "optional": false,
      "field": "name"
    },
    {
      "type": "int64",
      "optional": false,
      "field": "ts_ms"
    },
    {
      "type": "boolean",
      "optional": true,
      "default": false,
      "field": "snapshot"
    },
    {
      "type": "string",
      "optional": false,
      "field": "db"
    },
    {
      "type": "string",
      "optional": true,
      "field": "table"
    },
    {
      "type": "int64",
      "optional": false,
      "field": "server_id"
    },
    {
      "type": "string",
      "optional": true,
      "field": "gtid"
    },
    {
      "type": "string",
      "optional": false,
      "field": "file"
    },
    {
      "type": "int64",
      "optional": false,
      "field": "pos"
    },
    {
      "type": "int32",
      "optional": false,
```

```
        "field": "row"
      },
      {
        "type": "int64",
        "optional": true,
        "field": "thread"
      },
      {
        "type": "string",
        "optional": true,
        "field": "query"
      }
    ],
    "optional": false,
    "name": "io.debezium.connector.mysql.Source", 3
    "field": "source"
  },
  {
    "type": "string",
    "optional": false,
    "field": "op"
  },
  {
    "type": "int64",
    "optional": true,
    "field": "ts_ms"
  }
 ],
 "optional": false,
 "name": "mysql-server-1.inventory.customers.Envelope" 4
},
"payload": { 5
 "op": "c", 6
 "ts_ms": 1465491411815, 7
 "before": null, 8
 "after": { 9
  "id": 1004,
  "first_name": "Anne",
  "last_name": "Kretchmar",
  "email": "annek@noanswer.org"
 },
 "source": { 10
  "version": "2.3.4.Final",
  "connector": "mysql",
  "name": "mysql-server-1",
  "ts_ms": 0,
  "snapshot": false,
  "db": "inventory",
  "table": "customers",
  "server_id": 0,
  "gtid": null,
  "file": "mysql-bin.000003",
  "pos": 154,
  "row": 0,
  "thread": 7,
```

```
    "query": "INSERT INTO customers (first_name, last_name, email) VALUES ('Anne',
'Kretchmar', 'annek@noanswer.org')"
  }
 }
}
```

表 6.9. 创建 事件值字段的描述

项	字段名称	描述
1	**schema**	值的 schema，用于描述值有效负载的结构。当连接器为特定表生成的每次更改事件中，更改事件的值模式都是相同的。
2	**name**	在 **schema** 部分中，每个 **name** 字段指定值的有效负载中字段的 schema。 **mysql-server-1.inventory.customers.Value** 是有效负载 **before** 和 **after** 字段的 schema。这个模式特定于 **customers** 表。 **before** 和 **after** 字段的模式名称格式为 ***logicalName.tableName*.Value**，这样可确保 schema 名称在数据库中是唯一的。这意味着，在使用 Avro converter 时，每个逻辑源中的每个表生成的 Avro 模式都有自己的 evolution 和 history。
3	**name**	**io.debezium.connector.mysql.Source** 是有效负载的 **source** 字段的 schema。这个模式特定于 MySQL 连接器。连接器将其用于它生成的所有事件。
4	**name**	**mysql-server-1.inventory.customers.Envelope** 是负载总体结构的模式，其中 **dbserver1** 是连接器名称，**inventory** 是数据库，**customers** 是表。
5	**payload**	值的实际数据。这是更改事件提供的信息。 可能会出现事件的 JSON 表示比它们描述的行大得多。这是因为 JSON 表示必须包含消息的 schema 和 payload 部分。但是，通过使用 Avro converter，您可以显著减少连接器流到 Kafka 主题的信息大小。
6	**op**	描述导致连接器生成事件的操作类型的强制字符串。在本例中，**c** 表示操作创建了行。有效值为： • **c** = create • **u** = update • **d** = delete • **r** = 读取（仅适用于快照）

项	字段名称	描述
7	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
8	**before**	指定事件发生前行状态的可选字段。当 **op** 字段是 **c** 用于创建（如本例所示），**before** 字段为 **null**，因为此更改事件用于新内容。
9	**after**	指定事件发生后行状态的可选字段。在本例中，**after** 字段包含新行的 **id**、**first_name**、**last_name** 和 **email** 列的值。
10	**source**	描述事件源元数据的必需字段。此字段包含可用于将此事件与其他事件进行比较的信息，以及事件的来源、事件发生的顺序以及事件是否为同一事务的一部分。源元数据包括： ● Debezium 版本 ● 连接器名称 ● 记录事件的 binlog 名称 ● binlog 位置 ● 事件中的行 ● 如果事件是快照的一部分 ● 包含新行的数据库和表的名称 ● 创建事件的 MySQL 线程的 ID （仅限非快照） ● MySQL 服务器 ID （如果可用） ● 在数据库中进行更改时的时间戳 如果启用了 **binlog_rows_query_log_events** MySQL 配置选项，并且启用了连接器配置 **include.query** 属性，**source** 字段还提供 **query** 字段，其中包含导致更改事件的原始 SQL 语句。

更新 事件

示例 customers 表中一个更新的改变事件的值有与那个表的 create 事件相同的模式。同样，事件值有效负载具有相同的结构。但是，事件值有效负载在 update 事件中包含不同的值。以下是连接器为 customer 表中更新生成的更改事件值 的示例 ：

```
{
  "schema": { ... },
  "payload": {
    "before": { 1
      "id": 1004,
```

```
    "first_name": "Anne",
    "last_name": "Kretchmar",
    "email": "annek@noanswer.org"
   },
   "after": { 2
    "id": 1004,
    "first_name": "Anne Marie",
    "last_name": "Kretchmar",
    "email": "annek@noanswer.org"
   },
   "source": { 3
    "version": "2.3.4.Final",
    "name": "mysql-server-1",
    "connector": "mysql",
    "name": "mysql-server-1",
    "ts_ms": 1465581029100,
    "snapshot": false,
    "db": "inventory",
    "table": "customers",
    "server_id": 223344,
    "gtid": null,
    "file": "mysql-bin.000003",
    "pos": 484,
    "row": 0,
    "thread": 7,
    "query": "UPDATE customers SET first_name='Anne Marie' WHERE id=1004"
   },
   "op": "u", 4
   "ts_ms": 1465581029523 5
  }
 }
```

表 6.10. 更新 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在 *update* 事件值中，**before** 字段包含每个表列的字段，以及数据库提交前该列中的值。在本例中，**first_name** 值为 **Anne**。
2	**after**	指定事件发生后行状态的可选字段。您可以比较 **before** 和 **after** 结构，以确定这个行的更新是什么。在示例中，**first_name** 值现在是 **Anne Marie**。

项	字段名称	描述
3	**source**	描述事件源元数据的必需字段。**source** 字段结构与 *create* 事件中的字段相同，但一些值有所不同，例如，示例 *update* 事件来自 binlog 中的不同位置。源元数据包括： • Debezium 版本 • 连接器名称 • 记录事件的 binlog 名称 • binlog 位置 • 事件中的行 • 如果事件是快照的一部分 • 包含更新行的数据库和表的名称 • 创建事件的 MySQL 线程的 ID （仅限非快照） • MySQL 服务器 ID （如果可用） • 在数据库中进行更改时的时间戳 如果启用了 **binlog_rows_query_log_events** MySQL 配置选项，并且启用了连接器配置 **include.query** 属性，**source** 字段还提供 **query** 字段，其中包含导致更改事件的原始 SQL 语句。
4	**op**	描述操作类型的强制字符串。在 *update* 事件值中，**op** 字段值为 **u**，表示此行因为更新而改变。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

注意

更新行 primary/unique 键的列会更改行的键值。当键更改时，Debezium 会输出 三个事件：一个 DELETE 事件，以及一个带有行的旧键的 tombstone 事件，后跟一个带有行的新键的事件。详情在下一节中。

主密钥更新

更改行的主键字段的 UPDATE 操作称为主密钥更改。对于主键更改，以代替 UPDATE 事件记录，连接器为旧密钥发出 DELETE 事件记录，并为新的(updated)密钥的 CREATE 事件记录。这些事件具有常见的结构和内容，另外，每个事件都有与主密钥更改相关的消息标头：

•

DELETE 事件记录具有 __debezium.newkey 作为消息标头。此标头的值是更新行的新主键。

- *CREATE 事件记录具有 __debezium.oldkey 作为消息标头。此标头的值是更新行所具有的前一个主键（旧的）主键。*

删除 事件

delete 更改事件中的值与为同一表的 create 和 update 事件相同的 schema 部分。示例 customer 表的 delete 事件中 payload 部分类似如下：

```
{
  "schema": { ... },
  "payload": {
    "before": { 1
      "id": 1004,
      "first_name": "Anne Marie",
      "last_name": "Kretchmar",
      "email": "annek@noanswer.org"
    },
    "after": null, 2
    "source": { 3
      "version": "2.3.4.Final",
      "connector": "mysql",
      "name": "mysql-server-1",
      "ts_ms": 1465581902300,
      "snapshot": false,
      "db": "inventory",
      "table": "customers",
      "server_id": 223344,
      "gtid": null,
      "file": "mysql-bin.000003",
      "pos": 805,
      "row": 0,
      "thread": 7,
      "query": "DELETE FROM customers WHERE id=1004"
    },
    "op": "d", 4
    "ts_ms": 1465581902461 5
  }
}
```

表 6.11. 删除 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在一个 *delete* 事件值中，**before** 字段包含在使用数据库提交删除行前的值。

项	字段名称	描述
2	**after**	指定事件发生后行状态的可选字段。在 *delete* 事件值中，**after** 字段为 **null**，表示行不再存在。
3	**source**	描述事件源元数据的必需字段。在一个 *delete* 事件值中，**source** 字段结构与同一表的 *create* 和 *update* 事件相同。许多 **source** 字段值也相同。在 *delete* 事件值中，**ts_ms** 和 **pos** 字段值以及其他值可能已更改。但是 *delete* 事件值中的 **source** 字段提供相同的元数据： ● Debezium 版本 ● 连接器名称 ● 记录事件的 binlog 名称 ● binlog 位置 ● 事件中的行 ● 如果事件是快照的一部分 ● 包含更新行的数据库和表的名称 ● 创建事件的 MySQL 线程的 ID （仅限非快照） ● MySQL 服务器 ID （如果可用） ● 在数据库中进行更改时的时间戳 如果启用了 **binlog_rows_query_log_events** MySQL 配置选项，并且启用了连接器配置 **include.query** 属性，**source** 字段还提供 **query** 字段，其中包含导致更改事件的原始 SQL 语句。
4	**op**	描述操作类型的强制字符串。**op** 字段值为 **d**，表示此行已被删除。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

删除 更改事件记录为消费者提供处理此行删除所需的信息。包含旧值，因为有些用户可能需要它们才能正确处理删除。

MySQL 连接器事件被设计为使用 Kafka 日志压缩。只要保留每个密钥的最新消息，日志压缩就会启用删除一些旧的消息。这可让 Kafka 回收存储空间，同时确保主题包含完整的数据集，并可用于重新载入基于密钥的状态。

tombstone 事件

删除行时，delete 事件值仍可用于日志压缩，因为 Kafka 您可以删除具有相同键的所有之前信息。但是，要让 Kafka 删除具有相同键的所有消息，消息值必须为 null。为了实现此目的，在 Debezium 的 MySQL 连接器发出 一个 delete 事件后，连接器会发出一个特殊的 tombstone 事件，它具有相同的键但有一个 null 值。

Truncate 事件

截断 更改事件信号，表示表已被截断。在这种情况下，message 键为 null，message 值类似如下：

```
{
  "schema": { ... },
  "payload": {
    "source": { 1
      "version": "2.3.4.Final",
      "name": "mysql-server-1",
      "connector": "mysql",
      "name": "mysql-server-1",
      "ts_ms": 1465581029100,
      "snapshot": false,
      "db": "inventory",
      "table": "customers",
      "server_id": 223344,
      "gtid": null,
      "file": "mysql-bin.000003",
      "pos": 484,
      "row": 0,
      "thread": 7,
      "query": "UPDATE customers SET first_name='Anne Marie' WHERE id=1004"
    },
    "op": "t", 2
    "ts_ms": 1465581029523 3
  }
}
```

表 6.12. truncate 事件值字段的描述

项	字段名称	描述

项	字段名称	描述
1	**source**	描述事件源元数据的必需字段。在 *truncate* 事件值中，**source** 字段结构与为同一表的 *create*, *update*, 和 *delete* 事件相同，提供此元数据： ● Debezium 版本 ● 连接器类型和名称 ● 记录事件的 binlog 名称 ● binlog 位置 ● 事件中的行 ● 如果事件是快照的一部分 ● 数据库和表的名称 ● 截断事件的 MySQL 线程的 ID （仅限非快照） ● MySQL 服务器 ID （如果可用） ● 在数据库中进行更改时的时间戳
2	**op**	描述操作类型的强制字符串。**op** 字段值为 **t**，表示此表已被截断。
3	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 + **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

如果单个 TRUNCATE 语句应用到多个表，则会为每个删节的表发出一个 truncate 更改事件记录。

请注意，由于 truncate 事件代表对整个表进行的更改，且没有消息密钥，除非您使用单个分区的主题，否则与表相关的更改事件没有排序保证(创建、更新 等)和 截断 该表的事件。例如，当这些事件从不同的分区读取时，消费者只能在该表的 truncate 事件后收到 update 事件。

6.3. DEBEZIUM MYSQL 连接器如何映射数据类型

Debezium MySQL 连接器代表对行的更改，这些事件的结构与行存在的表类似。事件包含每个列值的一个字段。该列的 MySQL 数据类型指定 Debezium 如何代表事件中的值。

存储字符串的列在 MySQL 中定义，并带有字符集和合并。当读取 binlog 事件中列值的二进制表示时，MySQL 连接器使用列的字符集。

连接器可以将 MySQL 数据类型映射到 literal 和 semantic 类型。

- 字面类型 ：值如何使用 Kafka Connect 模式类型表示。
- 语义类型 ： Kafka Connect 模式如何捕获字段的含义(schema 名称)。

如果默认数据类型转换不满足您的需要，您可以为连接器 创建自定义转换器。

以下部分详情：

- 基本类型
- 时序类型
- 十进制类型
- 布尔值
- 空间类型

基本类型

下表显示了连接器如何映射基本 MySQL 数据类型。

表 6.13. 基本类型映射的描述

MySQL 类型	字面类型	语义类型
BOOLEAN, BOOL	布尔值	*不适用*
BIT(1)	布尔值	*不适用*

MySQL 类型	字面类型	语义类型
BIT(>1)	**BYTES**	**io.debezium.data.Bits** 。**length** 模式参数包含一个代表位数的整数。**byte[]** 包含 *little-endian* 表单中的位，使用 sized 来包含指定数量的位。例如，其中 **n** 为 bit: **numBytes = n/8 +（n%8== 0 ?0 : 1)**
TINYINT	**INT16**	*不适用*
SMALLINT[(M)]	**INT16**	*不适用*
MEDIUMINT[(M)]	**INT32**	*不适用*
INT, INTEGER[(M)]	**INT32**	*不适用*
BIGINT[(M)]	**INT64**	*不适用*
REAL[(M,D)]	**FLOAT32**	*不适用*
FLOAT[(P)]	**FLOAT32** 或 **FLOAT64**	精度仅用于确定存储大小。精度 **P** 从 0 到 23 会导致 4 字节单precision **FLOAT32** 列。从 24 到 53 的精度 **P** 会产生 8 字节双precision **FLOAT64** 列。
FLOAT (M,D)	**FLOAT64**	从 MySQL 8.0.17 开始，非标准 FLOAT (M,D)和 DOUBLE (M,D)语法已弃用，并预期在以后的 MySQL 版本中删除对它的支持，将 **FLOAT64** 设置为默认值。
DOUBLE[(M,D)]	**FLOAT64**	*不适用*
CHAR(M)]	**字符串**	*不适用*
VARCHAR(M)]	**字符串**	*不适用*
BINARY (M)]	**BYTES** 或 **STRING**	*N/A* 根据 **binary.handling.mode** 连接器配置属性设置，一个原始字节（默认）、base64 编码的 String 或 base64-url-safe-encoded String 或十六进制编码的字符串。
VARBINARY (M)]	**BYTES** 或 **STRING**	*N/A* 根据 **binary.handling.mode** 连接器配置属性设置，一个原始字节（默认）、base64 编码的 String 或 base64-url-safe-encoded String 或十六进制编码的字符串。

MySQL 类型	字面类型	语义类型
TINYBLOB	**BYTES** 或 **STRING**	*N/A* 根据 **binary.handling.mode** 连接器配置属性设置，一个原始字节（默认）、base64 编码的 String 或 base64-url-safe-encoded String 或十六进制编码的字符串。
TINYTEXT	**字符串**	*不适用*
BLOB	**BYTES** 或 **STRING**	*N/A* 根据 **binary.handling.mode** 连接器配置属性设置，一个原始字节（默认）、base64 编码的 String 或 base64-url-safe-encoded String 或十六进制编码的字符串。 仅支持大小为 2GB 的值。建议您使用声明检查模式外部化大列值。
TEXT	**字符串**	*N/A* 只支持大小为 2GB 的值。建议您使用声明检查模式外部化大列值。
MIUMBLOB	**BYTES** 或 **STRING**	*N/A* 根据 **binary.handling.mode** 连接器配置属性设置，一个原始字节（默认）、base64 编码的 String 或 base64-url-safe-encoded String 或十六进制编码的字符串。
中型	**字符串**	*不适用*
LONGBLOB	**BYTES** 或 **STRING**	*N/A* 根据 **binary.handling.mode** 连接器配置属性设置，一个原始字节（默认）、base64 编码的 String 或 base64-url-safe-encoded String 或十六进制编码的字符串。 仅支持大小为 2GB 的值。建议您使用声明检查模式外部化大列值。
LONGTEXT	**字符串**	*N/A* 只支持大小为 2GB 的值。建议您使用声明检查模式外部化大列值。
JSON	**字符串**	**io.debezium.data.Json** 包含 **JSON** 文档、数组或 scalar 的字符串表示。
ENUM	**字符串**	**io.debezium.data.Enum** **允许的** schema 参数包含以逗号分隔的值列表。
SET	**字符串**	**io.debezium.data.EnumSet** **允许的** schema 参数包含以逗号分隔的值列表。
YEAR[(2|4)]	**INT32**	**io.debezium.time.Year**

MySQL 类型	字面类型	语义类型
TIMESTAMP[(M)]	**字符串**	**io.debezium.time.ZonedTimestamp** in ISO 8601 格式带有 microsecond 精度。MySQL 允许**M**在 **0-6** 范围内。

时序类型

排除 TIMESTAMP 数据类型，MySQL temporal 类型取决于 time.precision.mode 连接器配置属性的值。对于 TIMESTAMP 列，它的默认值被指定为 CURRENT_TIMESTAMP 或 NOW，值 1970-01-01 00:00:00 被用于 Kafka Connect schema 中的默认值。

MySQL 允许 DATE, DATETIME, 和 TIMESTAMP 为零值，因为在有些情况下首先使用零值而不是 null 值。当列定义允许 null 值或当列不允许 null 值时，MySQL 连接器将零值表示为 null 值。

没有时区的临时值

DATETIME 类型代表一个本地日期和时间，如 "2018-01-13 09:48:27"。正如您所见，没有时区信息。这些列会根据使用 UTC 的精度，将此类列转换为 epoch 毫秒或微秒。TIMESTAMP 类型代表一个没有时区信息的时间戳。当读取后，通过 MySQL 将 MySQL 从服务器（或会话的）当前时区转换为 UTC，从 UTC 写入服务器（或会话）当前时区。例如：

- *DATETIME 的值 2018-06-20 06:37:03 变为 1529476623000。*
- *TIMESTAMP 的值 2018-06-20 06:37:03 变为 2018-06-20T13:37:03Z。*

此类列转换为根据服务器（或会话）当前时区的 UTC 中等效的 io.debezium.time.ZonedTimestamp。默认情况下，时区将从服务器查询。如果此操作失败，则必须由 database connectionTimeZone MySQL 配置选项明确指定。例如，如果数据库的时区（通过 connectionTimeZone 选项全局配置或为连接器配置）是 "America/Los_Angeles"，TIMESTAMP 值 "2018-06-20 06:37:03" 由 ZonedTimestamp 代表，值为 "2018-06-20T13:37:03Z"。

运行 Kafka Connect 和 Debezium 的 JVM 的时区不会影响这些转换。

有关 temporal 值的属性的更多详细信息，请参见 MySQL 连接器配置属性 的文档。

time.precision.mode=adaptive_time_microseconds(default)

MySQL 连接器根据列的数据类型定义确定字面类型和语义类型，以便事件准确代表数据库中的

值。所有时间字段都以微秒为单位。只有 00:00:00.000000 到 23:59:59.999999 范围内的正 TIME 字段值可以正确捕获。

表 6.14. 当 time.precision.mode=adaptive_time_microseconds时映射

MySQL 类型	字面类型	语义类型
DATE	**INT32**	**io.debezium.time.Date** 代表自 epoch 后的天数。
TIME[(M)]	**INT64**	**io.debezium.time.MicroTime** 代表微秒中的时间值，不包括时区信息。MySQL 允许 **M** 在 **0-6** 范围内。
DATETIME, **DATETIME (0),** **DATETIME (1),** **DATETIME (2),** **DATETIME (3)**	**INT64**	**io.debezium.time.Timestamp** 代表 epoch 过的毫秒数，不包括时区信息。
DATETIME (4), **DATETIME (5),** **DATETIME (6)**	**INT64**	**io.debezium.time.MicroTimestamp** 代表 epoch 过的微秒数，不包括时区信息。

time.precision.mode=connect

MySQL 连接器使用定义的 Kafka Connect 逻辑类型。这个方法比默认方法更精确，如果数据库列的 fractional second 精度值大于 3，则事件可能会更精确。只能处理 00:00:00.000 到 23:59 .999 范围内的值。仅在确保表中的 TIME 值不能超过支持的范围时，设置 time.precision.mode=connect。预计会在 Debezium 的未来版本中删除 connect 设置。

表 6.15. time.precision.mode=connect时映射

MySQL 类型	字面类型	语义类型
DATE	**INT32**	**org.apache.kafka.connect.data.Date** 代表自 epoch 后的天数。
TIME[(M)]	**INT64**	**org.apache.kafka.connect.data.Time** 代表自午夜起的微秒中的时间值，不包括时区信息。
DATETIME[(M)]	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 epoch 后的毫秒数，不包括时区信息。

十进制类型

Debezium 连接器根据 decimal.handling.mode 连接器配置属性的 设置处理十进制。

decimal.handling.mode=precise

表 6.16. 当 decimal.handling.mode=precise时映射

MySQL 类型	字面类型	语义类型
NUMERIC[(M[,D])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** 模式参数包括一个整数，它代表了十进制小数点移动了多少位。
DECIMAL[(M[,D])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** 模式参数包括一个整数，它代表了十进制小数点移动了多少位。

decimal.handling.mode=double

表 6.17. 当 decimal.handling.mode=double时映射

MySQL 类型	字面类型	语义类型
NUMERIC[(M[,D])]	**FLOAT64**	*不适用*
DECIMAL[(M[,D])]	**FLOAT64**	*不适用*

decimal.handling.mode=string

表 6.18. 当 decimal.handling.mode=string时映射

MySQL 类型	字面类型	语义类型
NUMERIC[(M[,D])]	**字符串**	*不适用*
DECIMAL[(M[,D])]	**字符串**	*不适用*

布尔值

MySQL 以特定的方式在内部处理 BOOLEAN 值。BOOLEAN 列内部映射到 TINYINT (1) 数据类型。在流期间创建表时，它会使用正确的 BOOLEAN 映射，因为 Debezium 接收原始 DDL。在快照期间，Debezium 执行 SHOW CREATE TABLE 来获取表定义，该定义为 BOOLEAN 和 TINYINT (1) 列返回 TINYINT (1) 列。然后，Debezium 无法获取原始类型映射，因此映射到 TINYINT (1)。

为了允许您将源列转换为布尔值数据类型，Dein yIntOneToBooleanConverter 自定义转换器，您可以使用以下方法之一使用：

- 将所有 TINYINT (1) 或 TINYINT (1) UNSIGNED 列映射到 BOOLEAN 类型。

- 使用以逗号分隔的正则表达式列表枚举列的子集。
要使用这种类型的转换，您必须使用 selector 参数设置 converters 配置属性，如下例所示：

```
converters=boolean
boolean.type=io.debezium.connector.mysql.converters.TinyIntOneToBooleanConverter
boolean.selector=db1.table1.*, db1.table2.column1
```

- 注意：当快照执行 SHOW CREATE TABLE 时，MySQL8 不会显示 tinyint 未签名类型的长度，这意味着此转换器不起作用。新选项 length.checker 可以解决这个问题，默认值为 true。禁用 length.checker 并指定需要转换为 selector 属性的列，而不是根据类型转换所有列，如下例所示：

```
converters=boolean
boolean.type=io.debezium.connector.mysql.converters.TinyIntOneToBooleanConverter
boolean.length.checker=false
boolean.selector=db1.table1.*, db1.table2.column1
```

空间类型

目前，Debezium MySQL 连接器支持以下空间数据类型：

表 6.19. 空间类型映射的描述

MySQL 类型	字面类型	语义类型
GEOMETRY, 行STRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION	STRUCT	io.debezium.data.geometry.Geometry 包含有两个字段的结构： • Srid （INT32: spatial reference system ID，用于定义存储在结构中的 geometry 对象的类型 • wkb (BYTES) ：以 Well-Known-Binary (wkb)格式编码的 geometry 对象的二进制表示。如需了解更多详细信息，请参阅 Open Geospatial Consortium。

6.4. 设置 MYSQL 以运行 DEBEZIUM 连接器

在安装和运行 Debezium 连接器前，需要一些 MySQL 设置任务。

以下部分详情：

- 第 6.4.1 节 “为 Debezium 连接器创建 MySQL 用户”
- 第 6.4.2 节 “为 Debezium 启用 MySQL binlog”
- 第 6.4.3 节 “为 Debezium 启用 MySQL 全局事务标识符”
- 第 6.4.4 节 “为 Debezium 配置 MySQL 会话超时”
- 第 6.4.5 节 “为 Debezium MySQL 连接器启用查询日志事件”

6.4.1. 为 Debezium 连接器创建 MySQL 用户

Debezium MySQL 连接器需要一个 MySQL 用户帐户。此 MySQL 用户必须具有 Debezium MySQL 连接器捕获更改的所有数据库的适当权限。

先决条件

- MySQL 服务器。
- SQL 命令的基本知识.

流程

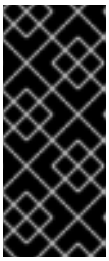
1. 创建 MySQL 用户：

```
mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';
```

2. 授予用户所需的权限：

```
mysql> GRANT SELECT, RELOAD, SHOW DATABASES, REPLICATION SLAVE,
REPLICATION CLIENT ON *.* TO 'user' IDENTIFIED BY 'password';
```

下表描述了权限。



重要

如果使用托管的选项，如 Amazon RDS 或 Amazon Aurora，则不允许全局读取锁定，则使用表级锁定来创建 一致的快照。在这种情况下，您还需要为您创建的用户授予 LOCK TABLES 权限。如需了解更多详细信息，请参阅 快照。

3. *完成用户的权限：*

```
mysql> FLUSH PRIVILEGES;
```

表 6.20. 用户权限的描述

关键字	描述
选择	启用连接器从数据库中的表中选择行。这仅在执行快照时使用。
RELOAD	启用连接器，使用 **FLUSH** 语句清除或重新载入内部缓存、清除表或获取锁定。这仅在执行快照时使用。
显示数据库	通过发出 **SHOW DATABASE** 语句来启用连接器查看数据库名称。这仅在执行快照时使用。
复制从	启用连接器来连接和读取 MySQL 服务器 binlog。
复制客户端	启用连接器使用以下语句： • **显示 MASTER 状态** • **显示从状态** • **SHOW BINARY LOGS** 连接器始终需要此操作。
ON	标识权限应用到的数据库。
TO 'user'	指定要向授予权限的用户。
IDENTIFIED BY 'password'	指定用户的 MySQL 密码。

6.4.2. 为 Debezium 启用 MySQL binlog

您必须为 MySQL 复制启用二进制日志记录。二进制日志记录复制工具的事务更新以传播更改。

先决条件

- *MySQL 服务器。*
- *适当的 MySQL 用户特权。*

流程

1. *检查 log-bin 选项是否已启用：*

```
// for MySql 5.x
mysql> SELECT variable_value as "BINARY LOGGING STATUS (log-bin) ::"
FROM information_schema.global_variables WHERE variable_name='log_bin';
// for MySql 8.x
mysql> SELECT variable_value as "BINARY LOGGING STATUS (log-bin) ::"
FROM performance_schema.global_variables WHERE variable_name='log_bin';
```

2. *如果是 OFF，请使用以下属性配置 MySQL 服务器配置文件，下表中描述如下：*

```
server-id        = 223344 # Querying variable is called server_id, e.g. SELECT
variable_value FROM information_schema.global_variables WHERE
variable_name='server_id';
log_bin          = mysql-bin
binlog_format    = ROW
binlog_row_image  = FULL
expire_logs_days  = 10
```

3. *通过检查 binlog 状态一次确认您的更改：*

```
// for MySql 5.x
mysql> SELECT variable_value as "BINARY LOGGING STATUS (log-bin) ::"
FROM information_schema.global_variables WHERE variable_name='log_bin';
// for MySql 8.x
mysql> SELECT variable_value as "BINARY LOGGING STATUS (log-bin) ::"
FROM performance_schema.global_variables WHERE variable_name='log_bin';
```

表 6.21. MySQL binlog 配置属性的描述

属性	描述
server-id	**server-id** 的值对于 MySQL 集群中的每台服务器和复制客户端必须是唯一的。在设置 MySQL 连接器的过程中，Debezium 为连接器分配唯一的服务器 ID。
log_bin	**log_bin** 的值是 binlog 文件序列的基本名称。
binlog_format	**binlog-format** 必须设置为 ROW **或 row**。
binlog_row_image	**binlog_row_image** 必须设置为 **FULL** 或 **full**。
expire_logs_days	这是自动删除 binlog 文件的天数。默认值为 **0,** 这意味着没有自动删除。将值设为与您的环境的需求匹配。请参阅 MySQL 清除 binlog 文件。

6.4.3. 为 Debezium 启用 MySQL 全局事务标识符

全局事务标识符(GTID)唯一地标识集群中服务器上的事务。虽然 Debezium MySQL 连接器不需要，但使用 GTID 简化了复制，并可让您更轻松地确认主和副本服务器是否一致。

在 MySQL 5.6.5 及更高版本中提供 GTID。详情请查看 MySQL 文档。

先决条件

- *MySQL 服务器。*
- *SQL 命令的基本知识.*
- *访问 MySQL 配置文件。*

流程

1. *启用 gtid_mode ：*

```
mysql> gtid_mode=ON
```

2. *启用 enforce_gtid_consistency：*

-

```
mysql> enforce_gtid_consistency=ON
```

3. *确认更改：*

```
mysql> show global variables like '%GTID%';
```

结果

```
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| enforce_gtid_consistency | ON    |
| gtid_mode                | ON    |
+--------------------------+-------+
```

表 6.22. GTID 选项的描述

选项	描述
gtid_mode	指定是否启用 MySQL 服务器的 GTID 模式的布尔值。 • **ON** = enabled • **OFF** = disabled
enforce_gtid_consistency	布尔值，指定服务器是否强制实施 GTID 一致性，方法是允许执行可以事务安全的方式登录的声明。使用 GTID 时需要。 • **ON** = enabled • **OFF** = disabled

6.4.4. 为 Debezium 配置 MySQL 会话超时

当为大型数据库制作初始一致的快照时，您在读取表时您的建立的连接可能会超时。您可以通过在 MySQL 配置文件中配置 interactive_timeout 和 wait_timeout 来防止此行为。

先决条件

- *MySQL 服务器。*
- *SQL 命令的基本知识.*
- *访问 MySQL 配置文件。*

流程

1. *配置 interactive_timeout ：*

```
mysql> interactive_timeout=<duration-in-seconds>
```

2. *配置 wait_timeout ：*

```
mysql> wait_timeout=<duration-in-seconds>
```

表 6.23. MySQL 会话超时选项的描述

选项	描述
interactive_timeout	服务器在互动连接中等待活动的秒数，然后再关闭它。如需了解更多详细信息，请参阅 MySQL 的文档。
wait_timeout	服务器在关闭前在非互动连接中等待活动的秒数。如需了解更多详细信息，请参阅 MySQL 的文档。

6.4.5. 为 Debezium MySQL 连接器启用查询日志事件

您可能希望查看每个 binlog 事件的原始 SQL 语句。您可以在 MySQL 配置文件中启用 binlog_rows_query_log_events 选项，您可以执行此操作。

MySQL 5.6 及更高版本中提供了这个选项。

先决条件

- *MySQL 服务器。*

- SQL 命令的基本知识.
- 访问 MySQL 配置文件。

流程

- 启用 binlog_rows_query_log_events ：

```
mysql> binlog_rows_query_log_events=ON
```

binlog_rows_query_log_events 设置为一个值，它启用/禁用支持，以便在 binlog 条目中包括原始 SQL 语句。

  - ON = enabled
  - OFF = disabled

6.4.6. 为 Debezium MySQL 连接器验证 binlog 行值选项

检查 binlog_row_value_options 变量，并确保该值 没有设置为 PARTIAL_JSON，因为在这种情况下，连接器可能无法使用 UPDATE 事件。

先决条件

- MySQL 服务器。
- SQL 命令的基本知识.
- 访问 MySQL 配置文件。

流程

1. 检查当前变量值

```
mysql> show global variables where variable_name = 'binlog_row_value_options';
```

2. 结果

```
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| binlog_row_value_options |       |
+--------------------------+-------+
```

3. 如果值为 PARTIAL_JSON，请通过以下方法取消设置此变量：

```
mysql> set @@global.binlog_row_value_options="" ;
```

6.5. 部署 DEBEZIUM MYSQL 连接器

您可以使用以下任一方法部署 Debezium MySQL 连接器：

- 使用 AMQ Streams 自动创建包含连接器插件的镜像。

  这是首选的方法。

- 从 Dockerfile 构建自定义 Kafka Connect 容器镜像。

其他资源

- 第 6.5.5 节 “Debezium MySQL 连接器配置属性的描述”

6.5.1. 使用 AMQ Streams 部署 MySQL 连接器

从 Debezium 1.7 开始，部署 Debezium 连接器的首选方法是使用 AMQ Streams 构建包含连接器插件的 Kafka Connect 容器镜像。

在部署过程中，您可以创建并使用以下自定义资源(CR)：

-

定义 Kafka Connect 实例的 KafkaConnect CR，并包含有关镜像中需要包含连接器工件的信息。

- KafkaConnector CR，提供包括连接器用来访问源数据库的信息。在 AMQ Streams 启动 Kafka Connect pod 后，您可以通过应用 KafkaConnector CR 来启动连接器。

在 Kafka Connect 镜像的构建规格中，您可以指定可用于部署的连接器。对于每个连接器插件，您还可以指定您的部署可以使用的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。当 AMQ Streams 构建 Kafka Connect 镜像时，它会下载指定的工件，并将其合并到镜像中。

KafkaConnect CR 中的 spec.build.output 参数指定存储生成的 Kafka Connect 容器镜像的位置。容器镜像可以存储在 Docker registry 中，也可以存储在 OpenShift ImageStream 中。要将镜像存储在 ImageStream 中，您必须在部署 Kafka Connect 前创建 ImageStream。镜像流不会被自动创建。

注意

如果使用 KafkaConnect 资源来创建集群，之后无法使用 Kafka Connect REST API 创建或更新连接器。您仍然可以使用 REST API 来检索信息。

其他资源

- 在 OpenShift 中使用 AMQ Streams 配置 Kafka 连接。
- 在 OpenShift 中部署和管理 AMQ Streams 中，使用 AMQ Streams 自动创建新容器镜像。

6.5.2. 使用 AMQ Streams 部署 Debezium MySQL 连接器

使用早期版本的 AMQ Streams 时，要在 OpenShift 上部署 Debezium 连接器，您需要首先为连接器构建 Kafka Connect 镜像。在 OpenShift 上部署连接器的当前首选方法是使用 AMQ Streams 中的构建配置来构建 Kafka Connect 容器镜像，其中包含您要使用的 Debezium 连接器插件。

在构建过程中，AMQ Streams Operator 将 KafkaConnect 自定义资源（包括 Debezium 连接器定义）中的输入参数转换为 Kafka Connect 容器镜像。构建会从 Red Hat Maven 存储库或其他配置的 HTTP 服务器下载必要的工件。

新创建的容器被推送到在 .spec.build.output 中指定的容器 registry，用于部署 Kafka Connect 集群。在 AMQ Streams 构建 Kafka Connect 镜像后，您可以创建 KafkaConnector 自定义资源来启动构

建中包含的连接器。

先决条件

- 您可以访问安装了集群 Operator 的 OpenShift 集群。
- AMQ Streams Operator 正在运行。
- 在 OpenShift 中部署和升级 AMQ Streams 所述，会部署 Apache Kafka 集群。
- Kafka Connect 在 AMQ Streams 上部署
- 您有一个 Red Hat Integration 许可证。
- 已安装 OpenShift oc CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。
- 根据您要存储 Kafka Connect 构建镜像的方式，您需要 registry 权限，或者您必须创建 ImageStream 资源：

  将构建镜像存储在镜像 registry 中，如 Red Hat Quay.io 或 Docker Hub

  - 在 registry 中创建和管理镜像的帐户和权限。

  将构建镜像存储为原生 OpenShift ImageStream

  - ImageStream 资源已部署到集群中，以存储新的容器镜像。您必须为集群显式创建 ImageStream。默认无法使用镜像流。如需有关 ImageStreams 的更多信息，请参阅在 OpenShift Container Platform 中管理镜像流。

流程

1. 登录 OpenShift 集群。
2.

为连接器创建 Debezium KafkaConnect 自定义资源(CR)，或修改现有的资源。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 metadata.annotations 和 spec.build 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

例 6.1. 定义包含 Debezium 连接器的 KafkaConnect 自定义资源的 dbz-connect.yaml 文件

在以下示例中，自定义资源被配置为下载以下工件：

- Debezium MySQL 连接器存档。
- Service Registry 归档。Service Registry 是一个可选组件。只有在打算将 Avro 序列化与连接器搭配使用时，才添加 Service Registry 组件。
- Debezium 脚本 SMT 归档以及您要与 Debezium 连接器一起使用的关联脚本引擎。SMT 归档和脚本语言依赖项是可选组件。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时，才添加这些组件。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: debezium-kafka-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" (1)
spec:
  version: 3.5.0
  build: (2)
    output: (3)
      type: imagestream (4)
      image: debezium-streams-connect:latest
    plugins: (5)
      - name: debezium-connector-mysql
        artifacts:
          - type: zip (6)
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-mysql/2.3.4.Final-redhat-00001/debezium-connector-mysql-2.3.4.Final-redhat-00001-plugin.zip (7)
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-distro-connect-converter/2.4.4.Final-redhat-<build-number>/apicurio-registry-distro-connect-converter-2.4.4.Final-redhat-<build-number>.zip (8)
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-scripting/2.3.4.Final-redhat-00001/debezium-scripting-2.3.4.Final-redhat-00001.zip (9)
          - type: jar
```

```
        url:
https://repo1.maven.org/maven2/org/codehaus/groovy/groovy/3.0.11/groovy-
3.0.11.jar 10
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-
jsr223/3.0.11/groovy-jsr223-3.0.11.jar
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-
json3.0.11/groovy-json-3.0.11.jar

  bootstrapServers: debezium-kafka-cluster-kafka-bootstrap:9093

  ...
```

表 6.24. Kafka Connect 配置设置的描述

项	描述
1	将 **strimzi.io/use-connector-resources** 注解设置为 **"true"**，使 Cluster Operator 使用 **KafkaConnector** 资源在此 Kafka Connect 集群中配置连接器。
2	**spec.build** 配置指定在镜像中存储构建镜像的位置，并列出要在镜像中包含的插件，以及插件工件的位置。
3	**build.output** 指定存储新构建镜像的 registry。
4	指定镜像输出的名称和镜像名称。**output.type** 的有效值是 **要推送到** 容器 registry（如 Docker Hub 或 Quay）或 **镜像流** 的有效值，以将镜像推送到内部 OpenShift ImageStream。要使用 ImageStream，必须将 ImageStream 资源部署到集群中。有关在 KafkaConnect 配置中指定 **build.output** 的更多信息，请参阅在 OpenShift 中配置 AMQ Streams 中的 AMQ Streams Build schema 参考。
5	**plugins** 配置列出了您要包含在 Kafka Connect 镜像中的所有连接器。对于列表中的每个条目，指定一个插件**名称**，以及有关构建连接器所需的工件的信息。另外，对于每个连接器插件，您还可以包含可用于连接器的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。
6	**artifacts.type** 的值指定在 **artifacts.url** 中指定的工件类型。有效类型为 **zip**、**tgz** 或 **jar**。Debezium 连接器存档以 **.zip** 文件格式提供。**类型** 值必须与 **url** 字段中引用的文件类型匹配。
7	**artifacts.url** 的值指定 HTTP 服务器的地址，如 Maven 存储库，用于存储连接器工件的文件。Debezium 连接器工件在 Red Hat Maven 存储库中提供。OpenShift 集群必须有权访问指定的服务器。
8	（可选）指定用于下载 Service Registry 组件的工件 **类型和 url**。包含 Service Registry 工件，只有在您希望连接器使用 Apache Avro 来序列化带有 Service Registry 的事件键和值时，而不是使用默认的 JSON 转换程序。
9	（可选）指定 Debezium 脚本 SMT 归档的工件 **类型和 url**，以用于 Debezium 连接器。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时才包括脚本 SMT。要使用脚本 SMT，您必须部署 JSR 223 兼容脚本实现，如 groovy。

项	描述
10	（可选）指定 JSR 223 兼容脚本实施的 JAR 文件的工件 **类型和 url**，这是 Debezium 脚本 SMT 所需的。  **重要** 如果使用 AMQ Streams 将连接器插件合并到 Kafka Connect 镜像中，每个所需的脚本语言 **工件。url** 必须指定 JAR 文件的位置，并且 **artifacts.type** 的值也必须设置为 **jar**。无效的值会导致连接器在运行时失败。 要启用带有脚本 SMT 的 Apache Groovy 语言，示例中的自定义资源会为以下库检索 JAR 文件： • **groovy** • **Groovy-jsr223**（指定代理） • **groovy-json**（解析 JSON 字符串的模块） 作为替代方案，Debebe Debezium 脚本 SMT 也支持使用 JSR 223 实现 GraalVM JavaScript。

3. *输入以下命令将 KafkaConnect 构建规格应用到 OpenShift 集群：*

```
oc create -f dbz-connect.yaml
```

根据自定义资源中指定的配置，Streams Operator 准备要部署的 Kafka Connect 镜像。构建完成后，Operator 将镜像推送到指定的 registry 或 ImageStream，并启动 Kafka Connect 集群。集群中提供了您在配置中列出的连接器工件。

4. *创建一个 KafkaConnector 资源来定义您要部署的每个连接器的实例。*
例如，创建以下 KafkaConnector CR，并将它保存为 mysql-inventory-connector.yaml

例 6.2. 为 Debezium 连接器定义 KafkaConnector 自定义资源的 mysql-inventory-connector.yaml 文件

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: debezium-kafka-connect-cluster
  name: inventory-connector-mysql (1)
spec:
  class: io.debezium.connector.mysql.MySqlConnector (2)
```

```
    tasksMax: 1 3
    config: 4
      schema.history.internal.kafka.bootstrap.servers: debezium-kafka-cluster-
kafka-bootstrap.debezium.svc.cluster.local:9092
      schema.history.internal.kafka.topic: schema-changes.inventory
      database.hostname: mysql.debezium-mysql.svc.cluster.local 5
      database.port: 3306 6
      database.user: debezium 7
      database.password: dbz 8
      database.server.id: 184054 9
      topic.prefix: inventory-connector-mysql 10
      table.include.list: inventory.* 11

      ...
```

表 6.25. 连接器配置设置的描述

项	描述
1	使用 Kafka Connect 集群注册的连接器名称。
2	连接器类的名称。
3	可以同时操作的任务数量。
4	连接器的配置。
5	主机数据库实例的地址。
6	数据库实例的端口号。
7	Debezium 用于连接到数据库的帐户名称。
8	Debezium 用于连接到数据库用户帐户的密码。
9	连接器的唯一数字 ID。
10	数据库实例或集群的主题前缀。 指定的名称只能由字母数字字符或下划线组成。 因为主题前缀被用作从这个连接器接收更改事件的任何 Kafka 主题的前缀，所以该名称在集群中的连接器之间必须是唯一的。 如果连接器与 Avro 连接器集成，则此命名空间也用于相关 Kafka Connect 模式的名称，以及相应 Avro 模式的命名空间。
11	连接器捕获更改事件的表列表。

5. 运行以下命令来创建连接器资源：

```
oc create -n <namespace> -f <kafkaConnector>.yaml
```

例如，

```
oc create -n debezium -f mysql-inventory-connector.yaml
```

连接器注册到 Kafka Connect 集群，并开始针对 KafkaConnector CR 中的 spec.config.database.dbname 指定的数据库运行。连接器 pod 就绪后，Debebe 正在运行。

现在，您已准备好 验证 Debezium MySQL 部署。

6.5.3. 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium MySQL 连接器

要部署 Debezium MySQL 连接器，您必须构建包含 Debezium 连接器存档的自定义 Kafka Connect 容器镜像，然后将此容器镜像推送到容器 registry。然后，您需要创建以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR。CR 中的 image 属性指定您创建的容器镜像的名称，以运行 Debezium 连接器。您可以将此 CR 应用到部署 Red Hat AMQ Streams 的 OpenShift 实例。AMQ Streams 提供将 Apache Kafka 带到 OpenShift 的 operator 和镜像。
- 定义 Debezium MySQL 连接器的 KafkaConnector CR。将此 CR 应用到应用 KafkaConnect CR 的同一 OpenShift 实例。

先决条件

- MySQL 正在运行，并完成了 设置 MySQL 以使用 Debezium 连接器 的步骤。
- AMQ Streams 部署在 OpenShift 中，并运行 Apache Kafka 和 Kafka Connect。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams。
- podman 或 Docker 已安装。
-

您有一个在容器 registry 中创建和管理容器（如 quay.io 或 docker.io）的帐户和权限，您要添加将运行 Debezium 连接器的容器。

流程

1. *为 Kafka Connect 创建 Debezium MySQL 容器：*

a. *创建一个使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 的 Dockerfile 作为基础镜像。例如，在终端窗口中输入以下命令：*

```
cat <<EOF >debezium-container-for-mysql.yaml (1)
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/debezium (2)
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-mysql/2.3.4.Final-redhat-00001/debezium-connector-mysql-2.3.4.Final-redhat-00001-plugin.zip \
&& unzip debezium-connector-mysql-2.3.4.Final-redhat-00001-plugin.zip \
&& rm debezium-connector-mysql-2.3.4.Final-redhat-00001-plugin.zip
RUN cd /opt/kafka/plugins/debezium/
USER 1001
EOF
```

项	描述
1	您可以指定您想要的任何文件名。
2	指定 Kafka Connect 插件目录的路径。如果您的 Kafka Connect 插件目录位于不同的位置，请将此路径替换为目录的实际路径。

该命令在当前目录中创建一个名为 debezium-container-for-mysql.yaml 的 Dockerfile。

b. *从您在上一步中创建的 debezium-container-for-mysql.yaml Docker 文件中构建容器镜像。在包含文件的目录中，打开终端窗口并输入以下命令之一：*

```
podman build -t debezium-container-for-mysql:latest .
```

```
docker build -t debezium-container-for-mysql:latest .
```

前面的命令使用名称 debezium-container-for-mysql 构建容器镜像。

c.

将自定义镜像推送到容器 registry，如 quay.io 或内部容器 registry。容器 registry 必须可供您要部署镜像的 OpenShift 实例使用。输入以下命令之一：

```
podman push <myregistry.io>/debezium-container-for-mysql:latest
```

```
docker push <myregistry.io>/debezium-container-for-mysql:latest
```

d.

创建新的 Debezium MySQL KafkaConnect 自定义资源(CR)。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 注解和 镜像 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  #...
  image: debezium-container-for-mysql 2

  ...
```

项	描述
1	**metadata.annotations** 表示 **KafkaConnector** 资源用于配置在这个 Kafka Connect 集群中使用的 Cluster Operator。
2	**spec.image** 指定您创建的镜像的名称，以运行 Debezium 连接器。此属性覆盖 Cluster Operator 中的 **STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE** 变量。

e.

输入以下命令将 KafkaConnect CR 应用到 OpenShift Kafka Connect 环境：

```
oc create -f dbz-connect.yaml
```

该命令添加了一个 Kafka Connect 实例，用于指定您为运行 Debezium 连接器而创建的镜像的名称。

2. 创建一个 KafkaConnector 自定义资源来配置 Debezium MySQL 连接器实例。

您可以在 .yaml 文件中配置 Debezium MySQL 连接器，该文件指定连接器的配置属性。连接器配置可能指示 Debezium 为 schema 和表的子集生成事件，或者可能会设置属性，以便 Debezium 忽略、掩码或截断敏感、太大或不需要的指定列中的值。

以下示例配置了一个 Debezium 连接器，它连接到 MySQL 主机 192.168.99.100，使用端口 3306，它会捕获 inventory 数据库的更改。dbserver1 是服务器的逻辑名称。

MySQL inventory-connector.yaml

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: inventory-connector-mysql (1)
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: io.debezium.connector.mysql.MySqlConnector
  tasksMax: 1 (2)
  config: (3)
    database.hostname: mysql (4)
    database.port: 3306
    database.user: debezium
    database.password: dbz
    database.server.id: 184054 (5)
    topic.prefix: inventory-connector-mysql (6)
    table.include.list: inventory (7)
    schema.history.internal.kafka.bootstrap.servers: my-cluster-kafka-bootstrap:9092
(8)
    schema.history.internal.kafka.topic: schema-changes.inventory (9)
```

表 6.26. 连接器配置设置的描述

项	描述
1	连接器的名称。

项	描述
2	任何时候都只能运行一个任务。因为 MySQL 连接器使用单一连接器任务读取 MySQL 服务器的 **binlog** 可确保正确的顺序和事件处理。Kafka Connect 服务使用连接器来启动一个或多个完成工作的任务，并在 Kafka Connect 服务集群中自动分发正在运行的任务。如果有任何服务停止或崩溃，则这些任务将重新分发到运行的服务。
3	连接器的配置。
4	数据库主机，这是运行 MySQL 服务器的容器的名称(**mysql**)。
5	连接器的唯一 ID。
6	MySQL 服务器或集群的主题前缀。此名称用作接收更改事件记录的所有 Kafka 主题的前缀。
7	连接器只捕获 **inventory** 表中的更改。
8	此连接器将用来写入和恢复 DDL 语句到数据库 schema 历史记录主题的 Kafka 代理列表。重启时，连接器会在连接器开始读取时恢复 binlog 中存在的数据库的模式。
9	数据库架构历史记录主题的名称。本主题仅用于内部使用，不应供消费者使用。

3. *使用 Kafka Connect 创建连接器实例。例如，如果您将 KafkaConnector 资源保存在 inventory-connector.yaml 文件中，您将运行以下命令：*

```
oc apply -f inventory-connector.yaml
```

前面的命令注册 inventory-connector，连接器开始针对 KafkaConnector CR 中定义的 inventory 数据库运行。

有关您可以为 Debezium MySQL 连接器设置的配置属性的完整列表，请参阅 MySQL 连接器配置属性。

结果

连接器启动后，它会对配置了连接器的 MySQL 数据库执行一致的快照。然后，连接器开始为行级操作生成数据更改事件，并将事件记录流传输到 Kafka 主题。

6.5.4. 验证 Debezium MySQL 连接器是否正在运行

如果连接器正确启动且没有错误，它会为每个连接器配置为捕获的表创建一个主题。下游应用程序可

以订阅这些主题，以检索源数据库中发生的信息事件。

要验证连接器是否正在运行，您可以从 OpenShift Container Platform Web 控制台或 OpenShift CLI 工具(oc)执行以下操作：

- 验证连接器状态。
- 验证连接器是否生成主题。
- 验证主题是否填充了读取操作("op":"r")的事件，连接器在每个表的初始快照中生成。

先决条件

- Debezium 连接器部署到 OpenShift 上的 AMQ Streams。
- 已安装 OpenShift oc CLI 客户端。
- 访问 OpenShift Container Platform web 控制台。

流程

1. 使用以下方法之一检查 KafkaConnector 资源的状态：
   - 在 OpenShift Container Platform Web 控制台中：
     a. 导航到 Home → Search。
     b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaConnector。
     c. 在 KafkaConnectors 列表中，点您要检查的连接器的名称，如 inventory-connector-mysql。

d.

在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- *在终端窗口中：*

a.

使用以下命令：

```
oc describe KafkaConnector <connector-name> -n <project>
```

例如，

```
oc describe KafkaConnector inventory-connector-mysql -n debezium
```

该命令返回类似以下示例的状态信息：

例 6.3. KafkaConnector 资源状态

```
Name:         inventory-connector-mysql
Namespace:    debezium
Labels:       strimzi.io/cluster=debezium-kafka-connect-cluster
Annotations:  <none>
API Version:  kafka.strimzi.io/v1beta2
Kind:         KafkaConnector

...

Status:
  Conditions:
    Last Transition Time:  2021-12-08T17:41:34.897153Z
    Status:                True
    Type:                  Ready
  Connector Status:
    Connector:
      State:      RUNNING
      worker_id:  10.131.1.124:8083
    Name:         inventory-connector-mysql
    Tasks:
      Id:              0
      State:           RUNNING
      worker_id:       10.131.1.124:8083
    Type:              source
  Observed Generation:  1
  Tasks Max:            1
  Topics:
    inventory-connector-mysql.inventory
```

```
    inventory-connector-mysql.inventory.addresses
    inventory-connector-mysql.inventory.customers
    inventory-connector-mysql.inventory.geom
    inventory-connector-mysql.inventory.orders
    inventory-connector-mysql.inventory.products
    inventory-connector-mysql.inventory.products_on_hand
Events:  <none>
```

2. 验证连接器是否创建了 Kafka 主题：

- 通过 OpenShift Container Platform Web 控制台。

a. 导航到 Home → Search。

b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaTopic。

c. 在 KafkaTopics 列表中，点您要检查的主题名称，例如 inventory-connector-mysql.inventory.orders---ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- 在终端窗口中：

a. 使用以下命令：

```
oc get kafkatopics
```

该命令返回类似以下示例的状态信息：

例 6.4. KafkaTopic 资源状态

```
NAME                                                CLUSTER
PARTITIONS   REPLICATION FACTOR   READY
connect-cluster-configs                             debezium-kafka-cluster   1
1                  True
```

```
connect-cluster-offsets                                    debezium-kafka-cluster   25
1               True
connect-cluster-status                                     debezium-kafka-cluster   5
1               True
consumer-offsets---84e7a678d08f4bd226872e5cdd4eb527fadc1c6a
debezium-kafka-cluster   50          1               True
inventory-connector-mysql--a96f69b23d6118ff415f772679da623fbbb99421
debezium-kafka-cluster   1           1               True
inventory-connector-mysql.inventory.addresses---
1b6beaf7b2eb57d177d92be90ca2b210c9a56480          debezium-kafka-cluster
1           1               True
inventory-connector-mysql.inventory.customers---
9931e04ec92ecc0924f4406af3fdace7545c483b          debezium-kafka-cluster   1
1               True
inventory-connector-mysql.inventory.geom---
9f7e136091f071bf49ca59bf99e86c713ee58dd5            debezium-kafka-cluster
1           1               True
inventory-connector-mysql.inventory.orders---
ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d            debezium-kafka-cluster
1           1               True
inventory-connector-mysql.inventory.products---
df0746db116844cee2297fab611c21b56f82dcef          debezium-kafka-cluster   1
1               True
inventory-connector-mysql.inventory.products_on_hand---
8649e0f17ffcc9212e266e31a7aeea4585e5c6b5   debezium-kafka-cluster   1
1               True
schema-changes.inventory                                   debezium-kafka-cluster
1           1               True
strimzi-store-topic---effb8e3e057afce1ecf67c3f5d8e4e3ff177fc55          debezium-
kafka-cluster   1           1               True
strimzi-topic-operator-kstreams-topic-store-changelog---
b75e702040b99be8a9263134de3507fc0cc4017b  debezium-kafka-cluster  1   1
True
```

3. *检查主题内容。*

- *在终端窗口中输入以下命令：*

```
oc exec -n <project>  -it <kafka-cluster> -- /opt/kafka/bin/kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=<topic-name>
```

例如，

```
oc exec -n debezium  -it debezium-kafka-cluster-kafka-0 -- /opt/kafka/bin/kafka-
console-consumer.sh \
```

```
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=inventory-connector-mysql.inventory.products_on_hand
```

指定主题名称的格式与 oc describe 命令返回的格式与第 1 步中返回，例如 inventory-connector-mysql.inventory.addresses。

对于主题中的每个事件，命令会返回类似以下示例的信息：

例 6.5. Debezium 更改事件的内容


```
{"schema":{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"inventory-
connector-mysql.inventory.products_on_hand.Key"},"payload":{"product_id":101}}
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-mysql.inventory.products_on_hand.Value","field":"before"},
{"type":"struct","fields":[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-mysql.inventory.products_on_hand.Value","field":"after"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":
{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connecto
r.mysql.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transacti
on"}],"optional":false,"name":"inventory-connector-
mysql.inventory.products_on_hand.Envelope"},"payload":{"before":null,"after":
{"product_id":101,"quantity":3},"source":{"version":"2.3.4.Final-redhat-
00001","connector":"mysql","name":"inventory-connector-
mysql","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"table":"p
roducts_on_hand","server_id":0,"gtid":null,"file":"mysql-
bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":1638985247805,"t
ransaction":null}}
```

在前面的示例中，有效负载 值显示连接器快照从表 inventory.products_on_hand 生成读取 (op" ="r")事件。 product_id 记录的 "before" 状态为 null，表示该记录不存在之前的 值。 "after" 状态对于 product_id 为 101 的项目的 quantity 显示为 3。

6.5.5. Debezium MySQL 连接器配置属性的描述

Debezium MySQL 连接器具有大量配置属性，可用于实现应用程序的正确连接器行为。许多属性都有默认值。有关属性的信息组织如下：

- *所需的连接器配置属性*
- *高级连接器配置属性*
- *数据库模式历史记录连接器配置属性，用于控制 Debezium 如何处理从数据库 schema 历史记录主题读取的事件。*
  - *透传数据库架构历史记录属性*
- *控制 数据库驱动程序行为的直通数据库驱动程序属性。*

除非默认值可用， 否则需要以下配置属性。

表 6.27. 所需的 Debezium MySQL 连接器配置属性

属性	默认	描述
name	没有默认值	连接器的唯一名称。尝试使用相同的名称再次注册会失败。所有 Kafka Connect 连接器都需要此属性。
connector.class	没有默认值	连接器的 Java 类的名称。始终为 MySQL 连接器指定 **io.debezium.connector.mysql.MySqlConnector**。
tasks.max	1	应该为此连接器创建的最大任务数量。MySQL 连接器始终使用单个任务，因此不使用这个值，因此默认值始终可以接受。
database.hostname	没有默认值	MySQL 数据库服务器的 IP 地址或主机名。

属性	默认	描述
database.port	**3306**	MySQL 数据库服务器的整数端口号。
database.user	没有默认值	连接到 MySQL 数据库服务器时要使用的 MySQL 用户的名称。
database.password	没有默认值	连接到 MySQL 数据库服务器时要使用的密码。
topic.prefix	没有默认值	为特定 MySQL 数据库服务器/集群提供命名空间的主题前缀，其中 Debezium 正在捕获更改。主题前缀应该在所有其他连接器中唯一，因为它被用作接收此连接器发送事件的所有 Kafka 主题名称的前缀。数据库服务器逻辑名称中只能使用字母数字字符、连字符、句点和下划线。 **警告** 不要更改此属性的值。如果您重启后更改了 name 值，而不是继续向原始主题发出事件，连接器会将后续事件发送到名称基于新值的主题。连接器也无法恢复其数据库架构历史记录主题。
database.server.id	没有默认值	此数据库客户端的数字 ID，它在 MySQL 集群中的所有当前运行数据库进程之间必须是唯一的。这个连接器将 MySQL 数据库集群作为另一个服务器（使用此唯一 ID）加入，以便它可以读取 binlog。
database.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与捕获更改的数据库的名称匹配。连接器不会捕获名称不在 **database.include.list** 的任何数据库中的更改。默认情况下，连接器捕获所有数据库中的更改。 要匹配数据库的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据库的整个名称字符串匹配；它与数据库名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **database.exclude.list** 属性。

属性	默认	描述
database.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与您不想捕获更改的数据库名称匹配。连接器捕获名称不在 **database.exclude.list** 中的任何数据库中的更改。 要匹配数据库的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据库的整个名称字符串匹配；它与数据库名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **database.include.list** 属性。
table.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与您要捕获的更改的全限定表标识符匹配。连接器不会捕获 **table.include.list** 中不包含的任何表中的更改。每个标识符都是 *databaseName.tableName*。默认情况下，连接器会捕获每个数据库中的每个非系统表中的更改，并捕获其更改。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.exclude.list** 属性。
table.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与您不想捕获的表的完全限定表标识符匹配。连接器捕获没有包括在 **table.exclude.list** 中的更改。每个标识符都是 *databaseName.tableName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.include.list** 属性。
column.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与列的完全限定域名匹配，以便从更改事件记录值中排除。列的完全限定域名格式为 *databaseName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；它与列中可能出现的子字符串匹配。如果您在配置中包含此属性，不要设置 **column.include.list** 属性。

属性	默认	描述
column.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与列的完全限定域名匹配，以在更改事件记录值中包含。列的完全限定域名格式为 *databaseName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；它与列中可能出现的子字符串匹配。 如果您在配置中包含此属性，请不要设置 **column.exclude.list** 属性。
skip.messages.without.change	**false**	指定在包含列中没有更改时是否跳过发布消息。如果列中没有包括每个 **column.include.list** 或 **column.exclude.list** 属性的列有变化，这将过滤消息。
column.truncate.to.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果在列中的数据超过了在属性名中的 *length* 指定的字符长度时删节数据，设置此属性。将 **length** 设置为正整数值，如 **column.truncate.to.20.chars**。 列的完全限定域名会观察以下格式：*databaseName.tableName.columnName*。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。
column.mask.with.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果您希望连接器屏蔽一组列的值，例如，如果它们包含敏感数据，则设置此属性。将 ***length*** 设置为一个正整数，替换在属性名称中的 *length* 指定的星号（*）的数量列中的数据。将 *length* 设置为 **0**（零）将指定列中的数据替换为空字符串。 列的完全限定域名会观察以下格式：*databaseName.tableName.columnName*。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。

属性	默认	描述
column.mask.hash.*hashAlgorithm*.with.salt.*salt*; column.mask.hash.v2.*hashAlgorithm*.with.salt.*salt*	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。列的完全限定域名格式为 ***<databaseName> . <tableName & gt; . <columnName>***。 要匹配一个列的名称，Debezium 应用正则表达式，它由您指定为 *anchored* 正则表达式。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。在生成的更改事件记录中，指定列的值替换为 pseudonyms。 一个 pseudonym，它包括了通过应用指定的 *hashAlgorithm* 和 *salt* 的结果的哈希值。根据所使用的哈希函数，会维护引用完整性，而列值则替换为 pseudonyms。支持的哈希功能在 Java Cryptography 架构标准 Algorithm Name 文档的 MessageDigest 部分中 进行了描述。 在以下示例中，**CzQMA0cB5K** 是一个随机选择的 salt。 `column.mask.hash.SHA-256.with.salt.CzQMA0cB5K = inventory.orders.customerName, inventory.shipment.customerName` 如有必要，pseudonym 会自动缩短为列的长度。连接器配置可以包含多个属性，用于指定不同的哈希算法和 salt。 根据所用的 *hashAlgorithm* （选择 *salt* ）和实际数据集，生成的数据集可能无法完全屏蔽。 应该使用哈希策略版本 2 来确保在不同的位置或系统中对值进行哈希处理。

属性	默认	描述
column.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，与您希望连接器发出代表列元数据的额外参数的完全限定名称匹配。当设置此属性时，连接器会将以下字段添加到事件记录的 schema 中： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*databaseName.tableName.columnName*, 或 *databaseName.schemaName.tableName.columnName*. 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。
datatype.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，用于指定为数据库中列定义的数据类型的完全限定名称。当设置此属性时，对于具有匹配数据类型的列，连接器会发出在 schema 中包含以下额外字段的事件记录： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*databaseName.tableName.typeName*, 或 *databaseName.schemaName.tableName.typeName*. 要匹配数据类型的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据类型的整个名称字符串匹配；表达式与类型名称中可能存在的子字符串不匹配。 有关特定于 MySQL 的数据类型名称的列表，请参阅 MySQL 数据类型映射。

属性	默认	描述
time.precision.mode	adaptive_time_microseconds	时间、日期和时间戳可以通过不同类型的精度来表示，包括： **adaptive_time_microseconds**（默认值）捕获日期、日期和时间戳值，使用 millisecond、microsecond 或 nanosecond 精度值，但 TIME 类型字段除外，但 TIME 类型字段除外， 它们始终被捕获为微秒。 **connect** 始终使用 Kafka Connect 的内置表示表示 Time, Date, 和 Timestamp，无论数据库列的精度是什么，无论数据库列的精度是什么，它都使用 millisecond 精度。
decimal.handling.mode	精确	指定连接器如何处理 **DECIMAL** 和 **NUMERIC** 列： **precise**（默认值）代表它们准确使用二进制格式更改事件中的 **java.math.BigDecimal** 值。 **double** 表示它们使用 **double** 值，这可能会丢失一些精度但更容易使用。 **string** 将值编码为格式化的字符串，它容易使用，但提供有关实际类型的语义信息会丢失。
bigint.unsigned.handling.mode	long	指定更改事件中应如何表示 BIGINT UNSIGNED 列。可能的设置是： **long** 代表使用 Java 的长 值，它可能不提供精度，但易于在消费者中使用。长 通常是首选设置。 **精确** 使用 **java.math.BigDecimal** 来代表值，使用二进制表示和 Kafka Connect 的 **org.apache.kafka.connect.data.Decimal** 类型来在更改事件中编码。在处理大于 2^63 的值时，请使用此设置，因为无法使用 **long** 来转换这些值。
include.schema.changes	true	布尔值，指定连接器是否应该将数据库模式中的更改发布到与数据库服务器 ID 的名称相同的 Kafka 主题。每个架构更改都由使用包含数据库名称的键进行记录，其值包含 DDL 语句。这独立于连接器内部记录数据库架构历史记录。
include.schema.comments	false	指定连接器是否应该解析和发布元数据对象上的表和列注释的布尔值。启用这个选项会对内存用量造成影响。逻辑模式对象的数量和大小对 Debezium 连接器消耗的内存数量和大小有很大的影响，并可能为每个连接器添加大的字符串数据可能会非常昂贵。

属性	默认	描述
include.query	**false**	指定连接器是否应该包含生成更改事件的原始 SQL 查询的布尔值。 如果将此选项设置为 **true**，还必须配置 MySQL，并将 **binlog_rows_query_log_events** 选项设置为 **ON**。当 **include.query** 为 **true** 时，对快照进程生成的事件不存在查询。 将 **include.query** 设置为 **true** 可能会通过在更改事件中包含原始 SQL 语句来公开明确排除或屏蔽的表或字段。因此，默认设置为 **false**。
event.deserialization.failure.handling.mode	**fail**	指定连接器在 binlog 事件反序列化过程中应如何响应异常。这个选项已弃用，请使用 **event.processing.failure.handling.mode** 选项。 **失败** 传播异常，它指示有问题的事件及其 binlog 偏移，并导致连接器停止。 **会警告** 记录有问题的事件及其 binlog 偏移，然后跳过 event. **忽略** 通过有问题的事件，且不会记录任何内容。
inconsistent.schema.handling.mode	**fail**	指定连接器应该如何响应与内部架构表示法中不存在的表相关的 binlog 事件。也就是说，内部表示与数据库不一致。 **失败** 会抛出一个异常，表示有问题的事件及其 binlog 偏移，并导致连接器停止。 **会警告** 记录有问题的事件及其 binlog 偏移并跳过 event. **跳过** 通过有问题的事件，且不会记录任何内容。
max.batch.size	**2048**	正整数值，指定每个应在此连接器迭代过程中处理的事件的最大大小。默认值为 2048。
max.queue.size	**8192**	正整数值，用于指定阻塞队列可以保存的最大记录数。当 Debezium 从数据库读取事件时，它会将事件放置在阻塞队列中，然后再将它们写入 Kafka。阻塞队列可以提供从数据库读取更改事件时，连接器最快于将其写入 Kafka 的信息，或者在 Kafka 不可用时从数据库读取更改事件。当连接器定期记录偏移时，队列中保存的事件会被忽略。始终将 **max.queue.size** 的值设置为大于 **max.batch.size** 的值。

属性	默认	描述
max.queue.size.in.bytes	**0**	一个长的整数值，用于指定阻塞队列的最大卷（以字节为单位）。默认情况下，不会为阻塞队列指定卷限制。要指定队列可以消耗的字节数，请将此属性设置为正长值。 如果还设置了 **max.queue.size**，当队列的大小达到任一属性指定的限制时，写入队列将被阻止。例如，如果您设置了 **max.queue.size=1000**、和 **max.queue.size.in.bytes=5000**，在队列包含 1000 个记录后，或者队列中记录的卷达到 5000 字节后，写入队列会被阻止。
poll.interval.ms	**500**	正整数值，指定连接器在开始处理批处理事件前应等待新更改事件数的毫秒数。默认值为 500 毫秒，或 0.5 秒。
connect.timeout.ms	**30000**	一个正整数值，用于指定此连接器在尝试连接到 MySQL 数据库服务器后应等待的时间（以毫秒为单位）。默认值为 30 秒。
gtid.source.includes	没有默认值	以逗号分隔的正则表达式列表，与 GTID 集合中的源 UUID 匹配，连接器用来在 MySQL 服务器上查找 binlog 位置。当设置此属性时，连接器只使用具有与其中一个指定特征匹配的源 UUID 的 GTID 范围。 **要匹配 GTID 的值，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与整个 UUID 字符串匹配；它不匹配 UUID 中可能存在的子字符串。 如果您在配置中包含此属性，不要同时设置 gtid.source.excludes 属性。**
gtid.source.excludes	没有默认值	以逗号分隔的正则表达式列表，与 GTID 集合中的源 UUID 匹配，连接器用来在 MySQL 服务器上查找 binlog 位置。当设置此属性时，连接器只使用具有与任何指定 **排除** 模式不匹配的源 UUID 的 GTID 范围。 要匹配 GTID 的值，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与整个 UUID 字符串匹配；它不匹配 UUID 中可能存在的子字符串。 如果您在配置中包含此属性，不要同时设置 **gtid.source.includes** 属性。

属性	默认	描述
tombstones.on.delete	**true**	控制 *delete* 事件是否后跟一个 tombstone 事件。 **true** - 一个 delete 操作由*delete* 事件和后续 tombstone 事件表示。 **false** - 仅有一个 *delete* 事件被抛出。 删除源记录后，发出 tombstone 事件（默认行为）可让 Kafka 在为主题启用了 日志 压缩时完全删除与已删除行键相关的所有事件。
message.key.columns	*不适用*	指定连接器用来组成自定义消息键的表达式列表，用于更改它发布到指定表的 Kafka 主题的事件记录。 默认情况下，Debezium 使用表的主键列作为它发出的记录的消息键。在默认位置，或者为缺少主密钥的表指定一个键，您可以根据一个或多个列配置自定义消息密钥。 要为表建立自定义消息键，请列出表，后跟要用作消息键的列。每个列表条目都采用以下格式： *<fully-qualified_tableName> : <keyColumn> , <keyColumn>* To base a table key on multiple column name, 在列名称间插入逗号。 每个完全限定表名称都是以下格式的一个正则表达式： *<databaseName > . <tableName>* 属性可以包括多个表的条目。使用分号分隔列表中的表条目。 以下示例为表 **inventory.customers** 和 **purchase.orders** 设置消息键： **inventory.customers:pk1,pk2; (any).purchaseorders:pk3,pk4** for the table **inventory.customer**, 列 **pk1** 和 **pk2** 被指定为消息键。对于任何 **数据库中的** 购买顺序表，列 **pk3** 和 **pk4** 服务器作为消息键。 对于用来创建自定义消息键的列数量没有限制。但是，最好使用指定唯一密钥所需的最小数量。

属性	默认	描述
binary.handling.mode	bytes	指定二进制列（如 **blob**、**二进制**、**varbinary**）应在更改事件中表示。可能的设置： **字节** 表示二进制数据作为字节数组。 **base64** 代表二进制数据作为 base64 编码的字符串。 **base64-url-safe** 代表二进制数据作为 base64-url-safe-encoded String。 **hex** 代表二进制数据作为十六进制编码(base16)字符串。
schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。可能的设置： ● **none** 不应用任何调整。 ● **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 ● **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列
field.name.adjustment.mode	none	指定应如何调整字段名称以与连接器使用的消息转换器兼容。可能的设置： ● **none** 不应用任何调整。 ● **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 ● **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列 如需了解更多详细信息，请参阅 Avro 命名。

高级 MySQL 连接器配置属性

下表描述了 高级 MySQL 连接器属性。这些属性的默认值很少需要更改。因此，您不需要在连接器配置中指定它们。

表 6.28. MySQL 连接器高级配置属性的描述

属性	默认	描述

属性	默认	描述
connect.keep.alive	**true**	一个布尔值，指定是否应使用单独的线程来确保与 MySQL 服务器/集群的连接保持活跃状态。

converters	没有默认值	枚举连接器可以使用的 自定义转换器 实例的符号链接列表。
		例如，**布尔值**。
		需要此属性来启用连接器以使用自定义转换器。
		
		对于您为连接器配置的每个转换器，您还必须添加一个 **.type** 属性，它指定了实现转换器接口的类的完整名称。**.type** 属性使用以下格式：
		
		***<converterSymbolicName>*.type**
		
		例如，
		
		`boolean.type: io.debezium.connector.mysql.converters.TinyIntOneToBooleanConverter`
		
		如果要进一步控制配置的转换器的行为，您可以添加一个或多个配置参数将值传递给转换器。要将这些额外的配置参数与转换器关联，请为 paraemeter 名称加上转换器的符号名作为前缀。
		
		例如，要定义 **一个选择器** 参数，用于指定 **布尔值** 转换器进程的列子集，请添加以下属性：
		
		`boolean.selector=db1.table1.*, db1.table2.column1`

table.ignore.builtin	**true**	指定是否应忽略内置系统表的布尔值。无论表包含和排除列表，这都适用。默认情况下，系统表不会捕获其更改，并在对任何系统表进行更改时不会生成任何事件。

属性	默认	描述
database.ssl.mode	**preferred**	指定是否使用加密连接。可能的设置是： **disabled** 指定使用未加密的连接。 **preferred** 如果服务器支持安全连接，建立一个加密连接。如果服务器不支持安全连接，则回退到未加密的连接。 ，如果因为任何原因无法建立加密的连接或失败。 **verify_ca** 的行为如 **required**，但它还会验证服务器 TLS 证书与配置的证书颁发机构(CA)证书验证服务器 TLS 证书，如果服务器 TLS 证书与任何有效的 CA 证书不匹配。 **verify_identity** 的行为如 **verify_ca** 的行为，但还要验证服务器证书是否与远程主机匹配。

属性	默认	描述
snapshot.mode	初始	指定在连接器启动时运行快照的条件。可能的设置是： **initial** - 只有在没有为逻辑服务器名称记录偏移时，连接器才会运行快照。 **initial_only** - 连接器仅在不为逻辑服务器名称记录偏移时运行快照，然后停止；即，它不会从 binlog. 读取更改事件。**也就是说，如果没有偏移可用，或者当之前记录的偏移指定了服务器中不可用的 binlog 位置或 GTID 时。** **never - 连接器永远不会使用快照。第一次使用逻辑服务器名称启动时，连接器会从 binlog 的开头读取。谨慎配置此行为。只有在 binlog 被保证包含数据库的完整历史记录时才有效。** **schema_only - 连接器运行模式的快照而不是数据。当您不需要主题包含数据的一致性快照时，此设置很有用，但需要它们只在连接器启动后才有变化。** **schema_only_recovery - 这是已经捕获更改的连接器的恢复设置。重启连接器时，此设置启用恢复损坏或丢失的数据库 schema 历史记录主题。您可以定期将其设置为"清理"数据库架构历史记录主题，该主题被意外增长。数据库架构历史记录主题需要无限保留。**

属性	默认	描述
snapshot.locking.mode	**Minimal**	控制连接器保存全局 MySQL 读取锁定的时间，这会阻止对数据库的任何更新，同时连接器执行快照。可能的设置有： **minimal** - 连接器只保存快照的初始部分的全局读锁，连接器在其上读取数据库模式和其他元数据。快照中剩余的工作涉及从每个表中选择所有行。连接器可以使用 REPEATABLE READ 事务以一致的方式执行此操作。即使全局读取锁定不再保存，其他 MySQL 客户端也会更新数据库。 **minimal_percona** - 连接器只为连接器读取数据库模式和其它元数据的快照的初始部分保存 全局备份锁定。快照中剩余的工作涉及从每个表中选择所有行。连接器可以使用 REPEATABLE READ 事务以一致的方式执行此操作。即使全局备份锁定不再保存，其他 MySQL 客户端也会更新数据库，也是如此。这个模式不会将表刷新到磁盘，不会被长时间运行的读取阻止，且仅在 Percona Server. **extended** - 阻止快照期间的所有写入。如果有客户端提交 MySQL 从 REPEATABLE READ 语义中排除了操作。 **none** - 防止连接器在快照过程中获取任何表锁定。虽然此设置允许所有快照模式，*但只有在* 快照运行时没有发生模式更改时，才可以使用此设置。对于使用 MyISAM 引擎定义的表，表仍然会被锁定，尽管此属性被设置为 MyISAM 获取表锁定。此行为与 InnoDB 引擎不同，后者获取行级锁定。
snapshot.include.collection.list	**table.include.list** 中指定的所有表	可选的、以逗号分隔的正则表达式列表，与表的完全限定域名(<***databaseName>.<tableName>***)匹配，以包括在快照中。指定的项目必须在连接器的 **table.include.list** 属性中命名。只有在连接器的 **snapshot.mode** 属性设置为除 **never** 的值时，此属性才会生效。 此属性不会影响增量快照的行为。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。

属性	默认	描述
snapshot.select.statement.overrides	没有默认值	指定要包含在快照中的表行。如果您希望快照只包含表中的行的子集，请使用属性。此属性仅影响快照。它不适用于连接器从日志中读取的事件。 属性包含以逗号分隔的、完全限定表名称列表，格式为 < ***databaseName>.<tableName>***。例如， **"snapshot.select.statement.overrides": "inventory.products,customers.orders"** 对于列表中的每个表，添加一个进一步的配置属性，用于指定连接器在获取快照时要在表上运行的 **SELECT** 语句。指定的 **SELECT** 语句决定了快照中包含的表行的子集。使用以下格式指定此 **SELECT** 语句属性的名称： **snapshot.select.statement.overrides. *<databaseName> . <tableName>***。例如，**snapshot.select.statement.overrides.customers.orders**。 Example: 在包含 soft-delete 列 **delete_flag** 的 **customers.orders** 表中，如果您希望快照只包含没有软删除的记录，请添加以下属性： "snapshot.select.statement.overrides": "customer.orders", "snapshot.select.statement.overrides.customer.orders": "SELECT * FROM [customers].[orders] WHERE delete_flag = 0 ORDER BY id DESC" 在生成的快照中，连接器只包括 **delete_flag = 0** 的记录。
min.row.count.to.stream.results	**1000**	在快照中，连接器会查询每个表，连接器被配置为捕获更改。连接器使用每个查询结果生成包含该表中所有行数据的读取事件。此属性决定了 MySQL 连接器是否将表的结果放在内存中，这速度非常快，但需要大量内存，或者流出结果，对于非常大的表来说，这个属性可能较慢，但可以正常工作。这个属性的设置指定表在连接器流结果前必须包含的最小行数。 要跳过所有表大小检查，并且始终在快照期间流传输所有结果，请 **将此属性设置为** 0。

属性	默认	描述
heartbeat.interval.ms	0	控制连接器将心跳信息发送到 Kafka 主题的频率。默认行为是连接器不会发送心跳信息。 心跳消息可用于监控连接器是否从数据库接收更改事件。心跳消息有助于减少在连接器重启时需要重新更改事件的数量。要发送心跳消息，请将此属性设置为正整数，这代表心跳消息之间的毫秒数。
heartbeat.action.query	没有默认值	指定连接器发送心跳消息的查询，连接器在源数据库上执行。 例如，这可用于定期捕获源数据库中设置的已执行 GTID 的状态。 **INSERT INTO gtid_history_table (select * from mysql.gtid_executed)**
database.initial.statements	没有默认值	当 JDBC 连接（而不是读取事务日志的连接）到数据库时，将执行分号分隔的 SQL 语句列表。要将分号指定为 SQL 语句中的字符，而不是作为分隔符，请使用两个分号(**;;**)。 连接器可能会自行自行建立 JDBC 连接，因此此属性适合配置会话参数。它不是执行 DML 语句。
snapshot.delay.ms	没有默认值	连接器在连接器启动时执行快照前应等待的时间（以毫秒为单位）。如果您在集群中启动多个连接器，则此属性可用于避免快照中断，这可能会导致连接器的重新平衡。
snapshot.fetch.size	没有默认值	在快照期间，连接器以每行的批处理读取表内容。此属性指定批处理中的最大行数。
snapshot.lock.timeout.ms	10000	正整数，用于指定执行快照时等待的最大时间（以毫秒为单位）来获取表锁定。如果连接器无法在这个时间间隔内获取表锁定，则快照会失败。了解 MySQL 连接器如何执行数据库快照。
enable.time.adjuster	true	指明连接器是否将 2 位年规格转换为 4 位的布尔值。当转换完全委派给数据库时，设置为 **false**。 MySQL 允许用户使用 2 位或 4 位数字插入年值。对于 2 位值，该值映射至 1970 年 - 2069 范围。默认行为是连接器进行转换。

属性	默认	描述
skipped.operations	t	在流过程中将跳过的操作类型的逗号分隔列表。操作包括：用于 inserts/create、u 表示更新、d 表示 delete、t 表示 truncates，none 不跳过任何操作。默认情况下会跳过 truncate 操作。
signal.data.collection	没有默认值	用于向连接器发送信号的数据收集的完全限定名称。https://access.redhat.com/documentation/zh-cn/red_hat_integration/2023.q4/html-single/debezium_user_guide/index#debezium-signaling-enabling-source-signaling-channel 使用以下格式指定集合名称：<*databaseName> . < tableName>*
signal.enabled.channels	source	为连接器启用的信号频道名称列表。默认情况下，以下频道可用： • source • kafka • file • jmx
notification.enabled.channels	没有默认值	为连接器启用的通知频道名称列表。默认情况下，以下频道可用： • sink • log • jmx
incremental.snapshot.allow.schema.changes	false	允许在增量快照期间更改模式。启用后，连接器将在增量快照期间检测模式更改，并重新选择当前块以避免锁定 DDL。 请注意，不支持对主密钥的更改，并在增量快照期间执行时可能会导致不正确的结果。另一个限制是，如果模式更改只影响列的默认值，则不会检测到更改，直到从 binlog 流处理 DDL 为止。这不会影响快照事件的值，但快照事件的 schema 可能具有过时的默认值。

属性	默认	描述
incremental.snapshot.chunk.size	1024	连接器在增量快照块期间获取并读取内存的最大行数。增加块大小可提高效率，因为快照会运行更多大小的快照查询。但是，较大的块大小还需要更多内存来缓冲快照数据。将块大小调整为提供环境中最佳性能的值。
provide.transaction.metadata	false	确定连接器是否生成带有事务边界的事件，并使用事务元数据增强更改事件信。如果您希望连接器进行此操作，请指定 **true**。详情请参阅 Transaction metadata。
event.processing.failure.handling.mode	fail	指定在处理事件期间的故障（例如，当遇到损坏事件时）。默认情况下，**失败的** 模式会引发指示有问题的事件及其位置的异常，从而导致连接器停止。**warn** 模式不会引发异常，而是记录有问题的事件，其位置将被跳过。**忽略** 模式会完全忽略有问题的事件，且没有日志记录。
topic.naming.strategy	io.debezium.schema.DefaultTopicNamingStrategy	应该用来确定数据更改、模式更改、事务、心跳事件等的主题名称，默认为 **DefaultTopicNamingStrategy**。
topic.delimiter	.	指定主题名称的分隔符，默认为 **.**。
topic.cache.size	10000	在绑定的并发哈希映射中用于保存主题名称的大小。此缓存将有助于确定与给定数据收集对应的主题名称。
topic.heartbeat.prefix	__debezium-heartbeat	控制连接器向其发送心跳信息的主题名称。主题名称具有此模式： *topic.heartbeat.prefix.topic.prefix* 例如，如果主题前缀是 **fulfillment**，则默认主题名称为 **__debezium-heartbeat.fulfillment**。
topic.transaction	Transactions	控制连接器向其发送事务元数据消息的主题名称。主题名称具有此模式： *topic.prefix.topic.transaction* 例如，如果主题前缀是 **fulfillment**，默认的主题名称为 **fulfillment.transaction**。

属性	默认	描述
snapshot.max.threads	1	指定连接器执行初始快照时使用的线程数量。要启用并行初始快照，请将属性设置为大于 1 的值。在并行初始快照中，连接器会同时处理多个表。 **重要** 并行初始快照只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。
snapshot.tables.order.by.count	disabled	控制连接器执行初始快照时处理表的顺序。指定以下选项之一： **descending** 连接器快照表，按顺序根据最高到最低的行数。 **ascending** 连接器根据行数从最低到最高的顺序快照表。 **disabled** 在执行初始快照时，连接器忽略行计数。
errors.max.retries	-1	在失败前，retriable 错误（如连接错误）的最大重试次数(-1 = no limit, 0 = disabled, > 0 = num of retries)。

Debezium 连接器数据库架构历史记录配置属性

Debezium 提供了一组 schema.history.internal. 属性，用于控制连接器如何与 schema 历史记录主题进行交互。*

下表描述了用于配置 Debezium 连接器的 schema.history.internal 属性。

表 6.29. 连接器数据库架构历史记录配置属性

属性	默认	描述
schema.history.internal.kafka.topic	没有默认值	连接器存储数据库 schema 历史记录的 Kafka 主题的全名。
schema.history.internal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。此连接用于检索之前由连接器存储的数据库架构历史记录，以及用于从源数据库读取的每个 DDL 语句。每个对都应指向 Kafka Connect 进程使用的相同 Kafka 集群。
schema.history.internal.kafka.recovery.poll.interval.ms	**100**	整数值，用于指定连接器在启动/恢复期间应等待的最大毫秒数，同时轮询持久数据。默认值为 100ms。
schema.history.internal.kafka.query.timeout.ms	**3000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端获取集群信息时应等待的最大毫秒数。
schema.history.internal.kafka.create.timeout.ms	**30000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端创建 kafka 历史记录主题时应等待的最大毫秒数。
schema.history.internal.kafka.recovery.attempts	**100**	连接器在连接器恢复失败前应尝试读取持久性历史记录数据的次数上限，并显示错误。接收数据后等待的最大时间为 restore. **attempts recovery. poll.interval.ms**。
schema.history.internal.skip.unparseable.ddl	**false**	指定连接器是否应忽略格式或未知数据库语句的布尔值，或者停止处理，以便人可以解决这个问题。安全默认值为 **false**。跳过应只用于小心，因为在处理 binlog 时可能会导致数据丢失或中断。
schema.history.internal.store.only.captured.tables.ddl	**false**	一个布尔值，用于指定连接器是否记录来自 schema 或数据库中的所有表的模式结构，还是仅从为捕获的表中指定的表。 指定以下值之一： **false （默认）** 在数据库快照过程中，连接器会记录数据库中所有非系统表的 schema 数据，包括没有指定用于捕获的表。最好保留默认设置。如果您稍后决定捕获您最初未指定用于捕获的表的更改，则连接器可以轻松地开始从这些表中捕获数据，因为它们的模式结构已经存储在 schema 历史记录主题中。Debezium 需要表的 schema 历史记录，以便它可以识别发生更改事件时存在的结构。 **true** 在数据库快照过程中，连接器只记录 Debezium 捕获更改事件的表模式。如果您更改了默认值，稍后将连接器配置为从数据库中其他表捕获数据，则连接器缺少从表中捕获更改事件所需的 schema 信息。

属性	默认	描述
schema.history.internal.store.only.captured.databases.ddl	**false**	一个布尔值，用于指定连接器是否记录来自数据库实例中的所有逻辑数据库的架构结构。 指定以下值之一： **true** 连接器只记录 Debezium 捕获更改事件的逻辑数据库和模式中的表的架构结构。 **false** 连接器记录所有逻辑数据库的模式结构。 **注意** MySQL Connector 的默认值为 **true**

配置制作者和消费者客户端的直通数据库架构历史记录属性

Debezium 依赖于 Kafka producer 将模式更改写入数据库架构历史记录主题。同样，它依赖于 Kafka 使用者在连接器启动时从数据库 schema 历史记录主题中读取。您可以通过将值分配给以 schema.history.internal.producer 和 schema.history.internal.consumer ruby 前缀开头的 pass-through 配置属性来定义 Kafka producer 和 消费者 客户端的配置。直通生成者和消费者数据库模式历史记录属性控制一系列行为，如这些客户端与 Kafka 代理的连接的方式，如下例所示：

```
schema.history.internal.producer.security.protocol=SSL
schema.history.internal.producer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.producer.ssl.keystore.password=test1234
schema.history.internal.producer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.producer.ssl.truststore.password=test1234
schema.history.internal.producer.ssl.key.password=test1234

schema.history.internal.consumer.security.protocol=SSL
schema.history.internal.consumer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.consumer.ssl.keystore.password=test1234
schema.history.internal.consumer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.consumer.ssl.truststore.password=test1234
schema.history.internal.consumer.ssl.key.password=test1234
```

Debezium 从属性名称中剥离前缀，然后再将属性传递给 Kafka 客户端。

如需有关 Kafka producer 配置属性和 Kafka 使用者配置属性的更多详情，请参阅 Kafka 文档。

Debezium 连接器 Kafka 信号配置属性

Debezium 提供了一组 signal. 属性，用于控制连接器如何与 Kafka 信号主题进行交互。*

下表描述了 Kafka 信号 属性。

表 6.30. Kafka 信号配置属性

属性	默认	描述
signal.kafka.topic	<topic.prefix>-signal	连接器监控用于临时信号的 Kafka 主题的名称。 **注意** 如果禁用了 自动主题创建，您必须手动创建所需的信号主题。需要信号主题来保留信号排序。信号主题必须具有单个分区。
signal.kafka.groupId	kafka-signal	Kafka 用户使用的组 ID 的名称。
signal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。每个对都引用 Debezium Kafka Connect 进程使用的 Kafka 集群。
signal.kafka.poll.timeout.ms	100	一个整数值，用于指定连接器在轮询信号时等待的最大毫秒数。

Debezium 连接器传递信号 Kafka 使用者客户端配置属性

Debezium 连接器为信号 Kafka 使用者提供直通配置。透传信号属性以 signals.consumer. 前缀开始。例如，连接器将 signal.consumer.security.protocol=SSL 等属性传递给 Kafka 消费者。*

Debezium 从属性中剥离前缀，然后再将属性传递给 Kafka 信号消费者。

Debezium 连接器接收器通知配置属性

下表描述了 通知 属性。

表 6.31. sink 通知配置属性

属性	默认	描述

属性	默认	描述
notification.sink.topic.name	没有默认值	从 Debezium 接收通知的主题名称。当您将 notification.enabled.channels 属性配置为将 sink 作为启用的通知频道之一时，需要此属性。

Debezium 连接器透传数据库驱动程序配置属性

Debezium 连接器为数据库驱动程序的直通配置提供。直通数据库属性以前缀 driver metric 开头。例如，连接器将 driver.foobar=false 等属性传递给 JDBC URL。

与 数据库架构历史记录客户端通过直通属性 一样，Debebe 会在将前缀传递给数据库驱动程序之前从属性中剥离前缀。

6.6. 监控 DEBEZIUM MYSQL 连接器性能

Debezium MySQL 连接器提供三种指标类型，除了对 Zookeeper、Kafka 和 Kafka Connect 提供的 JMX 指标的内置支持之外。

- *快照指标 提供有关执行快照时连接器操作的信息。*
- *当连接器读取 binlog 时，流指标 提供有关连接器操作的信息。*
- *模式历史记录指标 提供有关连接器模式历史记录状态的信息。*

Debezium 监控文档 提供了如何使用 JMX 公开这些指标的详细信息。

6.6.1. 在 MySQL 数据库快照过程中监控 Debezium

MBean 是 debezium.mysql:type=connector-metrics,context=snapshot,server= <topic.prefix>。

快照指标不会公开，除非快照操作处于活跃状态，或者快照自上次连接器启动以来发生。

下表列出了可用的 shapshot 指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个快照事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上次启动或重置后看到的事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在快照和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	队列的空闲容量，用于在快照和主 Kafka Connect 循环之间传递事件。
TotalTableCount	int	包括在快照中的表的总数。
RemainingTableCount	int	快照必须复制的表数。
SnapshotRunning	布尔值	快照是否已启动。
SnapshotPaused	布尔值	快照是否已暂停。
SnapshotAborted	布尔值	快照是否中止。
SnapshotCompleted	布尔值	快照是否完成。
SnapshotDurationInSeconds	long	快照为止所花费的秒数，即使未完成也是如此。也包括快照暂停的时间。
SnapshotPausedDurationInSeconds	long	快照暂停的秒数。如果快照暂停几次，暂停的时间会添加。

属性	类型	描述
RowsScanned	Map<String, Long>	包含快照中每个表的行数的映射。表会在处理过程中逐步添加到映射中。更新每个 10,000 行扫描并在完成表后。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

连接器还在执行增量快照时提供以下额外快照指标：

属性	类型	描述
ChunkId	字符串	当前快照块的标识符。
ChunkFrom	字符串	定义当前块的主密钥集的下限。
ChunkTo	字符串	定义当前块的主密钥集的上限。
TableFrom	字符串	当前快照表的主密钥集的下限。
TableTo	字符串	当前快照表的主密钥集的上限。

Debezium MySQL 连接器还提供 HoldingGlobalLock 自定义快照指标。此指标设置为布尔值，指示连接器当前是否包含全局或表写入锁定。

6.6.2. 监控 Debezium MySQL 连接器记录流

只有在启用了 binlog 事件缓冲时，才会提供与事务相关的属性。MBean 是 debezium.mysql:type=connector-metrics,context=streaming,server= <topic.prefix>。

下表列出了可用的流指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个流事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上一次启动或指标重置以来看到的事件总数。
TotalNumberOfCreateEventsSeen	long	此连接器自上次启动或指标重置以来看到的创建事件总数。
TotalNumberOfUpdateEventsSeen	long	此连接器自上次启动或指标重置以来看到的更新事件总数。
TotalNumberOfDeleteEventsSeen	long	此连接器自上次启动或指标重置以来看到的删除事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的队列的可用容量。
Connected	布尔值	表示连接器目前是否连接到数据库服务器的标记。
MilliSecondsBehindSource	long	最后一次更改事件时间戳和连接器处理它之间的毫秒数。这些值将讨论运行数据库服务器和连接器的计算机上时钟之间的任何区别。
NumberOfCommittedTransactions	long	已提交的已处理事务的数量。
SourceEventPosition	Map<String, String>	最后收到的事件的协调。

属性	类型	描述
LastTransactionId	字符串	最后处理事务的事务的事务标识符。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

Debezium MySQL 连接器还提供以下额外流指标：

表 6.32. 其他流指标的描述

属性	类型	描述
BinlogFilename	字符串	连接器最近读取的 binlog 文件的名称。
BinlogPosition	long	连接器已读取的 binlog 中最新位置（以字节为单位）。
IsGtidModeEnabled	布尔值	表示连接器目前是否从 MySQL 服务器跟踪 GTID 的标记。
GtidSet	字符串	在读取 binlog 时连接器处理的最新 GTID 设置的字符串表示。
NumberOfSkippedEvents	long	MySQL 连接器跳过的事件数量。通常，由于来自 MySQL 的 binlog 的 malformed 或 unparseable 事件而跳过事件。
NumberOfDisconnects	long	MySQL 连接器断开连接的数量。
NumberOfRolledBackTransactions	long	已回滚且未流传输的已处理事务的数量。
NumberOfNotWellFormedTransactions	long	尚未符合预期协议的 **BEGIN** + **COMMIT**/DESTINATIONL**BACK** 的事务数量。在正常情况下，这个值应该为 **0**。

属性	类型	描述
NumberOfLargeTransactions	long	不适用于 look-ahead 缓冲区的事务数量。为获得最佳性能，这个值应显著小于 **NumberOfCommittedTransactions** 和 **NumberOfRolledBackTransactions**。

6.6.3. 监控 Debezium MySQL 连接器模式历史记录

MBean 是 debezium.mysql:type=connector-metrics,context=schema-history,server=<topic.prefix>。

下表列出了可用的模式历史记录指标。

属性	类型	描述
Status	字符串	**STOPPED** 之一，**RECOVERING**（从存储恢复历史记录），**RUNNING** 描述数据库架构历史记录的状态。
RecoveryStartTime	long	恢复启动时的 epoch 秒的时间（以秒为单位）。
ChangesRecovered	long	在恢复阶段读取的更改数量。
ChangesApplied	long	恢复和运行时期间应用的模式更改总数。
MilliSecondsSinceLastRecoveredChange	long	从历史记录存储中恢复自上次更改以来经过的毫秒数。
MilliSecondsSinceLastAppliedChange	long	从上次更改被应用后经过的毫秒数。
LastRecoveredChange	字符串	从历史记录存储中恢复的最后一个更改的字符串表示。
LastAppliedChange	字符串	最后应用的更改的字符串表示。

6.7. DEBEZIUM MYSQL 连接器如何处理错误和问题

Debezium 是一个分布式系统，用于捕获多个上游数据库中的所有更改，它不会丢失或丢失事件。当系统正常运行或谨慎管理时，Debezium 会 精确 发送每个更改事件记录。

如果出现错误，则系统不会丢失任何事件。但是，当它从错误中恢复时，可能会重复一些更改事件。在这些异常情况下，Debezium （如 Kafka）在发送更改事件时 至少 提供。

以下部分详情：

- 配置和启动错误
- MySQL 变得不可用
- Kafka Connect 正常停止
- Kafka Connect 进程崩溃
- Kafka 变得不可用
- MySQL 清除 binlog 文件

配置和启动错误

在以下情况下，连接器在尝试启动时失败，在日志中报告错误或异常，并停止运行：

- 连接器的配置无效。
- 连接器无法使用指定的连接参数成功连接到 MySQL 服务器。
- 连接器会在 binlog 中尝试重启，MySQL 不再有历史记录。

在这些情况下，错误消息包含有关问题的详情，并可能会有推荐的临时解决方案。更正配置或解决 MySQL 问题后，重启连接器。

MySQL 变得不可用

如果您的 MySQL 服务器不可用，Debezium MySQL 连接器会失败，并显示错误，连接器会停止。当服务器再次可用时，重启连接器。

但是，如果为高可用性 MySQL 集群启用了 GTID，您可以立即重启连接器。它将连接到集群中的不同 MySQL 服务器，在服务器的 binlog 中找到代表最后一个事务的位置，并开始从该特定位置读取新的服务器的 binlog。

如果没有启用 GTID，连接器只记录 MySQL 服务器的 binlog 位置。要从正确的 binlog 位置重启，您必须重新连接到该特定服务器。

Kafka Connect 正常停止

当 Kafka Connect 正常停止时，当 Debezium MySQL 连接器任务在新的 Kafka Connect 进程中停止并重启时会有一个短暂的延迟。

Kafka Connect 进程崩溃

如果 Kafka Connect 崩溃，则进程会停止，且任何 Debezium MySQL 连接器任务都会在没有记录的最新进程的偏移的情况下终止。在分布式模式中，Kafka Connect 会在其他进程上重启连接器任务。但是，MySQL 连接器会从之前进程记录的最后一个偏移中恢复。这意味着替换任务可能会生成崩溃前处理的一些相同事件，从而创建重复的事件。

每个更改事件消息都包含可用于识别重复事件的源特定信息，例如：

- 事件来源
- MySQL 服务器的事件时间
- binlog 文件名和位置
- GTID （如果使用）

Kafka 变得不可用

Kafka Connect 框架使用 Kafka producer API 记录 Kafka 中的 Debezium 更改事件。如果 Kafka 代理不可用，Debezium MySQL 连接器会暂停，直到重新建立连接，连接器恢复其关闭的位置。

MySQL 清除 binlog 文件

如果 Debezium MySQL 连接器停止了很长时间，则 MySQL 服务器会清除旧的 binlog 文件，连接器的最后位置可能会丢失。当连接器重启时，MySQL 服务器不再有起点，连接器会执行另一个初始快照。如果禁用了快照，连接器会失败并显示错误。

有关 MySQL 连接器如何执行初始 快照的详情，请查看快照。

第 7 章 ORACLE 的 DEBEZIUM CONNECTOR

Debezium 的 Oracle 连接器捕获并记录在 Oracle 服务器上的数据库中发生的行级更改，包括在连接器运行时添加的表。您可以将连接器配置为为特定模式和表的子集发出更改事件，或者在特定列中忽略、掩码或截断值。

有关与此连接器兼容的 Oracle 数据库版本的详情，请查看 Debezium 支持的配置页面。

Debezium 使用原生 LogMiner 数据库软件包更改来自 Oracle 的事件。

使用 Debezium Oracle 连接器的信息和步骤进行组织，如下所示：

7.1. DEBEZIUM ORACLE 连接器如何工作

为了优化配置和运行 Debezium Oracle 连接器，了解连接器如何执行快照、流更改事件、确定 Kafka

主题名称、使用元数据并实现事件缓冲。

如需更多信息，请参阅以下主题：

- 第 7.1.1 节 “Debezium Oracle 连接器如何执行数据库快照”
- 第 7.1.2 节 “临时快照”
- 第 7.1.3 节 “增量快照”
- 第 7.1.4 节 “接收 Debezium Oracle 更改事件记录的默认 Kafka 主题名称”
- 第 7.1.6 节 “Debezium Oracle 连接器如何公开数据库 schema 的变化”
- 第 7.1.7 节 “Debezium Oracle 连接器生成的事件代表事务边界”
- 第 7.1.8 节 “Debezium Oracle 连接器如何使用事件缓冲”

7.1.1. Debezium Oracle 连接器如何执行数据库快照

通常，Oracle 服务器上的 redo 日志配置为不保留数据库的完整历史记录。因此，Debezium Oracle 连接器无法从日志检索数据库的完整历史记录。要让连接器为数据库的当前状态建立基准，连接器首次启动时，它会执行数据库的初始 一致快照。

注意

如果完成初始快照所需的时间超过为数据库设置的 UNDO_RETENTION 时间（默认为十分钟），则可能会出现 ORA-01555 异常。有关错误以及您可以从其中恢复的步骤的更多信息，请参阅 常见问题。

您可以在以下部分找到有关快照的更多信息：

- 第 7.1.2 节 “临时快照”
- 第 7.1.3 节 “增量快照”

Oracle 连接器用来执行初始快照的默认工作流

以下工作流列出了 Debezium 创建快照所采取的步骤。这些步骤描述了当 snapshot.mode 配置属性设置为其默认值时（即 的初始 ）时快照的流程。您可以通过 changig snapshot.mode 属性的值来自定义连接器创建快照的方式。如果您配置不同的快照模式，连接器使用这个工作流的修改版本完成快照。

当快照模式被设置为默认模式时，连接器会完成以下任务来创建快照：

1. 建立与数据库的连接。
2. 确定要捕获的表。默认情况下，连接器捕获除 从捕获中排除的模式 以外的所有表。快照完成后，连接器将继续流传输指定表的数据。如果您希望连接器只从特定表捕获数据，您可以通过设置 table.include.list 或 table.exclude.list 等属性来只捕获表或表元素子集的数据。
3. 在每个捕获的表中获取 ROW SHARE MODE 锁定，以防止在创建快照过程中发生结构更改。Debezium 只保存一个短时间的锁定。
4. 从服务器的 redo 日志中读取当前系统更改号(SCN)位置。
5. 捕获所有数据库表的结构，或指定用于捕获的所有表。连接器在其内部数据库模式历史记录主题中保留模式信息。架构历史记录提供有关发生更改事件时生效的结构的信息。

注意

默认情况下，连接器捕获数据库中每个表的模式，这些模式处于捕获模式，包括没有配置为捕获的表。如果没有为捕获配置表，则初始快照只捕获其结构；它不会捕获任何表数据。有关为什么没有包括在初始快照中的表的快照保留模式信息，请参阅 了解为什么初始快照捕获所有表的 schema。

6. *释放在第 3 步中获取的锁定。其他数据库客户端现在可以写入任何之前锁定的表。*

7. *在步骤 4 中读取的 SCN 位置，连接器会扫描为捕获指定的表(SELECT * FROM … AS OF SCN 123)。在扫描过程中，连接器完成以下任务：*

   a. *确认表已在快照开始前创建。如果表是在快照启动后创建的，连接器会跳过表。快照完成后，连接器过渡到 streaming，它会发出快照开始后创建的任何表的更改事件。*

   b. *为从表获取的每行生成 读取 事件。所有 读取 事件都包含相同的 SCN 位置，这是在第 4 步中获得的 SCN 位置。*

   c. *将每个 读取 事件发送到源表的 Kafka 主题。*

   d. *释放数据表锁定（如果适用）。*

8. *在连接器偏移中记录快照成功完成。*

生成的初始快照捕获捕获捕获的表中每行的当前状态。在这个基准状态中，连接器会捕获后续更改。

在快照进程开始后，如果进程因为连接器失败、重新平衡或其他原因而中断，则进程会在连接器重启后重启。连接器完成初始快照后，它会继续从在第 3 步中读取的位置进行流，使其不会错过任何更新。如果连接器因为任何原因而再次停止，它会在重启后从之前关闭的位置恢复流更改。

表 7.1. snapshot.mode 连接器配置属性的设置

设置	描述
always	在每个连接器启动时执行快照。快照完成后，连接器开始流传输后续数据库更改的事件记录。
初始	连接器执行数据库快照，如 创建初始快照的默认工作流 中所述。快照完成后，连接器开始流传输后续数据库更改的事件记录。
initial_only	连接器在流传输任何更改事件记录前执行数据库快照并停止，不允许捕获任何后续更改事件。

设置	描述
schema_only	连接器捕获所有相关表的结构，执行 默认快照工作流 中描述的所有步骤，但它不会创建 **READ** 事件来代表连接器启动时(Step 6)时所设置的数据集。
schema_only_recovery	设置这个选项来恢复丢失或损坏的数据库架构历史记录主题。重启后，连接器会运行一个快照，它从源表中重建主题。您还可以设置该属性来定期修剪出现意外增长的数据库架构历史记录主题。 警告：如果在最后一个连接器关闭后，如果模式更改提交到数据库，则不要使用此模式来执行快照。

如需更多信息，请参阅连接器配置属性表中的 snapshot.mode。

7.1.1.1. 初始快照捕获所有表的 schema 历史记录的描述

连接器运行的初始快照捕获两种类型的信息：

表数据

在连接器的 table.include.list 属性中命名的表中的 INSERT、UPDATE 和 DELETE 操作的信息。

模式数据

描述应用到表的结构更改的 DDL 语句。模式数据会保留给内部模式历史记录主题，以及连接器的 schema 更改主题（如果配置了）。

运行初始快照后，您可能会注意到快照捕获没有指定用于捕获的表的模式信息。默认情况下，初始快照旨在捕获数据库中存在的每个表的模式信息，而不仅仅是从指定为捕获的表的表。连接器要求表的模式存在于架构历史记录主题中，然后才能捕获表。通过启用初始快照来捕获不是原始捕获集一部分的表的 schema 数据，Debebe 准备好连接器，以便稍后需要捕获这些表中的事件数据。如果初始快照没有捕获表的 schema，您必须将模式添加到历史记录主题，然后才能从表中捕获数据。

在某些情况下，您可能想要限制初始快照中的模式捕获。当您要减少完成快照所需的时间时，这非常有用。或者，当 Debezium 通过可访问多个逻辑数据库的用户帐户连接到数据库实例时，但您希望连接器只从特定逻辑数据库中的表捕获更改。

附加信息

- 从不是由初始快照捕获的表捕获数据（没有模式更改）
- 从不是由初始快照捕获的表捕获数据（应用程序更改）
- 设置 schema.history.internal.store.only.captured.tables.ddl 属性，以指定从中捕获模式信息的表。
- 设置 schema.history.internal.store.only.captured.databases.ddl 属性，以指定从中捕获模式更改的逻辑数据库。

7.1.1.2. 从不是由初始快照捕获的表捕获数据（没有模式更改）

在某些情况下，您可能希望连接器从其模式未被初始快照捕获的表中捕获数据。根据连接器配置，初始快照只能捕获数据库中特定表的表模式。如果历史记录主题中没有表模式，连接器将无法捕获表，并报告缺少的 schema 错误。

您可能仍然能够从表中捕获数据，但您必须执行额外的步骤来添加表模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。
- 事务日志中表的所有条目都使用相同的模式。有关从具有结构性更改的新表中捕获数据的详情，请参考 第 7.1.1.3 节 “从不是由初始快照捕获的表捕获数据（应用程序更改）”。

流程

1. 停止连接器。
2. 删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。
3. 在连接器配置中：
   a. 将 snapshot.mode 设置为 schema_only_recovery

将 snapshot.mode 设置为 schema_only_recovery。

b. 将 schema.history.internal.store.only.captured.tables.ddl 的值设置为 false。

c. 添加您希望连接器捕获至 table.include.list 的表。这样可保证将来，连接器可以重建所有表的 schema 历史记录。

4. 重启连接器。快照恢复过程根据表的当前结构重建模式历史记录。

5. （可选）在快照完成后，启动一个 增量快照 来捕获新添加的表的现有数据，以及该连接器关闭时发生的其他表的更改。

6. （可选）将 snapshot.mode 重置为 schema_only，以防止连接器在以后的重启后启动恢复。

7.1.1.3. 从不是由初始快照捕获的表捕获数据（应用程序更改）

如果架构更改应用到表，则在架构更改前提交的记录与更改后提交的不同结构不同。当 Debezium 从表中捕获数据时，它会读取 schema 历史记录，以确保它为每个事件应用正确的模式。如果 schema 历史记录主题中没有 schema，则连接器无法捕获表，并出现错误结果。

如果要从初始快照捕获的表中捕获数据，并且修改了表的 schema，则必须将模式添加到历史记录主题中（如果它还没有可用）。您可以通过运行新的模式快照或运行表的初始快照来添加模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。
- 架构更改应用于表，以便捕获的记录没有统一结构。

流程

初始快照捕获了所有表的模式(storage.only.captured.tables.ddl 设置为 false)

1. 编辑 table.include.list 属性，以指定您要捕获的表。

2. 重启连接器。

3. 如果要从新添加的表中捕获现有数据，则启动 增量快照。

初始快照没有捕获所有表的模式(storage.only.captured.tables.ddl 设置为 true)

如果初始快照没有保存您要捕获的表的模式，请完成以下步骤之一：

流程 1：架构快照，后跟增量快照

在此过程中，连接器首先执行 schema 快照。然后，您可以启动增量快照，使连接器能够同步数据。

1. 停止连接器。

2. 删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。

3. 清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 为连接器配置中的属性设置值，如以下步骤所述：

a. 将 snapshot.mode 属性的值设置为 schema_only。

b. 编辑 table.include.list 以添加您要捕获的表。

5. 重启连接器。

6. 等待 Debezium 捕获新表和现有表的模式。在连接器停止后发生任何表的数据更改不会被捕获。

7. 为确保没有丢失数据，请启动 增量快照。

步骤 2：初始快照，后跟可选的增量快照

在此过程中，连接器执行数据库的完整初始快照。与任何初始快照一样，在具有多个大型表的数据库中，运行初始快照可能会非常耗时。快照完成后，您可以选择触发增量快照来捕获连接器离线时发生的任何更改。

1. 停止连接器。

2. 删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。

3. 清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 编辑 table.include.list 以添加您要捕获的表。

5. 为连接器配置中的属性设置值，如以下步骤所述：

a. 将 snapshot.mode 属性的值设置为 initial。

b. （可选）将 schema.history.internal.store.only.captured.tables.ddl 设置为 false。

6. 重启连接器。连接器获取完整的数据库快照。快照完成后，连接器会过渡到 streaming。

7. （可选）要捕获连接器离线时更改的任何数据，请启动 增量快照。

7.1.2. 临时快照

默认情况下，连接器仅在首次启动后运行初始快照操作。在正常情况下，在这个初始快照后，连接器不会重复快照过程。连接器捕获的任何更改事件数据都只通过流处理。

然而，在某些情况下，连接器在初始快照期间获得的数据可能会过时、丢失或不完整。为了提供总结表数据的机制，Debezium 包含一个执行临时快照的选项。数据库中的以下更改可能会导致执行临时快照：

- 连接器配置会被修改为捕获不同的表集合。
- Kafka 主题已删除，必须重建。
- 由于配置错误或某些其他问题导致数据损坏。

您可以通过启动所谓的 临时快照来为之前捕获的表重新运行快照。临时快照需要使用 信号表。您可以通过向 Debezium 信号表发送信号请求来发起临时快照。

当您启动现有表的临时快照时，连接器会将内容附加到表已存在的主题中。如果删除了之前存在的主题，如果启用了 自动主题创建，Debezium 可以自动创建主题。

临时快照信号指定要包含在快照中的表。快照可以捕获整个数据库的内容，或者仅捕获数据库中表的

子集。另外，快照也可以捕获数据库中表的内容子集。

您可以通过将 execute-snapshot 消息发送到信号表来指定要捕获的表。将 execute-snapshot 信号类型设置为 增量，并提供快照中包含的表名称，如下表所述：

表 7.2. 临时 execute-snapshot 信号记录的示例

字段	默认	值
type	**incremental**	指定您要运行的快照类型。 设置类型是可选的。目前，您只能请求 **增量** 快照。
data-collections	*N/A*	包含与要快照的表的完全限定域名匹配的正则表达式的数组。 名称的格式与 **signal.data.collection** 配置选项的格式相同。
additional-condition	*N/A*	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。
surrogate-key	*N/A*	可选字符串，指定连接器在快照过程中用作表的主键的列名称。

触发临时快照

您可以通过向信号表中添加 execute-snapshot 信号类型的条目来发起临时快照。连接器处理消息后，它会开始快照操作。快照进程读取第一个和最后一个主密钥值，并使用这些值作为每个表的开头和结束点。根据表中的条目数量以及配置的块大小，Debezium 会将表划分为块，并一次性执行每个块的快照。

目前，execute-snapshot 操作类型仅触发 增量快照。如需更多信息，请参阅 增加快照。

7.1.3. 增量快照

为了提供管理快照的灵活性，Debezium 包含附加快照机制，称为 增量快照。增量快照依赖于 Debezium 机制 向 Debezium 连接器发送信号。

在增量快照中，除了一次捕获数据库的完整状态，就像初始快照一样，Debebe 会在一系列可配置的块中捕获每个表。您可以指定您希望快照捕获的表 以及每个块的大小。块大小决定了快照在数据库的每个获取操作期间收集的行数。增量快照的默认块大小为 1024 行。

当增量快照进行时，Debebe 使用 watermarks 跟踪其进度，维护它捕获的每个表行的记录。与标准初始快照过程相比，捕获数据的阶段方法具有以下优点：

- 您可以使用流化数据捕获并行运行增量快照，而不是在快照完成前进行后流。连接器会在快照过程中从更改日志中捕获接近实时事件，且操作都不会阻止其他操作。

- 如果增量快照的进度中断，您可以在不丢失任何数据的情况下恢复它。在进程恢复后，快照从停止的点开始，而不是从开始计算表。

- 您可以随时根据需要运行增量快照，并根据需要重复该过程以适应数据库更新。例如，您可以在修改连接器配置后重新运行快照，以将表添加到其 table.include.list 属性中。

增量快照过程

当您运行增量快照时，Debezium 会按主键对每个表进行排序，然后根据 配置的块大小 将表分成块。然后，按块的工作块会捕获块中的每个表行。对于它捕获的每行，快照会发出 READ 事件。该事件代表块的快照开始时的行值。

当快照继续进行时，其他进程可能会继续访问数据库，可能会修改表记录。为了反映此类更改，INSERT、UPDATE 或 DELETE 操作会按照常常提交到事务日志。同样，持续 Debezium 流进程将继续检测这些更改事件，并将相应的更改事件记录发送到 Kafka。

Debezium 如何使用相同的主密钥在记录间解决冲突

在某些情况下，streaming 进程发出的 UPDATE 或 DELETE 事件会停止序列。也就是说，流流过程可能会发出一个修改表行的事件，该事件捕获包含该行的 READ 事件的块。当快照最终为行发出对应的 READ 事件时，其值已被替换。为确保以正确的逻辑顺序处理到达序列的增量快照事件，Debebe 使用缓冲方案来解析冲突。仅在快照事件和流化事件之间发生冲突后，De Debezium 会将事件记录发送到 Kafka。

快照窗口

为了帮助解决修改同一表行的后期事件和流化事件之间的冲突，Debebe 会使用一个所谓的 快照窗口。快照窗口分解了增量快照捕获指定表块数据的间隔。在块的快照窗口打开前，Debebe 会使用其常见行为，并将事件从事务日志直接下游发送到目标 Kafka 主题。但从特定块的快照打开后，直到关闭为止，De-duplication 步骤会在具有相同主密钥的事件之间解决冲突。

对于每个数据收集，Debezium 会发出两种类型的事件，并将其存储在单个目标 Kafka 主题中。从表直接捕获的快照记录作为 READ 操作发送。同时，当用户继续更新数据收集中的记录，并且会更新事务日志来反映每个提交，Debezium 会为每个更改发出 UPDATE 或 DELETE 操作。

当快照窗口打开时，Debezium 开始处理快照块，它会向内存缓冲区提供快照记录。在快照窗口期间，缓冲区中 READ 事件的主密钥与传入流事件的主键进行比较。如果没有找到匹配项，则流化事件记录将直接发送到 Kafka。如果 Debezium 检测到匹配项，它会丢弃缓冲的 READ 事件，并将流化记录写

入目标主题，因为流的事件逻辑地取代静态快照事件。在块关闭的快照窗口后，缓冲区仅包含 READ 事件，这些事件不存在相关的事务日志事件。Debezium 将这些剩余的 READ 事件发送到表的 Kafka 主题。

连接器为每个快照块重复这个过程。

警告

Oracle 的 Debezium 连接器不支持增量快照运行时的模式更改。

7.1.3.1. 触发增量快照

目前，启动增量快照的唯一方法是向源数据库上的 信号表发送临时快照 信号。

作为 SQL INSERT 查询，您将向信号提交信号。

在 Debezium 检测到信号表中的更改后，它会读取信号并运行请求的快照操作。

您提交的查询指定要包含在快照中的表，并可以选择指定快照操作的类型。目前，快照操作的唯一有效选项是默认值 incremental。

要指定快照中包含的表，请提供列出表或用于匹配表的正则表达式数组的 数据集合，例如：

```
{"data-collections": ["public.MyFirstTable", "public.MySecondTable"]}
```

增量快照信号的 data-collections 数组没有默认值。如果 data-collections 数组为空，Debezium 会检测到不需要任何操作，且不会执行快照。

注意

如果要包含在快照中的表的名称在数据库、模式或表的名称中包含句点(.)，以将表添加到 data-collections 数组中，您必须使用双引号转义名称的每个部分。

例如，要包含一个存在于 公共 模式的表，其名称为 My.Table，请使用以下格式：\"public\".\"My.Table \"。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道来触发增量快照

1. 发送 SQL 查询，将临时增量快照请求添加到信号表中：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) (1)
values ('ad-hoc-1', (2)
  'execute-snapshot', (3)
  '{"data-collections": ["schema1.table1", "schema2.table2"], (4)
  "type":"incremental"}, (5)
  "additional-condition":"color=blue"}'); (6)
```

命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 7.3. SQL 命令中字段的描述，用于将增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。相反，Debebe 会在快照期间生成自己的 **id** 字符串作为水位线信号。
3	**execute-snapshot**	**type** 参数指定信号旨在触发的操作。
4	**data-collections**	信号的 **data** 字段所需的组件，用于指定表名称或正则表达式数组，以匹配快照中包含的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。
5	**incremental**	信号的 **data** 字段的可选 **类型** 组件，用于指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 如果没有指定值，连接器将运行增量快照。
6	**additional-condition**	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。有关 **additional-condition** 参数的更多信息，请参阅 **带有额外条件的临时增量快照**。

带有额外条件的临时增量快照

如果您希望快照只包含表中的内容子集，您可以通过向快照信号附加 additional-condition 参数来修改信号请求。

典型的快照的 SQL 查询采用以下格式：

```
SELECT * FROM <tableName> ....
```

通过添加 additional-condition 参数，您可以将 WHERE 条件附加到 SQL 查询中，如下例所示：

```
SELECT * FROM <tableName> WHERE <additional-condition> ....
```

以下示例显示了向信号表发送带有额外条件的临时增量快照请求的 SQL 查询：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，假设您有一个包含以下列的 products 表：

- ID （主键）
- color
- quantity

如果您需要 product 表的增量快照，其中只包含 color=blue 的数据项，您可以使用以下 SQL 语句来触发快照：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue"}');
```

additional-condition 参数还允许您传递基于多个列的条件。例如，使用上例中的 product 表，您可以提交查询来触发增量快照，该快照仅包含 color=blue 和 quantity>10 的项数据：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue AND quantity>10"}');
```

以下示例显示了连接器捕获的增量快照事件的 JSON。

示例：增加快照事件消息

```
{
  "before":null,
  "after": {
    "pk":"1",
    "value":"New data"
  },
  "source": {
    ...
```

```
        "snapshot":"incremental" 1
    },
    "op":"r", 2
    "ts_ms":"1620393591654",
    "transaction":null
}
```

项	字段名称	描述
1	**snapshot**	指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 在 SQL 查询中指定**type** 值，您提交到信号表是可选的。 如果没有指定值，连接器将运行增量快照。
2	**op**	指定事件类型。 快照事件的值是 **r**，表示 **READ** 操作。

7.1.3.2. 使用 Kafka 信号频道来触发增量快照

您可以向 配置的 Kafka 主题 发送消息，以请求连接器来运行临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 execute-snapshot，data 字段必须具有以下字段：

表 7.4. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与快照中包含的表的完全限定域名匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

字段	默认	值
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的列子集的条件。

execute-snapshot Kafka 消息示例：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

带有额外条件的临时增量快照

Debezium 使用 additional-condition 字段来选择表内容的子集。

通常，当 Debezium 运行快照时，它会运行 SQL 查询，例如：

SELECT * FROM <tableName> ....

当快照请求包含 additional-condition 时，extra-condition 会附加到 SQL 查询中，例如：

SELECT * FROM <tableName> WHERE <additional-condition> ....

例如，如果一个 product table with the column id （主键）、color 和 brand，如果您希望快照只包含 color='blue' 的内容，当您请求快照时，您可以附加一个 additional-condition 语句来过滤内容：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type": "INCREMENTAL", "additional-condition":"color='blue'"}}`
```

您可以使用 additional-condition 语句根据多个列传递条件。例如，如果您希望快照只包含 color='blue' 的 products 表中，以及 brand='MyBrand'，则您可以发送以下请求：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
```

```
"INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}`
```

7.1.3.3. 停止增量快照

您还可以通过向源数据库上的表发送信号来停止增量快照。您可以通过发送 SQL INSERT 查询向表提交停止快照信号。

在 Debezium 检测到信号表中的更改后，它会读取信号，并在正在进行时停止增量快照操作。

您提交的查询指定 增量 的快照操作，以及要删除的当前运行快照的表。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道停止增量快照

1. 发送 SQL 查询以停止临时增量快照到信号表：

```
INSERT INTO <signalTable> (id, type, data) values ('<id>', 'stop-snapshot', '{"data-collections": ["<tableName>","<tableName>"],"type":"incremental"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) 1
values ('ad-hoc-1', 2
  'stop-snapshot', 3
  '{"data-collections": ["schema1.table1", "schema2.table2"], 4
  "type":"incremental"}'); 5
```

signal 命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 7.5. SQL 命令中字段的描述，用于将停止增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。 使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。
3	**stop-snapshot**	指定 **type** 参数指定信号要触发的操作。
4	**data-collections**	信号的 **data** 字段的可选组件，用于指定表名称或正则表达式数组，以匹配要从快照中删除的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。如果省略了 **data** 字段的这一组件，信号将停止正在进行的整个增量快照。
5	**incremental**	信号的 **data** 字段所需的组件，用于指定要停止的快照操作类型。 目前，唯一有效的选项是 **增量的**。 如果没有指定 **类型** 值，信号将无法停止增量快照。

7.1.3.4. 使用 Kafka 信号频道停止增量快照

您可以将信号消息发送到 配置的 Kafka 信号主题，以停止临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 stop-snapshot，data 字段必须具有以下字段：

表 7.6. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。

字段	默认	值
data-collections	*N/A*	可选数组，以逗号分隔的正则表达式，与表的完全限定域名匹配，以包含在快照中。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

以下示例显示了典型的 stop-snapshot Kafka 信息：

```
Key = `test_connector`

Value = `{"type":"stop-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

7.1.4. 接收 Debezium Oracle 更改事件记录的默认 Kafka 主题名称

默认情况下，Oracle 连接器将所有 INSERT、UPDATE 和 DELETE 操作的更改事件写入特定于该表的单一 Apache Kafka 主题。连接器使用以下惯例来命名更改事件主题：

topicPrefix.schemaName.tableName

以下列表为默认名称的组件提供定义：

topicPrefix

由 topic.prefix 连接器配置属性指定的主题前缀。

schemaName

操作所在的模式的名称。

tableName

操作所在的表的名称。

例如，如果 fulfillment 是服务器名称，inventory 是 schema 名称，数据库包括名为 orders, customers, 和 products 的表，Debezium Oracle 连接器会向以下 Kafka 主题发送事件，数据库中的每个表有一个。

```
fulfillment.inventory.orders
fulfillment.inventory.customers
fulfillment.inventory.products
```

连接器应用类似的命名约定，以标记其内部数据库架构历史记录主题、架构更改主题 和事务元数据主题。

如果默认主题名称不满足您的要求，您可以配置自定义主题名称。要配置自定义主题名称，您可以在逻辑主题路由 SMT 中指定正则表达式。有关使用逻辑主题路由 SMT 来自定义主题命名的更多信息，请参阅 主题路由。

7.1.5. Debezium Oracle 连接器如何处理数据库架构更改

当数据库客户端查询数据库时，客户端将使用数据库的当前架构。但是，数据库模式可以随时更改，这意味着连接器必须能够识别每个插入、更新或删除操作被记录的时间。另外，连接器不一定将当前的模式应用到每个事件。如果事件相对旧，则应用当前模式之前可能会记录该事件。

为确保在架构更改后正确处理事件，Oracle 包含在红色日志中，不仅影响数据的行级更改，还应用于数据库的 DDL 语句。当连接器在红色日志中遇到这些 DDL 语句时，它会解析它们并更新每个表模式的内存表示。连接器使用此模式表示来识别每个插入、更新或删除操作时表的结构，并生成适当的更改事件。在单独的数据库架构历史记录 Kafka 主题中，连接器记录所有 DDL 语句，以及在红色日志中记录每个 DDL 语句的位置。

当连接器在崩溃或安全停止后重启时，它从特定位置（即时间点）开始读取 redo 日志。连接器通过读取数据库模式历史记录 Kafka 主题，并将所有 DDL 语句解析为连接器启动的红色日志点，以此重建此时存在的表结构。

此数据库架构历史记录主题为内部连接器，仅用于内部连接器。另外，连接器也可以将 模式更改事件发送到用于消费者应用程序的不同主题。

其他资源

- 接收 Debezium 事件记录 的主题的默认名称。

7.1.6. Debezium Oracle 连接器如何公开数据库 schema 的变化

您可以配置 Debezium Oracle 连接器来生成模式更改事件，该事件描述了应用到数据库中表的结构更改。连接器将模式更改事件写入名为 < serverName> 的 Kafka 主题，其中 topicName 是 topic.prefix 配置属性中指定的命名空间。

当 Debezium 从新表中流数据或更改表结构时，Debezium 会向 schema 更改主题发送一个新消息。

连接器发送到 schema 更改主题的消息包含一个有效负载，以及可选的包含更改事件消息的 schema。模式更改事件消息的有效负载包括以下元素：

ddl

提供会导致架构更改的 SQL CREATE、ALTER 或 DROP 语句。

databaseName

将语句应用到的数据库的名称。databaseName 的值充当 message 键。

tableChanges

架构更改后整个表模式的结构化表示。tableChanges 字段包含一个数组，其中包含表的每个列的条目。由于结构化表示以 JSON 或 Avro 格式呈现数据，因此用户可轻松读取消息，而不必先通过 DDL 解析器处理它们。

重要

默认情况下，连接器使用 ALL_TABLES 数据库视图来识别要存储在 schema 历史记录主题中的表名称。在该视图中，连接器只能访问可通过它连接到数据库的用户帐户的表中的数据。

您可以修改设置，以便 schema 历史记录主题存储不同的表子集。使用以下方法之一更改主题存储的表集合：

- 更改 Debezium 用于访问数据库的帐户的权限，以便在 ALL_TABLES 视图中查看不同的表集合。
- 将 connector 属性 schema.history.internal.store.only.captured.tables.ddl 设置为 true。

重要

当连接器配置为捕获表时，它只会在 schema 更改主题中存储表的历史记录，也存储在内部数据库 schema 历史记录主题中。内部数据库架构历史记录主题仅用于连接器，它不适用于消耗应用程序直接使用。确保需要通知架构更改的应用程序只消耗来自 schema 更改主题的信息。

重要

切勿对数据库架构历史记录主题进行分区。要使数据库架构历史记录主题正常工作，它必须维护连接器发出的事件记录的全局顺序。

要确保主题没有在分区间分割，请使用以下方法之一为主题设置分区计数：

- 如果您手动创建数据库架构历史记录主题，请指定分区计数 1。
- 如果您使用 Apache Kafka 代理自动创建数据库 schema 历史记录主题，则会创建该主题，将 Kafka num.partitions配置选项 的值设置为 1。

示例：消息发送到 Oracle 连接器模式更改主题

以下示例显示了 JSON 格式的典型的模式更改消息。该消息包含表模式的逻辑表示。

```
{
 "schema": {
 ...
 },
 "payload": {
  "source": {
   "version": "2.3.4.Final",
   "connector": "oracle",
   "name": "server1",
   "ts_ms": 1588252618953,
   "snapshot": "true",
   "db": "ORCLPDB1",
   "schema": "DEBEZIUM",
   "table": "CUSTOMERS",
   "txId" : null,
   "scn" : "1513734",
   "commit_scn": "1513754",
   "lcr_position" : null,
   "rs_id": "001234.00012345.0124",
   "ssn": 1,
   "redo_thread": 1,
```

```
    "user_name": "user"
  },
  "ts_ms": 1588252618953, (1)
  "databaseName": "ORCLPDB1", (2)
  "schemaName": "DEBEZIUM", //
  "ddl": "CREATE TABLE \"DEBEZIUM\".\"CUSTOMERS\" \n   (    \"ID\" NUMBER(9,0) NOT
NULL ENABLE, \n    \"FIRST_NAME\" VARCHAR2(255), \n    \"LAST_NAME" VARCHAR2(255),
\n    \"EMAIL\" VARCHAR2(255), \n     PRIMARY KEY (\"ID\") ENABLE, \n     SUPPLEMENTAL
LOG DATA (ALL) COLUMNS\n   ) SEGMENT CREATION IMMEDIATE \n  PCTFREE 10
PCTUSED 40 INITRANS 1 MAXTRANS 255 \n NOCOMPRESS LOGGING\n  STORAGE(INITIAL
65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645\n  PCTINCREASE 0
FREELISTS 1 FREELIST GROUPS 1\n  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT
CELL_FLASH_CACHE DEFAULT)\n  TABLESPACE \"USERS\" ", (3)
  "tableChanges": [ (4)
    {
      "type": "CREATE", (5)
      "id": "\"ORCLPDB1\".\"DEBEZIUM\".\"CUSTOMERS\"", (6)
      "table": { (7)
        "defaultCharsetName": null,
        "primaryKeyColumnNames": [ (8)
          "ID"
        ],
        "columns": [ (9)
          {
            "name": "ID",
            "jdbcType": 2,
            "nativeType": null,
            "typeName": "NUMBER",
            "typeExpression": "NUMBER",
            "charsetName": null,
            "length": 9,
            "scale": 0,
            "position": 1,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "FIRST_NAME",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "VARCHAR2",
            "typeExpression": "VARCHAR2",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 2,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "LAST_NAME",
            "jdbcType": 12,
            "nativeType": null,
```

```
            "typeName": "VARCHAR2",
            "typeExpression": "VARCHAR2",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 3,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "EMAIL",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "VARCHAR2",
            "typeExpression": "VARCHAR2",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 4,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          }
        ],
        "attributes": [ 10
          {
            "customAttribute": "attributeValue"
          }
        ]
      }
    }
  ]
 }
}
```

表 7.7. 向 schema 更改主题发送的消息中字段的描述

项	字段名称	描述
1	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 在源对象中，ts_ms 表示数据库中进行更改的时间。通过将 payload.source.ts_ms 的值与 payload.ts_ms 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
2	**databaseName** **schemaName**	标识包含更改的数据库和架构。
3	**ddl**	此字段包含负责架构更改的 DDL。

项	字段名称	描述
4	**tableChanges**	包含 DDL 命令生成的模式更改的一个或多个项目的数组。
5	**type**	描述更改的类型。**type** 可以设置为以下值之一： **创建** 已创建的表。 **更改** 修改表。 **DROP** 表已删除。
6	**id**	创建、更改或丢弃的表的完整标识符。如果是表重命名，这个标识符是 < ***old>、<new>*** 表名称的串联。
7	**table**	代表应用更改后的表元数据。
8	**primaryKeyColumnNames**	组成表主密钥的列的列表。
9	**columns**	更改表中每个列的元数据。
10	**属性**	每个表更改的自定义属性元数据。

在连接器发送到 schema 更改主题的消息中，message 键是包含 schema 更改的数据库的名称。在以下示例中，payload 字段包含 databaseName 键：

```
{
  "schema": {
    "type": "struct",
    "fields": [
      {
        "type": "string",
        "optional": false,
        "field": "databaseName"
      }
    ],
    "optional": false,
    "name": "io.debezium.connector.oracle.SchemaChangeKey"
  },
  "payload": {
    "databaseName": "ORCLPDB1"
  }
}
```

7.1.7. Debezium Oracle 连接器生成的事件代表事务边界

Debezium 可以生成代表事务元数据边界的事件，并增强数据更改事件消息。

DEBEZIUM 接收事务元数据时的限制

Debezium 注册并只针对部署连接器后发生的事务接收元数据。部署连接器前发生的事务元数据不可用。

数据库事务由语句块表示，该块包含在 BEGIN 和 END 关键字之间。Debezium 为每个事务中的 BEGIN 和 END 分隔符生成事务边界事件。事务边界事件包含以下字段：

status

BEGIN 或 END.

id

唯一事务标识符的字符串。

ts_ms

数据源的事务边界事件(BEGIN 或 END 事件)的时间。如果数据源没有向事件时间提供 Debezium，则该字段代表 Debezium 处理事件的时间。

event_count （用于 END 事件）

事务提供的事件总数。

data_collections （用于 END 事件）

data_collection 和 event_count 元素的数组，用于指示连接器发出来自数据收集的更改的事件数量。

以下示例显示了典型的事务边界消息：

示例：Oracle 连接器事务边界事件

```
{
  "status": "BEGIN",
```

```
  "id": "5.6.641",
  "ts_ms": 1486500577125,
  "event_count": null,
  "data_collections": null
}

{
  "status": "END",
  "id": "5.6.641",
  "ts_ms": 1486500577691,
  "event_count": 2,
  "data_collections": [
    {
      "data_collection": "ORCLPDB1.DEBEZIUM.CUSTOMER",
      "event_count": 1
    },
    {
      "data_collection": "ORCLPDB1.DEBEZIUM.ORDER",
      "event_count": 1
    }
  ]
}
```

除非通过 topic.transaction 选项覆盖，否则连接器会将事务事件发送到 < topic.prefix>.transaction 主题。

7.1.7.1. Debezium Oracle 连接器如何使用事务元数据增强更改事件信息

如果启用了事务元数据，数据消息 Envelope 会增加一个新的 transaction 字段。此字段以字段复合的形式提供有关每个事件的信息：

id

唯一事务标识符的字符串。

total_order

事件在事务生成的所有事件中绝对位置。

data_collection_order

在事务发出的所有事件间，按数据收集位置。

以下示例显示了典型的事务事件信息：

```
{
  "before": null,
  "after": {
    "pk": "2",
    "aa": "1"
  },
  "source": {
...
  },
  "op": "c",
  "ts_ms": "1580390884335",
  "transaction": {
    "id": "5.6.641",
    "total_order": "1",
    "data_collection_order": "1"
  }
}
```

查询模式

Debezium Oracle 连接器默认与 Oracle LogMiner 集成。此集成需要一组专用的步骤，其中包括生成复杂的 JDBC SQL 查询，以评估事务日志中记录的更改作为更改事件。JDBC SQL 查询使用的 V$LOGMNR_CONTENTS 视图没有任何索引来改进查询的性能，因此有不同的查询模式来控制 SQL 查询的生成方式，以改进查询的执行方式。

log.mining.query.filter.mode 连接器属性可使用以下任一方式配置 JDBC SQL 查询：

none

（默认）此模式会创建一个 JDBC 查询，该查询仅根据不同操作类型（如插入、更新或删除）过滤到数据库级别。当根据 schema, table, 或 username include/exclude 列表过滤数据时，会在连接器中的处理循环中完成此操作。

当从数据库捕获少量表时，这个模式通常很有用，这些表没有大量更改饱和。生成的查询非常简单，主要以低数据库开销尽快阅读。

in

此模式创建 JDBC 查询，该查询不仅过滤在数据库级别上的操作类型，还创建 schema、表和用户名 include/exclude 列表。查询的 predicates 会根据 include/exclude 列表配置属性中指定的值使用 SQL in-clause 生成。

从数据库中捕获大量表时，这个模式通常很有用。生成的查询比 none 模式要复杂得多，它侧重于减少网络开销，并尽可能在数据库级别上执行一次过滤。

最后，不要将 正则表达式指定为 schema 和 table include/exclude 配置属性的一部分。使用正则表达式将导致连接器与基于这些配置属性的更改不匹配，从而导致更改丢失。

regex

此模式创建 JDBC 查询，该查询不仅过滤在数据库级别上的操作类型，还创建 schema、表和用户名 include/exclude 列表。但是，与 in 模式不同，这个模式使用 Oracle REGEXP_LIKE 运算符生成 SQL 查询，具体取决于是否指定了 include 或 exclude 值。

当捕获可以使用少量正则表达式来标识的表数时，此模式通常很有用。生成的查询比任何其他模式更复杂，它侧重于减少网络开销，并尽可能在数据库级别执行一次过滤。

7.1.8. Debezium Oracle 连接器如何使用事件缓冲

Oracle 按照它们发生的顺序将所有更改写入 redo 日志，包括回滚后丢弃的更改。因此，来自独立事务的并发更改会被干预。当连接器首次读取更改流时，因为它无法立即决定提交或回滚哪些更改，它会临时将更改事件存储在内部缓冲区中。提交更改后，连接器会将更改事件从缓冲区写入 Kafka。连接器丢弃回滚丢弃的更改事件。

您可以通过设置属性 log.mining.buffer.type 来配置连接器使用的缓冲机制。

Heap

默认缓冲区类型使用 memory 进行配置。在默认 内存设置 下，连接器使用 JVM 进程的堆内存来分配和管理缓冲的事件记录。如果使用 内存 缓冲区设置，请确保分配给 Java 进程的内存量可以容纳环境中的长时间运行和大型事务。

7.1.9. Debezium Oracle 连接器如何检测 SCN 值中的差距

当 Debezium Oracle 连接器被配置为使用 LogMiner 时，它会使用基于系统更改号(SCN)的开始和结束范围从 Oracle 收集更改事件。连接器会自动管理这个范围，根据连接器是否能够流向实时流更改，或者因为数据库中大或批量事务的卷而处理更改，以自动增加或减少范围。

在某些情况下，Oracle 数据库会以一个非常高的数量来提升 SCN，而不是以恒定率增加 SCN 值。由于特定集成与数据库交互方式或出现热备份等事件，所以可能会出现 SCN 值的跳过。

Debezium Oracle 连接器依赖于以下配置属性来检测 SCN 差距并调整 mining 范围。

log.mining.scn.gap.detection.gap.size.min

指定最小空白大小。

log.mining.scn.gap.detection.time.interval.max.ms

指定最大时间间隔。

连接器首先比较当前 SCN 和当前 mining 范围内的最高 SCN 之间的变化数量。如果当前 SCN 值和最高 SCN 值之间的区别大于最小空白大小，则连接器可能会检测到 SCN 差距。要确认是否存在差距，连接器会随后比较当前 SCN 和 SCN 在前一个 mining 范围末尾的时间戳。如果时间戳之间的区别小于最大时间间隔，则确认存在 SCN 差距。

当发生 SCN 差距时，Debezium 连接器会自动使用当前的 SCN 作为当前 mining 会话范围的端点。这允许连接器在没有返回任何更改之间快速捕获实时事件，因为 SCN 值意外增加。当连接器执行前面的步骤以响应 SCN 差距时，它会忽略 log.mining.batch.size.max 属性指定的值。在连接器完成 mining 会话并捕获到实时事件后，它会恢复最大日志最小批处理大小的强制。

警告

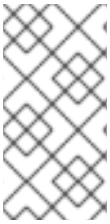
只有在连接器运行和处理接近实时事件时，才会提供 SCN 差距检测。

7.1.10. Debezium 如何管理数据库中不经常更改的偏移量

Debezium Oracle 连接器跟踪连接器偏移中的系统更改号，以便在连接器重启时，它可以从离开的位置开始。这些偏移是每个发出的更改事件的一部分；但是，当数据库更改的频率较低（几小时或天）时，偏移可能会过时，并阻止连接器在事务日志中不再可用时成功重启。

对于使用非CDB 模式连接到 Oracle 的连接器，您可以启用 heartbeat.interval.ms 来强制连接器定期发出心跳事件，以便偏移保持同步。

对于使用 CDB 模式连接到 Oracle 的连接器，维护同步更为复杂。不仅必须设置 heartbeat.interval.ms，还需要设置 heartbeat.action.query。需要指定这两个属性，因为在 CDB 模式中，连接器仅跟踪 PDB 中的更改。需要补充机制来从可插拔数据库内触发更改事件。定期，心跳操作查询会导致连接器插入一个新的表行，或者在可插拔数据库中更新现有行。Debezium 检测到表更改，并为它们发出更改事件，确保偏移保持同步，即使在可插拔数据库中，进程不经常更改。



注意

要使连接器使用 heartbeat.action.query 与 连接器用户帐户 所有的表，您必须授予连接器用户权限来对这些表运行必要的 INSERT 或 UPDATE 查询。

7.2. DEBEZIUM ORACLE 连接器数据更改事件的描述

每个数据更改事件，而 Oracle 连接器发出的事件都有一个键和值。键和值的结构取决于更改事件源自的表。有关 Debezium 如何构造主题名称的信息，请参阅 主题名称。

警告

Debezium Oracle 连接器确保所有 Kafka Connect 模式名称 是有效的 Avro 模式名称。这意味着逻辑服务器名称必须以字母字符或下划线([a-z,A-Z,_])以及逻辑服务器名称中的剩余字符以及模式中的所有字符([a-z,A-Z,0-9,_])开头。连接器会自动将无效字符替换为下划线字符。

当多个逻辑服务器名称、模式名称或表名称之间没有有效字符且这些字符被下划线替换时，这些命名冲突可能会导致。

Debezium 和 Kafka Connect 围绕 事件消息的持续流 设计。但是，这些事件的结构可能会随时间推移而改变，因此主题消费者很难处理。为了便于处理 mutable 事件结构，Kafka Connect 中的每个事件都是自包含的。每个消息键和值有两个部分：schema 和 payload。架构描述了有效负载的结构，而有效负载包含实际数据。

警告

SYS 或 SYSTEM 用户帐户执行的更改不会被连接器捕获。

以下主题包含有关数据更改事件的更多详情：

- 第 7.2.1 节 “关于 Debezium Oracle 连接器更改事件中的键”
- 第 7.2.2 节 “关于 Debezium Oracle 连接器更改事件中的值”

7.2.1. 关于 Debezium Oracle 连接器更改事件中的键

对于每个更改的表，更改事件键是结构化的，以便在创建事件时，主键（或唯一键约束）中的每个列都有一个字段。

例如，在 inventory 数据库 schema 中定义的 customers 表，可能有以下更改事件键：

```
CREATE TABLE customers (
  id NUMBER(9) GENERATED BY DEFAULT ON NULL AS IDENTITY (START WITH 1001) NOT NULL PRIMARY KEY,
  first_name VARCHAR2(255) NOT NULL,
  last_name VARCHAR2(255) NOT NULL,
  email VARCHAR2(255) NOT NULL UNIQUE
);
```

如果 < topic.prefix>.transaction 配置属性被设置为 server1，则数据库表中的 customer 表中发生的每个更改事件的 JSON 表示如下关键结构：

```
{
    "schema": {
        "type": "struct",
        "fields": [
            {
                "type": "int32",
                "optional": false,
                "field": "ID"
            }
        ],
        "optional": false,
        "name": "server1.INVENTORY.CUSTOMERS.Key"
    },
    "payload": {
        "ID": 1004
    }
}
```

键的 schema 部分包含一个 Kafka Connect 模式，它描述了 key 部分的内容。在前面的示例中，有效负载 值不是可选，该结构由名为 server1.DEBEZIUM.CUSTOMERS.Key 的 schema 定义，并且有一

个名为 int32 的 id 的必要字段。键的 payload 字段的值表示它实际上是一个带有 id 字段的结构（在 JSON 中，是一个对象），其值为 1004。

因此，您可以将这个键解释为 inventory.customers 表中的行（来自名为 server1 的连接器），其 id 主键列的值为 1004。

7.2.2. 关于 Debezium Oracle 连接器更改事件中的值

更改事件消息中值的结构反映了消息中更改事件中的 message 键 结构，并包含 schema 部分和 payload 部分。

更改事件值的有效负载

更改事件值的有效数据部分中有一个 envelope 数据结构，它包含以下字段：

op

包含用于描述操作类型的字符串值的必需字段。Oracle 连接器更改事件值中的 op 字段包含以下值之一：c （创建或插入）、u （更新）、d （删除）或 r （表示快照）。

before

可选字段（如果存在）描述事件 发生前 行的状态。该结构由 server1.INVENTORY.CUSTOMERS.Value Kafka Connect 模式描述，server1 连接器用于 inventory.customers 表中的所有行。

after

可选字段（如果存在）在发生 更改后 包含行的状态。该结构由用于 before 字段的同一 server1.INVENTORY.CUSTOMERS.Value Kafka Connect 模式描述。

source

包含描述事件源元数据的结构的必填字段。对于 Oracle 连接器，结构包括以下字段：

- Debezium 版本。
- 连接器名称。

- 事件是持续快照的一部分。
- 事务 ID （不包括快照）。
- 变化的 SCN。
- 指示源数据库中记录何时更改（对于快照，时间戳表示快照何时发生）。
- 进行更改的用户名

提示

commit_scn 字段是可选的，描述了更改事件参与的事务提交的 SCN。

ts_ms

可选字段（如果存在），其中包含运行 Kafka Connect 任务的 JVM 中的系统时钟（基于连接器处理该事件）的时间（基于时钟）。

更改事件值的 schema

事件消息的 schema 部分包含一个 schema，用于描述有效负载的信封结构及其其中嵌套字段。

有关更改事件值的更多信息，请参阅以下主题：

- 创建 事件
- 更新 事件
- 删除 事件

- *Truncate 事件*

创建 事件

以下示例显示了来自 change 事件键示例中描述的 customer 表中 create event 值的值：

```
{
    "schema": {
        "type": "struct",
        "fields": [
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "int32",
                        "optional": false,
                        "field": "ID"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "FIRST_NAME"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "LAST_NAME"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "EMAIL"
                    }
                ],
                "optional": true,
                "name": "server1.DEBEZIUM.CUSTOMERS.Value",
                "field": "before"
            },
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "int32",
                        "optional": false,
                        "field": "ID"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "FIRST_NAME"
                    },
                    {
```

```
                "type": "string",
                "optional": false,
                "field": "LAST_NAME"
            },
            {
                "type": "string",
                "optional": false,
                "field": "EMAIL"
            }
        ],
        "optional": true,
        "name": "server1.DEBEZIUM.CUSTOMERS.Value",
        "field": "after"
    },
    {
        "type": "struct",
        "fields": [
            {
                "type": "string",
                "optional": true,
                "field": "version"
            },
            {
                "type": "string",
                "optional": false,
                "field": "name"
            },
            {
                "type": "int64",
                "optional": true,
                "field": "ts_ms"
            },
            {
                "type": "string",
                "optional": true,
                "field": "txId"
            },
            {
                "type": "string",
                "optional": true,
                "field": "scn"
            },
            {
                "type": "string",
                "optional": true,
                "field": "commit_scn"
            },
            {
                "type": "string",
                "optional": true,
                "field": "rs_id"
            },
            {
                "type": "int64",
                "optional": true,
                "field": "ssn"
```

```
            },
            {
                "type": "int32",
                "optional": true,
                "field": "redo_thread"
            },
            {
                "type": "string",
                "optional": true,
                "field": "user_name"
            },
            {
                "type": "boolean",
                "optional": true,
                "field": "snapshot"
            }
          ],
          "optional": false,
          "name": "io.debezium.connector.oracle.Source",
          "field": "source"
        },
        {
          "type": "string",
          "optional": false,
          "field": "op"
        },
        {
          "type": "int64",
          "optional": true,
          "field": "ts_ms"
        }
      ],
      "optional": false,
      "name": "server1.DEBEZIUM.CUSTOMERS.Envelope"
    },
    "payload": {
      "before": null,
      "after": {
          "ID": 1004,
          "FIRST_NAME": "Anne",
          "LAST_NAME": "Kretchmar",
          "EMAIL": "annek@noanswer.org"
      },
      "source": {
          "version": "2.3.4.Final",
          "name": "server1",
          "ts_ms": 1520085154000,
          "txId": "6.28.807",
          "scn": "2122185",
          "commit_scn": "2122185",
          "rs_id": "001234.00012345.0124",
          "ssn": 1,
          "redo_thread": 1,
          "user_name": "user",
          "snapshot": false
      },
```

```
        "op": "c",
        "ts_ms": 1532592105975
    }
}
```

在上例中，请注意事件如何定义以下模式：

- envelope (server1.DEBEZIUM.CUSTOMERS.Envelope)。
- 源 结构（io.debezium.connector.oracle.Source，它特定于 Oracle 连接器并在所有事件间重复使用。
- before 和 after 字段的特定于表的模式。

提示

before 和 after 字段的 schema 的名称的格式为 <logicalName>.<schemaName>.<tableName>.Value, 因此完全独立与所有其他表的 schema。因此，当您使用 Avro converter 时，每个逻辑源中表的 Avro 模式都有自己的演进和历史记录。

此事件的 value 的 payload 部分提供有关事件的信息。它描述了已创建了行(op=c)，并显示 after 字段值，其中包含插入到 ID、FIRST_NAME、LAST_NAME 和 EMAIL 列的值。

提示

默认情况下，事件的 JSON 表示比它们描述的行大得多。较大的大小是由于 JSON 表示，包括消息的 schema 和 payload 部分。您可以使用 Avro Converter 来缩小连接器写入 Kafka 主题的信息大小。

更新 事件

以下示例显示了一个 update 更改事件，连接器从与以前的 create 事件相同的表中捕获。

```
{
    "schema": { ... },
    "payload": {
        "before": {
            "ID": 1004,
            "FIRST_NAME": "Anne",
            "LAST_NAME": "Kretchmar",
```

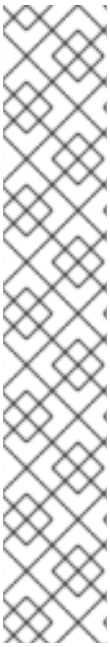
```
        "EMAIL": "annek@noanswer.org"
      },
      "after": {
        "ID": 1004,
        "FIRST_NAME": "Anne",
        "LAST_NAME": "Kretchmar",
        "EMAIL": "anne@example.com"
      },
      "source": {
        "version": "2.3.4.Final",
        "name": "server1",
        "ts_ms": 1520085811000,
        "txId": "6.9.809",
        "scn": "2125544",
        "commit_scn": "2125544",
        "rs_id": "001234.00012345.0124",
        "ssn": 1,
        "redo_thread": 1,
        "user_name": "user",
        "snapshot": false
      },
      "op": "u",
      "ts_ms": 1532592713485
    }
}
```


有效负载的结构与 create （插入）事件的有效负载相同，但以下值不同：

- op 字段的值是 u，表示此行因为更新而更改。
- before 字段显示行的前一个状态，以及更新 数据库提交前存在的值。
- after 字段显示行的更新状态，EMAIL 值现在设置为 anne@example.com。
- source 字段的结构包含与之前相同的字段，但这些值不同，因为连接器从 redo 日志中的不同位置捕获事件。
- ts_ms 字段显示 Debezium 处理事件的时间戳。

payload 部分显示一些其他有用的信息。例如，通过比较 before 和 after 结构，我们可以确定提交后的行如何更改。源 结构提供有关此变化的 Oracle 记录的信息，从而提供可追溯性。它还有助于我们深入

了解此事件与本主题中的其他事件以及其他主题相关的情况。它是否在之前、之后或作为与另一个事件相同的提交的一部分？



注意

当更新行的主/唯一键的列时，行的键值会改变。因此，Debezium 会在此类 更新后发出三个 事件：

- *DELETE 事件。*
- *一个 tombstone 事件，带有行的旧键。*
- *为行提供新密钥的 INSERT 事件。*

删除 事件

以下示例显示了上一次 create 和 update 事件示例中显示的表的 delete 事件。delete 事件的 schema 部分与这些事件的 schema 部分相同。

```
{
    "schema": { ... },
    "payload": {
        "before": {
            "ID": 1004,
            "FIRST_NAME": "Anne",
            "LAST_NAME": "Kretchmar",
            "EMAIL": "anne@example.com"
        },
        "after": null,
        "source": {
            "version": "2.3.4.Final",
            "name": "server1",
            "ts_ms": 1520085153000,
            "txId": "6.28.807",
            "scn": "2122184",
            "commit_scn": "2122184",
            "rs_id": "001234.00012345.0124",
            "ssn": 1,
            "redo_thread": 1,
            "user_name": "user",
            "snapshot": false
        },
        "op": "d",
```

```
        "ts_ms": 1532592105960
    }
}
```

与 create 或 update 事件相比，事件的 payload 部分显示了几个不同之处：

- op 字段的值是 d，表示行已被删除。
- before 字段显示与数据库提交中删除的行的前导状态。
- after 字段的值为 null，表示行不再存在。
- source 字段的结构中包括了多个在 create 或 update 事件中存在的键， 但 ts_ms, scn, 和 txId 中的值不同。
- ts_ms 显示指示 Debezium 处理此事件的时间戳。

删除 事件为消费者提供处理此行删除所需的信息。

Oracle 连接器的事件被设计为使用 Kafka 日志压缩，只要保留每个键的最新消息，就可以删除一些旧的信息。这允许 Kafka 回收存储空间，同时确保主题包含完整的数据集，并可用于重新载入基于密钥的状态。

删除行时，上例中显示的 delete 事件值仍可用于日志压缩，因为 Kafka 能够删除使用同一键的所有之前消息。message 值必须设置为 null，以指示 Kafka 删除共享同一键的所有消息。为了实现此目的，Debezium 的 Oracle 连接器总是遵循一个 delete 事件，它有一个特殊的 tombstone 事件，它具有相同的键但 null 值。您可以通过设置连接器属性 tombstones.on.delete 来改变默认的行为。

Truncate 事件

截断 更改事件信号，表示表已被截断。在这种情况下，message 键为 null，message 值类似如下：

```
{
    "schema": { ... },
    "payload": {
        "before": null,
        "after": null,
```

```
        "source": { 1
            "version": "2.3.4.Final",
            "connector": "oracle",
            "name": "oracle_server",
            "ts_ms": 1638974535000,
            "snapshot": "false",
            "db": "ORCLPDB1",
            "sequence": null,
            "schema": "DEBEZIUM",
            "table": "TEST_TABLE",
            "txId": "02000a0037030000",
            "scn": "13234397",
            "commit_scn": "13271102",
            "lcr_position": null,
            "rs_id": "001234.00012345.0124",
            "ssn": 1,
            "redo_thread": 1,
            "user_name": "user"
        },
        "op": "t", 2
        "ts_ms": 1638974558961, 3
        "transaction": null
    }
}
```

表 7.8. truncate 事件值字段的描述

项	字段名称	描述
1	**source**	描述事件源元数据的必需字段。在 *truncate* 事件值中，**source** 字段结构与为同一表的 *create*, *update*, 和 *delete* 事件相同，提供此元数据： • Debezium 版本 • 连接器类型和名称 • 包含新行的数据库和表 • 模式名称 • 如果事件是快照的一部分（对于 *截断* 事件，始终为 **false** ） • 执行操作的事务的 ID • 操作的 SCN • 在数据库中进行更改时的时间戳 • 执行更改的用户名
2	**op**	描述操作类型的强制字符串。**op** 字段值为 **t**，表示此表已被截断。

项	字段名称	描述
3	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

因为 truncate 事件代表对整个表所做的更改，且没有消息密钥，所以在带有多个分区的主题中，不能保证消费者接收 truncate 事件和更改事件(创建、更新 等等)以订购表。例如，当消费者从不同的分区读取事件时，它可能会在为同一表接收 truncate 事件后收到表 的更新 事件。只有在主题使用单个分区时，才能保证排序。

如果您不想捕获 truncate 事件，请使用 skipped.operations 选项过滤它们。

7.3. DEBEZIUM ORACLE 连接器如何映射数据类型

当 Debezium Oracle 连接器检测到表行值中的更改时，它会发出一个代表更改的事件。每个更改事件记录的结构化方式与原始表相同，事件记录包含每个列值的字段。表列的数据类型决定了连接器如何代表更改事件字段的值，如以下部分的表中所示。

对于表中的每个列，Debezium 将源数据类型映射到 字面类型，在某些情况下，在相应的事件字段中都有一个 语义类型。

字面类型

描述值如何表示，使用以下 Kafka Connect 模式类型之一： INT8,INT16,INT32,INT64, INT64 ,FLOAT32,FLOAT64,BOOLEAN,STRING,BYTES,ARRAY,MAP, 和 STRUCT.

语义类型

描述 Kafka Connect 模式如何使用字段的名称捕获字段 的含义。

如果默认数据类型转换不满足您的需要，您可以为连接器 创建自定义转换器。

对于某些 Oracle 大对象(CLOB、NCLOB 和 BLOB)和数字数据类型，您可以通过更改默认配置属性设置来操作连接器执行类型映射的方式。有关如何对这些数据类型的 Debezium 属性控制映射的更多信息，请参阅 Binary 和 Character LOB type 和 Numeric 类型。

有关 Debezium 连接器如何映射 Oracle 数据类型的更多信息，请参阅以下主题：

- *字符类型*
- *二进制和 Character LOB 类型*
- *数字类型*
- *布尔值类型*
- *时序类型*
- *ROWID 类型*
- *用户定义的类型*
- *Oracle 提供的类型*
- *默认值*

字符类型

下表描述了连接器如何映射基本字符类型。

表 7.9. Oracle 基本字符类型的映射

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
CHAR[(M)]	**字符串**	不适用
NCHAR[(M)]	**字符串**	不适用

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
NVARCHAR2[(M)]	**字符串**	不适用
VARCHAR[(M)]	**字符串**	不适用
VARCHAR2[(M)]	**字符串**	不适用

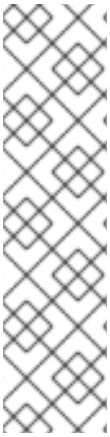
二进制和 CHARACTER LOB 类型

使用带有 Debezium Oracle 连接器的 BLOB、CLOB 和 NCLOB 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview。

下表描述了连接器如何映射二进制和字符大对象(LOB)数据类型。

表 7.10. Oracle 二进制和字符 LOB 类型的映射

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
BFILE	不适用	*不支持这个数据类型*
BLOB	**BYTES**	raw 字节（默认值）、base64 编码的 String 或 base64-url-safe-encoded String 或基于 **binary.handling.mode** 连接器配置属性设置的十六进制编码的字符串。
CLOB	**字符串**	不适用
LONG	不适用	*不支持此数据类型。*
长**原始**	不适用	*不支持此数据类型。*
NCLOB	**字符串**	不适用
RAW	不适用	*不支持此数据类型。*



注意

Oracle 仅提供 CLOB、NCLOB 和 BLOB 数据类型的列值（如果它们在 SQL 语句中明确设置或更改）。因此，更改事件永远不会包含没有变化的 CLOB、NCLOB 或 BLOB 列的值。相反，它们包含由连接器属性 unavailable.value.placeholder 定义的占位符。

如果更新了 CLOB、NCLOB 或 BLOB 列的值，则新值将放置在相应更新更改事件的 after 项中。before 元素包含不可用值占位符。

数字类型

下表描述了 Debezium Oracle 连接器如何映射数字类型。

注意

您可以改变连接器映射 Oracle DECIMAL, NUMBER, NUMERIC, 和 REAL 数据类型的方式，方法是修改连接器的 decimal.handling.mode 配置属性的值。当属性设置为 精确的 默认值时，连接器会将这些 Oracle 数据类型映射到 Kafka Connect org.apache.kafka.connect.data.Decimal 逻辑类型，如表中所示。当属性的值设为 double 或 string 时，连接器对某些 Oracle 数据类型使用备用映射。如需更多信息，请参阅 下表中的 Semantic 类型和 notes 列。

表 7.11. Oracle 数字数据类型的映射

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
BINARY_FLOAT	**FLOAT32**	不适用
BINARY_DOUBLE	**FLOAT64**	不适用
DECIMAL[(P, S)]	**BYTES** / **INT8** / **INT16** / **INT32** / **INT64**	**org.apache.kafka.connect.data.Decimal** if if **BYTES** Handled to **NUMBER** （请注意，对于 **DECIMAL**，S 默认为 0。 当将 **decimal.handling.mode** 属性设置为 **double** 时，连接器将 **DECIMAL** 值表示为 Java **双** 值，类型为 **FLOAT64**。 当将 **decimal.handling.mode** 属性设置为 **string** 时，连接器代表 DECIMAL 值，其格式的字符串表示为 **STRING**。
双精度	**STRUCT**	**io.debezium.data.VariableScaleDecimal** Contains 一个结构，它有两个字段：类型为 **INT32** 的扩展，其中包含未 **扩展** 格式的传输 **值** 和值 **BYTES**。

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
FLOAT[(P)]	**STRUCT**	**io.debezium.data.VariableScaleDecimal** Contains 一个结构，它有两个字段：类型为 **INT32** 的扩展，其中包含未 **扩展** 格式的传输 **值** 和值 **BYTES**。
整数， INT	**BYTES**	**org.apache.kafka.connect.data.Decimal** **INTEGER** 在 Oracle 中映射到 NUMBER (38,0)，因此可以保存大于 **INT** 类型的值可以存储
NUMBER[(P[, *])]	**STRUCT**	**io.debezium.data.VariableScaleDecimal** Contains 一个结构，它有两个字段：类型为 **INT32** 的扩展，其中包含未 **扩展** 格式的传输 **值** 和值 **BYTES**。 当将 **decimal.handling.mode** 属性设置为 **double** 时，连接器将 **NUMBER** 值表示为 Java **双** 值，类型为 **FLOAT64**。 当将 **decimal.handling.mode** 属性设置为 **string** 时，连接器显示 **NUMBER** 值作为一个有特定格式的字符串（如 schema 类型 **STRING**）。
NUMBER(P, S <= 0)	**INT8** / **INT16** / **INT32** / **INT64**	**NUMBER** 列，扩展为 0 代表整数。负比例表示 Oracle 中的舍入，例如，规模为 -2 会导致舍入成百。 根据精度和规模，选择以下匹配的 Kafka Connect 整数类型之一： • P - S < 3, **INT8** • P - S < 5, **INT16** • P - S < 10, **INT32** • P - S < 19, **INT64** • P - S >= 19, **BYTES** (**org.apache.kafka.connect.data.Decimal**) 当将 **decimal.handling.mode** 属性设置为 **double** 时，连接器将 **NUMBER** 值表示为 Java **双** 值，类型为 **FLOAT64**。 当将 **decimal.handling.mode** 属性设置为 **string** 时，连接器显示 **NUMBER** 值作为一个有特定格式的字符串（如 schema 类型 **STRING**）。
NUMBER(P, S > 0)	**BYTES**	**org.apache.kafka.connect.data.Decimal**

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
数字[(P, S)]	**BYTES** / **INT8** / **INT16** / **INT32** / **INT64**	**org.apache.kafka.connect.data.Decimal** if if **BYTES** Handled 与 **NUMBER** （请注意，S 默认为 0 代表 **NUMERIC**）。 当将 **decimal.handling.mode** 属性设置为 **double** 时，连接器将 **NUMERIC** 值表示为 Java **双** 值，类型为 **FLOAT64**。 当将 **decimal.handling.mode** 属性设置为 **string** 时，连接器代表 **NUMERIC** 值，其格式的字符串表示为 **STRING**。
SMALLINT	**BYTES**	**org.apache.kafka.connect.data.Decimal** **SMALLINT** 在 Oracle 中映射到 NUMBER (38,0)，因此可以保存大于 **INT** 类型的值可以存储
REAL	**STRUCT**	**io.debezium.data.VariableScaleDecimal** Contains 一个结构，它有两个字段：类型为 **INT32** 的扩展，其中包含未 **扩展** 格式的传输 **值** 和值 **BYTES**。 当将 **decimal.handling.mode** 属性设置为 **double** 时，连接器将 **REAL** 值表示为 Java **双** 值，类型为 **FLOAT64**。 当将 **decimal.handling.mode** 属性设置为 **string** 时，连接器使用 schema type **STRING** 代表 **REAL** 值作为格式的字符串表示。

如上面提到的，Oracle 允许在 NUMBER 类型中进行负扩展。当数字以 Decimal 表示时，这可能会导致转换为 Avro 格式时出现问题。十进制 类型包括扩展信息，但 Avro 规格 只允许缩放的正值。根据所使用的模式 registry，可能会导致 Avro 序列化失败。要避免这个问题，您可以使用 NumberToZeroScaleConverter，它将带有负精度（小数点左面）的高的数字 (P - S >= 19) 转换为小数点右面零位的 Decimal 类型。它可以配置如下：

```
converters=zero_scale
zero_scale.type=io.debezium.connector.oracle.converters.NumberToZeroScaleConverter
zero_scale.decimal.mode=precise
```

默认情况下，数字转换为 Decimal 类型(zero_scale.decimal.mode=precise)，但支持完整的两种支持类型(双 和 字符串)。

布尔值类型

Oracle 不支持 BOOLEAN 数据类型。但是，通常使用带有特定语义的其他数据类型来模拟逻辑 BOOLEAN 数据类型的概念。

为了允许您将源列转换为布尔值数据类型，Debezium 提供了一个 NumberOneTo BooleanConverter 自定义转换器，您可以使用以下方法之一使用：

- 将所有 NUMBER (1) 列映射到 BOOLEAN 类型。
- 使用以逗号分隔的正则表达式列表枚举列的子集。

要使用这种类型的转换，您必须使用 selector 参数设置 converters 配置属性，如下例所示：

```
converters=boolean
boolean.type=io.debezium.connector.oracle.converters.NumberOneToBooleanConverter
boolean.selector=.*MYTABLE.FLAG,.*.IS_ARCHIVED
```

时序类型

除了 Oracle INTERVAL,TIMESTAMP 之外，TIME ZONE 和 TIMESTAMP 利用 LOCAL TIME ZONE 数据类型，连接器转换 temporal 类型取决于 time.precision.mode 配置属性的值。

当将 time.precision.mode 配置属性设置为 adaptive （默认值）时，连接器会根据列的数据类型确定 temporal 类型的字面和语义类型，以便事件 准确 代表数据库中的值：

Oracle 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT64**	**io.debezium.time.Timestamp** 代表自 UNIX epoch 起的毫秒数，不包括时区信息。
INTERVAL DAY[(M)] 到 SECOND	**FLOAT64**	**io.debezium.time.MicroDuration** 使用每月平均值的 **365.25 / 12.0** 公式的微秒数。 **io.debezium.time.Interval**（当 **interval.handling.mode** 设置为 **字符串**） 字符串表示 遵循格式 **P<years>Y<months>M<days>DT<hours>H<minutes>M<seconds>S**, 例如，**P1Y2M3DT4H5M6.78S**。

Oracle 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
INTERVAL YEAR[(M)] 到月份	**FLOAT64**	**io.debezium.time.MicroDuration** 使用每月平均值的 **365.25 / 12.0** 公式的微秒数。 **io.debezium.time.Interval**（当 **interval.handling.mode** 设置为 **字符串**） 字符串表示 遵循格式 **P<years>Y<months>M<days>DT<hours>H<minutes>M<seconds>S**, 例如，**P1Y2M3DT4H5M6.78S**。
TIMESTAMP(0 - 3)	**INT64**	**io.debezium.time.Timestamp** 代表自 UNIX epoch 起的毫秒数，不包括时区信息。
时间戳、时间戳(4 - 6)	**INT64**	**io.debezium.time.MicroTimestamp** 代表自 UNIX epoch 起的微秒数，不包括时区信息。
TIMESTAMP (7 - 9)	**INT64**	**io.debezium.time.NanoTimestamp** 代表 UNIX epoch 后的纳秒数量，不包括时区信息。
带有时区的时间戳	**字符串**	**io.debezium.time.ZonedTimestamp** 是带有时区信息的时间戳的字符串。
带有本地时区的时间戳	**字符串**	**io.debezium.time.ZonedTimestamp** 是 UTC 中时间戳的字符串。

当将 time.precision.mode 配置属性设为 connect 时，连接器使用预定义的 Kafka Connect 逻辑类型。当消费者只了解内置的 Kafka Connect 逻辑类型，且无法处理变量-precision 时间值时，这很有用。由于 Oracle 支持的精度级别超过 Kafka Connect 支持中的逻辑类型，如果将 time.precision.mode 设置为 connect，当数据库列的 fractional second precision 值大于 2 时，会出现丢失精度的结果：

Oracle 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**org.apache.kafka.connect.data.Date** 代表 UNIX epoch 后的天数。

Oracle 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
INTERVAL DAY[(M)] 到 SECOND	**FLOAT64**	**io.debezium.time.MicroDuration** 使用每月平均值的 **365.25 / 12.0** 公式的微秒数。 **io.debezium.time.Interval**（当 **interval.handling.mode** 设置为 **字符串**） 字符串表示 遵循格式 **P<years>Y<months>M<days>DT<hours>H<minutes>M<seconds>S**, 例如，**P1Y2M3DT4H5M6.78S**。
INTERVAL YEAR[(M)] 到月份	**FLOAT64**	**io.debezium.time.MicroDuration** 使用每月平均值的 **365.25 / 12.0** 公式的微秒数。 **io.debezium.time.Interval**（当 **interval.handling.mode** 设置为 **字符串**） 字符串表示 遵循格式 **P<years>Y<months>M<days>DT<hours>H<minutes>M<seconds>S**, 例如，**P1Y2M3DT4H5M6.78S**。
TIMESTAMP(0 - 3)	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 UNIX epoch 起的毫秒数，不包括时区信息。
TIMESTAMP (4 - 6)	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 UNIX epoch 起的毫秒数，不包括时区信息。
TIMESTAMP (7 - 9)	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 UNIX epoch 起的毫秒数，不包括时区信息。
带有时区的时间戳	**字符串**	**io.debezium.time.ZonedTimestamp** 是带有时区信息的时间戳的字符串。
带有本地时区的时间戳	**字符串**	**io.debezium.time.ZonedTimestamp** 是 UTC 中时间戳的字符串。

ROWID 类型

下表描述了连接器如何映射 ROWID （托管地址）数据类型。

表 7.12. Oracle ROWID 数据类型的映射

Oracle 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
ROWID	**字符串**	不适用
UROWID	不适用	*不支持此数据类型。*

用户定义的类型

Oracle 可让您定义自定义数据类型，以便在内置数据类型无法满足您的要求时提供灵活性。有几种用户定义的类型，如对象类型、REF 数据类型、Varrays 和 Nested Tables。目前，您不能将 Debezium Oracle 连接器与任何这些用户定义的类型一起使用。

Oracle 提供的类型

Oracle 提供基于 SQL 的接口，可用于在内置或 ANSI 支持的类型不足时定义新类型。Oracle 提供多种常用的数据类型来满足各种目的，如 Any、XML 或 Spatial 类型。目前，您不能将 Debezium Oracle 连接器与任何这些数据类型一起使用。

默认值

如果为数据库模式中的列指定了默认值，Oracle 连接器将尝试将此值传播到对应 Kafka 记录字段的 schema。最常见的数据类型受到支持，包括：

- *字符类型(CHAR、NCHAR、VARCHAR、VARCHAR 2、NVARCHAR、NVARCHAR2)*
- *数字类型(INTEGER 、数字化等)*
- *时序类型(DATE、TIMESTAMP、INTERVAL 等)*

如果临时类型使用 TO_TIMESTAMP 或 TO_DATE 等函数调用来代表默认值，则连接器将通过进行额外的数据库调用来评估函数来解析默认值。例如，如果 DATE 列定义了默认值 TO_DATE ('2021-01-02', 'YYYY-MM-DD')，则列的默认值将是该日期的 UNIX epoch 或 18629 的天数。

如果临时类型使用 SYSDATE 常数来代表默认值，则连接器将根据列是否定义为 NOT NULL 或 NULL 来解决此问题。如果列可为空，则不会设置默认值；但是，如果列不可为空，则默认值将解析为 0（用于 DATE 或 TIMESTAMP(n) 数据类型）或 1970-01-01T00:00:00Z (用于 TIMESTAMP WITH TIME ZONE 或 TIMESTAMP WITH LOCAL TIME ZONE 数据类型)。默认值为数字，除非列是 TIMESTAMP WITH TIME ZONE 或 TIMESTAMP WITH LOCAL TIME ZONE，在这种情况下，它作为字符串发出。

7.4. 设置 ORACLE 以使用 DEBEZIUM

以下步骤设置用于 Debezium Oracle 连接器的 Oracle。这些步骤假定将多租户配置与容器数据库和至少一个可插拔数据库搭配使用。如果您不打算使用多租户配置，可能需要调整以下步骤。

有关设置用于 Debezium 连接器的 Oracle 的详情，请查看以下部分：

- 第 7.4.1 节 “Debezium Oracle 连接器与 Oracle 安装类型的兼容性”
- 第 7.4.2 节 “Debezium Oracle 连接器在捕获更改事件时排除的 schema”
- 第 7.4.4 节 “准备用于 Debezium 的 Oracle 数据库”
- 第 7.4.5 节 “重新定义 Oracle redo 日志大小以容纳数据字典”
- 第 7.4.6 节 “为 Debezium Oracle 连接器创建 Oracle 用户”
- 第 7.4.7 节 “支持 Oracle 待机数据库”

7.4.1. Debezium Oracle 连接器与 Oracle 安装类型的兼容性

Oracle 数据库可以作为独立实例安装，也可以使用 Oracle Real Application Cluster (RAC)安装。Debezium Oracle 连接器与两种类型的安装兼容。

7.4.2. Debezium Oracle 连接器在捕获更改事件时排除的 schema

当 Debezium Oracle 连接器捕获表时，它会自动从以下模式中排除表：

- appqossys
- audsys

- ***ctxsys***
- ***dvsys***
- ***dbsfwuser***
- ***dbsnmp***
- ***qsmadmin_internal***
- ***lbacsys***
- ***mdsys***
- ***ojvmsys***
- ***olapsys***
- ***orddata***
- ***ordsys***
- ***outln***
- ***sys***
- ***system***

- *wmsys*
- *xdb*

要启用连接器来捕获表中的更改，表必须使用前面列表中未命名的 schema。

7.4.3. Debezium Oracle 连接器在捕获更改事件时排除的表

当 Debezium Oracle 连接器捕获表时，它会自动排除与以下规则匹配的表：

- *压缩顾问表与模式 CMP[3|4 reporting[0-9]+ 匹配。*
- *与 SYS_IOT_OVER_% 模式匹配的索引组织表。*
- *与 MDRT_%、MDRS_% 或 MDXT_% 模式匹配的空间表。*
- *嵌套表*

要让连接器捕获名称与上述任何规则匹配的表，您必须重命名表。

7.4.4. 准备用于 Debezium 的 Oracle 数据库

Oracle LogMiner 所需的配置

```
ORACLE_SID=ORACLCDB dbz_oracle sqlplus /nolog

CONNECT sys/top_secret AS SYSDBA
alter system set db_recovery_file_dest_size = 10G;
alter system set db_recovery_file_dest = '/opt/oracle/oradata/recovery_area' scope=spfile;
shutdown immediate
startup mount
alter database archivelog;
alter database open;
-- Should now "Database log mode: Archive Mode"
```

```
archive log list

exit;
```

Oracle AWS RDS 不允许执行上述命令，也不允许您以 sysdba 身份登录。AWS 提供了这些替代命令来配置 LogMiner。在执行这些命令前，请确保您的 Oracle AWS RDS 实例已启用了备份。

要确认 Oracle 启用了备份，请首先执行以下命令。LOG_MODE 应该称 ARCHIVELOG。如果没有，您可能需要重启 Oracle AWS RDS 实例。

Oracle AWS RDS LogMiner 所需的配置

```
SQL> SELECT LOG_MODE FROM V$DATABASE;

LOG_MODE
------------
ARCHIVELOG
```

当 LOG_MODE 设为 ARCHIVELOG 后，执行命令来完成 LogMiner 配置。第一个命令将数据库设置为 archivelogs，第二个添加了补充日志记录。

Oracle AWS RDS LogMiner 所需的配置

```
exec rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',24);

exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD');
```

要让 Debezium 捕获更改数据库行之前的状态，还必须为捕获的表或整个数据库启用附件日志记录。以下示例演示了如何在单个 inventory.customers 表中为所有列配置补充日志记录。

```
ALTER TABLE inventory.customers ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

为所有表列启用附加日志记录会增加 Oracle redo 日志的卷。为防止日志大小过量增长，请选择性地应用前面的配置。

最少的附件日志记录必须在数据库级别启用，并可配置如下：

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

7.4.5. 重新定义 Oracle redo 日志大小以容纳数据字典

根据数据库配置，大小和红色日志的数量可能不足以达到可接受的性能。在设置 Debezium Oracle 连接器前，请确保 redo 日志的容量足以支持数据库。

数据库红色日志的容量必须足以存储其数据字典。通常，数据字典的大小会随着数据库中的表和列的数量增加。如果 redo 日志缺少足够容量，则数据库和 Debezium 连接器可能会遇到性能问题。

请参考您的数据库管理员来评估数据库是否可能需要增加日志容量。

7.4.6. 为 Debezium Oracle 连接器创建 Oracle 用户

要使 Debezium Oracle 连接器捕获更改事件，它必须以具有特定权限的 Oracle LogMiner 用户身份运行。以下示例显示了在多租户数据库模型中为连接器创建 Oracle 用户帐户的 SQL。

警告

连接器捕获由其自身 Oracle 用户帐户进行的数据库更改。但是，它不会捕获 SYS 或 SYSTEM 用户帐户所做的更改。

创建连接器的 LogMiner 用户

```
sqlplus sys/top_secret@//localhost:1521/ORCLCDB as sysdba
```

```
  CREATE TABLESPACE logminer_tbs DATAFILE
'/opt/oracle/oradata/ORCLCDB/logminer_tbs.dbf'
    SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
  exit;

sqlplus sys/top_secret@//localhost:1521/ORCLPDB1 as sysdba
  CREATE TABLESPACE logminer_tbs DATAFILE
'/opt/oracle/oradata/ORCLCDB/ORCLPDB1/logminer_tbs.dbf'
    SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
  exit;

sqlplus sys/top_secret@//localhost:1521/ORCLCDB as sysdba

  CREATE USER c##dbzuser IDENTIFIED BY dbz
    DEFAULT TABLESPACE logminer_tbs
    QUOTA UNLIMITED ON logminer_tbs
    CONTAINER=ALL;

  GRANT CREATE SESSION TO c##dbzuser CONTAINER=ALL; 1
  GRANT SET CONTAINER TO c##dbzuser CONTAINER=ALL; 2
  GRANT SELECT ON V_$DATABASE to c##dbzuser CONTAINER=ALL; 3
  GRANT FLASHBACK ANY TABLE TO c##dbzuser CONTAINER=ALL; 4
  GRANT SELECT ANY TABLE TO c##dbzuser CONTAINER=ALL; 5
  GRANT SELECT_CATALOG_ROLE TO c##dbzuser CONTAINER=ALL; 6
  GRANT EXECUTE_CATALOG_ROLE TO c##dbzuser CONTAINER=ALL; 7
  GRANT SELECT ANY TRANSACTION TO c##dbzuser CONTAINER=ALL; 8
  GRANT LOGMINING TO c##dbzuser CONTAINER=ALL; 9

  GRANT CREATE TABLE TO c##dbzuser CONTAINER=ALL; 10
  GRANT LOCK ANY TABLE TO c##dbzuser CONTAINER=ALL; 11
  GRANT CREATE SEQUENCE TO c##dbzuser CONTAINER=ALL; 12

  GRANT EXECUTE ON DBMS_LOGMNR TO c##dbzuser CONTAINER=ALL; 13
  GRANT EXECUTE ON DBMS_LOGMNR_D TO c##dbzuser CONTAINER=ALL; 14

  GRANT SELECT ON V_$LOG TO c##dbzuser CONTAINER=ALL; 15
  GRANT SELECT ON V_$LOG_HISTORY TO c##dbzuser CONTAINER=ALL; 16
  GRANT SELECT ON V_$LOGMNR_LOGS TO c##dbzuser CONTAINER=ALL; 17
  GRANT SELECT ON V_$LOGMNR_CONTENTS TO c##dbzuser CONTAINER=ALL; 18
  GRANT SELECT ON V_$LOGMNR_PARAMETERS TO c##dbzuser CONTAINER=ALL; 19
  GRANT SELECT ON V_$LOGFILE TO c##dbzuser CONTAINER=ALL; 20
  GRANT SELECT ON V_$ARCHIVED_LOG TO c##dbzuser CONTAINER=ALL; 21
  GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO c##dbzuser CONTAINER=ALL; 22
  GRANT SELECT ON V_$TRANSACTION TO c##dbzuser CONTAINER=ALL; 23

  GRANT SELECT ON V_$MYSTAT TO c##dbzuser CONTAINER=ALL; 24
  GRANT SELECT ON V_$STATNAME TO c##dbzuser CONTAINER=ALL; 25

  exit;
```

表 7.13. 权限/授予的描述

项	角色名称	描述
1	创建会话	启用连接器连接到 Oracle。
2	设置容器	启用连接器在可插拔数据库间切换。只有在启用了容器数据库支持(CDB)时，才需要这样做。
3	SELECT ON V_$DATABASE	启用连接器读取 **V$DATABASE** 表。
4	FLASHBACK ANY 表	启用连接器来执行 Flashback 查询，这是连接器如何执行数据的初始快照。
5	选择任何表	启用连接器来读取任何表。
6	SELECT_CATALOG_ROLE	启用连接器读取 Oracle LogMiner 会话所需的数据字典。
7	EXECUTE_CATALOG_ROLE	启用连接器将数据字典写入 Oracle redo 日志，这是跟踪架构更改所需要的。
8	选择任何事务	启用快照过程对任何事务执行 Flashback 快照查询。当授予 **FLASHBACK TABLE** 时，也应授予这一操作。
9	LOGMINING	在较新版本的 Oracle 中添加了此角色，作为授予 Oracle LogMiner 及其软件包的完整访问权限的方法。在没有此角色的 Oracle 的旧版本中，您可以忽略这个授权。
10	创建表	启用连接器在默认表空间中创建其 flush 表。flush 表允许连接器将 LGWR 内部缓冲区的清除显式控制到磁盘。
11	锁定任何表	启用连接器在 schema 快照期间锁定表。如果通过配置明确禁用了快照锁定，可以安全地忽略此授权。
12	创建序列	启用连接器在默认表空间中创建序列。
13	EXECUTE ON DBMS_LOGMNR	启用连接器在 **DBMS_LOGMNR** 软件包中运行方法。这需要与 Oracle LogMiner 交互。在较新的 Oracle 版本中，这通过 **LOGMINING** 角色授予，但在旧版本中，必须明确授予这。

项	角色名称	描述
14	EXECUTE ON DBMS_LOGMNR_D	启用连接器在 **DBMS_LOGMNR_D** 软件包中运行方法。这需要与 Oracle LogMiner 交互。在较新的 Oracle 版本中，这通过 **LOGMINING** 角色授予，但在旧版本中，必须明确授予这。
15 到 25	SELECT ON V_$....	启用连接器来读取这些表。连接器必须能够读取 Oracle redo 和归档日志的信息，以及当前事务状态，以准备 Oracle LogMiner 会话。如果没有这些授权，连接器将无法操作。

7.4.7. 支持 Oracle 待机数据库

重要

Debezium Oracle 连接器从只读逻辑待机数据库进行最大更改的功能是开发者预览功能。红帽以任何方式支持开发人员预览功能，且功能不完整或生产就绪。对于生产环境或关键业务工作负载，不要使用开发人员预览软件。开发人员预览软件可提前访问即将发布的产品软件。客户可以使用此软件测试功能并在开发过程中提供反馈。此软件可能没有任何文档，可以随时更改或删除，并收到有限的测试。红帽可能会提供在没有关联的 SLA 的情况下提交开发人员预览软件反馈的方法。

有关 Red Hat Developer Preview 软件支持范围的更多信息，请参阅 开发人员预览支持范围。

7.5. 部署 DEBEZIUM ORACLE 连接器

您可以使用以下任一方法部署 Debezium Oracle 连接器：

- ***使用 AMQ Streams 自动创建包含连接器插件的镜像。***

  这是首选的方法。

- ***从 Dockerfile 构建自定义 Kafka Connect 容器镜像。***

重要

由于许可证要求，Debezium Oracle 连接器存档不包括连接器连接到 Oracle 数据库所需的 Oracle JDBC 驱动程序。要启用连接器访问数据库，您必须将驱动程序添加到连接器环境中。如需更多信息，请参阅 获取 Oracle JDBC 驱动程序。

其他资源

- 第 7.6 节 “Debezium Oracle 连接器配置属性的描述”

7.5.1. 获取 Oracle JDBC 驱动程序

由于许可证要求，Debezium 需要连接到 Oracle 数据库的 Oracle JDBC 驱动程序文件不包含在 Debezium Oracle 连接器存档中。驱动程序可从 Maven Central 下载。根据您使用的部署方法，您可以通过向 Kafka Connect 自定义资源添加命令或用于构建连接器镜像的 Dockerfile 来检索驱动程序。

- 如果您使用 AMQ Streams 将连接器添加到 Kafka Connect 镜像，请将驱动程序的 Maven Central 位置添加到 KafkaConnect 自定义资源中的 builds.plugins.artifact.url 中，如 第 7.5.3 节 “使用 AMQ Streams 部署 Debezium Oracle 连接器” 所示。

- 如果您使用 Dockerfile 为连接器构建容器镜像，请在 Dockerfile 中插入 curl 命令，以指定从 Maven Central 下载所需驱动程序文件的 URL。如需更多信息，请参阅 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium Oracle 连接器。

7.5.2. 使用 AMQ Streams 部署 Debezium Oracle 连接器

从 Debezium 1.7 开始，部署 Debezium 连接器的首选方法是使用 AMQ Streams 构建包含连接器插件的 Kafka Connect 容器镜像。

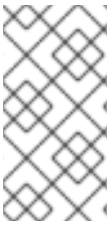
在部署过程中，您可以创建并使用以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR，并包含有关镜像中需要包含连接器工件的信息。

- KafkaConnector CR，提供包括连接器用来访问源数据库的信息。在 AMQ Streams 启动 Kafka Connect pod 后，您可以通过应用 KafkaConnector CR 来启动连接器。

在 Kafka Connect 镜像的构建规格中，您可以指定可用于部署的连接器。对于每个连接器插件，您还可以指定您的部署可以使用的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。当 AMQ Streams 构建 Kafka Connect 镜像时，它会下载指定的工件，并将其合并到镜像中。

KafkaConnect CR 中的 spec.build.output 参数指定存储生成的 Kafka Connect 容器镜像的位置。容器镜像可以存储在 Docker registry 中，也可以存储在 OpenShift ImageStream 中。要将镜像存储在 ImageStream 中，您必须在部署 Kafka Connect 前创建 ImageStream。镜像流不会被自动创建。



注意

如果使用 KafkaConnect 资源来创建集群，之后无法使用 Kafka Connect REST API 创建或更新连接器。您仍然可以使用 REST API 来检索信息。

其他资源

- 在 OpenShift 中使用 AMQ Streams 配置 Kafka 连接。
- 在 OpenShift 中部署和管理 AMQ Streams 中，使用 AMQ Streams 自动创建新容器镜像。

7.5.3. 使用 AMQ Streams 部署 Debezium Oracle 连接器

使用早期版本的 AMQ Streams 时，要在 OpenShift 上部署 Debezium 连接器，您需要首先为连接器构建 Kafka Connect 镜像。在 OpenShift 上部署连接器的当前首选方法是使用 AMQ Streams 中的构建配置来构建 Kafka Connect 容器镜像，其中包含您要使用的 Debezium 连接器插件。

在构建过程中，AMQ Streams Operator 将 KafkaConnect 自定义资源（包括 Debezium 连接器定义）中的输入参数转换为 Kafka Connect 容器镜像。构建会从 Red Hat Maven 存储库或其他配置的 HTTP 服务器下载必要的工件。

新创建的容器被推送到在 .spec.build.output 中指定的容器 registry，用于部署 Kafka Connect 集群。在 AMQ Streams 构建 Kafka Connect 镜像后，您可以创建 KafkaConnector 自定义资源来启动构建中包含的连接器。

先决条件

- 您可以访问安装了集群 Operator 的 OpenShift 集群。

- AMQ Streams Operator 正在运行。
- 在 OpenShift 中部署和升级 AMQ Streams 所述，会部署 Apache Kafka 集群。
- Kafka Connect 在 AMQ Streams 上部署
- 您有一个 Red Hat Integration 许可证。
- 已安装 OpenShift oc CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。
- 根据您要存储 Kafka Connect 构建镜像的方式，您需要 registry 权限，或者您必须创建 ImageStream 资源：

  将构建镜像存储在镜像 registry 中，如 Red Hat Quay.io 或 Docker Hub

  - 在 registry 中创建和管理镜像的帐户和权限。

  将构建镜像存储为原生 OpenShift ImageStream

  - ImageStream 资源已部署到集群中，以存储新的容器镜像。您必须为集群显式创建 ImageStream。默认无法使用镜像流。如需有关 ImageStreams 的更多信息，请参阅在 OpenShift Container Platform 中管理镜像流。

流程

1. 登录 OpenShift 集群。
2. 为连接器创建 Debezium KafkaConnect 自定义资源(CR)，或修改现有的资源。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 metadata.annotations 和 spec.build 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

> 例 7.1. 定义包含 Debezium 连接器的 KafkaConnect 自定义资源的 dbz-connect.yaml 文件

在以下示例中，自定义资源被配置为下载以下工件：

- Debezium Oracle 连接器存档。
- Service Registry 归档。Service Registry 是一个可选组件。只有在打算将 Avro 序列化与连接器搭配使用时，才添加 Service Registry 组件。
- Debezium 脚本 SMT 归档以及与 Debezium 连接器一起使用的相关语言依赖项。SMT 归档和语言依赖项是可选组件。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时，才添加这些组件。
- Oracle JDBC 驱动程序，需要连接到 Oracle 数据库，但不包含在连接器存档中。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: debezium-kafka-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  version: 3.5.0
  build: 2
    output: 3
      type: imagestream 4
      image: debezium-streams-connect:latest
    plugins: 5
      - name: debezium-connector-oracle
        artifacts:
          - type: zip 6
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-oracle/2.3.4.Final-redhat-00001/debezium-connector-oracle-2.3.4.Final-redhat-00001-plugin.zip 7
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-distro-connect-converter/2.4.4.Final-redhat-<build-number>/apicurio-registry-distro-connect-converter-2.4.4.Final-redhat-<build-number>.zip 8
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-scripting/2.3.4.Final-redhat-00001/debezium-scripting-2.3.4.Final-redhat-00001.zip
9
          - type: jar
            url:
https://repo1.maven.org/maven2/org/codehaus/groovy/groovy/3.0.11/groovy-3.0.11.jar 10
          - type: jar
```

```
        url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-jsr223/3.0.11/groovy-jsr223-3.0.11.jar
      - type: jar
        url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-json3.0.11/groovy-json-3.0.11.jar
      - type: jar                                                                                          (11)
        url: https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc8/21.6.0.0/ojdbc8-21.6.0.0.jar

  bootstrapServers: debezium-kafka-cluster-kafka-bootstrap:9093

  ...
```

表 7.14. Kafka Connect 配置设置的描述

项	描述
1	将 **strimzi.io/use-connector-resources** 注解设置为 **"true"**，使 Cluster Operator 使用 **KafkaConnector** 资源在此 Kafka Connect 集群中配置连接器。
2	**spec.build** 配置指定在镜像中存储构建镜像的位置，并列出要在镜像中包含的插件，以及插件工件的位置。
3	**build.output** 指定存储新构建镜像的 registry。
4	指定镜像输出的名称和镜像名称。**output.type** 的有效值是 **要推送到** 容器 registry（如 Docker Hub 或 Quay）或 **镜像流** 的有效值，以将镜像推送到内部 OpenShift ImageStream。要使用 ImageStream，必须将 ImageStream 资源部署到集群中。有关在 KafkaConnect 配置中指定 **build.output** 的更多信息，请参阅在 OpenShift 中配置 AMQ Streams 中的 AMQ Streams Build schema 参考。
5	**plugins** 配置列出了您要包含在 Kafka Connect 镜像中的所有连接器。对于列表中的每个条目，指定一个插件**名称**，以及有关构建连接器所需的工件的信息。另外，对于每个连接器插件，您还可以包含可用于连接器的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。
6	**artifacts.type** 的值指定在 **artifacts.url** 中指定的工件类型。有效类型为 **zip**、**tgz** 或 **jar**。Debezium 连接器存档以 **.zip** 文件格式提供。JDBC 驱动程序文件采用 **.jar** 格式。**类型** 值必须与 **url** 字段中引用的文件类型匹配。
7	**artifacts.url** 的值指定 HTTP 服务器的地址，如 Maven 存储库，用于存储连接器工件的文件。Debezium 连接器工件在 Red Hat Maven 存储库中提供。OpenShift 集群必须有权访问指定的服务器。
8	（可选）指定用于下载 Service Registry 组件的工件 **类型和 url**。包含 Service Registry 工件，只有在您希望连接器使用 Apache Avro 来序列化带有 Service Registry 的事件键和值时，而不是使用默认的 JSON 转换程序。
9	（可选）指定 Debezium 脚本 SMT 归档的工件 **类型和 url**，以用于 Debezium 连接器。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时才包括脚本 SMT。要使用脚本 SMT，您必须部署 JSR 223 兼容脚本实现，如 groovy。

项	描述
10	（可选）指定 JSR 223 兼容脚本实施的 JAR 文件的工件 **类型和 url**，这是 Debezium 脚本 SMT 所需的。  **重要** 如果使用 AMQ Streams 将连接器插件合并到 Kafka Connect 镜像中，每个所需的脚本语言 **工件。url** 必须指定 JAR 文件的位置，并且 **artifacts.type** 的值也必须设置为 **jar**。无效的值会导致连接器在运行时失败。 要启用带有脚本 SMT 的 Apache Groovy 语言，示例中的自定义资源会为以下库检索 JAR 文件： • **groovy** • **Groovy-jsr223**（指定代理） • **groovy-json**（解析 JSON 字符串的模块） Debezium 脚本 SMT 还支持使用 JSR 223 实现 GraalVM JavaScript。
11	在 Maven Central 中指定 Oracle JDBC 驱动程序的位置。Debezium Oracle 连接器存档中没有包括所需的驱动程序。

3. *输入以下命令将 KafkaConnect 构建规格应用到 OpenShift 集群：*

```
oc create -f dbz-connect.yaml
```

根据自定义资源中指定的配置，Streams Operator 准备要部署的 Kafka Connect 镜像。构建完成后，Operator 将镜像推送到指定的 registry 或 ImageStream，并启动 Kafka Connect 集群。集群中提供了您在配置中列出的连接器工件。

4. *创建一个 KafkaConnector 资源来定义您要部署的每个连接器的实例。*
例如，创建以下 KafkaConnector CR，并将它保存为 oracle-inventory-connector.yaml

例 7.2. 为 Debezium 连接器定义 KafkaConnector 自定义资源的 Oracle -inventory-connector.yaml 文件

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: debezium-kafka-connect-cluster
  name: inventory-connector-oracle (1)
spec:
```

```
    class: io.debezium.connector.oracle.OracleConnector 2
    tasksMax: 1 3
    config: 4
      schema.history.internal.kafka.bootstrap.servers: debezium-kafka-cluster-kafka-
    bootstrap.debezium.svc.cluster.local:9092
      schema.history.internal.kafka.topic: schema-changes.inventory
      database.hostname: oracle.debezium-oracle.svc.cluster.local 5
      database.port: 1521 6
      database.user: debezium 7
      database.password: dbz 8
      database.dbname: mydatabase 9
      topic.prefix: inventory-connector-oracle 10
      table.include.list: PUBLIC.INVENTORY 11

      ...
```

表 7.15. 连接器配置设置的描述

项	描述
1	使用 Kafka Connect 集群注册的连接器名称。
2	连接器类的名称。
3	可以同时操作的任务数量。
4	连接器的配置。
5	主机数据库实例的地址。
6	数据库实例的端口号。
7	Debezium 用于连接到数据库的帐户名称。
8	Debezium 用于连接到数据库用户帐户的密码。
9	要从中捕获更改的数据库名称。
10	数据库实例或集群的主题前缀。 指定的名称只能由字母数字字符或下划线组成。 因为主题前缀被用作从这个连接器接收更改事件的任何 Kafka 主题的前缀，所以该名称在集群中的连接器之间必须是唯一的。 如果连接器与 Avro 连接器集成，则此命名空间也用于相关 Kafka Connect 模式的名称，以及相应 Avro 模式的命名空间。
11	连接器捕获更改事件的表列表。

5. 运行以下命令来创建连接器资源：

```
oc create -n <namespace> -f <kafkaConnector>.yaml
```

例如，

```
oc create -n debezium -f {context}-inventory-connector.yaml
```

连接器注册到 Kafka Connect 集群，并开始针对 KafkaConnector CR 中的 spec.config.database.dbname 指定的数据库运行。连接器 pod 就绪后，Debebe 正在运行。

现在，您已准备好 验证 Debezium Oracle 部署。

7.5.4. 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium Oracle 连接器

要部署 Debezium Oracle 连接器，您必须构建包含 Debezium 连接器存档的自定义 Kafka Connect 容器镜像，然后将此容器镜像推送到容器 registry。然后，您需要创建以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR。CR 中的 image 属性指定您创建的容器镜像的名称，以运行 Debezium 连接器。您可以将此 CR 应用到部署 Red Hat AMQ Streams 的 OpenShift 实例。AMQ Streams 提供将 Apache Kafka 带到 OpenShift 的 operator 和镜像。
- 定义 Debezium Oracle 连接器的 KafkaConnector CR。将此 CR 应用到应用 KafkaConnect CR 的同一 OpenShift 实例。

先决条件

- Oracle 数据库正在运行，您完成了 设置 Oracle 以使用 Debezium 连接器 的步骤。
- AMQ Streams 部署在 OpenShift 中，并运行 Apache Kafka 和 Kafka Connect。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams
- podman 或 Docker 已安装。
-

您有一个在容器 registry 中创建和管理容器（如 quay.io 或 docker.io）的帐户和权限，您要添加将运行 Debezium 连接器的容器。

- *Kafka Connect 服务器有权访问 Maven Central，以下载 Oracle 所需的 JDBC 驱动程序。您还可以使用驱动程序的本地副本，或者从本地 Maven 存储库或其他 HTTP 服务器可用的本地副本。*

如需更多信息，请参阅 获取 Oracle JDBC 驱动程序。

流程

1. *为 Kafka Connect 创建 Debezium Oracle 容器：*

   a. *创建一个使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 的 Dockerfile 作为基础镜像。例如，在终端窗口中输入以下命令：*

```
cat <<EOF >debezium-container-for-oracle.yaml (1)
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/debezium (2)
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-oracle/2.3.4.Final-redhat-00001/debezium-connector-oracle-2.3.4.Final-redhat-00001-plugin.zip \
&& unzip debezium-connector-oracle-2.3.4.Final-redhat-00001-plugin.zip \
&& rm debezium-connector-oracle-2.3.4.Final-redhat-00001-plugin.zip
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O
https://repo1.maven.org/maven2/com/oracle/ojdbc/ojdbc8/21.1.0.0/ojdbc8-21.1.0.0.jar
USER 1001
EOF
```

项	描述
1	您可以指定您想要的任何文件名。
2	指定 Kafka Connect 插件目录的路径。如果您的 Kafka Connect 插件目录位于不同的位置，请将此路径替换为目录的实际路径。

该命令在当前目录中创建一个名为 debezium-container-for-oracle.yaml 的 Dockerfile。

b.

从您在上一步中创建的 debezium-container-for-oracle.yaml Docker 文件中构建容器镜像。在包含文件的目录中，打开终端窗口并输入以下命令之一：

```
podman build -t debezium-container-for-oracle:latest .
```

```
docker build -t debezium-container-for-oracle:latest .
```

前面的命令使用名称 debezium-container-for-oracle 构建容器镜像。

c.

将自定义镜像推送到容器 registry，如 quay.io 或内部容器 registry。容器 registry 必须可供您要部署镜像的 OpenShift 实例使用。输入以下命令之一：

```
podman push <myregistry.io>/debezium-container-for-oracle:latest
```

```
docker push <myregistry.io>/debezium-container-for-oracle:latest
```

d.

创建新的 Debezium Oracle KafkaConnect 自定义资源(CR)。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 注解和 镜像 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" (1)
spec:
  image: debezium-container-for-oracle (2)

  ...
```

项	描述
1	**metadata.annotations** 表示 **KafkaConnector** 资源用于配置在这个 Kafka Connect 集群中使用的 Cluster Operator。
2	**spec.image** 指定您创建的镜像的名称，以运行 Debezium 连接器。此属性覆盖 Cluster Operator 中的 **STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE** 变量。

e. 输入以下命令将 KafkaConnect CR 应用到 OpenShift Kafka Connect 环境：

```
oc create -f dbz-connect.yaml
```

该命令添加了一个 Kafka Connect 实例，用于指定您为运行 Debezium 连接器而创建的镜像的名称。

2. 创建一个 KafkaConnector 自定义资源来配置 Debezium Oracle 连接器实例。

您可以在 .yaml 文件中配置 Debezium Oracle 连接器，该文件指定连接器的配置属性。连接器配置可能指示 Debezium 为 schema 和表的子集生成事件，或者可能会设置属性，以便 Debezium 忽略、掩码或截断敏感、太大或不需要的指定列中的值。

以下示例配置了一个 Debezium 连接器，它连接到端口 1521 上的 Oracle 主机 IP 地址。此主机具有名为 ORCLCDB 的数据库，server1 是服务器的逻辑名称。

Oracle inventory-connector.yaml

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: inventory-connector-oracle (1)
  labels:
    strimzi.io/cluster: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: 'true'
spec:
  class: io.debezium.connector.oracle.OracleConnector (2)
  config:
    database.hostname: <oracle_ip_address> (3)
    database.port: 1521 (4)
    database.user: c##dbzuser (5)
    database.password: dbz (6)
    database.dbname: ORCLCDB (7)
    database.pdb.name : ORCLPDB1, (8)
    topic.prefix: inventory-connector-oracle (9)
    schema.history.internal.kafka.bootstrap.servers: kafka:9092 (10)
    schema.history.internal.kafka.topic: schema-changes.inventory (11)
```

表 7.16. 连接器配置设置的描述

项	描述
1	在我们使用 Kafka Connect 服务注册时连接器的名称。
2	此 Oracle 连接器类的名称。
3	Oracle 实例的地址。
4	Oracle 实例的端口号。
5	Oracle 用户的名称，如为 连接器创建用户 中所述。
6	Oracle 用户的密码，如 为 连接器创建用户 中所述。
7	要从中捕获更改的数据库名称。
8	连接器捕获更改的 Oracle 可插拔数据库的名称。仅用于容器数据库(CDB)安装。
9	主题前缀为连接器捕获更改的 Oracle 数据库服务器识别并提供命名空间。
10	此连接器用来写入和恢复 DDL 语句到数据库 schema 历史记录主题的 Kafka 代理列表。
11	连接器写入和恢复 DDL 语句的数据库模式历史记录主题的名称。本主题仅用于内部使用，不应供消费者使用。

3. *使用 Kafka Connect 创建连接器实例。例如，如果您将 KafkaConnector 资源保存在 inventory-connector.yaml 文件中，您将运行以下命令：*

```
oc apply -f inventory-connector.yaml
```

前面的命令注册 inventory-connector，连接器开始针对 KafkaConnector CR 中定义的 server1 数据库运行。

有关您可以为 Debezium Oracle 连接器设置的配置属性的完整列表，请参阅 Oracle 连接器属性。

结果

连接器启动后，它会为连接器进行配置的 Oracle 数据库 执行一致的快照。然后，连接器开始为行级操作生成数据更改事件，并将更改事件记录流传输到 Kafka 主题。

7.5.5. 配置容器数据库和非容器数据库

Oracle 数据库支持以下部署类型：

容器数据库(CDB)

可以包含多个可插拔数据库(PDB)的数据库。数据库客户端连接到每个 PDB，就像它是一个标准的非CDB 数据库一样。

非容器数据库（非CDB）

标准 Oracle 数据库，不支持创建可插拔数据库。

7.5.6. 验证 Debezium Oracle 连接器是否正在运行

如果连接器正确启动且没有错误，它会为每个连接器配置为捕获的表创建一个主题。下游应用程序可以订阅这些主题，以检索源数据库中发生的信息事件。

要验证连接器是否正在运行，您可以从 OpenShift Container Platform Web 控制台或 OpenShift CLI 工具(oc)执行以下操作：

- 验证连接器状态。
- 验证连接器是否生成主题。
- 验证主题是否填充了读取操作("op":"r")的事件，连接器在每个表的初始快照中生成。

先决条件

- Debezium 连接器部署到 OpenShift 上的 AMQ Streams。
- 已安装 OpenShift oc CLI 客户端。
- 访问 OpenShift Container Platform web 控制台。

流程

1. 使用以下方法之一检查 KafkaConnector 资源的状态：

- 在 OpenShift Container Platform Web 控制台中：

a. 导航到 Home → Search。

b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaConnector。

c. 在 KafkaConnectors 列表中，点您要检查的连接器的名称，如 inventory-connector-oracle。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- 在终端窗口中：

a. 使用以下命令：

```
oc describe KafkaConnector <connector-name> -n <project>
```

例如，

```
oc describe KafkaConnector inventory-connector-oracle -n debezium
```

该命令返回类似以下示例的状态信息：

例 7.3. KafkaConnector 资源状态

```
Name:         inventory-connector-oracle
Namespace:    debezium
Labels:       strimzi.io/cluster=debezium-kafka-connect-cluster
Annotations:  <none>
API Version:  kafka.strimzi.io/v1beta2
Kind:         KafkaConnector
```

```
...

Status:
  Conditions:
    Last Transition Time:  2021-12-08T17:41:34.897153Z
    Status:                True
    Type:                  Ready
  Connector Status:
    Connector:
      State:      RUNNING
      worker_id:  10.131.1.124:8083
    Name:         inventory-connector-oracle
    Tasks:
      Id:               0
      State:            RUNNING
      worker_id:        10.131.1.124:8083
    Type:               source
  Observed Generation:  1
  Tasks Max:            1
  Topics:
    inventory-connector-oracle.inventory
    inventory-connector-oracle.inventory.addresses
    inventory-connector-oracle.inventory.customers
    inventory-connector-oracle.inventory.geom
    inventory-connector-oracle.inventory.orders
    inventory-connector-oracle.inventory.products
    inventory-connector-oracle.inventory.products_on_hand
Events:  <none>
```

2. 验证连接器是否创建了 Kafka 主题：

- 通过 OpenShift Container Platform Web 控制台。

a. 导航到 Home → Search。

b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaTopic。

c. 在 KafkaTopics 列表中，点您要检查的主题名称，例如 inventory-connector-oracle.inventory.orders---ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- *在终端窗口中：*

a. *使用以下命令：*

```
oc get kafkatopics
```

该命令返回类似以下示例的状态信息：

例 7.4. KafkaTopic 资源状态

```
NAME                                                CLUSTER
PARTITIONS   REPLICATION FACTOR   READY
connect-cluster-configs                              debezium-kafka-cluster   1
1                  True
connect-cluster-offsets                              debezium-kafka-cluster   25
1                  True
connect-cluster-status                               debezium-kafka-cluster   5
1                  True
consumer-offsets---84e7a678d08f4bd226872e5cdd4eb527fadc1c6a
debezium-kafka-cluster   50          1                  True
inventory-connector-oracle--a96f69b23d6118ff415f772679da623fbbb99421
debezium-kafka-cluster   1           1                  True
inventory-connector-oracle.inventory.addresses---
1b6beaf7b2eb57d177d92be90ca2b210c9a56480          debezium-kafka-cluster
1           1                  True
inventory-connector-oracle.inventory.customers---
9931e04ec92ecc0924f4406af3fdace7545c483b          debezium-kafka-cluster   1
1                  True
inventory-connector-oracle.inventory.geom---
9f7e136091f071bf49ca59bf99e86c713ee58dd5            debezium-kafka-cluster
1           1                  True
inventory-connector-oracle.inventory.orders---
ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d            debezium-kafka-cluster
1           1                  True
inventory-connector-oracle.inventory.products---
df0746db116844cee2297fab611c21b56f82dcef          debezium-kafka-cluster   1
1                  True
inventory-connector-oracle.inventory.products_on_hand---
8649e0f17ffcc9212e266e31a7aeea4585e5c6b5   debezium-kafka-cluster   1
1                  True
schema-changes.inventory                             debezium-kafka-cluster
1           1                  True
strimzi-store-topic---effb8e3e057afce1ecf67c3f5d8e4e3ff177fc55          debezium-
kafka-cluster   1           1                  True
strimzi-topic-operator-kstreams-topic-store-changelog---
b75e702040b99be8a9263134de3507fc0cc4017b  debezium-kafka-cluster  1   1
True
```

3. *检查主题内容。*

- *在终端窗口中输入以下命令：*

```
oc exec -n <project>  -it <kafka-cluster> -- /opt/kafka/bin/kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=<topic-name>
```

例如，

```
oc exec -n debezium  -it debezium-kafka-cluster-kafka-0 -- /opt/kafka/bin/kafka-
console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=inventory-connector-oracle.inventory.products_on_hand
```

指定主题名称的格式与 oc describe 命令返回的格式与第 1 步中返回，例如 inventory-connector-oracle.inventory.addresses。

对于主题中的每个事件，命令会返回类似以下示例的信息：

例 7.5. Debezium 更改事件的内容

```
{"schema":{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"inventory-
connector-oracle.inventory.products_on_hand.Key"},"payload":{"product_id":101}}
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-oracle.inventory.products_on_hand.Value","field":"before"},
{"type":"struct","fields":[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-oracle.inventory.products_on_hand.Value","field":"after"},
{"type":"struct","fields":[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":
{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
```

{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connector.oracle.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transaction"}],"optional":false,"name":"inventory-connector-oracle.inventory.products_on_hand.Envelope"},"payload":{"before":null,"after":
{"product_id":101,"quantity":3},"source":{"version":"2.3.4.Final-redhat-00001","connector":"oracle","name":"inventory-connector-oracle","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"table":"products_on_hand","server_id":0,"gtid":null,"file":"oracle-bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":1638985247805,"transaction":null}}

在前面的示例中，有效负载 值显示连接器快照从表 **inventory.products_on_hand** 生成读取 **(op" ="r")事件。product_id** 记录的 **"before" 状态为 null，表示该记录不存在之前的** 值。 **"after" 状态对于 product_id** 为 **101** 的项目的 **quantity** 显示为 **3。**

7.6. DEBEZIUM ORACLE 连接器配置属性的描述

Debezium Oracle 连接器具有大量配置属性， 可用于实现应用程序的正确连接器行为。许多属性都有默认值。有关属性的信息组织如下：

- 所需的 **Debezium Oracle** 连接器配置属性
- 数据库模式历史记录连接器配置属性，用于控制 **Debezium** 如何处理从数据库 **schema** 历史记录主题读取的事件。
  - 透传数据库架构历史记录属性
- 控制 数据库驱动程序行为的直通数据库驱动程序属性。

所需的 Debezium Oracle 连接器配置属性

除非默认值可用， 否则需要以下配置属性。

属性	默认	描述
name	没有默认值	连接器的唯一名称。尝试使用相同的名称再次注册将失败。（所有 Kafka Connect 连接器都需要此属性。）
connector.class	没有默认值	连接器的 Java 类的名称。始终为 **Oracle 连接器使用 io.debezium.connector.oracle.oracleConnector** 值。

converters	没有默认值	枚举连接器可以使用的 自定义转换器 实例的符号链接列表。
		例如，**布尔值**。
		需要此属性来启用连接器以使用自定义转换器。
		
		对于您为连接器配置的每个转换器，您还必须添加一个 **.type** 属性，它指定了实现转换器接口的类的完整名称。**.type** 属性使用以下格式：
		
		***<converterSymbolicName>*.type**
		
		例如，
		
		`boolean.type: io.debezium.connector.oracle.converters.NumberOneToBooleanConverter`
		
		如果要进一步控制配置的转换器的行为，您可以添加一个或多个配置参数将值传递给转换器。要将任何其他配置参数与转换器关联，请为参数名称加上转换器的符号名作为前缀。
		
		例如，要定义 **一个选择器** 参数，用于指定 **布尔值** 转换器进程的列子集，请添加以下属性：
		
		`boolean.selector: .*MYTABLE.FLAG,.*.IS_ARCHIVED`

tasks.max	**1**	为此连接器创建的最大任务数量。Oracle 连接器始终使用单个任务，因此不使用这个值，因此默认值始终可以接受。
database.hostname	没有默认值	Oracle 数据库服务器的 IP 地址或主机名。
database.port	没有默认值	Oracle 数据库服务器的整数端口号。
database.user	没有默认值	连接器用来连接到 Oracle 数据库服务器的 Oracle 用户帐户的名称。
database.password	没有默认值	连接到 Oracle 数据库服务器时要使用的密码。

database.dbname	没有默认值	要连接的数据库的名称。在容器数据库环境中，指定根容器数据库的名称(CDB)，而不是包含的可插拔数据库(PDB)的名称。
database.url	没有默认值	指定原始数据库 JDBC URL。使用此属性在定义该数据库连接时提供灵活性。有效值包括原始 TNS 名称和 RAC 连接字符串。
database.pdb.name	没有默认值	要连接的 Oracle 可插拔数据库的名称。仅将此属性与容器数据库(CDB)安装一起使用。
topic.prefix	没有默认值	为连接器捕获更改的 Oracle 数据库服务器提供命名空间的主题前缀。您设置的值用作连接器发出的所有 Kafka 主题名称的前缀。指定在 Debezium 环境中所有连接器间唯一的主题前缀。以下字符有效：字母数字字符、连字符、句点和下划线。 **警告** 不要更改此属性的值。如果您重启后更改了 name 值，而不是继续向原始主题发出事件，连接器会将后续事件发送到名称基于新值的主题。连接器也无法恢复其数据库架构历史记录主题。
database.connection.adapter	**LogMiner**	连接器在流数据库更改时使用的适配器实现。您可以设置以下值： **logminer**（默认）：连接器使用原生 Oracle LogMiner API。

snapshot.mode	*初始*	指定连接器用来获取捕获表的快照的模式。您可以设置以下值： **always** 快照包括捕获表的结构和数据。指定这个值来填充主题，并显示每个连接器启动时捕获的表中的数据的完整表示。 **初始** 快照包括捕获表的结构和数据。指定这个值来填充来自捕获表的数据的完整表示。如果快照成功完成，则在下一个连接器启动快照不会被再次执行。 **initial_only** 快照包括捕获表的结构和数据。连接器执行初始快照，然后停止，而不处理任何后续更改。 **schema_only** 快照仅包含捕获的表的结构。如果您希望连接器只针对快照后发生的变化捕获数据，请指定这个值。 **schema_only_recovery** 这是已经捕获更改的连接器的恢复设置。重启连接器时，此设置启用恢复损坏或丢失的数据库 schema 历史记录主题。您可以定期将其设置为"清理"数据库架构历史记录主题，该主题被意外增长。数据库架构历史记录主题需要无限保留。请注意，只有在保证没有发生模式时，这个模式才会安全使用，因为连接器之前和进行快照的时间点没有发生。 快照完成后，连接器将继续从数据库的 redo 日志中读取更改事件，除非将 **snapshot.mode** 配置为 **initial_only** 时。 如需更多信息，请参阅 **snapshot.mode** 选项表。
snapshot.locking.mode	*shared*	控制连接器保存表锁定的时长。表锁定可防止在连接器执行快照时发生某些类型的更改表操作。您可以设置以下值： **shared** 启用并发访问表，但阻止任何会话获取专用表锁定。连接器在捕获表模式时获取 **ROW SHARE** 级别锁定。 **none** 防止连接器在快照期间获取任何表锁定。仅在创建快照过程中没有架构更改时使用此设置。

snapshot.include.collection.list	连接器的 **table.include.list** 属性中指定的所有表。	可选的、以逗号分隔的正则表达式列表，与表的完全限定域名(<***databaseName>.<schemaName> . <tableName> . <tableName***>)匹配，以包括在快照中。 在多租户容器数据库(CDB)环境中，正则表达式必须包含 可插拔数据库(PDB)名称，格式为 <***pdbName> . <schemaName> . <tableName>***。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 只有 POSIX 正则表达式有效。 快照只能包含在连接器的 **table.include.list** 属性中命名的表。 只有在连接器的 **snapshot.mode** 属性设置为除 **never** 的值时，此属性才会生效。 此属性不会影响增量快照的行为。

snapshot.select.statement.overrides	没有默认值	指定要包含在快照中的表行。如果您希望快照只包含表中的行的子集，请使用属性。此属性仅影响快照。它不适用于连接器从日志中读取的事件。 该属性包含以逗号分隔的、完全限定表名称列表，格式为 < ***schemaName>.<tableName>***。例如， **"snapshot.select.statement.overrides": "inventory.products,customers.orders"** 对于列表中的每个表，添加一个进一步的配置属性，用于指定连接器在获取快照时要在表上运行的 **SELECT** 语句。指定的 **SELECT** 语句决定了快照中包含的表行的子集。使用以下格式指定这个 **SELECT** 语句属性的名称： **snapshot.select.statement.overrides.*<schemaName> . <tableName>*** 例如， **snapshot.select.statement.overrides.customers.orders** Example: 在包含 soft-delete 列 **delete_flag** 的 **customers.orders** 表中，如果您希望快照只包含没有软删除的记录，请添加以下属性： "snapshot.select.statement.overrides": "customer.orders", "snapshot.select.statement.overrides.customer.orders": "SELECT * FROM [customers].[orders] WHERE delete_flag = 0 ORDER BY id DESC" 在生成的快照中，连接器只包括 **delete_flag = 0** 的记录。

schema.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，**与您要**捕获更改的模式名称匹配。只有 POSIX 正则表达式有效。不包括在 schema. **include.list 中的任何架构** 名称都会从捕获其更改中排除。默认情况下，所有非系统模式都会捕获其更改。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个名称字符串匹配；它与 schema 名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **schema.exclude.list** 属性。
include.schema.comments	**false**	指定连接器是否应该解析和发布元数据对象上的表和列注释的布尔值。启用这个选项会对内存用量造成影响。逻辑模式对象的数量和大小对 Debezium 连接器消耗的内存数量和大小有很大的影响，并可能为每个连接器添加大的字符串数据可能会非常昂贵。
schema.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与 **您不想** 捕获更改的模式名称匹配。只有 POSIX 正则表达式有效。任何名称不包含在 schema. **exclude.list 中的模式**，其更改会被捕获，但系统模式除外。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个名称字符串匹配；它与 schema 名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，请不要设置'schema.include.list' 属性。
table.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与要捕获的表的完全限定表标识符匹配。只有 POSIX 正则表达式有效。当设置此属性时，连接器只捕获指定表中的更改。每个表标识符都使用以下格式： ‹***schema_name>.<table_name***› 默认情况下，连接器会监控每个捕获的数据库中的每个非系统表。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.exclude.list** 属性。

table.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与要排除在监控中的表的完全限定域名匹配。只有 POSIX 正则表达式有效。连接器从排除列表中列出的任何表中捕获更改事件。使用以下格式为每个表指定标识符： *<schemaName>.<tableName>*。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.include.list** 属性。
column.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与更改事件消息值中包含的列的完全限定域名匹配。只有 POSIX 正则表达式有效。列的完全限定域名使用以下格式： *<Schema_name>.<table_name>.<column*_name> 。主键列始终包含在事件的键中，即使您没有使用此属性显式包含其值。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配，它与列名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **column.exclude.list** 属性。
column.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您要从更改事件消息值中排除的列的完全限定域名匹配。只有 POSIX 正则表达式有效。完全限定列名称使用以下格式： *<schema_name>.<table_name>.<*column_name> 。主键列始终包含在事件的键中，即使您使用此属性显式排除其值。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配，它与列名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，请不要设置 **column.include.list** 属性。

skip.messages.without.change	**false**	指定在包含列中没有更改时是否跳过发布消息。如果列中没有包括每个 **column.include.list** 或 **column.exclude.list** 属性的列有变化，这将过滤消息。
column.mask.hash.*hashAlgorithm*.with.salt.*salt*; column.mask.hash.v2.*hashAlgorithm*.with.salt.*salt*	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。列的完全限定域名格式为 ***<schemaName>.<tableName>.<columnName>***. 要匹配一个列的名称，Debezium 应用正则表达式，它由您指定为 *anchored* 正则表达式。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 在生成的更改事件记录中，指定列的值替换为 pseudonyms。 一个 pseudonym，它包括了通过应用指定的 *hashAlgorithm* 和 *salt* 的结果的哈希值。根据所使用的哈希函数，会维护引用完整性，而列值则替换为 pseudonyms。支持的哈希功能在 Java Cryptography 架构标准 Algorithm Name 文档的 MessageDigest 部分中 进行了描述。 在以下示例中，**CzQMA0cB5K** 是一个随机选择的 salt。 > column.mask.hash.SHA-256.with.salt.CzQMA0cB5K = inventory.orders.customerName, inventory.shipment.customerName 如有必要，pseudonym 会自动缩短为列的长度。连接器配置可以包含多个属性，用于指定不同的哈希算法和 salt。 根据所用的 *hashAlgorithm* （选择 *salt* ）和实际数据集，生成的数据集可能无法完全屏蔽。 应该使用哈希策略版本 2 来确保在不同的位置或系统中对值进行哈希处理。
binary.handling.mode	bytes	指定在更改事件中二进制(**blob**)列应该代表，包括： **bytes** 代表二进制数据作为字节数组（默认），**base64** 代表二进制数据作为 base64 编码的字符串，**base64-url-safe** -encoded String 代表二进制数据，**hex** 代表二进制数据作为十六进制编码(base16)字符串

schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列
field.name.adjustment.mode	none	指定应如何调整字段名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列 如需了解更多详细信息，请参阅 Avro 命名。
decimal.handling.mode	**精确**	指定连接器应该如何处理 **NUMBER**,**DECIMAL** 和 **NUMERIC** 列的浮点值。您可以设置以下选项之一： **精确 （默认）** 通过使用以二进制形式更改事件表示的 **java.math.BigDecimal** 值来准确代表值。 **double** 使用 **双** 值表示值。使用 **双** 值更为简单，但可能会导致精度丢失。 **字符串** 将值编码为格式化的字符串。使用 **string** 选项更易于使用，但会导致丢失有关实际类型的语义信息。更多信息请参阅 数字类型。
interval.handling.mode	**数字**	指定连接器如何处理 **interval** 列的值： **numeric** 代表使用大约微秒数的间隔。 **string** 代表间隔，使用 **P<years>Y<months>M<days>DT<hours>H<minutes>M<seconds>S** 代表。例如：**P1Y2M3DT4H5M6.78S**。

event.processing.failure.handling.mode	**fail**	指定连接器在处理事件时应如何响应异常。您可以设置以下选项之一： **fail** 传播异常（代表有问题的事件的偏移），从而导致连接器停止。 **warn** 导致有问题的事件被跳过。然后会记录有问题的事件的偏移量。 **skip** 导致有问题的事件被跳过。
max.batch.size	**2048**	一个正整数值，用于指定每个事件批处理的最大大小，以便在这个连接器的每个迭代过程中处理。
max.queue.size	**8192**	正整数值，用于指定阻塞队列可以保存的最大记录数。当 Debezium 从数据库读取事件时，它会将事件放置在阻塞队列中，然后再将它们写入 Kafka。阻塞队列可以提供从数据库读取更改事件时，连接器最快于将其写入 Kafka 的信息，或者在 Kafka 不可用时从数据库读取更改事件。当连接器定期记录偏移时，队列中保存的事件会被忽略。始终将 **max.queue.size** 的值设置为大于 **max.batch.size** 的值。
max.queue.size.in.bytes	**0**（禁用）	一个长的整数值，用于指定阻塞队列的最大卷（以字节为单位）。默认情况下，不会为阻塞队列指定卷限制。要指定队列可以消耗的字节数，请将此属性设置为正长值。 如果还设置了 **max.queue.size**，当队列的大小达到任一属性指定的限制时，写入队列将被阻止。例如，如果您设置了 **max.queue.size=1000**、和 **max.queue.size.in.bytes=5000**，在队列包含 1000 个记录后，或者队列中记录的卷达到 5000 字节后，写入队列会被阻止。
poll.interval.ms	**500** (0.5 秒)	正整数值，指定连接器在每个迭代过程中应等待的毫秒数，以便出现新更改事件。

tombstones.on.delete	**true**	控制 *delete* 事件是否后跟一个 tombstone 事件。可能会有以下值： **true** 对于每个删除操作，连接器会发出一个 *delete* 事件和一个后续 tombstone 事件。 **false** 对于每个删除操作，连接器只发出一个 *delete* 事件。 删除源记录后，一个 tombstone 事件（默认行为）可让 Kafka 完全删除在启用了 日志压缩 主题中的已删除行键的所有事件。
message.key.columns	没有默认值	指定连接器用来组成自定义消息键的表达式列表，用于更改它发布到指定表的 Kafka 主题的事件记录。 默认情况下，Debezium 使用表的主键列作为它发出的记录的消息键。在默认位置，或者为缺少主密钥的表指定一个键，您可以根据一个或多个列配置自定义消息密钥。 要为表建立自定义消息键，请列出表，后跟要用作消息键的列。每个列表条目都采用以下格式： < ***fullyQualifiedTableName> :*** &lt ***;keyColumn&*** gt; , &***lt;keyColumn***> To base a table key on multiple column name, 在列名称之间插入逗号。 每个完全限定表名称都是以下格式的一个正则表达式： <***schemaName*** **> . &** ***lt;tableName***> 属性可以包括多个表的条目。使用分号分隔列表中的表条目。 以下示例为表 **inventory.customers** 和 **purchase.orders** 设置消息键： **inventory.customers:pk1,pk2;** **(any).purchaseorders:pk3,pk4** for the table **inventory.customer**, 列 **pk1** 和 **pk2** 被指定为消息键。对于任何 **模式中的** 购买顺序表，列 **pk3** 和 **pk4** 服务器作为消息键。 对于用来创建自定义消息键的列数量没有限制。但是，最好使用指定唯一密钥所需的最小数量。

column.truncate.to.*length*.chars	没有默认值	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果您希望连接器屏蔽一组列的值，例如，如果它们包含敏感数据，则设置此属性。将 ***length*** 设置为一个正整数，替换在属性名称中的 *length* 指定的星号（*）的数量列中的数据。将 *length* 设置为 **0**（零）将指定列中的数据替换为空字符串。 列的完全限定域名会观察以下格式：< ***schemaName> . <tableName> . <columnName>***。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。
column.mask.with.*length*.chars	没有默认值	可选的、以逗号分隔的正则表达式列表，用于对更改事件中的列名称进行掩码处理，将字符替换为星号 (*)。 指定要替换的属性名称中的字符数，例如 **column.mask.with.8.chars**。 将 length 指定为正整数或零。然后，在要应用掩码的每个基于字符的列名称中添加正则表达式。 使用以下格式指定完全限定列名称：< ***schemaName> . <tableName> . <columnName>***。 连接器配置可以包含多个属性，用于指定不同的长度。

column.propagate.source.type	没有默认值	可选的、以逗号分隔的正则表达式列表，与您希望连接器发出代表列元数据的额外参数的完全限定名称匹配。当设置此属性时，连接器会将以下字段添加到事件记录的 schema 中： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*tableName> . <columnName> , 或 < schemaName&gt ; . &* lt;*tableName> . &* lt;*columnName&gt*; . 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。

datatype.propagate.source.type	没有默认值	可选的、以逗号分隔的正则表达式列表，用于指定为数据库中列定义的数据类型的完全限定名称。当设置此属性时，对于具有匹配数据类型的列，连接器会发出在 schema 中包含以下额外字段的事件记录： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*tableName> . <typeName> , 或 < schemaName&gt ; . & lt;tableName> . & lt;typeName>*。 要匹配数据类型的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据类型的整个名称字符串匹配；表达式与类型名称中可能存在的子字符串不匹配。 有关 Oracle 特定数据类型名称的列表，请查看 Oracle 数据类型映射。
heartbeat.interval.ms	**0**	指定，以毫秒为单位，连接器将信息发送到心跳主题的频率。 使用此属性来确定连接器是否继续从源数据库接收更改事件。 在没有在捕获的表中发生更改事件的扩展周期时，设置属性也很有用。 在这种情况下，虽然连接器继续读取 redo 日志，但它会发出任何更改事件信息，因此 Kafka 主题中的偏移保持不变。因为连接器不会清除从数据库读取的最新系统更改号(SCN)，所以数据库可能会保留红色日志文件，以便比必要长。如果连接器重启，扩展保留周期可能会导致连接器冗余地发送一些更改事件。 默认值 **0** 可防止连接器发送任何心跳信息。

heartbeat.action.query	没有默认值	指定连接器发送心跳消息的查询，连接器在源数据库上执行。 例如： **INSERT INTO test_heartbeat_table (text) VALUES ('test_heartbeat')** 。连接器在发出 心跳消息 后运行查询。 设置此属性并创建 heartbeat 表来接收心跳消息，以解决 Debezium 在低流量数据库中与高流量数据库同步偏移的情况。连接器将记录插入到配置的表中后，它可以从 low-traffic 数据库接收更改，并确认数据库中的 SCN 更改，以便偏移可以与代理同步。
snapshot.delay.ms	没有默认值	指定连接器在进行快照前在启动后等待的时间间隔（以毫秒为单位）。 使用此属性来防止在集群中启动多个连接器时快照中断，这可能会导致连接器的重新平衡。
snapshot.fetch.size	**10000**	指定在拍摄快照时每个表中读取的最大行数。连接器以指定大小的多个批处理读取表内容。
query.fetch.size	**10000**	指定将针对给定查询的每个数据库往返获取的行数。使用值 **0** 将使用 JDBC 驱动程序的默认获取大小。
provide.transaction.metadata	**false**	如果您希望 Debezium 生成带有事务边界的事件，并使用事务元数据增强数据事件信封，则将属性设为 **true**。 如需了解更多详细信息，请参阅 事务元数据。
log.mining.strategy	**redo_log_catalog**	指定控制 Oracle LogMiner 构建并使用给定数据字典来解析表和列 id 的 mining 策略。 **redo_log_catalog**:: 将数据字典写入在线红色日志会导致更多归档日志被随着时间的推移生成。这也启用了对捕获的表跟踪 DDL 更改，因此如果架构更改经常是理想的选择。 **online_catalog**:: 使用数据库的当前数据字典解析对象 ID，且不会将任何额外信息写入在线红色日志。这允许 LogMiner 更迅速，但无法跟踪 DDL 更改。如果捕获的表不经常或没有变化，这是理想的选择。

log.mining.query.filter.mode	none	指定控制 Oracle LogMiner 查询的构建方式的 mining query 模式。 **none**:: 查询中任何模式、表或用户名过滤时会生成查询。 **中的** 查询是使用标准的 SQL in-clause 在数据库端过滤模式、表和用户名生成的。架构、表和用户名配置 include/exclude 列表不应指定任何正则表达式，因为查询是直接使用的值构建的。 **regex**:: 查询是使用 Oracle 的**REGEXP_LIKE** 运算符在数据库侧过滤模式和表名称以及使用 SQL in-clause 的用户名生成的。schema 和 table 配置 include/exclude 列表可以安全地指定正则表达式。
log.mining.buffer.type	内存	缓冲区类型控制连接器如何管理缓冲区数据。 **内存** - 使用 JVM 进程的堆来缓冲所有事务数据。如果您不希望连接器处理大量长时间运行或大型事务，请选择这个选项。当这个选项处于活跃状态时，缓冲区状态在重启后不会被保留。重启后，从当前偏移的 SCN 值重新创建缓冲区。
log.mining.session.max.ms	0	LogMiner 会话在使用新会话前可以处于活跃状态的最大毫秒数。 对于低卷系统，当相同会话用于长时间时，LogMiner 会话可能会消耗太多 PGA 内存。默认行为是在检测到日志交换机时只使用新的 LogMiner 会话。通过将此值设置为大于 **0** 的值，这指定了 LogMiner 会话在停止前可以处于活跃状态的最大毫秒数，然后开始取消分配和重新分配 PGA 内存。
log.mining.restart.connection	false	指定 JDBC 连接是否关闭并在日志交换机上重新打开，或者当 mining 会话达到最大生命周期阈值时。 默认情况下，JDBC 连接不会在日志交换机或最大会话生命周期中关闭。 如果您使用 LogMiner 过量 Oracle SGA 增长，则应启用此项。
log.mining.batch.size.min	1000	这个连接器尝试从 redo/archive 日志中读取的最小 SCN 间隔大小。活跃的批处理大小也会增加/减少这个数量，以便在需要时调整连接器吞吐量。
log.mining.batch.size.max	100000	此连接器从 redo/archive 日志读取时使用的最大 SCN 间隔大小。

log.mining.batch.size.default	**20000**	连接器用来从 redo/archive 日志中读取数据的起始 SCN 间隔大小。这也为调整批处理大小的测量服务器 - 当批处理的当前 SCN 和开始/结束 SCN 之间的差别大于这个值时，批处理大小会增加/减少。
log.mining.sleep.time.min.ms	**0**	从 redo/archive 日志读取数据后以及再次读取数据前，连接器在读取数据后休眠的最小时间。值以毫秒为单位。
log.mining.sleep.time.max.ms	**3000**	从 redo/archive 日志读取数据后，连接器 ill 休眠的最大时间，并在再次开始读取数据前休眠。值以毫秒为单位。
log.mining.sleep.time.default.ms	**1000**	连接器从 redo/archive 日志中读取数据后休眠的时间开始，并在再次开始读取数据前休眠。值以毫秒为单位。
log.mining.sleep.time.increment.ms	**200**	在从 logminer 读取数据时，连接器用来调整最佳睡眠时间的最大时间。值以毫秒为单位。
log.mining.archive.log.hours	**0**	过去从 SYSDATE 到 mine 归档日志的小时数。当使用默认设置(**0**)时，连接器会减去所有归档日志。
log.mining.archive.log.only.mode	**false**	控制连接器是否只从归档日志或在线红色日志和归档日志（默认）中减去更改。 redo 日志使用可在任意点存档的循环缓冲区。在经常归档在线红色日志的环境中，这可能会导致 LogMiner 会话失败。与 redo logs 不同，归档日志保证可靠。将这个选项设置为 **true** 以强制连接器只减 mine 归档日志。在将连接器设置为只减归档日志后，正在提交的操作和连接器之间的延迟可能会增加。延迟程度取决于数据库配置为在线红色日志的频率。
log.mining.archive.log.only.scn.poll.interval.ms	**10000**	连接器在轮询之间休眠的毫秒数，以确定启动系统更改号是否在归档日志中。如果没有启用 **log.mining.archive.log.only.mode**，则不会使用此设置。

log.mining.transaction.retention.ms	**0**	正整数值，指定在 redo 日志交换机之间保留长时间运行的事务的毫秒数。当设置为 **0** 时，事务会被保留，直到检测到提交或回滚为止。 默认情况下，LogMiner 适配器维护所有正在运行的事务的内存缓冲。因为所有属于事务一部分的 DML 操作在检测到提交或回滚前会被缓冲，所以应该避免长时间运行的事务，以便不会溢出该缓冲区。任何超过这个配置的值的事务都会被完全丢弃，连接器不会为属于事务的操作发出任何消息。
log.mining.archive.destination.name	没有默认值	指定在减去带有 LogMiner 的归档日志时要使用的 Oracle 归档目的地。 默认行为会自动选择第一个有效的本地配置目的地。但是，您可以通过提供目的地名称（例如 **LOG_ARCHIVE_DEST_5**）来使用特定的目的地。
log.mining.username.include.list	没有默认值	要从 LogMiner 查询中包含的数据库用户列表。如果您希望捕获进程包含指定用户的更改，则设置此属性非常有用。
log.mining.username.exclude.list	没有默认值	要从 LogMiner 查询中排除的数据库用户列表。如果您希望捕获过程始终排除特定用户所做的更改，则设置此属性非常有用。
log.mining.scn.gap.detection.gap.size.min	**1000000**	指定连接器与当前和以前的 SCN 值之间的区别进行比较的值，以确定 SCN 差距是否存在。如果 SCN 值之间的区别大于指定的值，且时间差异小于 **log.mining.scn.gap.detection.time.max.ms**，则检测到 SCN 差距，连接器使用大于配置的最大批处理的 mining 窗口。
log.mining.scn.gap.detection.time.interval.max.ms	**20000**	指定一个值，以毫秒为单位，连接器与当前和以前的 SCN 时间戳之间的区别进行比较，以确定是否存在 SCN 差距。如果时间戳之间的区别小于指定的值，SCN delta 大于 **log.mining.scn.gap.detection.gap.size.min**，则检测到 SCN gap，连接器使用大于配置的最大批处理的 mining 窗口。
log.mining.flush.table.name	**LOG_MINING_FLUSH**	指定协调将 Oracle LogWriter Buffer (LGWR)刷新到 redo 日志的 flush 表的名称。通常，多个连接器可以使用相同的 flush 表。但是，如果连接器遇到表锁定争用错误，请使用此属性为每个连接器部署指定专用表。

lob.enabled	**false**	控制是否在更改事件中发出大型对象(CLOB 或 BLOB)列值。 默认情况下，更改事件具有大型对象列，但列不包含任何值。处理和管理大型对象列类型和有效负载方面有一定的开销。要捕获大型对象值并在更改事件中序列化它们，请将此选项设置为 **true**。  **注意** 使用大型对象数据类型是一个技术预览功能。
unavailable.value.placeholder	**__debezium_unavailable_value**	指定连接器提供的常量，以指示原始值保持不变，而不是由数据库提供。

rac.nodes	没有默认值	以逗号分隔的 Oracle Real Application Clusters (RAC)节点主机名或地址列表。此字段需要启用与 Oracle RAC 部署的兼容性。
		
		使用以下方法之一指定 RAC 节点列表：
		
		• 为 **database.port** 指定一个值，并为 **rac.nodes** 列表中每个地址使用指定的端口值。例如：
		
		`database.port=1521`
		`rac.nodes=192.168.1.100,192.168.1.101`
		
		• 为 **database.port** 指定一个值，并覆盖列表中一个或多个条目的默认端口。列表中可以包含使用默认 **database.port** 值的条目，以及定义其自身唯一端口值的条目。例如：
		
		`database.port=1521`
		`rac.nodes=192.168.1.100,192.168.1.101:1522`
		
		如果您使用 **database.url** 属性为数据库提供原始 JDBC URL，而不是为 **database.port** 定义值，则每个 RAC 节点条目必须明确指定端口值。

skipped.operations	**t**	以逗号分隔的操作类型列表，您希望连接器在流期间跳过。您可以将连接器配置为跳过以下类型的操作： • **c** (insert/create) • **u** (update) • **D** （删除） • **t** （截断） 默认情况下，只跳过 truncate 操作。
signal.data.collection	没有默认值	用于向连接器发送信号的数据收集的完全限定名称。https://access.redhat.com/documentation/zh-cn/red_hat_integration/2023.q4/html-single/debezium_user_guide/index#debezium-signaling-enabling-source-signaling-channel当您将此属性与 Oracle 可插拔数据库(PDB)搭配使用时，将其值设为 root 数据库的名称。 使用以下格式指定集合名称： ***<databaseName> . <schemaName& gt; . <tableName>***
signal.enabled.channels	source	为连接器启用的信号频道名称列表。默认情况下，以下频道可用： • **source** • **kafka** • **file** • **jmx**
notification.enabled.channels	没有默认值	为连接器启用的通知频道名称列表。默认情况下，以下频道可用： • **sink** • **log** • **jmx**
incremental.snapshot.chunk.size	**1024**	连接器在增量快照块期间获取并读取内存的最大行数。增加块大小可提高效率，因为快照会运行更多大小的快照查询。但是，较大的块大小还需要更多内存来缓冲快照数据。将块大小调整为提供环境中最佳性能的值。
topic.naming.strategy	**io.debezium.schema.SchemaTopicNamingStrategy**	应该用来确定数据更改、模式更改、事务、心跳事件等的 TopicNamingStrategy 类的名称，默认为 **SchemaTopicNamingStrategy**。

topic.delimiter	**.**	指定主题名称的分隔符，默认为 **.**。
topic.cache.size	**10000**	在绑定的并发哈希映射中用于保存主题名称的大小。此缓存将有助于确定与给定数据收集对应的主题名称。
topic.heartbeat.prefix	**__debezium-heartbeat**	控制连接器向其发送心跳信息的主题名称。主题名称具有此模式： *topic.heartbeat.prefix.topic.prefix* 例如，如果主题前缀是 **fulfillment**，则默认主题名称为 **__debezium-heartbeat.fulfillment**。
topic.transaction	**Transactions**	控制连接器向其发送事务元数据消息的主题名称。主题名称具有此模式： *topic.prefix.topic.transaction* 例如，如果主题前缀是 **fulfillment**，默认的主题名称为 **fulfillment.transaction**。
snapshot.max.threads	**1**	指定连接器执行初始快照时使用的线程数量。要启用并行初始快照，请将属性设置为大于 1 的值。在并行初始快照中，连接器会同时处理多个表。  **重要** 并行初始快照只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

errors.max.retries	-1	在失败前，retriable 错误（如连接错误）的最大重试次数(-1 = no limit, 0 = disabled, > 0 = num of retries)。

Debezium Oracle 连接器数据库模式历史记录配置属性

Debezium 提供了一组 schema.history.internal.* 属性，用于控制连接器如何与 schema 历史记录主题进行交互。

下表描述了用于配置 Debezium 连接器的 schema.history.internal 属性。

表 7.17. 连接器数据库架构历史记录配置属性

属性	默认	描述
schema.history.internal.kafka.topic	没有默认值	连接器存储数据库 schema 历史记录的 Kafka 主题的全名。
schema.history.internal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。此连接用于检索之前由连接器存储的数据库架构历史记录，以及用于从源数据库读取的每个 DDL 语句。每个对都应指向 Kafka Connect 进程使用的相同 Kafka 集群。
schema.history.internal.kafka.recovery.poll.interval.ms	**100**	整数值，用于指定连接器在启动/恢复期间应等待的最大毫秒数，同时轮询持久数据。默认值为 100ms。
schema.history.internal.kafka.query.timeout.ms	**3000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端获取集群信息时应等待的最大毫秒数。
schema.history.internal.kafka.create.timeout.ms	**30000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端创建 kafka 历史记录主题时应等待的最大毫秒数。
schema.history.internal.kafka.recovery.attempts	**100**	连接器在连接器恢复失败前应尝试读取持久性历史记录数据的次数上限，并显示错误。接收数据后等待的最大时间为 restore. **attempts recovery. poll.interval.ms**。
schema.history.internal.skip.unparseable.ddl	**false**	指定连接器是否应忽略格式或未知数据库语句的布尔值，或者停止处理，以便人可以解决这个问题。安全默认值为 **false**。跳过应只用于小心，因为在处理 binlog 时可能会导致数据丢失或中断。

属性	默认	描述
schema.history.internal.store.only.captured.tables.ddl	**false**	一个布尔值，用于指定连接器是否记录来自 schema 或数据库中的所有表的模式结构，还是仅从为捕获的表中指定的表。 指定以下值之一： **false （默认）** 在数据库快照过程中，连接器会记录数据库中所有非系统表的 schema 数据，包括没有指定用于捕获的表。最好保留默认设置。如果您稍后决定捕获您最初未指定用于捕获的表的更改，则连接器可以轻松地开始从这些表中捕获数据，因为它们的模式结构已经存储在 schema 历史记录主题中。Debezium 需要表的 schema 历史记录，以便它可以识别发生更改事件时存在的结构。 **true** 在数据库快照过程中，连接器只记录 Debezium 捕获更改事件的表模式。如果您更改了默认值，稍后将连接器配置为从数据库中其他表捕获数据，则连接器缺少从表中捕获更改事件所需的 schema 信息。
schema.history.internal.store.only.captured.databases.ddl	**false**	一个布尔值，用于指定连接器是否记录来自数据库实例中的所有逻辑数据库的架构结构。 指定以下值之一： **true** 连接器只记录 Debezium 捕获更改事件的逻辑数据库和模式中的表的架构结构。 **false** 连接器记录所有逻辑数据库的模式结构。 **注意** MySQL Connector 的默认值为 **true**

配置制作者和消费者客户端的直通数据库架构历史记录属性

Debezium 依赖于 Kafka producer 将模式更改写入数据库架构历史记录主题。同样，它依赖于 Kafka 使用者在连接器启动时从数据库 schema 历史记录主题中读取。您可以通过将值分配给以 schema.history.internal.producer 和 schema.history.internal.consumer ruby 前缀开头的 pass-through 配置属性来定义 Kafka producer 和 消费者 客户端的配置。直通生成者和消费者数据库模式历史记录属性控制一系列行为，如这些客户端与 Kafka 代理的连接的方式，如下例所示：

```
schema.history.internal.producer.security.protocol=SSL
schema.history.internal.producer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.producer.ssl.keystore.password=test1234
schema.history.internal.producer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.producer.ssl.truststore.password=test1234
schema.history.internal.producer.ssl.key.password=test1234
```

```
schema.history.internal.consumer.security.protocol=SSL
schema.history.internal.consumer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.consumer.ssl.keystore.password=test1234
schema.history.internal.consumer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.consumer.ssl.truststore.password=test1234
schema.history.internal.consumer.ssl.key.password=test1234
```

Debezium 从属性名称中剥离前缀，然后再将属性传递给 Kafka 客户端。

如需有关 Kafka producer 配置属性和 Kafka 使用者配置属性的更多详情，请参阅 Kafka 文档。

Debezium 连接器 Kafka 信号配置属性

Debezium 提供了一组 signal.* 属性，用于控制连接器如何与 Kafka 信号主题进行交互。

下表描述了 Kafka 信号 属性。

表 7.18. Kafka 信号配置属性

属性	默认	描述
signal.kafka.topic	<topic.prefix>-signal	连接器监控用于临时信号的 Kafka 主题的名称。 **注意** 如果禁用了 自动主题创建，您必须手动创建所需的信号主题。需要信号主题来保留信号排序。信号主题必须具有单个分区。
signal.kafka.groupId	kafka-signal	Kafka 用户使用的组 ID 的名称。
signal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。每个对都引用 Debezium Kafka Connect 进程使用的 Kafka 集群。
signal.kafka.poll.timeout.ms	100	一个整数值，用于指定连接器在轮询信号时等待的最大毫秒数。

Debezium 连接器传递信号 Kafka 使用者客户端配置属性

Debezium 连接器为信号 Kafka 使用者提供直通配置。透传信号属性以 signals.consumer.* 前缀开始。例如，连接器将 signal.consumer.security.protocol=SSL 等属性传递给 Kafka 消费者。

Debezium 从属性中剥离前缀，然后再将属性传递给 Kafka 信号消费者。

Debezium 连接器接收器通知配置属性

下表描述了 通知 属性。

表 7.19. sink 通知配置属性

属性	默认	描述
notification.sink.topic.name	没有默认值	从 Debezium 接收通知的主题名称。当您将 notification.enabled.channels 属性配置为将 sink 作为启用的通知频道之一时，需要此属性。

Debezium Oracle 连接器直通数据库驱动程序配置属性

Debezium 连接器为数据库驱动程序的直通配置提供。直通数据库属性以前缀 driver metric 开头。例如，连接器将 driver.foobar=false 等属性传递给 JDBC URL。

与 数据库架构历史记录客户端通过直通属性 一样，Debebe 会在将前缀传递给数据库驱动程序之前从属性中剥离前缀。

7.7. 监控 DEBEZIUM ORACLE 连接器性能

Debezium Oracle 连接器除了支持 Apache Zookeeper、Apache Kafka 和 Kafka Connect 的内置支持外，还提供三种指标类型。

- *快照指标; 在执行快照时监控连接器*
- *流指标; 在处理更改事件时监控连接器*
- *模式历史记录指标; 用于监控连接器模式历史记录的状态*

有关如何通过 JMX 公开这些指标的详情，请参阅监控 文档。

7.7.1. Debezium Oracle 连接器快照指标

MBean 是 debezium.oracle:type=connector-metrics,context=snapshot,server= <topic.prefix>。

快照指标不会公开，除非快照操作处于活跃状态，或者快照自上次连接器启动以来发生。

下表列出了可用的 shapshot 指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个快照事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上次启动或重置后看到的事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在快照和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	队列的空闲容量，用于在快照和主 Kafka Connect 循环之间传递事件。
TotalTableCount	int	包括在快照中的表的总数。
RemainingTableCount	int	快照必须复制的表数。
SnapshotRunning	布尔值	快照是否已启动。
SnapshotPaused	布尔值	快照是否已暂停。
SnapshotAborted	布尔值	快照是否中止。
SnapshotCompleted	布尔值	快照是否完成。

属性	类型	描述
SnapshotDurationInSeconds	long	快照为止所花费的秒数，即使未完成也是如此。也包括快照暂停的时间。
SnapshotPausedDurationInSeconds	long	快照暂停的秒数。如果快照暂停几次，暂停的时间会添加。
RowsScanned	Map<String, Long>	包含快照中每个表的行数的映射。表会在处理过程中逐步添加到映射中。更新每个 10,000 行扫描并在完成表后。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

连接器还在执行增量快照时提供以下额外快照指标：

属性	类型	描述
ChunkId	字符串	当前快照块的标识符。
ChunkFrom	字符串	定义当前块的主密钥集的下限。
ChunkTo	字符串	定义当前块的主密钥集的上限。
TableFrom	字符串	当前快照表的主密钥集的下限。
TableTo	字符串	当前快照表的主密钥集的上限。

7.7.2. Debezium Oracle 连接器流指标

MBean 是 debezium.oracle:type=connector-metrics,context=streaming,server=<topic.prefix>。

下表列出了可用的流指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个流事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上一次启动或指标重置以来看到的事件总数。
TotalNumberOfCreateEventsSeen	long	此连接器自上次启动或指标重置以来看到的创建事件总数。
TotalNumberOfUpdateEventsSeen	long	此连接器自上次启动或指标重置以来看到的更新事件总数。
TotalNumberOfDeleteEventsSeen	long	此连接器自上次启动或指标重置以来看到的删除事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的队列的可用容量。
Connected	布尔值	表示连接器目前是否连接到数据库服务器的标记。
MilliSecondsBehindSource	long	最后一次更改事件时间戳和连接器处理它之间的毫秒数。这些值将讨论运行数据库服务器和连接器的计算机上时钟之间的任何区别。
NumberOfCommittedTransactions	long	已提交的已处理事务的数量。
SourceEventPosition	Map<String, String>	最后收到的事件的协调。

属性	类型	描述
LastTransactionId	字符串	最后处理事务的事务的事务标识符。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

Debezium Oracle 连接器还提供以下额外流指标：

表 7.20. 其他流指标的描述

属性	类型	描述
CurrentScn	BigInteger	最近处理的系统更改号。
OldestScn	BigInteger	事务缓冲区中最旧的系统更改号。
CommittedScn	BigInteger	最后提交的系统更改来自事务缓冲区的编号。
OffsetScn	BigInteger	系统更改号当前写入连接器的偏移量。
CurrentRedoLogFileName	string[]	当前 mined 的日志文件数组。
MinimumMinedLogCount	long	为任何 LogMiner 会话指定的最小日志数。
MaximumMinedLogCount	long	为任何 LogMiner 会话指定的最大日志数。
RedoLogStatus	string[]	每个 mined 日志文件的当前状态数组，格式为 ***filename|status***.
SwitchCounter	int	数据库为最后一天执行日志交换机的次数。

属性	类型	描述
LastCapturedDmlCount	long	最后一次 LogMiner 会话查询中观察到的 DML 操作数量。
MaxCapturedDmlInBatch	long	处理单个 LogMiner 会话查询时观察到的最大 DML 操作数。
TotalCapturedDmlCount	long	观察到的 DML 操作总数。
FetchingQueryCount	long	执行的 LogMiner 会话查询总数（也称为批处理）。
LastDurationOfFetchQueryInMilliseconds	long	最后一次 LogMiner 会话查询的获取周期（以毫秒为单位）。
MaxDurationOfFetchQueryInMilliseconds	long	任何 LogMiner 会话查询的最长持续时间（以毫秒为单位）。
LastBatchProcessingTimeInMilliseconds	long	处理最后 LogMiner 查询批处理结果的持续时间以毫秒为单位。
TotalParseTimeInMilliseconds	long	解析 DML 事件 SQL 语句的时间（毫秒）。
LastMiningSessionStartTimeInMilliseconds	long	启动最后一次 LogMiner 会话的时间（以毫秒为单位）。
MaxMiningSessionStartTimeInMilliseconds	long	启动 LogMiner 会话的最长时间（以毫秒为单位）。
TotalMiningSessionStartTimeInMilliseconds	long	连接器启动 LogMiner 会话的总持续时间（以毫秒为单位）。
MinBatchProcessingTimeInMilliseconds	long	单个 LogMiner 会话处理结果的最短持续时间（以毫秒为单位）。
MaxBatchProcessingTimeInMilliseconds	long	单一 LogMiner 会话处理结果的最长时间（以毫秒为单位）。

属性	类型	描述
TotalProcessingTimeInMilliseconds	long	LogMiner 会话处理结果的总持续时间（以毫秒为单位）。
TotalResultSetNextTimeInMilliseconds	long	JDBC 驱动程序花费的总持续时间（以毫秒为单位）从日志减法视图中获取要处理的下一行。
TotalProcessedRows	long	从所有会话的日志减法视图中处理的行总数。
BatchSize	int	由每个数据库往返的日志减号查询获取的条目数。
MillisecondToSleep betweenMiningQuery	long	连接器在从日志减减视图中获取另一批处理结果前休眠的毫秒数。
MaxBatchProcessingThroughput	long	从日志减法视图中处理的最大行/秒数。
AverageBatchProcessingThroughput	long	从日志减减处理的平均行数/秒。
LastBatchProcessingThroughput	long	从最后一个批处理的日志减减视图中处理的平均行/秒数。
NetworkConnectionProblemsCounter	long	检测到的连接问题的数量。
HoursToKeepTransactionInBuffer	int	事务的小时数由连接器的内存中缓冲区保留，而不会在丢弃前提交或回滚。如需更多信息，请参阅 log.mining.transaction.retention.ms。
NumberOfActiveTransactions	long	事务缓冲区中当前活动事务的数量。
NumberOfCommittedTransactions	long	事务缓冲区中提交事务的数量。
NumberOfRolledBackTransactions	long	在事务缓冲区中回滚事务的数量。
CommitThroughput	long	事务缓冲区中每秒提交事务的平均数量。

属性	类型	描述
RegisteredDmlCount	long	事务缓冲区中注册的 DML 操作数量。
LagFromSourceInMilliseconds	long	事务日志中发生更改时以及添加到事务缓冲区的时间差异（毫秒）。
MaxLagFromSourceInMilliseconds	long	事务日志中发生更改时以及添加到事务缓冲区的时间之间的最大时间差（毫秒）。
MinLagFromSourceInMilliseconds	long	在事务日志中发生更改时以及添加到事务缓冲区的时间之间的最短时间差（毫秒）。
AbandonedTransactionIds	string[]	最新取消的事务标识符的数组，因为其年龄从事务缓冲区中删除。详情请参阅 log.mining.transaction.retention.ms。
RolledBackTransactionIds	string[]	在事务缓冲区中重试并回滚的最新事务标识符的数组。
LastCommitDurationInMilliseconds	long	最后一次事务缓冲区提交操作的持续时间（以毫秒为单位）。
MaxCommitDurationInMilliseconds	long	最长的事务缓冲区提交操作的持续时间（以毫秒为单位）。
ErrorCount	int	检测到的错误数量。
WarningCount	int	检测到的警告数量。
ScnFreezeCount	int	检查系统更改号以提前检查的次数，并保持不变。高值可以表示长期运行的事务正在进行，并阻止连接器刷新最近处理的系统更改号到连接器的偏移。当条件最佳时，该值应接近或等于 0。

属性	类型	描述
UnparsableDdlCount	int	已检测到的 DDL 记录数量，但无法通过 DDL 解析器解析。这应该始终为 0 ；但是，当允许无法解析的 DDL 时，可以使用此指标来确定是否将任何警告写入连接器日志。
MiningSessionUserGlobalAreaMemoryInBytes	long	当前 mining 会话的用户全局区域(UGA)内存消耗（以字节为单位）。
MiningSessionUserGlobalAreaMaxMemoryInBytes	long	所有 mining 会话中的最大 mining 会话用户全局区域(UGA)内存消耗（以字节为单位）。
MiningSessionProcessGlobalAreaMemoryInBytes	long	当前 mining 会话的进程全局区域(PGA)内存消耗（以字节为单位）。
MiningSessionProcessGlobalAreaMaxMemoryInBytes	long	所有 mining 会话的最大 mining 会话全局区域(PGA)内存消耗（以字节为单位）。

7.7.3. Debezium Oracle 连接器模式历史记录指标

MBean 是 debezium.oracle:type=connector-metrics,context=schema-history,server=<topic.prefix>。

下表列出了可用的模式历史记录指标。

属性	类型	描述
Status	字符串	**STOPPED** 之一，**RECOVERING** （从存储恢复历史记录），**RUNNING** 描述数据库架构历史记录的状态。
RecoveryStartTime	long	恢复启动时的 epoch 秒的时间（以秒为单位）。
ChangesRecovered	long	在恢复阶段读取的更改数量。

属性	类型	描述
ChangesApplied	long	恢复和运行时期间应用的模式更改总数。
MilliSecondsSinceLastRecoveredChange	long	从历史记录存储中恢复自上次更改以来经过的毫秒数。
MilliSecondsSinceLastAppliedChange	long	从上次更改被应用后经过的毫秒数。
LastRecoveredChange	字符串	从历史记录存储中恢复的最后一个更改的字符串表示。
LastAppliedChange	字符串	最后应用的更改的字符串表示。

7.8. ORACLE 连接器常见问题

是否支持 Oracle 11g？

Oracle 11g 不支持；但是，我们的目标是以最佳的方式与 Oracle 11g 向后兼容。我们依赖社区与 Oracle 11g 沟通兼容性问题，并在识别回归时提供 bug 修复。

是否弃用了 Oracle LogMiner？

不，Oracle 只弃用了 Oracle LogMiner 在 Oracle 12c 中带有 Oracle LogMiner 的持续 mining 选项，并删除了从 Oracle 19c 开始的该选项。Debezium Oracle 连接器不依赖于这个选项来正常工作，因此可以安全地与 Oracle 的新版本一起使用，且不会影响。

如何更改偏移中的位置？

Debezium Oracle 连接器在偏移中维护两个关键值，名为 scn，另一个名为 commit_scn 的字段。scn 字段是一个字符串，代表在捕获更改时使用的连接器的低水位开始位置。

1. *找到包含连接器偏移的主题名称。这基于设置为 offset.storage.topic 配置属性的值进行配置。*

2. *找到连接器的最后偏移，存储它的密钥，并标识用于存储偏移的分区。这可以通过 Kafka 代理安装提供的 kafkacat 工具脚本来完成。一个示例可能类似如下：*

```
kafkacat -b localhost -C -t my_connect_offsets -f 'Partition(%p) %k %s\n'
Partition(11) ["inventory-connector",{"server":"server1"}] {"scn":"324567897",
"commit_scn":"324567897: 0x2832343233323:1"}
```

inventory-connector 的密钥是 ["inventory-connector",{"server":"server1"}]，分区是 11，最后一个偏移是遵循该键的内容。

3. *要移动到以前的偏移中，应该停止连接器，且必须发出以下命令：*

```
echo '["inventory-connector",{"server":"server1"}]|
{"scn":"3245675000","commit_scn":"324567500"}' | \
kafkacat -P -b localhost -t my_connect_offsets -K \| -p 11
```

这会写入给定 key 和 offset 值的 my_connect_offsets 主题的分区 11。在本例中，我们将连接器重新定向到 SCN 3245675000 而不是 324567897。

如果连接器无法找到给定偏移 SCN 的日志，会发生什么？

Debezium 连接器在连接器偏移中维护低和高水位线 SCN 值。低水位线 SCN 代表起始位置，必须存在于可用的在线红色或存档日志中，以便连接器成功启动。当连接器报告它找不到这个偏移 SCN 时，这表示仍然可用的日志不包含 SCN，因此连接器无法从其离开的位置进行减弱更改。

发生这种情况时，有两个选项。第一个是删除连接器的历史记录主题和偏移，并重启连接器，按照建议执行新的快照。这样可保证任何主题消费者不会发生数据丢失。第二个方法是手动操作偏移，将 SCN 提升到 redo 或 archive 日志中可用的位置。这会导致在旧的 SCN 值和新提供的 SCN 值之间发生的更改会丢失，且不会写入主题。不建议这样做。

各种 mining 策略之间有什么区别？

Debezium Oracle 连接器为 log.mining.strategy 提供两个选项。

默认值为 redo_in_catalog，这指示连接器每次检测到日志交换机时将 Oracle 数据字典写入 redo 日志。在解析 redo 和归档日志时，Oracle LogMiner 需要这个数据字典来跟踪模式更改。这个选项将生成超过常规的归档日志数，但允许捕获的表实时操作，而不会对捕获数据更改产生任何影响。这个选项通常需要更多 Oracle 数据库内存，并会导致 Oracle LogMiner 会话和进程在每次日志切换后启动的时间稍长。

备用选项 online_catalog 不会将数据字典写入 redo 日志。相反，Oracle LogMiner 始终使用包含表结构当前状态的在线数据字典。这也意味着，如果表的结构发生变化，且不再与在线数据字典匹配，如果表的结构已更改，Oracle LogMiner 将无法解析表或列名称。如果捕获的表受到频繁的模式更改，则不应使用这个 mining 策略选项。重要的是，所有数据更改都会锁定模式更改，以便所有更改都已从表的日志捕获，停止连接器，应用模式更改，然后重新启动连接器，并恢复表中的数据更改。这个选项需要较少的 Oracle 数据库内存和 Oracle LogMiner 会话，因为数据字典不需要由 LogMiner 进程加载或入门。

为什么连接器似乎停止捕获 AWS 上的更改？

由于 AWS 网关 Load Balancer 上 350 秒的固定空闲超时，需要超过 350 秒的 JDBC 调用可以无限期地挂起。

如果调用 Oracle LogMiner API 需要超过 350 秒才能完成，则可能会触发超时，从而导致 AWS 网关 Load Balancer 挂起。例如，当一个 LogMiner 会话处理大量数据与 Oracle 的定期检查点任务同时运行时，可能会出现这样的超时。

要防止在 AWS 网关 Load Balancer 上出现超时，请以 root 或超级用户身份执行以下步骤，从 Kafka Connect 环境启用 keep-alive 数据包：

1. 在终端中运行以下命令：

```
sysctl -w net.ipv4.tcp_keepalive_time=60
```

2. 编辑 /etc/sysctl.conf 并设置以下变量值，如下所示：

```
net.ipv4.tcp_keepalive_time=60
```

3. 为 Oracle 连接器重新配置 Debezium，以使用 database.url 属性而不是 database.hostname，并添加 (ENABLE=broken) Oracle 连接字符串描述符，如下例所示：

```
database.url=jdbc:oracle:thin:username/password!@(DESCRIPTION=
(ENABLE=broken)(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
(Host=hostname)(Port=port)))(CONNECT_DATA=(SERVICE_NAME=serviceName)))
```

前面的步骤将 TCP 网络堆栈配置为每 60 秒发送 keep-alive 数据包。因此，当对 LogMiner API 的 JDBC 调用超过 350 秒时，AWS Gateway Load Balancer 不会超时，使连接器能够继续从数据库的事务日志中读取更改。

ORA-01555 的原因是什么以及如何处理它？

Debezium Oracle 连接器在初始快照阶段执行时使用闪存查询。闪存查询是特殊的查询类型，它依赖于闪存区域（由数据库的 UNDO_RETENTION 数据库参数维护），根据表在给定时间的内容，或在给定 SCN 时返回查询的结果。默认情况下，Oracle 通常只维护大约 15 分钟的撤消或闪存区域，除非数据库管理员已增加或减少。对于捕获大表的配置，可能需要超过 15 分钟，或者您配置的 UNDO_RETENTION 执行初始快照，最终会导致这个异常：

```
ORA-01555: snapshot too old: rollback segment number 12345 with name
"_SYSSMU11_1234567890$" too small
```

处理此例外的第一个方法是与您的数据库管理员合作，并查看他们是否可以临时增加 UNDO_RETENTION 数据库参数。这不需要重新启动 Oracle 数据库，因此可以在不影响数据库可用性的情况下在线完成此操作。但是，如果表空间没有足够的空间来存储必要的撤销数据，则更改这仍然可能会导致上述异常或"快照太旧"异常。

处理此例外的第二个方法是不依赖于初始快照，将 snapshot.mode 设置为 schema_only，而是依赖增量快照。增量快照不依赖于闪存查询，因此不受到 ORA-01555 异常的影响。

ORA-04036 的原因是什么以及如何处理它？

当数据库更改不常时，De Debezium Oracle 连接器可能会报告 ORA-04036 异常。在检测到日志切换前，启动了 Oracle LogMiner 会话并重新使用。会话被重新使用，因为它为 Oracle LogMiner 提供最佳性能利用率，但应该有长时间运行的 mining 会话，这可能会导致过量 PGA 内存用量，最终导致以下异常：

```
ORA-04036: PGA memory used by the instance exceeds PGA_AGGREGATE_LIMIT
```

通过指定 Oracle 交换机红色的日志或 Debezium Oracle 连接器可以重复使用 mining 会话，可以避免此异常。Debezium Oracle 连接器提供了一个配置选项 log.mining.session.max.ms，它控制当前 Oracle LogMiner 会话在关闭和启动新会话前可以重新使用的时间。这允许数据库资源保持在检查，而不超过数据库允许的 PGA 内存。

ORA-01882 的原因是什么以及如何处理它？

Debezium Oracle 连接器可能会在连接到 Oracle 数据库时报告以下异常：

```
ORA-01882: timezone region not found
```

当时区信息无法被 JDBC 驱动程序正确解决时，会出现这种情况。为解决与此驱动程序相关的问题，需要告知驱动程序不使用地区解析时区详细信息。这可以通过使用 driver.oracle.jdbc.timezoneAsRegion=false 指定驱动程序 pass through 属性来实现。

ORA-25191 的原因是什么以及如何处理它？

Debezium Oracle 连接器会自动忽略索引组织表(IOT)，因为 Oracle LogMiner 不支持它们。但是，如果抛出 ORA-25191 异常，这可能是因为映射的唯一基础情况，并且可能需要自动排除这些映射的额外规则。ORA-25191 异常示例可能类似如下：

```
ORA-25191: cannot reference overflow table of an index-organized table
```

如果抛出 ORA-25191 异常，请引发 JIRA 问题，其中包含有关表的详细信息，以及与其它父表相

关的映射，等等。作为临时解决方案，可以调整 include/exclude 配置选项，以防止连接器访问这些表。

第 8 章 POSTGRESQL 的 DEBEZIUM 连接器

Debezium PostgreSQL 连接器捕获 PostgreSQL 数据库模式中的行级更改。有关与连接器兼容的 PostgreSQL 版本的详情，请参考 Debezium 支持的配置页面。

第一次连接到 PostgreSQL 服务器或集群时，连接器会获取所有模式的一致性快照。完成该快照后，连接器会持续捕获插入、更新和删除数据库内容以及提交到 PostgreSQL 数据库的行级更改。连接器生成数据更改事件记录，并将其流传输到 Kafka 主题。对于每个表，默认行为是连接器所有生成的事件都流传输到该表的独立 Kafka 主题。应用程序和服务会消耗来自该主题的数据更改事件记录。

使用 Debezium PostgreSQL 连接器的信息和步骤进行组织，如下所示：

8.1. DEBEZIUM POSTGRESQL 连接器概述

PostgreSQL 的逻辑解码 功能是在版本 9.4 中引入的。它是一个允许提取提交至事务日志的更改的机制，并通过 输出插件 帮助以用户友好的方式处理这些更改。output 插件可让客户端使用更改。

PostgreSQL 连接器包含两个主要部分，它们协同工作来读取和处理数据库更改：

- pgoutput 是 PostgreSQL 10+ 中的标准逻辑解码输出插件。这是此 Debezium 发行版本中唯一支持的逻辑解码输出插件。此插件由 PostgreSQL 社区维护，供 PostgreSQL 本身用于 逻辑复制。此插件始终存在，因此不需要安装额外的库。Debezium 连接器将原始复制事件流直接解释为更改事件。

- Java 代码（实际 Kafka Connect 连接器），通过使用 PostgreSQL 的 流传输复制协议 和 PostgreSQL JDBC 驱动程序 读取逻辑解码输出插件生成的更改。

连接器会为捕获的每行级别的插入、更新和删除操作生成更改事件，并在单独的 Kafka 主题中为每个表发送更改事件记录。客户端应用程序读取与感兴趣的数据库表对应的 Kafka 主题，并可响应它们从这些主题接收的每行级别的事件。

PostgreSQL 通常会在一段时间后清除 write-ahead 日志(WAL)片段。这意味着连接器没有对数据库进行的所有更改的完整历史记录。因此，当 PostgreSQL 连接器首先连接到特定的 PostgreSQL 数据库时，它首先对每个数据库模式执行 一致的快照。连接器完成快照后，它会从执行快照的确切点继续流出更改。这样，连接器会以所有数据的一致性视图开始，且不会省略在快照被生成时所做的任何更改。

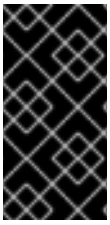
连接器可以接受故障。当连接器读取更改并生成事件时，它会记录每个事件的 WAL 位置。如果连接器因任何原因（包括通信故障、网络问题或崩溃）停止，在重启连接器时，在重启连接器会继续读取最后一次关闭的 WAL。这包括快照。如果连接器在快照期间停止，连接器会在重启时启动新快照。

重要

连接器依赖于并反映 PostgreSQL 逻辑解码功能，其有以下限制：

- 逻辑解码不支持 DDL 更改。这意味着连接器无法将 DDL 更改事件报告回消费者。
- 逻辑解码复制插槽只支持在 主服务器中。当有 PostgreSQL 服务器集群时，连接器只能在活跃的 主服务器中 运行。它无法在 热 或 温 待机副本上运行。如果主服务器 失败或被降级，连接器会停止。主服务器 恢复后，您可以重启连接器。如果不同的 PostgreSQL 服务器已提升到 主，请在重启连接器前调整连接器配置。

出错时的行为 描述了连接器在出现问题时如何响应。



重要

Debezium 目前仅支持使用 UTF-8 字符编码的数据库。使用单字节字符编码时，无法正确处理包含扩展 ASCII 代码字符的字符串。

8.2. DEBEZIUM POSTGRESQL 连接器的工作方式

为了优化配置和运行 Debezium PostgreSQL 连接器，了解连接器如何执行快照、流更改事件、决定 Kafka 主题名称并使用元数据非常有用。

详情包括在以下主题中：

- 第 8.2.2 节 “Debezium PostgreSQL 连接器如何执行数据库快照”
- 第 8.2.3 节 “临时快照”
- 第 8.2.4 节 “增量快照”
- 第 8.2.5 节 “Debezium PostgreSQL 连接器流更改事件记录”

- 第 8.2.6 节 “接收 Debezium PostgreSQL 更改事件记录的默认 Kafka 主题名称”
- 第 8.2.7 节 “Debezium PostgreSQL 连接器生成的事件代表事务边界”

8.2.1. PostgreSQL 连接器的安全性

要使用 Debezium 连接器从 PostgreSQL 数据库流更改，连接器必须使用数据库中的特定权限运行。虽然授予必要的特权的一种方法是为用户提供 超级用户特权，这样做可能会将您的 PostgreSQL 数据暴露给未经授权的访问。最好创建一个专用的 Debezium 复制用户，而不是为 Debezium 用户授予特定的特权。

有关为 Debezium PostgreSQL 用户配置特权的更多信息，请参阅 设置权限。有关 PostgreSQL 逻辑复制安全性的更多信息，请参阅 PostgreSQL 文档。

8.2.2. Debezium PostgreSQL 连接器如何执行数据库快照

大多数 PostgreSQL 服务器都配置为不保留 WAL 段中数据库的完整历史记录。这意味着 PostgreSQL 连接器无法通过只读取 WAL 来查看数据库的完整历史记录。因此，连接器首次启动时，它会执行数据库的初始 一致快照。

您可以在以下部分找到有关快照的更多信息：

- 第 8.2.3 节 “临时快照”
- 第 8.2.4 节 “增量快照”

初始快照的默认工作流行为

执行快照的默认行为由以下步骤组成。您可以通过将 snapshot.mode 连接器配置属性设置为 初始 以外的值来更改此行为。

1. 使用 SERIALIZABLE、READ ONLY、DEFERRABLE 隔离级别启动事务，以确保此事务中的后续读取会根据一个一致的数据版本进行。由于后续 INSERT、UPDATE 和 DELETE 操作而对数据的任何更改都不对这个事务可见。

2. 阅读服务器的事务日志中的当前位置。

3. 扫描数据库表和模式，为每个行生成一个 READ 事件，并将该事件写入适当的表特定 Kafka 主题。

4. 提交事务。

5. 在连接器偏移中记录快照成功完成。

如果连接器失败，会在步骤 1 开始后重新平衡或停止，但在重启连接器开始新快照前。连接器完成其初始快照后，PostgreSQL 连接器会从在第 2 步中读取的位置继续流。这样可确保连接器不会错过任何更新。如果连接器因为任何原因而再次停止，则在重启时，连接器会继续从之前关闭的位置进行流更改。

表 8.1. snapshot.mode 连接器配置属性的选项

选项	描述
always	连接器总是在启动时执行快照。快照完成后，连接器将继续在上述序列中从第 3 步进行流更改。这个模式在以下情况下很有用： ● 已知一些 WAL 段已被删除，且不再可用。 ● 集群失败后，会提升新的主设备。**always** 快照模式可确保连接器不会错过在新主提升后所做的任何更改，但在连接器在新主上重启前。
never	连接器永远不会执行快照。当以这种方式配置连接器时，其启动时的行为如下。如果 Kafka offsets 主题中存在之前存储的 LSN，则连接器将继续从该位置流更改。如果没有存储 LSN，则连接器会在服务器上创建 PostgreSQL 逻辑复制插槽时从时间点开始流更改。只有在您知道所有关注的数据仍然反映在 WAL 中时，**never** 快照模式很有用。
初始（默认）	当没有 Kafka offsets 主题时，连接器会执行数据库快照。数据库快照完成后，将写入 Kafka offsets 主题。如果 Kafka offsets 主题中存在之前存储的 LSN，则连接器将继续从该位置流更改。
initial_only	连接器执行数据库快照并在流任何更改事件记录前停止。如果连接器已启动但在停止前没有完成快照，连接器会重启快照过程并在快照完成后停止。
exported	弃用，所有模式都是没有锁定的。

8.2.3. 临时快照

默认情况下，连接器仅在首次启动后运行初始快照操作。在正常情况下，在这个初始快照后，连接器

不会重复快照过程。连接器捕获的任何更改事件数据都只通过流处理。

然而，在某些情况下，连接器在初始快照期间获得的数据可能会过时、丢失或不完整。为了提供总结表数据的机制，Debezium 包含一个执行临时快照的选项。数据库中的以下更改可能会导致执行临时快照：

- 连接器配置会被修改为捕获不同的表集合。
- Kafka 主题已删除，必须重建。
- 由于配置错误或某些其他问题导致数据损坏。

您可以通过启动所谓的 临时快照来为之前捕获的表重新运行快照。临时快照需要使用 信号表。您可以通过向 Debezium 信号表发送信号请求来发起临时快照。

当您启动现有表的临时快照时，连接器会将内容附加到表已存在的主题中。如果删除了之前存在的主题，如果启用了 自动主题创建，Debezium 可以自动创建主题。

临时快照信号指定要包含在快照中的表。快照可以捕获整个数据库的内容，或者仅捕获数据库中表的子集。另外，快照也可以捕获数据库中表的内容子集。

您可以通过将 execute-snapshot 消息发送到信号表来指定要捕获的表。将 execute-snapshot 信号类型设置为 增量，并提供快照中包含的表名称，如下表所述：

表 8.2. 临时 execute-snapshot 信号记录的示例

字段	默认	值
type	**incremental**	指定您要运行的快照类型。 设置类型是可选的。目前，您只能请求 **增量** 快照。
data-collections	*N/A*	包含与要快照的表的完全限定域名匹配的正则表达式的数组。 名称的格式与 **signal.data.collection** 配置选项的格式相同。
additional-condition	*N/A*	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。
surrogate-key	*N/A*	可选字符串，指定连接器在快照过程中用作表的主键的列名称。

字段	默认	值

触发临时快照

您可以通过向信号表中添加 execute-snapshot 信号类型的条目来发起临时快照。连接器处理消息后，它会开始快照操作。快照进程读取第一个和最后一个主密钥值，并使用这些值作为每个表的开头和结束点。根据表中的条目数量以及配置的块大小，Debezium 会将表划分为块，并一次性执行每个块的快照。

目前，execute-snapshot 操作类型仅触发 增量快照。如需更多信息，请参阅 增加快照。

8.2.4. 增量快照

为了提供管理快照的灵活性，Debezium 包含附加快照机制，称为 增量快照。增量快照依赖于 Debezium 机制 向 Debezium 连接器发送信号。

在增量快照中，除了一次捕获数据库的完整状态，就像初始快照一样，Debebe 会在一系列可配置的块中捕获每个表。您可以指定您希望快照捕获的表 以及每个块的大小。块大小决定了快照在数据库的每个获取操作期间收集的行数。增量快照的默认块大小为 1024 行。

当增量快照进行时，Debebe 使用 watermarks 跟踪其进度，维护它捕获的每个表行的记录。与标准初始快照过程相比，捕获数据的阶段方法具有以下优点：

- *您可以使用流化数据捕获并行运行增量快照，而不是在快照完成前进行后流。连接器会在快照过程中从更改日志中捕获接近实时事件，且操作都不会阻止其他操作。*
- *如果增量快照的进度中断，您可以在不丢失任何数据的情况下恢复它。在进程恢复后，快照从停止的点开始，而不是从开始计算表。*
- *您可以随时根据需要运行增量快照，并根据需要重复该过程以适应数据库更新。例如，您可以在修改连接器配置后重新运行快照，以将表添加到其 table.include.list 属性中。*

增量快照过程

当您运行增量快照时，Debezium 会按主键对每个表进行排序，然后根据 配置的块大小 将表分成块。然后，按块的工作块会捕获块中的每个表行。对于它捕获的每行，快照会发出 READ 事件。该事件代表

块的快照开始时的行值。

当快照继续进行时，其他进程可能会继续访问数据库，可能会修改表记录。为了反映此类更改，INSERT、UPDATE 或 DELETE 操作会按照常常提交到事务日志。同样，持续 Debezium 流进程将继续检测这些更改事件，并将相应的更改事件记录发送到 Kafka。

Debezium 如何使用相同的主密钥在记录间解决冲突

在某些情况下，streaming 进程发出的 UPDATE 或 DELETE 事件会停止序列。也就是说，流流过程可能会发出一个修改表行的事件，该事件捕获包含该行的 READ 事件的块。当快照最终为行发出对应的 READ 事件时，其值已被替换。为确保以正确的逻辑顺序处理到达序列的增量快照事件，Debebe 使用缓冲方案来解析冲突。仅在快照事件和流化事件之间发生冲突后，De Debezium 会将事件记录发送到 Kafka。

快照窗口

为了帮助解决修改同一表行的后期事件和流化事件之间的冲突，Debebe 会使用一个所谓的 快照窗口。快照窗口分解了增量快照捕获指定表块数据的间隔。在块的快照窗口打开前，Debebe 会使用其常见行为，并将事件从事务日志直接下游发送到目标 Kafka 主题。但从特定块的快照打开后，直到关闭为止，De-duplication 步骤会在具有相同主密钥的事件之间解决冲突。

对于每个数据收集，Debezium 会发出两种类型的事件，并将其存储在单个目标 Kafka 主题中。从表直接捕获的快照记录作为 READ 操作发送。同时，当用户继续更新数据收集中的记录，并且会更新事务日志来反映每个提交，Debezium 会为每个更改发出 UPDATE 或 DELETE 操作。

当快照窗口打开时，Debezium 开始处理快照块，它会向内存缓冲区提供快照记录。在快照窗口期间，缓冲区中 READ 事件的主密钥与传入流事件的主键进行比较。如果没有找到匹配项，则流化事件记录将直接发送到 Kafka。如果 Debezium 检测到匹配项，它会丢弃缓冲的 READ 事件，并将流化记录写入目标主题，因为流的事件逻辑地取代静态快照事件。在块关闭的快照窗口后，缓冲区仅包含 READ 事件，这些事件不存在相关的事务日志事件。Debezium 将这些剩余的 READ 事件发送到表的 Kafka 主题。

连接器为每个快照块重复这个过程。

警告

PostgreSQL 的 Debezium 连接器不支持增量快照运行时的模式更改。如果在增量快照启动前执行 schema 更改，但在以后发送信号，passthrough 配置选项 database.autosave 被设置为 conservative 以正确处理 schema 的更改。

8.2.4.1. 触发增量快照

目前，启动增量快照的唯一方法是向源数据库上的 信号表发送临时快照 信号。

作为 SQL INSERT 查询，您将向信号提交信号。

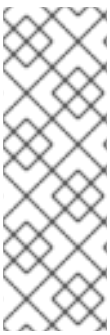
在 Debezium 检测到信号表中的更改后，它会读取信号并运行请求的快照操作。

您提交的查询指定要包含在快照中的表，并可以选择指定快照操作的类型。目前，快照操作的唯一有效选项是默认值 incremental。

要指定快照中包含的表，请提供列出表或用于匹配表的正则表达式数组的 数据集合，例如：

```
{"data-collections": ["public.MyFirstTable", "public.MySecondTable"]}
```

增量快照信号的 data-collections 数组没有默认值。如果 data-collections 数组为空，Debezium 会检测到不需要任何操作，且不会执行快照。



注意

如果要包含在快照中的表的名称在数据库、模式或表的名称中包含句点(.)，以将表添加到 data-collections 数组中，您必须使用双引号转义名称的每个部分。

例如，要包含一个存在于 公共 模式的表，其名称为 My.Table，请使用以下格式：`"public"."My.Table "`*。*

先决条件

- *启用了信号。*
  - *源数据库中存在信号数据收集。*
  - *信号数据收集在 signal.data.collection 属性中指定。*

使用源信号频道来触发增量快照

1. *发送 SQL 查询，将临时增量快照请求添加到信号表中：*

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) (1)
values ('ad-hoc-1', (2)
  'execute-snapshot', (3)
  '{"data-collections": ["schema1.table1", "schema2.table2"], (4)
  "type":"incremental"}, (5)
  "additional-condition":"color=blue"}'); (6)
```

命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 8.3. SQL 命令中字段的描述，用于将增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。

项	值	描述
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。相反，Debebe 会在快照期间生成自己的 **id** 字符串作为水位线信号。
3	**execute-snapshot**	**type** 参数指定信号旨在触发的操作。
4	**data-collections**	信号的 **data** 字段所需的组件，用于指定表名称或正则表达式数组，以匹配快照中包含的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。
5	**incremental**	信号的 **data** 字段的可选 **类型** 组件，用于指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 如果没有指定值，连接器将运行增量快照。
6	**additional-condition**	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。有关 **additional-condition** 参数的更多信息，请参阅 **带有额外条件的临时增量快照**。

带有额外条件的临时增量快照

如果您希望快照只包含表中的内容子集，您可以通过向快照信号附加 additional-condition 参数来修改信号请求。

典型的快照的 SQL 查询采用以下格式：

```
SELECT * FROM <tableName> ....
```

通过添加 additional-condition 参数，您可以将 WHERE 条件附加到 SQL 查询中，如下例所示：

```
SELECT * FROM <tableName> WHERE <additional-condition> ....
```

以下示例显示了向信号表发送带有额外条件的临时增量快照请求的 SQL 查询：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，假设您有一个包含以下列的 products 表：

- ID （主键）
- color
- quantity

如果您需要 product 表的增量快照，其中只包含 color=blue 的数据项，您可以使用以下 SQL 语句来触发快照：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue"}');
```

additional-condition 参数还允许您传递基于多个列的条件。例如，使用上例中的 product 表，您可以提交查询来触发增量快照，该快照仅包含 color=blue 和 quantity>10 的项数据：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue AND quantity>10"}');
```

以下示例显示了连接器捕获的增量快照事件的 JSON。

示例：增加快照事件消息

```
{
  "before":null,
  "after": {
    "pk":"1",
    "value":"New data"
  },
  "source": {
    ...
    "snapshot":"incremental" 1
  },
  "op":"r", 2
```

```
    "ts_ms":"1620393591654",
    "transaction":null
}
```

项	字段名称	描述
1	**snapshot**	指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 在 SQL 查询中指定 **type** 值，您提交到信号表是可选的。 如果没有指定值，连接器将运行增量快照。
2	**op**	指定事件类型。 快照事件的值是 **r**，表示 **READ** 操作。

8.2.4.2. 使用 Kafka 信号频道来触发增量快照

您可以向 配置的 Kafka 主题 发送消息，以请求连接器来运行临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 execute-snapshot，data 字段必须具有以下字段：

表 8.4. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与快照中包含的表的完全限定域名匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的列子集的条件。

execute-snapshot Kafka 消息示例：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"],
"type": "INCREMENTAL"}}`
```

带有额外条件的临时增量快照

Debezium 使用 additional-condition 字段来选择表内容的子集。

通常，当 Debezium 运行快照时，它会运行 SQL 查询，例如：

SELECT * FROM <tableName> ....

当快照请求包含 additional-condition 时，extra-condition 会附加到 SQL 查询中，例如：

SELECT * FROM <tableName> WHERE <additional-condition> ....

例如，如果一个 product table with the column id （主键）、color 和 brand，如果您希望快照只包含 color='blue' 的内容，当您请求快照时，您可以附加一个 additional-condition 语句来过滤内容：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue'"}}`
```

您可以使用 additional-condition 语句根据多个列传递条件。例如，如果您希望快照只包含 color='blue' 的 products 表中，以及 brand='MyBrand'，则您可以发送以下请求：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}`
```

8.2.4.3. 停止增量快照

您还可以通过向源数据库上的表发送信号来停止增量快照。您可以通过发送 SQL INSERT 查询向表提

交停止快照信号。

在 Debezium 检测到信号表中的更改后，它会读取信号，并在正在进行时停止增量快照操作。

您提交的查询指定 增量 的快照操作，以及要删除的当前运行快照的表。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道停止增量快照

1. 发送 SQL 查询以停止临时增量快照到信号表：

```
INSERT INTO <signalTable> (id, type, data) values ('<id>', 'stop-snapshot', '{"data-collections": ["<tableName>","<tableName>"],"type":"incremental"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) (1)
values ('ad-hoc-1', (2)
  'stop-snapshot', (3)
  '{"data-collections": ["schema1.table1", "schema2.table2"], (4)
  "type":"incremental"}'); (5)
```

signal 命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 8.5. SQL 命令中字段的描述，用于将停止增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。 使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。
3	**stop-snapshot**	指定 **type** 参数指定信号要触发的操作。
4	**data-collections**	信号的 **data** 字段的可选组件，用于指定表名称或正则表达式数组，以匹配要从快照中删除的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。如果省略了 **data** 字段的这一组件，信号将停止正在进行的整个增量快照。
5	**incremental**	信号的 **data** 字段所需的组件，用于指定要停止的快照操作类型。 目前，唯一有效的选项是 **增量的**。 如果没有指定 **类型** 值，信号将无法停止增量快照。

8.2.4.4. 使用 Kafka 信号频道停止增量快照

您可以将信号消息发送到 配置的 Kafka 信号主题，以停止临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 stop-snapshot，data 字段必须具有以下字段：

表 8.6. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	可选数组，以逗号分隔的正则表达式，与表的完全限定域名匹配，以包含在快照中。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

以下示例显示了典型的 stop-snapshot Kafka 信息：

```
Key = `test_connector`

Value = `{"type":"stop-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"],
"type": "INCREMENTAL"}}`
```

8.2.5. Debezium PostgreSQL 连接器流更改事件记录

PostgreSQL 连接器通常将其大部分的时间流更改从 PostgreSQL 服务器连接到其中。这种机制依赖于 PostgreSQL 的复制协议。这个协议可让客户端从服务器接收更改，因为它们在服务器的事务日志中提交，这些位置被称为 Log Sequence Numbers (LSN)。

每当服务器提交事务时，单独的服务器进程会从 逻辑解码插件 调用回调功能。此功能处理事务中的更改，将其转换为特定格式（如果是 Debezium 插件则为Protobuf 或 JSON），并在输出流上写入它们，然后可以被客户端使用。

Debezium PostgreSQL 连接器充当 PostgreSQL 客户端。当连接器收到更改时，它会将事件转换为 Debezium 的 create, update, 或 delete 事件，包含该事件的 LSN 的事件。PostgreSQL 连接器将记录中的这些更改事件转发到 Kafka Connect 框架，该框架在同一进程中运行。Kafka Connect 进程异步写入更改事件记录，其顺序与生成到适当的 Kafka 主题相同。

定期，Kafka Connect 在另一个 Kafka 主题中记录最新的 偏移量。偏移表示 Debezium 包含在每个事件中的特定于源的位置信息。对于 PostgreSQL 连接器，在每次更改事件中记录的 LSN 是偏移量。

当 Kafka Connect 正常关闭时，它会停止连接器，将所有事件记录刷新到 Kafka，并记录从每个连接器接收的最后偏移。当 Kafka Connect 重启时，它会为每个连接器读取最后记录的偏移，并在其最后记录的偏移上启动每个连接器。当连接器重启时，它会将请求发送到 PostgreSQL 服务器，来仅在该位置后开始发送事件。

注意

PostgreSQL 连接器检索模式信息，作为逻辑解码插件发送的事件的一部分。但是，连接器不会检索有关构成主键的列的信息。连接器从 JDBC 元数据（在频道侧）获取此信息。如果表的主键定义有变化（通过添加、删除或重命名主键列），当来自 JDBC 的主密钥信息与逻辑解码插件生成的更改事件没有同步时，会有一个小时段。在此小期间内，可以创建带有不一致的密钥结构的消息。要防止这种不一致，请按如下所示更新主键结构：

1. 将数据库或应用程序置于只读模式。
2. 让 Debezium 处理所有剩余的事件。
3. 停止 Debezium。
4. 更新相关表中的主密钥定义。
5. 将数据库或应用程序置于读/写模式。
6. 重启 Debezium。

PostgreSQL 10+ 逻辑解码支持(pgoutput)

从 PostgreSQL 10+ 开始，有一个逻辑复制流模式，称为 pgoutput，它被 PostgreSQL 原生支持。这意味着 Debezium PostgreSQL 连接器可以消耗该复制流，而无需额外的插件。这对不支持或不支持或不支持插件的环境特别有用。

如需更多信息，请参阅设置 PostgreSQL。

8.2.6. 接收 Debezium PostgreSQL 更改事件记录的默认 Kafka 主题名称

默认情况下，PostgreSQL 连接器将所有 INSERT、UPDATE 和 DELETE 操作的更改事件写入特定于该表的单一 Apache Kafka 主题。连接器使用以下惯例来命名更改事件主题：

topicPrefix.schemaName.tableName

以下列表为默认名称的组件提供定义：

topicPrefix

由 topic.prefix 配置属性指定的主题前缀。

schemaName

发生更改事件的数据库模式的名称。

tableName

发生更改事件的数据库表的名称。

例如，假设 fulfillment 是连接器中的逻辑服务器名称，该连接器捕获 PostgreSQL 安装中的更改，该安装具有一个 postgres 数据库和一个 inventory schem，它包含四个表：products, products_on_hand, customers, 和 orders连接器会将记录流传输到这四个 Kafka 主题：

- fulfillment.inventory.products
- fulfillment.inventory.products_on_hand
- fulfillment.inventory.customers
- fulfillment.inventory.orders

现在假设表不是特定架构的一部分，但在默认的 公共 PostgreSQL 模式中创建。Kafka 主题的名称为：

- fulfillment.public.products
- fulfillment.public.products_on_hand
- fulfillment.public.customers

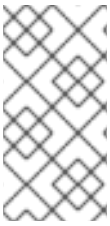
- *fulfillment.public.orders*

连接器应用类似的命名约定，以标记其 事务元数据主题。

如果默认主题名称不满足您的要求，您可以配置自定义主题名称。要配置自定义主题名称，您可以在逻辑主题路由 SMT 中指定正则表达式。有关使用逻辑主题路由 SMT 来自定义主题命名的更多信息，请参阅 主题路由。

8.2.7. Debezium PostgreSQL 连接器生成的事件代表事务边界

Debezium 可以生成代表事务边界的事件，以及丰富的数据更改事件消息。



DEBEZIUM 接收事务元数据时的限制

Debezium 注册并只针对部署连接器后发生的事务接收元数据。部署连接器前发生的事务元数据不可用。

对于每个事务 BEGIN 和 END，Debezium 会生成一个包含以下字段的事件：

status

BEGIN 或 END.

id

由 Postgres 事务 ID 本身和给定操作的 LSN 组成的唯一事务标识符的字符串，即格式为 txID:LSN。

ts_ms

数据源的事务边界事件(BEGIN 或 END 事件)的时间。如果数据源没有向事件时间提供 Debezium，则该字段代表 Debezium 处理事件的时间。

event_count （用于 END 事件）

事务提供的事件总数。

data_collections （用于 END 事件）

data_collection 和 event_count 元素的数组，用于指示连接器发出来自数据收集的更改的事件

数量。

示例

```
{
  "status": "BEGIN",
  "id": "571:53195829",
  "ts_ms": 1486500577125,
  "event_count": null,
  "data_collections": null
}

{
  "status": "END",
  "id": "571:53195832",
  "ts_ms": 1486500577691,
  "event_count": 2,
  "data_collections": [
    {
      "data_collection": "s1.a",
      "event_count": 1
    },
    {
      "data_collection": "s2.a",
      "event_count": 1
    }
  ]
}
```

除非通过 topic.transaction 选项覆盖，否则事务事件将写入名为 <topic. prefix>.transaction 的主题。

更改数据事件增强

启用事务元数据后，数据消息 Envelope 通过新的 transaction 字段进行了增强。此字段以字段复合的形式提供有关每个事件的信息：

id

唯一事务标识符的字符串。

total_order

事件在事务生成的所有事件中绝对位置。

data_collection_order

在事务发出的所有事件间，按数据收集位置。

以下是消息的示例：

```
{
 "before": null,
 "after": {
  "pk": "2",
  "aa": "1"
 },
 "source": {
  ...
 },
 "op": "c",
 "ts_ms": "1580390884335",
 "transaction": {
  "id": "571:53195832",
  "total_order": "1",
  "data_collection_order": "1"
 }
}
```

8.3. DEBEZIUM POSTGRESQL 连接器数据更改事件的描述

Debezium PostgreSQL 连接器为每个行级 INSERT、UPDATE 和 DELETE 操作生成数据更改事件。每个事件包含一个键和值。键的结构和值取决于已更改的表。

Debezium 和 Kafka Connect 围绕 事件消息的持续流 设计。但是，这些事件的结构可能会随时间推移而改变，而用户很难处理这些事件。要解决这个问题，每个事件都包含其内容的 schema，或者如果您正在使用 schema registry，用户可以使用该模式 ID 从 registry 获取 schema。这使得每个事件都自包含。

以下框架 JSON 显示更改事件的基本四部分。但是，如何配置您选择在应用程序中使用的 Kafka Connect converter，决定更改事件中的这四个部分的表示。只有在将转换器配置为生成它时，schema 字段才会处于更改事件中。同样，只有在您配置转换器来生成它时，事件密钥和事件有效负载才会处于更改事件中。如果您使用 JSON 转换程序，并将其配置为生成所有四个基本更改事件部分，更改事件具有此结构：

```
{
 "schema": { 1
```

```
  ...
 },
 "payload": { 2
  ...
},
 "schema": { 3
  ...
},
 "payload": { 4
  ...
},
}
```

表 8.7. 更改事件基本内容概述

项	字段名称	描述
1	**schema**	第一个 **schema** 字段是事件键的一部分。它指定一个 Kafka Connect 模式，用于描述事件键的 **payload** 部分的内容。换句话说，第一个 **schema** 字段描述了主密钥的结构，如果表没有主键，则描述主键的结构。 可以通过设置 **message.key.columns** 连接器配置属性 来覆盖表的主键。在这种情况下，第一个 schema 字段描述了该属性标识的键的结构。
2	**payload**	第一个 **payload** 字段是 event 键的一部分。它具有前面的**schema** 字段描述的结构，它包含已更改的行的密钥。
3	**schema**	第二个 **schema** 字段是事件值的一部分。它指定 Kafka Connect 模式，用于描述事件值 **有效负载部分的内容**。换句话说，第二个 **模式** 描述了已更改的行的结构。通常，此模式包含嵌套模式。
4	**payload**	第二个 **payload** 字段是事件值的一部分。它具有上一个 **schema** 字段描述的结构，它包含已更改的行的实际数据。

默认的行为是，连接器流将事件记录更改为 名称与事件原始表相同的主题。

注意

从 Kafka 0.10 开始，Kafka 可以选择使用创建消息 的时间戳 （由 producer 记录）或 Kafka 写入日志的时间戳记录事件键和值。

警告

PostgreSQL 连接器确保所有 Kafka Connect 模式名称都遵循 Avro 模式名称格式。这意味着逻辑服务器名称必须以拉丁字母或下划线开头，即 a-z、A-Z 或 _。逻辑服务器名称和 schema 和表名称中的每个字符都必须是一个拉丁字母、数字或下划线，即 a-z、A-Z、0-9 或 _。如果存在无效字符，它将使用下划线字符替换。

如果逻辑服务器名称、模式名称或表名称包含无效字符，且唯一与另一个名称区分名称的字符无效，这可能会导致意外冲突冲突，从而被下划线替换。

详情包括在以下主题中：

- 第 8.3.1 节 “关于 Debezium PostgreSQL 中的键更改事件”
- 第 8.3.2 节 “关于 Debezium PostgreSQL 更改事件中的值”

8.3.1. 关于 Debezium PostgreSQL 中的键更改事件

对于给定表，更改事件的键具有结构，该结构在表的主键中包含创建事件时每个列的字段。或者，如果表将 REPLICA IDENTITY 设置为 FULL 或 USING INDEX，则每个唯一键约束都有一个字段。

考虑在 公共 数据库架构中定义的 customers 表以及该表的更改事件密钥示例。

表示例

```
CREATE TABLE customers (
  id SERIAL,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY(id)
);
```

更改事件键示例

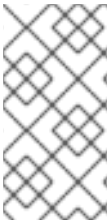
如果 topic.prefix 连接器配置属性的值为 PostgreSQL_server，则 customer 表的每个更改事件都有相同的键结构，JSON 类似如下：

```
{
 "schema": { 1
  "type": "struct",
  "name": "PostgreSQL_server.public.customers.Key", 2
  "optional": false, 3
  "fields": [ 4
      {
          "name": "id",
          "index": "0",
          "schema": {
            "type": "INT32",
            "optional": "false"
          }
      }
    ]
 },
 "payload": { 5
    "id": "1"
 },
}
```

表 8.8. 更改事件键的描述

项	字段名称	描述
1	**schema**	键的 schema 部分指定一个 Kafka Connect 模式，它描述了键的 **payload** 部分的内容。
2	**PostgreSQL_server.inventory.customers.Key**	定义密钥有效负载结构的模式名称。这个 schema 描述了已更改的表的主键的结构。键模式名称的格式是 *connector-name.database-name.table-name*.**Key**.在本例中： • **PostgreSQL_server** 是生成此事件的连接器的名称。 • **inventory** 是包含已更改表的数据库。 • **客户** 是更新的表。
3	**optional**	指明 event 键是否必须在其 **payload** 字段中包含一个值。在本例中，键有效负载中的值是必需的。当表没有主键时，键的 payload 字段中的值是可选的。

项	字段名称	描述
4	**fields**	指定 **有效负载中** 预期的每个字段，包括每个字段的名称、索引和模式。
5	**payload**	包含生成此更改事件的行的密钥。在本例中，键 包含一个 **id** 字段，其值为 **1**。



注意

虽然 column.exclude.list 和 column.include.list 连接器配置属性允许您只捕获表列的子集，但主键或唯一键中的所有列始终包含在事件键中。

警告

如果表没有主密钥或唯一密钥，则更改事件的密钥为 null。没有主或唯一键约束的表中的行无法唯一标识。

8.3.2. 关于 Debezium PostgreSQL 更改事件中的值

更改事件中的值比键复杂一些。与键一样，该值有一个 schema 部分和 payload 部分。schema 部分包含描述 payload 部分的 Envelope 结构的 schema，包括其嵌套字段。为创建、更新或删除数据的操作更改事件，它们都有一个带有 envelope 结构的值有效负载。

考虑用于显示更改事件键示例的相同示例表：

```
CREATE TABLE customers (
  id SERIAL,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY(id)
);
```

更改此表的更改事件的值部分因 REPLICA IDENTITY 设置和事件所针对的操作而异。

这些部分中的详情如下：

- *副本身份*
- *创建 事件*
- *更新 事件*
- *主密钥更新*
- *删除 事件*
- *tombstone 事件*

副本身份

REPLICA IDENTITY 是一个特定于 PostgreSQL 的表级设置，它决定了 UPDATE 和 DELETE 事件的逻辑解码插件可用的信息量。更具体地说，当发生 UPDATE 或 DELETE 事件时，会控制 REPLICA IDENTITY 的设置可用于表列的前面值。

REPLICA IDENTITY 有 4 个可能的值：

- *DEFAULT - 如果该表有主键，则默认行为是 UPDATE 和 DELETE 事件包含表的主键列的前面值。对于 UPDATE 事件，只有带有更改值的主键列才会存在。*

  如果表没有主密钥，连接器不会为该表发出 UPDATE 或 DELETE 事件。对于没有主密钥的表，连接器只发出 创建事件。通常，没有主键的表用于将消息附加到表的末尾，这意味着 UPDATE 和 DELETE 事件不有用。

- *NOTHING - 为 UPDATE 和 DELETE 操作传输的事件不包含任何表列的之前值的信息。*
- *FULL - Emitted 事件用于 UPDATE 和 DELETE 操作，包含表中所有列的前面值。*
- *INDEX index-name - Emitted 事件用于 UPDATE 和 DELETE 操作，包含指定索引中包含的列的先前值。UPDATE 事件也包含带有更新值的索引列。*

创建 事件

以下示例显示了一个更改事件的值部分，连接器为在 customer 表中创建数据的操作生成的更改事件的值部分：

```
{
  "schema": { 1
    "type": "struct",
    "fields": [
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "first_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "last_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "email"
          }
        ],
        "optional": true,
        "name": "PostgreSQL_server.inventory.customers.Value", 2
        "field": "before"
      },
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "first_name"
          },
          {
            "type": "string",
```

```
            "optional": false,
            "field": "last_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "email"
          }
        ],
        "optional": true,
        "name": "PostgreSQL_server.inventory.customers.Value",
        "field": "after"
      },
      {
        "type": "struct",
        "fields": [
          {
            "type": "string",
            "optional": false,
            "field": "version"
          },
          {
            "type": "string",
            "optional": false,
            "field": "connector"
          },
          {
            "type": "string",
            "optional": false,
            "field": "name"
          },
          {
            "type": "int64",
            "optional": false,
            "field": "ts_ms"
          },
          {
            "type": "boolean",
            "optional": true,
            "default": false,
            "field": "snapshot"
          },
          {
            "type": "string",
            "optional": false,
            "field": "db"
          },
          {
            "type": "string",
            "optional": false,
            "field": "schema"
          },
          {
            "type": "string",
            "optional": false,
            "field": "table"
```

```
            },
            {
                "type": "int64",
                "optional": true,
                "field": "txId"
            },
            {
                "type": "int64",
                "optional": true,
                "field": "lsn"
            },
            {
                "type": "int64",
                "optional": true,
                "field": "xmin"
            }
          ],
          "optional": false,
          "name": "io.debezium.connector.postgresql.Source", 3
          "field": "source"
        },
        {
          "type": "string",
          "optional": false,
          "field": "op"
        },
        {
          "type": "int64",
          "optional": true,
          "field": "ts_ms"
        }
      ],
      "optional": false,
      "name": "PostgreSQL_server.inventory.customers.Envelope" 4
    },
    "payload": { 5
      "before": null, 6
      "after": { 7
        "id": 1,
        "first_name": "Anne",
        "last_name": "Kretchmar",
        "email": "annek@noanswer.org"
      },
      "source": { 8
        "version": "2.3.4.Final",
        "connector": "postgresql",
        "name": "PostgreSQL_server",
        "ts_ms": 1559033904863,
        "snapshot": true,
        "db": "postgres",
        "sequence": "[\"24023119\",\"24023128\"]",
        "schema": "public",
        "table": "customers",
        "txId": 555,
        "lsn": 24023128,
```

```
        "xmin": null
      },
      "op": "c", 9
      "ts_ms": 1559033904863 10
    }
  }
```

表 8.9. 创建 事件值字段的描述

项	字段名称	描述
1	**schema**	值的 schema，用于描述值有效负载的结构。当连接器为特定表生成的每次更改事件中，更改事件的值模式都是相同的。
2	**name**	在 **schema** 部分中，每个 **name** 字段为值有效负载中的字段指定 schema。 **PostgreSQL_server.inventory.customers.Value** 是有效负载 **before** 和 **after** 字段的 schema。这个模式特定于 **customers** 表。 **before** 和 **after** 字段的模式名称格式为 ***logicalName.tableName*.Value**，这样可确保 schema 名称在数据库中是唯一的。这意味着，在使用 Avro converter 时，每个逻辑源中的每个表生成的 Avro 模式都有自己的 evolution 和 history。
3	**name**	**io.debezium.connector.postgresql.Source** 是有效负载的 **source** 字段的 schema。这个模式特定于 PostgreSQL 连接器。连接器将其用于它生成的所有事件。
4	**name**	**PostgreSQL_server.inventory.customers.Envelope** 是有效负载的整体结构的 schema，其中 **PostgreSQL_server** 是连接器名称，**inventory** 是数据库，**customers** 是表。
5	**payload**	值的实际数据。这是更改事件提供的信息。 可能会出现事件的 JSON 表示比它们描述的行大得多。这是因为 JSON 表示必须包含消息的 schema 和 payload 部分。但是，通过使用 Avro converter，您可以显著减少连接器流到 Kafka 主题的信息大小。
6	**before**	指定事件发生前行状态的可选字段。当 **op** 字段是 **c** 用于创建（如本例所示），**before** 字段为 **null**，因为此更改事件用于新内容。  **注意** 此字段是否可用取决于每个表的 **REPLICA IDENTITY** 设置。
7	**after**	指定事件发生后行状态的可选字段。在本例中，**after** 字段包含新行的 **id**、**first_name**、**last_name** 和 **email** 列的值。

项	字段名称	描述
8	**source**	描述事件源元数据的必需字段。此字段包含可用于将此事件与其他事件进行比较的信息，以及事件的来源、事件发生的顺序以及事件是否为同一事务的一部分。源元数据包括： ● Debezium 版本 ● 连接器类型和名称 ● 包含新行的数据库和表 ● 专用 JSON 数组的额外偏移信息。第一个值是最后一个提交的 LSN，第二个值是当前的 LSN。两个值可以是 **null**。 ● 模式名称 ● 如果事件是快照的一部分 ● 执行操作的事务的 ID ● 数据库日志中操作的偏移量 ● 在数据库中进行更改时的时间戳
9	**op**	描述导致连接器生成事件的操作类型的强制字符串。在本例中，**c** 表示操作创建了行。有效值为： ● **c** = create ● **u** = update ● **d** = delete ● **r** = 读取（仅适用于快照） ● **t** = truncate ● **m** = message
10	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

更新 事件

示例 customers 表中一个更新的改变事件的值有与那个表的 create 事件相同的模式。同样，事件值有效负载具有相同的结构。但是，事件值有效负载在 update 事件中包含不同的值。以下是连接器为 customer 表中更新生成的更改事件值 的示例 ：

```
{
    "schema": { ... },
    "payload": {
        "before": { 1
            "id": 1
        },
        "after": { 2
            "id": 1,
            "first_name": "Anne Marie",
            "last_name": "Kretchmar",
            "email": "annek@noanswer.org"
        },
        "source": { 3
            "version": "2.3.4.Final",
            "connector": "postgresql",
            "name": "PostgreSQL_server",
            "ts_ms": 1559033904863,
            "snapshot": false,
            "db": "postgres",
            "schema": "public",
            "table": "customers",
            "txId": 556,
            "lsn": 24023128,
            "xmin": null
        },
        "op": "u", 4
        "ts_ms": 1465584025523 5
    }
}
```

表 8.10. 更新 事件值字段的描述

项	字段名称	描述
1	**before**	包含数据库提交前行中的值的可选字段。在此例中只有一个主列 **id**，因为表的 **REPLICA IDENTITY** 设置默认为 **DEFAULT**。 + 对于一个 *update* 事件来包括行中所有列的以前的值，您需要修改 **customers** 表（通过运行 **ALTER TABLE customers REPLICA IDENTITY FULL**）。
2	**after**	指定事件发生后行状态的可选字段。在本例中，**first_name** 值现在是 **Anne Marie**。

项	字段名称	描述
3	**source**	描述事件源元数据的必需字段。**source** 字段结构与 *create* 事件中的字段相同，但一些值不同。源元数据包括： • Debezium 版本 • 连接器类型和名称 • 包含新行的数据库和表 • 模式名称 • 如果事件是快照的一部分（对于 *更新* 事件，始终为 **false** ） • 执行操作的事务的 ID • 数据库日志中操作的偏移量 • 在数据库中进行更改时的时间戳
4	**op**	描述操作类型的强制字符串。在 *update* 事件值中，**op** 字段值为 **u**，表示此行因为更新而改变。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

注意

更新行 primary/unique 键的列会更改行的键值。当键更改时，Debezium 会输出 三个事件：一个 DELETE 事件，以及一个带有行的旧键的 tombstone 事件，后跟一个带有行的新键的事件。详情在下一节中。

主密钥更新

更改行的主键字段的 UPDATE 操作称为主密钥更改。对于主键更改，以代替发送 UPDATE 事件记录，连接器会为旧密钥发送 DELETE 事件记录，并为新的(updated)密钥创建一个 CREATE 事件记录。这些事件具有常见的结构和内容，另外，每个事件都有与主密钥更改相关的消息标头：

- ***DELETE 事件记录具有 __debezium.newkey 作为消息标头。此标头的值是更新行的新主键。***
- ***CREATE 事件记录具有 __debezium.oldkey 作为消息标头。此标头的值是更新行所具有的***

前一个主键（旧的）主键。

删除 事件

delete 更改事件中的值与为同一表的 create 和 update 事件相同的 schema 部分。示例 customer 表的 delete 事件中 payload 部分类似如下：

```
{
  "schema": { ... },
  "payload": {
    "before": { 1
      "id": 1
    },
    "after": null, 2
    "source": { 3
      "version": "2.3.4.Final",
      "connector": "postgresql",
      "name": "PostgreSQL_server",
      "ts_ms": 1559033904863,
      "snapshot": false,
      "db": "postgres",
      "schema": "public",
      "table": "customers",
      "txId": 556,
      "lsn": 46523128,
      "xmin": null
    },
    "op": "d", 4
    "ts_ms": 1465581902461 5
  }
}
```

表 8.11. 删除 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在 *delete* 事件值中，**before** 字段包含在使用数据库提交删除行前的值。 在本例中，**before** 字段仅包含主键列，因为表的 **REPLICA IDENTITY** 设置为 **DEFAULT**。
2	**after**	指定事件发生后行状态的可选字段。在 *delete* 事件值中，**after** 字段为 **null**，表示行不再存在。

项	字段名称	描述
3	**source**	描述事件源元数据的必需字段。在一个 *delete* 事件值中，**source** 字段结构与同一表的 *create* 和 *update* 事件相同。许多 **source** 字段值也相同。在 *delete* 事件值中，**ts_ms** 和 **lsn** 字段值以及其他值可能已更改。但是 *delete* 事件值中的 **source** 字段提供相同的元数据： • Debezium 版本 • 连接器类型和名称 • 包含已删除行的数据库和表 • 模式名称 • 如果事件是快照的一部分（对于 *删除* 事件，始终为 **false** ） • 执行操作的事务的 ID • 数据库日志中操作的偏移量 • 在数据库中进行更改时的时间戳
4	**op**	描述操作类型的强制字符串。**op** 字段值为 **d**，表示此行已被删除。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

删除 更改事件记录为消费者提供处理此行删除所需的信息。

警告

要使消费者处理为没有主键的表生成的 删除 事件，请将表的 REPLICA IDENTITY 设置为 FULL。当表没有主键且表的 REPLICA IDENTITY 设置为 DEFAULT 或 NOTHING 时，删除 事件在字段 之前没有。

PostgreSQL 连接器事件旨在使用 Kafka 日志压缩。只要保留每个密钥的最新消息，日志压缩就会启用删除一些旧的消息。这可让 Kafka 回收存储空间，同时确保主题包含完整的数据集，并可用于重新载入基于密钥的状态。

tombstone 事件

删除行时，delete 事件值仍可用于日志压缩，因为 Kafka 您可以删除具有相同键的所有之前信息。但是，要让 Kafka 删除具有相同键的所有消息，消息值必须为 null。为了实现此目的，PostgreSQL 连接器遵循一个 delete 事件，其中包含一个特殊的tombstone 事件，它有相同的键但值为 null。

Truncate 事件

截断 更改事件信号，表示表已被截断。在这种情况下，message 键为 null，message 值类似如下：

```
{
  "schema": { ... },
  "payload": {
    "source": { 1
      "version": "2.3.4.Final",
      "connector": "postgresql",
      "name": "PostgreSQL_server",
      "ts_ms": 1559033904863,
      "snapshot": false,
      "db": "postgres",
      "schema": "public",
      "table": "customers",
      "txId": 556,
      "lsn": 46523128,
      "xmin": null
    },
    "op": "t", 2
    "ts_ms": 1559033904961 3
  }
}
```

表 8.12. truncate 事件值字段的描述

项	字段名称	描述

项	字段名称	描述
1	**source**	描述事件源元数据的必需字段。在 *truncate* 事件值中，**source** 字段结构与为同一表的 *create*, *update*, 和 *delete* 事件相同，提供此元数据： ● Debezium 版本 ● 连接器类型和名称 ● 包含新行的数据库和表 ● 模式名称 ● 如果事件是快照的一部分（对于 *删除* 事件，始终为 **false** ） ● 执行操作的事务的 ID ● 数据库日志中操作的偏移量 ● 在数据库中进行更改时的时间戳
2	**op**	描述操作类型的强制字符串。**op** 字段值为 **t**，表示此表已被截断。
3	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

如果单个 TRUNCATE 语句应用到多个表，则会为每个删节的表发出一个 truncate 更改事件记录。

请注意，由于 truncate 事件代表对整个表进行的更改，且没有消息密钥，除非您使用单个分区的主题，否则与表相关的更改事件没有排序保证(创建、更新 等)和 截断 该表的事件。例如，当这些事件从不同的分区读取时，消费者只能在该表的 truncate 事件后收到 update 事件。

消息 事件

只有 Postgres 14+ 上的 pgoutput 插件支持此事件类型(Postgres 文档)

消息 事件信号，一个通用逻辑解码消息已被直接插入到 WAL 中，通常带有 pg_logical_emit_message 函数。message 键是一个 Struct，其中包含一个名为 prefix 的单个字段，它会执行插入消息时指定的前缀。message 值与事务性信息类似：

```
{
  "schema": { ... },
```

```
  "payload": {
    "source": { 1
      "version": "2.3.4.Final",
      "connector": "postgresql",
      "name": "PostgreSQL_server",
      "ts_ms": 1559033904863,
      "snapshot": false,
      "db": "postgres",
      "schema": "",
      "table": "",
      "txId": 556,
      "lsn": 46523128,
      "xmin": null
    },
    "op": "m", 2
    "ts_ms": 1559033904961, 3
    "message": { 4
      "prefix": "foo",
      "content": "Ymfy"
    }
  }
}
```

与其他事件类型不同，非事务性消息将没有任何关联的 BEGIN 或 END 事务事件。对于非事务信息，message 值类似如下：

```
{
  "schema": { ... },
  "payload": {
    "source": { 1
      "version": "2.3.4.Final",
      "connector": "postgresql",
      "name": "PostgreSQL_server",
      "ts_ms": 1559033904863,
      "snapshot": false,
      "db": "postgres",
      "schema": "",
      "table": "",
      "lsn": 46523128,
      "xmin": null
    },
    "op": "m", 2
    "ts_ms": 1559033904961 3
    "message": { 4
      "prefix": "foo",
      "content": "Ymfy"
  }
}
```

表 8.13. 消息 事件值字段的描述

项	字段名称	描述
1	**source**	描述事件源元数据的必需字段。在 *message* 事件值中，**source** 字段结构将不会包括任何 *message* 事件的 **table** 或 **schema** 信息，并只有在 *message* 事件是事务性时才有 **txId**。 ● Debezium 版本 ● 连接器类型和名称 ● 数据库名称 ● schema name (always **""** for *message* events) ● 表名称（始终为 **""** *消息* 事件） ● 如果事件是快照的一部分（对于 *消息* 事件，始终为 **false** ） ● 执行操作的事务的 ID （非事务 *message* 事件为 **null**） ● 数据库日志中操作的偏移量 ● 事务消息：当消息插入到 WAL 中时的时间戳 ● 非交易消息；当连接器遇到消息时的 Timestamp
2	**op**	描述操作类型的强制字符串。**op** 字段值为 **m**，表示这是 *消息* 事件。
3	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 对于事务 *消息* 事件，源对象的 **ts_ms** 属性指示数据库中更改的时间。 ***通过将 payload.source.ts_ms 的值与 payload.ts_ms 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。*** ***对于非事务 消息 事件，源对象的 ts_ms 表示连接器遇到 消息事件的时间，而 payload.ts_ms 表示连接器处理事件的时间。这是因为，提交时间戳在 Postgres 的通用逻辑消息格式中不存在，且非事务逻辑消息没有以 BEGIN 事件（具有时间戳信息）的前面。***
4	**message**	包含消息元数据的字段 ● 前缀（文本） ● 内容（基于 二进制处理模式 设置编码的字节阵列）

8.4. DEBEZIUM POSTGRESQL 连接器如何映射数据类型

PostgreSQL 连接器代表对带有结构的事件的更改，这些事件与行存在的表类似。事件包含每个列值的一个字段。在事件中如何代表该值取决于列的 PostgreSQL 数据类型。以下小节描述了连接器如何将 PostgreSQL 数据类型映射到 字面类型以及 事件字段中 的语义类型。

- literal type 代表值如何被代表，使用 Kafka Connect schema 类型：INT8, INT16, INT32, INT64, FLOAT32, FLOAT64, BOOLEAN, STRING, BYTES, ARRAY, MAP, 和 STRUCT。
- 语义类型 描述了 Kafka Connect 模式如何使用字段名称来捕获字段 的含义。

如果默认数据类型转换不满足您的需要，您可以为连接器 创建自定义转换器。

以下部分详情：

- 基本类型
- 时序类型
- TIMESTAMP 类型
- 十进制类型
- HSTORE 类型
- 域类型
- 网络地址类型
- PostGIS 类型

- *要粘贴的值*

基本类型

下表描述了连接器如何映射基本类型。

表 8.14. PostgreSQL 基本数据类型的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
布尔值	布尔值	不适用
BIT(1)	布尔值	不适用
BIT(> 1)	BYTES	io.debezium.data.Bits 。length 模式参数包含一个代表位数的整数。生成的 byte[] 包含 little-endian 格式的位，并的大小包含指定的位数。例如，numBytes = n/8 +（n % 8 == 0 ?0 : 1） 其中 n 是位数。
位不同[(M)]	BYTES	io.debezium.data.Bits 。length 模式参数包含一个整数，代表位数(2^31 - 1，如果没有为列提供长度)。生成的 byte[] 包含 little-endian 表单中的位，并根据内容的大小。指定的大小 (M) 存储在 io.debezium.data.Bits 类型的 length 参数中。
SMALLINT,SMALLSERIAL	INT16	不适用
整数， 串行	INT32	不适用
BIGINT,BIGSERIAL OID	INT64	不适用
REAL	FLOAT32	不适用
双精度	FLOAT64	不适用
CHAR[(M)]	字符串	不适用
VARCHAR[(M)]	字符串	不适用
CHARACTER[(M)]	字符串	不适用

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
CHARACTER VARYING[(M)]	**字符串**	不适用
TIMESTAMPTZ，带有时区的时间戳	**字符串**	**io.debezium.time.ZonedTimestamp** 是带有时区信息的时间戳的字符串，其中时区为 GMT。
TIMETZ，带有时区的时间	**字符串**	**io.debezium.time.ZonedTime** 是时区信息的字符串，其中时区为 GMT。
间隔 [P]	**INT64**	**io.debezium.time.MicroDuration** （默认） 使用每月平均值的 **365.25 / 12.0** 公式的间隔的大约微秒数。
间隔 [P]	**字符串**	**io.debezium.time.Interval** （当 **interval.handling.mode** 设置为 **字符串**） 遵循特征 **P<years>Y<months>Y<months>M<days>DT<hours>H<minutes>M<seconds>S** 的间隔值的字符串表示，例如 **P1Y2M3DT4H5M6.78S**。
BYTEA	**BYTES** 或 **STRING**	N/a 遵循原始字节（默认）、base64 编码的字符串或 base64-url-safe-encoded 字符串或十六进制编码的字符串，具体取决于连接器 的二进制处理模式 设置。 Debezium 只支持 Postgres **bytea_output** 配置值 **hex**。有关 PostgreSQL 二进制文件数据类型的更多信息，请参阅 PostgreSQL 文档。
JSON, JSONB	**字符串**	**io.debezium.data.Json** 包含 JSON 文档、数组或 scalar 的字符串表示。
XML	**字符串**	**io.debezium.data.Xml** 包含 XML 文档的字符串表示。
UUID	**字符串**	**io.debezium.data.Uuid** 包含 PostgreSQL UUID 值的字符串表示。

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
点	**STRUCT**	**io.debezium.data.geometry.Point** 包含有两个 **FLOAT64** 字段的结构，**(x,y)**。每个字段代表 geometric 点的协调。
LTREE	**字符串**	**io.debezium.data.Ltree** 包含 PostgreSQL LTREE 值的字符串表示。
CITEXT	**字符串**	不适用
INET	**字符串**	不适用
INT4RANGE	**字符串**	N/a 整数范围。
INT8RANGE	**字符串**	N/A range of **bigint**.
NUMRANGE	**字符串**	N/A 范围 **数字**.
TSRANGE	**字符串**	n/a 包含时间戳范围的字符串表示，没有时区。
TSTZRANGE	**字符串**	n/a 包含带有本地系统时区的时间戳范围的字符串表示。
DATERANGE	**字符串**	n/a 包括代表一个日期范围的字符串。它始终具有专用的上限。
ENUM	**字符串**	**io.debezium.data.Enum** 包含 PostgreSQL **ENUM** 值的字符串表示。允许的值集合在 **允许的** schema 参数中维护。

时序类型

除 PostgreSQL 的 TIMESTAMPTZ 和 TIMETZ 数据类型外，包含时区信息，临时类型是如何映射的，具体类型取决于 time.precision.mode 连接器配置属性的值。以下小节描述了这些映射：

- *time.precision.mode=adaptive*
- *time.precision.mode=adaptive_time_microseconds*
- *time.precision.mode=connect*

time.precision.mode=adaptive

当将 time.precision.mode 属性设置为 adaptive 时，连接器会根据列的数据类型定义确定字面类型和语义类型。这样可确保事件 完全 代表数据库中的值。

表 8.15. time.precision.mode 为 adaptive时的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**io.debezium.time.Date** 代表自时期起的天数。
TIME (1)、TIME (2), TIME (3)	**INT32**	**io.debezium.time.Time** 代表过去的毫秒数，不包括时区信息。
TIME (4), TIME (5), TIME (6)	**INT64**	**io.debezium.time.MicroTime** 代表过去的微秒数，不包括时区信息。
TIMESTAMP (1), TIMESTAMP (2), TIMESTAMP (3)	**INT64**	**io.debezium.time.Timestamp** 代表自 epoch 后的毫秒数，不包括时区信息。
TIMESTAMP (4), TIMESTAMP (5), TIMESTAMP (6), TIMESTAMP	**INT64**	**io.debezium.time.MicroTimestamp** 代表自 epoch 起的微秒数，不包括时区信息。

time.precision.mode=adaptive_time_microseconds

当将 time.precision.mode 配置属性设置为 adaptive_time_microseconds 时，连接器会根据列的数据类型确定 temporal 类型的字面类型和 semantic 类型。这样可确保事件 准确 代表数据库中的值，但所

有 TIME 字段都捕获为微秒。

表 8.16. 当 time.precision.mode 为 adaptive_time_microseconds时映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**io.debezium.time.Date** 代表自时期起的天数。
TIME([P])	**INT64**	**io.debezium.time.MicroTime** 以微秒为单位代表时间值，不包括时区信息。PostgreSQL 允许精度 **P** 在范围 0-6 中存储最多 microsecond 精度。
TIMESTAMP (1), **TIMESTAMP (2)**, **TIMESTAMP (3)**	**INT64**	**io.debezium.time.Timestamp** 代表 epoch 过的毫秒数，不包括时区信息。
TIMESTAMP (4), TIMESTAMP **(5)**, **TIMESTAMP (6)**, **TIMESTAMP**	**INT64**	**io.debezium.time.MicroTimestamp** 代表 epoch 过的微秒数，不包括时区信息。

time.precision.mode=connect

当将 time.precision.mode 配置属性设为 connect 时，连接器会使用 Kafka Connect 逻辑类型。当消费者只能处理内置的 Kafka Connect 逻辑类型，且无法处理变量-precision 时间值时，这非常有用。但是，由于 PostgreSQL 支持 microsecond 精度，因此当数据库列具有大于 3 的 fractional second precision 值时，带有 connect 时间精度的连接器会导致 精度丢失。

表 8.17. 当 time.precision.mode 为 connect时映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**org.apache.kafka.connect.data.Date** 代表自 epoch 后的天数。
TIME([P])	**INT64**	**org.apache.kafka.connect.data.Time** 代表自午夜起的毫秒数，不包括时区信息。PostgreSQL 允许 **P** 在范围 0-6 中存储到 microsecond 精度，但当**P** 大于 3 时，这个模式会导致精度丢失。

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
TIMESTAMP([P])	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 epoch 起的毫秒数，不包括时区信息。PostgreSQL 允许 **P** 在范围 0-6 中存储到 microsecond 精度，但当**P** 大于 3 时，这个模式会导致精度丢失。

***TIMESTAMP** 类型*

***TIMESTAMP** 类型代表一个没有时区信息的时间戳。此类列根据 UTC 转换为对等的 Kafka Connect 值。例如，当 time.precision.mode 没有设置为 connect 时， TIMESTAMP 值 "2018-06-20 15:13:16.945104" 由一个带有值 "1529507596945104" 的 io.debezium.time.MicroTimestamp 代表。*

运行 Kafka Connect 和 Debezium 的 JVM 时区不会影响此转换。

PostgreSQL 支持在 TIMESTAMP 列中使用 +/-infinite 值。这些特殊的值转换为时间戳，在正无限循环的情况下值为 9223372036825200000，在负无限循环的情况值为 -9223372036832400000。这个行为模拟 PostgreSQL JDBC 驱动程序的标准行为。如需更多信息，请参阅 org.postgresql.PGStatement 接口。

十进制类型

PostgreSQL 连接器配置属性 decimal.handling.mode 的设置决定了连接器如何映射十进制类型。

当将 decimal.handling.mode 属性设置为 precise 时，连接器使用 Kafka Connect org.apache.kafka.connect.data.Decimal logical type for all DECIMAL,NUMERIC 和 MONEY 列。这是默认的模式。

表 8.18. 当 decimal.handling.mode 准确时映射

PostgreSQL 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
NUMERIC[(M[,D])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** schema 参数包含一个整数，代表十进制点被转换了多少位。

PostgreSQL 数据类型	字面类型 (schema 类型)	语义类型（模式名称）和备注
DECIMAL[(M[,D])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** schema 参数包含一个整数，代表十进制点被转换了多少位。
MONEY[(M[,D])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** schema 参数包含一个整数，代表十进制点被转换了多少位。**scale** 模式参数由 **money.fraction.digits** 连接器配置属性决定。

此规则例外。当在没有扩展限制的情况下使用 NUMERIC 或 DECIMAL 类型时，来自数据库的值会为每个值有不同的（变量）扩展。在这种情况下，连接器使用 io.debezium.data.VariableScaleDecimal，其中包含传输的值和扩展。

表 8.19. 没有扩展限制时 DECIMAL 和 NUMERIC 类型的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
数字	**STRUCT**	**io.debezium.data.VariableScaleDecimal** Contains 一个结构，它有两个字段：类型为 **INT32** 的扩展，其中包含未 **扩展** 格式的传输 **值** 和值 **BYTES**。
十进制	**STRUCT**	**io.debezium.data.VariableScaleDecimal** Contains 一个结构，它有两个字段：类型为 **INT32** 的扩展，其中包含未 **扩展** 格式的传输 **值** 和值 **BYTES**。

当将 decimal.handling.mode 属性设置为 double 时，连接器代表所有 DECIMAL、NUMERIC 和 MONEY 值作为 Java 双值，并对它们进行编码，如下表所示。

表 8.20. 当 decimal.handling.mode 为 双时的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型(schema name)
NUMERIC[(M[,D])]	**FLOAT64**	
DECIMAL[(M[,D])]	**FLOAT64**	
MONEY[(M[,D])]	**FLOAT64**	

decimal.handling.mode 配置属性的最后可能设置为 字符串。在这种情况下，连接器代表 DECIMAL、NUMERIC 和 MONEY 值作为其格式化的字符串表示，并对它们进行编码，如下表所示。

表 8.21. 当 decimal.handling.mode 是 字符串时的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型(schema name)
NUMERIC[(M[,D])]	字符串	
DECIMAL[(M[,D])]	字符串	
MONEY[(M[,D])]	字符串	

当将 decimal.handling.mode 为 字符串 或 双 时，PostgreSQL 支持 NaN （不是数字）作为存储在 DECIMAL/NUMERIC 值中的特殊值。在这种情况下，连接器将 NaN 编码为 Double.NaN 或字符串常量 NAN。

HSTORE 类型

PostgreSQL 连接器配置属性 hstore.handling.mode 的设置决定了连接器如何映射 HSTORE 值。

当 dhstore.handling.mode 属性设置为 json （默认值）时，连接器将 HSTORE 值表示为 JSON 值的字符串表示，如下表所示。当 hstore.handling.mode 属性设置为 map 时，连接器使用 HSTORE 值的 MAP 模式类型。

表 8.22. HSTORE 数据类型的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
HSTORE	字符串	**io.debezium.data.Json** 示例：使用 JSON 转换的输出表示为 **{"key" : "val"}**
HSTORE	MAP	n/a 示例：使用 JSON 转换程序的输出表示为 **{"key" : "val"}**

域类型

PostgreSQL 支持基于其他底层类型的用户定义的类型。使用此类列类型时，Debezium 会根据完整的类型层次结构公开列的表示。

重要

捕获使用 PostgreSQL 域类型的列的更改需要特别考虑。当定义列以包含扩展默认数据库类型的域类型，并且域类型定义了自定义长度或规模时，生成的模式将继承该定义长度或规模的模式。

当定义列包含扩展自定义长度或规模的另一个域类型的域类型时，生成的模式 不会继承定义的长度或扩展，因为 PostgreSQL 驱动程序的列元数据中没有这些信息。

网络地址类型

PostgreSQL 具有可以存储 IPv4、IPv6 和 MAC 地址的数据类型。最好使用这些类型而不是纯文本类型来存储网络地址。网络地址类型提供输入错误检查和专用操作器和功能。

表 8.23. 网络地址类型的映射

PostgreSQL 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
INET	**字符串**	N/a IPv4 和 IPv6 网络
CIDR	**字符串**	N/a IPv4 和 IPv6 主机和网络
MACADDR	**字符串**	N/A MAC 地址
MACADDR8	**字符串**	N/a MAC 地址(EUI-64 格式)

PostGIS 类型

PostgreSQL 连接器支持所有 PostGIS 数据类型。

表 8.24. PostGIS 数据类型的映射

PostGIS 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
GEOMETRY （计划）	**STRUCT**	**io.debezium.data.geometry.Geometry** 包含有两个字段的结构： • **srid (INT32)** - Spatial Reference System Identifier，用于定义存储在结构中的 geometry 对象类型。 • **wkb (BYTES)** - 以 Well-Known-Binary 格式编码的 geometry 对象的二进制表示。 有关格式详情，请参阅 Open Geospatial Consortium Simple Features Access specification。
GEOGRAPHY （设备）	**STRUCT**	**io.debezium.data.geometry.Geography** 包含有两个字段的结构： • **srid (INT32)** - Spatial Reference System Identifier，用于定义存储在结构中的geography 对象类型。 • **wkb (BYTES)** - 以 Well-Known-Binary 格式编码的 geometry 对象的二进制表示。 有关格式详情，请参阅 Open Geospatial Consortium Simple Features Access specification。

要粘贴的值

***PostgreSQL** 对页面大小具有硬限制。这意味着，大于 8 KB 的值需要使用 **TOAST** 存储来存储。这会影响来自数据库的复制消息。使用 TOAST 机制存储的值且尚未更改的值不会包含在消息中，除非它们是表的副本身份的一部分。Debezium 无法安全地从数据库读取缺少的值，因为这可能导致竞争条件。因此，Debezium 遵循这些规则来处理粘贴值：*

- ***具有 REPLICA IDENTITY FULL 的表 - TOAST 列值是更改事件的 before 和 after 字段的一部分，就像任何其他列一样。***

- ***带有 REPLICA IDENTITY DEFAULT 的表 - 从数据库接收 UPDATE 事件时，任何没有包括在事件中的 TOAST 列值。同样，在收到 DELETE 事件时，如果是 TOAST，则没有 TOAST 列（若有）。因为 Debezium 无法安全地提供列值，因此连接器会返回一个占位符值，如连接器配置属性 unavailable.value.placeholder 定义。***

默认值

如果为数据库模式中的列指定了默认值，PostgreSQL 连接器将尽可能尝试将此值传播到 Kafka 模式。最常见的数据类型受到支持，包括：

- 布尔值
- 数字类型(INT、FLOAT、NUMERIC 等)
- 文本类型(CHAR、VARCHAR、TEXT 等)
- 时序类型(DATE,TIME,INTERVAL,TIMESTAMP,TIMESTAMPTZ)
- JSON,JSONB,XML
- UUID

请注意，对于临时类型，对默认值的解析由 PostgreSQL 库提供；因此，PostgreSQL 通常支持的任何字符串表示也应该被连接器支持。

如果默认值由函数生成，而不是直接指定，则连接器将为给定的数据类型导出等效的 0。这些值包括：

- FALSE for BOOLEAN
- 0 带有适当的精度，用于数字类型
- text/XML 类型的空字符串
- JSON 类型的 {}
- 1970-01-01 for DATE,TIMESTAMP,TIMESTAMPTZ 类型

- TIME的 00:00
- INTERVAL 为EPOCH
- 00000000-0000-0000-0000-000000000000 用于 UUID

目前，这个支持只扩展到明确使用功能。例如，CURRENT_TIMESTAMP (6) 支持括号，但 CURRENT_TIMESTAMP 不支持。

重要

当将 PostgreSQL 连接器与强制实施模式版本间兼容的模式 registry 时，支持对默认值的传播主要允许安全模式演进。由于这个主要关注，以及不同插件的刷新行为，Kafka 模式中存在的默认值无法保证始终与数据库模式中的默认值同步。

- 默认值可能会在 Kafka 模式中显示"late"，具体取决于给定插件触发刷新内存模式的时间/方式。如果默认更改多次更改，则值永远不会在 Kafka 模式中显示/被跳过
- 如果在连接器等待处理记录时触发模式刷新，则默认值可能会在 Kafka 模式中出现 'early'。这是因为列元数据在刷新时从数据库读取，而不是在复制消息中存在。如果连接器后端并出现刷新，或者如果连接器在更新继续写入源数据库时停止了连接器，则可能会发生这种情况。

此行为可能是意外的，但它仍然是安全的。只有架构定义会受到影响，而消息中存在的实际值将与写入源数据库的实际值保持一致。

8.5. 设置 POSTGRESQL 以运行 DEBEZIUM 连接器

此 Debezium 发行版本只支持原生 pgoutput 逻辑复制流。要设置 PostgreSQL，使其使用 pgoutput 插件，您必须启用复制插槽，并配置具有足够特权的用户来执行复制。

详情包括在以下主题中：

- 第 8.5.1 节 “为 Debezium pgoutput 插件配置复制插槽”
- 第 8.5.2 节 “为 Debezium 连接器设置 PostgreSQL 权限”
- 第 8.5.3 节 “设置特权，以便 Debezium 创建 PostgreSQL 出版物”
- 第 8.5.4 节 “配置 PostgreSQL 以允许使用 Debezium 连接器主机进行复制”
- 第 8.5.5 节 “配置 PostgreSQL 以管理 Debezium WAL 磁盘空间消耗”
- 第 8.5.6 节 “升级 Debezium 捕获的 PostgreSQL 数据库”

8.5.1. 为 Debezium pgoutput 插件配置复制插槽

PostgreSQL 的逻辑解码使用复制插槽。要配置复制插槽，请在 postgresql.conf 文件中指定以下内容：

```
wal_level=logical
max_wal_senders=1
max_replication_slots=1
```

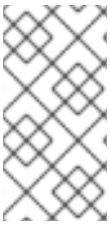
这些设置指示 PostgreSQL 服务器，如下所示：

- wal_level - 使用 write-ahead 日志的逻辑解码。
- max_wal_senders - 使用最多一个单独的进程来处理 WAL 更改。
- max_replication_slots - 允许最多创建一个复制插槽来流传输 WAL 更改。

保证复制插槽可以保证保留 Debezium 所需的所有 WAL 条目，即使在 Debezium 中断期间也是如此。因此，务必要密切监控复制插槽以避免：

- 过多的内存消耗
- 任何条件，如目录 bloat，当复制插槽未使用过长时会出现这种情况

如需更多信息，请参阅 PostgreSQL 文档用于复制插槽。



注意

熟悉 PostgreSQL write-ahead 日志的 mechanics 和 configuration 有助于使用 Debezium PostgreSQL 连接器。

8.5.2. 为 Debezium 连接器设置 PostgreSQL 权限

设置 PostgreSQL 服务器以运行 Debezium 连接器需要数据库用户可以执行复制。复制只能由具有适当权限的数据库用户执行，并且仅对配置的数量的主机执行。

虽然默认情况下，超级用户具有必要的 REPLICATION 和 LOGIN 角色，如 Security 中所述，最好不要为 Debezium 复制用户提供升级特权。相反，创建一个具有最低所需权限的 Debezium 用户。

先决条件

- PostgreSQL 管理权限。

流程

1. 要为用户授予复制权限，请定义一个 至少具有 REPLICATION 和 LOGIN 权限的 PostgreSQL 角色，然后将该角色授予该用户。例如：

```
CREATE ROLE <name> REPLICATION LOGIN;
```

8.5.3. 设置特权，以便 Debezium 创建 PostgreSQL 出版物

来自为表创建的 publications 的 PostgreSQL 源表的 Debezium 流改变事件。出版物包含一组过滤的更改事件，这些事件由一个或多个表生成。每个发布中的数据会根据发布规格进行过滤。规范可由 PostgreSQL 数据库管理员或 Debezium 连接器创建。要允许 Debezium PostgreSQL 连接器创建发布并指定要复制到它们的数据，连接器必须使用数据库中的特定权限运行。

有几个选项可用于确定如何创建发布。通常，在设置连接器前，最好为要捕获的表手动创建发布。但是，您可以以允许 Debezium 自动创建发布的方式配置您的环境，并指定添加到其中的数据。

Debezium 使用 list 和 exclude list 属性来指定如何在发布中插入数据。有关启用 Debezium 创建发布的选项的更多信息，请参阅 publication.autocreate.mode。

要使 Debezium 创建 PostgreSQL 发布，它必须以具有以下权限的用户运行：

- 数据库中的复制特权，将表添加到发布中。
- 数据库上的 CREATE 特权来添加发布。
- 表上的 SELECT 特权，以复制初始表数据。表所有者自动具有表的 SELECT 权限。

要向发布添加表，用户必须是表的所有者。但是，由于源表已存在，您需要一种与原始所有者共享所有权的机制。要启用共享所有权，您可以创建一个 PostgreSQL 复制组，然后将现有的表所有者和复制用户添加到组中。

流程

1. 创建复制组。

```
CREATE ROLE <replication_group>;
```

2. 将表的原始所有者添加到组。

```
GRANT REPLICATION_GROUP TO <original_owner>;
```

3. 将 Debezium 复制用户添加到组中。

```
GRANT REPLICATION_GROUP TO <replication_user>;
```

4. 将表的所有权转让到 < replication_group>。

```
ALTER TABLE <table_name> OWNER TO REPLICATION_GROUP;
```

对于 Debezium 指定捕获配置，必须将 publication.autocreate.mode 的值设置为 过滤。

8.5.4. 配置 PostgreSQL 以允许使用 Debezium 连接器主机进行复制

要启用 Debezium 复制 PostgreSQL 数据，您必须将数据库配置为允许使用运行 PostgreSQL 连接器的主机进行复制。要指定允许使用数据库复制的客户端，请在基于 PostgreSQL 主机的身份验证文件 pg_hba.conf 中添加条目。有关 pg_hba.conf 文件的更多信息，请参阅 PostgreSQL 文档。

流程

- 在 pg_hba.conf 文件中添加条目，以指定可以使用数据库主机复制的 Debezium 连接器主机。例如，

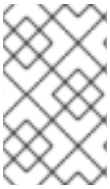
  pg_hba.conf 文件示例：

```
local   replication     <youruser>                  trust   1
host    replication     <youruser>  127.0.0.1/32        trust   2
host    replication     <youruser>  ::1/128           trust   3
```

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 指示服务器允许在本地对 < youruser& gt; 进行复制，即在服务器机器上进行复制。

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 指示服务器允许 localhost 上的 & lt;youruser > 使用 IPV4 接收复制更改。

3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 指示服务器允许 localhost 上的 & lt;youruser > 使用 IPV6 接收复制更改。



注意

有关网络掩码的更多信息，请参阅 PostgreSQL 文档。

8.5.5. 配置 PostgreSQL 以管理 Debezium WAL 磁盘空间消耗

在某些情况下，PostgreSQL 磁盘空间可以被 WAL 文件使用，以激增或增加通常的比例。这种情况下有几个可能的原因：

- 连接器收到数据的 LSN 在服务器的 pg_replication_slots 视图的 confirmed_flush_lsn 列中可用。此 LSN 旧数据不再可用，数据库负责回收磁盘空间。

  另外，在 pg_replication_slots 视图中，restart_lsn 列包含连接器可能需要的最旧的 WAL 的 LSN。如果 confirmed_flush_lsn 的值定期增加和 restart_lsn lags 的值，则数据库需要回收空间。

  数据库通常会回收批处理块中的磁盘空间。这是预期的行为，用户不需要任何操作。

- 数据库中有很多更新被跟踪，但只有少量更新与连接器捕获更改的表和模式相关。这种情形可以通过定期的心跳事件轻松解决。设置 heartbeat.interval.ms 连接器配置属性。

- PostgreSQL 实例包含多个数据库，其中一个是高流量数据库。Debezium 捕获另一个数据库中的更改，这些数据库与其它数据库相比是低流量。然后，Debezium 无法确认 LSN 作为每个数据库的复制插槽工作，而且 Debezium 不会被调用。当 WAL 由所有数据库共享时，使用的数量会增加，直到 Debezium 正在捕获更改的数据库发出事件。要克服这个问题，需要：

  - 启用使用 heartbeat.interval.ms 连接器配置属性的定期心跳记录生成。
  - 定期从 Debezium 捕获更改的数据库发出更改事件。

  然后，一个单独的进程会通过插入新行或重复更新同一行来定期更新表。然后 PostgreSQL 会调用 Debezium，它会确认最新的 LSN 并允许数据库回收 WAL 空间。此任务可以通过 heartbeat.action.query 连接器配置属性进行自动处理。

在同一数据库服务器设置多个连接器

Debezium 使用复制插槽从数据库流更改。这些复制插槽以 LSN (Log Sequence Number)的形式维

护当前位置，该位置是 Debezium 连接器消耗的 WAL 中的位置。这有助于 PostgreSQL 使 WAL 可用，直到 Debezium 处理为止。单个复制插槽只能针对单个消费者或进程存在，因为不同的消费者可能具有不同的状态，并且可能需要来自不同位置的数据。

由于复制插槽只能由单个连接器使用，因此每个 Debezium 连接器都需要创建一个唯一的复制插槽。虽然当连接器没有激活时，Postgres 可能会允许其他连接器消耗复制插槽 - 这可能会导致数据丢失，因为插槽将只发出每个更改 [请参阅 more]。

除了复制插槽外，Debebe 使用发布来流传输事件，在使用 pgoutput 插件时。与复制插槽类似，发布在数据库级别，并定义为一组表。因此，您需要为每个连接器有一个唯一的发布，除非连接器可用于同一组表。有关启用 Debezium 创建发布的选项的更多信息，请参阅 publication.autocreate.mode

有关如何为每个连接器设置唯一的复制插槽名称和发布名称，请参阅 slot.name 和 publication.name。

8.5.6. 升级 Debezium 捕获的 PostgreSQL 数据库

当升级 Debezium 使用的 PostgreSQL 数据库时，您必须执行特定的步骤来防止数据丢失，并确保 Debezium 继续操作。通常，Debezium 能够应对网络故障和其他中断导致的中断。例如，当连接器监控的数据库服务器停止或崩溃时，在连接器重新建立与 PostgreSQL 服务器的通信后，它会继续从日志序列号(LSN)偏移记录的最后位置读取。连接器从 Kafka Connect offsets 主题检索有关最后一次记录的偏移信息，并查询配置的 PostgreSQL 复制插槽以获取具有相同值的日志序列号(LSN)。

要使连接器启动和捕获 PostgreSQL 数据库中的更改事件，必须存在复制插槽。但是，作为 PostgreSQL 升级过程的一部分，复制插槽会被删除，在升级完成后不会恢复原始插槽。因此，当连接器重启并请求复制插槽的最后已知的偏移时，PostgreSQL 无法返回信息。

您可以创建新的复制插槽，但您必须有超过创建新插槽才能防止数据丢失。新的复制插槽只能为创建插槽后发生的更改提供 LSNs；无法为升级前发生的事件提供偏移量。当连接器重启时，它会首先从 Kafka offsets 主题请求最后已知的偏移量。然后，它会向复制插槽发送请求，以返回从偏移主题中检索的偏移信息。但是，新的复制插槽无法提供连接器从预期位置恢复流所需的信息。然后，连接器会跳过日志中任何现有更改事件，且只从日志中的最新位置恢复流。这可能导致静默数据丢失：连接器没有为跳过的事件发出记录，而不提供任何信息来指示这些事件已被跳过。

有关如何执行 PostgreSQL 数据库升级以便 Debezium 能够继续捕获事件，同时最大程度降低数据丢失的风险，请参阅以下步骤。

流程

1. 临时停止写入数据库的应用程序，或将其置于只读模式。

2. 备份数据库。

3. 临时禁用对数据库的写入访问权限。

4. 在阻止写操作保存到 write-ahead 日志(WAL)前，验证数据库中发生了的任何更改，并且 WAL LSN 是否反映在复制插槽上。

5. 为连接器提供足够的时间，以捕获写入复制插槽的所有事件记录。
此步骤可确保在停机被考虑前发生的所有更改事件，并将其保存到 Kafka 中。

6. 通过检查 flushed LSN 的值，验证连接器是否已消耗来自复制插槽的条目。

7. 通过停止 Kafka Connect 来安全地关闭连接器。
Kafka Connect 会停止连接器，将所有事件记录刷新到 Kafka，并记录从每个连接器接收的最后一个偏移。

注意

作为停止整个 Kafka Connect 集群的替代方案，您可以通过删除它来停止连接器。不要删除偏移主题，因为它可能由其他 Kafka 连接器共享。之后，在恢复对数据库的写入访问并准备好重启连接器后，您必须重新创建连接器。

8. 作为 PostgreSQL 管理员，丢弃主数据库服务器上的复制插槽。不要使用 slot.drop.on.stop 属性来丢弃复制插槽。此属性仅用于测试。

9. 停止数据库。

10. 使用批准的 PostgreSQL 升级步骤（如 pg_upgrade ）或 pg_dump 和 pg_restore 执行升级。

11.

（可选）使用标准 Kafka 工具从偏移存储主题中删除连接器偏移。
有关如何删除连接器偏移的示例，请参阅 如何删除 Debezium 社区常见问题解答中的连接器偏移。

12. 重新启动数据库。

13. 作为 PostgreSQL 管理员，在数据库上创建 Debezium 逻辑复制插槽。在启用对数据库进行写入前，您必须创建插槽。否则，Debezium 无法捕获更改，从而导致数据丢失。

    有关设置复制插槽的详情，请参考 第 8.5.1 节 “为 Debezium pgoutput 插件配置复制插槽”。

14. 验证在升级后是否仍然存在了定义 Debezium 要捕获的表的发布。如果发布不可用，请以 PostgreSQL 管理员身份连接到数据库，以创建新的发布。

15. 如果需要在上一步中创建新发布，请更新 Debezium 连接器配置，将新发布的名称添加到 publication.name 属性。

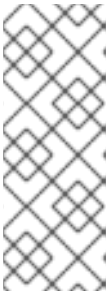
16. 在连接器配置中，重命名连接器。

17. 在连接器配置中，将 slot.name 设置为 Debezium 复制插槽的名称。

18. 验证新复制插槽是否可用。

19. 恢复对数据库的写入访问权限，然后重新启动写入数据库的任何应用程序。

20. 在连接器配置中，将 snapshot.mode 属性设置为 never，然后重启连接器。



注意

如果您无法验证 Debezium 是否完成了第 6 步中的所有数据库更改，您可以通过设置 snapshot.mode=initial 将连接器配置为执行新快照。如果需要，您可以通过检查升级前立即执行的数据库备份内容来确认连接器是否从复制插槽读取所有更改。

其他资源

- 为 Debezium 配置复制插槽。

8.6. 部署 DEBEZIUM POSTGRESQL 连接器

您可以使用以下任一方法部署 Debezium PostgreSQL 连接器：

- 使用 AMQ Streams 自动创建包含连接器插件的镜像。

  这是首选的方法。

- 从 Dockerfile 构建自定义 Kafka Connect 容器镜像。

其他资源

- 第 8.6.5 节 “Debezium PostgreSQL 连接器配置属性的描述”

8.6.1. 使用 AMQ Streams 部署 PostgreSQL 连接器

从 Debezium 1.7 开始，部署 Debezium 连接器的首选方法是使用 AMQ Streams 构建包含连接器插件的 Kafka Connect 容器镜像。

在部署过程中，您可以创建并使用以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR，并包含有关镜像中需要包含连接器工件的信息。

- KafkaConnector CR，提供包括连接器用来访问源数据库的信息。在 AMQ Streams 启动 Kafka Connect pod 后，您可以通过应用 KafkaConnector CR 来启动连接器。

在 Kafka Connect 镜像的构建规格中，您可以指定可用于部署的连接器。对于每个连接器插件，您还可以指定您的部署可以使用的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。当 AMQ Streams 构建 Kafka Connect 镜像时，它会下载指定的工件，并将其合并到镜像中。

KafkaConnect CR 中的 spec.build.output 参数指定存储生成的 Kafka Connect 容器镜像的位置。容器镜像可以存储在 Docker registry 中，也可以存储在 OpenShift ImageStream 中。要将镜像存储在 ImageStream 中，您必须在部署 Kafka Connect 前创建 ImageStream。镜像流不会被自动创建。

注意

如果使用 KafkaConnect 资源来创建集群，之后无法使用 Kafka Connect REST API 创建或更新连接器。您仍然可以使用 REST API 来检索信息。

其他资源

- 在 OpenShift 中使用 AMQ Streams 配置 Kafka 连接。
- 在 OpenShift 中部署和管理 AMQ Streams 中，使用 AMQ Streams 自动创建新容器镜像。

8.6.2. 使用 AMQ Streams 部署 Debezium PostgreSQL 连接器

使用早期版本的 AMQ Streams 时，要在 OpenShift 上部署 Debezium 连接器，您需要首先为连接器构建 Kafka Connect 镜像。在 OpenShift 上部署连接器的当前首选方法是使用 AMQ Streams 中的构建配置来构建 Kafka Connect 容器镜像，其中包含您要使用的 Debezium 连接器插件。

在构建过程中，AMQ Streams Operator 将 KafkaConnect 自定义资源（包括 Debezium 连接器定义）中的输入参数转换为 Kafka Connect 容器镜像。构建会从 Red Hat Maven 存储库或其他配置的 HTTP 服务器下载必要的工件。

新创建的容器被推送到在 .spec.build.output 中指定的容器 registry，用于部署 Kafka Connect 集群。在 AMQ Streams 构建 Kafka Connect 镜像后，您可以创建 KafkaConnector 自定义资源来启动构建中包含的连接器。

先决条件

- 您可以访问安装了集群 Operator 的 OpenShift 集群。
- AMQ Streams Operator 正在运行。

- 在 OpenShift 中部署和升级 AMQ Streams 所述，会部署 Apache Kafka 集群。
- Kafka Connect 在 AMQ Streams 上部署
- 您有一个 Red Hat Integration 许可证。
- 已安装 OpenShift oc CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。
- 根据您要存储 Kafka Connect 构建镜像的方式，您需要 registry 权限，或者您必须创建 ImageStream 资源：

  将构建镜像存储在镜像 registry 中，如 Red Hat Quay.io 或 Docker Hub

  - 在 registry 中创建和管理镜像的帐户和权限。

  将构建镜像存储为原生 OpenShift ImageStream

  - ImageStream 资源已部署到集群中，以存储新的容器镜像。您必须为集群显式创建 ImageStream。默认无法使用镜像流。如需有关 ImageStreams 的更多信息，请参阅在 OpenShift Container Platform 中管理镜像流。

流程

1. 登录 OpenShift 集群。
2. 为连接器创建 Debezium KafkaConnect 自定义资源(CR)，或修改现有的资源。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 metadata.annotations 和 spec.build 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

> 例 8.1. 定义包含 Debezium 连接器的 KafkaConnect 自定义资源的 dbz-connect.yaml 文件
>
> 在以下示例中，自定义资源被配置为下载以下工件：

- *Debezium PostgreSQL 连接器存档。*

- *Service Registry 归档。Service Registry 是一个可选组件。只有在打算将 Avro 序列化与连接器搭配使用时，才添加 Service Registry 组件。*

- *Debezium 脚本 SMT 归档以及您要与 Debezium 连接器一起使用的关联脚本引擎。SMT 归档和脚本语言依赖项是可选组件。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时，才添加这些组件。*

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: debezium-kafka-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  version: 3.5.0
  build: 2
    output: 3
      type: imagestream 4
      image: debezium-streams-connect:latest
    plugins: 5
      - name: debezium-connector-postgres
        artifacts:
          - type: zip 6
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-
connector-postgres/2.3.4.Final-redhat-00001/debezium-connector-postgres-
2.3.4.Final-redhat-00001-plugin.zip 7
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-
distro-connect-converter/2.4.4.Final-redhat-<build-number>/apicurio-registry-distro-
connect-converter-2.4.4.Final-redhat-<build-number>.zip 8
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-
scripting/2.3.4.Final-redhat-00001/debezium-scripting-2.3.4.Final-redhat-00001.zip
9
          - type: jar
            url:
https://repo1.maven.org/maven2/org/codehaus/groovy/groovy/3.0.11/groovy-
3.0.11.jar 10
          - type: jar
            url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-
jsr223/3.0.11/groovy-jsr223-3.0.11.jar
          - type: jar
            url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-
json3.0.11/groovy-json-3.0.11.jar
```

```
    bootstrapServers: debezium-kafka-cluster-kafka-bootstrap:9093
    ...
```

表 8.25. Kafka Connect 配置设置的描述

项	描述
1	将 **strimzi.io/use-connector-resources** 注解设置为 **"true"**，使 Cluster Operator 使用 **KafkaConnector** 资源在此 Kafka Connect 集群中配置连接器。
2	**spec.build** 配置指定在镜像中存储构建镜像的位置，并列出要在镜像中包含的插件，以及插件工件的位置。
3	**build.output** 指定存储新构建镜像的 registry。
4	指定镜像输出的名称和镜像名称。**output.type** 的有效值是 **要推送到** 容器 registry（如 Docker Hub 或 Quay）或 **镜像流** 的有效值，以将镜像推送到内部 OpenShift ImageStream。要使用 ImageStream，必须将 ImageStream 资源部署到集群中。有关在 KafkaConnect 配置中指定 **build.output** 的更多信息，请参阅在 OpenShift 中配置 AMQ Streams 中的 AMQ Streams Build schema 参考。
5	**plugins** 配置列出了您要包含在 Kafka Connect 镜像中的所有连接器。对于列表中的每个条目，指定一个插件**名称**，以及有关构建连接器所需的工件的信息。另外，对于每个连接器插件，您还可以包含可用于连接器的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。
6	**artifacts.type** 的值指定在 **artifacts.url** 中指定的工件类型。有效类型为 **zip**、**tgz** 或 **jar**。Debezium 连接器存档以 **.zip** 文件格式提供。**类型** 值必须与 **url** 字段中引用的文件类型匹配。
7	**artifacts.url** 的值指定 HTTP 服务器的地址，如 Maven 存储库，用于存储连接器工件的文件。Debezium 连接器工件在 Red Hat Maven 存储库中提供。OpenShift 集群必须有权访问指定的服务器。
8	（可选）指定用于下载 Service Registry 组件的工件 **类型和 url**。包含 Service Registry 工件，只有在您希望连接器使用 Apache Avro 来序列化带有 Service Registry 的事件键和值时，而不是使用默认的 JSON 转换程序。
9	（可选）指定 Debezium 脚本 SMT 归档的工件 **类型和 url**，以用于 Debezium 连接器。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时才包括脚本 SMT。要使用脚本 SMT，您必须部署 JSR 223 兼容脚本实现，如 groovy。

项	描述
10	（可选）指定 JSR 223 兼容脚本实施的 JAR 文件的工件 **类型和 url**，这是 Debezium 脚本 SMT 所需的。  **重要** 如果使用 AMQ Streams 将连接器插件合并到 Kafka Connect 镜像中，每个所需的脚本语言 **工件。url** 必须指定 JAR 文件的位置，并且 **artifacts.type** 的值也必须设置为 **jar**。无效的值会导致连接器在运行时失败。 要启用带有脚本 SMT 的 Apache Groovy 语言，示例中的自定义资源会为以下库检索 JAR 文件： • **groovy** • **Groovy-jsr223**（指定代理） • **groovy-json**（解析 JSON 字符串的模块） 作为替代方案，Debebe Debezium 脚本 SMT 也支持使用 JSR 223 实现 GraalVM JavaScript。

3. *输入以下命令将 KafkaConnect 构建规格应用到 OpenShift 集群：*

```
oc create -f dbz-connect.yaml
```

根据自定义资源中指定的配置，Streams Operator 准备要部署的 Kafka Connect 镜像。构建完成后，Operator 将镜像推送到指定的 registry 或 ImageStream，并启动 Kafka Connect 集群。集群中提供了您在配置中列出的连接器工件。

4. *创建一个 KafkaConnector 资源来定义您要部署的每个连接器的实例。*
例如，创建以下 KafkaConnector CR，并将它保存为 postgresql-inventory-connector.yaml

例 8.2. 为 Debezium 连接器定义 KafkaConnector 自定义资源的 postgresql-inventory-connector.yaml 文件

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: debezium-kafka-connect-cluster
  name: inventory-connector-postgresql 1
spec:
  class: io.debezium.connector.postgresql.PostgresConnector 2
  tasksMax: 1 3
  config: 4
```

```
database.hostname: postgresql.debezium-postgresql.svc.cluster.local 5
database.port: 5432 6
database.user: debezium 7
database.password: dbz 8
database.dbname: mydatabase 9
topic.prefix: inventory-connector-postgresql 10
table.include.list: public.inventory 11

...
```

表 8.26. 连接器配置设置的描述

项	描述
1	使用 Kafka Connect 集群注册的连接器名称。
2	连接器类的名称。
3	可以同时操作的任务数量。
4	连接器的配置。
5	主机数据库实例的地址。
6	数据库实例的端口号。
7	Debezium 用于连接到数据库的帐户名称。
8	Debezium 用于连接到数据库用户帐户的密码。
9	要从中捕获更改的数据库名称。
10	数据库实例或集群的主题前缀。 指定的名称只能由字母数字字符或下划线组成。 因为主题前缀被用作从这个连接器接收更改事件的任何 Kafka 主题的前缀，所以该名称在集群中的连接器之间必须是唯一的。 如果连接器与 Avro 连接器集成，则此命名空间也用于相关 Kafka Connect 模式的名称，以及相应 Avro 模式的命名空间。
11	连接器捕获更改事件的表列表。

5. *运行以下命令来创建连接器资源：*

```
oc create -n <namespace> -f <kafkaConnector>.yaml
```

例如，

```
oc create -n debezium -f {context}-inventory-connector.yaml
```

连接器注册到 Kafka Connect 集群，并开始针对 KafkaConnector CR 中的 spec.config.database.dbname 指定的数据库运行。连接器 pod 就绪后，Debebe 正在运行。

现在，您可以验证 Debezium PostgreSQL 部署。

8.6.3. 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium PostgreSQL 连接器

要部署 Debezium PostgreSQL 连接器，您需要构建包含 Debezium 连接器存档的自定义 Kafka Connect 容器镜像，并将此容器镜像推送到容器 registry。然后，您需要创建两个自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR。CR 中的 image 属性指定您创建的容器镜像的名称，以运行 Debezium 连接器。您可以将此 CR 应用到部署 Red Hat AMQ Streams 的 OpenShift 实例。AMQ Streams 提供将 Apache Kafka 带到 OpenShift 的 operator 和镜像。
- 定义 Debezium Db2 连接器的 KafkaConnector CR。将此 CR 应用到应用 KafkaConnect CR 的同一 OpenShift 实例。

先决条件

- PostgreSQL 正在运行，并执行 了将 PostgreSQL 设置为运行 Debezium 连接器 的步骤。
- AMQ Streams 部署在 OpenShift 中，并运行 Apache Kafka 和 Kafka Connect。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams。
- podman 或 Docker 已安装。
- 您有一个在容器 registry 中创建和管理容器（如 quay.io 或 docker.io）的帐户和权限，您要添加将运行 Debezium 连接器的容器。

流程

1. *为 Kafka Connect 创建 Debezium PostgreSQL 容器：*

a. *创建一个使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 的 Dockerfile 作为基础镜像。例如，在终端窗口中输入以下命令：*

```
cat <<EOF >debezium-container-for-postgresql.yaml (1)
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/debezium (2)
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-postgres/2.3.4.Final-redhat-00001/debezium-connector-postgres-2.3.4.Final-redhat-00001-plugin.zip \
&& unzip debezium-connector-postgres-2.3.4.Final-redhat-00001-plugin.zip \
&& rm debezium-connector-postgres-2.3.4.Final-redhat-00001-plugin.zip
RUN cd /opt/kafka/plugins/debezium/
USER 1001
EOF
```

项	描述
1	您可以指定您想要的任何文件名。
2	指定 Kafka Connect 插件目录的路径。如果您的 Kafka Connect 插件目录位于不同的位置，请将此路径替换为目录的实际路径。

该命令在当前目录中创建一个名为 debezium-container-for-postgresql.yaml 的 Dockerfile。

b. *从您在上一步中创建的 debezium-container-for-postgresql.yaml Docker 文件中构建容器镜像。在包含文件的目录中，打开终端窗口并输入以下命令之一：*

```
podman build -t debezium-container-for-postgresql:latest .
```

```
docker build -t debezium-container-for-postgresql:latest .
```

build 命令使用名称 debezium-container-for-postgresql 构建容器镜像。

c. *将自定义镜像推送到容器 registry，如 quay.io 或内部容器 registry。容器 registry 必须可供您要部署镜像的 OpenShift 实例使用。输入以下命令之一：*

```
podman push <myregistry.io>/debezium-container-for-postgresql:latest
```

```
docker push <myregistry.io>/debezium-container-for-postgresql:latest
```

d.

创建新的 Debezium PostgreSQL KafkaConnect 自定义资源(CR)。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 注解和 镜像 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  image: debezium-container-for-postgresql 2

  ...
```

项	描述
1	**metadata.annotations** 表示 **KafkaConnector** 资源用于配置在这个 Kafka Connect 集群中使用的 Cluster Operator。
2	**spec.image** 指定您创建的镜像的名称，以运行 Debezium 连接器。此属性覆盖 Cluster Operator 中的 **STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE** 变量。

e.

运行以下命令，将 KafkaConnect CR 应用到 OpenShift Kafka 实例：

```
oc create -f dbz-connect.yaml
```

这会更新 OpenShift 中的 Kafka Connect 环境，以添加 Kafka Connector 实例，该实例指定您为运行 Debezium 连接器而创建的镜像名称。

2.

创建一个 KafkaConnector 自定义资源来配置 Debezium PostgreSQL 连接器实例。

您可以在 .yaml 文件中配置 Debezium PostgreSQL 连接器，该文件指定连接器的配置属性。连接器配置可能指示 Debezium 为 schema 和表的子集生成事件，或者可能会设置属性，以便 Debezium 忽略、掩码或截断敏感、太大或不需要的指定列中的值。有关您可以为 Debezium PostgreSQL 连接器设置的配置属性的完整列表，请参阅 PostgreSQL 连接器属性。

以下示例显示了一个自定义资源的摘录，该资源在端口 5432 上配置一个 Debezium 连接器连接到 PostgreSQL 服务器主机 192.168.99.100。此主机有一个名为 sampledb 的数据库，名为 public 的 schema，inventory-connector-postgresql 是服务器的逻辑名称。

inventory-connector.yaml

```
apiVersion: kafka.strimzi.io/v1beta2
 kind: KafkaConnector
 metadata:
  name: inventory-connector-postgresql (1)
  labels:
   strimzi.io/cluster: my-connect-cluster
 spec:
  class: io.debezium.connector.postgresql.PostgresConnector
  tasksMax: 1 (2)
  config: (3)
   database.hostname: 192.168.99.100 (4)
   database.port: 5432
   database.user: debezium
   database.password: dbz
   database.dbname: sampledb
   topic.prefix: inventory-connector-postgresql (5)
   schema.include.list: public (6)
   plugin.name: pgoutput (7)

   ...
```

(1)(1)(1)(1)(1)

连接器的名称。

(2)(2)(2)(2)(2)

任何时候都只能运行一个任务。因为 PostgreSQL 连接器使用单一连接器任务读取 PostgreSQL 服务器的 binlog 可确保正确的顺序和事件处理。Kafka Connect 服务使用连接器来启动一个或多个完成工作的任务，并在 Kafka Connect 服务集群中自动分发正在运行的任务。如果有任何服务停止或崩溃，则这些任务将重新分发到运行的服务。

(3)(3)(3)

连接器的配置。

(4)(4)(4)

5 5 5

唯一的主题前缀。服务器名称是 PostgreSQL 服务器或服务器集群的逻辑标识符。此名称用作接收更改事件记录的所有 Kafka 主题的前缀。

6 6 6

连接器只捕获 public 模式中的更改。可以将连接器配置为仅捕获您选择的表中的更改。如需更多信息，请参阅 table.include.list。

7 7 7

在 PostgreSQL 服务器上安装的 PostgreSQL 逻辑解码插件的名称。虽然 PostgreSQL 10 及之后的版本唯一支持的值是 pgoutput，但您必须将 plugin.name 明确设置为 pgoutput。

3. 使用 Kafka Connect 创建连接器实例。例如，如果您将 KafkaConnector 资源保存在 inventory-connector.yaml 文件中，您将运行以下命令：

```
oc apply -f inventory-connector.yaml
```

这会注册 inventory-connector，连接器开始针对 KafkaConnector CR 中定义的 sampledb 数据库运行。

结果

连接器启动后，它会对配置了连接器的 PostgreSQL 服务器数据库执行一致的快照。然后，连接器开始为行级操作生成数据更改事件，并将事件记录流传输到 Kafka 主题。

8.6.4. 验证 Debezium PostgreSQL 连接器是否正在运行

如果连接器正确启动且没有错误，它会为每个连接器配置为捕获的表创建一个主题。下游应用程序可以订阅这些主题，以检索源数据库中发生的信息事件。

要验证连接器是否正在运行，您可以从 OpenShift Container Platform Web 控制台或 OpenShift CLI 工具(oc)执行以下操作：

- 验证连接器状态。

- *验证连接器是否生成主题。*
- *验证主题是否填充了读取操作("op":"r")的事件，连接器在每个表的初始快照中生成。*

先决条件

- **Debezium** *连接器部署到* **OpenShift** *上的* **AMQ Streams。**
- *已安装* **OpenShift oc CLI** *客户端。*
- *访问* **OpenShift Container Platform web** *控制台。*

流程

1. *使用以下方法之一检查* **KafkaConnector** *资源的状态：*
   - *在* **OpenShift Container Platform Web** *控制台中：*
     a. *导航到* **Home → Search。**
     b. *在* **Search** *页面中，点* **Resources** *打开* **Select Resource** *框，然后键入* **KafkaConnector。**
     c. *在* **KafkaConnectors** *列表中，点您要检查的连接器的名称，如* **inventory-connector-postgresql。**
     d. *在* **Conditions** *部分，验证* **Type** *和* **Status** *列中的值是否已设置为* **Ready** *和* **True。**
   - *在终端窗口中：*

a.

使用以下命令：

```
oc describe KafkaConnector <connector-name> -n <project>
```

例如，

```
oc describe KafkaConnector inventory-connector-postgresql -n debezium
```

该命令返回类似以下示例的状态信息：

例 8.3. KafkaConnector 资源状态

```
Name:         inventory-connector-postgresql
Namespace:    debezium
Labels:       strimzi.io/cluster=debezium-kafka-connect-cluster
Annotations:  <none>
API Version:  kafka.strimzi.io/v1beta2
Kind:         KafkaConnector

...

Status:
  Conditions:
    Last Transition Time:  2021-12-08T17:41:34.897153Z
    Status:                True
    Type:                  Ready
  Connector Status:
    Connector:
      State:      RUNNING
      worker_id:  10.131.1.124:8083
    Name:         inventory-connector-postgresql
    Tasks:
      Id:               0
      State:            RUNNING
      worker_id:        10.131.1.124:8083
    Type:               source
  Observed Generation:  1
  Tasks Max:            1
  Topics:
    inventory-connector-postgresql.inventory
    inventory-connector-postgresql.inventory.addresses
    inventory-connector-postgresql.inventory.customers
    inventory-connector-postgresql.inventory.geom
    inventory-connector-postgresql.inventory.orders
    inventory-connector-postgresql.inventory.products
    inventory-connector-postgresql.inventory.products_on_hand
Events:  <none>
```

2. 验证连接器是否创建了 Kafka 主题：

- 通过 OpenShift Container Platform Web 控制台。

a. 导航到 Home → Search。

b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaTopic。

c. 在 KafkaTopics 列表中，点您要检查的主题名称，例如 inventory-connector-postgresql.inventory.orders---ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- 在终端窗口中：

a. 使用以下命令：

```
oc get kafkatopics
```

该命令返回类似以下示例的状态信息：

例 8.4. KafkaTopic 资源状态

```
NAME                                                 CLUSTER
PARTITIONS   REPLICATION FACTOR   READY
connect-cluster-configs                               debezium-kafka-cluster   1
1                  True
connect-cluster-offsets                               debezium-kafka-cluster   25
1                  True
connect-cluster-status                                debezium-kafka-cluster   5
1                  True
consumer-offsets---84e7a678d08f4bd226872e5cdd4eb527fadc1c6a
debezium-kafka-cluster   50           1                  True
inventory-connector-postgresql--a96f69b23d6118ff415f772679da623fbbb99421
debezium-kafka-cluster   1            1                  True
inventory-connector-postgresql.inventory.addresses---
```

```
1b6beaf7b2eb57d177d92be90ca2b210c9a56480          debezium-kafka-cluster
1          1                  True
inventory-connector-postgresql.inventory.customers---
9931e04ec92ecc0924f4406af3fdace7545c483b          debezium-kafka-cluster   1
1                  True
inventory-connector-postgresql.inventory.geom---
9f7e136091f071bf49ca59bf99e86c713ee58dd5          debezium-kafka-cluster
1          1                  True
inventory-connector-postgresql.inventory.orders---
ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d          debezium-kafka-cluster
1          1                  True
inventory-connector-postgresql.inventory.products---
df0746db116844cee2297fab611c21b56f82dcef          debezium-kafka-cluster   1
1                  True
inventory-connector-postgresql.inventory.products_on_hand---
8649e0f17ffcc9212e266e31a7aeea4585e5c6b5   debezium-kafka-cluster   1
1                  True
schema-changes.inventory                          debezium-kafka-cluster
1          1                  True
strimzi-store-topic---effb8e3e057afce1ecf67c3f5d8e4e3ff177fc55          debezium-
kafka-cluster   1          1                  True
strimzi-topic-operator-kstreams-topic-store-changelog---
b75e702040b99be8a9263134de3507fc0cc4017b  debezium-kafka-cluster  1   1
True
```

3. *检查主题内容。*

- *在终端窗口中输入以下命令：*

```
oc exec -n <project>  -it <kafka-cluster> -- /opt/kafka/bin/kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=<topic-name>
```

例如，

```
oc exec -n debezium  -it debezium-kafka-cluster-kafka-0 -- /opt/kafka/bin/kafka-
console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=inventory-connector-postgresql.inventory.products_on_hand
```

指定主题名称的格式与 oc describe 命令返回的格式与第 1 步中返回，例如 inventory-connector-postgresql.inventory.addresses。

对于主题中的每个事件，命令会返回类似以下示例的信息：

例 8.5. Debezium 更改事件的内容


```
{"schema":{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"inventory-
connector-postgresql.inventory.products_on_hand.Key"},"payload":{"product_id":101}}
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-postgresql.inventory.products_on_hand.Value","field":"before"},
{"type":"struct","fields":[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-
connector-postgresql.inventory.products_on_hand.Value","field":"after"},
{"type":"struct","fields":[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":
{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connecto
r.postgresql.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transacti
on"}],"optional":false,"name":"inventory-connector-
postgresql.inventory.products_on_hand.Envelope"},"payload":{"before":null,"after":
{"product_id":101,"quantity":3},"source":{"version":"2.3.4.Final-redhat-
00001","connector":"postgresql","name":"inventory-connector-
postgresql","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"tabl
e":"products_on_hand","server_id":0,"gtid":null,"file":"postgresql-
bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":1638985247805,"t
ransaction":null}}
```


在前面的示例中，有效负载 值显示连接器快照从表 **inventory.products_on_hand** 生成读取 **(op" ="r")事件。product_id** 记录的 **"before" 状态为 null**，表示该记录不存在之前的值。**"after" 状态对于 product_id** 为 **101** 的项目的 **quantity** 显示为 **3**。

8.6.5. Debezium PostgreSQL 连接器配置属性的描述

Debezium PostgreSQL 连接器有许多配置属性，可用于实现应用程序的正确连接器行为。许多属性

都有默认值。有关属性的信息组织如下：

- *所需的配置属性*
- *高级配置属性*
- *透传配置属性*

除非默认值可用， 否则需要以下配置属性。

表 8.27. 所需的连接器配置属性

属性	默认	描述
name	没有默认值	连接器的唯一名称。尝试使用相同的名称再次注册将失败。所有 Kafka Connect 连接器都需要此属性。
connector.class	没有默认值	连接器的 Java 类的名称。始终为 PostgreSQL 连接器使用 **io.debezium.connector.postgresql.PostgresConnector** 值。
tasks.max	**1**	应该为此连接器创建的最大任务数量。PostgreSQL 连接器始终使用单个任务，因此不使用这个值，因此默认值始终可以接受。
plugin.name	**decoderbufs**	在 PostgreSQL 服务器上安装的 PostgreSQL 逻辑解码插件的名称。 唯一支持的值是 **pgoutput**。您必须将 **plugin.name** 明确设置为 **pgoutput**。
slot.name	**Debezium**	为流传输特定数据库/schema 的特定插件创建的 PostgreSQL 逻辑解码插槽的名称。服务器使用此插槽将事件流传输到您要配置的 Debezium 连接器。 插槽名称必须符合 PostgreSQL 复制插槽命名规则，*其状态为 "每个复制插槽名称，其中可以包含小写字母、数字和下划线字符"。*

属性	默认	描述
slot.drop.on.stop	**false**	当连接器以安全、预期的方式停止时，是否删除逻辑复制插槽。默认行为是，当连接器停止时为连接器配置复制插槽。当连接器重启时，具有相同复制插槽可让连接器开始处理它的位置。 仅在测试或开发环境中设置为 **true**。丢弃插槽可让数据库丢弃 WAL 段。当连接器重启它执行新快照时，或者可以从 Kafka Connect offsets 主题中的持久偏移继续。
publication.name	**dbz_publication**	使用 **pgoutput** 时创建的用于流更改的 PostgreSQL 发布的名称。 如果尚未存在，则在启动时创建此发布，并且*包括所有表*。然后，Debezium 应用自己的 include/exclude 列表过滤（如果已配置），以限制发布以更改感兴趣的事件。连接器用户必须具有创建此出版物的超级用户权限，因此通常最好在第一次启动连接器前创建发布。 如果发布已存在，可以是所有表，或配置了表子集，Debebe 会使用发布，因为它被定义。
database.hostname	没有默认值	PostgreSQL 数据库服务器的 IP 地址或主机名。
database.port	**5432**	PostgreSQL 数据库服务器的整数端口号。
database.user	没有默认值	用于连接到 PostgreSQL 数据库服务器的 PostgreSQL 数据库用户的名称。
database.password	没有默认值	连接到 PostgreSQL 数据库服务器时要使用的密码。
database.dbname	没有默认值	从中流传输更改的 PostgreSQL 数据库的名称。

属性	默认	描述
topic.prefix	没有默认值	为特定 PostgreSQL 数据库服务器或集群提供命名空间的主题前缀，其中 Debezium 正在捕获更改。前缀应该在所有其他连接器中唯一，因为它被用作从这个连接器接收记录的所有 Kafka 主题的主题名称前缀。数据库服务器逻辑名称中只能使用字母数字字符、连字符、句点和下划线。 **警告** 不要更改此属性的值。如果您重启后更改了 name 值，而不是继续向原始主题发出事件，连接器会将后续事件发送到名称基于新值的主题。
schema.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，**与您要**捕获更改的模式名称匹配。不包括在 schema. **include.list 中的任何架构** 名称都会从捕获其更改中排除。默认情况下，所有非系统模式都会捕获其更改。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个标识符匹配；它与 schema 名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **schema.exclude.list** 属性。
schema.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与 **您不想** 捕获更改的模式名称匹配。任何名称不包含在 schema. **exclude.list 中的模式**，其更改会被捕获，但系统模式除外。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个标识符匹配；它与 schema 名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，请不要设置 **schema.include.list** 属性。

属性	默认	描述
table.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您要捕获的表的完全限定表标识符匹配。当设置此属性时，连接器只捕获指定表中的更改。每个标识符都是 *schemaName.tableName*。默认情况下，连接器在每个捕获了其更改的每个模式中捕获每个非系统表中的更改。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的完整标识符匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.exclude.list** 属性。
table.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您不想捕获的表的完全限定表标识符匹配。每个标识符都是 *schemaName.tableName*。当设置此属性时，连接器会捕获您未指定的每个表的更改。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的完整标识符匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，请不要设置 **table.include.list** 属性。
column.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与更改事件记录值中包含的列的完全限定域名匹配。列的完全限定域名格式为 *schemaName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，表达式用于匹配列的整个名称字符串；它与列中可能出现的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **column.exclude.list** 属性。
column.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与更改事件记录值中排除的列的完全限定域名匹配。列的完全限定域名格式为 *schemaName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，表达式用于匹配列的整个名称字符串；它与列中可能出现的子字符串不匹配。 如果您在配置中包含此属性，请不要设置 **column.include.list** 属性。

属性	默认	描述
skip.messages.without.change	**false**	指定在包含列中没有更改时是否跳过发布消息。如果列中没有包括每个 **column.include.list** 或 **column.exclude.list** 属性的列有变化，这将过滤消息。 注：仅在表的 REPLICA IDENTITY 设置为 FULL 时有效
time.precision.mode	**adaptive**	时间、日期和时间戳可以通过不同类型的精度来表示： **adaptive** 使用 millisecond、microsecond 或 nanosecond 精度值，根据数据库列的类型的类型来捕获日期、日期和时间戳。 **adaptive_time_microseconds** 使用 millisecond、microsecond 或 nanosecond 精度来捕获数据库中的日期、日期和时间戳值。一个例外是 **TIME** 类型字段，它总是被捕获为微秒。 **connect** 始终通过使用 Kafka Connect 的内置表示表示 **Time**,**Date**, 和 **Timestamp** 的值，无论数据库列的精度是什么。如需更多信息，请参阅 临时值。
decimal.handling.mode	**精确**	指定连接器应该如何处理 **DECIMAL** 和 **NUMERIC** 列的值： **precise** 代表使用 **java.math.BigDecimal** 来以二进制形式代表改变事件的值。 **double** 代表使用 **double** 值来代表值。它可能会降低一些精度，但更加容易使用。 **string** 以特定格式的字符串来对值进行编码。这容易使用，但其代表的真实类型的信息可能会丢失。如需更多信息，请参阅 Decimal type。
hstore.handling.mode	**map**	指定连接器应该如何处理 **hstore** 列的值： **map** 代表使用 **MAP**。 **json** 代表使用 **json 字符串** 代表值。此设置对格式的字符串进行编码，如 **{"key" : "val"}**。如需更多信息，请参阅 PostgreSQL **HSTORE** 类型。

属性	默认	描述
interval.handling.mode	**数字**	指定连接器如何处理 **interval** 列的值： **numeric** 代表使用大约微秒数的间隔。 **string** 代表间隔，使用 **P<years>Y<months>M<days>DT<hours>H<minutes>M<seconds>S** 代表。例如：**P1Y2M3DT4H5M6.78S**。如需更多信息，请参阅 PostgreSQL 基本类型。
database.sslmode	**prefer**	是否使用到 PostgreSQL 服务器的加密连接。options include: **disable** 使用未加密的连接。 **允许** 首先尝试使用未加密的连接，失败，失败（安全（加密）连接。 **首选** 尝试先使用安全（加密）连接，失败，未加密的连接失败。 **需要使用** 安全（加密）连接，如果出现以下情况，则会失败。 无法建立。 **verify-ca** 的行为与 **require** 一样，但也会根据配置的证书颁发机构(CA)证书验证服务器 TLS 证书。 如果没有找到有效的匹配 CA 证书， **verify-full** 的行为与 **verify-ca** 类似，但也验证服务器证书是否与连接器尝试连接的主机匹配。如需更多信息，请参阅 PostgreSQL 文档。
database.sslcert	没有默认值	包含客户端的 SSL 证书的文件的路径。如需更多信息，请参阅 PostgreSQL 文档。
database.sslkey	没有默认值	包含客户端的 SSL 私钥的文件的路径。如需更多信息，请参阅 PostgreSQL 文档。
database.sslpassword	没有默认值	从 **database.sslkey** 指定的文件访问客户端私钥的密码。如需更多信息，请参阅 PostgreSQL 文档。
database.sslrootcert	没有默认值	包含验证服务器的根证书的文件的路径。如需更多信息，请参阅 PostgreSQL 文档。
database.tcpKeepAlive	**true**	启用 TCP keep-alive 探测，以验证数据库连接是否仍然处于活动状态。如需更多信息，请参阅 PostgreSQL 文档。

属性	默认	描述
tombstones.on.delete	**true**	控制 *delete* 事件是否后跟一个 tombstone 事件。 **true** - 一个 delete 操作由 *delete* 事件和后续 tombstone 事件表示。 **false** - 仅有一个 *delete* 事件被抛出。 删除源记录后，发出 tombstone 事件（默认行为）可让 Kafka 在为主题启用了 日志 压缩时完全删除与已删除行键相关的所有事件。
column.truncate.to.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果在列中的数据超过了在属性名中的 *length* 指定的字符长度时删节数据，设置此属性。将 **length** 设置为正整数值，如 **column.truncate.to.20.chars**。 列的完全限定域名会观察以下格式：< ***schemaName> . <tableName> . &*** **lt;*columnName>***。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。
column.mask.with.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果您希望连接器屏蔽一组列的值，例如，如果它们包含敏感数据，则设置此属性。将 ***length*** 设置为一个正整数，替换在属性名称中的 *length* 指定的星号（*）的数量列中的数据。将 *length* 设置为 **0**（零）将指定列中的数据替换为空字符串。 列的完全限定域名观察以下格式：*schemaName.tableName.columnName*。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。

属性	默认	描述
column.mask.hash.*hashAlgorithm*.with.salt.*salt*; column.mask.hash.v2.*hashAlgorithm*.with.salt.*salt*	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。列的完全限定域名格式为 *<schemaName>.<tableName>.<columnName>*. 要匹配一个列的名称，Debezium 应用正则表达式，它由您指定为 *anchored* 正则表达式。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。在生成的更改事件记录中，指定列的值替换为 pseudonyms。 一个 pseudonym，它包括了通过应用指定的 *hashAlgorithm* 和 *salt* 的结果的哈希值。根据所使用的哈希函数，会维护引用完整性，而列值则替换为 pseudonyms。支持的哈希功能在 Java Cryptography 架构标准 Algorithm Name 文档的 MessageDigest 部分中 进行了描述。 在以下示例中，**CzQMA0cB5K** 是一个随机选择的 salt。 column.mask.hash.SHA-256.with.salt.CzQMA0cB5K = inventory.orders.customerName, inventory.shipment.customerName 如有必要，pseudonym 会自动缩短为列的长度。连接器配置可以包含多个属性，用于指定不同的哈希算法和 salt。 根据所用的 *hashAlgorithm* （选择 *salt* ）和实际数据集，生成的数据集可能无法完全屏蔽。 应该使用哈希策略版本 2 来确保在不同的位置或系统中对值进行哈希处理。

属性	默认	描述
column.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，与您希望连接器发出代表列元数据的额外参数的完全限定名称匹配。当设置此属性时，连接器会将以下字段添加到事件记录的 schema 中： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*databaseName.tableName.columnName*, 或 *databaseName.schemaName.tableName.columnName*. 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。

属性	默认	描述
datatype.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，用于指定为数据库中列定义的数据类型的完全限定名称。当设置此属性时，对于具有匹配数据类型的列，连接器会发出在 schema 中包含以下额外字段的事件记录： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名会观察以下格式之一：*databaseName.tableName.typeName*, 或 *databaseName.schemaName.tableName.typeName*. 要匹配数据类型的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据类型的整个名称字符串匹配；表达式与类型名称中可能存在的子字符串不匹配。 有关 PostgreSQL 特定数据类型名称的列表，请参阅 PostgreSQL 数据类型映射。

属性	默认	描述
message.key.columns	*空字符串*	指定连接器用来组成自定义消息键的表达式列表，用于更改它发布到指定表的 Kafka 主题的事件记录。 默认情况下，Debezium 使用表的主键列作为它发出的记录的消息键。在默认位置，或者为缺少主密钥的表指定一个键，您可以根据一个或多个列配置自定义消息密钥。 要为表建立自定义消息键，请列出表，后跟要用作消息键的列。每个列表条目都采用以下格式： *<fully-qualified_tableName> : <keyColumn> , <keyColumn>* To base a table key on multiple column name, 在列名称间插入逗号。 每个完全限定表名称都是以下格式的一个正则表达式： *<schemaName > . <tableName>* 属性可以包括多个表的条目。使用分号分隔列表中的表条目。 以下示例为表 **inventory.customers** 和 **purchase.orders** 设置消息键： **inventory.customers:pk1,pk2; (any).purchaseorders:pk3,pk4** for the table **inventory.customer**, 列 **pk1** 和 **pk2** 被指定为消息键。对于任何 **模式中的** 购买顺序表，列 **pk3** 和 **pk4** 服务器作为消息键。 对于用来创建自定义消息键的列数量没有限制。但是，最好使用指定唯一密钥所需的最小数量。 请注意，在表上将此属性设置并将 **REPLICA IDENTITY** 设置为 **DEFAULT** 时，如果键列不是表的主键的一部分，则会导致 tombstone 事件不会被正确创建。 将 **REPLICA IDENTITY** 设置为 **FULL** 是唯一解决方案。

属性	默认	描述
publication.autocreate.mode	*all_tables*	仅在使用 **pgoutput** 插件 流更改时应用。此设置决定了如何创建 发布。指定以下值之一： **all_tables** - 如果存在发布，则连接器会使用它。如果发布不存在，连接器会为数据库中捕获更改的所有表创建一个发布。要使连接器创建发布，它必须通过具有创建发布和执行复制权限的数据库用户帐户访问数据库。您可以使用以下 SQL 命令 **CREATE PUBLICATION <publication_name> FOR ALL TABLES;**. **disabled** - 连接器不会尝试创建出版物来授予所需的权限。数据库管理员或配置为执行复制的用户必须在运行连接器前创建发布。如果连接器无法找到发布，连接器会抛出异常并停止。 **过滤** - 如果一个发布存在，连接器会使用它。如果不存在发布，连接器会为表创建一个新的发布，该表与 **schema.include.list**, **schema.exclude.list**, and **table.include.list**, and **table.exclude.list** 配置属性指定的当前过滤器配置属性匹配。例如：**CREATE PUBLICATION <publication_name> FOR TABLE <tbl1, tbl2, tbl3>**。如果存在发布，连接器会更新与当前过滤器配置匹配的表的发布。例如：**ALTER PUBLICATION <publication_name> SET TABLE <tbl1, tbl2, tbl3>**。

属性	默认	描述
replica.identity.autoset.values	*空字符串*	此设置决定了表级别 的副本身份 的值。 这个选项将覆盖数据库中的现有值。以逗号分隔的正则表达式列表，与要在表格中使用的完全限定表和副本身份值匹配。 每个表达式必须与模式 '<fully-qualified table name>:<replica identity>' 匹配，其中表名称可以定义为 (**SCHEMA_NAME.TABLE_NAME**)，并且副本身份值为： **DEFAULT** - 记录主键列的旧值（若有）。这是非系统表的默认值。 **INDEX index_name** - 记录指定索引涵盖的列的旧值，它必须是唯一的，而不是部分，不能延迟，且仅包含标记为 NOT NULL 的列。如果丢弃此索引，则行为与 NOTHING 相同。 **FULL** - 记录行中所有列的旧值。 **NOTHING** - 记录不有关旧行的信息。这是系统表的默认设置。 例如， `schema1.*:FULL,schema2.table2:NOTHING,schema2.table3:INDEX idx_name`
binary.handling.mode	bytes	指定在更改事件中二进制(**bytea**)列应该代表： **bytes** 代表二进制数据作为字节数组。 **base64** 代表二进制数据作为 base64 编码的字符串。 **base64-url-safe** 代表二进制数据作为 base64-url-safe-encoded 字符串。 **hex** 代表二进制数据以十六进制编码(base16)字符串表示。

属性	默认	描述
schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列
field.name.adjustment.mode	none	指定应如何调整字段名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列 如需更多信息，请参阅 Avro 命名。
money.fraction.digits	**2**	指定在将 Postgres **money** 类型转换为 **java.math.BigDecimal**（它代表更改事件中的值）时应使用多少位的十进制数字。仅在将 **decimal.handling.mode** 设置为 exact 时**才适用**。
message.prefix.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您希望连接器要捕获的逻辑解码消息前缀的名称匹配。默认情况下，连接器捕获所有逻辑解码信息。当设置此属性时，连接器只捕获带有属性指定的前缀的逻辑解码消息。所有其他逻辑解码信息都不包括。 要匹配消息前缀的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与整个消息前缀字符串匹配；表达式与前缀中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **message.prefix.exclude.list** 属性。 有关 *消息* 事件结构及其排序语义的详情，请参考 *消息* 事件。

属性	默认	描述
message.prefix.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您不希望连接器捕获的逻辑解码消息前缀匹配。当设置此属性时，连接器不会捕获使用指定前缀的逻辑解码消息。所有其他消息都会被捕获。 要排除所有逻辑解码信息，请将此属性的值设置为 **adtrust**。 要匹配消息前缀的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与整个消息前缀字符串匹配；表达式与前缀中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **message.prefix.include.list** 属性。 有关 *消息* 事件结构及其排序语义的详情，请参考 *消息* 事件。

以下 高级配置 属性在大多数情况下可以正常工作，因此很少需要在连接器的配置中指定。

表 8.28. 高级连接器配置属性

属性	默认	描述

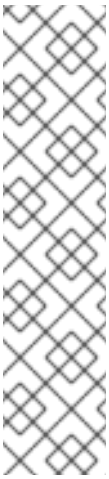
属性	默认	描述

converters	没有默认值	枚举连接器可以使用的 自定义转换器 实例的符号链接列表。例如，
		
		isbn
		
		您必须设置 **converters** 属性，使连接器能够使用自定义转换器。
		
		对于您为连接器配置的每个转换器，您还必须添加一个 **.type** 属性，它指定了实现转换器接口的类的完整名称。**.type** 属性使用以下格式：
		
		***<converterSymbolicName>*.type**
		
		例如，
		
		`isbn.type: io.debezium.test.IsbnConverter`
		
		如果要进一步控制配置的转换器的行为，您可以添加一个或多个配置参数将值传递给转换器。要将任何其他配置参数与转换器关联，请为参数名称加上转换器的符号名作为前缀。
		例如，
		
		`isbn.schema.name: io.debezium.postgresql.type.Isbn`

属性	默认	描述
snapshot.mode	**初始**	指定在连接器启动时执行快照的条件： **initial** - 连接器只有在没有为逻辑服务器名称记录偏移时才执行快照。 **always** - 在连接器每次启动时都执行快照。 **never** - 连接器永不执行快照。当以这种方式配置连接器时，其启动时的行为如下。如果 Kafka offsets 主题中存在之前存储的 LSN，则连接器将继续从该位置流更改。如果没有存储 LSN，则连接器会在服务器上创建 PostgreSQL 逻辑复制插槽时从时间点开始流更改。只有在您知道所有关注的数据仍然反映在 WAL 中时，**never** 快照模式很有用。 **initial_only** - 连接器会执行初始快照，然后停止，而无需处理任何后续更改。 **导出** - 弃用的 了解更多信息，请参阅 **snapshot.mode** 选项表。
snapshot.include.collection.list	**table.include.list**中指定的所有表	可选的、以逗号分隔的正则表达式列表，与要包含在快照中的表的完全限定名称（<***schemaName>.<tableName***>）匹配。指定的项目必须在连接器的 **table.include.list** 属性中命名。只有在连接器的 **snapshot.mode** 属性设置为除 **never** 的值时，此属性才会生效。 此属性不会影响增量快照的行为。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。
snapshot.lock.timeout.ms	**10000**	正整数值，指定执行快照时要等待的最大时间（以毫秒为单位）。如果连接器无法在这个时间间隔内获取表锁定，则快照会失败。连接器如何执行快照 提供详细信息。

属性	默认	描述
snapshot.select.statement.overrides	没有默认值	指定要包含在快照中的表行。如果您希望快照只包含表中的行的子集，请使用属性。此属性仅影响快照。它不适用于连接器从日志中读取的事件。 该属性包含以逗号分隔的、完全限定表名称列表，格式为 < ***schemaName>.<tableName>***。例如， **"snapshot.select.statement.overrides": "inventory.products,customers.orders"** 对于列表中的每个表，添加一个进一步的配置属性，用于指定连接器在获取快照时要在表上运行的 **SELECT** 语句。指定的 **SELECT** 语句决定了快照中包含的表行的子集。使用以下格式指定此 **SELECT** 语句属性的名称： **snapshot.select.statement.overrides.*<schemaName> . <tableName>***。例如，**snapshot.select.statement.overrides.customers.orders**。 Example: 在包含 soft-delete 列 **delete_flag** 的 **customers.orders** 表中，如果您希望快照只包含没有软删除的记录，请添加以下属性： "snapshot.select.statement.overrides" : "customer.orders", "snapshot.select.statement.overrides.customer.orders": "SELECT * FROM [customers].[orders] WHERE delete_flag = 0 ORDER BY id DESC" 在生成的快照中，连接器只包括 **delete_flag = 0** 的记录。

属性	默认	描述
event.processing.failure.handling.mode	fail	指定连接器在处理事件期间应如何响应异常： **失败** 传播异常，表示有问题的事件的偏移，并导致连接器停止。 **会警告** 记录有问题的事件的偏移，跳过该事件，并继续处理。 **跳过** 有问题的事件并继续处理。
max.batch.size	2048	正整数值，用于指定连接器处理的每个批处理的最大大小。
max.queue.size	8192	正整数值，用于指定阻塞队列可以保存的最大记录数。当 Debezium 从数据库读取事件时，它会将事件放置在阻塞队列中，然后再将它们写入 Kafka。阻塞队列可以提供从数据库读取更改事件时，连接器最快于将其写入 Kafka 的信息，或者在 Kafka 不可用时从数据库读取更改事件。当连接器定期记录偏移时，队列中保存的事件会被忽略。始终将 **max.queue.size** 的值设置为大于 **max.batch.size** 的值。
max.queue.size.in.bytes	0	一个长的整数值，用于指定阻塞队列的最大卷（以字节为单位）。默认情况下，不会为阻塞队列指定卷限制。要指定队列可以消耗的字节数，请将此属性设置为正长值。 如果还设置了 **max.queue.size**，当队列的大小达到任一属性指定的限制时，写入队列将被阻止。例如，如果您设置了 **max.queue.size=1000**、和 **max.queue.size.in.bytes=5000**，在队列包含 1000 个记录后，或者队列中记录的卷达到 5000 字节后，写入队列会被阻止。
poll.interval.ms	500	正整数值，指定连接器在开始处理批处理事件前应等待新更改事件数的毫秒数。默认值为 500 毫秒。

属性	默认	描述
include.unknown.datatypes	**false**	当连接器遇到数据类型未知的字段时，指定连接器行为。默认行为是，连接器从更改事件中省略字段并记录警告。 如果您希望更改事件包含字段的不透明二进制表示，请将此属性设置为 **true**。这可让消费者解码字段。您可以通过设置 **二进制处理模式** 属性来控制确切的表示。  **注意** 当 **include.unknown.datatypes** 设为 **true** 时，消费者会面临向后兼容性问题。仅可能仅在发行版本间更改数据库特定的二进制表示，但如果 Debezium 最终支持数据类型，则数据类型将在逻辑类型中发送下游，这需要用户调整。通常，当遇到不支持的数据类型时，创建一个功能请求，以便可以添加支持。
database.initial.statements	没有默认值	分号分隔的 SQL 语句列表，连接器在建立 JDBC 与数据库的连接时执行。要将分号用作字符而不是分隔符，请指定两个连续的分号 **；；**。 连接器可以自行决定建立 JDBC 连接。因此，此属性仅适用于配置会话参数，而不适用于执行 DML 语句。 当连接器创建用于读取事务日志的连接时，不会执行这些语句。
status.update.interval.ms	**10000**	向服务器发送复制连接状态更新的频率，以毫秒为单位。 属性还控制检查数据库状态在关闭数据库时检测死连接的频率。

属性	默认	描述
heartbeat.interval.ms	0	控制连接器将心跳信息发送到 Kafka 主题的频率。默认行为是连接器不会发送心跳信息。 心跳消息可用于监控连接器是否从数据库接收更改事件。心跳消息有助于减少在连接器重启时需要重新更改事件的数量。要发送心跳消息，请将此属性设置为正整数，这代表心跳消息之间的毫秒数。 当数据库中有多个更新被跟踪时，需要心跳消息，但只有少量更新与连接器捕获更改的表和 schema 相关。在这种情况下，连接器照常从数据库事务日志中读取，但很少向 Kafka 发出更改记录。这意味着，没有将偏移更新提交到 Kafka，连接器没有将最新检索到的 LSN 发送到数据库的机会。数据库保留 WAL 文件，其中包含已经由连接器处理的事件。发送心跳消息可让连接器将最新检索到的 LSN 发送到数据库，这使得数据库能够回收不再需要 WAL 文件所使用的磁盘空间。
heartbeat.action.query	没有默认值	指定连接器发送心跳消息的查询，连接器在源数据库上执行。 这可用于解决 WAL 磁盘空间消耗 中描述的情况，其中从与高流量数据库在同一主机上的低流量数据库捕获更改可防止 Debezium 处理 WAL 记录，从而防止 Debezium 处理 WAL 记录，从而处理数据库。要解决这种情况，请在低流量数据库中创建一个心跳表，并将此属性设置为将记录插入到该表中，例如： **INSERT INTO test_heartbeat_table (text) VALUES ('test_heartbeat')** 这允许连接器从低流量数据库中接收更改，并确认其 LSN，这可防止数据库主机上的未绑定 WAL 增长。

属性	默认	描述
schema.refresh.mode	**columns_diff**	指定为表触发内存模式刷新的条件。 column_**diff** 是安全模式。它确保内存模式始终与数据库表的同步。 **columns_diff_exclude_unchanged_toast** 会指示连接器刷新内存模式缓存（如果从传入消息派生的模式），除非有未更改的 TOASTable 数据完全帐户用于差异。 如果经常更新的表有 TOASTed 数据，则此设置可能会显著提高连接器性能。但是，如果从表中丢弃了 TOASTable 列，则内存中模式可能会过时。
snapshot.delay.ms	没有默认值	连接器在连接器启动时执行快照前应等待的时间（以毫秒为单位）。如果您在集群中启动多个连接器，则此属性可用于避免快照中断，这可能会导致连接器的重新平衡。
snapshot.fetch.size	**10240**	在快照期间，连接器以每行的批处理读取表内容。此属性指定批处理中的最大行数。
slot.stream.params	没有默认值	分号分隔的参数列表，以传递给配置的逻辑解码插件。例如：**add-tables=public.table,public.table2;include-lsn=true**。
slot.max.retries	**6**	如果连接到复制插槽失败，这是连续尝试的最大尝试数。
slot.retry.delay.ms	**10000** (10 秒)	当连接器无法连接到复制插槽时，重试尝试之间等待的毫秒数。
unavailable.value.placeholder	**__debezium_unavailable_value**	指定连接器提供的常量，以指示原始值是不由数据库提供的粘贴值。如果 **unavailable.value.placeholder** 的设置以 **hex:** 前缀开头，则预期字符串的其余部分代表十六进制编码的 octets。如需更多信息，请参阅粘贴 值。
provide.transaction.metadata	**false**	确定连接器是否生成带有事务边界的事件，并使用事务元数据增强更改事件信。如果您希望连接器进行此操作，请指定 **true**。如需更多信息，请参阅 事务元数据。

属性	默认	描述
flush.lsn.source	**true**	决定连接器是否应该提交源 postgres 数据库中已处理记录的 LSN，以便可以删除 WAL 日志。如果您不希望连接器进行此操作，请指定 **false**。请注意，如果设置为 **false** LSN 不会被 Debezium 确认，因此 WAL 日志不会被清除，从而导致磁盘空间问题。用户应该处理 Debezium 之外的 LSN 的确认。
retriable.restart.connector.wait.ms	10000 (10 秒)	在发生可检索错误后重启连接器前等待的毫秒数。
skipped.operations	**t**	在流过程中将跳过的操作类型的逗号分隔列表。操作包括：用于 inserts/create、**u** 表示更新、**d** 表示 delete、**t** 表示 truncates，**none** 不跳过任何操作。**默认情况下会跳过 truncate 操作。**
signal.data.collection	没有默认值	用于向连接器发送信号的数据收集的完全限定名称。 使用以下格式指定集合名称： ***<schemaName> . < tableName>***
signal.enabled.channels	source	为连接器启用的信号频道名称列表。默认情况下，以下频道可用： • **source** • **kafka** • **file** • **jmx**
notification.enabled.channels	没有默认值	为连接器启用的通知频道名称列表。默认情况下，以下频道可用： • **sink** • **log** • **jmx**
incremental.snapshot.chunk.size	1024	连接器在增量快照块期间获取并读取内存的最大行数。增加块大小可提高效率，因为快照会运行更多大小的快照查询。但是，较大的块大小还需要更多内存来缓冲快照数据。将块大小调整为提供环境中最佳性能的值。

属性	默认	描述
xmin.fetch.interval.ms	**0**	XMIN 将从复制插槽读取的频率，以毫秒为单位。XMIN 值提供从其中开始新复制插槽的低限。默认值 **0** 禁用跟踪 XMIN 跟踪。
topic.naming.strategy	**io.debezium.schema.SchemaTopicNamingStrategy**	应该用来确定数据更改、模式更改、事务、心跳事件等的 TopicNamingStrategy 类的名称，默认为 **SchemaTopicNamingStrategy**。
topic.delimiter	**.**	指定主题名称的分隔符，默认为 **.**。
topic.cache.size	**10000**	在绑定的并发哈希映射中用于保存主题名称的大小。此缓存将有助于确定与给定数据收集对应的主题名称。
topic.heartbeat.prefix	**__debezium-heartbeat**	控制连接器向其发送心跳信息的主题名称。主题名称具有此模式： *topic.heartbeat.prefix.topic.prefix* 例如，如果主题前缀是 **fulfillment**，则默认主题名称为 **__debezium-heartbeat.fulfillment**。
topic.transaction	**Transactions**	控制连接器向其发送事务元数据消息的主题名称。主题名称具有此模式： *topic.prefix.topic.transaction* 例如，如果主题前缀是 **fulfillment**，默认的主题名称为 **fulfillment.transaction**。

属性	默认	描述
snapshot.max.threads	1	指定连接器执行初始快照时使用的线程数量。要启用并行初始快照，请将属性设置为大于 1 的值。在并行初始快照中，连接器会同时处理多个表。  **重要** 并行初始快照只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。
errors.max.retries	-1	在失败前，retriable 错误（如连接错误）的最大重试次数(-1 = no limit, 0 = disabled, > 0 = num of retries)。

透传连接器配置属性

连接器还支持创建 Kafka 生成者和消费者时使用的 直通 配置属性。

确保参考 Kafka 文档了解 Kafka 生成者和消费者的所有配置属性。PostgreSQL 连接器 使用新的消费者配置属性。

Debezium 连接器 Kafka 信号配置属性

Debezium 提供了一组 signal. 属性，用于控制连接器如何与 Kafka 信号主题进行交互。*

下表描述了 Kafka 信号 属性。

表 8.29. Kafka 信号配置属性

属性	默认	描述

属性	默认	描述
signal.kafka.topic	<topic.prefix>-signal	连接器监控用于临时信号的 Kafka 主题的名称。 **注意** 如果禁用了 自动主题创建，您必须手动创建所需的信号主题。需要信号主题来保留信号排序。信号主题必须具有单个分区。
signal.kafka.groupId	kafka-signal	Kafka 用户使用的组 ID 的名称。
signal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。每个对都引用 Debezium Kafka Connect 进程使用的 Kafka 集群。
signal.kafka.poll.timeout.ms	100	一个整数值，用于指定连接器在轮询信号时等待的最大毫秒数。

Debezium 连接器传递信号 Kafka 使用者客户端配置属性

Debezium 连接器为信号 Kafka 使用者提供直通配置。透传信号属性以 signals.consumer.* 前缀开始。例如，连接器将 signal.consumer.security.protocol=SSL 等属性传递给 Kafka 消费者。

Debezium 从属性中剥离前缀，然后再将属性传递给 Kafka 信号消费者。

Debezium 连接器接收器通知配置属性

下表描述了 通知 属性。

表 8.30. sink 通知配置属性

属性	默认	描述
notification.sink.topic.name	没有默认值	从 Debezium 接收通知的主题名称。当您将 notification.enabled.channels 属性配置为将 **sink** 作为启用的通知频道之一时，需要此属性。

8.7. 监控 DEBEZIUM POSTGRESQL 连接器性能

Debezium PostgreSQL 连接器提供了两种类型的指标，除了对 Zookeeper、Kafka 和 Kafka Connect 提供的 JMX 指标的内置支持之外。

- *快照指标 提供有关执行快照时连接器操作的信息。*
- *流指标 在连接器捕获更改和流更改事件记录时提供有关连接器操作的信息。*

Debezium 监控文档 提供了如何使用 JMX 公开这些指标的详细信息。

8.7.1. 在 PostgreSQL 数据库快照过程中监控 Debezium

MBean 是 debezium.postgres:type=connector-metrics,context=snapshot,server=<topic.prefix>。

快照指标不会公开，除非快照操作处于活跃状态，或者快照自上次连接器启动以来发生。

下表列出了可用的 shapshot 指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个快照事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上次启动或重置后看到的事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在快照和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	队列的空闲容量，用于在快照和主 Kafka Connect 循环之间传递事件。

属性	类型	描述
TotalTableCount	int	包括在快照中的表的总数。
RemainingTableCount	int	快照必须复制的表数。
SnapshotRunning	布尔值	快照是否已启动。
SnapshotPaused	布尔值	快照是否已暂停。
SnapshotAborted	布尔值	快照是否中止。
SnapshotCompleted	布尔值	快照是否完成。
SnapshotDurationInSeconds	long	快照为止所花费的秒数，即使未完成也是如此。也包括快照暂停的时间。
SnapshotPausedDurationInSeconds	long	快照暂停的秒数。如果快照暂停几次，暂停的时间会添加。
RowsScanned	Map<String, Long>	包含快照中每个表的行数的映射。表会在处理过程中逐步添加到映射中。更新每个 10,000 行扫描并在完成表后。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

连接器还在执行增量快照时提供以下额外快照指标：

属性	类型	描述
ChunkId	字符串	当前快照块的标识符。
ChunkFrom	字符串	定义当前块的主密钥集的下限。
ChunkTo	字符串	定义当前块的主密钥集的上限。

属性	类型	描述
TableFrom	字符串	当前快照表的主密钥集的下限。
TableTo	字符串	当前快照表的主密钥集的上限。

8.7.2. 监控 Debezium PostgreSQL 连接器记录流

MBean 是 debezium.postgres:type=connector-metrics,context=streaming,server=<topic.prefix>。

下表列出了可用的流指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个流事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上一次启动或指标重置以来看到的事件总数。
TotalNumberOfCreateEventsSeen	long	此连接器自上次启动或指标重置以来看到的创建事件总数。
TotalNumberOfUpdateEventsSeen	long	此连接器自上次启动或指标重置以来看到的更新事件总数。
TotalNumberOfDeleteEventsSeen	long	此连接器自上次启动或指标重置以来看到的删除事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的长度。

属性	类型	描述
QueueRemainingCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的队列的可用容量。
Connected	布尔值	表示连接器目前是否连接到数据库服务器的标记。
MilliSecondsBehindSource	long	最后一次更改事件时间戳和连接器处理它之间的毫秒数。这些值将讨论运行数据库服务器和连接器的计算机上时钟之间的任何区别。
NumberOfCommittedTransactions	long	已提交的已处理事务的数量。
SourceEventPosition	Map<String, String>	最后收到的事件的协调。
LastTransactionId	字符串	最后处理事务的事务的事务标识符。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

8.8. DEBEZIUM POSTGRESQL 连接器如何处理错误和问题

Debezium 是一个分布式系统，用于捕获多个上游数据库中的所有更改，它不会丢失或丢失事件。当系统正常运行或谨慎管理时，Debezium 会 精确 发送每个更改事件记录。

如果出现错误，则系统不会丢失任何事件。但是，当它从错误中恢复时，可能会重复一些更改事件。在这些异常情况下，Debezium （如 Kafka）在发送更改事件时 至少 提供。

以下部分详情：

配置和启动错误

在以下情况下，连接器在尝试启动时失败，在日志中报告错误/exception，并停止运行：

- 连接器的配置无效。
- 连接器无法使用指定的连接参数成功连接到 PostgreSQL。
- 连接器从 PostgreSQL WAL （使用 LSN）中的之前记录的位置重启，PostgreSQL 不再有那个历史记录可用。

在这些情况下，错误消息包含有关问题的详情，并可能会有推荐的临时解决方案。更正配置或解决 PostgreSQL 问题后，重启连接器。

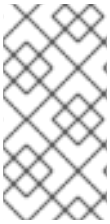
PostgreSQL 变得不可用

当连接器运行时，连接到的 PostgreSQL 服务器可能会因任何原因而不可用。如果发生这种情况，连接器会失败并显示错误并停止。当服务器再次可用时，重启连接器。

外部 PostgreSQL 连接器以 PostgreSQL LSN 的形式存储最后一个处理偏移。连接器重启并连接到服务器实例后，连接器与服务器通信，以继续来自该特定偏移的流。只要 Debezium 复制插槽保持不变，就可以使用这个偏移。永远不会丢弃主服务器上的复制插槽，否则您将丢失数据。有关删除插槽的故障情况的详情，请参考下一节。

集群故障

从版本 12 开始，PostgreSQL 只允许在主服务器上进行 逻辑复制插槽。这意味着，您可以将 Debezium PostgreSQL 连接器指向数据库集群的活跃主服务器。另外，复制插槽本身不会传播到副本。如果主服务器停机，则必须提升新的主服务器。



注意

有些受管 PostgresSQL 服务（如 AWS RDS 和 GCP CloudSQL）通过磁盘复制实施到备用的复制。这意味着复制插槽会被复制，并在故障转移后仍然可用。

新主必须具有一个复制插槽，供 pgoutput 插件配置，以及您要捕获更改的数据库。然后，您只能将连接器指向新的服务器并重启连接器。

发生故障切换时需要注意，您应该暂停 Debezium，直到您验证您有一个没有丢失数据的复制插槽。故障转移后：

- 在允许应用程序写入 新 主前，必须有一个创建 Debezium 复制插槽的进程。这至关重要。如果没有此过程，您的应用程序可能会错过更改事件。
- 您可能需要验证 Debezium 是否能够 在旧主失败前读取插槽中的所有更改。

一种可靠的恢复和验证任何更改是否已丢失的方法是在失败前立即恢复故障主点的备份。虽然这可能会造成管理困难，但允许您检查任何未消耗的更改的复制插槽。

Kafka Connect 进程正常停止

假设 Kafka Connect 以分布式模式运行，并且安全停止 Kafka Connect 进程。在关闭此过程前，Kafka Connect 会将进程的连接器任务迁移到该组中的另一个 Kafka Connect 进程。新的连接器任务会完全开始处理之前任务停止的位置。在处理连接器任务时，在新进程中安全停止并重新启动时会有一个短暂的延迟。

Kafka Connect 进程崩溃

如果 Kafka Connector 进程意外停止，则运行的任何连接器任务都会终止，而不记录他们最近处理的偏移量。当 Kafka Connect 以分布式模式运行时，Kafka Connect 会在其他进程中重启这些连接器任务。但是，PostgreSQL 连接器从之前进程 记录 的最后一个偏移中恢复。这意味着新的替换任务可能会生成一些在崩溃前处理的相同更改事件。重复事件的数量取决于偏移刷新周期和数据卷在崩溃前更改。

因为在从故障恢复过程中可能会重复一些事件，所以消费者应始终预测一些重复的事件。Debezium 更改是幂等的，因此一系列事件始终产生相同的状态。

在每个更改事件记录中，Debezium 连接器会插入有关事件来源的源特定信息，包括 PostgreSQL 服务器的时间、服务器事务的 ID 以及写入事务更改的位置。消费者可以跟踪这些信息，特别是 LSN，以确定事件是否重复。

Kafka 变得不可用

当连接器生成更改事件时，Kafka Connect 框架使用 Kafka producer API 在 Kafka 中记录这些事件。定期在 Kafka Connect 配置中指定的频率，Kafka Connect 会记录这些更改事件中显示的最新偏移。如果 Kafka 代理不可用，则运行连接器的 Kafka Connect 进程会重复尝试重新连接到 Kafka 代理。换句话说，连接器任务会暂停，直到可以重新建立连接，此时连接器会完全恢复它们关闭的位置。

在持续时间内停止连接器

如果连接器被安全停止，则可以继续使用数据库。在 PostgreSQL WAL 中记录任何更改。当连接器重启时，它会恢复流更改。也就是说，它会为连接器停止期间创建的所有数据库更改生成更改事件记录。

正确配置的 Kafka 集群可以处理大量吞吐量。Kafka Connect 根据 Kafka 最佳实践编写，并为 Kafka Connect 连接器有足够的资源处理大量数据库更改事件。因此，在停止一段时间后，当 Debezium 连接器重启时，可能会捕获在停止时进行的数据库更改。发生这种情况的时间取决于 Kafka 的功能和性能以及 PostgreSQL 中数据更改的卷。

第 9 章 SQL SERVER 的 DEBEZIUM 连接器

Debezium SQL Server 连接器捕获 SQL Server 数据库模式中发生的行级更改。

有关与此连接器兼容的 SQL Server 版本的详情，请查看 Debezium 支持的配置 页面。

有关 Debezium SQL Server 连接器及其使用的详情，请查看以下主题：

- 第 9.1 节 “Debezium SQL Server 连接器概述”
- 第 9.2 节 “Debezium SQL Server 连接器如何工作”
- 第 9.2.10 节 “Debezium SQL Server 连接器数据更改事件的描述”
- 第 9.2.12 节 “Debezium SQL Server 连接器如何映射数据类型”
- 第 9.3 节 “设置 SQL Server 以运行 Debezium 连接器”
- 第 9.4 节 “部署 Debezium SQL Server 连接器”
- 第 9.5 节 “在 schema 更改后刷新捕获表”
- 第 9.6 节 “监控 Debezium SQL Server 连接器性能”

Debezium SQL Server 连接器第一次连接到 SQL Server 数据库或集群时，它会获取数据库中模式的一致性快照。初始快照完成后，连接器会持续捕获 INSERT、UPDATE 或 DELETE 操作的行级更改，这些操作提交到为 CDC 启用的 SQL Server 数据库。连接器为每个数据更改操作生成事件，并将其流传输到 Kafka 主题。连接器将表的所有事件流传输到专用 Kafka 主题。然后，应用程序和服务可以使用该主题中的数据更改事件记录。

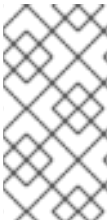
9.1. DEBEZIUM SQL SERVER 连接器概述

Debezium SQL Server 连接器基于 SQL Server 2016 Service Pack 1 (SP1)及更高版本中 提供的 更改数据捕获 功能。SQL Server 捕获进程监控指定的数据库和表，并将更改存储在特定创建的 change tables 中。

要启用 Debezium SQL Server 连接器捕获数据库操作的更改事件记录，您必须首先在 SQL Server 数据库上启用更改数据捕获。在要捕获的每个表上必须同时启用 CDC。在源数据库上设置 CDC 后，连接器可以捕获数据库中发生的行级 INSERT、UPDATE、和 DELETE 操作。连接器将每个源表的事件记录写入 Kafka 主题，特别是专用于该表。每个捕获的表都有一个主题。客户端应用程序读取它们遵循的数据库表的 Kafka 主题，并可以响应它们从这些主题中使用的行级事件。

当连接器第一次连接到 SQL Server 数据库或集群时，它会为其配置的所有表获取一致的模式快照，并将其配置为捕获更改，并将这个状态流传输到 Kafka。快照完成后，连接器会持续捕获后续的行级更改。通过首先建立所有数据的一致性视图，连接器可以继续读取，而不会丢失快照发生时所做的任何更改。

Debezium SQL Server 连接器可以接受故障。当连接器读取更改并生成事件时，它会定期记录数据库日志中事件的位置(LSN / Log Sequence Number)。如果连接器因任何原因（包括通信失败、网络问题或崩溃）停止，重启连接器会从它读取的最后一个点恢复 SQL Server CDC 表。



注意

偏移定期提交。它们不会在发生更改事件时提交。因此，在中断后，可能会生成重复的事件。

容错也适用于快照。也就是说，如果连接器在快照过程中停止，连接器会在重启时启动新快照。

9.2. DEBEZIUM SQL SERVER 连接器如何工作

为了优化配置和运行 Debezium SQL Server 连接器，了解连接器如何执行快照、流更改事件、决定 Kafka 主题名称并使用元数据非常有用。

有关连接器如何工作的详情，请查看以下部分：

- 第 9.2.1 节 “Debezium SQL Server 连接器如何执行数据库快照”
- 第 9.2.2 节 “临时快照”

- 第 9.2.3 节 “增量快照”
- 第 9.2.4 节 “Debezium SQL Server 连接器如何读取更改数据表”
- 第 9.2.7 节 “接收 Debezium SQL Server 更改事件的 Kafka 主题的默认名称”
- 第 9.2.9 节 “Debezium SQL Server 连接器如何使用 schema 更改主题”
- 第 9.2.10 节 “Debezium SQL Server 连接器数据更改事件的描述”
- 第 9.2.11 节 “Debezium SQL Server 连接器生成的事件代表事务边界”

9.2.1. Debezium SQL Server 连接器如何执行数据库快照

SQL Server CDC 的设计不是存储数据库更改的完整历史记录。对于 Debezium SQL Server 连接器，为数据库的当前状态建立基准，它使用名为 snapshotting 的进程。初始快照捕获数据库中表的结构和数据。

您可以在以下部分找到有关快照的更多信息：

- 第 9.2.2 节 “临时快照”
- 第 9.2.3 节 “增量快照”

Debezium SQL Server 连接器用来执行初始快照的默认工作流

以下工作流列出了 Debezium 创建快照所采取的步骤。这些步骤描述了当 snapshot.mode 配置属性设置为其默认值时（即 的初始 ）时快照的流程。您可以通过更改 snapshot.mode 属性的值来自定义连接器创建快照的方式。如果您配置不同的快照模式，连接器使用这个工作流的修改版本完成快照。

1. 建立与数据库的连接。
2.

确定要捕获的表。默认情况下，连接器捕获所有非系统表。要让连接器捕获表或表元素的子集，您可以设置多个 include 和 exclude 属性来过滤数据，如 table.include.list 或 table.exclude.list。

3. 在启用了 CDC 的 SQL Server 表上获得锁定，以防止在创建快照过程中发生结构更改。锁定的级别由 snapshot.isolation.mode 配置属性决定。

4. 在服务器的事务日志中读取最大日志序列号(LSN)位置。

5. 捕获所有非系统的结构，或者为捕获指定的所有表。连接器在其内部数据库模式历史记录主题中保留此信息。架构历史记录提供有关发生更改事件时生效的结构的信息。

注意

默认情况下，连接器捕获数据库中每个表的模式，这些模式处于捕获模式，包括没有配置为捕获的表。如果没有为捕获配置表，则初始快照只捕获其结构；它不会捕获任何表数据。有关为什么没有包括在初始快照中的表的快照保留模式信息，请参阅 了解为什么初始快照捕获所有表的 schema。

6. 如有必要，释放在第 3 步中获得的锁定。其他数据库客户端现在可以写入任何之前锁定的表。

7. 在 LSN 分步读取时，连接器会扫描要捕获的表。在扫描过程中，连接器完成以下任务：

   a. 确认表已在快照开始前创建。如果表是在快照启动后创建的，连接器会跳过表。快照完成后，连接器过渡到 streaming，它会发出快照开始后创建的任何表的更改事件。

   b. 为从表获取的每行生成 读取 事件。所有 读取 事件都包含相同的 LSN 位置，这是在第 4 步中获取的 LSN 位置。

   c. 将每个 读取 事件发送到表的 Kafka 主题。

8. 在连接器偏移中记录快照成功完成。

生成的初始快照捕获了为 CDC 启用的表中每行的当前状态。在这个基准状态中，连接器会捕获后续更改。

在快照进程开始后，如果进程因为连接器失败、重新平衡或其他原因而中断，则进程会在连接器重启后重启。

连接器完成初始快照后，它会继续从在第 4 步中读取的位置进行流，使其不会错过任何更新。

如果连接器因为任何原因而再次停止，它会在重启后从之前关闭的位置恢复流更改。

9.2.1.1. 初始快照捕获所有表的 schema 历史记录的描述

连接器运行的初始快照捕获两种类型的信息：

表数据

在连接器的 table.include.list 属性中命名的表中的 INSERT、UPDATE 和 DELETE 操作的信息。

模式数据

描述应用到表的结构更改的 DDL 语句。模式数据会保留给内部模式历史记录主题，以及连接器的 schema 更改主题（如果配置了）。

运行初始快照后，您可能会注意到快照捕获没有指定用于捕获的表的模式信息。默认情况下，初始快照旨在捕获数据库中存在的每个表的模式信息，而不仅仅是从指定为捕获的表的表。连接器要求表的模式存在于架构历史记录主题中，然后才能捕获表。通过启用初始快照来捕获不是原始捕获集一部分的表的 schema 数据，Debebe 准备好连接器，以便稍后需要捕获这些表中的事件数据。如果初始快照没有捕获表的 schema，您必须将模式添加到历史记录主题，然后才能从表中捕获数据。

在某些情况下，您可能想要限制初始快照中的模式捕获。当您要减少完成快照所需的时间时，这非常有用。或者，当 Debezium 通过可访问多个逻辑数据库的用户帐户连接到数据库实例时，但您希望连接器只从特定逻辑数据库中的表捕获更改。

附加信息

- 从不是由初始快照捕获的表捕获数据（没有模式更改）

- 从不是由初始快照捕获的表捕获数据（应用程序更改）

- 设置 schema.history.internal.store.only.captured.tables.ddl 属性，以指定从中捕获模式信息的表。

- 设置 schema.history.internal.store.only.captured.databases.ddl 属性，以指定从中捕获模式更改的逻辑数据库。

9.2.1.2. 从不是由初始快照捕获的表捕获数据（没有模式更改）

在某些情况下，您可能希望连接器从其模式未被初始快照捕获的表中捕获数据。根据连接器配置，初始快照只能捕获数据库中特定表的表模式。如果历史记录主题中没有表模式，连接器将无法捕获表，并报告缺少的 schema 错误。

您可能仍然能够从表中捕获数据，但您必须执行额外的步骤来添加表模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。

- 没有模式更改应用于连接器读取的 LSN 和最新更改表条目之间的表。有关从具有结构性更改的新表中捕获数据的详情，请参考 第 3.2.1.3 节 “从不是由初始快照捕获的表捕获数据（应用程序更改）”。

流程

1. 停止连接器。

2. 删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。

3. 清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 对连接器配置应用以下更改：

a. （可选）将 schema.history.internal.captured.tables.ddl 的值设置为 false。此设置会导致快照捕获所有表的 schema，并保证以后可以重建所有表的 schema 历史记录。

注意

捕获所有表的架构的快照需要更多时间来完成。

b. 添加您希望连接器捕获至 table.include.list 的表。

c. 将 snapshot.mode 设置为以下值之一：

初始

重启连接器时，它会获取捕获表数据和表结构的数据库的完整快照。如果您选择这个选项，请考虑将 schema.history.internal.captured.tables.ddl 属性的值设置为 false，以便连接器捕获所有表的 schema。

schema_only

重启连接器时，它会获取仅捕获表模式的快照。与完整数据快照不同，这个选项不会捕获任何表数据。如果您要比使用完整快照更快地重启连接器，请使用这个选项。

5. 重启连接器。连接器完成 snapshot.mode 指定的快照类型。

6. （可选）如果连接器执行了 schema_only 快照，在快照完成后，启动一个增量快照来 从您添加的表中捕获数据。连接器在继续从表中实时更改时运行快照。运行增量快照可捕获以下数据

更改：

- 对于之前捕获的连接器的表，增量 snapsot 捕获连接器停机时所发生的变化，即在连接器停止和当前重启之间的时间间隔。
- 对于新添加的表，增量快照会捕获所有现有表行。

9.2.1.3. 从不是由初始快照捕获的表捕获数据（应用程序更改）

如果架构更改应用到表，则在架构更改前提交的记录与更改后提交的不同结构不同。当 Debezium 从表中捕获数据时，它会读取 schema 历史记录，以确保它为每个事件应用正确的模式。如果 schema 历史记录主题中没有 schema，则连接器无法捕获表，并出现错误结果。

如果要从初始快照捕获的表中捕获数据，并且修改了表的 schema，则必须将模式添加到历史记录主题中（如果它还没有可用）。您可以通过运行新的模式快照或运行表的初始快照来添加模式。

前提条件

- 您希望从带有连接器在初始快照期间没有捕获的 schema 捕获数据。
- 架构更改应用于表，以便捕获的记录没有统一结构。

流程

初始快照捕获了所有表的模式(storage.only.captured.tables.ddl 设置为 false)

1. 编辑 table.include.list 属性，以指定您要捕获的表。
2. 重启连接器。
3. 如果要从新添加的表中捕获现有数据，则启动 增量快照。

初始快照没有捕获所有表的模式(storage.only.captured.tables.ddl 设置为 true)

如果初始快照没有保存您要捕获的表的模式，请完成以下步骤之一：

流程 1：架构快照，后跟增量快照

在此过程中，连接器首先执行 schema 快照。然后，您可以启动增量快照，使连接器能够同步数据。

1. 停止连接器。
2. 删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。
3. 清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 为连接器配置中的属性设置值，如以下步骤所述：
   a. 将 snapshot.mode 属性的值设置为 schema_only。
   b. 编辑 table.include.list 以添加您要捕获的表。
5. 重启连接器。
6. 等待 Debezium 捕获新表和现有表的模式。在连接器停止后发生任何表的数据更改不会被捕获。
7. 为确保没有丢失数据，请启动 增量快照。

步骤 2：初始快照，后跟可选的增量快照

在此过程中，连接器执行数据库的完整初始快照。与任何初始快照一样，在具有多个大型表的数据库中，运行初始快照可能会非常耗时。快照完成后，您可以选择触发增量快照来捕获连接器离线时发生的任何更改。

1. 停止连接器。
2. 删除由 schema.history.internal. kafka.topic 属性指定的内部数据库架构历史记录 主题。
3. 清除配置的 Kafka Connect offset.storage.topic 中的偏移量。有关如何删除偏移的更多信息，请参阅 Debezium 社区常见问题解答。

警告

删除偏移应仅由具有操作内部 Kafka Connect 数据经验的高级用户执行。此操作可能具有破坏性，应仅作为最后的手段来执行。

4. 编辑 table.include.list 以添加您要捕获的表。
5. 为连接器配置中的属性设置值，如以下步骤所述：
   a. 将 snapshot.mode 属性的值设置为 initial。
   b. （可选）将 schema.history.internal.store.only.captured.tables.ddl 设置为 false。
6. 重启连接器。连接器获取完整的数据库快照。快照完成后，连接器会过渡到 streaming。
7.

（可选）要捕获连接器离线时更改的任何数据，请启动 增量快照。

9.2.2. 临时快照

默认情况下，连接器仅在首次启动后运行初始快照操作。在正常情况下，在这个初始快照后，连接器不会重复快照过程。连接器捕获的任何更改事件数据都只通过流处理。

然而，在某些情况下，连接器在初始快照期间获得的数据可能会过时、丢失或不完整。为了提供总结表数据的机制，Debezium 包含一个执行临时快照的选项。数据库中的以下更改可能会导致执行临时快照：

- 连接器配置会被修改为捕获不同的表集合。
- Kafka 主题已删除，必须重建。
- 由于配置错误或某些其他问题导致数据损坏。

您可以通过启动所谓的 临时快照来为之前捕获的表重新运行快照。临时快照需要使用 信号表。您可以通过向 Debezium 信号表发送信号请求来发起临时快照。

当您启动现有表的临时快照时，连接器会将内容附加到表已存在的主题中。如果删除了之前存在的主题，如果启用了 自动主题创建，Debezium 可以自动创建主题。

临时快照信号指定要包含在快照中的表。快照可以捕获整个数据库的内容，或者仅捕获数据库中表的子集。另外，快照也可以捕获数据库中表的内容子集。

您可以通过将 execute-snapshot 消息发送到信号表来指定要捕获的表。将 execute-snapshot 信号类型设置为 增量，并提供快照中包含的表名称，如下表所述：

表 9.1. 临时 execute-snapshot 信号记录的示例

字段	默认	值

字段	默认	值
type	**incremental**	指定您要运行的快照类型。 设置类型是可选的。目前，您只能请求 **增量** 快照。
data-collections	*N/A*	包含与要快照的表的完全限定域名匹配的正则表达式的数组。 名称的格式与 **signal.data.collection** 配置选项的格式相同。
additional-condition	*N/A*	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。
surrogate-key	*N/A*	可选字符串，指定连接器在快照过程中用作表的主键的列名称。

触发临时快照

您可以通过向信号表中添加 execute-snapshot 信号类型的条目来发起临时快照。连接器处理消息后，它会开始快照操作。快照进程读取第一个和最后一个主密钥值，并使用这些值作为每个表的开头和结束点。根据表中的条目数量以及配置的块大小，Debezium 会将表划分为块，并一次性执行每个块的快照。

目前，execute-snapshot 操作类型仅触发 增量快照。如需更多信息，请参阅 增加快照。

9.2.3. 增量快照

为了提供管理快照的灵活性，Debezium 包含附加快照机制，称为 增量快照。增量快照依赖于 Debezium 机制 向 Debezium 连接器发送信号。

在增量快照中，除了一次捕获数据库的完整状态，就像初始快照一样，Debebe 会在一系列可配置的块中捕获每个表。您可以指定您希望快照捕获的表 以及每个块的大小。块大小决定了快照在数据库的每个获取操作期间收集的行数。增量快照的默认块大小为 1024 行。

当增量快照进行时，Debebe 使用 watermarks 跟踪其进度，维护它捕获的每个表行的记录。与标准初始快照过程相比，捕获数据的阶段方法具有以下优点：

- ***您可以使用流化数据捕获并行运行增量快照，而不是在快照完成前进行后流。连接器会在快照过程中从更改日志中捕获接近实时事件，且操作都不会阻止其他操作。***
- ***如果增量快照的进度中断，您可以在不丢失任何数据的情况下恢复它。在进程恢复后，快照从停止的点开始，而不是从开始计算表。***

- 您可以随时根据需要运行增量快照，并根据需要重复该过程以适应数据库更新。例如，您可以在修改连接器配置后重新运行快照，以将表添加到其 table.include.list 属性中。

增量快照过程

当您运行增量快照时，Debezium 会按主键对每个表进行排序，然后根据 配置的块大小 将表分成块。然后，按块的工作块会捕获块中的每个表行。对于它捕获的每行，快照会发出 READ 事件。该事件代表块的快照开始时的行值。

当快照继续进行时，其他进程可能会继续访问数据库，可能会修改表记录。为了反映此类更改，INSERT、UPDATE 或 DELETE 操作会按照常常提交到事务日志。同样，持续 Debezium 流进程将继续检测这些更改事件，并将相应的更改事件记录发送到 Kafka。

Debezium 如何使用相同的主密钥在记录间解决冲突

在某些情况下，streaming 进程发出的 UPDATE 或 DELETE 事件会停止序列。也就是说，流流过程可能会发出一个修改表行的事件，该事件捕获包含该行的 READ 事件的块。当快照最终为行发出对应的 READ 事件时，其值已被替换。为确保以正确的逻辑顺序处理到达序列的增量快照事件，Debebe 使用缓冲方案来解析冲突。仅在快照事件和流化事件之间发生冲突后，De Debezium 会将事件记录发送到 Kafka。

快照窗口

为了帮助解决修改同一表行的后期事件和流化事件之间的冲突，Debebe 会使用一个所谓的 快照窗口。快照窗口分解了增量快照捕获指定表块数据的间隔。在块的快照窗口打开前，Debebe 会使用其常见行为，并将事件从事务日志直接下游发送到目标 Kafka 主题。但从特定块的快照打开后，直到关闭为止，De-duplication 步骤会在具有相同主密钥的事件之间解决冲突。

对于每个数据收集，Debezium 会发出两种类型的事件，并将其存储在单个目标 Kafka 主题中。从表直接捕获的快照记录作为 READ 操作发送。同时，当用户继续更新数据收集中的记录，并且会更新事务日志来反映每个提交，Debezium 会为每个更改发出 UPDATE 或 DELETE 操作。

当快照窗口打开时，Debezium 开始处理快照块，它会向内存缓冲区提供快照记录。在快照窗口期间，缓冲区中 READ 事件的主密钥与传入流事件的主键进行比较。如果没有找到匹配项，则流化事件记录将直接发送到 Kafka。如果 Debezium 检测到匹配项，它会丢弃缓冲的 READ 事件，并将流化记录写入目标主题，因为流的事件逻辑地取代静态快照事件。在块关闭的快照窗口后，缓冲区仅包含 READ 事件，这些事件不存在相关的事务日志事件。Debezium 将这些剩余的 READ 事件发送到表的 Kafka 主题。

连接器为每个快照块重复这个过程。

警告

SQL Server 的 Debezium 连接器不支持增量快照运行时的模式更改。

9.2.3.1. 触发增量快照

目前，启动增量快照的唯一方法是向源数据库上的 信号表发送临时快照 信号。

作为 SQL INSERT 查询，您将向信号提交信号。

在 Debezium 检测到信号表中的更改后，它会读取信号并运行请求的快照操作。

您提交的查询指定要包含在快照中的表，并可以选择指定快照操作的类型。目前，快照操作的唯一有效选项是默认值 incremental。

要指定快照中包含的表，请提供列出表或用于匹配表的正则表达式数组的 数据集合，例如：

```
{"data-collections": ["public.MyFirstTable", "public.MySecondTable"]}
```

增量快照信号的 data-collections 数组没有默认值。如果 data-collections 数组为空，Debezium 会检测到不需要任何操作，且不会执行快照。

注意

如果要包含在快照中的表的名称在数据库、模式或表的名称中包含句点(.)，以将表添加到 data-collections 数组中，您必须使用双引号转义名称的每个部分。

例如，要包含一个存在于 公共 模式的表，其名称为 My.Table，请使用以下格式：`"public"."My.Table "`。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道来触发增量快照

1. 发送 SQL 查询，将临时增量快照请求添加到信号表中：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) 1
values ('ad-hoc-1', 2
  'execute-snapshot', 3
  '{"data-collections": ["schema1.table1", "schema2.table2"], 4
  "type":"incremental"}, 5
  "additional-condition":"color=blue"}'); 6
```

命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 9.2. SQL 命令中字段的描述，用于将增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。相反，Debebe 会在快照期间生成自己的 **id** 字符串作为水位线信号。

项	值	描述
3	**execute-snapshot**	**type** 参数指定信号旨在触发的操作。
4	**data-collections**	信号的 **data** 字段所需的组件，用于指定表名称或正则表达式数组，以匹配快照中包含的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。
5	**incremental**	信号的 **data** 字段的可选 **类型** 组件，用于指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 如果没有指定值，连接器将运行增量快照。
6	**additional-condition**	可选字符串，根据表的列指定条件，用于捕获表的内容的子集。有关 **additional-condition** 参数的更多信息，请参阅 **带有额外条件的临时增量快照**。

带有额外条件的临时增量快照

如果您希望快照只包含表中的内容子集，您可以通过向快照信号附加 additional-condition 参数来修改信号请求。

典型的快照的 SQL 查询采用以下格式：

```
SELECT * FROM <tableName> ....
```

通过添加 additional-condition 参数，您可以将 WHERE 条件附加到 SQL 查询中，如下例所示：

```
SELECT * FROM <tableName> WHERE <additional-condition> ....
```

以下示例显示了向信号表发送带有额外条件的临时增量快照请求的 SQL 查询：

```
INSERT INTO <signalTable> (id, type, data) VALUES ('<id>', '<snapshotType>', '{"data-collections": ["<tableName>","<tableName>"],"type":"<snapshotType>","additional-condition":"<additional-condition>"}');
```

例如，假设您有一个包含以下列的 products 表：

- ID （主键）
- color
- quantity

如果您需要 product 表的增量快照，其中只包含 color=blue 的数据项，您可以使用以下 SQL 语句来触发快照：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue"}');
```

additional-condition 参数还允许您传递基于多个列的条件。例如，使用上例中的 product 表，您可以提交查询来触发增量快照，该快照仅包含 color=blue 和 quantity>10 的项数据：

```
INSERT INTO myschema.debezium_signal (id, type, data) VALUES('ad-hoc-1', 'execute-snapshot', '{"data-collections": ["schema1.products"],"type":"incremental", "additional-condition":"color=blue AND quantity>10"}');
```

以下示例显示了连接器捕获的增量快照事件的 JSON。

示例：增加快照事件消息

```
{
  "before":null,
  "after": {
    "pk":"1",
    "value":"New data"
  },
  "source": {
    ...
    "snapshot":"incremental" 1
  },
  "op":"r", 2
  "ts_ms":"1620393591654",
  "transaction":null
}
```

项	字段名称	描述
1	**snapshot**	指定要运行的快照操作类型。 目前，唯一有效的选项是默认值 **incremental**。 在 SQL 查询中指定 **type** 值，您提交到信号表是可选的。 如果没有指定值，连接器将运行增量快照。
2	**op**	指定事件类型。 快照事件的值是 **r**，表示 **READ** 操作。

9.2.3.2. 使用 Kafka 信号频道来触发增量快照

您可以向 配置的 Kafka 主题 发送消息，以请求连接器来运行临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 execute-snapshot，data 字段必须具有以下字段：

表 9.3. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与快照中包含的表的完全限定域名匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的列子集的条件。

execute-snapshot Kafka 消息示例：

```
Key = `test_connector`
```

```
Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"],
"type": "INCREMENTAL"}}`
```

带有额外条件的临时增量快照

Debezium 使用 additional-condition 字段来选择表内容的子集。

通常，当 Debezium 运行快照时，它会运行 SQL 查询，例如：

*SELECT * FROM <tableName> ....*

当快照请求包含 additional-condition 时，extra-condition 会附加到 SQL 查询中，例如：

*SELECT * FROM <tableName> WHERE <additional-condition> ....*

例如，如果一个 product table with the column id （主键）、color 和 brand，如果您希望快照只包含 color='blue' 的内容，当您请求快照时，您可以附加一个 additional-condition 语句来过滤内容：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue'"}}`
```

您可以使用 additional-condition 语句根据多个列传递条件。例如，如果您希望快照只包含 color='blue' 的 products 表中，以及 brand='MyBrand'，则您可以发送以下请求：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.products"], "type":
"INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}`
```

9.2.3.3. 停止增量快照

您还可以通过向源数据库上的表发送信号来停止增量快照。您可以通过发送 SQL INSERT 查询向表提交停止快照信号。

在 Debezium 检测到信号表中的更改后，它会读取信号，并在正在进行时停止增量快照操作。

您提交的查询指定 增量 的快照操作，以及要删除的当前运行快照的表。

先决条件

- 启用了信号。
  - 源数据库中存在信号数据收集。
  - 信号数据收集在 signal.data.collection 属性中指定。

使用源信号频道停止增量快照

1. 发送 SQL 查询以停止临时增量快照到信号表：

```
INSERT INTO <signalTable> (id, type, data) values ('<id>', 'stop-snapshot', '{"data-collections": ["<tableName>","<tableName>"],"type":"incremental"}');
```

例如，

```
INSERT INTO myschema.debezium_signal (id, type, data) (1)
values ('ad-hoc-1', (2)
  'stop-snapshot', (3)
  '{"data-collections": ["schema1.table1", "schema2.table2"], (4)
  "type":"incremental"}'); (5)
```

signal 命令中的 id、type 和 data 参数的值对应于 信号表 的字段。

下表描述了示例中的参数：

表 9.4. SQL 命令中字段的描述，用于将停止增量快照信号发送到信号表

项	值	描述
1	**myschema.debezium_signal**	指定源数据库上信号表的完全限定名称。
2	**ad-hoc-1**	**id** 参数指定一个任意字符串，它被分配为信号请求的 **id** 标识符。使用此字符串识别信号表中的条目的日志记录消息。Debezium 不使用此字符串。
3	**stop-snapshot**	指定 **type** 参数指定信号要触发的操作。
4	**data-collections**	信号的 **data** 字段的可选组件，用于指定表名称或正则表达式数组，以匹配要从快照中删除的表名称。 数组列出了按照完全限定名称匹配表的正则表达式，其格式与您在 **signal.data.collection** 配置属性中指定连接器信号表的名称相同。如果省略了 **data** 字段的这一组件，信号将停止正在进行的整个增量快照。
5	**incremental**	信号的 **data** 字段所需的组件，用于指定要停止的快照操作类型。目前，唯一有效的选项是 **增量的**。 如果没有指定 **类型** 值，信号将无法停止增量快照。

9.2.3.4. 使用 Kafka 信号频道停止增量快照

您可以将信号消息发送到 配置的 Kafka 信号主题，以停止临时增量快照。

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

message 的值是带有 type 和 data 字段的 JSON 对象。

信号类型是 stop-snapshot，data 字段必须具有以下字段：

表 9.5. 执行快照数据字段

字段	默认	值
type	**incremental**	要执行的快照的类型。目前，Debeium 仅支持 **增量** 类型。 详情请查看下一节。

字段	默认	值
data-collections	*N/A*	可选数组，以逗号分隔的正则表达式，与表的完全限定域名匹配，以包含在快照中。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。

以下示例显示了典型的 stop-snapshot Kafka 信息：

```
Key = `test_connector`

Value = `{"type":"stop-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

9.2.4. Debezium SQL Server 连接器如何读取更改数据表

当连接器首次启动时，它会获取捕获表的结构快照，并将此信息保留在其内部数据库 schema 历史记录主题中。然后，连接器会识别每个源表的更改表，并完成以下步骤。

1. *对于每个更改表，连接器会读取在上次存储的最大 LSN 和当前的最大 LSN 之间创建的所有更改。*
2. *连接器会根据提交 LSN 的值对读取的更改进行排序，并更改 LSN。这种排序顺序可确保，Debezium 会按照数据库中的相同顺序重新执行更改。*
3. *连接器传递提交并将 LSNs 作为偏移改为 Kafka Connect。*
4. *连接器存储最大 LSN，并从第 1 步中重启该过程。*

重启后，连接器会从它读取的最后一个偏移（提交并更改 LSN）中恢复处理。

连接器可以检测为包含的源表启用或禁用 CDC，并调整其行为。

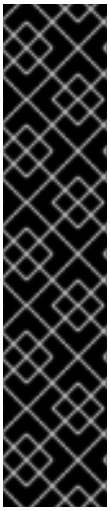
9.2.5. 没有数据库中记录的最大 LSN

有些情况下，在数据库中没有记录最大 LSN，因为：

1. SQL Server Agent 没有运行
2. 更改表中尚未记录任何更改
3. 数据库具有低活动，cdc 清理作业会定期清除 cdc 表中的条目

由于运行 SQL Server Agent 是这些可能性，因此没有问题，因此没有问题（没有 2. 和 3. 正常）。

为了缓解这个问题，并区分 No 1. 和其他不同，SQL 服务器代理的状态是通过以下查询 "SELECT CASE WHEN dss.[status]=4 THEN 1 ELSE 0 END AS isRunning FROM [#db].sys.dm_server_services dss WHERE dss.[servicename] LIKE NSQL'SQL Server Agent;%'. 如果 SQL Server Agent 没有运行，日志中会编写 ERROR: "No maximum LSN recorded in the database; SQL Server Agent is not running"。



重要

运行状态查询的 SQL Server Agent 需要 VIEW SERVER STATE 服务器权限。如果您不想为配置的用户授予此权限，您可以选择通过 database.sqlserver.agent.status.query 属性配置自己的查询。您可以定义一个函数，它返回 true 或 1，如果 SQL Server Agent 在运行（其他情况返回 false 或 0）并安全地使用高级别权限而无需对它们进行授权，如 What minimum permissions do I need to provide to a user so that it can check the status of SQL Server Agent Service? 或 Safely and Easily Use High-Level Permissions Without Granting Them to Anyone: Server-level 所述。query 属性的配置应类似：database.sqlserver.agent.status.query=SELECT [#db].func_is_sql_server_agent_running() - 您需要使用 [#db] a作为数据库名称的占位符。

9.2.6. Debezium SQL Server 连接器的限制

SQL Server 专门要求基础对象成为表，以便创建更改捕获实例。因此，SQL Server 不支持从索引视图（也称为材料化视图）捕获更改，因此 Debezium SQL Server 连接器。

9.2.7. 接收 Debezium SQL Server 更改事件的 Kafka 主题的默认名称

默认情况下，SQL Server 连接器会将表中的所有 INSERT、UPDATE 和 DELETE 操作的事件写入特定于该表的单一 Apache Kafka 主题。连接器使用以下惯例来命名更改事件主题： < topicPrefix> .

<schemaName& gt; . <tableName>

以下列表为默认名称的组件提供定义：

topicPrefix

由 topic.prefix 配置属性指定的服务器的逻辑名称。

schemaName

发生更改事件的数据库模式的名称。

tableName

发生更改事件的数据库表的名称。

例如，如果 fulfillment 是逻辑服务器名称，dbo 是 schema 名称，数据库包括名为 products, products_on_hand, customers, 和 orders 的表，连接器将更改事件发送到以下 Kafka 主题：

- *fulfillment.testDB.dbo.products*
- *fulfillment.testDB.dbo.products_on_hand*
- *fulfillment.testDB.dbo.customers*
- *fulfillment.testDB.dbo.orders*

连接器应用类似的命名约定，以标记其内部数据库架构历史记录主题、架构更改主题 和事务元数据主题。

如果默认主题名称不满足您的要求，您可以配置自定义主题名称。要配置自定义主题名称，您可以在逻辑主题路由 SMT 中指定正则表达式。有关使用逻辑主题路由 SMT 来自定义主题命名的更多信息，请参阅 主题路由。

9.2.8. Debezium SQL Server 连接器如何处理数据库架构更改

当数据库客户端查询数据库时，客户端将使用数据库的当前架构。但是，数据库模式可以随时更改，这意味着连接器必须能够识别每个插入、更新或删除操作被记录的时间。另外，连接器不一定将当前的模式应用到每个事件。如果事件相对旧，则应用当前模式之前可能会记录该事件。

为确保在 schema 更改后正确处理更改事件，Debezium SQL Server 连接器会根据 SQL Server 更改表中的结构存储新模式的快照，该表反映了其相关数据表的结构。连接器在数据库 schema 历史记录 Kafka 主题中存储表 schema 信息，以及结果更改 LSN。连接器使用存储的 schema 表示来生成更改事件，这些事件在每次插入、更新或删除操作时正确镜像表结构。

当连接器在崩溃或安全停止后重启时，它会在它读取的最后一个位置中恢复 SQL Server CDC 表中的条目。根据连接器从数据库架构历史记录主题读取的 schema 信息，连接器应用存在于连接器重启的位置上的表结构。

如果您更新处于捕获模式的 Db2 表的 schema，您也务必要更新对应更改表的模式。您必须是一个具有升级权限的 SQL Server 数据库管理员，才能更新数据库架构。有关在 Debezium 环境中更新 SQL Server 数据库模式的更多信息，请参阅 数据库模式演进。

数据库架构历史记录主题仅用于内部连接器。另外，连接器也可以将 模式更改事件发送到用于消费者应用程序的不同主题。

其他资源

- 接收 Debezium 事件记录 的主题的默认名称。

9.2.9. Debezium SQL Server 连接器如何使用 schema 更改主题

对于启用了 CDC 的每个表，Debezium SQL Server 连接器会存储应用于数据库中表的模式更改事件的历史记录。连接器将模式更改事件写入名为 < topicPrefix> 的 Kafka 主题，其中 topicPrefix 是 topic.prefix 配置属性中指定的逻辑服务器名称。

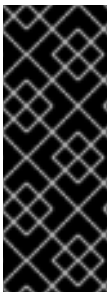
连接器发送到 schema 更改主题的消息包含一个有效负载，以及可选的包含更改事件消息的 schema。模式更改事件消息的有效负载包括以下元素：

databaseName

将语句应用到的数据库的名称。databaseName 的值充当 message 键。

tableChanges

架构更改后整个表模式的结构化表示。tableChanges 字段包含一个数组，其中包含表的每个列的条目。由于结构化表示以 JSON 或 Avro 格式呈现数据，因此用户可轻松读取消息，而不必先通过 DDL 解析器处理它们。



重要

当连接器配置为捕获表时，它只会在 schema 更改主题中存储表的历史记录，也存储在内部数据库 schema 历史记录主题中。内部数据库架构历史记录主题仅用于连接器，它不适用于消耗应用程序直接使用。确保需要通知架构更改的应用程序只消耗来自 schema 更改主题的信息。

警告

连接器发出到其 schema 更改主题的消息格式处于 incubating 状态，且可以在不通知的情况下改变。

当发生以下事件时，Debezium 会向 schema 更改主题发出一条消息：

- 您可以为表启用 CDC。
- 您可以为表禁用 CDC。
- 您可以按照 架构演变流程 更改启用了 CDC 的表结构。

示例：消息发送到 SQL Server 连接器模式更改主题

以下示例显示了 schema 更改主题中的消息。该消息包含表模式的逻辑表示。

```
{
 "schema": {
 ...
 },
 "payload": {
  "source": {
    "version": "2.3.4.Final",
    "connector": "sqlserver",
```

```
    "name": "server1",
    "ts_ms": 0,
    "snapshot": "true",
    "db": "testDB",
    "schema": "dbo",
    "table": "customers",
    "change_lsn": null,
    "commit_lsn": "00000025:00000d98:00a2",
    "event_serial_no": null
  },
  "ts_ms": 1588252618953, (1)
  "databaseName": "testDB", (2)
  "schemaName": "dbo",
  "ddl": null, (3)
  "tableChanges": [ (4)
    {
      "type": "CREATE", (5)
      "id": "\"testDB\".\"dbo\".\"customers\"", (6)
      "table": { (7)
        "defaultCharsetName": null,
        "primaryKeyColumnNames": [ (8)
          "id"
        ],
        "columns": [ (9)
          {
            "name": "id",
            "jdbcType": 4,
            "nativeType": null,
            "typeName": "int identity",
            "typeExpression": "int identity",
            "charsetName": null,
            "length": 10,
            "scale": 0,
            "position": 1,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "first_name",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "varchar",
            "typeExpression": "varchar",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 2,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "last_name",
            "jdbcType": 12,
```

```
            "nativeType": null,
            "typeName": "varchar",
            "typeExpression": "varchar",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 3,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          },
          {
            "name": "email",
            "jdbcType": 12,
            "nativeType": null,
            "typeName": "varchar",
            "typeExpression": "varchar",
            "charsetName": null,
            "length": 255,
            "scale": null,
            "position": 4,
            "optional": false,
            "autoIncremented": false,
            "generated": false
          }
        ],
        "attributes": [ 10
          {
            "customAttribute": "attributeValue"
          }
        ]
      }
    }
  ]
 }
}
```

表 9.6. 向 schema 更改主题发送的消息中字段的描述

项	字段名称	描述
1	**ts_ms**	可选字段，显示连接器处理事件的时间。这个时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 在源对象中，ts_ms 表示数据库中进行更改的时间。通过将 payload.source.ts_ms 的值与 payload.ts_ms 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。
2	**databaseName** **schemaName**	标识包含更改的数据库和架构。

项	字段名称	描述
3	**ddl**	对于 SQL Server 连接器，始终为 **null**。对于其他连接器，此字段包含负责架构更改的 DDL。此 DDL 不适用于 SQL Server 连接器。
4	**tableChanges**	包含 DDL 命令生成的模式更改的一个或多个项目的数组。
5	**type**	描述更改的类型。该值如下之一： • **CREATE** - 已创建表 • **ALTER** - 表被修改 • **DROP** - 表被删除
6	**id**	创建、更改或丢弃的表的完整标识符。
7	**table**	代表应用更改后的表元数据。
8	**primaryKeyColumnNames**	组成表主密钥的列的列表。
9	**columns**	更改表中每个列的元数据。
10	**属性**	每个表更改的自定义属性元数据。

在连接器发送到 schema 更改主题的消息中，键是包含 schema 更改的数据库的名称。在以下示例中，payload 字段包含键：

```
{
  "schema": {
    "type": "struct",
    "fields": [
      {
        "type": "string",
        "optional": false,
        "field": "databaseName"
      }
    ],
    "optional": false,
    "name": "io.debezium.connector.sqlserver.SchemaChangeKey"
  },
  "payload": {
    "databaseName": "testDB"
  }
}
```

9.2.10. Debezium SQL Server 连接器数据更改事件的描述

Debezium SQL Server 连接器为每个行级 INSERT、UPDATE 和 DELETE 操作生成数据更改事件。每个事件包含一个键和值。键的结构和值取决于已更改的表。

Debezium 和 Kafka Connect 围绕 事件消息的持续流 设计。但是，这些事件的结构可能会随时间推移而改变，而用户很难处理这些事件。要解决这个问题，每个事件都包含其内容的 schema，或者如果您正在使用 schema registry，用户可以使用该模式 ID 从 registry 获取 schema。这使得每个事件都自包含。

以下框架 JSON 显示更改事件的基本四部分。但是，如何配置您选择在应用程序中使用的 Kafka Connect converter，决定更改事件中的这四个部分的表示。只有在将转换器配置为生成它时，schema 字段才会处于更改事件中。同样，只有在您配置转换器来生成它时，事件密钥和事件有效负载才会处于更改事件中。如果您使用 JSON 转换程序，并将其配置为生成所有四个基本更改事件部分，更改事件具有此结构：

```
{
 "schema": { 1
   ...
 },
 "payload": { 2
   ...
 },
 "schema": { 3
   ...
 },
 "payload": { 4
   ...
 },
}
```

表 9.7. 更改事件基本内容概述

项	字段名称	描述
1	**schema**	第一个 **schema** 字段是事件键的一部分。它指定一个 Kafka Connect 模式，用于描述事件键的 **payload** 部分的内容。换句话说，第一个 **schema** 字段描述了主密钥的结构，如果表没有主键，则描述主键的结构。 可以通过设置 **message.key.columns** 连接器配置属性 来覆盖表的主键。在这种情况下，第一个 schema 字段描述了该属性标识的键的结构。
2	**payload**	第一个 **payload** 字段是 event 键的一部分。它具有前面的**schema** 字段描述的结构，它包含已更改的行的密钥。

项	字段名称	描述
3	**schema**	第二个 **schema** 字段是事件值的一部分。它指定 Kafka Connect 模式，用于描述事件值 **有效负载部分的内容**。换句话说，第二个 **模式** 描述了已更改的行的结构。通常，此模式包含嵌套模式。
4	**payload**	第二个 **payload** 字段是事件值的一部分。它具有上一个 **schema** 字段描述的结构，它包含已更改的行的实际数据。

默认情况下，连接器流将事件记录改为与事件原始表相同的主题。如需更多信息，请参阅 主题名称。

警告

SQL Server 连接器确保所有 Kafka Connect 模式名称都遵循 Avro 模式名称格式。这意味着逻辑服务器名称必须以拉丁字母或下划线开头，即 a-z、A-Z 或 _。逻辑服务器名称和数据库和表名称中的每个字符都必须是拉丁字母、数字或下划线，即 a-z、A-Z、0-9 或 _。如果存在无效字符，它将使用下划线字符替换。

如果逻辑服务器名称、数据库名称或表名称包含无效字符，且唯一与另一个名称区分名称的字符无效，这可能会导致意外冲突冲突，从而被下划线替换。

有关更改事件的详情，请查看以下主题：

- *第 9.2.10.1 节 “关于 Debezium SQL Server 中的键更改事件”*
- *第 9.2.10.2 节 “关于 Debezium SQL Server 中的值更改事件”*

9.2.10.1. 关于 Debezium SQL Server 中的键更改事件

更改事件的密钥包含更改表的密钥和更改行的实际键的 schema。当连接器创建事件时，schema 及其对应有效负载都会包含更改表的主键（或唯一键约束）中每个列的字段。

考虑以下 客户 表，后跟此表的更改事件键的示例。

表示例

```
CREATE TABLE customers (
  id INTEGER IDENTITY(1001,1) NOT NULL PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE
);
```

更改事件键示例

每次捕获 customer 表的更改事件都有相同的事件关键模式。只要 customers 表有以前的定义，可以捕获 customer 表更改的事件都有以下关键结构（JSON），它类似于：

```
{
   "schema": { 1
      "type": "struct",
      "fields": [ 2
         {
            "type": "int32",
            "optional": false,
            "field": "id"
         }
      ],
      "optional": false, 3
      "name": "server1.testDB.dbo.customers.Key" 4
   },
   "payload": { 5
      "id": 1004
   }
}
```

表 9.8. 更改事件键的描述

项	字段名称	描述
1	**schema**	键的 schema 部分指定一个 Kafka Connect 模式，它描述了键的 **payload** 部分的内容。
2	**fields**	指定 **有效负载中** 预期的每个字段，包括每个字段的名称、类型以及是否需要。在本例中，有一个名为 **id** 的必填字段，类型为 **int32**。

项	字段名称	描述
3	**optional**	指明 event 键是否必须在其 **payload** 字段中包含一个值。在本例中，键有效负载中的值是必需的。当表没有主键时，键的 payload 字段中的值是可选的。
4	**server1.dbo.testDB.customers.Key**	定义密钥有效负载结构的模式名称。这个 schema 描述了已更改的表的主键的结构。键模式名称的格式是 *connector-name.database-schema-name.table-name*.**Key**.在本例中： • **server1** 是生成此事件的连接器的名称。 • **dbo** 是已更改的表的数据库架构。 • **客户** 是更新的表。
5	**payload**	包含生成此更改事件的行的密钥。在本例中，键 包含一个 **id** 字段，其值为 **1004**。

9.2.10.2. 关于 Debezium SQL Server 中的值更改事件

更改事件中的值比键复杂一些。与键一样，该值有一个 schema 部分和 payload 部分。schema 部分包含描述 payload 部分的 Envelope 结构的 schema，包括其嵌套字段。为创建、更新或删除数据的操作更改事件，它们都有一个带有 envelope 结构的值有效负载。

考虑用于显示更改事件键示例的相同示例表：

```
CREATE TABLE customers (
  id INTEGER IDENTITY(1001,1) NOT NULL PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE
);
```

每个事件类型都描述了更改此表的更改事件的值部分。

- *创建 事件*
- *更新 事件*
- *删除 事件*

创建 事件

以下示例显示了一个更改事件的值部分，连接器为在 customer 表中创建数据的操作生成的更改事件的值部分：

```
{
  "schema": { (1)
    "type": "struct",
    "fields": [
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "first_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "last_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "email"
          }
        ],
        "optional": true,
        "name": "server1.dbo.testDB.customers.Value", (2)
        "field": "before"
      },
      {
        "type": "struct",
        "fields": [
          {
            "type": "int32",
            "optional": false,
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "first_name"
          },
          {
            "type": "string",
            "optional": false,
            "field": "last_name"
```

```
        },
        {
          "type": "string",
          "optional": false,
          "field": "email"
        }
      ],
      "optional": true,
      "name": "server1.dbo.testDB.customers.Value",
      "field": "after"
    },
    {
      "type": "struct",
      "fields": [
        {
          "type": "string",
          "optional": false,
          "field": "version"
        },
        {
          "type": "string",
          "optional": false,
          "field": "connector"
        },
        {
          "type": "string",
          "optional": false,
          "field": "name"
        },
        {
          "type": "int64",
          "optional": false,
          "field": "ts_ms"
        },
        {
          "type": "boolean",
          "optional": true,
          "default": false,
          "field": "snapshot"
        },
        {
          "type": "string",
          "optional": false,
          "field": "db"
        },
        {
          "type": "string",
          "optional": false,
          "field": "schema"
        },
        {
          "type": "string",
          "optional": false,
          "field": "table"
        },
        {
```

```
          "type": "string",
          "optional": true,
          "field": "change_lsn"
        },
        {
          "type": "string",
          "optional": true,
          "field": "commit_lsn"
        },
        {
          "type": "int64",
          "optional": true,
          "field": "event_serial_no"
        }
      ],
      "optional": false,
      "name": "io.debezium.connector.sqlserver.Source", 3
      "field": "source"
    },
    {
      "type": "string",
      "optional": false,
      "field": "op"
    },
    {
      "type": "int64",
      "optional": true,
      "field": "ts_ms"
    }
  ],
  "optional": false,
  "name": "server1.dbo.testDB.customers.Envelope" 4
},
"payload": { 5
  "before": null, 6
  "after": { 7
    "id": 1005,
    "first_name": "john",
    "last_name": "doe",
    "email": "john.doe@example.org"
  },
  "source": { 8
    "version": "2.3.4.Final",
    "connector": "sqlserver",
    "name": "server1",
    "ts_ms": 1559729468470,
    "snapshot": false,
    "db": "testDB",
    "schema": "dbo",
    "table": "customers",
    "change_lsn": "00000027:00000758:0003",
    "commit_lsn": "00000027:00000758:0005",
    "event_serial_no": "1"
  },
  "op": "c", 9
```

```
    "ts_ms": 1559729471739 10
  }
}
```

表 9.9. 创建 事件值字段的描述

项	字段名称	描述
1	**schema**	值的 schema，用于描述值有效负载的结构。当连接器为特定表生成的每次更改事件中，更改事件的值模式都是相同的。
2	**name**	在 **schema** 部分中，每个 **name** 字段指定值的有效负载中字段的 schema。 **server1.dbo.testDB.customers.Value** 是有效负载 **before** 和 **after** 字段的 schema。这个模式特定于 **customers** 表。 **before** 和 **after** 字段的 schema 的名称格式为 ***logicalName.database-schemaName.tableName*.Value**，这样可确保 schema 名称在数据库中是唯一的。这意味着，在使用 Avro converter 时，每个逻辑源中的每个表生成的 Avro 模式都有自己的 evolution 和 history。
3	**name**	**io.debezium.connector.sqlserver.Source** 是有效负载的 **source** 字段的 schema。这个模式特定于 SQL Server 连接器。连接器将其用于它生成的所有事件。
4	**name**	**server1.dbo.testDB.customers.Envelope** 是有效负载的整体结构的 schema，其中 **server1** 是连接器名称，**dbo** 是数据库架构名称，**customers** 是表。
5	**payload**	值的实际数据。这是更改事件提供的信息。 可能会出现事件的 JSON 表示比它们描述的行大得多。这是因为 JSON 表示必须包含消息的 schema 和 payload 部分。但是，通过使用 Avro converter，您可以显著减少连接器流到 Kafka 主题的信息大小。
6	**before**	指定事件发生前行状态的可选字段。当 **op** 字段是 **c** 用于创建（如本例所示），**before** 字段为 **null**，因为此更改事件用于新内容。
7	**after**	指定事件发生后行状态的可选字段。在本例中，**after** 字段包含新行的 **id**、**first_name**、**last_name** 和 **email** 列的值。

项	字段名称	描述
8	**source**	描述事件源元数据的必需字段。此字段包含可用于将此事件与其他事件进行比较的信息，以及事件的来源、事件发生的顺序以及事件是否为同一事务的一部分。源元数据包括： ● Debezium 版本 ● 连接器类型和名称 ● 数据库和架构名称 ● 在数据库中进行更改时的时间戳 ● 如果事件是快照的一部分 ● 包含新行的表名称 ● 服务器日志偏移
9	**op**	描述导致连接器生成事件的操作类型的强制字符串。在本例中，**c** 表示操作创建了行。有效值为： ● **c** = create ● **u** = update ● **d** = delete ● **r** = 读取（仅适用于快照）
10	**ts_ms**	可选字段，显示连接器处理事件的时间。在事件消息 envelope 中，时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 在 **source** 对象中，**ts_ms** 表示更改在数据库中提交的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

更新 事件

示例 customers 表中一个更新的改变事件的值有与那个表的 create 事件相同的模式。同样，事件值有效负载具有相同的结构。但是，事件值有效负载在 update 事件中包含不同的值。以下是连接器为 customer 表中更新生成的更改事件值 的示例 ：

```
{
  "schema": { ... },
  "payload": {
    "before": { 1
      "id": 1005,
      "first_name": "john",
      "last_name": "doe",
      "email": "john.doe@example.org"
    },
```

```
    "after": { (2)
      "id": 1005,
      "first_name": "john",
      "last_name": "doe",
      "email": "noreply@example.org"
    },
    "source": { (3)
      "version": "2.3.4.Final",
      "connector": "sqlserver",
      "name": "server1",
      "ts_ms": 1559729995937,
      "snapshot": false,
      "db": "testDB",
      "schema": "dbo",
      "table": "customers",
      "change_lsn": "00000027:00000ac0:0002",
      "commit_lsn": "00000027:00000ac0:0007",
      "event_serial_no": "2"
    },
    "op": "u", (4)
    "ts_ms": 1559729998706 (5)
  }
}
```

表 9.10. 更新 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在 *update* 事件值中，**before** 字段包含每个表列的字段，以及数据库提交前该列中的值。在这个示例中，**电子邮件**值为 **john.doe@example.org**。
2	**after**	指定事件发生后行状态的可选字段。您可以比较 **before** 和 **after** 结构，以确定这个行的更新是什么。在这个示例中，**电子邮件** 值现在是 **noreply@example.org**。

项	字段名称	描述
3	**source**	描述事件源元数据的必需字段。**source** 字段结构与 *create* 事件中的字段相同，但一些值不同，例如，示例 *update* 事件具有不同的偏移。源元数据包括： • Debezium 版本 • 连接器类型和名称 • 数据库和架构名称 • 在数据库中进行更改时的时间戳 • 如果事件是快照的一部分 • 包含新行的表名称 • 服务器日志偏移 **event_serial_no** 字段区分具有相同提交并更改 LSN 的事件。当此字段具有 **1** 以外的值时，典型的情况： • *更新* 事件将值设为 **2**，因为更新会在 SQL Server 的 CDC 更改表中生成两个事件(请参阅 源文档了解详细信息)。第一个事件包含旧值，第二个包含新值。连接器使用第一个事件中的值来创建第二个事件。连接器丢弃第一个事件。 • 更新主键时，SQL Server 会发出两个事件。对于删除带有旧主键的记录，有一个 *delete* 事件；对于添加带有新主键的记录，有一个 *create* 事件。这两个操作都共享相同的提交并更改 LSN 及其事件号为 **1** 和 **2**。
4	**op**	描述操作类型的强制字符串。在 *update* 事件值中，**op** 字段值为 **u**，表示此行因为更新而改变。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。在事件消息 envelope 中，时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 在 **source** 对象中，**ts_ms** 表示更改提交到数据库的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

注意

更新行 primary/unique 键的列会更改行的键值。当键更改时，Debezium 会输出 三个 事件：一个 delete 事件和带有行的旧键的 tombstone 事件，后跟一个带有行的新键的 create 事件。

删除 事件

delete 更改事件中的值与为同一表的 create 和 update 事件相同的 schema 部分。示例 customer 表的 delete 事件中 payload 部分类似如下：

```
{
  "schema": { ... },
 },
 "payload": {
  "before": { <>
    "id": 1005,
    "first_name": "john",
    "last_name": "doe",
    "email": "noreply@example.org"
  },
  "after": null, (1)
  "source": { (2)
    "version": "2.3.4.Final",
    "connector": "sqlserver",
    "name": "server1",
    "ts_ms": 1559730445243,
    "snapshot": false,
    "db": "testDB",
    "schema": "dbo",
    "table": "customers",
    "change_lsn": "00000027:00000db0:0005",
    "commit_lsn": "00000027:00000db0:0007",
    "event_serial_no": "1"
  },
  "op": "d", (3)
  "ts_ms": 1559730450205 (4)
 }
}
```

表 9.11. 删除 事件值字段的描述

项	字段名称	描述
1	**before**	指定事件发生前行状态的可选字段。在一个 *delete* 事件值中，**before** 字段包含在使用数据库提交删除行前的值。
2	**after**	指定事件发生后行状态的可选字段。在 *delete* 事件值中，**after** 字段为 **null**，表示行不再存在。

项	字段名称	描述
3	**source**	描述事件源元数据的必需字段。在一个 *delete* 事件值中，**source** 字段结构与同一表的 *create* 和 *update* 事件相同。许多 **source** 字段值也相同。在 *delete* 事件值中，**ts_ms** 和 **pos** 字段值以及其他值可能已更改。但是 *delete* 事件值中的 **source** 字段提供相同的元数据： • Debezium 版本 • 连接器类型和名称 • 数据库和架构名称 • 在数据库中进行更改时的时间戳 • 如果事件是快照的一部分 • 包含新行的表名称 • 服务器日志偏移
4	**op**	描述操作类型的强制字符串。**op** 字段值为 **d**，表示此行已被删除。
5	**ts_ms**	可选字段，显示连接器处理事件的时间。在事件消息 envelope 中，时间基于运行 Kafka Connect 任务的 JVM 中的系统时钟。 **在源** 对象中，**ts_ms** 表示数据库中进行更改的时间。通过将 **payload.source.ts_ms** 的值与 **payload.ts_ms** 的值进行比较，您可以确定源数据库更新和 Debezium 之间的滞后。

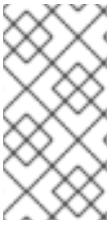
SQL Server 连接器事件旨在使用 Kafka 日志压缩。只要保留每个密钥的最新消息，日志压缩就会启用删除一些旧的消息。这可让 Kafka 回收存储空间，同时确保主题包含完整的数据集，并可用于重新载入基于密钥的状态。

tombstone 事件

删除行时，delete 事件值仍可用于日志压缩，因为 Kafka 您可以删除具有相同键的所有之前信息。但是，要让 Kafka 删除具有相同键的所有消息，消息值必须为 null。为了实现此目的，在 Debezium 的 SQL Server 连接器发出 delete 事件后，连接器会发出一个特殊的 tombstone 事件，它具有相同的键但有一个 null 值。

9.2.11. Debezium SQL Server 连接器生成的事件代表事务边界

Debezium 可以生成代表事务边界的事件，以及丰富的数据更改事件消息。



DEBEZIUM 接收事务元数据时的限制

Debezium 注册并只针对部署连接器后发生的事务接收元数据。部署连接器前发生的事务元数据不可用。

数据库事务由语句块表示，该块包含在 BEGIN 和 END 关键字之间。Debezium 为每个事务中的 BEGIN 和 END 分隔符生成事务边界事件。事务边界事件包含以下字段：

status

BEGIN 或 END.

id

唯一事务标识符的字符串。

ts_ms

数据源的事务边界事件(BEGIN 或 END 事件)的时间。如果数据源没有向事件时间提供 Debezium，则该字段代表 Debezium 处理事件的时间。

event_count （用于 END 事件）

事务提供的事件总数。

data_collections （用于 END 事件）

data_collection 和 event_count 元素的数组，用于指示连接器发出来自数据收集的更改的事件数量。

警告

Debezium 无法可靠地识别事务何时结束。因此，事务的 END 标记仅在另一个事务的第一个事件到达后发出。在低流量系统中，这可能会导致延迟发布 END 标记。

以下示例显示了典型的事务边界消息：

示例：SQL Server 连接器事务边界事件

```
{
  "status": "BEGIN",
  "id": "00000025:00000d08:0025",
  "ts_ms": 1486500577125,
  "event_count": null,
  "data_collections": null
}

{
  "status": "END",
  "id": "00000025:00000d08:0025",
  "ts_ms": 1486500577691,
  "event_count": 2,
  "data_collections": [
    {
      "data_collection": "testDB.dbo.testDB.tablea",
      "event_count": 1
    },
    {
      "data_collection": "testDB.dbo.testDB.tableb",
      "event_count": 1
    }
  ]
}
```

除非通过 topic.transaction 选项覆盖，否则事务事件将写入名为 <topic. prefix>.transaction 的主题。

9.2.11.1. 更改数据事件增强

如果启用了事务元数据，数据消息 Envelope 会增加一个新的 transaction 字段。此字段以字段复合的形式提供有关每个事件的信息：

id

唯一事务标识符的字符串

total_order

事件在事务生成的所有事件中绝对位置

data_collection_order

在事务发送的所有事件间事件的每个数据收集位置

以下示例显示了典型的信息是什么：

```
{
  "before": null,
  "after": {
    "pk": "2",
    "aa": "1"
  },
  "source": {
...
  },
  "op": "c",
  "ts_ms": "1580390884335",
  "transaction": {
    "id": "00000025:00000d08:0025",
    "total_order": "1",
    "data_collection_order": "1"
  }
}
```

9.2.12. Debezium SQL Server 连接器如何映射数据类型

Debezium SQL Server 连接器通过生成与行存在表类似的事件来代表表行数据的更改。每个事件都包含用于代表行列值的字段。事件代表操作的列值的方式取决于列的 SQL 数据类型。在事件中，连接器将每个 SQL Server 数据类型的字段映射到 字面类型和 语义类型。

连接器可以将 SQL Server 数据类型映射到 literal 和 semantic 类型。

字面类型

描述如何使用 Kafka Connect 模式类型（即 INT8, INT16, INT32, INT64, FLOAT32, FLOAT64, BOOLEAN, STRING, BYTES, ARRAY, MAP, 和 STRUCT）来代表值。

语义类型

描述 Kafka Connect 模式如何使用字段的名称捕获字段 的含义。

如果默认数据类型转换不满足您的需要，您可以为连接器 创建自定义转换器。

有关数据类型映射的更多信息，请参阅以下部分：

- *基本类型*
- *时序值*
- *十进制值*
- *时间戳值*

基本类型

下表显示了连接器如何映射基本 SQL Server 数据类型。

表 9.12. SQL Server 连接器使用的数据类型映射

SQL Server 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
位	**布尔值**	不适用
TINYINT	**INT16**	不适用
SMALLINT	**INT16**	不适用
INT	**INT32**	不适用
BIGINT	**INT64**	不适用
REAL	**FLOAT32**	不适用
FLOAT[(N)]	**FLOAT64**	不适用
CHAR[(N)]	**字符串**	不适用
VARCHAR[(N)]	**字符串**	不适用
TEXT	**字符串**	不适用
NCHAR[(N)]	**字符串**	不适用

SQL Server 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
NVARCHAR[(N)]	**字符串**	不适用
NTEXT	**字符串**	不适用
XML	**字符串**	**io.debezium.data.Xml** 包含 XML 文档的字符串表示
DATETIMEOFFSET[(P)]	**字符串**	**io.debezium.time.ZonedTimestamp** 是带有时区信息的时间戳的字符串，其中时区为 GMT

以下部分描述了其他数据类型映射。

如果存在，则列的默认值会被传播到对应字段的 Kafka Connect 模式。更改消息将包含字段的默认值（除非给出了显式列值），因此很少需要从 schema 获取默认值。

时序值

除 SQL Server 的 DATETIMEOFFSET 数据类型（包含时区信息）外，其他 temporal 类型取决于 time.precision.mode 配置属性值。当将 time.precision.mode 配置属性设置为 adaptive （默认值）时，连接器将根据列的数据类型确定 temporal 类型的字面类型和语义类型，以便事件 准确 代表数据库中的值：

SQL Server 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**io.debezium.time.Date** 代表自时期起的天数。
TIME (0), **TIME (1)**, **TIME (2)**, **TIME (3)**	**INT32**	**io.debezium.time.Time** 代表过去的毫秒数，不包括时区信息。
TIME (4), **TIME (5)**, **TIME (6)**	**INT64**	**io.debezium.time.MicroTime** 代表过去的微秒数，不包括时区信息。
TIME (7)	**INT64**	**io.debezium.time.NanoTime** 代表过去午夜的纳秒数量，不包括时区信息。

SQL Server 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATETIME	**INT64**	**io.debezium.time.Timestamp** 代表 epoch 过的毫秒数，不包括时区信息。
SMALLDATETIME	**INT64**	**io.debezium.time.Timestamp** 代表 epoch 过的毫秒数，不包括时区信息。
DATETIME2 (0), **DATETIME2 (1**), **DATETIME2 (2)**, **DATETIME2 (3)**	**INT64**	**io.debezium.time.Timestamp** 代表 epoch 过的毫秒数，不包括时区信息。
DATETIME2 (4), **DATETIME2 (5)**, **DATETIME2 (6)**	**INT64**	**io.debezium.time.MicroTimestamp** 代表 epoch 过的微秒数，不包括时区信息。
DATETIME2 (7)	**INT64**	**io.debezium.time.NanoTimestamp** 代表 epoch 过的纳秒的数量，不包括时区信息。

当将 time.precision.mode 配置属性设为 connect 时，连接器将使用预定义的 Kafka Connect 逻辑类型。当消费者只了解内置的 Kafka Connect 逻辑类型，且无法处理变量-precision 时间值时，这很有用。另一方面，因为 SQL 服务器支持十分之一的微秒精度，带有 connect 时间精度模式的连接器将在有一个大于 3 的 fractional second precision 数据库列时丢失一些精度：

SQL Server 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
DATE	**INT32**	**org.apache.kafka.connect.data.Date** 代表自 epoch 后的天数。
TIME([P])	**INT64**	**org.apache.kafka.connect.data.Time** 代表自午夜起的毫秒数，不包括时区信息。SQL Server 允许 **P** 在 0-7 范围内存储最多几十微秒精度，但当 **P** > 3 时，这个模式会导致精度丢失。
DATETIME	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 epoch 起的毫秒数，不包括时区信息。

SQL Server 数据类型	字面类型(schema 类型)	语义类型（模式名称）和备注
SMALLDATETIME	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表 epoch 过的毫秒数，不包括时区信息。
DATETIME2	**INT64**	**org.apache.kafka.connect.data.Timestamp** 代表自 epoch 起的毫秒数，不包括时区信息。SQL Server 允许 **P** 在 0-7 范围内存储最多几十微秒精度，但当 **P** > 3 时，这个模式会导致精度丢失。

时间戳值

DATETIME,SMALLDATETIME 和 DATETIME2 类型代表一个没有时区信息的时间戳。此类列根据 UTC 转换为对等的 Kafka Connect 值。对于实例，DATETIME2 值 "2018-06-20 15:13:16.945104" 由值 "1529507596945104" 为 io.debezium.time.MicroTimestamp 代表。

请注意，运行 Kafka Connect 和 Debezium 的 JVM 时区不会影响这个转换。

十进制值

Debezium 连接器根据 decimal.handling.mode 连接器配置属性的 设置处理十进制。

decimal.handling.mode=precise

表 9.13. 当 decimal.handling.mode=precise时映射

SQL Server 类型	字面类型 (schema 类型)	语义类型(schema name)
NUMERIC[(P[,S])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** 模式参数包括一个整数，它代表了十进制小数点移动了多少位。
DECIMAL[(P[,S])]	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** 模式参数包括一个整数，它代表了十进制小数点移动了多少位。
SMALLMONEY	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** 模式参数包括一个整数，它代表了十进制小数点移动了多少位。

SQL Server 类型	字面类型 (schema 类型)	语义类型(schema name)
金钱	**BYTES**	**org.apache.kafka.connect.data.Decimal** **scale** 模式参数包括一个整数，它代表了十进制小数点移动了多少位。

decimal.handling.mode=double

表 9.14. 当 decimal.handling.mode=double时映射

SQL Server 类型	字面类型	语义类型
NUMERIC[(M[,D])]	**FLOAT64**	*不适用*
DECIMAL[(M[,D])]	**FLOAT64**	*不适用*
SMALLMONEY[(M[,D])]	**FLOAT64**	*不适用*
MONEY[(M[,D])]	**FLOAT64**	*不适用*

decimal.handling.mode=string

表 9.15. 当 decimal.handling.mode=string时映射

SQL Server 类型	字面类型	语义类型
NUMERIC[(M[,D])]	**字符串**	*不适用*
DECIMAL[(M[,D])]	**字符串**	*不适用*
SMALLMONEY[(M[,D])]	**字符串**	*不适用*
MONEY[(M[,D])]	**字符串**	*不适用*

9.3. 设置 SQL SERVER 以运行 DEBEZIUM 连接器

要使 Debezium 从 SQL Server 表捕获更改事件，具有所需权限的 SQL Server 管理员必须首先运行查询，以便在数据库中启用 CDC。然后，管理员必须为您要 Debezium 捕获的每个表启用 CDC。

注意

默认情况下，与 Microsoft SQL Server 的 JDBC 连接受 SSL 加密保护。如果没有为 SQL Server 数据库启用 SSL，或者您希望在没有使用 SSL 的情况下连接到数据库，您可以通过将连接器配置中的 database.encrypt 属性的值设置为 false 来禁用 SSL。

有关设置 SQL Server 以与 Debezium 连接器搭配使用的详情，请查看以下部分：

- 第 9.3.1 节 “在 SQL Server 数据库中启用 CDC”
- 第 9.3.2 节 “在 SQL Server 表中启用 CDC”
- 第 9.3.3 节 “验证用户是否有权访问 CDC 表”
- 第 9.3.4 节 “Azure 上的 SQL Server”
- 第 9.3.5 节 “SQL Server 捕获服务器负载和延迟上的作业代理配置的影响”
- 第 9.3.6 节 “SQL Server 捕获作业代理配置参数”

应用 CDC 后，它会捕获所有 INSERT、UPDATE 和 DELETE 操作，它们提交到启用了 CDD 的表。然后，Debezium 连接器可以捕获这些事件并将其发送到 Kafka 主题。

9.3.1. 在 SQL Server 数据库中启用 CDC

在为表启用 CDC 之前，您必须为 SQL Server 数据库启用它。SQL Server 管理员通过运行系统存储的步骤启用 CDC。系统存储的流程可以使用 SQL Server Management Studio 运行，也可以使用 Transact-SQL 运行。

先决条件

- 您是 SQL Server 的 sysadmin 固定服务器角色的成员。

- 您是数据库的 db_owner。
- SQL Server Agent 正在运行。

注意

SQL Server CDC 功能只处理在用户创建的表中发生的更改。您无法在 SQL Server master 数据库中启用 CDC。

流程

1. 在 SQL Server Management Studio 中的 View 菜单中，单击 Template Explorer。
2. 在 Template Browser 中，展开 SQL Server Templates。
3. 展开 Change Data Capture > Configuration，然后点 CDC 的 Enable Database。
4. 在模板中，将 USE 语句中的数据库名称替换为您要为 CDC 启用的数据库的名称。
5. 运行存储的步骤 sys.sp_cdc_enable_db，为 CDC 启用数据库。

为 CDC 启用数据库后，会创建一个名称 cdc 的模式，以及 CDC 用户、元数据表和其他系统对象。

以下示例演示了如何为数据库 MyDB 启用 CDC：

示例：为 CDC 模板启用 SQL Server 数据库

```
USE MyDB
GO
EXEC sys.sp_cdc_enable_db
GO
```

9.3.2. 在 SQL Server 表中启用 CDC

SQL Server 管理员必须在您要捕获的源表中启用更改数据捕获。必须已经为 CDC 启用数据库。要在表中启用 CDC，SQL Server 管理员为表运行存储的步骤 sys.sp_cdc_enable_table。可以使用 SQL Server Management Studio 或使用 Transact-SQL 运行存储的流程。对于您要捕获的每个表，必须启用 SQL Server CDC。

先决条件

- 在 SQL Server 数据库上启用了 CDC。
- SQL Server Agent 正在运行。
- 您是 db_owner 固定数据库角色的成员。

流程

1. 在 SQL Server Management Studio 中的 View 菜单中，单击 Template Explorer。
2. 在 Template Browser 中，展开 SQL Server Templates。
3. 展开 Change Data Capture > Configuration，然后单击 Enable Table Specifying Filegroup Option。
4. 在模板中，将 USE 语句中的表名称替换为您要捕获的表的名称。
5. 运行存储的步骤 sys.sp_cdc_enable_table。

   以下示例演示了如何为表 MyTable 启用 CDC：

示例：为 SQL Server 表启用 CDC

```
USE MyDB
GO

EXEC sys.sp_cdc_enable_table
@source_schema = N'dbo',
@source_name   = N'MyTable', //<.>
@role_name     = N'MyRole',  //<.>
@filegroup_name = N'MyDB_CT',//<.>
@supports_net_changes = 0
GO
```

<.> 指定要捕获的表的名称。<.> 指定角色 MyRole，您可以将其添加到您要为源表捕获的列授予 SELECT 权限的用户。sysadmin 或 db_owner 角色的用户也有权访问指定的更改表。将 @role_name 的值设置为 NULL，仅允许 sysadmin 或 db_owner 中的成员具有捕获信息的完整访问权限。<.> 指定 SQL Server 放置捕获表的 filegroup。named filegroup 必须已经存在。最好不要找到用于源表的同一 filegroup 中的更改表。

9.3.3. 验证用户是否有权访问 CDC 表

SQL Server 管理员可以运行系统存储的步骤来查询数据库或表来检索其 CDC 配置信息。可以使用 SQL Server Management Studio 或使用 Transact-SQL 运行存储的流程。

先决条件

- 您有 capture 实例捕获的所有列的 SELECT 权限。db_owner 数据库角色的成员可以查看所有定义的捕获实例的信息。
- 您在为查询包括的表信息定义的任何 gating 角色中有一个成员资格。

流程

1. 在 SQL Server Management Studio 中的 View 菜单中，单击 Object Explorer。
2. 从 Object Explorer，展开 Databases，然后展开您的数据库对象，如 MyDB。

3. 展开 Programmability > Stored processs > System Stored process。

4. 运行 sys.sp_cdc_help_change_data_capture 存储的流程来查询表。

查询不应返回空结果。

以下示例在数据库 MyDB 上运行存储的步骤 sys.sp_cdc_help_change_data_capture ：

示例：查询 CDC 配置信息的表

```
USE MyDB;
GO
EXEC sys.sp_cdc_help_change_data_capture
GO
```

查询会返回为 CDC 启用的每个表的配置信息，其中包含调用者有权访问的更改数据。如果结果为空，请验证用户具有访问捕获实例和 CDC 表的特权。

9.3.4. Azure 上的 SQL Server

Debezium SQL Server 连接器可用于 Azure 上的 SQL Server。有关在 Azure 上为 SQL Server 配置 CDC 并在 Debezium 中使用它的信息，请参阅此示例。https://learn.microsoft.com/en-us/samples/azure-samples/azure-sql-db-change-stream-debezium/azure-sql%2D%2Dsql-server-change-stream-with-debezium/

9.3.5. SQL Server 捕获服务器负载和延迟上的作业代理配置的影响

当数据库管理员为源表启用更改数据捕获时，捕获作业代理开始运行。代理从事务日志中读取新的更改事件记录，并将事件记录复制到更改数据表中。在源表中提交更改的时间以及更改出现在对应更改表中的时间，总有较小的延迟间隔。这个延迟间隔代表在源表中发生更改时以及 Debezium 可用于 Apache Kafka 的更改时之间的差距。

理想情况下，对于必须快速响应数据变化的应用程序，您希望在源和更改表之间保持关闭同步。您可能想，运行捕获代理以尽可能快地持续处理事件更改事件，可能会导致吞吐量增加，并减少 latency netobserv-wagoning 更改表，以便在事件发生后马上使用新事件记录（在最近实时发生）。但是，这不

一定如此。在寻求更多即时同步时，需要支付性能损失。每次捕获作业代理查询数据库以获取新事件记录时，它会增加数据库主机上的 CPU 负载。服务器上的额外的负载可能会对整个数据库性能造成负面影响，并可能会降低事务效率，特别是在高峰数据库使用时。

监控数据库指标非常重要，以便您知道数据库是否达到服务器无法支持捕获代理的活动级别。如果您注意到性能问题，可以修改 SQL Server 捕获代理设置，以帮助平衡数据库主机上的总体 CPU 负载与可容忍的延迟程度。

9.3.6. SQL Server 捕获作业代理配置参数

在 SQL Server 上，控制捕获作业代理行为的参数在 SQL Server 表 msdb.dbo.cdc_jobs 中定义。如果您在运行捕获作业代理时遇到问题，请调整捕获作业设置，以通过运行 sys.sp_cdc_change_job 存储的步骤并提供新值来减少 CPU 负载。

注意

有关如何配置 SQL Server 捕获作业代理参数的具体指导超出了本文档的范围。

以下参数对于修改捕获代理行为与 Debezium SQL Server 连接器一起使用的最显著：

pollinginterval

- 指定捕获代理在日志扫描周期之间等待的时间。
- 较高的值可减少数据库主机上的负载并增加延迟。
- 值 0 指定扫描之间没有等待。
- 默认值为 5。

maxtrans

- 指定每个日志扫描周期内进程的最大事务数。在捕获作业处理指定数量的事务后，它会暂停 pollinginterval 指定下一次扫描开始前的时间长度。

- 较低值可减少数据库主机上的负载并增加延迟。
- 默认值为 500。

maxscans

- 指定捕获作业可以尝试捕获数据库事务日志的扫描周期数的限制。如果 continuous 参数设为 1，则作业会在恢复扫描前暂停 pollinginterval 指定的时间长度。
- 较低值可减少数据库主机上的负载并增加延迟。
- 默认值为 10。

其他资源

- 有关捕获代理参数的更多信息，请参阅 SQL Server 文档。

9.4. 部署 DEBEZIUM SQL SERVER 连接器

您可以使用以下任一方法部署 Debezium SQL Server 连接器：

- 使用 AMQ Streams 自动创建包含连接器插件的镜像。

  这是首选的方法。

- 从 Dockerfile 构建自定义 Kafka Connect 容器镜像。

其他资源

- 第 9.4.4 节 “Debezium SQL Server 连接器配置属性的描述”

9.4.1. 使用 AMQ Streams 的 SQL Server 连接器部署

从 Debezium 1.7 开始，部署 Debezium 连接器的首选方法是使用 AMQ Streams 构建包含连接器插件的 Kafka Connect 容器镜像。

在部署过程中，您可以创建并使用以下自定义资源(CR)：

- 定义 Kafka Connect 实例的 KafkaConnect CR，并包含有关镜像中需要包含连接器工件的信息。
- KafkaConnector CR，提供包括连接器用来访问源数据库的信息。在 AMQ Streams 启动 Kafka Connect pod 后，您可以通过应用 KafkaConnector CR 来启动连接器。

在 Kafka Connect 镜像的构建规格中，您可以指定可用于部署的连接器。对于每个连接器插件，您还可以指定您的部署可以使用的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。当 AMQ Streams 构建 Kafka Connect 镜像时，它会下载指定的工件，并将其合并到镜像中。

KafkaConnect CR 中的 spec.build.output 参数指定存储生成的 Kafka Connect 容器镜像的位置。容器镜像可以存储在 Docker registry 中，也可以存储在 OpenShift ImageStream 中。要将镜像存储在 ImageStream 中，您必须在部署 Kafka Connect 前创建 ImageStream。镜像流不会被自动创建。

注意

如果使用 KafkaConnect 资源来创建集群，之后无法使用 Kafka Connect REST API 创建或更新连接器。您仍然可以使用 REST API 来检索信息。

其他资源

- 在 OpenShift 中使用 AMQ Streams 配置 Kafka 连接。
- 在 OpenShift 中部署和管理 AMQ Streams 中，使用 AMQ Streams 自动创建新容器镜像。

9.4.2. 使用 AMQ Streams 部署 Debezium SQL Server 连接器

使用早期版本的 AMQ Streams 时，要在 OpenShift 上部署 Debezium 连接器，您需要首先为连接器构建 Kafka Connect 镜像。在 OpenShift 上部署连接器的当前首选方法是使用 AMQ Streams 中的构建配置来构建 Kafka Connect 容器镜像，其中包含您要使用的 Debezium 连接器插件。

在构建过程中，AMQ Streams Operator 将 KafkaConnect 自定义资源（包括 Debezium 连接器定义）中的输入参数转换为 Kafka Connect 容器镜像。构建会从 Red Hat Maven 存储库或其他配置的 HTTP 服务器下载必要的工件。

新创建的容器被推送到在 .spec.build.output 中指定的容器 registry，用于部署 Kafka Connect 集群。在 AMQ Streams 构建 Kafka Connect 镜像后，您可以创建 KafkaConnector 自定义资源来启动构建中包含的连接器。

先决条件

- 您可以访问安装了集群 Operator 的 OpenShift 集群。
- AMQ Streams Operator 正在运行。
- 在 OpenShift 中部署和升级 AMQ Streams 所述，会部署 Apache Kafka 集群。
- Kafka Connect 在 AMQ Streams 上部署
- 您有一个 Red Hat Integration 许可证。
- 已安装 OpenShift oc CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。
- 根据您要存储 Kafka Connect 构建镜像的方式，您需要 registry 权限，或者您必须创建 ImageStream 资源：

  将构建镜像存储在镜像 registry 中，如 Red Hat Quay.io 或 Docker Hub

  - 在 registry 中创建和管理镜像的帐户和权限。

  将构建镜像存储为原生 OpenShift ImageStream

  - ImageStream 资源已部署到集群中，以存储新的容器镜像。您必须为集群显式创建 ImageStream。默认无法使用镜像流。如需有关 ImageStreams 的更多信息，请参阅在 OpenShift Container Platform 中管理镜像流。

流程

1. 登录 OpenShift 集群。

2. 为连接器创建 Debezium KafkaConnect 自定义资源(CR)，或修改现有的资源。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 metadata.annotations 和 spec.build 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

例 9.1. 定义包含 Debezium 连接器的 KafkaConnect 自定义资源的 dbz-connect.yaml 文件

在以下示例中，自定义资源被配置为下载以下工件：

- Debezium SQL Server 连接器存档。
- Service Registry 归档。Service Registry 是一个可选组件。只有在打算将 Avro 序列化与连接器搭配使用时，才添加 Service Registry 组件。
- Debezium 脚本 SMT 归档以及您要与 Debezium 连接器一起使用的关联脚本引擎。SMT 归档和脚本语言依赖项是可选组件。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时，才添加这些组件。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: debezium-kafka-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  version: 3.5.0
  build: 2
    output: 3
      type: imagestream 4
      image: debezium-streams-connect:latest
    plugins: 5
      - name: debezium-connector-sqlserver
        artifacts:
          - type: zip 6
            url: https://maven.repository.redhat.com/ga/io/debezium/debezium-
connector-sqlserver/2.3.4.Final-redhat-00001/debezium-connector-sqlserver-
2.3.4.Final-redhat-00001-plugin.zip 7
          - type: zip
            url: https://maven.repository.redhat.com/ga/io/apicurio/apicurio-registry-
```

```
distro-connect-converter/2.4.4.Final-redhat-<build-number>/apicurio-registry-distro-connect-converter-2.4.4.Final-redhat-<build-number>.zip (8)
        - type: zip
          url: https://maven.repository.redhat.com/ga/io/debezium/debezium-scripting/2.3.4.Final-redhat-00001/debezium-scripting-2.3.4.Final-redhat-00001.zip (9)
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy/3.0.11/groovy-3.0.11.jar (10)
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-jsr223/3.0.11/groovy-jsr223-3.0.11.jar
        - type: jar
          url: https://repo1.maven.org/maven2/org/codehaus/groovy/groovy-json3.0.11/groovy-json-3.0.11.jar

    bootstrapServers: debezium-kafka-cluster-kafka-bootstrap:9093

    ...
```

表 9.16. Kafka Connect 配置设置的描述

项	描述
1	将 **strimzi.io/use-connector-resources** 注解设置为 **"true"**，使 Cluster Operator 使用 **KafkaConnector** 资源在此 Kafka Connect 集群中配置连接器。
2	**spec.build** 配置指定在镜像中存储构建镜像的位置，并列出要在镜像中包含的插件，以及插件工件的位置。
3	**build.output** 指定存储新构建镜像的 registry。
4	指定镜像输出的名称和镜像名称。**output.type** 的有效值是 **要推送到** 容器 registry（如 Docker Hub 或 Quay）或 **镜像流** 的有效值，以将镜像推送到内部 OpenShift ImageStream。要使用 ImageStream，必须将 ImageStream 资源部署到集群中。有关在 KafkaConnect 配置中指定 **build.output** 的更多信息，请参阅在 OpenShift 中配置 AMQ Streams 中的 AMQ Streams Build schema 参考。
5	**plugins** 配置列出了您要包含在 Kafka Connect 镜像中的所有连接器。对于列表中的每个条目，指定一个插件**名称**，以及有关构建连接器所需的工件的信息。另外，对于每个连接器插件，您还可以包含可用于连接器的其他组件。例如，您可以添加 Service Registry 工件或 Debezium 脚本组件。
6	**artifacts.type** 的值指定在 **artifacts.url** 中指定的工件类型。有效类型为 **zip**、**tgz** 或 **jar**。Debezium 连接器存档以 **.zip** 文件格式提供。**类型** 值必须与 **url** 字段中引用的文件类型匹配。
7	**artifacts.url** 的值指定 HTTP 服务器的地址，如 Maven 存储库，用于存储连接器工件的文件。Debezium 连接器工件在 Red Hat Maven 存储库中提供。OpenShift 集群必须有权访问指定的服务器。

项	描述
8	（可选）指定用于下载 Service Registry 组件的工件 **类型和 url**。包含 Service Registry 工件，只有在您希望连接器使用 Apache Avro 来序列化带有 Service Registry 的事件键和值时，而不是使用默认的 JSON 转换程序。
9	（可选）指定 Debezium 脚本 SMT 归档的工件 **类型和 url**，以用于 Debezium 连接器。只有在打算使用 Debezium 的基于内容的路由 SMT 或 过滤 SMT 时才包括脚本 SMT。要使用脚本 SMT，您必须部署 JSR 223 兼容脚本实现，如 groovy。
10	（可选）指定 JSR 223 兼容脚本实施的 JAR 文件的工件 **类型和 url**，这是 Debezium 脚本 SMT 所需的。  **重要** 如果使用 AMQ Streams 将连接器插件合并到 Kafka Connect 镜像中，每个所需的脚本语言 **工件。url** 必须指定 JAR 文件的位置，并且 **artifacts.type** 的值也必须设置为 **jar**。无效的值会导致连接器在运行时失败。 要启用带有脚本 SMT 的 Apache Groovy 语言，示例中的自定义资源会为以下库检索 JAR 文件： • **groovy** • **Groovy-jsr223**（指定代理） • **groovy-json**（解析 JSON 字符串的模块） 作为替代方案，Debebe Debezium 脚本 SMT 也支持使用 JSR 223 实现 GraalVM JavaScript。

3. *输入以下命令将 KafkaConnect 构建规格应用到 OpenShift 集群：*

```
oc create -f dbz-connect.yaml
```

根据自定义资源中指定的配置，Streams Operator 准备要部署的 Kafka Connect 镜像。构建完成后，Operator 将镜像推送到指定的 registry 或 ImageStream，并启动 Kafka Connect 集群。集群中提供了您在配置中列出的连接器工件。

4. *创建一个 KafkaConnector 资源来定义您要部署的每个连接器的实例。*
例如，创建以下 KafkaConnector CR，并将它保存为 sqlserver-inventory-connector.yaml

例 9.2. sqlserver-inventory-connector.yaml 文件，为 Debezium 连接器定义 KafkaConnector 自定义资源

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  labels:
    strimzi.io/cluster: debezium-kafka-connect-cluster
  name: inventory-connector-sqlserver (1)
spec:
  class: io.debezium.connector.sqlserver.SqlServerConnector (2)
  tasksMax: 1 (3)
  config: (4)
    schema.history.internal.kafka.bootstrap.servers: debezium-kafka-cluster-
kafka-bootstrap.debezium.svc.cluster.local:9092
    schema.history.internal.kafka.topic: schema-changes.inventory
    database.hostname: sqlserver.debezium-sqlserver.svc.cluster.local (5)
    database.port: 1433 (6)
    database.user: debezium (7)
    database.password: dbz (8)
    topic.prefix: inventory-connector-sqlserver (9)
    table.include.list: dbo.customers (10)

    ...
```

表 9.17. 连接器配置设置的描述

项	描述
1	使用 Kafka Connect 集群注册的连接器名称。
2	连接器类的名称。
3	可以同时操作的任务数量。
4	连接器的配置。
5	主机数据库实例的地址。
6	数据库实例的端口号。
7	Debezium 用于连接到数据库的帐户名称。
8	Debezium 用于连接到数据库用户帐户的密码。
9	数据库实例或集群的主题前缀。 指定的名称只能由字母数字字符或下划线组成。 因为主题前缀被用作从这个连接器接收更改事件的任何 Kafka 主题的前缀，所以该名称在集群中的连接器之间必须是唯一的。 如果连接器与 Avro 连接器集成，则此命名空间也用于相关 Kafka Connect 模式的名称，以及相应 Avro 模式的命名空间。

项	描述
10	连接器捕获更改事件的表列表。

5. *运行以下命令来创建连接器资源：*

```
oc create -n <namespace> -f <kafkaConnector>.yaml
```

例如，

```
oc create -n debezium -f sqlserver-inventory-connector.yaml
```

连接器注册到 Kafka Connect 集群，并开始针对 KafkaConnector CR 中的 spec.config.database.dbname 指定的数据库运行。连接器 pod 就绪后，Debebe 正在运行。

现在，您可以验证 Debezium SQL Server 部署。

9.4.3. 通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium SQL Server 连接器

要部署 Debezium SQL Server 连接器，您必须构建包含 Debezium 连接器归档的自定义 Kafka Connect 容器镜像，然后将此容器镜像推送到容器 registry。然后，您需要创建以下自定义资源(CR)：

- *定义 Kafka Connect 实例的 KafkaConnect CR。CR 中的 image 属性指定您创建的容器镜像的名称，以运行 Debezium 连接器。您可以将此 CR 应用到部署 Red Hat AMQ Streams 的 OpenShift 实例。AMQ Streams 提供将 Apache Kafka 带到 OpenShift 的 operator 和镜像。*
- *定义 Debezium SQL Server 连接器的 KafkaConnector CR。将此 CR 应用到应用 KafkaConnect CR 的同一 OpenShift 实例。*

先决条件

- *SQL Server 正在运行，您完成了 设置 SQL Server 的步骤，以便使用 Debezium 连接器。*
- *AMQ Streams 部署在 OpenShift 中，并运行 Apache Kafka 和 Kafka Connect。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams*

- *podman 或 Docker 已安装。*
- *您有一个在容器 registry 中创建和管理容器（如 quay.io 或 docker.io）的帐户和权限，您要添加将运行 Debezium 连接器的容器。*

流程

1. *为 Kafka Connect 创建 Debezium SQL Server 容器：*

a. *创建一个使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 的 Dockerfile 作为基础镜像。例如，在终端窗口中输入以下命令：*

```
cat <<EOF >debezium-container-for-sqlserver.yaml (1)
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/debezium (2)
RUN cd /opt/kafka/plugins/debezium/ \
&& curl -O https://maven.repository.redhat.com/ga/io/debezium/debezium-connector-sqlserver/2.3.4.Final-redhat-00001/debezium-connector-sqlserver-2.3.4.Final-redhat-00001-plugin.zip \
&& unzip debezium-connector-sqlserver-2.3.4.Final-redhat-00001-plugin.zip \
&& rm debezium-connector-sqlserver-2.3.4.Final-redhat-00001-plugin.zip
RUN cd /opt/kafka/plugins/debezium/
USER 1001
EOF
```

项	描述
1	您可以指定您想要的任何文件名。
2	指定 Kafka Connect 插件目录的路径。如果您的 Kafka Connect 插件目录位于不同的位置，请将此路径替换为目录的实际路径。

该命令在当前目录中创建一个名为 debezium-container-for-sqlserver.yaml 的 Dockerfile。

b. *从您在上一步中创建的 debezium-container-for-sqlserver.yaml Docker 文件中构建容器镜像。在包含文件的目录中，打开终端窗口并输入以下命令之一：*

```
podman build -t debezium-container-for-sqlserver:latest .
```

```
docker build -t debezium-container-for-sqlserver:latest .
```

前面的命令使用名称 debezium-container-for-sqlserver 构建容器镜像。

c. 将自定义镜像推送到容器 registry，如 quay.io 或内部容器 registry。容器 registry 必须可供您要部署镜像的 OpenShift 实例使用。输入以下命令之一：

```
podman push <myregistry.io>/debezium-container-for-sqlserver:latest
```

```
docker push <myregistry.io>/debezium-container-for-sqlserver:latest
```

d. 创建新的 Debezium SQL Server KafkaConnect 自定义资源(CR)。例如，创建一个名为 dbz-connect.yaml 的 KafkaConnect CR，用于指定 注解和 镜像 属性。以下示例显示了一个 dbz-connect.yaml 文件的摘录，该文件描述了 KafkaConnect 自定义资源。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" 1
spec:
  #...
  image: debezium-container-for-sqlserver 2

  ...
```

项	描述
1	**metadata.annotations** 表示 **KafkaConnector** 资源用于配置在这个 Kafka Connect 集群中使用的 Cluster Operator。
2	**spec.image** 指定您创建的镜像的名称，以运行 Debezium 连接器。此属性覆盖 Cluster Operator 中的 **STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE** 变量。

e. 输入以下命令将 KafkaConnect CR 应用到 OpenShift Kafka Connect 环境：

```
oc create -f dbz-connect.yaml
```

该命令添加了一个 Kafka Connect 实例，用于指定您为运行 Debezium 连接器而创建的镜像的名称。

2. 创建一个 KafkaConnector 自定义资源来配置 Debezium SQL Server 连接器实例。

您可以在 .yaml 文件中配置 Debezium SQL Server 连接器，该文件指定连接器的配置属性。连接器配置可能指示 Debezium 为 schema 和表的子集生成事件，或者可能会设置属性，以便 Debezium 忽略、掩码或截断敏感、太大或不需要的指定列中的值。

以下示例配置了一个 Debezium 连接器，它连接到端口 1433 上的 SQL 服务器主机 192.168.99.100。此主机有一个名为 testDB 的数据库，名为 customer 的表，inventory-connector-sqlserver 是服务器的逻辑名称。

SQL Server inventory-connector.yaml

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: inventory-connector-sqlserver (1)
  labels:
    strimzi.io/cluster: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: 'true'
spec:
  class: io.debezium.connector.sqlserver.SqlServerConnector (2)
  config:
    database.hostname: 192.168.99.100 (3)
    database.port: 1433 (4)
    database.user: debezium (5)
    database.password: dbz (6)
    database.names: testDB1,testDB2 (7)
    topic.prefix: inventory-connector-sqlserver (8)
    table.include.list: dbo.customers (9)
    schema.history.internal.kafka.bootstrap.servers: my-cluster-kafka-bootstrap:9092
(10)
    schema.history.internal.kafka.topic: schemahistory.fullfillment (11)
    database.ssl.truststore: path/to/trust-store (12)
    database.ssl.truststore.password: password-for-trust-store (13)
```

表 9.18. 连接器配置设置的描述

项	描述
1	在我们使用 Kafka Connect 服务注册时连接器的名称。
2	此 SQL Server 连接器类的名称。
3	SQL Server 实例的地址。
4	SQL Server 实例的端口号。
5	SQL Server 用户的名称。
6	SQL Server 用户的密码。
7	要从中捕获更改的数据库名称。
8	SQL Server 实例/集群的主题前缀，它组成一个命名空间，并在使用 Avro converter 时用于连接器写入的 Kafka 主题、Kafka Connect 模式名称和对应 Avro 模式的命名空间。
9	连接器只捕获 **dbo.customers** 表中的更改。
10	此连接器将用来写入和恢复 DDL 语句到数据库 schema 历史记录主题的 Kafka 代理列表。
11	连接器要写入和恢复 DDL 语句的数据库模式历史记录主题的名称。本主题仅用于内部使用，不应供消费者使用。
12	存储服务器的签名证书的 SSL 信任存储的路径。除非禁用了数据库加密 (**database.encrypt=false**)，否则需要此属性。
13	SSL 信任存储密码。除非禁用了数据库加密(**database.encrypt=false**)，否则需要此属性。

3. ***使用 Kafka Connect 创建连接器实例。例如，如果您将 KafkaConnector 资源保存在 inventory-connector.yaml 文件中，您将运行以下命令：***

```
oc apply -f inventory-connector.yaml
```

前面的命令注册 inventory-connector，连接器开始针对 KafkaConnector CR 中定义的 testDB 数据库运行。

*验证 **Debezium SQL Server** 连接器是否正在运行*

如果连接器正确启动且没有错误，它会为每个连接器配置为捕获的表创建一个主题。下游应用程序可以订阅这些主题，以检索源数据库中发生的信息事件。

要验证连接器是否正在运行，您可以从 OpenShift Container Platform Web 控制台或 OpenShift CLI 工具(oc)执行以下操作：

- *验证连接器状态。*
- *验证连接器是否生成主题。*
- *验证主题是否填充了读取操作("op":"r")的事件，连接器在每个表的初始快照中生成。*

先决条件

- *Debezium 连接器部署到 OpenShift 上的 AMQ Streams。*
- *已安装 OpenShift oc CLI 客户端。*
- *访问 OpenShift Container Platform web 控制台。*

流程

1. *使用以下方法之一检查 KafkaConnector 资源的状态：*
   - *在 OpenShift Container Platform Web 控制台中：*
     a. *导航到 Home → Search。*
     b. *在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaConnector。*
     c. *在 KafkaConnectors 列表中，点您要检查的连接器的名称，如 inventory-connector-sqlserver。*

d.

在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- *在终端窗口中：*

a.

使用以下命令：

```
oc describe KafkaConnector <connector-name> -n <project>
```

例如，

```
oc describe KafkaConnector inventory-connector-sqlserver -n debezium
```

该命令返回类似以下示例的状态信息：

例 9.3. KafkaConnector 资源状态

```
Name:         inventory-connector-sqlserver
Namespace:    debezium
Labels:       strimzi.io/cluster=debezium-kafka-connect-cluster
Annotations:  <none>
API Version:  kafka.strimzi.io/v1beta2
Kind:         KafkaConnector

...

Status:
  Conditions:
    Last Transition Time:  2021-12-08T17:41:34.897153Z
    Status:                True
    Type:                  Ready
  Connector Status:
    Connector:
      State:      RUNNING
      worker_id:  10.131.1.124:8083
    Name:         inventory-connector-sqlserver
    Tasks:
      Id:             0
      State:          RUNNING
      worker_id:      10.131.1.124:8083
    Type:             source
  Observed Generation:  1
  Tasks Max:            1
  Topics:
    inventory-connector-sqlserver.inventory
```

```
    inventory-connector-sqlserver.inventory.addresses
    inventory-connector-sqlserver.inventory.customers
    inventory-connector-sqlserver.inventory.geom
    inventory-connector-sqlserver.inventory.orders
    inventory-connector-sqlserver.inventory.products
    inventory-connector-sqlserver.inventory.products_on_hand
Events:  <none>
```

2. 验证连接器是否创建了 Kafka 主题：

- 通过 OpenShift Container Platform Web 控制台。

a. 导航到 Home → Search。

b. 在 Search 页面中，点 Resources 打开 Select Resource 框，然后键入 KafkaTopic。

c. 在 KafkaTopics 列表中，点您要检查的主题名称，例如 inventory-connector-sqlserver.inventory.orders--ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d。

d. 在 Conditions 部分，验证 Type 和 Status 列中的值是否已设置为 Ready 和 True。

- 在终端窗口中：

a. 使用以下命令：

```
oc get kafkatopics
```

该命令返回类似以下示例的状态信息：

例 9.4. KafkaTopic 资源状态

```
NAME                                                CLUSTER
PARTITIONS   REPLICATION FACTOR   READY
connect-cluster-configs                             debezium-kafka-cluster   1
1                   True
```

```
connect-cluster-offsets                                    debezium-kafka-cluster   25
1               True
connect-cluster-status                                     debezium-kafka-cluster   5
1               True
consumer-offsets---84e7a678d08f4bd226872e5cdd4eb527fadc1c6a
debezium-kafka-cluster   50          1               True
inventory-connector-sqlserver--a96f69b23d6118ff415f772679da623fbbb99421
debezium-kafka-cluster   1           1               True
inventory-connector-sqlserver.inventory.addresses---
1b6beaf7b2eb57d177d92be90ca2b210c9a56480          debezium-kafka-cluster
1           1               True
inventory-connector-sqlserver.inventory.customers---
9931e04ec92ecc0924f4406af3fdace7545c483b          debezium-kafka-cluster   1
1               True
inventory-connector-sqlserver.inventory.geom---
9f7e136091f071bf49ca59bf99e86c713ee58dd5          debezium-kafka-cluster
1           1               True
inventory-connector-sqlserver.inventory.orders---
ac5e98ac6a5d91e04d8ec0dc9078a1ece439081d          debezium-kafka-cluster
1           1               True
inventory-connector-sqlserver.inventory.products---
df0746db116844cee2297fab611c21b56f82dcef          debezium-kafka-cluster   1
1               True
inventory-connector-sqlserver.inventory.products_on_hand---
8649e0f17ffcc9212e266e31a7aeea4585e5c6b5   debezium-kafka-cluster   1
1               True
schema-changes.inventory                                   debezium-kafka-cluster
1           1               True
strimzi-store-topic---effb8e3e057afce1ecf67c3f5d8e4e3ff177fc55          debezium-
kafka-cluster   1           1               True
strimzi-topic-operator-kstreams-topic-store-changelog---
b75e702040b99be8a9263134de3507fc0cc4017b  debezium-kafka-cluster  1   1
True
```

3. *检查主题内容。*

- *在终端窗口中输入以下命令：*

```
oc exec -n <project>  -it <kafka-cluster> -- /opt/kafka/bin/kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=<topic-name>
```

例如，

```
oc exec -n debezium  -it debezium-kafka-cluster-kafka-0 -- /opt/kafka/bin/kafka-
console-consumer.sh \
```

```
> --bootstrap-server localhost:9092 \
> --from-beginning \
> --property print.key=true \
> --topic=inventory-connector-sqlserver.inventory.products_on_hand
```

指定主题名称的格式与 oc describe 命令返回的格式与第 1 步中返回，例如 inventory-connector-sqlserver.inventory.addresses。

对于主题中的每个事件，命令会返回类似以下示例的信息：

例 9.5. Debezium 更改事件的内容


```
{"schema":{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"inventory-connector-sqlserver.inventory.products_on_hand.Key"},"payload":{"product_id":101}}
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-connector-sqlserver.inventory.products_on_hand.Value","field":"before"},
{"type":"struct","fields":[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"inventory-connector-sqlserver.inventory.products_on_hand.Value","field":"after"},
{"type":"struct","fields":[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":
{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connector.sqlserver.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transaction"}],"optional":false,"name":"inventory-connector-sqlserver.inventory.products_on_hand.Envelope"},"payload":{"before":null,"after":
{"product_id":101,"quantity":3},"source":{"version":"2.3.4.Final-redhat-00001","connector":"sqlserver","name":"inventory-connector-sqlserver","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"table":"products_on_hand","server_id":0,"gtid":null,"file":"sqlserver-bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":1638985247805,"transaction":null}}
```

在前面的示例中，有效负载 值显示连接器快照从表 inventory.products_on_hand 生成读取 (op" ="r")事件。 product_id 记录的 "before" 状态为 null，表示该记录不存在之前的 值。 "after" 状态对于 product_id 为 101 的项目的 quantity 显示为 3。

有关您可以为 Debezium SQL Server 连接器设置的配置属性的完整列表，请参阅 SQL Server 连接器属性。

结果

当连接器启动时， 它会 对连接器进行配置的 SQL Server 数据库执行一致的快照。 然后， 连接器开始为行级操作生成数据更改事件， 并将更改事件记录流传输到 Kafka 主题。

9.4.4. Debezium SQL Server 连接器配置属性的描述

Debezium SQL Server 连接器具有大量配置属性， 您可以使用它来实现应用程序的正确连接器行为。 许多属性都有默认值。

有关属性的信息组织如下：

- 所需的连接器配置属性
- 高级连接器配置属性
- 数据库模式历史记录连接器配置属性， 用于控制 Debezium 如何处理从数据库 schema 历史记录主题读取的事件。
  - 透传数据库架构历史记录属性
- 控制 数据库驱动程序行为的直通数据库驱动程序属性。

所需的 Debezium SQL Server 连接器配置属性

除非默认值可用， 否则需要以下配置属性。

属性	默认	描述
name	没有默认值	连接器的唯一名称。尝试使用相同的名称再次注册将失败。（所有 Kafka Connect 连接器都需要此属性。）
connector.class	没有默认值	连接器的 Java 类的名称。始终为 SQL Server 连接器使用 **io.debezium.connector.sqlserver.SqlServerConnector** 的值。
tasks.max	**1**	指定连接器可用于从数据库实例中捕获数据的最大任务数量。
database.hostname	没有默认值	SQL Server 数据库服务器的 IP 地址或主机名。
database.port	**1433**	SQL Server 数据库服务器的整数端口号。
database.user	没有默认值	连接到 SQL Server 数据库服务器时要使用的用户名。使用 Kerberos 身份验证时可以省略，可以使用 pass-through 属性进行配置。
database.password	没有默认值	连接到 SQL Server 数据库服务器时要使用的密码。
database.instance	没有默认值	指定 SQL 服务器名称实例的实例名称。
topic.prefix	没有默认值	为您希望 Debezium 捕获的 SQL Server 数据库服务器提供命名空间的主题前缀。前缀应该在所有其他连接器中唯一，因为它被用作从这个连接器接收记录的所有 Kafka 主题名称的前缀。数据库服务器逻辑名称中只能使用字母数字字符、连字符、句点和下划线。 **警告** 不要更改此属性的值。如果您重启后更改了 name 值，而不是继续向原始主题发出事件，连接器会将后续事件发送到名称基于新值的主题。连接器也无法恢复其数据库架构历史记录主题。

属性	默认	描述
schema.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，**与您要**捕获更改的模式名称匹配。不包括在 schema.**include.list 中的任何架构**名称都会从捕获其更改中排除。默认情况下，连接器捕获所有非系统模式的更改。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个名称字符串匹配；它与 schema 名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，不要设置 **schema.exclude.list** 属性。
schema.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与 **您不想**捕获更改的模式名称匹配。任何名称不包含在 schema.**exclude.list 中的模式**，其更改会被捕获，但系统模式除外。 要匹配 schema 的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与 schema 的整个名称字符串匹配；它与 schema 名称中可能存在的子字符串匹配。 如果您在配置中包含此属性，请不要设置 **schema.include.list** 属性。
table.include.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您希望 Debezium 捕获的表的完全限定表标识符匹配。默认情况下，连接器为指定模式捕获所有非系统表。当设置此属性时，连接器只捕获指定表中的更改。每个标识符都是 *schemaName.tableName*。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.exclude.list** 属性。

属性	默认	描述
table.exclude.list	没有默认值	可选的、以逗号分隔的正则表达式列表，与您要从捕获中排除的表的完全限定表标识符匹配。Debezium 捕获 **table.exclude.list** 中不包含的所有表。每个标识符都是 *schemaName.tableName*。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。 如果您在配置中包含此属性，不要设置 **table.include.list** 属性。
column.include.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与更改事件消息值中包含的列的完全限定域名匹配。列的完全限定域名格式为 *schemaName.tableName.columnName*。请注意，主键列始终包含在事件的键中，即使没有包含在值中。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；它与列中可能出现的子字符串匹配。 如果您在配置中包含此属性，不要设置 **column.exclude.list** 属性。
column.exclude.list	*空字符串*	可选的、以逗号分隔的正则表达式列表，与更改事件消息值中排除的列的完全限定域名匹配。列的完全限定域名格式为 *schemaName.tableName.columnName*。请注意，如果从值中排除了主键列，则始终包含在事件的键中。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；它与列中可能出现的子字符串匹配。 如果您在配置中包含此属性，不要设置 **column.include.list** 属性。
skip.messages.without.change	**false**	指定在包含列中没有更改时是否跳过发布消息。如果列中没有包括每个 **column.include.list** 或 **column.exclude.list** 属性的列有变化，这将过滤消息。

属性	默认	描述
column.mask.hash.*hashAlgorithm*.with.salt.*salt*; column.mask.hash.v2.*hashAlgorithm*.with.salt.*salt*	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。列的完全限定域名格式为 `<schemaName>.<tableName>.<columnName>`. *要匹配一个列的名称，Debezium 应用正则表达式，它由您指定为 _anchored* 正则表达式。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。在生成的更改事件记录中，指定列的值替换为 pseudonyms。 一个 pseudonym，它包括了通过应用指定的 *hashAlgorithm* 和 *salt* 的结果的哈希值。根据所使用的哈希函数，会维护引用完整性，而列值则替换为 pseudonyms。支持的哈希功能在 Java Cryptography 架构标准 Algorithm Name 文档的 MessageDigest 部分中 进行了描述。 在以下示例中，**CzQMA0cB5K** 是一个随机选择的 salt。 `column.mask.hash.SHA-256.with.salt.CzQMA0cB5K = inventory.orders.customerName, inventory.shipment.customerName` 如有必要，pseudonym 会自动缩短为列的长度。连接器配置可以包含多个属性，用于指定不同的哈希算法和 salt。 根据所用的 *hashAlgorithm* （选择 *salt* ）和实际数据集，生成的数据集可能无法完全屏蔽。 应该使用哈希策略版本 2 来确保在不同的位置或系统中对值进行哈希处理。
time.precision.mode	**adaptive**	时间、日期和时间戳可以以不同的精度类型代表：**adaptive**（默认）基于数据库栏的类型，使用 millisecond, microsecond, 或 nanosecond 精度值，捕获数据库中的时间和时间戳；或 **connect** 始终使用 Kafka Connect 的内置的 Time, Date, 和 Timestamp 的代表（无论数据库栏的精度，始终使用 millisecond 精度）来表示时间和时间戳的值。如需更多信息，请参阅 临时值。

属性	默认	描述
decimal.handling.mode	**精确**	指定连接器如何处理 **DECIMAL** 和 **NUMERIC** 列： **precise**（默认值）代表它们准确使用二进制格式更改事件中的 **java.math.BigDecimal** 值。 **double** 表示它们使用 **double** 值，这可能会丢失一些精度但更容易使用。 **string** 将值编码为格式化的字符串，它容易使用，但提供有关实际类型的语义信息会丢失。
include.schema.changes	**true**	布尔值，指定连接器是否应该将数据库模式中的更改发布到与数据库服务器 ID 的名称相同的 Kafka 主题。每个架构更改都使用包含数据库名称和一个 JSON 结构的键记录，该键描述了 schema 更新。这独立于连接器内部记录数据库架构历史记录。默认值是 **true**。
tombstones.on.delete	**true**	控制 *delete* 事件是否后跟一个 tombstone 事件。 **true** - 一个 delete 操作由 *delete* 事件和后续 tombstone 事件表示。 **false** - 仅有一个 *delete* 事件被抛出。 删除源记录后，发出 tombstone 事件（默认行为）可让 Kafka 在为主题启用了 日志 压缩时完全删除与已删除行键相关的所有事件。
column.truncate.to.*length*.chars	*不适用*	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果在列中的数据超过了在属性名中的 *length* 指定的字符长度时删节数据，设置此属性。将 **length** 设置为正整数值，如 **column.truncate.to.20.chars**。 列的完全限定域名会观察以下格式：< ***schemaName> . <tableName> . & lt;columnName&gt***。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。

属性	默认	描述
column.mask.with.*length*.chars	列的 *N/a* Fully-qualified 名称格式为 *schemaName.tableName.columnName*。	可选的、以逗号分隔的正则表达式列表，与基于字符列的完全限定名称匹配。如果您希望连接器屏蔽一组列的值，例如，如果它们包含敏感数据，则设置此属性。将 ***length*** 设置为一个正整数，替换在属性名称中的 *length* 指定的星号（*）的数量列中的数据。将 *length* 设置为 **0**（零）将指定列中的数据替换为空字符串。 列的完全限定域名观察以下格式：*schemaName.tableName.columnName*。要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。 您可以在单个配置中指定多个长度不同的属性。
column.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，与您希望连接器发出代表列元数据的额外参数的完全限定名称匹配。当设置此属性时，连接器会将以下字段添加到事件记录的 schema 中： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名观察以下格式：*schemaName.tableName.columnName*。 要匹配列的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与列的整个名称字符串匹配；表达式与列名称中可能存在的子字符串不匹配。

属性	默认	描述
datatype.propagate.source.type	*不适用*	可选的、以逗号分隔的正则表达式列表，用于指定为数据库中列定义的数据类型的完全限定名称。当设置此属性时，对于具有匹配数据类型的列，连接器会发出在 schema 中包含以下额外字段的事件记录： • **__debezium.source.column.type** • **__debezium.source.column.length** • **__debezium.source.column.scale** 这些参数会分别传播列的原始类型名称和长度（用于变量宽度类型）。 启用连接器来发出这个额外数据可帮助正确调整 sink 数据库中的特定数字或基于字符的列。 列的完全限定域名观察以下格式：*schemaName.tableName.typeName*。 要匹配数据类型的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与数据类型的整个名称字符串匹配；表达式与类型名称中可能存在的子字符串不匹配。 有关 SQL Server 特定数据类型名称的列表，请参阅 SQL Server 数据类型映射。

属性	默认	描述
message.key.columns	*不适用*	指定连接器用来组成自定义消息键的表达式列表，用于更改它发布到指定表的 Kafka 主题的事件记录。 默认情况下，Debezium 使用表的主键列作为它发出的记录的消息键。在默认位置，或者为缺少主密钥的表指定一个键，您可以根据一个或多个列配置自定义消息密钥。 要为表建立自定义消息键，请列出表，后跟要用作消息键的列。每个列表条目都采用以下格式： *<fully-qualified_tableName> : <keyColumn> , <keyColumn>* To base a table key on multiple column name, 在列名称间插入逗号。 每个完全限定表名称都是以下格式的一个正则表达式： *<schemaName > . <tableName>* 属性可以包括多个表的条目。使用分号分隔列表中的表条目。 以下示例为表 **inventory.customers** 和 **purchase.orders** 设置消息键： **inventory.customers:pk1,pk2;(any).purchaseorders:pk3,pk4** for the table **inventory.customer**, 列 **pk1** 和 **pk2** 被指定为消息键。对于任何 **模式中的** 购买顺序表，列 **pk3** 和 **pk4** 服务器作为消息键。 对于用来创建自定义消息键的列数量没有限制。但是，最好使用指定唯一密钥所需的最小数量。
binary.handling.mode	bytes	指定二进制(**二进制**，**varbinary**)列应在更改事件中表示，包括： **bytes** 代表二进制数据作为字节数组（默认），**base64** 代表二进制数据作为 base64 编码的字符串，**base64url-safe** 代表二进制数据为 base64url-safe-encoded String，**十六进制表示二进制数据** 为十六进制编码(base16)字符串

属性	默认	描述
schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列
field.name.adjustment.mode	none	指定应如何调整字段名称以与连接器使用的消息转换器兼容。可能的设置： • **none** 不应用任何调整。 • **avro** 将无法在 Avro 类型名中使用的字符替换为下划线。 • **avro_unicode** 将无法在 Avro 类型名称中使用的下划线或字符替换为对应的 unicode，如 _uxxxx。注意：_ 是 Java 中反斜杠的转义序列 如需更多信息，请参阅 Avro 命名。

高级 SQL Server 连接器配置属性

以下 高级配置 属性具有很好的默认值，这些默认值在大多数情况下将可以正常工作，因此很少需要在连接器的配置中指定。

属性	默认	描述

属性	默认	描述

converters	没有默认值	枚举连接器可以使用的 自定义转换器 实例的符号链接列表。例如，
		
		isbn
		
		您必须设置 **converters** 属性，使连接器能够使用自定义转换器。
		
		对于您为连接器配置的每个转换器，您还必须添加一个 **.type** 属性，它指定了实现转换器接口的类的完整名称。**.type** 属性使用以下格式：
		
		***<converterSymbolicName>*.type**
		
		例如，
		
		`isbn.type: io.debezium.test.IsbnConverter`
		
		如果要进一步控制配置的转换器的行为，您可以添加一个或多个配置参数将值传递给转换器。要将任何其他配置参数与转换器关联，请为参数名称加上转换器的符号名作为前缀。例如，
		
		`isbn.schema.name: io.debezium.sqlserver.type.Isbn`

snapshot.mode	*初始*	一个模式，用于获取捕获表的结构的初始快照和可选数据。快照完成后，连接器将继续从数据库的 redo 日志中读取更改事件。支持以下值： • **初始** ：获取捕获表的结构和数据的快照；当主题应该使用捕获的表中数据的完整表示填充时，非常有用。 • **initial_only** ：获取诸如 **initial** 的结构和数据的快照，而是在快照完成后不会过渡到流更改。 • **schema_only**: 仅获取捕获表结构的快照；当当当当当从现在发生的变化时，才应传播到主题。

属性	默认	描述
snapshot.include.collection.list	**table.include.list**中指定的所有表	一个可选的、以逗号分隔的正则表达式列表，匹配表的完全限定名（***<dbName>.<schemaName>.<tableName>***）以包括在快照中。指定的项目必须在连接器的 table.include.list 属性中命名。只有在连接器的 snapshot.mode 属性设置为除 **never** 的值时，此属性才会生效。 此属性不会影响增量快照的行为。 要匹配表的名称，Debebe 会使用正则表达式，它由您作为 *anchored* 正则表达式指定。也就是说，指定的表达式与表的整个名称字符串匹配；它与表名称中可能存在的子字符串不匹配。
snapshot.isolation.mode	*repeatable_read*	控制使用哪个事务隔离级别以及为捕获指定连接器锁定表的模式。支持以下值： • **read_uncommitted** • **read_committed** • **repeatable_read** • **snapshot** • **exclusive**（**exclusive** 模式使用可重复的读取隔离级别，但对要读取的所有表采用独占锁定）。 **快照**、**read_committed** 和 **read_uncommitted** 模式不会阻止其他事务在初始快照期间更新表行。**exclusive** 和 **repeatable_read** 模式可防止并发更新。 模式选择也会影响数据一致性。只有 **专用 和快照** 模式可以保证完全一致性，即初始快照和流日志构成了线性历史记录。如果 **可重复_read** 和 **read_committed** 模式，可能会发生这种情况，例如，添加的记录会在初始快照中出现两次，一次在流传输阶段。然而，对于数据镜像，一致性级别应该这样做。对于 **read_uncommitted**，根本没有数据一致性保证（某些数据可能会丢失或损坏）。
event.processing.failure.handling.mode	**fail**	指定连接器在处理事件时应如何响应异常。**失败** 会传播异常（代表有问题的事件的偏移），从而导致连接器停止。 **warn** 会导致有问题的事件被跳过，并记录有问题的事件的偏移。 **跳过** 跳过有问题的事件。
poll.interval.ms	**500**	正整数值，指定连接器在每个迭代过程中应等待的毫秒数，以便出现新更改事件。默认值为 500 毫秒，或 0.5 秒。

属性	默认	描述
max.queue.size	8192	正整数值，用于指定阻塞队列可以保存的最大记录数。当 Debezium 从数据库读取事件时，它会将事件放置在阻塞队列中，然后再将它们写入 Kafka。阻塞队列可以提供从数据库读取更改事件时，连接器最快于将其写入 Kafka 的信息，或者在 Kafka 不可用时从数据库读取更改事件。当连接器定期记录偏移时，队列中保存的事件会被忽略。始终将 **max.queue.size** 的值设置为大于 **max.batch.size** 的值。
max.queue.size.in.bytes	0	一个长的整数值，用于指定阻塞队列的最大卷（以字节为单位）。默认情况下，不会为阻塞队列指定卷限制。要指定队列可以消耗的字节数，请将此属性设置为正长值。 如果还设置了 **max.queue.size**，当队列的大小达到任一属性指定的限制时，写入队列将被阻止。例如，如果您设置了 **max.queue.size=1000**、和 **max.queue.size.in.bytes=5000**，在队列包含 1000 个记录后，或者队列中记录的卷达到 5000 字节后，写入队列会被阻止。
max.batch.size	2048	正整数值，指定每个应在此连接器迭代过程中处理的事件的最大大小。
heartbeat.interval.ms	0	控制发送心跳消息的频率。 此属性包含一个间隔，以毫秒为单位定义连接器将信息发送到 heartbeat 主题的频率。该属性可用于确认连接器是否仍然从数据库接收更改事件。在较长的时间段内，您还应在非捕获的表中记录更改时利用心跳消息。在这种情况下，连接器将继续从数据库读取日志，但不会将任何更改信息发送到 Kafka，这意味着没有将偏移更新提交到 Kafka。这可能会导致在连接器重启后重新更改事件。将此参数设置为 **0** 以不发送心跳信息。 默认禁用此选项。
snapshot.delay.ms	没有默认值	在启动后，连接器在进行快照前应等待的间隔为 milli-seconds； 可用于在集群中启动多个连接器时避免快照中断，这可能会导致连接器的重新平衡。
snapshot.fetch.size	2000	指定在拍摄快照时每个表中读取的最大行数。连接器将在这个大小的多个批处理中读取表内容。默认值为 2000。

属性	默认	描述
query.fetch.size	没有默认值	指定将针对给定查询的每个数据库往返获取的行数。默认为 JDBC 驱动程序的默认获取大小。
snapshot.lock.timeout.ms	10000	整数值，用于指定执行快照时要等待的最大时间（以毫秒为单位）。如果此时间间隔内无法获取表锁定，则快照将失败（同时看到 快照）。当设置为 0 时，当无法获取锁定时，连接器会立即失败。值 -1 表示无限等待。
snapshot.select.statement.overrides	没有默认值	指定要包含在快照中的表行。如果您希望快照只包含表中的行的子集，请使用属性。此属性仅影响快照。它不适用于连接器从日志中读取的事件。 该属性包含以逗号分隔的、完全限定表名称列表，格式为 < ***schemaName>.<tableName>***。例如， **"snapshot.select.statement.overrides": "inventory.products,customers.orders"** 对于列表中的每个表，添加一个进一步的配置属性，用于指定连接器在获取快照时要在表上运行的 **SELECT** 语句。指定的 **SELECT** 语句决定了快照中包含的表行的子集。使用以下格式指定此 **SELECT** 语句属性的名称： **snapshot.select.statement.overrides.*<schemaName> . <tableName>***。例如，**snapshot.select.statement.overrides.customers.orders**。 Example: 在包含 soft-delete 列 **delete_flag** 的 **customers.orders** 表中，如果您希望快照只包含没有软删除的记录，请添加以下属性： "snapshot.select.statement.overrides": "customer.orders", "snapshot.select.statement.overrides.customer.orders": "SELECT * FROM [customers].[orders] WHERE delete_flag = 0 ORDER BY id DESC" 在生成的快照中，连接器只包括 **delete_flag = 0** 的记录。

属性	默认	描述
provide.transaction.metadata	**false**	当设置为 **true** Debezium 时，使用事务边界生成事件，并使用事务元数据增强数据事件信封。
retriable.restart.connector.wait.ms	10000 (10 秒)	在发生可分配错误后重启连接器前要等待的 milli-seconds 数量。
skipped.operations	**t**	在流过程中将跳过的操作类型的逗号分隔列表。操作包括：用于 inserts/create、**u** 表示更新、**d** 表示 delete、**t** 表示 truncates，**none** 不跳过任何操作。**默认情况下，会跳过 truncate 操作（不会由此连接器发出）。**
signal.data.collection	没有默认值	用于向连接器发送信号的数据收集的完全限定名称。https://access.redhat.com/documentation/zh-cn/red_hat_integration/2023.q4/html-single/debezium_user_guide/index#debezium-signaling-enabling-source-signaling-channel 使用以下格式指定集合名称：***<databaseName> . <schemaName& gt; . <tableName>***
signal.enabled.channels	source	为连接器启用的信号频道名称列表。默认情况下，以下频道可用： • **source** • **kafka** • **file** • **jmx**
notification.enabled.channels	没有默认值	为连接器启用的通知频道名称列表。默认情况下，以下频道可用： • **sink** • **log** • **jmx**

属性	默认	描述
incremental.snapshot.allow.schema.changes	**false**	允许在增量快照期间更改模式。启用后，连接器将在增量快照期间检测模式更改，并重新选择当前块以避免锁定 DDL。 请注意，不支持对主密钥的更改，并在增量快照期间执行时可能会导致不正确的结果。另一个限制是，如果模式更改只影响列的默认值，则不会检测到更改，直到从事务日志流处理 DDL 为止。这不会影响快照事件的值，但快照事件的 schema 可能具有过时的默认值。
incremental.snapshot.chunk.size	**1024**	连接器在增量快照块期间获取并读取内存的最大行数。增加块大小可提高效率，因为快照会运行更多大小的快照查询。但是，较大的块大小还需要更多内存来缓冲快照数据。将块大小调整为提供环境中最佳性能的值。
max.iteration.transactions	0	指定在从数据库中的多个表流更改时，使用每个迭代的最大事务数来减少内存占用量。当设置为 **0**（默认）时，连接器使用当前的最大 LSN 作为范围从中获取更改。当设置为大于零的值时，连接器使用这个设置指定的第 n-th LSN 作为从中获取更改的范围。
incremental.snapshot.option.recompile	**false**	将 OPTION (RECOMPILE)查询选项用于增量快照期间使用的所有 SELECT 语句。这有助于解决可能发生的参数嗅探问题，但可能会导致源数据库的 CPU 负载增加，具体取决于查询执行的频率。
topic.naming.strategy	**io.debezium.schema.SchemaTopicNamingStrategy**	应该用来确定数据更改、模式更改、事务、心跳事件等的 TopicNamingStrategy 类的名称，默认为 **SchemaTopicNamingStrategy**。
topic.delimiter	**.**	指定主题名称的分隔符，默认为 **.**。
topic.cache.size	**10000**	在绑定的并发哈希映射中用于保存主题名称的大小。此缓存将有助于确定与给定数据收集对应的主题名称。
topic.heartbeat.prefix	**__debezium-heartbeat**	控制连接器向其发送心跳信息的主题名称。主题名称具有此模式： *topic.heartbeat.prefix.topic.prefix* 例如，如果主题前缀是 **fulfillment**，则默认主题名称为 **__debezium-heartbeat.fulfillment**。

属性	默认	描述
topic.transaction	Transactions	控制连接器向其发送事务元数据消息的主题名称。主题名称具有此模式： *topic.prefix.topic.transaction* 例如，如果主题前缀是 **fulfillment**，默认的主题名称为 **fulfillment.transaction**。 如需更多信息，请参阅 事务元数据。
snapshot.max.threads	1	指定连接器执行初始快照时使用的线程数量。要启用并行初始快照，请将属性设置为大于 1 的值。在并行初始快照中，连接器会同时处理多个表。 **重要** 并行初始快照只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。
errors.max.retries	-1	在失败前，retriable 错误（如连接错误）的最大重试次数(-1 = no limit, 0 = disabled, > 0 = num of retries)。

Debezium SQL Server 连接器数据库模式历史记录配置属性

Debezium 提供了一组 schema.history.internal.* 属性，用于控制连接器如何与 schema 历史记录主题进行交互。

下表描述了用于配置 Debezium 连接器的 schema.history.internal 属性。

表 9.19. 连接器数据库架构历史记录配置属性

属性	默认	描述
schema.history.internal.kafka.topic	没有默认值	连接器存储数据库 schema 历史记录的 Kafka 主题的全名。
schema.history.internal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。此连接用于检索之前由连接器存储的数据库架构历史记录，以及用于从源数据库读取的每个 DDL 语句。每个对都应指向 Kafka Connect 进程使用的相同 Kafka 集群。
schema.history.internal.kafka.recovery.poll.interval.ms	**100**	整数值，用于指定连接器在启动/恢复期间应等待的最大毫秒数，同时轮询持久数据。默认值为 100ms。
schema.history.internal.kafka.query.timeout.ms	**3000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端获取集群信息时应等待的最大毫秒数。
schema.history.internal.kafka.create.timeout.ms	**30000**	一个整数值，用于指定连接器在使用 Kafka admin 客户端创建 kafka 历史记录主题时应等待的最大毫秒数。
schema.history.internal.kafka.recovery.attempts	**100**	连接器在连接器恢复失败前应尝试读取持久性历史记录数据的次数上限，并显示错误。接收数据后等待的最大时间为 restore. **attempts recovery. poll.interval.ms**。
schema.history.internal.skip.unparseable.ddl	**false**	指定连接器是否应忽略格式或未知数据库语句的布尔值，或者停止处理，以便人可以解决这个问题。安全默认值为 **false**。跳过应只用于小心，因为在处理 binlog 时可能会导致数据丢失或中断。
schema.history.internal.store.only.captured.tables.ddl	**false**	一个布尔值，用于指定连接器是否记录来自 schema 或数据库中的所有表的模式结构，还是仅从为捕获的表中指定的表。 指定以下值之一： **false （默认）** 在数据库快照过程中，连接器会记录数据库中所有非系统表的 schema 数据，包括没有指定用于捕获的表。最好保留默认设置。如果您稍后决定捕获您最初未指定用于捕获的表的更改，则连接器可以轻松地开始从这些表中捕获数据，因为它们的模式结构已经存储在 schema 历史记录主题中。 Debezium 需要表的 schema 历史记录，以便它可以识别发生更改事件时存在的结构。 **true** 在数据库快照过程中，连接器只记录 Debezium 捕获更改事件的表模式。如果您更改了默认值，稍后将连接器配置为从数据库中其他表捕获数据，则连接器缺少从表中捕获更改事件所需的 schema 信息。

属性	默认	描述
schema.history.internal.store.only.captured.databases.ddl	false	一个布尔值，用于指定连接器是否记录来自数据库实例中的所有逻辑数据库的架构结构。 指定以下值之一： **true** 连接器只记录 Debezium 捕获更改事件的逻辑数据库和模式中的表的架构结构。 **false** 连接器记录所有逻辑数据库的模式结构。 **注意** MySQL Connector 的默认值为 **true**

配置制作者和消费者客户端的直通数据库架构历史记录属性

Debezium 依赖于 Kafka producer 将模式更改写入数据库架构历史记录主题。同样，它依赖于 Kafka 使用者在连接器启动时从数据库 schema 历史记录主题中读取。您可以通过将值分配给以 schema.history.internal.producer 和 schema.history.internal.consumer ruby 前缀开头的 pass-through 配置属性来定义 Kafka producer 和 消费者 客户端的配置。直通生成者和消费者数据库模式历史记录属性控制一系列行为，如这些客户端与 Kafka 代理的连接的方式，如下例所示：

```
schema.history.internal.producer.security.protocol=SSL
schema.history.internal.producer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.producer.ssl.keystore.password=test1234
schema.history.internal.producer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.producer.ssl.truststore.password=test1234
schema.history.internal.producer.ssl.key.password=test1234

schema.history.internal.consumer.security.protocol=SSL
schema.history.internal.consumer.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
schema.history.internal.consumer.ssl.keystore.password=test1234
schema.history.internal.consumer.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
schema.history.internal.consumer.ssl.truststore.password=test1234
schema.history.internal.consumer.ssl.key.password=test1234
```

Debezium 从属性名称中剥离前缀，然后再将属性传递给 Kafka 客户端。

如需有关 Kafka producer 配置属性和 Kafka 使用者配置属性的更多详情，请参阅 Kafka 文档。

Debezium 连接器 Kafka 信号配置属性

Debezium 提供了一组 signal. 属性，用于控制连接器如何与 Kafka 信号主题进行交互。*

下表描述了 Kafka 信号 属性。

表 9.20. Kafka 信号配置属性

属性	默认	描述
signal.kafka.topic	<topic.prefix>-signal	连接器监控用于临时信号的 Kafka 主题的名称。 **注意** 如果禁用了 自动主题创建，您必须手动创建所需的信号主题。需要信号主题来保留信号排序。信号主题必须具有单个分区。
signal.kafka.groupId	kafka-signal	Kafka 用户使用的组 ID 的名称。
signal.kafka.bootstrap.servers	没有默认值	连接器用来建立到 Kafka 集群的初始连接的主机/端口对列表。每个对都引用 Debezium Kafka Connect 进程使用的 Kafka 集群。
signal.kafka.poll.timeout.ms	100	一个整数值，用于指定连接器在轮询信号时等待的最大毫秒数。

Debezium 连接器传递信号 Kafka 使用者客户端配置属性

Debezium 连接器为信号 Kafka 使用者提供直通配置。透传信号属性以 signals.consumer. 前缀开始。例如，连接器将 signal.consumer.security.protocol=SSL 等属性传递给 Kafka 消费者。*

Debezium 从属性中剥离前缀，然后再将属性传递给 Kafka 信号消费者。

Debezium 连接器接收器通知配置属性

下表描述了 通知 属性。

表 9.21. sink 通知配置属性

属性	默认	描述

属性	默认	描述
notification.sink.topic.name	没有默认值	从 Debezium 接收通知的主题名称。当您将 notification.enabled.channels 属性配置为将 sink 作为启用的通知频道之一时，需要此属性。

Debezium SQL Server 连接器直通数据库驱动程序配置属性

Debezium 连接器为数据库驱动程序的直通配置提供。直通数据库属性以前缀 driver metric 开头。例如，连接器将 driver.foobar=false 等属性传递给 JDBC URL。

与 数据库架构历史记录客户端通过直通属性 一样，Debebe 会在将前缀传递给数据库驱动程序之前从属性中剥离前缀。

9.5. 在 SCHEMA 更改后刷新捕获表

为 SQL Server 表启用更改数据捕获时，随着表中的更改，事件记录将保留到服务器上的捕获表中。如果您引入源表更改结构的变化，例如，通过添加新列，则该更改不会动态反映在更改表中。只要捕获表继续使用过时的模式，Debezium 连接器无法正确发出表的数据更改事件。您必须进行干预才能刷新捕获表，以便连接器恢复处理更改事件。

由于 CDC 在 SQL Server 中实施的方式，您无法使用 Debezium 更新捕获表。要刷新捕获表，一个必须是具有升级权限的 SQL Server 数据库 operator。作为 Debezium 用户，您必须使用 SQL Server 数据库 Operator 协调任务，以完成 schema 刷新并恢复到 Kafka 主题。

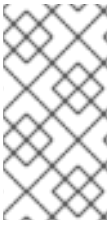
您可以使用以下方法之一在 schema 更改后更新捕获表：

- *离线 schema 更新 要求您在更新捕获表前停止 Debezium 连接器。*
- *在线 schema 更新 可以在 Debezium 连接器运行时更新捕获表。*

使用每种的步骤有优缺点。

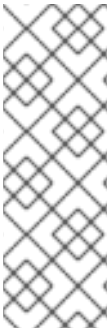
警告

无论您使用在线更新方法，还是离线更新方法，您必须在同一源表中应用后续模式更新前完成整个模式更新过程。最佳实践是在单一批处理中执行所有 DDL，因此只能运行一次。



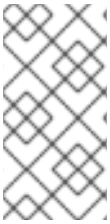
注意

在启用了 CDC 的源表中不支持一些架构更改。例如，如果在表中启用了 CDC，如果重命名了其中一个列或更改列类型，则 SQL Server 不允许更改表的 schema。



注意

在将源表中的列从 NULL 改为 NOT NULL 或反之亦然后，SQL Server 连接器无法正确捕获更改的信息，直到创建新捕获实例后。如果您在列设计后没有创建新的捕获表，请更改连接器发出的事件记录无法正确指示列是可选的。也就是说，之前定义为可选（或 NULL）的列继续是，尽管现在被定义为 NOT NULL。同样，已根据需要定义的列 (notNULL)保留该设计，尽管它们现在被定义为 NULL。



注意

使用 sp_rename 功能重命名表后，它将继续在旧源表名称下发出更改，直到连接器重启为止。在重启连接器时，它将在新源表名称下发出更改。

9.5.1. 在 schema 更改后运行离线更新

离线 schema 更新提供了更新捕获表的安全方法。但是，离线更新可能不适用于需要高可用性的应用程序。

先决条件

- *更新被提交到启用了 CDC 的 SQL Server 表的 schema。*
- *您是一个具有升级权限的 SQL Server 数据库 operator。*

流程

1. 暂停更新数据库的应用程序。

2. 等待 Debezium 连接器流传输所有未流更改事件记录。

3. 停止 Debezium 连接器。

4. 对源表 schema 应用所有更改。

5. 使用 sys.sp_cdc_enable_table 过程为 update 源表创建一个新的捕获表，参数 @capture_instance 的唯一值。

6. 恢复在第 1 步中暂停的应用程序。

7. 启动 Debezium 连接器。

8. 在 Debezium 连接器从新的捕获表开始流后，通过运行存储的流程 sys.sp_cdc_disable_table 来丢弃旧的捕获表，并将参数 @capture_instance 设置为旧的捕获实例名称。

9.5.2. 在 schema 更改后运行在线更新

完成在线模式更新的过程要比运行离线模式更新的步骤简单，您可以完成它，而无需应用程序和数据处理中的任何停机时间。但是，随着在线架构更新，在更新源数据库中的架构后，可能会出现潜在的处理差距，但在创建新的捕获实例之前。在这个间隔中，更改事件仍然会被更改表的旧实例捕获，而保存到旧表的更改数据会保留之前模式的结构。例如，如果您向源表中添加新列，更改新捕获表就绪前生成的事件，请不要包含新列的字段。如果您的应用程序不容许这样的转换周期，则最好使用离线模式更新过程。

先决条件

- 更新被提交到启用了 CDC 的 SQL Server 表的 schema。

- 您是一个具有升级权限的 SQL Server 数据库 operator。

流程

1. 对源表 schema 应用所有更改。

2. 使用 @capture_instance 参数的唯一值，运行 sys.sp_cdc_enable_table，为 update 源表创建一个新的捕获表。

3. 当 Debezium 从新的捕获表开始流时，您可以通过运行 sys.sp_cdc_disable_table，并将参数 @capture_instance 设置为旧的捕获实例名称来丢弃旧的捕获表。

示例：在数据库架构更改后运行在线模式更新

以下示例演示了如何在将 phone_number 添加到 customers 源表后在更改表中完成在线模式更新。

1. 运行以下查询来修改 customers 源表的 schema，以添加 phone_number 字段：

```
ALTER TABLE customers ADD phone_number VARCHAR(32);
```

2. 通过运行 sys.sp_cdc_enable_table 的步骤，创建新的捕获实例。

```
EXEC sys.sp_cdc_enable_table @source_schema = 'dbo', @source_name =
'customers', @role_name = NULL, @supports_net_changes = 0, @capture_instance =
'dbo_customers_v2';
GO
```

3. 运行以下查询，将新数据插入到 customers 表中：

```
INSERT INTO customers(first_name,last_name,email,phone_number) VALUES
('John','Doe','john.doe@example.com', '+1-555-123456');
GO
```

Kafka Connect 日志通过类似以下信息的条目报告配置更新：

```
connect_1    | 2019-01-17 10:11:14,924 INFO   ||  Multiple capture instances present for
the same table: Capture instance "dbo_customers"
[sourceTableId=testDB.dbo.customers,
changeTableId=testDB.cdc.dbo_customers_CT, startLsn=00000024:00000d98:0036,
changeTableObjectId=1525580473, stopLsn=00000025:00000ef8:0048] and Capture
instance "dbo_customers_v2" [sourceTableId=testDB.dbo.customers,
changeTableId=testDB.cdc.dbo_customers_v2_CT, startLsn=00000025:00000ef8:0048,
```

```
changeTableObjectId=1749581271, stopLsn=NULL]
[io.debezium.connector.sqlserver.SqlServerStreamingChangeEventSource]
connect_1    | 2019-01-17 10:11:14,924 INFO   ||  Schema will be changed for
ChangeTable [captureInstance=dbo_customers_v2,
sourceTableId=testDB.dbo.customers,
changeTableId=testDB.cdc.dbo_customers_v2_CT, startLsn=00000025:00000ef8:0048,
changeTableObjectId=1749581271, stopLsn=NULL]
[io.debezium.connector.sqlserver.SqlServerStreamingChangeEventSource]
...
connect_1    | 2019-01-17 10:11:33,719 INFO   ||  Migrating schema to ChangeTable
[captureInstance=dbo_customers_v2, sourceTableId=testDB.dbo.customers,
changeTableId=testDB.cdc.dbo_customers_v2_CT, startLsn=00000025:00000ef8:0048,
changeTableObjectId=1749581271, stopLsn=NULL]
[io.debezium.connector.sqlserver.SqlServerStreamingChangeEventSource]
```

最后，phone_number 字段添加到 schema，其值会出现在写入 Kafka 主题的消息中。

```
...
    {
      "type": "string",
      "optional": true,
      "field": "phone_number"
    }
...
  "after": {
    "id": 1005,
    "first_name": "John",
    "last_name": "Doe",
    "email": "john.doe@example.com",
    "phone_number": "+1-555-123456"
  },
```

4. 运行 sys.sp_cdc_disable_table 的步骤丢弃旧的捕获实例。

```
EXEC sys.sp_cdc_disable_table @source_schema = 'dbo', @source_name =
'dbo_customers', @capture_instance = 'dbo_customers';
GO
```

9.6. 监控 DEBEZIUM SQL SERVER 连接器性能

Debezium SQL Server 连接器提供三种指标类型，除了对 Zookeeper、Kafka 和 Kafka Connect 提供的 JMX 指标的内置支持之外。连接器提供以下指标：

- 在执行快照时监控连接器的快照 指标。

- *在读取 CDC 表数据时监控连接器的指标。*
- *用于监控连接器模式历史记录状态的 schema 历史记录指标。*

有关如何通过 JMX 公开前面的指标的详情，请参考 Debezium 监控文档。

9.6.1. Debezium SQL Server 连接器快照指标

MBean 是 debezium.sql_server:type=connector-metrics,server= <topic.prefix& gt; ,task= <task.id>,context=snapshot。

快照指标不会公开，除非快照操作处于活跃状态，或者快照自上次连接器启动以来发生。

下表列出了可用的 shapshot 指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个快照事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上次启动或重置后看到的事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在快照和主 Kafka Connect 循环之间传递事件的长度。
QueueRemainingCapacity	int	队列的空闲容量，用于在快照和主 Kafka Connect 循环之间传递事件。
TotalTableCount	int	包括在快照中的表的总数。

属性	类型	描述
RemainingTableCount	int	快照必须复制的表数。
SnapshotRunning	布尔值	快照是否已启动。
SnapshotPaused	布尔值	快照是否已暂停。
SnapshotAborted	布尔值	快照是否中止。
SnapshotCompleted	布尔值	快照是否完成。
SnapshotDurationInSeconds	long	快照为止所花费的秒数，即使未完成也是如此。也包括快照暂停的时间。
SnapshotPausedDurationInSeconds	long	快照暂停的秒数。如果快照暂停几次，暂停的时间会添加。
RowsScanned	Map<String, Long>	包含快照中每个表的行数的映射。表会在处理过程中逐步添加到映射中。更新每个 10,000 行扫描并在完成表后。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

连接器还在执行增量快照时提供以下额外快照指标：

属性	类型	描述
ChunkId	字符串	当前快照块的标识符。
ChunkFrom	字符串	定义当前块的主密钥集的下限。
ChunkTo	字符串	定义当前块的主密钥集的上限。

属性	类型	描述
TableFrom	字符串	当前快照表的主密钥集的下限。
TableTo	字符串	当前快照表的主密钥集的上限。

9.6.2. Debezium SQL Server 连接器流指标

MBean 是 debezium.sql_server:type=connector-metrics,server= <topic.prefix& gt; ,task= <task.id>,context=streaming。

下表列出了可用的流指标。

属性	类型	描述
LastEvent	字符串	连接器已读取的最后一个流事件。
MilliSecondsSinceLastEvent	long	连接器已读取并处理最新事件以来的毫秒数。
TotalNumberOfEventsSeen	long	此连接器自上一次启动或指标重置以来看到的事件总数。
TotalNumberOfCreateEventsSeen	long	此连接器自上次启动或指标重置以来看到的创建事件总数。
TotalNumberOfUpdateEventsSeen	long	此连接器自上次启动或指标重置以来看到的更新事件总数。
TotalNumberOfDeleteEventsSeen	long	此连接器自上次启动或指标重置以来看到的删除事件总数。
NumberOfEventsFiltered	long	通过连接器上配置的 include/exclude 列表过滤规则过滤的事件数量。
CapturedTables	string[]	连接器捕获的表列表。
QueueTotalCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的长度。

属性	类型	描述
QueueRemainingCapacity	int	在流器和主 Kafka Connect 循环之间传递事件的队列的可用容量。
Connected	布尔值	表示连接器目前是否连接到数据库服务器的标记。
MilliSecondsBehindSource	long	最后一次更改事件时间戳和连接器处理它之间的毫秒数。这些值将讨论运行数据库服务器和连接器的计算机上时钟之间的任何区别。
NumberOfCommittedTransactions	long	已提交的已处理事务的数量。
SourceEventPosition	Map<String, String>	最后收到的事件的协调。
LastTransactionId	字符串	最后处理事务的事务的事务标识符。
MaxQueueSizeInBytes	long	队列的最大缓冲区（以字节为单位）。如果将 max.queue.size.in.bytes 设置为正长值，则此指标可用。
CurrentQueueSizeInBytes	long	队列中记录的当前卷（以字节为单位）。

9.6.3. Debezium SQL Server 连接器模式历史记录指标

MBean 是 debezium.sql_server:type=connector-metrics,context=schema-history,server=<topic.prefix>。

下表列出了可用的模式历史记录指标。

属性	类型	描述

属性	类型	描述
Status	字符串	STOPPED 之一，RECOVERING（从存储恢复历史记录），RUNNING 描述数据库架构历史记录的状态。
RecoveryStartTime	long	恢复启动时的 epoch 秒的时间（以秒为单位）。
ChangesRecovered	long	在恢复阶段读取的更改数量。
ChangesApplied	long	恢复和运行时期间应用的模式更改总数。
MilliSecondsSinceLast RecoveredChange	long	从历史记录存储中恢复自上次更改以来经过的毫秒数。
MilliSecondsSinceLastAppliedChange	long	从上次更改被应用后经过的毫秒数。
LastRecoveredChange	字符串	从历史记录存储中恢复的最后一个更改的字符串表示。
LastAppliedChange	字符串	最后应用的更改的字符串表示。

第 10 章 监控 DEBEZIUM

您可以使用 Apache Zookeeper、Apache Kafka 和 Kafka Connect 提供的 JMX 指标来监控 Debezium。要使用这些指标，您必须在启动 Zookeeper、Kafka 和 Kafka Connect 服务时启用它们。启用 JMX 涉及设置正确的环境变量。

注意

如果您在同一台机器上运行多个服务，请确保为每个服务使用不同的 JMX 端口。

10.1. 监控 DEBEZIUM 连接器的指标

除了 Kafka、Zookeeper 和 Kafka Connect 中对 JMX 指标的内置支持外，每个连接器还提供可用于监控其活动的额外指标。

- Db2 连接器指标
- MongoDB 连接器指标
- MySQL 连接器指标
- Oracle 连接器指标
- PostgreSQL 连接器指标
- SQL Server 连接器指标

10.2. 在本地安装中启用 JMX

使用 Zookeeper、Kafka 和 Kafka Connect，您可以在启动每个服务时通过设置适当的环境变量来启用 JMX。

10.2.1. ZooKeeper JMX 环境变量

ZooKeeper 具有对 JMX 的内置支持。当使用本地安装运行 Zookeeper 时，zkServer.sh 脚本可以识别以下环境变量：

JMXPORT

启用 JMX 并指定用于 JMX 的端口号。该值用于指定 JVM 参数 -Dcom.sun.management.jmxremote.port=$JMXPORT。

JMXAUTH

JMX 客户端在连接时必须使用密码身份验证。必须为 true 或 false。默认值为 false。该值用于指定 JVM 参数 -Dcom.sun.management.jmxremote.authenticate=$JMXAUTH。

JMXSSL

JMX 客户端是否使用 SSL/TLS 进行连接。必须为 true 或 false。默认值为 false。该值用于指定 JVM 参数 -Dcom.sun.management.jmxremote.ssl=$JMXSSL。

JMXLOG4J

是否应禁用 Log4J JMX MBeans。必须为 true （默认）或 false。默认值是 true。该值用于指定 JVM 参数 -Dzookeeper.jmx.log4j.disable=$JMXLOG4J。

10.2.2. Kafka JMX 环境变量

使用本地安装运行 Kafka 时，kafka-server-start.sh 脚本可以识别以下环境变量：

JMX_PORT

启用 JMX 并指定用于 JMX 的端口号。该值用于指定 JVM 参数 -Dcom.sun.management.jmxremote.port=$JMX_PORT。

KAFKA_JMX_OPTS

JMX 选项，它们在启动时直接传递给 JVM。默认选项包括：

- -Dcom.sun.management.jmxremote
- -Dcom.sun.management.jmxremote.authenticate=false
- -Dcom.sun.management.jmxremote.ssl=false

10.2.3. Kafka Connect JMX 环境变量

使用本地安装运行 Kafka 时，connect-distributed.sh 脚本可以识别以下环境变量：

JMX_PORT

启用 JMX 并指定用于 JMX 的端口号。该值用于指定 JVM 参数 -Dcom.sun.management.jmxremote.port=$JMX_PORT。

KAFKA_JMX_OPTS

JMX 选项，它们在启动时直接传递给 JVM。默认选项包括：

- -Dcom.sun.management.jmxremote
- -Dcom.sun.management.jmxremote.authenticate=false
- -Dcom.sun.management.jmxremote.ssl=false

10.3. 监控 OPENSHIFT 上的 DEBEZIUM

如果您在 OpenShift 上使用 Debezium，可以通过打开 9999 上的 JMX 端口来获取 JMX 指标。如需更多信息，请参阅在 OpenShift 中使用 AMQ Streams 中的 JMX 选项。

另外，您可以使用 Prometheus 和 Grafana 来监控 JMX 指标。如需更多信息，请参阅在 OpenShift 中部署和升级 AMQ Streams 中的 向 Kafka 引入指标。

第 11 章 DEBEZIUM 日志记录

Debezium 在连接器中内置了广泛的日志记录，您可以更改日志配置，以控制这些日志语句出现在日志中以及这些日志的发送位置。Debezium （以及 Kafka、Kafka Connect 和 Zookeeper）使用 Java 的 Log4j 日志记录框架。

默认情况下，连接器在启动时会产生一个公平的有用信息，但在连接器与源数据库保持同步时会生成非常少的日志。当连接器正常运行时，这通常足够了，但当连接器意外处理时可能不足。在这种情况下，您可以更改日志级别，以便连接器生成更详细的日志消息，描述连接器正在做什么以及它没有做什么。

11.1. DEBEZIUM 日志记录概念

在配置日志记录前，您应该了解 Log4J loggers、日志级别 和附加 者 是什么。

日志记录器

应用生成的每个日志消息都发送到特定的 日志记录器 （如 io.debezium.connector.mysql）。日志记录器在层次结构中排列。例如，io.debezium.connector.mysql 日志记录器是 io.debezium.connector 日志记录器的子级，它是 io.debezium 日志记录器的子项。在层次结构的顶部，根日志记录器定义 其下所有日志记录器的默认日志记录器配置。

日志级别

应用程序生成的每个日志消息也具有特定 的日志级别 ：

1. ERROR - 错误、异常和其他严重问题
2. WARN - 潜在的问题 和问题
3. INFO - 状态和常规活动（通常为低卷）
4. DEBUG - 有助于诊断意外行为的更详细的活动
5. TRACE - 非常详细和详细的活动（通常是高容量）

Appenders

附加程序 基本上是记录消息的目的地。每个附加程序控制其日志消息的格式，让您可以更多地控制日志消息的样子。

要配置日志记录，您需要为每个日志记录器指定所需的级别，以及应该写入这些日志消息的附加程序。由于日志记录器是分层的，根日志记录器的配置充当以下所有日志记录器的默认值，尽管您可以覆盖任何子（或子级）日志记录器。

11.2. 默认 DEBEZIUM 日志记录配置

如果您在 Kafka Connect 进程中运行 Debezium 连接器，则 Kafka Connect 会使用 Kafka 安装中的 Log4j 配置文件（例如：/opt/kafka/config/connect-log4j.properties）。默认情况下，此文件包含以下配置：

connect-log4j.properties

```
log4j.rootLogger=INFO, stdout 1

log4j.appender.stdout=org.apache.log4j.ConsoleAppender 2
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout 3
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c)%n 4
...
```

1 1 1 1 1 1 1 1 1 1 1 1 1

根日志记录器，用于定义默认日志记录器配置。默认情况下，日志记录器包括 INFO,WARN, 和 ERROR 消息。这些日志消息被写入 stdout 附加程序。

2 2 2 2 2 2 2 2 1 2 2 2 2 2 2

stdout 附加程序将日志消息写入控制台（而不是文件）。

3 3 3 3 3 3 3 2 3 3 3 3

stdout 附加程序使用模式匹配算法来格式化日志消息。

4 4 4 4 4 4 4 3 4 4 4 4

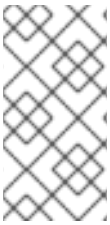
stdout 附加器的模式（请参阅 Log4j 文档 了解详细信息）。

除非配置了其他日志记录器，否则 Debezium 使用的所有日志记录器都会继承 rootLogger 配置。

11.3. 配置 DEBEZIUM 日志记录

默认情况下，Debezium 连接器将所有 INFO、WARN 和 ERROR 消息写入控制台。您可以使用以下方法之一更改默认日志配置：

- 通过配置日志记录器来设置日志记录级别
- 使用 Kafka Connect REST API 动态设置日志级别
- 通过添加映射的诊断上下文来设置日志记录级别



注意

您可以使用其他方法使用 Log4j 配置 Debezium 日志记录。如需更多信息，请搜索有关设置和使用附加程序将日志消息发送到特定目的地的教程。

11.3.1. 通过配置日志记录器来更改 Debezium 日志记录级别

默认 Debezium 日志记录级别提供了足够的信息来显示连接器是否健康。但是，如果连接器不健康，您可以更改其日志级别来排除这个问题。

通常，Debezium 连接器会将其日志消息发送到日志记录器，其名称与生成日志消息的 Java 类的完全限定名称匹配。Debezium 使用软件包来整理具有类似或相关功能的代码。这意味着，您可以控制特定类或特定软件包中的所有类的所有日志消息。

流程

1. 打开 log4j.properties 文件。
2. 为连接器配置日志记录器。

本例为 MySQL 连接器和连接器使用的数据库架构历史记录实现配置日志记录器，并将其设置为 log DEBUG 级别消息：

log4j.properties

```
...
log4j.logger.io.debezium.connector.mysql=DEBUG, stdout (1)
log4j.logger.io.debezium.relational.history=DEBUG, stdout (2)

log4j.additivity.io.debezium.connector.mysql=false (3)
log4j.additivity.io.debezium.storage.kafka.history=false (4)
...
```

(1)

配置名为 io.debezium.connector.mysql 的日志记录器，将 DEBUG,INFO,WARN, 和 ERROR 消息发送到 stdout appender。

(2)

配置名为 io.debezium.relational.history 的日志记录器，以将 DEBUG、INFO、WARN 和 ERROR 消息发送到 stdout 附加程序。

(3) (4)

关闭 添加性，这会导致日志消息不会发送到父日志记录器的附加者（这可以防止在使用多个附件器时看到重复的日志消息）。

3. 如有必要，更改连接器中类的特定子集的日志级别。

增加整个连接器的日志记录级别会增加日志详细程度，这可能会难以了解发生的情况。在这些情况下，您只能为与您要故障排除的问题相关的类子集更改日志级别。

a. 将连接器的日志记录级别设置为 DEBUG 或 TRACE。

b. 查看连接器的日志消息。

查找与您要故障排除的问题相关的日志消息。每个日志消息的末尾显示生成消息的 Java 类的名称。

c.

将连接器的日志记录级别设置为 INFO。

d.

为您识别的每个 Java 类配置日志记录器。

例如，假设一个场景，您不确定为什么 MySQL 连接器在处理 binlog 时跳过某些事件。除了为整个连接器打开 DEBUG 或 TRACE 日志记录外，您可以将连接器的日志级别保持为 INFO，然后只在读取 binlog 的类上配置 DEBUG 或 TRACE ：

log4j.properties

```
...
log4j.logger.io.debezium.connector.mysql=INFO, stdout
log4j.logger.io.debezium.connector.mysql.BinlogReader=DEBUG, stdout
log4j.logger.io.debezium.relational.history=INFO, stdout

log4j.additivity.io.debezium.connector.mysql=false
log4j.additivity.io.debezium.storage.kafka.history=false
log4j.additivity.io.debezium.connector.mysql.BinlogReader=false
...
```

11.3.2. 使用 Kafka Connect API 动态更改 Debezium 日志记录级别

您可以使用 Kafka Connect REST API 在运行时动态设置连接器的日志级别。与您在 log4j.properties 中设置的日志级别更改不同，通过 API 所做的更改会立即生效，且不需要重启 worker。

您在 API 中指定的日志级别设置只适用于接收请求的端点上的 worker。集群中其他 worker 的日志级别保持不变。

指定级别不会在 worker 重启后保留。要对日志记录级别进行持久更改，请在 log4j.properties 中设置日志级别，方法是 配置日志记录器 或添加 映射的诊断上下文。

流程

- 通过将 PUT 请求发送到指定以下信息的 admin/loggers 端点来设置日志级别：
  - 要更改日志级别的软件包。
  - 要设置的日志级别。

```
curl -s -X PUT -H "Content-Type:application/json"
http://localhost:8083/admin/loggers/io.debezium.connector.<connector_package>
-d '{"level": "<log_level>"}'
```

例如，要记录 Debezium MySQL 连接器的调试信息，请将以下请求发送到 Kafka Connect：

```
curl -s -X PUT -H "Content-Type:application/json"
http://localhost:8083/admin/loggers/io.debezium.connector.mysql -d '{"level":
"DEBUG"}'
```

11.3.3. 通过添加映射的诊断上下文来更改 Debezium 日志记录级别

大多数 Debezium 连接器（和 Kafka Connect worker）都使用多个线程来执行不同的活动。这可能会使查看日志文件变得困难，并只查找特定逻辑活动的日志消息。为了便于查找日志消息，Debebe 提供了几个 映射的诊断上下文 (MDC)，用于为每个线程提供额外的信息。

Debezium 提供以下 MDC 属性：

dbz.connectorType

连接器类型的简短别名。例如，MySql、Mongo、Postgre 等等。与 相同类型的 连接器关联的所有线程都使用相同的值，因此您可以使用它来查找给定类型的连接器生成的所有日志消息。

dbz.connectorName

连接器或数据库服务器的名称，如连接器配置中定义的。例如，产品、serverA 等等。与特定 连接器实例 关联的所有线程都使用相同的值，以便您可以查找特定连接器实例生成的所有日志消息。

dbz.connectorContext

作为在连接器任务中运行的独立线程运行的活动的短名称。例如，主、binlog、快照 等等。在某些情况下，当连接器为特定资源（如表或集合）分配线程时，可以改为使用该资源的名称。与连接器关联的每个线程都将使用不同的值，以便您可以查找与此特定活动关联的所有日志消息。

要为连接器启用 MDC，您可以在 log4j.properties 文件中配置附加程序。

流程

1. *打开 log4j.properties 文件。*

2. *配置附加程序以使用任何支持的 Debezium MDC 属性。*

   在以下示例中，stdout 附加程序被配置为使用这些 MDC 属性：

   log4j.properties

```
...
log4j.appender.stdout.layout.ConversionPattern=%d{ISO8601} %-5p
%X{dbz.connectorType}|%X{dbz.connectorName}|%X{dbz.connectorContext}  %m
[%c]%n
...
```

   上例中的配置会生成类似以下示例的日志消息：

```
...
2017-02-07 20:49:37,692 INFO   MySQL|dbserver1|snapshot  Starting snapshot for
jdbc:mysql://mysql:3306/?
useInformationSchema=true&nullCatalogMeansCurrent=false&useSSL=false&useUnic
ode=true&characterEncoding=UTF-8&characterSetResults=UTF-
8&zeroDateTimeBehavior=convertToNull with user 'debezium'
[io.debezium.connector.mysql.SnapshotReader]
2017-02-07 20:49:37,696 INFO   MySQL|dbserver1|snapshot  Snapshot is using user
'debezium' with these MySQL grants:
[io.debezium.connector.mysql.SnapshotReader]
2017-02-07 20:49:37,697 INFO   MySQL|dbserver1|snapshot   GRANT SELECT,
RELOAD, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON *.*
TO 'debezium'@'%'   [io.debezium.connector.mysql.SnapshotReader]
...
```

   日志中的每一行包括连接器类型（例如，MySQL）、连接器的名称（如 dbserver1），以及线程的活动(如快照)。

11.4. OPENSHIFT 中的 DEBEZIUM 日志记录

如果您在 OpenShift 上使用 Debezium，您可以使用 Kafka Connect 日志记录器配置 Debezium 日志记录器和日志记录级别。有关在 Kafka Connect 模式中配置日志属性的更多信息，请参阅在 OpenShift 中使用 AMQ Streams。

第 12 章 为应用程序配置 DEBEZIUM 连接器

当默认的 Debezium 连接器行为不适合您的应用程序时，您可以使用以下 Debezium 功能来配置您需要的行为。

Kafka Connect 自动主题创建

启用 Connect 在运行时创建主题，并根据名称将配置设置应用到这些主题。

avro serialization

支持将 Debezium PostgreSQL、MongoDB 或 SQL Server 连接器配置为使用 Avro 来序列化消息键和值，从而使更改事件记录用户更容易适应更改记录模式。

xref:configuring-notifications-to-report-connector-status

提供了一种机制，可以通过一组可配置的频道来公开有关连接器的状态信息。

CloudEvents converter

启用 Debezium 连接器来发出符合 CloudEvents 规格的更改事件记录。

向 Debezium 连接器发送信号

提供修改连接器行为或触发操作的方法，如启动临时快照。

12.1. 自定义 KAFKA CONNECT 自动主题创建

Kafka 提供了两种创建主题的机制。您可以为 Kafka 代理启用自动主题创建，并以 Kafka 2.6.0 开始，也可以启用 Kafka Connect 来创建主题。Kafka 代理使用 auto.create.topics.enable 属性来控制自动主题创建。在 Kafka Connect 中，topic.creation.enable 属性指定是否允许 Kafka Connect 创建主题。在这两种情况下，属性的默认设置都启用自动主题创建。

当启用自动主题创建时，如果 Debezium 源连接器为没有目标主题的表发出更改事件记录，则会在运行时创建该主题，因为事件记录被嵌套到 Kafka 中。

在代理和 Kafka Connect 中自动创建主题之间的区别

代理创建的主题仅限于共享单个默认配置。代理无法将唯一配置应用到不同的主题或一组主题。相反，Kafka Connect 可以在创建主题时应用任何多个配置，设置复制因素、分区数量和其他特定于主题的设置，如 Debezium 连接器配置中指定的。连接器配置定义了一组主题创建组，并将一组主题配置属性与每个组相关联。

代理配置和 Kafka Connect 配置相互独立。Kafka Connect 都可以创建主题，无论是否在代理中禁用主题创建。如果您在代理和 Kafka Connect 上启用自动主题创建，则 Connect 配置才会具有优先权，且代理仅在 Kafka Connect 配置中没有设置时创建主题。

如需更多信息，请参阅以下内容：

- *第 12.1.1 节 “为 Kafka 代理禁用自动主题创建”*
- *第 12.1.2 节 “在 Kafka Connect 中配置自动主题创建”*
- *第 12.1.3 节 “配置自动创建的主题”*
- *第 12.1.3.1 节 “主题创建组”*
- *第 12.1.3.2 节 “主题创建组配置属性”*
- *第 12.1.3.3 节 “指定 Debezium 默认主题创建组的配置”*
- *第 12.1.3.4 节 “指定 Debezium 自定义主题创建组的配置”*
- *第 12.1.3.5 节 “注册 Debezium 自定义主题创建组”*

12.1.1. 为 Kafka 代理禁用自动主题创建

默认情况下，如果主题尚不存在，Kafka 代理配置可让代理在运行时创建主题。代理创建的主题无法使用自定义属性配置。如果您使用早于 2.6.0 的 Kafka 版本，且您希望使用特定配置创建主题，则必须在代理中禁用自动创建主题，然后显式创建主题，或者通过自定义部署过程创建。

流程

- *在代理配置中，将 auto.create.topics.enable 的值设置为 false。*

12.1.2. 在 Kafka Connect 中配置自动主题创建

Kafka Connect 中的自动主题创建由 topic.creation.enable 属性控制。属性的默认值为 true，启用自动主题创建，如下例所示：

```
topic.creation.enable = true
```

topic.creation.enable 属性的设置应用到 Connect 集群中的所有 worker。

Kafka Connect 自动主题创建需要您定义 Kafka Connect 在创建主题时应用的配置属性。您可以通过定义主题组，然后在 Debezium 连接器配置中指定主题配置属性，然后指定要应用到每个组的属性。连接器配置定义了默认主题创建组，以及一个或多个自定义主题创建组。自定义主题创建组使用主题名称模式列表来指定组设置要应用到的主题。

有关 Kafka Connect 如何匹配主题到主题创建组的详情，请参阅 主题创建组。有关如何将配置属性分配给组的更多信息，请参阅 主题创建组配置属性。

默认情况下，Kafka Connect 创建的主题会根据模式 server.schema.table 命名，例如 dbserver.myschema.inventory。

流程

- 要防止 Kafka Connect 自动创建主题，在 Kafka Connect 自定义资源中将 topic.creation.enable 的值设置为 false，如下例所示：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster

...

spec:
  config:
    topic.creation.enable: "false"
```

注意

Kafka Connect 自动主题创建需要至少为 default 主题创建组设置 replication.factor 和 partitions 属性。组对组有效，可从 Kafka 代理的默认值获取所需属性的值。

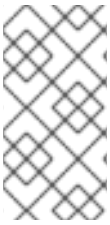
12.1.3. 配置自动创建的主题

对于 Kafka Connect 自动创建主题，它需要源连接器中有关配置属性的信息，以便在创建主题时应用。您可以在每个 Debezium 连接器的配置中定义控制主题创建的属性。当 Kafka Connect 为连接器发出的事件记录创建主题时，生成的主题会从适用的组获取其配置。该配置仅适用于该连接器发送的事件记录。

12.1.3.1. 主题创建组

一组主题属性与主题创建组关联。最小，您必须定义一个 默认 主题创建组并指定其配置属性。除此之外，您还可以定义一个或多个自定义主题创建组，并为每个组指定唯一属性。

在创建自定义主题创建组时，您可以根据主题名称模式为每个组定义成员主题。您可以指定描述每个组中包含或排除的主题的命名模式。include 和 exclude 属性包含用于定义主题名称模式的正则表达式列表。例如，如果您想一个组包含以字符串 dbserver1.inventory 开头的所有主题，请将 topic.creation.inventory.include 属性的值设置为 dbserver1\\.inventoryRaft。



注意

如果您同时为自定义主题组指定 include 和 exclude 属性，则排除规则具有优先权，并覆盖包含的规则。

12.1.3.2. 主题创建组配置属性

默认 主题创建组以及每个自定义组都与一组唯一的配置属性关联。您可以将组配置为包含任何 Kafka 主题级配置属性。例如，您可以为旧主题片段 指定清理策略，保留时间，或主题 组的主题压缩类型。您必须至少定义一组最小属性，来描述要创建的主题的配置。

如果没有注册自定义组，或者任何注册的组的 include 模式与要创建的任何主题的名称不匹配，则 Kafka Connect 使用默认组的配置来创建主题。

有关配置主题的常规信息，请参阅在 OpenShift 上安装 Debezium 中的 Kafka 主题创建建议。

12.1.3.3. 指定 Debezium 默认主题创建组的配置

在使用 Kafka Connect 自动主题创建前，您必须创建一个默认主题创建组并为它定义配置。默认主题创建组的配置应用于名称与自定义主题创建组的 include 列表模式匹配的任何主题。

先决条件

- *在 Kafka Connect 自定义资源中，metadata.annotations 中的 use-connector-resources 值指定集群 Operator 使用 KafkaConnector 自定义资源在集群中配置连接器。例如：*

```
...
  metadata:
    name: my-connect-cluster
    annotations: strimzi.io/use-connector-resources: "true"
...
```

流程

- *要为 topic.creation.default 组定义属性，请将它们添加到连接器自定义资源的 spec.config 中，如下例所示：*

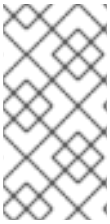
```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
metadata:
  name: inventory-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
...

  config:
...
    topic.creation.default.replication.factor: 3 (1)
    topic.creation.default.partitions: 10 (2)
    topic.creation.default.cleanup.policy: compact (3)
    topic.creation.default.compression.type: lz4 (4)
...
```

您可以在 default 组配置中包含任何 Kafka 主题级配置属性。

表 12.1. 默认 主题创建组的连接器配置

项	描述
1	**topic.creation.default.replication.factor** 定义了由默认组创建的主题的复制因素。**replication.factor** 对于 **default** 组是必需的，但对于自定义组是可选的。如果没有设置，自定义组将回退到**默认**组的值。使用 **-1** 使用 Kafka 代理的默认值。
2	**topic.creation.default.partitions** 为默认组创建的主题定义分区数量。**partitions** 对于**默认**组来说是强制的，但对于自定义组是可选的。如果没有设置，自定义组将回退到**默认**组的值。使用 **-1** 使用 Kafka 代理的默认值。

项	描述
3	**topic.creation.default.cleanup.policy** 映射到 主题级别配置参数 的 **cleanup.policy** 属性，并定义日志保留策略。
4	**topic.creation.default.compression.type** 映射到 主题级别配置参数 的 **compression.type** 属性，并定义如何在硬盘上压缩消息。



注意

自定义组仅回退到所需的 replication.factor 和 partitions 属性 的默认 组设置。如果自定义主题组的配置保留了其他属性未定义，则不会应用在默认组中指定的值。

12.1.3.4. 指定 Debezium 自定义主题创建组的配置

您可以定义多个自定义主题组，每个组都有自己的配置。

流程

- *要定义自定义主题组，在连接器自定义资源中添加一个 topic.creation.<group_name>.include property to spec.config，后跟您要应用到自定义组中的主题的配置属性。*

  以下示例显示了定义自定义主题创建组 inventory 和 applicationlogs 的自定义资源摘录：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
metadata:
  name: inventory-connector
...
spec:
...

  config:
... (1)
  topic.creation.inventory.include: dbserver1\\.inventory\\.* (2)
  topic.creation.inventory.partitions: 20
  topic.creation.inventory.cleanup.policy: compact
  topic.creation.inventory.delete.retention.ms: 7776000000

  (3)
  topic.creation.applicationlogs.include: dbserver1\\.logs\\.applog-.* (4)
  topic.creation.applicationlogs.exclude": dbserver1\\.logs\\.applog-old-.* (5)
```

```
    topic.creation.applicationlogs.replication.factor: 1
    topic.creation.applicationlogs.partitions: 20
    topic.creation.applicationlogs.cleanup.policy: delete
    topic.creation.applicationlogs.retention.ms: 7776000000
    topic.creation.applicationlogs.compression.type: lz4
...
...
```

表 12.2. 自定义 清单和 applicationlogs 主题创建组的连接器配置

项	描述
1	定义 **清单组** 的配置。 对于 **自定义组，copy.factor** 和 **partitions** 属性是可选的。如果没有设置值，自定义组会返回到 **default** 组设置的值。将值设为 **-1** 以使用为 Kafka 代理设置的值。
2	**topic.creation.inventory.include** 定义一个正则表达式，以匹配以 **dbserver1.inventory.** 开头的所有主题。为 **清单组** 定义的配置仅应用于名称与指定正则表达式匹配的主题。
3	定义 **applicationlogs** 组的配置。 对于 **自定义组，copy.factor** 和 **partitions** 属性是可选的。如果没有设置值，自定义组会返回到 **default** 组设置的值。将值设为 **-1** 以使用为 Kafka 代理设置的值。
4	**topic.creation.applicationlogs.include** 定义了一个正则表达式，以匹配以 **dbserver1.logs.applog-** 开头的所有主题。为 **applicationlogs** 组定义的配置仅应用于名称与指定正则表达式匹配的主题。由于此组也定义了 **exclude** 属性，所以与该 **exclude** 属性限制与 **include** 正则表达式匹配的主题可能会进一步限制。
5	**topic.creation.applicationlogs.exclude** 定义了一个正则表达式，以匹配以 **dbserver1.logs.applog-old-** 开头的所有主题。为 **applicationlogs** 组定义的配置仅应用于名称与给定正则表达式 **不匹配的** 主题。因为已为这个组定义了一个 **include** 属性，**applicationlogs** 组的配置仅应用于名称与指定的 **include** 正则表达式的名称匹配的，与指定的 **exclude** 正则表达式**不**匹配的主题。

12.1.3.5. 注册 Debezium 自定义主题创建组

在为任何自定义主题创建组指定配置后，注册组。

流程

- *通过将 topic.creation.groups 属性添加到连接器自定义资源，并指定以逗号分隔的自定义主题创建组列表来注册自定义组。*

  连接器自定义资源的以下摘录注册自定义主题创建组 inventory 和 applicationlogs ：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
```

```
metadata:
  name: inventory-connector
...
spec:
...

   config:
     topic.creation.groups: inventory,applicationlogs

...
```

完成的配置

以下示例显示了包含 默认 主题组的配置的已完成配置，以及 清单 的配置和 applicationlogs 自定义主题创建组：

示例：配置默认主题创建组和两个自定义组

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
metadata:
  name: inventory-connector
...
spec:
...

   config:
...
    topic.creation.default.replication.factor: 3,
    topic.creation.default.partitions: 10,
    topic.creation.default.cleanup.policy: compact
    topic.creation.default.compression.type: lz4
    topic.creation.groups: inventory,applicationlogs
    topic.creation.inventory.include: dbserver1\\.inventory\\.*
    topic.creation.inventory.partitions: 20
    topic.creation.inventory.cleanup.policy: compact
    topic.creation.inventory.delete.retention.ms: 7776000000
    topic.creation.applicationlogs.include: dbserver1\\.logs\\.applog-.*
    topic.creation.applicationlogs.exclude": dbserver1\\.logs\\.applog-old-.*
    topic.creation.applicationlogs.replication.factor: 1
    topic.creation.applicationlogs.partitions: 20
    topic.creation.applicationlogs.cleanup.policy: delete
    topic.creation.applicationlogs.retention.ms: 7776000000
    topic.creation.applicationlogs.compression.type: lz4
...
```

12.2. 配置 DEBEZIUM 连接器以使用 AVRO 序列化

Debezium 连接器在 Kafka Connect 框架中的工作原理，通过生成更改事件记录来捕获数据库中的每个行级更改。对于每个更改事件记录，Debezium 连接器完成以下操作：

1. 应用配置的转换。
2. 使用配置的 Kafka Connect 转换器 将记录键和值序列化为二进制形式。
3. 将记录写入正确的 Kafka 主题。

您可以为每个单独的 Debezium 连接器实例指定转换器。Kafka Connect 提供了一个 JSON 转换程序，可将记录键和值序列化为 JSON 文档。默认行为是 JSON 转换程序包含记录的消息模式，这使得每个记录非常详细。Debezium 入门指南 显示了当包含有效负载和模式时记录的内容。如果您希望记录使用 JSON 序列化，请考虑将以下连接器配置属性设置为 false ：

- key.converter.schemas.enable
- value.converter.schemas.enable

将这些属性设置为 false 可排除每个记录的详细模式信息。

或者，您可以使用 Apache Avro 来序列化记录键和值。Avro 二进制格式是紧凑有效的。通过 avro 模式，可以确保每个记录都有正确的结构。avro 的模式演进机制使模式能够演变。这对 Debezium 连接器至关重要，它会动态生成每个记录的模式，以匹配已更改的数据库表的结构。随着时间的推移，更改写入同一 Kafka 主题的事件记录可能具有相同的 schema 版本。avro serialization 可让更改事件记录的消费者更容易适应更改的记录模式。

要使用 Apache Avro serialization，您必须部署一个管理 Avro 消息模式及其版本的 schema registry。有关设置此 registry 的详情，请参考在 OpenShift 上安装和部署 Service Registry 的文档。

12.2.1. 关于 Service Registry

Service Registry

Service Registry 提供以下与 Avro 一起工作的组件：

- 您可以在 Debezium 连接器配置中指定 Avro 转换程序。这个转换器将 Kafka Connect 模式映射到 Avro 模式。然后，转换器使用 Avro 模式将记录键和值序列化为 Avro 的紧凑二进制形式。
- 一个 API 和 schema registry，用于跟踪：
  - Kafka 主题中使用的 Avro 模式。
  - 其中 Avro converter 发送生成的 Avro 模式。

  由于 Avro 模式存储在此 registry 中，因此每个记录需要仅包含 tiny 模式标识符。这使得每个记录更小。对于 Kafka 等 I/O 绑定系统，这意味着生产者和消费者的总吞吐量。
- Kafka 生成者和消费者的 avro Serdes （序列化器和反序列化器）。您编写以消耗更改事件记录的 Kafka 消费者应用程序可以使用 Avro Serdes 来反序列化更改事件记录。

要将 Service Registry 与 Debezium 搭配使用，请将 Service Registry 转换器及其依赖项添加到用于运行 Debezium 连接器的 Kafka Connect 容器镜像。

注意

Service Registry 项目还提供 JSON 转换程序。这个转换器将不太详细消息的好处与人类可读的 JSON 合并。消息不包含自身模式信息，而仅包含一个模式 ID。

注意

要使用 Service Registry 提供的转换器，您需要提供 apicurio.registry.url。

12.2.2. 部署使用 Avro 序列化的 Debezium 连接器概述

要部署使用 Avro 序列化的 Debezium 连接器，您必须完成三个主要任务：

1. 按照在 OpenShift 上安装和部署 Service Registry 中的说明，部署 Service Registry 实例。

2. 通过下载 Debezium Service Registry Kafka Connect zip 文件并将其提取到 Debezium 连接器的目录中，来安装 Avro 转换。

3. 通过设置配置属性，将 Debezium 连接器实例配置为使用 Avro 序列化，如下所示：

```
key.converter=io.apicurio.registry.utils.converter.AvroConverter
key.converter.apicurio.registry.url=http://apicurio:8080/apis/registry/v2
key.converter.apicurio.registry.auto-register=true
key.converter.apicurio.registry.find-latest=true
value.converter=io.apicurio.registry.utils.converter.AvroConverter
value.converter.apicurio.registry.url=http://apicurio:8080/apis/registry/v2
value.converter.apicurio.registry.auto-register=true
value.converter.apicurio.registry.find-latest=true
schema.name.adjustment.mode=avro
```

在内部，Kafka Connect 始终使用 JSON 键/值转换器来存储配置和偏移。

12.2.3. 在 Debezium 容器中部署使用 Avro 的连接器

在您的环境中，您可能想要使用提供的 Debezium 容器来部署使用 Avro 序列化的 Debezium 连接器。完成以下步骤，为 Debezium 构建自定义 Kafka Connect 容器镜像，并将 Debezium 连接器配置为使用 Avro converter。

先决条件

- 已安装 Docker 并有足够的权限来创建和管理容器。
- 您下载了您要使用 Avro 序列化部署的 Debezium 连接器插件。

流程

1. 部署 Service Registry 实例。请参阅在 OpenShift 上安装和部署 Service Registry，该 registry 提供了以下说明：

- 安装 Service Registry
- 安装 AMQ Streams
- 设置 AMQ Streams 存储

2. 提取 Debezium 连接器存档，以便为连接器插件创建目录结构。如果您为多个 Debezium 连接器下载并提取存档，则生成的目录结构类似以下示例：

```
tree ./my-plugins/
./my-plugins/
├── debezium-connector-mongodb
|   ├── ...
├── debezium-connector-mysql
|   ├── ...
├── debezium-connector-postgres
|   ├── ...
└── debezium-connector-sqlserver
    ├── ...
```

3. 将 Avro converter 添加到包含您要配置为使用 Avro 序列化的 Debezium 连接器的目录中：

   a. 进入 Red Hat Integration 下载站点 并下载 Service Registry Kafka Connect zip 文件。

   b. 将存档提取到所需的 Debezium 连接器目录中。

要将多个 Debezium 连接器配置为使用 Avro 序列化，请将存档提取到每个相关连接器类型的目录中。虽然提取存档到每个目录会复制文件，但这样做消除了冲突依赖项的可能性。

4. 创建并发布自定义镜像，以运行 Debezium 连接器，该连接器配置为使用 Avro 转换程序：

   a. 使用 registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0 作为基础镜像来创建一个新的 Dockerfile。在以下示例中，将 my-plugins 替换为插件目录的名称：

```
FROM registry.redhat.io/amq-streams-kafka-35-rhel8:2.5.0
USER root:root
```

```
COPY ./my-plugins/ /opt/kafka/plugins/
USER 1001
```

在 Kafka Connect 开始运行连接器前，Kafka Connect 会加载 /opt/kafka/plugins 目录中任何第三方插件。

b. 构建 docker 容器镜像。例如，如果您将您在上一步中创建的 docker 文件保存为 debezium-container-with-avro，则您将运行以下命令：

```
docker build -t debezium-container-with-avro:latest
```

c. 将自定义镜像推送到容器 registry 中，例如：

```
docker push <myregistry.io>/debezium-container-with-avro:latest
```

d. 指向新容器镜像。执行以下操作之一：

- 编辑 KafkaConnect 自定义资源的 KafkaConnect.spec.image 属性。如果设置，此属性覆盖 Cluster Operator 中的 STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE 变量。例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  image: debezium-container-with-avro
```

- 在 install/cluster-operator/050-Deployment-strimzi-cluster-operator.yaml 文件中，编辑 STRIMZI_DEFAULT_KAFKA_CONNECT_IMAGE 变量以指向新的容器镜像并重新安装 Cluster Operator。如果编辑此文件，则需要将其应用到 OpenShift 集群。

5. 部署配置为使用 Avro converter 的每个 Debezium 连接器。对于每个 Debezium 连接器：

a. 创建 Debezium 连接器实例。以下 inventory-connector.yaml 文件示例会创建一个 KafkaConnector 自定义资源，该资源定义配置为使用 Avro converter 的 MySQL 连接器实

例：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
metadata:
  name: inventory-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: io.debezium.connector.mysql.MySqlConnector
  tasksMax: 1
  config:
    database.hostname: mysql
    database.port: 3306
    database.user: debezium
    database.password: dbz
    database.server.id: 184054
    topic.prefix: dbserver1
    database.include.list: inventory
    schema.history.internal.kafka.bootstrap.servers: my-cluster-kafka-
bootstrap:9092
    schema.history.internal.kafka.topic: schema-changes.inventory
    schema.name.adjustment.mode: avro
    key.converter: io.apicurio.registry.utils.converter.AvroConverter
    key.converter.apicurio.registry.url: http://apicurio:8080/api
    key.converter.apicurio.registry.global-id:
io.apicurio.registry.utils.serde.strategy.GetOrCreateIdStrategy
    value.converter: io.apicurio.registry.utils.converter.AvroConverter
    value.converter.apicurio.registry.url: http://apicurio:8080/api
    value.converter.apicurio.registry.global-id:
io.apicurio.registry.utils.serde.strategy.GetOrCreateIdStrategy
```

b.

应用连接器实例，例如：

```
oc apply -f inventory-connector.yaml
```

这会注册 inventory-connector，连接器开始针对 inventory 数据库运行。

6.

验证连接器是否已创建并已启动，以跟踪指定数据库中的更改。您可以通过观察 Kafka Connect 日志输出来验证连接器实例，例如 inventory-connector 启动。

a.

显示 Kafka Connect 日志输出：

```
oc logs $(oc get pods -o name -l strimzi.io/name=my-connect-cluster-connect)
```

b.

检查日志输出，以验证初始快照是否已执行。您应该看到类似如下的行：

```
...
2020-02-21 17:57:30,801 INFO Starting snapshot for jdbc:mysql://mysql:3306/?
useInformationSchema=true&nullCatalogMeansCurrent=false&useSSL=false&use
Unicode=true&characterEncoding=UTF-8&characterSetResults=UTF-
8&zeroDateTimeBehavior=CONVERT_TO_NULL&connectTimeout=30000 with user
'debezium' with locking mode 'minimal'
(io.debezium.connector.mysql.SnapshotReader) [debezium-mysqlconnector-
dbserver1-snapshot]
2020-02-21 17:57:30,805 INFO Snapshot is using user 'debezium' with these MySQL
grants: (io.debezium.connector.mysql.SnapshotReader) [debezium-
mysqlconnector-dbserver1-snapshot]
...
```

执行快照涉及多个步骤：

```
...
2020-02-21 17:57:30,822 INFO Step 0: disabling autocommit, enabling repeatable
read transactions, and setting lock wait timeout to 10
(io.debezium.connector.mysql.SnapshotReader) [debezium-mysqlconnector-
dbserver1-snapshot]
2020-02-21 17:57:30,836 INFO Step 1: flush and obtain global read lock to prevent
writes to database (io.debezium.connector.mysql.SnapshotReader) [debezium-
mysqlconnector-dbserver1-snapshot]
2020-02-21 17:57:30,839 INFO Step 2: start transaction with consistent snapshot
(io.debezium.connector.mysql.SnapshotReader) [debezium-mysqlconnector-
dbserver1-snapshot]
2020-02-21 17:57:30,840 INFO Step 3: read binlog position of MySQL primary
server (io.debezium.connector.mysql.SnapshotReader) [debezium-
mysqlconnector-dbserver1-snapshot]
2020-02-21 17:57:30,843 INFO   using binlog 'mysql-bin.000003' at position '154'
and gtid '' (io.debezium.connector.mysql.SnapshotReader) [debezium-
mysqlconnector-dbserver1-snapshot]
...
2020-02-21 17:57:34,423 INFO Step 9: committing transaction
(io.debezium.connector.mysql.SnapshotReader) [debezium-mysqlconnector-
dbserver1-snapshot]
2020-02-21 17:57:34,424 INFO Completed snapshot in 00:00:03.632
(io.debezium.connector.mysql.SnapshotReader) [debezium-mysqlconnector-
dbserver1-snapshot]
...
```

完成快照后，Debebe 开始跟踪更改，例如，库存 数据库的 binlog 用于更改事件：

```
...
2020-02-21 17:57:35,584 INFO Transitioning from the snapshot reader to the binlog
reader (io.debezium.connector.mysql.ChainedReader) [task-thread-inventory-
connector-0]
```

```
2020-02-21 17:57:35,613 INFO Creating thread debezium-mysqlconnector-dbserver1-binlog-client (io.debezium.util.Threads) [task-thread-inventory-connector-0]
2020-02-21 17:57:35,630 INFO Creating thread debezium-mysqlconnector-dbserver1-binlog-client (io.debezium.util.Threads) [blc-mysql:3306]
Feb 21, 2020 5:57:35 PM com.github.shyiko.mysql.binlog.BinaryLogClient connect
INFO: Connected to mysql:3306 at mysql-bin.000003/154 (sid:184054, cid:5)
2020-02-21 17:57:35,775 INFO Connected to MySQL binlog at mysql:3306, starting at binlog file 'mysql-bin.000003', pos=154, skipping 0 events plus 0 rows (io.debezium.connector.mysql.BinlogReader) [blc-mysql:3306]
...
```

12.2.4. 关于 Avro 名称要求

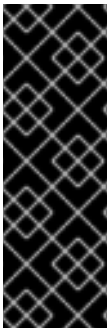
如 Avro 文档中所述，名称必须遵循以下规则：

- 以 [A-Za-z_]开始。
- 因此，仅包含 [A-Za-z0-9_] 字符

Debezium 使用列的名称作为对应 Avro 字段的基础。如果列名称不遵循 Avro 命名规则，这可能会导致序列化过程中出现问题。每个 Debezium 连接器都提供一个配置属性 field.name.adjustment.mode，如果您有没有遵循 Avro 规则的列，则可以将其设置为 avro。将 field.name.adjustment.mode 设置为 avro 允许对非格式字段进行序列化，而无需实际修改您的模式。

12.3. 以 CLOUDEVENTS 格式发出 DEBEZIUM 更改事件记录

CloudEvents 是以常见方式描述事件数据的规格。其目的是跨服务、平台和系统提供互操作性。Debezium 允许您配置 MongoDB、MySQL、PostgreSQL 或 SQL Server 连接器来发出符合 CloudEvents 规格的更改事件记录。



重要

以 CloudEvents 格式发出更改事件记录是一项技术预览功能。技术预览功能不被红帽产品服务级别协议(SLA)支持，且可能无法完成。因此，红帽不推荐在生产环境中实施任何技术预览功能。此技术预览功能为您提供对即将推出的产品创新的早期访问，允许您在开发过程中测试并提供反馈。如需有关支持范围的更多信息，请参阅 技术预览功能支持范围。

CloudEvents 规格定义：

- 一组标准化事件属性
- 定义自定义属性的规则
- 将事件格式映射到序列化表示的编码规则，如 JSON 或 Avro
- 用于传输层的协议绑定，如 Apache Kafka、HTTP 或 AMQP

要将 Debezium 连接器配置为发出符合 CloudEvents 规格的更改事件记录，Debebe 提供了 io.debezium.converters.CloudEventsConverter，它是一个 Kafka Connect 消息转换器。

目前，只支持结构化映射模式。CloudEvents 更改事件 envelope 可以是 JSON 或 Avro，每个 envelope 类型都支持 JSON 或 Avro 作为 数据格式。预计未来的 Debezium 发行版本将支持二进制映射模式。

有关以 CloudEvents 格式发出更改事件的信息如下：

- 第 12.3.1 节 “CloudEvents 格式的 Debezium 更改事件记录示例”
- 第 12.3.2 节 “配置 Debezium CloudEvents converter 示例”
- 第 12.3.3 节 “Debezium CloudEvents converter 配置选项”

有关使用 Avro 的详情，请参考：

- avro serialization
- Apicurio Registry

12.3.1. CloudEvents 格式的 Debezium 更改事件记录示例

以下示例显示了 PostgreSQL 连接器发送的 CloudEvents 更改事件记录类似如下。在本例中，PostgreSQL 连接器配置为使用 JSON 作为 CloudEvents 格式 envelope，也可以用作 数据格式。

```
{
  "id" : "name:test_server;lsn:29274832;txId:565",  1
  "source" : "/debezium/postgresql/test_server",  2
  "specversion" : "1.0",  3
  "type" : "io.debezium.postgresql.datachangeevent",  4
  "time" : "2020-01-13T13:55:39.738Z",  5
  "datacontenttype" : "application/json",  6
  "iodebeziumop" : "r",  7
  "iodebeziumversion" : "2.3.4.Final",  8
  "iodebeziumconnector" : "postgresql",
  "iodebeziumname" : "test_server",
  "iodebeziumtsms" : "1578923739738",
  "iodebeziumsnapshot" : "true",
  "iodebeziumdb" : "postgres",
  "iodebeziumschema" : "s1",
  "iodebeziumtable" : "a",
  "iodebeziumlsn" : "29274832",
  "iodebeziumxmin" : null,
  "iodebeziumtxid": "565",  9
  "iodebeziumtxtotalorder": "1",
  "iodebeziumtxdatacollectionorder": "1",
  "data" : {  10
    "before" : null,
    "after" : {
      "pk" : 1,
      "name" : "Bob"
    }
  }
}
```

1 1 1

连接器根据更改事件的内容为更改事件生成的唯一 ID。

2 2 2

事件源，这是由连接器配置中 topic.prefix 属性指定的数据库的逻辑名称。

3 3 3

CloudEvents 规格版本。

4 4 4

生成更改事件的连接器类型。此字段的格式是 io.debezium.CONNECTOR_TYPE.datachangeevent。CONNECTOR_TYPE 的值是

mongodb、mysql、postgresql 或 sqlserver。

5 5

源数据库中更改的时间。

6

描述 data 属性的内容类型，本例中为 JSON。唯一的替代方法是 Avro。

7

操作标识符。可能的值有 r 用于读取、c 表示 create、u 表示 update，或 d 用于 delete。

8

来自 Debezium 更改事件已知的所有 源 属性都会通过使用属性名称的 iodebezium 前缀映射到 CloudEvents 扩展属性。

9

在连接器中启用时，从 Debezium 更改事件已知的 事务 属性都会通过使用属性名称的 iodebeziumtx 前缀映射到 CloudEvents 扩展属性。

10

实际数据更改本身。根据操作和连接器，数据可能包含、after 和/或 patch 字段。

以下示例还显示了 PostgreSQL 连接器发出的 CloudEvents 更改事件记录是什么样子。在本例中，PostgreSQL 连接器再次配置为使用 JSON 作为 CloudEvents 格式 envelope，但这一次连接器被配置为使用 Avro 作为 数据格式。

```
{
  "id" : "name:test_server;lsn:33227720;txId:578",
  "source" : "/debezium/postgresql/test_server",
  "specversion" : "1.0",
  "type" : "io.debezium.postgresql.datachangeevent",
  "time" : "2020-01-13T14:04:18.597Z",
  "datacontenttype" : "application/avro",          1
  "dataschema" : "http://my-registry/schemas/ids/1", 2
  "iodebeziumop" : "r",
  "iodebeziumversion" : "2.3.4.Final",
  "iodebeziumconnector" : "postgresql",
  "iodebeziumname" : "test_server",
  "iodebeziumtsms" : "1578924258597",
  "iodebeziumsnapshot" : "true",
```

```
  "iodebeziumdb" : "postgres",
  "iodebeziumschema" : "s1",
  "iodebeziumtable" : "a",
  "iodebeziumtxId" : "578",
  "iodebeziumlsn" : "33227720",
  "iodebeziumxmin" : null,
  "iodebeziumtxid": "578",
  "iodebeziumtxtotalorder": "1",
  "iodebeziumtxdatacollectionorder": "1",
  "data" : "AAAAAAEAAglCAg=="                    3
}
```

1

表示 data 属性包含 Avro 二进制数据。

2

Avro 数据遵循的 schema 的 URI。

3

data 属性包含 base64 编码的 Avro 二进制文件数据。

也可以将 Avro 用于 envelope 和 data 属性。

12.3.2. 配置 Debezium CloudEvents converter 示例

在 Debezium 连接器配置中配置 io.debezium.converters.CloudEventsConverter。以下示例演示了如何配置 CloudEvents converter 来发出具有以下特征的更改事件记录：

- 使用 JSON 作为信封。
- 使用 http://my-registry/schemas/ids/1 中的 schema registry 将 data 属性序列化为二进制 Avro 数据。

```
...
"value.converter": "io.debezium.converters.CloudEventsConverter",
"value.converter.serializer.type" : "json",                    1
"value.converter.data.serializer.type" : "avro",
"value.converter.avro.schema.registry.url": "http://my-registry/schemas/ids/1"
...
```

1

指定 serializer.type 是可选的，因为 json 是默认值。

CloudEvents 转换器转换 Kafka 记录值。在同一个连接器配置中，如果要对记录键进行操作，您可以指定 key.converter。例如，您可以指定 StringConverter、LongConverter、JsonConverter 或 AvroConverter。

12.3.3. Debezium CloudEvents converter 配置选项

当您将 Debezium 连接器配置为使用 CloudEvent converter 时，您可以指定以下选项。

表 12.3. CloudEvents converter 配置选项的描述

选项	默认	描述
serializer.type	**json**	用于 CloudEvents envelope 结构的编码类型。该值可以是 **json** 或 **avro**。
data.serializer.type	**json**	用于 **data** 属性的编码类型。该值可以是 **json** 或 **avro**。
json. ...	N/A	使用 JSON 时要传递给底层转换器的任何配置选项。**json.** 前缀已被删除。
avro. ...	N/A	使用 Avro 时要传递给底层转换器的任何配置选项。**avro.** 前缀已被删除。例如，对于 Avro **数据**，您可以指定 **avro.schema.registry.url** 选项。
schema.name.adjustment.mode	none	指定应如何调整模式名称以与连接器使用的消息转换器兼容。该值可以是 **none** 或 **avro**。

12.4. 配置通知以报告连接器状态

Debezium 通知提供了一种机制来获取连接器的状态信息。通知可以发送到以下频道：

SinkNotificationChannel

通过 Connect API 向配置的主题发送通知。

LogNotificationChannel

通知附加到日志中。

JmxNotificationChannel

通知作为 JMX bean 中的属性公开。

有关 Debezium 通知的详情，请查看以下主题

- *第 12.4.1 节 “Debezium 通知的格式描述”*
- *第 12.4.2 节 “Debezium 通知的类型”*
- *第 12.4.3 节 “启用 Debezium 将事件发送到通知频道”*

12.4.1. Debezium 通知的格式描述

通知消息包含以下信息：

属性	Description
id	分配给通知的唯一标识符。对于增量快照通知，**id** 与 **execute-snapshot** 信号一同发送。
aggregate_type	与通知相关的聚合根的数据类型。在域驱动的设计中，导出的事件应始终引用聚合。
type	提供有关 **aggregate_type** 字段中指定的事件的状态信息。
additional_data	一个 Map<String,String>，其中包含有关通知的详细信息。例如，请参阅 Debezium 通知有关增量快照的进度。

12.4.2. Debezium 通知的类型

Debezium 通知提供有关 初始快照 或增量快照 进度的信息。

有关初始快照状态的 Debezium 通知

以下示例显示了提供初始快照状态的典型通知：

```
{
  "id": "5563ae14-49f8-4579-9641-c1bbc2d76f99",
  "aggregate_type": "Initial Snapshot",
  "type": "COMPLETED" 1
}
```

1

type 字段可以包含以下值之一：

- 完成
- 中止
- SKIPPED

12.4.2.1. 示例： Debezium 通知，该通知报告增量快照的进度

下表显示了在通知中报告增量快照状态的不同有效负载示例：

Status	payload

Start	{
	"id":"ff81ba59-15ea-42ae-b5d0-4d74f1f4038f",
	"aggregate_type":"Incremental Snapshot",
	"type":"STARTED",
	"additional_data":{
	"connector_name":"my-connector",
	"data_collections":"table1, table2"
	}
	}

Status	payload

paused	{
	"id":"068d07a5-d16b-4c4a-b95f-8ad061a69d51",
	"aggregate_type":"Incremental Snapshot",
	"type":"PAUSED",
	"additional_data":{
	"connector_name":"my-connector",
	"data_collections":"table1, table2"
	}
	}

resumed	{
	"id":"a9468204-769d-430f-96d2-b0933d4839f3",
	"aggregate_type":"Incremental Snapshot",
	"type":"RESUMED",
	"additional_data":{
	"connector_name":"my-connector",
	"data_collections":"table1, table2"
	}
	}

stopped	{
	"id":"83fb3d6c-190b-4e40-96eb-f8f427bf482c",
	"aggregate_type":"Incremental Snapshot",
	"type":"ABORTED",
	"additional_data":{
	"connector_name":"my-connector"
	}
	}

Status	payload

处理块	<pre>{
	"id":"d02047d6-377f-4a21-a4e9-cb6e817cf744",
	"aggregate_type":"Incremental Snapshot",
	"type":"IN_PROGRESS",
	"additional_data":{
	"connector_name":"my-connector",
	"data_collections":"table1, table2",
	"current_collection_in_progress":"table1",
	"maximum_key":"100",
	"last_processed_key":"50"
	}
	}</pre>

快照为表完成	<pre>{
	"id":"6d82a3ec-ba86-4b36-9168-7423b0dd5c1d",
	"aggregate_type":"Incremental Snapshot",
	"type":"TABLE_SCAN_COMPLETED",
	"additional_data":{
	"connector_name":"my-connector",
	"data_collection":"table1, table2",
	"scanned_collection":"table1",
	"total_rows_scanned":"100",
	"status":"SUCCEEDED" (1)
	}
	}</pre>
	(1) 可能的值有：
	• EMPTY - 表为空
	• NO_PRIMARY_KEY - 表没有快照所需的主密钥
	• SKIPPED - 不支持此类表的快照，检查日志以了解详情
	• SQL_EXCEPTION - SQL 异常，同时处理快照
	• SUCCEEDED - 快照成功完成
	• UNKNOWN_SCHEMA - 未找到模式，检查日志中的已知表列表

Status	payload

完成	<pre>{
	"id":"6d82a3ec-ba86-4b36-9168-7423b0dd5c1d",
	"aggregate_type":"Incremental Snapshot",
	"type":"COMPLETED",
	"additional_data":{
	"connector_name":"my-connector"
	}
	}</pre>

12.4.3. 启用 Debezium 将事件发送到通知频道

要启用 Debezium 来发出通知，请通过设置 notification.enabled.channels 配置属性来指定通知频道列表。默认情况下，提供了以下通知频道：

- sink
- log
- jmx

重要

要使用 sink 通知频道，还必须将 notification.sink.topic.name 配置属性设置为您要 Debezium 发送通知的主题名称。

12.4.3.1. 启用 Debezium 通知以报告通过 JMX Bean 公开的事件

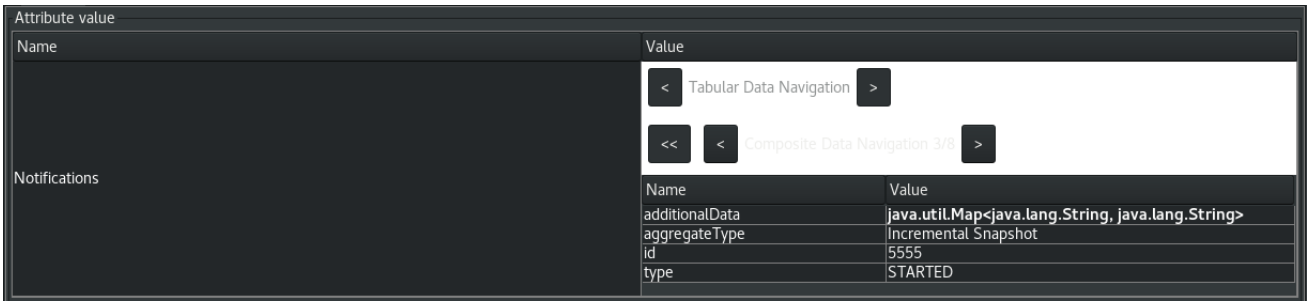
要启用 Debezium 报告通过 JMX Bean 公开的事件，请完成以下步骤：

1. 启用 JMX MBean 服务器 以公开通知 bean。
2. 在连接器配置中的 notification.enabled.channels 属性中添加 jmx。

3. 将您的首选 JMX 客户端连接到 MBean 服务器。

通知通过带有名称 debezium. <connector-type> .management.notifications. < server > 的 bean 的 Notifications 属性公开。

下图显示了报告增量快照开始的通知：



要丢弃通知，请在 bean 上调用 reset 操作。

该通知也作为 JMX 通知公开，类型为 debezium.notification。要让应用程序侦听 MBean 发出的 JMX 通知，请将 应用程序订阅到通知。

12.5. 向 DEBEZIUM 连接器发送信号

Debezium 信号机制提供了一种修改连接器行为的方法，或者触发一次性操作，如启动表 的临时快照。要使用信号来触发连接器来执行指定操作，您可以将连接器配置为使用以下一个或多个频道：

SourceSignalChannel

您可以发出 SQL 命令，来向专用信号数据收集添加信号消息。在源数据库中创建的信号数据收集被指定为与 Debezium 通信。

KafkaSignalChannel

您可以将信号信息提交至可配置的 Kafka 主题。

JmxSignalChannel

您可以通过 JMX 信号操作提交 信号。当 Debezium 检测到新的 日志记录 或临时快照记录 被添加到频道时，它会读取信号并启动请求的操作。

信号可用于以下 Debezium 连接器：

- Db2
- MongoDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server

您可以通过设置 signal.enabled.channels 配置属性来指定启用哪个频道。属性列出启用的频道的名称。默认情况下，Debebebe 提供以下频道： source 和 kafka。源 频道会被默认启用，因为增量快照信号需要它。

12.5.1. 启用 Debezium 源信号频道

默认情况下启用 Debezium 源信号频道。

您必须明确为您要使用它的每个连接器配置信号。

流程

1. 在源数据库中，创建一个信号数据收集表来向连接器发送信号。有关信号数据收集所需的结构的详情，请参考 信号数据收集的结构。
2. 对于实现原生更改数据捕获 (CDC) 机制的源数据库，请为信号表启用 CDC。
3.

将信号数据收集的名称添加到 Debezium 连接器配置中。
在连接器配置中，添加属性 signal.data.collection，并将其值设置为在第 1 步中创建的信号数据收集的完全限定名称。

例如, signal.data.collection = inventory.debezium_signals。

信号集合的完全限定名称的格式取决于连接器。
以下示例显示了每个连接器使用的命名格式：

Db2

<schemaName>.<tableName>

MongoDB

<databaseName>.<collectionName>

MySQL

<databaseName>.<tableName>

Oracle

<databaseName>.<schemaName>.<tableName>

PostgreSQL

<schemaName>.<tableName>

SQL Server

<databaseName>.<schemaName>.<tableName>

如需有关设置 signal.data.collection 属性的信息，请参阅您的连接器的配置属性列表。

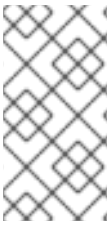
12.5.1.1. Debezium 信号数据收集的必要结构

信号数据收集或信号表会存储您发送到连接器的信号，以触发指定操作。信号表的结构必须符合以下标准格式：

- 包含三个字段（列）。
- 字段按特定顺序排列，如 表 1 所示。

表 12.4. 信号数据收集的必要结构

字段	类型	描述
ID（必需）	**字符串**	标识信号实例的任意唯一字符串。 您可以为提交到信号表的每个信号分配一个 **id**。 通常，ID 是一个 UUID 字符串。 您可以使用信号实例来记录、调试或重复数据删除。 当信号触发 Debezium 来执行增量快照时，它会生成带有任意 **id** 字符串的信号消息。生成的消息包含的 **id** 字符串与提交信号中的 **id** 字符串无关。
类型（必需）	**字符串**	指定要发送的信号类型。 您可以将一些信号类型与任何连接器搭配使用，而其他信号类型则仅适用于特定的连接器。
数据（可选）	**字符串**	指定 JSON 格式的参数以传递给信号操作。 每种信号类型都需要一组特定的数据。



注意

数据收集中的字段名称是任意的。前面的表中提供了推荐的名称。如果您使用不同的命名约定，请确保每个字段中的值与预期内容一致。

12.5.1.2. 创建 Debezium 信号数据收集

您可以通过向源数据库提交标准 SQL DDL 查询来创建信号表。

前提条件

- ***您有足够的权限在源数据库中创建表。***

流程

- ***向源数据库提交一个 SQL 查询以创建一个表，它与 required structure 一致，如以下示例所示：***

```
CREATE TABLE <tableName> (id VARCHAR(<varcharValue>) PRIMARY KEY, type
VARCHAR(<varcharValue>) NOT NULL, data VARCHAR(<varcharValue>) NULL);
```

注意

您分配给 id 变量的 VARCHAR 参数的空间量必须足够，以适应发送到信号表的信号 ID 字符串的大小。
如果 ID 的大小超过可用空间，则连接器无法处理信号。

以下示例显示了一个 CREATE TABLE 命令，它会创建一个三列 debezium_signal 表：

```
CREATE TABLE debezium_signal (id VARCHAR(42) PRIMARY KEY, type VARCHAR(32) NOT NULL, data VARCHAR(2048) NULL);
```

12.5.2. 启用 Debezium Kafka 信号频道

您可以通过将其添加到 signal.enabled.channels 配置属性来启用 Kafka 信号频道，然后将接收信号的主题名称添加到 signal.kafka.topic 属性中。启用信号频道后，会创建一个 Kafka 使用者来消耗发送到配置的信号主题的信号。

可供消费者使用的额外配置

- Db2 连接器 Kafka 信号配置属性
- MongoDB 连接器 Kafka 信号配置属性
- MySQL 连接器 Kafka 信号配置属性
- Oracle 连接器 Kafka 信号配置属性
- PostgreSQL 连接器 Kafka 信号配置属性
- SQL Server 连接器 Kafka 信号配置属性

注意

要使用 Kafka 信号为连接器触发临时增量快照，您必须首先在连接器配置 中启用 源 信号频道。源频道实施一个水位线机制，用于去除被增量快照捕获的事件，然后在流恢复后再次捕获。

消息格式

Kafka 消息的键必须与 topic.prefix 连接器配置选项的值匹配。

该值是一个带有 type 和 data 字段的 JSON 对象。

当信号类型设置为 execute-snapshot 时，data 字段必须包含下表中列出的字段：

表 12.5. 执行快照数据字段

字段	默认	值
type	**incremental**	要运行的快照的类型。目前，Debeium 仅支持 **增量** 类型。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与要包含在快照中的表的完全限定名称匹配。 使用与 **signal.data.collection** 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的记录子集的条件。

以下示例显示了典型的 execute-snapshot Kafka 信息：

```
Key = `test_connector`

Value = `{"type":"execute-snapshot","data": {"data-collections": ["schema1.table1", "schema1.table2"], "type": "INCREMENTAL"}}`
```

12.5.3. 启用 Debezium JMX 信号频道

您可以通过在连接器配置中的 signal.enabled.channels 属性中添加 JMX 信号来启用 JMX 信号，然后启用 JMX MBean 服务器 来公开信号 bean。

流程

1. *使用您首选的 JMX 客户端（例如：JConsole 或 JDK Mission Control，以连接到 MBean 服务器。*

2. *搜索 Mbean debezium. <connector-type>.management.signals. <server>。Mbean 公开接受以下输入参数 的信号 操作：*

   p0

   信号的 id。

   p1

   信号的类型，如 execute-snapshot。

   p2

   包含指定信号类型的附加信息的 JSON 数据字段。

3. *通过提供输入参数的值来发送 execute-snapshot 信号。*
在 JSON 数据字段中，包含下表中列出的信息：

表 12.6. 执行快照数据字段

字段	默认	值
type	**incremental**	要运行的快照的类型。目前，Debeium 仅支持 **增量** 类型。
data-collections	*N/A*	以逗号分隔的正则表达式数组，与要包含在快照中的表的完全限定名称匹配。 使用与 signal.data.collection 配置选项所需的格式相同的格式指定名称。
additional-condition	*N/A*	可选字符串，指定连接器评估为指定要包含在快照中的记录子集的条件。

下图显示了如何使用 JConsole 发送信号的示例：


Operation invocation

void signal (p0 5555 , p1 execute-snapshot , p2 {"data-collections")

MBeanOperationInfo

Name	Value
Operation:	
Name	signal
Description	signal
Impact	UNKNOWN
ReturnType	void
Parameter-0:	
Name	p0
Description	p0
Type	java.lang.String
Parameter-1:	
Name	p1
Description	p1
Type	java.lang.String

Descriptor

Name	Value
Operation:	
openType	javax.management.openmbean.SimpleType(name=java.lang.Void)
originalType	void
Parameter-0:	
openType	javax.management.openmbean.SimpleType(name=java.lang.String)
originalType	java.lang.String
Parameter-1:	
openType	javax.management.openmbean.SimpleType(name=java.lang.String)
originalType	java.lang.String
Parameter-2:	
openType	javax.management.openmbean.SimpleType(name=java.lang.String)
originalType	java.lang.String

12.5.4. Debezium 信号操作的类型

您可以使用信号启动以下操作：

- 添加消息到日志。
- 触发临时快照。
- 停止执行临时快照。
- 暂停增量快照。
- 恢复异常快照。

有些信号与所有连接器不兼容。

12.5.4.1. 日志记录信号

您可以通过创建带有日志信号类型的信号表条目来请求连接器来向 日志 添加条目。处理信号后，连接器会将指定的消息输出到日志。另外，您可以配置信号，以便生成的消息包含流协调。

表 12.7. 添加日志消息的信号记录示例

column	值	描述
id	**924e3ff8-2245-43ca-ba77-2af9af02fa07**	
type	**log**	信号的操作类型。
data	**{"message": "Signal message at offset {}"}**	**message** 参数指定要打印到日志的字符串。 如果您在消息中添加占位符(**{}**)，它将被流传输协调替换。

12.5.4.2. 临时快照信号

您可以通过创建一个带有 execute-snapshot 信号类型的信号来请求连接器来启动临时快照。处理信号后，连接器运行请求的快照操作。

与连接器首次启动后运行的初始快照不同，在连接器已经开始流更改事件后会在运行时发生临时快照。您可以随时启动临时快照。

临时快照可用于以下 Debezium 连接器：

- *Db2*
- *MongoDB*
- *MySQL*
- *Oracle*

- *PostgreSQL*
- *SQL Server*

表 12.8. 临时快照信号记录示例

column	值
id	**d139b9b7-7777-4547-917d-e1775ea61d41**
type	**execute-snapshot**
data	**{"data-collections": ["public.MyFirstTable", "public.MySecondTable"]}**

表 12.9. 临时快照信号消息示例

键	值
test_connector	**{"type":"execute-snapshot","data": {"data-collections": ["public.MyFirstTable"], "type": "INCREMENTAL", "additional-condition":"color='blue' AND brand='MyBrand'"}}**

有关临时快照的更多信息，请参阅您的连接器文档中的 Snapshots 主题。

其他资源

- *Db2 连接器临时快照*
- *MongoDB 连接器临时快照*
- *MySQL 连接器临时快照*
- *Oracle 连接器临时快照*
- *PostgreSQL 连接器临时快照*

- SQL Server 连接器临时快照

临时快照停止信号

您可以通过创建一个带有 stop-snapshot 信号类型的信号表条目来请求连接器来停止 in-progress 临时快照。处理信号后，连接器将停止当前的 in-progress 快照操作。

您可以停止以下 Debezium 连接器的临时快照：

- Db2
- MongoDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server

表 12.10. 停止临时快照信号记录的示例

column	值
id	d139b9b7-7777-4547-917d-e1775ea61d41
type	stop-snapshot
data	{"type":"INCREMENTAL", "data-collections": ["public.MyFirstTable"]}

您必须指定信号 的类型。data-collections 字段是可选的。将 data-collections 字段留空，以请求连接器停止当前快照中的所有活动。如果您希望增量快照继续，但您想要从快照中排除特定的集合，请提供要排除的集合或正则表达式的名称列表。连接器处理信号后，增量快照会进行，但它从您指定的集合中排除数据。

12.5.4.3. 增量快照

增量快照是特定类型的临时快照。在增量快照中，连接器捕获您指定的表的基准状态，类似于初始快照。但是，与初始快照不同，增量快照会捕获块中的表，而不是一次捕获表。连接器使用水位线方法来跟踪快照的进度。

通过捕获块中指定表的初始状态，而不是在单个单体操作中捕获，与初始快照进程相比，增量快照具有以下优势：

- 虽然连接器捕获指定表的基准状态，但事务日志中接近实时事件流将继续不间断。
- 如果增量快照进程中断，可以从其停止的时间点恢复。
- 您可以随时启动增量快照。

增量快照暂停信号

您可以通过创建一个带有 pause-snapshot 信号类型的信号表条目来请求连接器来暂停 in-progress 增量快照。处理信号后，连接器将停止暂停当前 in-progress 快照操作。因此，无法指定数据收集，因为快照处理将在处理信号时的位置暂停。

您可以暂停以下 Debezium 连接器的增量快照：

- Db2
- MongoDB
- MySQL
- Oracle
- PostgreSQL

- *SQL Server*

表 12.11. 暂停增量快照信号记录的示例

column	值
id	**d139b9b7-7777-4547-917d-e1775ea61d41**
type	**pause-snapshot**

您必须指定信号 的类型。data 字段将被忽略。

增量快照恢复信号

您可以通过创建带有 resume-snapshot 信号类型的信号表条目来请求连接器来恢复暂停的增量快照。处理信号后，连接器将恢复之前暂停的快照操作。

您可以为以下 Debezium 连接器恢复增量快照：

- *Db2*
- *MongoDB*
- *MySQL*
- *Oracle*
- *PostgreSQL*
- *SQL Server*

表 12.12. 恢复增量快照信号记录的示例

column	值
id	**d139b9b7-7777-4547-917d-e1775ea61d41**
type	**resume-snapshot**

您必须指定信号 的类型。data 字段将被忽略。

有关增量快照的更多信息，请参阅您的连接器文档中的 Snapshots 主题。

其他资源

- *Db2 连接器增量快照*
- *MongoDB 连接器增量快照*
- *MySQL 连接器增量快照*
- *Oracle 连接器增量快照*
- *PostgreSQL 连接器增量快照*
- *SQL Server 连接器增量快照*

第 13 章 应用转换以修改使用 APACHE KAFKA 交换的消息

Debezium 提供多个消息转换(SMT)，可用于修改事件记录。您可以配置连接器来应用一个转换，在将记录发送到 Apache Kafka 前修改记录。您还可以将 Debezium SMT 应用到接收器连接器，以便在连接器从 Kafka 主题读取记录前修改记录。

如果要只把转换应用到特定的消息，您可以配置 Kafka Connect predicate 来定义应用 SMT 的条件。

Debezium 提供以下 SMT：

主题路由器 SMT

根据应用于原始主题名称的正则表达式，将事件记录重新设置为特定主题。

基于内容的路由器 SMT

根据事件内容重新路由指定的更改事件记录。

事件记录更改 SMT

增强事件消息，以识别值在数据库操作后更改或保持不变的字段

消息过滤 SMT

允许您将事件记录的子集传播到目标 Kafka 主题。转换对连接器发出的更改事件记录（根据事件记录的内容）应用正则表达式。只有与表达式匹配的记录才会写入目标主题。其他记录将被忽略。

HeaderToValue SMT

从事件记录中提取指定的标头字段，然后将标头字段复制或移到事件记录中的值。

新的记录状态提取 SMT

将 Debezium 更改事件记录的复杂结构扁平化为简化的格式。简化的结构允许由接收器连接器处理，这些连接器无法使用原始结构。

MongoDB 新记录状态提取

简化 Debezium MongoDB 连接器更改事件记录的复杂结构。简化的结构允许由接收器连接器处理，这些连接器无法使用原始事件结构。

outbox 事件路由器 SMT

提供对 outbox 模式的支持，以启用在多个服务间的安全可靠数据交换。

MongoDB outbox 事件路由器 SMT

支持将 outbox 模式与 MongoDB 连接器一起使用，以启用在多个服务间的安全可靠数据交换。

分区路由 SMT

根据一个或多个指定有效负载字段的值将事件路由到特定的目标分区。

13.1. 使用 SMT PREDICATES 有选择地应用转换

当您为连接器配置单个消息转换(SMT)时，您可以为转换定义 predicate。predicate 指定如何将转换条件应用到连接器进程的消息的子集。您可以分配 predicates 来转换您为源连接器配置，如 Debezium 或 sink 连接器。

13.1.1. 关于 SMT predicates

Debezium 提供多个消息转换(SMT)，可用于在 Kafka Connect 将记录保存到 Kafka 主题前修改事件记录。默认情况下，当您为 Debezium 连接器配置其中一个 SMT 时，Kafka Connect 会将该转换应用到连接器发出的每个记录。但是，您可能有实例有选择地应用转换，以便它只修改共享共同特征的事件消息的子集。

例如，对于 Debezium 连接器，您可能希望仅在来自特定表的事件消息或包含特定标头键的事件信息上运行转换。在运行 Apache Kafka 2.6 或更高版本的环境中，您可以将 predicate 语句附加到转换中，以指示 Kafka Connect 只将 SMT 应用到特定的记录。在 predicate 中，您可以指定 Kafka Connect 用来评估它处理的每个消息的条件。当 Debezium 连接器发出更改事件消息时，Kafka Connect 会根据配置的 predicate 条件检查消息。如果事件消息条件为 true，Kafka Connect 会应用转换，然后将消息写入 Kafka 主题。与条件不匹配的消息会不修改地发送到 Kafka。

这种情况与您为接收器连接器 SMT 定义的 predicates 类似。连接器从 Kafka 主题读取信息，Kafka Connect 根据 predicate 条件评估信息。如果消息与条件匹配，Kafka Connect 会应用转换，然后将信息传递给接收器连接器。

在定义了 predicate 后，您可以重复使用它并将其应用到多个转换。predicates 还包括可用于反转 predicate 的 negate 选项，以便将 predicate 条件应用到与 predicate 语句中定义的条件 不匹配的 记录。您可以使用 negate 选项将 predicate 与基于负条件的其他转换配对。

predicate 元素

predicates 包含以下元素：

- *predicates 前缀*
- *alias （例如， isOutboxTable）*
- *type （例如， org.apache.kafka.connect.transforms.predicates.TopicNameMatches）。Kafka Connect 提供了一组默认的 predicate 类型，您可以通过定义自己的自定义 predicates 来补充。*
- *condition 语句和任何其他配置属性，具体取决于 predicate 的类型（例如，正则表达式命名模式）*

默认 predicate 类型

默认提供以下 predicate 类型：

HasHeaderKey

在您希望 Kafka Connect 要评估的事件消息中指定一个键名称。对于包含具有指定名称的标头键的任何记录，predicate 会评估为 true。

RecordIsTombstone

匹配 Kafka tombstone 记录。对于具有 null 值的任何记录，predicate 会评估为 true。将此 predicate 与过滤器 SMT 结合使用，以删除 tombstone 记录。这个 predicate 没有配置参数。

Kafka 中的 tombstone 是一个记录，它有一个带有 0 字节、null 有效负载的密钥。当 Debezium 连接器在源数据库中处理 delete 操作时，连接器会为 delete 操作发出两个更改事件：

- *提供数据库记录的先前值的删除操作(op" : "d")事件。*
- *有一个 tombstone 事件，它具有相同的键，但有一个 null 值。*

tombstone 代表行的一个删除标记。当为 Kafka 启用 日志 压缩时，在压缩 Kafka 过程中会删除与 tombstone 共享的所有事件。日志压缩会定期进行，由主题的

delete.retention.ms 设置控制的压缩间隔。

虽然可以 配置 Debezium，使其不发出 tombstone 事件，但最好允许 Debezium 发出 tombstones 在日志压缩过程中维护预期的行为。限制 tombstones 可防止 Kafka 在日志压缩过程中删除删除密钥的记录。如果您的环境包含无法处理 tombstones 的接收器连接器，您可以将 sink 连接器配置为使用带有 RecordIsTombstone predicate 的 SMT 来过滤 tombstone 记录。

TopicNameMatches

指定您要 Kafka Connect 匹配的主题名称的正则表达式。连接器名称与指定正则表达式匹配的连接器记录为 true。使用此 predicate 根据源表的名称将 SMT 应用到记录。

其他资源

- KIP-585: Filter 和 Conditional SMTs
- Kafka Connect predicates 的 Apache Kafka 文档

13.1.2. 定义 SMT predicates

默认情况下，Kafka Connect 会将 Debezium 连接器配置中的每个单一消息转换应用到它从 Debezium 接收的每个更改事件记录。从 Apache Kafka 2.6 开始，您可以在控制 Kafka Connect 应用转换的连接器配置中定义 SMT predicate。predicate 语句定义了 Kafka Connect 将转换应用到 Debezium 发出的事件记录的条件。Kafka Connect 评估 predicate 语句，然后有选择地将 SMT 应用到与 predicate 中定义的条件匹配的记录子集。配置 Kafka Connect predicates 与配置转换类似。您可以指定 predicate 别名，将别名与转换关联，然后定义 predicate 的类型和配置。

先决条件

- Debezium 环境运行 Apache Kafka 2.6 或更高版本。
- 为 Debezium 连接器配置 SMT。

流程

1. 在 Debezium 连接器配置中，为 predicates 参数指定 predicate 别名，例如 IsOutboxTable。

2. 通过将 predicate 别名附加到连接器配置中的转换中，将 predicate 别名与您要有条件地应用的转换相关联：

```
transforms.<TRANSFORM_ALIAS>.predicate=<PREDICATE_ALIAS>
```

例如：

```
transforms.outbox.predicate=IsOutboxTable
```

3. 通过指定类型并为配置参数提供值来配置 predicate。

   a. 对于类型，指定 Kafka Connect 中提供的以下默认类型之一：

   - HasHeaderKey
   - RecordIsTombstone
   - TopicNameMatches

   例如：

   ```
   predicates.IsOutboxTable.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
   ```

   b. 对于 TopicNameMatch 或 HasHeaderKey predicates，请为您要匹配的主题或标头名称指定一个正则表达式。

   例如：

   ```
   predicates.IsOutboxTable.pattern=outbox.event.*
   ```

4. 如果要对条件进行求值，请将 negate 关键字附加到转换别名，并将它设为 true。

例如：

```
transforms.outbox.negate=true
```

前面的属性会反转 predicate 匹配的记录集合，以便 Kafka Connect 将转换应用到与 predicate 中指定的条件不匹配的任何记录。

示例： outbox 事件路由器转换的 TopicNameMatch predicate

以下示例显示了一个 Debezium 连接器配置，它只会将 outbox 事件路由器转换为 Debezium 发送到 Kafka outbox.event.order 主题的消息。

由于只有对于来自 outbox 表 (outbox.event.*) 的信息 TopicNameMatch predicate 被评估为 true，所以源来自数据库中其他表的消息不会进行转换。

```
transforms=outbox
transforms.outbox.predicate=IsOutboxTable
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
predicates=IsOutboxTable
predicates.IsOutboxTable.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
predicates.IsOutboxTable.pattern=outbox.event.*
```

13.1.3. 忽略 tombstone 事件

您可以控制 Debezium 是否发出 tombstone 事件，以及 Kafka 如何保留它们。根据您的数据管道，您可能需要为连接器设置 tombstones.on.delete 属性，以便 Debezium 不会发出 tombstone 事件。

是否启用 Debezium 来发出 tombstones 取决于环境中如何消耗主题，以及 sink 消费者的特性。一些接收器(sink)连接器依赖于 tombstone 事件从下游数据存储中删除记录。如果接收器连接器依赖 tombstone 记录来指示何时删除下游数据存储中的记录，请将 Debezium 配置为发出它们。

当您将 Debezium 配置为生成 tombstones 时，需要进一步配置以确保接收器连接器接收 tombstone 事件。必须设置主题的保留策略，以便连接器在 Kafka 在日志压缩过程中删除事件信息前有时间读取事件信息。主题在压缩前保留 tombstones 的时间长度由主题的 delete.retention.ms 属性控制。

默认情况下，连接器的 tombstones.on.delete 属性被设置为 true，以便连接器在每个 delete 事件后生成一个 tombstone。如果将属性设置为 false 以防止 Debezium 将 tombstone 记录保存到 Kafka 主题，没有 tombstone 记录可能会导致意外的后果。Kafka 在日志压缩过程中依赖于 tombstone 来删除与已删除密钥相关的记录。

如果您需要支持无法使用 null 值处理记录的接收器连接器或下游 Kafka 用户，而不是防止 Debezium 发出 tombstones，请考虑使用 RecordIsTombstone predicate 类型在消费者读取它们前删除 tombstone 信息。

流程

- 要防止 Debezium 为删除的数据库记录发出 tombstone 事件，请将连接器的 tombstones.on.delete 选项设置为 false。

  例如：

  ```
  “tombstones.on.delete”: “false”
  ```

13.2. 将 DEBEZIUM 事件记录路由到您指定的主题

每个包含数据更改事件的 Kafka 记录都有一个默认目的地主题。如果需要，您可以将记录重新路由到您在记录到达 Kafka Connect 转换前指定的主题。要做到这一点，Debebebe 提供主题路由单一消息转换(SMT)。在 Debezium 连接器的 Kafka Connect 配置中配置此转换。配置选项允许您指定以下内容：

- 用于标识要重新路由的记录的表达式
- 解析到目标主题的表达式
- 如何确保将记录中的唯一密钥重新路由到目标主题

您要确保转换配置提供了您想要的行为。Debezium 不会验证您的转换配置的结果行为。

主题路由转换是一个 Kafka Connect SMT。

以下主题提供详情：

- 第 13.2.1 节 “将 Debezium 记录路由到您指定的主题用例”

- 第 13.2.2 节 “多个表的路由 Debezium 记录示例”
- 第 13.2.3 节 “确保路由到同一主题的 Debezium 记录间的唯一键”
- 第 13.2.5 节 “配置 Debezium 主题路由转换的选项”

13.2.1. 将 Debezium 记录路由到您指定的主题用例

默认行为是，Debezium 连接器会将每个更改事件记录发送到一个主题，其名称是从数据库的名称以及进行更改的表的名称。换句话说，主题接收一个物理表的记录。当希望一个主题接收多个物理表的记录时，您必须将 Debezium 连接器配置为将记录重新路由到该主题。

逻辑表

逻辑表是多个物理表路由到一个主题的常见用例。在逻辑表中，有多个物理表都有相同的模式。例如，分片表具有相同的模式。逻辑表可能由两个或多个分片表组成： db_shard1.my_table 和 db_shard2.my_table。表在不同的分片中，物理上是不同的，但它们组成一个逻辑表。您可以将任何分片中表的事件记录重新路由到同一主题。

分区的 PostgreSQL 表

当 Debezium PostgreSQL 连接器捕获分区表中的更改时，默认行为是更改事件记录会路由到每个分区的不同主题。要将记录从所有分区发送到一个主题，请配置主题路由 SMT。因为分区表中的每个键都保证是唯一的，所以请配置 key.enforce.uniqueness=false，以便 SMT 不添加 key 字段以确保唯一的密钥。额外的 key 字段是默认行为。

13.2.2. 多个表的路由 Debezium 记录示例

要将多个物理表的事件记录更改为同一主题，请在 Debezium 连接器的 Kafka Connect 配置中配置主题路由转换。配置主题路由 SMT 要求您指定确定的正则表达式：

- 要路由记录的表。这些表都必须具有相同的模式。
- 目标主题名称。

以下示例中的连接器配置为主题路由 SMT 设置几个选项：

```
transforms=Reroute
transforms.Reroute.type=io.debezium.transforms.ByLogicalTableRouter
transforms.Reroute.topic.regex=(.*)customers_shard(.*)
transforms.Reroute.topic.replacement=$1customers_all_shards
```

topic.regex

指定转换适用于每个更改事件记录的正则表达式，以确定它是否应该路由到特定主题。

在示例中，正则表达式 (.*)customers_shard(.*) 与名称包含 customers_shard 字符串的表的记录匹配。这将使用以下名称为表重新路由记录：

myserver.mydb.customers_shard1
myserver.mydb.customers_shard2
myserver.mydb.customers_shard3

topic.replacement

指定代表目标主题名称的正则表达式。转换会将每个匹配记录路由到此表达式标识的主题。在本例中，上面列出的三个分片表的记录将路由到 myserver.mydb.customers_all_shards 主题。

schema.name.adjustment.mode

指定来自结果主题名称的消息键模式名称应该如何进行调整，以便与连接器使用的消息转换器兼容。该值可以是 none （默认）或 avro。

自定义配置

要自定义配置，您可以定义一个 SMT predicate 语句，用于指定希望转换过程的表，或不处理。如果您将 SMT 配置为与正则表达式匹配的路由表，且您不希望 SMT 重新路由与表达式匹配的特定表，则 predicate 可能会很有用。

13.2.3. 确保路由到同一主题的 Debezium 记录间的唯一键

Debezium 更改事件键使用组成表的主键的表列。要将多个物理表的记录路由到一个主题，事件键必须在所有这些表中唯一。但是，每个物理表都可以有一个主键，该密钥只在该表中唯一。例如，myserver.mydb.customers_shard1 表中的行可能具有与 myserver.mydb.customers_shard2 表中的行相同的键值。

为确保每个事件键在更改事件记录的表之间是唯一的，主题路由转换会将字段插入到更改事件键中。默认情况下，插入字段的名称为 __dbz__physicalTableIdentifier。insert 字段的值是默认的目的地主题名称。

如果要，您可以配置主题路由转换，将不同的字段插入到键中。要做到这一点，指定 key.field.name 选项，并将其设置为没有现有主键字段名称的字段名称。例如：

```
transforms=Reroute
transforms.Reroute.type=io.debezium.transforms.ByLogicalTableRouter
transforms.Reroute.topic.regex=(.*)customers_shard(.*)
transforms.Reroute.topic.replacement=$1customers_all_shards
transforms.Reroute.key.field.name=shard_id
```

本例将 shard_id 字段添加到路由记录中的键结构中。

如果要调整密钥的新字段的值，请配置这两个选项：

key.field.regex

指定转换应用到默认目的地主题名称的正则表达式，以捕获一个或多个字符组。

key.field.replacement

指定用来决定在捕获的组中插入的 key 字段的值的正则表达式。

例如：

```
transforms.Reroute.key.field.regex=(.*)customers_shard(.*)
transforms.Reroute.key.field.replacement=$2
```

使用这个配置，假设默认目标主题名称是：

```
myserver.mydb.customers_shard1
myserver.mydb.customers_shard2
myserver.mydb.customers_shard3
```

转换使用第二个捕获的组中的值，即分片编号，作为 key 的新字段的值。在本例中，插入的 key 字段的值为 1、2 或 3。

如果您的表包含全局唯一的密钥，而您不需要更改密钥结构，您可以将 key.enforce.uniqueness 选项设置为 false ：

```
...
transforms.Reroute.key.enforce.uniqueness=false
...
```

13.2.4. 有选择地应用主题路由转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。

您可以使用以下任一方法配置连接器来有选择地应用 SMT：

- 为转换配置 SMT predicate。
- 对 SMT 使用 topic.regex 配置选项。

13.2.5. 配置 Debezium 主题路由转换的选项

下表描述了主题路由 SMT 配置选项。

表 13.1. 主题路由 SMT 配置选项

选项	默认	描述
topic.regex		指定转换适用于每个更改事件记录的正则表达式，以确定它是否应该路由到特定主题。
topic.replacement		指定代表目标主题名称的正则表达式。转换会将每个匹配记录路由到此表达式标识的主题。此表达式可以引用您为 **topic.regex** 指定的正则表达式捕获的组。要引用组，请指定 **$1**、**$2** 等等。

选项	默认	描述
key.enforce.uniqueness	**true**	指明是否在记录的更改事件键中添加字段。添加 key 字段可确保每个事件键在更改事件记录写入同一主题的表之间是唯一的。这有助于防止更改事件对具有相同密钥但源自不同源表的记录发生。 如果您不希望转换添加 key 字段，请指定 **false**。例如，如果您要将记录从分区 PostgreSQL 表路由到一个主题，您可以配置 **key.enforce.uniqueness=false**，因为分区的 PostgreSQL 表中保证了唯一的密钥。
key.field.name	**__dbz__physicalTableIdentifier**	要添加到更改事件键的字段名称。此字段的值标识原始表名称。对于 SMT 添加此字段，**key.enforce.uniqueness** 必须为 **true**，这是默认值。
key.field.regex		指定转换应用到默认目的地主题名称的正则表达式，以捕获一个或多个字符组。要使 SMT 应用到这个表达式，**key.enforce.uniqueness** 必须为 **true**，这是默认值。
key.field.replacement		指定用于在为 **key.field.regex** 指定的表达式捕获的组方面确定插入的 key 字段的值的正则表达式。要使 SMT 应用到这个表达式，**key.enforce.uniqueness** 必须为 **true**，这是默认值。
schema.name.adjustment.mode	none	指定来自结果主题名称的消息键 schema 名称应该如何进行调整以保持与连接器使用的消息转换器匹配，这包括：**none** 不应用任何调整（默认），**aro** 将 Avro 类型名称中使用的字符替换为下划线。
logical.table.cache.size	**16**	在 LRUCache 中用于保存最大条目的大小。缓存将为逻辑表键和值保留旧的/新模式，也缓存派生的键和值，以改进源记录转换。

13.3. 根据事件内容将事件记录路由到主题

默认情况下，Debezium 将所有从表读取的事件流传输到单个静态主题。但是，在某些情况下，您可能希望根据事件内容将所选事件重新路由到其他主题。基于内容的路由消息的过程位于 基于内容的路由消息传递模式中。要在 Debezium 中应用此模式，您可以使用基于内容的路由 单一消息转换 (SMT)来编写为每个事件评估的表达式。根据如何评估事件，SMT 会将事件消息路由到原始目标主题，或将其重新路由到您在表达式中指定的主题。

虽然可以使用 Java 创建自定义 SMT 来对路由逻辑进行编码，但使用自定义代码的 SMT 具有它的缺陷。例如：

- 需要预先编译转换并将其部署到 Kafka Connect。
- 每次更改都需要代码重新编译和重新部署，从而导致不灵活的操作。

基于内容的路由 SMT 支持脚本语言与 JSR 223 集成（用于 Java™ 平台）。

Debezium 不附带 JSR 223 API 的任何实施。要将表达式语言与 Debezium 搭配使用，您必须下载用于语言的 JSR 223 脚本引擎实施。根据您用于部署 Debezium 的方法，您可以从 Maven Central 自动下载所需的工件，也可以手动下载工件，然后将它们添加到 Debezium 连接器插件目录中，以及语言实施使用的任何其他 JAR 文件。

13.3.1. 设置 Debezium content-based-routing SMT

为安全起见，Debezium 连接器存档不包括基于内容的路由 SMT。相反，它会在单独的工件 debezium-scripting-2.3.4.Final.tar.gz 中提供。

如果通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium 连接器，以使用过滤器 SMT，您必须将 SMT 工件显式添加到 Kafka Connect 环境中。当使用 AMQ Streams 部署连接器时，它可以根据您在 Kafka Connect 自定义资源中指定的配置参数自动下载所需的工件。重要信息：在 Kafka Connect 实例中存在路由 SMT 后，允许向实例添加连接器的任何用户都可以运行脚本表达式。为确保脚本表达式只能由授权用户运行，请务必在添加路由 SMT 前保护 Kafka Connect 实例及其配置接口。

如果您从 Dockerfile 构建 Kafka Connect 容器镜像，则应用以下步骤。如果使用 AMQ Streams 创建 Kafka Connect 镜像，请按照您的连接器部署主题中的说明进行操作。

流程

1. 在浏览器中，打开 Red Hat Integration 下载站点，并下载 Debezium 脚本 SMT 归档 (debezium-scripting-2.3.4.Final.tar.gz)。
2. 将存档的内容提取到 Kafka Connect 环境的 Debezium 插件目录中。

3. 获取 JSR-223 脚本引擎实施，并将其内容添加到 Kafka Connect 环境的 Debezium 插件目录中。

4. 重启 Kafka Connect 进程以获取新的 JAR 文件。

Groovy 语言在 classpath 中需要以下库：

- groovy
- groovy-json （可选）
- groovy-jsr223

JavaScript 语言在 classpath 中需要以下库：

- graalvm.js
- graalvm.js.scriptengine

13.3.2. 示例： Debezium 基本基于内容的路由配置

要将 Debezium 连接器配置为根据事件内容路由更改事件记录，您可以在连接器的 Kafka Connect 配置中配置 ContentBasedRouter SMT。

配置基于内容的路由 SMT 要求您指定一个定义过滤条件的正则表达式。在配置中，您可以创建一个定义路由条件的正则表达式。表达式定义评估事件记录的模式。它还指定路由模式匹配的目标主题的名称。您指定的模式可能会指定事件类型，如表插入、更新或删除操作。您还可以定义与特定列或行中值匹配的模式。

例如，要将所有更新(u)记录重新路由到 updates 主题，您可以在连接器配置中添加以下配置：

```
...
```

```
transforms=route
transforms.route.type=io.debezium.transforms.ContentBasedRouter
transforms.route.language=jsr223.groovy
transforms.route.topic.expression=value.op == 'u' ? 'updates' : null
...
```

前面的示例指定了 Groovy 表达式语言的使用。

与模式匹配的记录将路由到默认主题。

自定义配置

前面的例子显示一个简单的 SMT 配置，它仅用于处理 DML 事件，其中包含一个 op 字段。连接器可能会发出的其他类型的信息(heartbeat 消息、tombstone 消息或有关事务或模式更改的元数据信息)不包含此字段。为了避免处理失败，您可以定义一个 SMT predicate 语句，该语句仅有选择地将转换 应用到特定的事件。

13.3.3. 基于 Debezium 内容的路由表达式中使用的变量

Debezium 将某些变量绑定到 SMT 的评估上下文中。当您创建表达式来指定控制路由目的地的条件时，SMT 可以查找并解释这些变量的值来评估表达式中的条件。

下表列出了 Debezium 绑定到基于内容的路由 SMT 的评估上下文的变量：

表 13.2. 基于内容的路由表达式变量

名称	描述	类型
key	消息的一个关键信息。	**org.apache.kafka.connect.data .Struct**
value	消息值。	**org.apache.kafka.connect.data .Struct**
keySchema	message 键的 schema。	**org.apache.kafka.connect.data .Schema**
valueSchema	message 值的 schema。	**org.apache.kafka.connect.data .Schema**
topic	目标主题的名称。	字符串

名称	描述	类型
标头	消息标头的 Java 映射。key 字段是标头名称。**headers** 变量公开以下属性： • 值 （类型为 **Object**） • **模式** （类型为 **org.apache.kafka.connect.data.Schema**）	**java.util.Map<String, io.debezium.transforms.scripting.RecordHeader>**

表达式可以在其变量上调用任意方法。表达式应该解析为布尔值，它决定了 SMT 分布消息的方式。当表达式中的路由条件评估为 true 时，消息会被保留。当路由条件评估为 false 时，消息将被删除。

表达式不应产生任何副作用。也就是说，它们不应修改它们通过的任何变量。

13.3.4. 有选择地应用基于内容的路由转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。您可以使用以下任一方法配置连接器来有选择地应用 SMT：

- *为转换配置 SMT predicate。*
- *对 SMT 使用 topic.regex 配置选项。*

13.3.5. 为其他脚本语言配置基于内容的路由条件

您表达基于内容的路由条件的方式取决于您所使用的脚本语言。例如，如 基本配置示例 所示，当您使用 Groovy 作为表达式语言时，以下表达式重新路由所有更新(u)记录，同时将其他记录路由到默认主题：

```
value.op == 'u' ? 'updates' : null
```

其他语言使用不同的方法来表达相同的条件。

提示

Debezium MongoDB 连接器会将 after 和 patch 字段作为序列化 JSON 文档而不是结构发送。要将 ContentBasedRouting SMT 与 MongoDB 连接器搭配使用，您必须首先将 JSON 中的数组字段卸载到单独的文档中。
您可以在表达式中使用 JSON 解析器为每个数组项生成单独的输出文档。例如，如果您使用 Groovy 作为表达式语言，请将 groovy-json 构件添加到类路径，然后添加一个如 (new groovy.json.JsonSlurper ()）.parseText (value.after).last_name == 'Kretchmar'。

Javascript

当使用 JavaScript 作为表达式语言时，您可以调用 Struct#get （） 方法来指定基于内容的路由条件，如下例所示：

```
value.get('op') == 'u' ? 'updates' : null
```

JavaScript with Graal.js

当您使用带有 Graal.js 的 JavaScript 创建基于内容的路由条件时，您可以使用类似于 Groovy 的方法。例如：

```
value.op == 'u' ? 'updates' : null
```

13.3.6. 用于配置基于内容的路由转换的选项

属性	默认	描述
topic.regex		为事件评估目标主题名称的可选正则表达式，以确定是否应用条件逻辑。如果目标主题的名称与 **topic.regex** 的值匹配，则转换会在将事件传递给主题前应用条件逻辑。如果主题的名称与 **topic.regex** 的值不匹配，则 SMT 会将事件传递给未修改的主题。
language		编写表达式的语言。必须以 **jsr223.** 开始，如 **jsr223.groovy** 或 **jsr223.graal.js**。Debezium 仅支持通过 JSR 223 API 进行引导 ("用于 Java ™ 平台")
topic.expression		针对每个消息评估的表达式。必须评估一个 **String** 值，其中非null 重新路由消息到新主题，而 **null** 值会将消息路由到默认主题。

null.handling.mode	keep	指定转换如何处理 **null** (tombstone)信息。您可以指定以下选项之一： **keep** （默认）传递消息。 **drop** 完全删除消息。 **评估** 将条件逻辑应用到消息。

13.4. 从 DEBEZIUM 事件记录中提取字段级别的更改

Debezium 数据更改事件具有复杂的结构，可提供大量信息。然而，在某些情况下，在下游消费者处理 Debezium 更改事件信息前，它需要有关原始数据库更改结果的信息。为了增强事件消息，其中包含数据库操作如何修改源数据库中的字段的详细信息，Debebe 提供 ExtractChangedRecordState 单个消息转换(SMT)。

事件更改转换是一个 Kafka Connect SMT。

13.4.1. Debezium 更改事件结构的描述

Debezium 生成具有复杂结构的数据更改事件。每个事件由以下部分组成：

- *元数据，其中包括但不仅限于以下类型：*
  - *更改数据的操作类型。*
  - *源信息，如数据库的名称以及所发生更改的表。*
  - *在进行更改时标识的时间戳。*
  - *可选的事务信息。*
- *更改前的行数据。*

- 更改后行数据。

以下示例显示了典型的 Debezium UPDATE 更改事件结构的一部分：

```
{
 "op": "u",
 "source": {
  ...
 },
 "ts_ms" : "...",
 "before" : {
  "field1" : "oldvalue1",
  "field2" : "oldvalue2"
 },
 "after" : {
  "field1" : "newvalue1",
  "field2" : "newvalue2"
 }
}
```

上例中的消息的复杂格式提供有关源数据库中发生更改的详细信息。但是，格式可能不适用于某些下游用户。sink 连接器或 Kafka 生态系统的其他部分可能会预期信息明确标识数据库操作更改或保持不变的字段。ExtractChangedRecordState SMT 将标头添加到更改事件消息中，以识别由数据库操作修改的字段，以及保持不变的字段。

13.4.2. Debezium 事件的行为更改 SMT

事件更改 SMT 在 Kafka 记录中从 Debezium UPDATE 更改事件中提取 before 和 after 字段。转换会检查事件状态结构的 before 和 after，以识别操作更改的字段，以及保持不变的字段。根据连接器配置，转换会生成修改的事件消息，添加消息标头来列出更改的字段、更改的字段或两者。如果事件代表 INSERT 或 DELETE，则这个单一消息转换无效。

您可以为 Debezium 连接器配置事件更改 SMT，或使用 Debezium 连接器发送的消息的接收器连接器。如果您希望 Apache Kafka 保留整个原始 Debezium 更改事件，请为接收器连接器配置 SMT。将 SMT 应用到源或接收器连接器的决定取决于您的特定用例。

根据您的用例，您可以通过执行以下任一任务来配置转换来修改原始消息：

- 通过在用户配置的 header.changed.name 标头中列出它们，来识别由 UPDATE 事件更改的字段。
-

通过在用户配置的 header.unchanged.name 标头中列出它们，识别 UPDATE 事件未更改的字段。

13.4.3. 配置 Debezium 事件更改 SMT

您可以通过在连接器的配置中添加 SMT 配置详情，为 Kafka Connect 源或 sink 连接器配置 Debezium 事件更改 SMT。要获取默认行为，不添加任何标头，请将转换添加到连接器配置中，如下例所示：

```
transforms=changes,...
transforms.changes.type=io.debezium.transforms.ExtractChangedRecordState
```

与任何 Kafka Connect 连接器配置一样，您可以将 transformations= 设置为多个用逗号分开的 SMT 别名，按您希望 Kafka Connect 应用 SMT 的顺序设置。

以下示例中的连接器配置为事件更改 SMT 设置几个选项：

```
transforms=changes,...
transforms.changes.type=io.debezium.transforms.ExtractChangedRecordState
transforms.changes.header.changed.name=Changed
transforms.changes.header.unchanged.name=Unchanged
```

header.changed.name

用于存储数据库操作更改的字段的逗号分隔列表的 Kafka 消息标头名称。

header.unchanged.name

Kafka 消息标头名称，用于存储在数据库操作后保持不变的字段列表。

自定义配置

连接器可能会发出许多类型的事件消息(heartbeat 消息、tombstone 消息或有关事务或模式更改的元数据信息)。要将转换应用到事件子集，您可以定义一个 SMT predicate 语句，该语句有选择地将转换 应用到特定的事件。

13.4.4. 用于有选择地应用事件更改转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。

有关如何有选择地应用 SMT 的更多信息，请参阅为转换配置 SMT predicate。

13.4.5. Debezium 事件的配置选项的描述会更改 SMT

下表描述了您可以指定配置事件更改 SMT 的选项。

表 13.3. 事件更改 SMT 配置选项的描述

选项	默认	描述
header.changed.name		用于存储数据库操作更改的字段的逗号分隔列表的 Kafka 消息标头名称。
header.unchanged.name		Kafka 消息标头名称，用于存储在数据库操作后保持不变的字段列表。

13.5. 过滤 DEBEZIUM 更改事件记录

默认情况下，Debezium 会发送它收到的每个数据更改事件。但是，在很多情况下，您可能只对制作者发出的事件的子集感兴趣。要只处理与您相关的记录，Debezium 提供了 过滤 单一消息转换 (SMT)。

虽然可以使用 Java 创建自定义 SMT 来对逻辑进行编码，但使用自定义代码的 SMT 具有它的缺陷。例如：

- 需要预先编译转换并将其部署到 Kafka Connect。
- 每次更改都需要代码重新编译和重新部署，从而导致不灵活的操作。

过滤器 SMT 支持脚本语言与 JSR 223 集成（用于 Java™ 平台）。

Debezium 不附带 JSR 223 API 的任何实施。要将表达式语言与 Debezium 搭配使用，您必须下载用于语言的 JSR 223 脚本引擎实施。根据您用于部署 Debezium 的方法，您可以从 Maven Central 自动下载所需的工件，也可以手动下载工件，然后将它们添加到 Debezium 连接器插件目录中，以及语言实施使用的任何其他 JAR 文件。

13.5.1. 设置 Debezium 过滤器 SMT

为安全起见，Debezium 连接器存档中没有包括过滤 SMT。相反，它会在单独的工件 debezium-scripting-2.3.4.Final.tar.gz 中提供。

如果您通过从 Dockerfile 构建自定义 Kafka Connect 容器镜像来部署 Debezium 连接器，以使用过滤器 SMT，您必须明确下载 SMT 归档，并将文件与连接器插件一起部署。当使用 AMQ Streams 部署连接器时，它可以根据您在 Kafka Connect 自定义资源中指定的配置参数自动下载所需的工件。重要信息：在 Kafka Connect 实例中存在过滤器 SMT 后，允许向实例添加连接器的任何用户都可以运行脚本表达式。为确保脚本表达式只能由授权用户运行，请务必在添加过滤器 SMT 前保护 Kafka Connect 实例及其配置接口。

如果您从 Dockerfile 构建 Kafka Connect 容器镜像，则应用以下步骤。如果使用 AMQ Streams 创建 Kafka Connect 镜像，请按照您的连接器部署主题中的说明进行操作。

流程

1. *在浏览器中，打开 Red Hat Integration 下载站点，并下载 Debezium 脚本 SMT 归档 (debezium-scripting-2.3.4.Final.tar.gz)。*
2. *将存档的内容提取到 Kafka Connect 环境的 Debezium 插件目录中。*
3. *获取 JSR-223 脚本引擎实施，并将其内容添加到 Kafka Connect 环境的 Debezium 插件目录中。*
4. *重启 Kafka Connect 进程以获取新的 JAR 文件。*

Groovy 语言在 classpath 中需要以下库：

- *groovy*
- *groovy-json （可选）*
- *groovy-jsr223*

JavaScript 语言在 classpath 中需要以下库：

- graalvm.js
- graalvm.js.scriptengine

13.5.2. 示例： Debezium 基本过滤器 SMT 配置

您可以在 Debezium 连接器的 Kafka Connect 配置中配置过滤器转换。在配置中，您可以通过定义基于业务规则的过滤器条件来指定您感兴趣的事件。当过滤 SMT 处理事件流时，它会根据配置的过滤器条件评估每个事件。只有满足过滤器条件条件的事件才会传递给代理。

要配置 Debezium 连接器来过滤更改事件记录，请在 Debezium 连接器的 Kafka Connect 配置中配置 Filter SMT。配置过滤器 SMT 要求您指定一个定义过滤条件的正则表达式。

例如，您可以在连接器配置中添加以下配置。

```
...
transforms=filter
transforms.filter.type=io.debezium.transforms.Filter
transforms.filter.language=jsr223.groovy
transforms.filter.condition=value.op == 'u' && value.before.id == 2
...
```

前面的示例指定了 Groovy 表达式语言的使用。正则表达式 value.op == 'u' && value.before.id == 2 会删除所有消息，但代表更新(u)的 id 值除外，其 id 值等于 2。

自定义配置

前面的例子显示一个简单的 SMT 配置，它仅用于处理 DML 事件，其中包含一个 op 字段。连接器可能会发出的其他类型的信息(heartbeat 消息、tombstone 消息或有关模式更改和事务的元数据信息)不包含此字段。为了避免处理失败，您可以定义一个 SMT predicate 语句，该语句仅有选择地将转换 应用到特定的事件。

13.5.3. 过滤器表达式中使用的变量

Debezium 将某些变量绑定到过滤器 SMT 的评估上下文中。在创建表达式来指定过滤器条件时，您可以使用 Debezium 绑定到评估上下文的变量。通过绑定变量，Debebe 可让 SMT 查找和解释它们的值，

因为它评估表达式中的条件。

下表列出了 Debezium 绑定到过滤器 SMT 的评估上下文的变量：

表 13.4. 过滤表达式变量

名称	描述	类型
key	消息的一个关键信息。	**org.apache.kafka.connect.data .Struct**
value	消息值。	**org.apache.kafka.connect.data .Struct**
keySchema	message 键的 schema。	**org.apache.kafka.connect.data .Schema**
valueSchema	message 值的 schema。	**org.apache.kafka.connect.data .Schema**
topic	目标主题的名称。	字符串
标头	消息标头的 Java 映射。key 字段是标头名称。**headers** 变量公开以下属性： • 值 （类型为 **Object**） • **模式** （类型为 **org.apache.kafka .connect.data .Schema**）	**java.util.Map<String, io.debezium.transforms.scripting .RecordHeader>**

表达式可以在其变量上调用任意方法。表达式应该解析为布尔值，它决定了 SMT 分布消息的方式。当表达式中的过滤器条件评估为 true 时，消息会被保留。当过滤器条件评估为 false 时，消息将被删除。

表达式不应产生任何副作用。也就是说，它们不应修改它们通过的任何变量。

13.5.4. 有选择地应用过滤器转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。您可以使用以下任一方法配置连接器来有选择地应用 SMT：

- 为转换配置 SMT predicate。
- 对 SMT 使用 topic.regex 配置选项。

13.5.5. 过滤其他脚本语言的条件配置

表达过滤条件的方式取决于您使用的脚本语言。

例如，如 基本配置示例 所示，当您使用 Groovy 作为表达式语言时，以下表达式会删除所有消息，但更新记录除外，其 id 值设置为 2：

```
value.op == 'u' && value.before.id == 2
```

其他语言使用不同的方法来表达相同的条件。

提示

Debezium MongoDB 连接器会将 after 和 patch 字段作为序列化 JSON 文档而不是结构发送。要将过滤器 SMT 与 MongoDB 连接器搭配使用，您必须首先将 JSON 中的数组字段解压缩到单独的文档中。
您可以在表达式中使用 JSON 解析器为每个数组项生成单独的输出文档。例如，如果您使用 Groovy 作为表达式语言，请将 groovy-json 构件添加到类路径，然后添加一个如 (new groovy.json.JsonSlurper（）).parseText (value.after).last_name == 'Kretchmar'。

Javascript

如果使用 JavaScript 作为表达式语言，您可以调用 Struct#get （） 方法来指定过滤条件，如下例所示：

```
value.get('op') == 'u' && value.get('before').get('id') == 2
```

JavaScript with Graal.js

如果您使用带有 Graal.js 的 JavaScript 定义过滤条件，您可以使用一个与您与 Groovy 一起使用的方法类似。例如：

```
value.op == 'u' && value.before.id == 2
```

13.5.6. 配置过滤器转换的选项

下表列出了可与过滤器 SMT 一起使用的配置选项。

表 13.5. 过滤 SMT 配置选项

属性	默认	描述
topic.regex		为事件评估目标主题名称的可选正则表达式，以确定是否应用过滤逻辑。如果目标主题的名称与 **topic.regex** 的值匹配，则转换会在将事件传递给主题前应用过滤器逻辑。如果主题的名称与 **topic.regex** 的值不匹配，则 SMT 会将事件传递给未修改的主题。
language		编写表达式的语言。必须以 **jsr223.** 开始，如 **jsr223.groovy** 或 **jsr223.graal.js**。Debezium 仅支持通过 JSR 223 API 进行引导 ("用于 Java ™ 平台")
condition		针对每个消息评估的表达式。必须评估一个布尔值，其中 **true** 的结果会保留该消息，并且删除 **false** 的结果。
null.handling.mode	**keep**	指定转换如何处理 **null** (tombstone)信息。您可以指定以下选项之一： **keep** （默认）传递消息。 **drop** 完全删除消息。 **评估** 将过滤器条件应用到消息。

13.6. 将消息标头转换为事件记录值

HeaderToValue SMT 从事件记录中提取指定的标头字段，然后将标头字段复制到事件记录中的值。移动 选项会完全从标头中删除字段，然后再将它们作为有效负载中的值添加。您可以配置 SMT 以操作原始消息中的多个标头。您可以使用点表示法在有效负载中指定您要嵌套标头字段的节点。有关配置 SMT 的更多信息，请参见以下示例。

13.6.1. 示例： Debezium HeaderToValue SMT 的基本配置

要在事件记录中提取消息标头到记录值中，请在连接器的 Kafka Connect 配置中配置 HeaderToValue SMT。您可以配置转换以删除原始标头或复制它们。要从记录中删除标头字段，请将

SMT 配置为使用 move 操作。要在原始记录中保留标头字段，请将 SMT 配置为使用 复制操作。例如，要从事件信息中删除标头 event_timestamp 和 key，请在连接器配置中添加以下行：

```
transforms=moveHeadersToValue
transforms.moveHeadersToValue.type=io.debezium.transforms.HeaderToValue
transforms.moveHeadersToValue.headers=event_timestamp,key
transforms.moveHeadersToValue.fields=timestamp,source.id
transforms.moveHeadersToValue.operation=move
```

以下示例显示了应用转换前和之后事件记录的标头和值。

例 13.1. 应用 HeaderToValue SMT 的影响

由 HeaderToValue 转换处理前的事件记录

SMT 处理事件记录的标头

```
{
    "header_x": 0,
    "event_timestamp": 1626102708861,
    "key": 100,
}
```

SMT 处理事件记录前的值

```
{
        "before": null,
        "after": {
            "id": 1,
            "first_name": "Anne",
            "last_name": "Kretchmar",
            "email": "annek@noanswer.org"
        },
        "source": {
            "version": "2.1.3.Final",
            "connector": "postgresql",
            "name": "PostgreSQL_server",
            "ts_ms": 1559033904863,
            "snapshot": true,
            "db": "postgres",
            "sequence": "[\"24023119\",\"24023128\"]"
            "schema": "public",
            "table": "customers",
            "txId": 555,
            "lsn": 24023128,
            "xmin": null
        },
        "op": "c",
        "ts_ms": 1559033904863
    }
```

由 HeaderToValue 转换处理的事件记录

SMT 删除指定字段后的标头

```
{
   "header_x": 0
}
```

SMT 移动标头字段到值后的值

```
{
      "before": null,
      "after": {
         "id": 1,
         "first_name": "Anne",
         "last_name": "Kretchmar",
         "email": "annek@noanswer.org"
      },
      "source": {
         "version": "2.1.3.Final",
         "connector": "postgresql",
         "name": "PostgreSQL_server",
         "ts_ms": 1559033904863,
         "snapshot": true,
         "db": "postgres",
         "sequence": "[\"24023119\",\"24023128\"]"
         "schema": "public",
         "table": "customers",
         "txId": 555,
         "lsn": 24023128,
         "xmin": null,
         "id": 100
      },
      "op": "c",
      "ts_ms": 1559033904863,
      "event_timestamp": 1626102708861
   }
```

13.6.2. 用于配置 HeaderToValue 转换的选项

下表列出了可用于 HeaderToValue SMT 的配置选项。

表 13.6. HeaderToValue SMT 配置选项

属性	描述	类型	Default（默认）	有效值	重要性
标头	记录中以逗号分隔的标头名称列表，其值要复制或移到记录值。	list	没有默认值	非空列表	high

字段	以逗号分隔的字段名称列表，其顺序与 **headers** 配置属性中列出的标头名称相同。使用点表示法来指示消息有效负载特定节点中的 SMT 嵌套字段。有关如何将 SMT 配置为使用点表示法的详情，请参考本节前面出现 的示例。	list	没有默认值	非空列表	high
operation	指定以下选项之一： **将**:: SMT 将标头字段移到事件记录中的值，并从标头中删除字段。**将**:: SMT 将标头字段复制到事件记录中的值，并保留原始标头字段。	string	没有默认值	move 或 copy	high

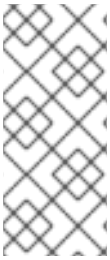
13.7. 从 DEBEZIUM 更改事件中提取源记录 AFTER 状态

Debezium 连接器发出数据更改消息来代表它们从源数据库捕获的每个操作。连接器发送到 Apache Kafka 的消息有一个复杂的结构，它代表原始数据库事件的详情。

虽然这种复杂的消息格式准确了解有关系统中发生更改的详细信息，但格式可能不适用于某些下游用户。sink 连接器或 Kafka 生态系统的其他部分可能需要格式化信息，以便以简化的扁平化结构呈现字段名称和值。

要简化 Debezium 连接器生成的事件记录格式，您可以使用 Debezium 事件扁平化单一消息转换 (SMT)。配置转换以支持需要 Kafka 记录的格式比连接器生成的默认格式更简单。根据您的特定用例，您可以将 SMT 应用到 Debezium 连接器，或应用到消耗 Debezium 连接器生成的消息的接收器连接器。要启用 Apache Kafka 以原始格式保留 Debezium 更改事件信息，请为接收器连接器配置 SMT。

事件扁平化转换是一个 Kafka Connect SMT。



注意

本章中的信息描述了基于 SQL 的数据库连接器的事件扁平化单一消息转换(SMT)。有关 Debezium MongoDB 连接器等效的 SMT 的详情，请参考 MongoDB New Document State Extraction。

以下主题提供详情：

- ***第 13.7.1 节 “Debezium 更改事件结构的描述”***

- 第 13.7.2 节 “Debezium 事件扁平化转换的行为”
- 第 13.7.3 节 “配置 Debezium 事件扁平化转换”
- 第 13.7.4 节 “在 Kafka 记录中添加 Debezium 元数据示例”
- 第 13.7.6 节 “用于配置 Debezium 事件扁平化转换的选项”

13.7.1. Debezium 更改事件结构的描述

Debezium 生成具有复杂结构的数据更改事件。每个事件由三个部分组成：

- 元数据，其中包括但不仅限于：
  - 更改数据的操作类型。
  - 源信息，如数据库的名称以及所发生更改的表。
  - 在进行更改时标识的时间戳。
  - 可选的事务信息。
- 更改前的行数据
- 更改后的行数据

以下示例显示了 UPDATE 更改事件的消息结构的一部分：

```
{
 "op": "u",
```

```
"source": {
 ...
},
"ts_ms" : "...",
"before" : {
 "field1" : "oldvalue1",
 "field2" : "oldvalue2"
},
"after" : {
 "field1" : "newvalue1",
 "field2" : "newvalue2"
}
}
```

有关连接器更改事件结构的更多信息，请参阅连接器的文档。

当事件扁平化 SMT 处理上例中的消息后，它简化了消息格式，如下例所示：

```
{
 "field1" : "newvalue1",
 "field2" : "newvalue2"
}
```

13.7.2. Debezium 事件扁平化转换的行为

事件扁平化 SMT 从 Kafka 记录中的 Debezium 更改事件中提取 after 字段。SMT 仅将原始更改事件替换为其 after 字段，以创建简单的 Kafka 记录。

您可以为 Debezium 连接器或消耗 Debezium 连接器发送消息的接收器连接器配置事件扁平化 SMT。为接收器连接器配置事件扁平化的优点在于 Apache Kafka 中存储的记录包含整个 Debezium 更改事件。将 SMT 应用到源或接收器连接器的决定取决于您的特定用例。

您可以将转换配置为执行以下操作之一：

- 将更改事件中的元数据添加到简化的 Kafka 记录中。默认行为是 SMT 不添加元数据。
- 在流中保留包含 DELETE 操作更改事件的 Kafka 记录。默认行为是 SMT 丢弃 DELETE 操作更改事件的 Kafka 记录，因为大多数消费者还没有处理它们。

数据库 DELETE 操作会导致 Debezium 生成两个 Kafka 记录：

- 包含 "op": "d"、 before 行数据以及一些其他字段的记录。
- 有一个 tombstone 记录，其键与已删除行和值 null 相同。此记录是 Apache Kafka 的标记。表示 日志压缩 可以删除具有此密钥的所有记录。

您可以将事件扁平化 SMT 配置为执行以下操作之一，而不是丢弃包含 before 行数据的记录：

- 在流中保留记录，并编辑它使其只有 "value": "null" 字段。
- 在流中保留记录，并编辑它，使其包含一个 value 字段，其中包含带有添加 "__deleted": "true" 条目的 before 字段中的键/值对。

同样，您可以配置事件扁平化 SMT 以在流中保留 tombstone 记录，而不是丢弃 tombstone 记录。

13.7.3. 配置 Debezium 事件扁平化转换

通过在连接器的配置中添加 SMT 配置详情，在 Kafka Connect 源或 sink 连接器中配置 Debezium 事件扁平化 SMT。例如，要获取转换的默认行为，请在没有指定任何选项的情况下将其添加到连接器配置中，如下例所示：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.transforms.ExtractNewRecordState
```

与任何 Kafka Connect 连接器配置一样，您可以将 transformations= 设置为多个用逗号分开的 SMT 别名，按您希望 Kafka Connect 应用 SMT 的顺序设置。

以下 .properties 示例设置几个事件扁平化 SMT 选项：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.transforms.ExtractNewRecordState
transforms.unwrap.drop.tombstones=false
transforms.unwrap.delete.handling.mode=rewrite
transforms.unwrap.add.fields=table,lsn
```

drop.tombstones=false

在事件流中为 DELETE 操作保留 tombstone 记录。

delete.handling.mode=rewrite

对于 DELETE 操作，请通过扁平化更改事件中的 value 字段来编辑 Kafka 记录。value 字段直接包含 before 字段中的键/值对。SMT 添加 __deleted 并将其设置为 true，例如：

```
"value": {
  "pk": 2,
  "cola": null,
  "__deleted": "true"
}
```

add.fields=table,lsn

在简化的 Kafka 记录中添加 table 和 lsn 字段的更改事件元数据。

自定义配置

连接器可能会发出许多类型的事件消息(heartbeat 消息、tombstone 消息或有关事务或模式更改的元数据信息)。要将转换应用到事件子集，您可以定义一个 SMT predicate 语句，该语句有选择地将转换应用到特定的事件。

13.7.4. 在 Kafka 记录中添加 Debezium 元数据示例

您可以配置事件扁平化 SMT，将原始更改事件元数据添加到简化的 Kafka 记录中。例如，您可能希望简化的记录的标头或值包含以下任一内容：

- 进行更改的操作类型
- 已更改的数据库或表的名称
- 特定于连接器的字段，如 Postgres LSN 字段

要在简化的 Kafka 记录标头中添加元数据，请指定 add.headers 选项。要在简化的 Kafka 记录值中添加元数据，请指定 add.fields 选项。每个选项都使用以逗号分隔的更改事件字段名称列表。不要指定空格。当存在重复的字段名称时，若要为其中一个字段添加元数据，请指定 struct 和字段。例如：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.transforms.ExtractNewRecordState
transforms.unwrap.add.fields=op,table,lsn,source.ts_ms
transforms.unwrap.add.headers=db
transforms.unwrap.delete.handling.mode=rewrite
```

使用这个配置，简化的 Kafka 记录将包含如下内容：

```
{
 ...
 "__op" : "c",
 "__table": "MY_TABLE",
 "__lsn": "123456789",
 "__source_ts_ms" : "123456789",
 ...
}
```

另外，简化的 Kafka 记录也会有一个 __db 标头。

在简化的 Kafka 记录中，SMT 使用双下划线前缀 metadata 字段名称。当您指定 struct 时，SMT 也在 struct 名称和字段名称之间插入下划线。

要在用于 DELETE 操作的简化 Kafka 记录中添加元数据，还必须配置 delete.handling.mode=rewrite。

13.7.5. 用于有选择地应用事件扁平化转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。

有关如何有选择地应用 SMT 的更多信息，请参阅为转换配置 SMT predicate。

13.7.6. 用于配置 Debezium 事件扁平化转换的选项

下表描述了您可以指定配置事件扁平化 SMT 的选项。

表 13.7. 事件扁平化 SMT 配置选项的描述

选项	默认	描述
drop.tombstones	**true**	Debezium 为每个 **DELETE** 操作生成一个 tombstone 记录。默认的行为是事件扁平化 SMT 从流中删除 tombstone 记录。要在流中保留 tombstone 记录，请指定 **drop.tombstones=false**。
delete.handling.mode	**drop**	Debezium 为每个 **DELETE** 操作生成更改事件记录。默认行为是事件扁平化 SMT 从流中删除这些记录。要在流中保留 **DELETE** 操作的 Kafka 记录，请将 **delete.handling.mode** 设置为 **none** 或 **rewrite**。 指定 **none** 在流中保留更改事件记录。记录仅包含 **"value": "null** "。 指定 **rewrite** 在流中保留更改事件记录，并编辑记录具有包含 **before** 字段中的键/值对的 **value** 字段，并将 **__deleted: true** 添加到 **值**中。这是指明该记录已被删除的另一种方式。 当您指定 **重写** 时，对 **DELETE** 操作的更新简化的记录都可能需要跟踪已删除的记录。您可以考虑接受丢弃 Debezium 连接器创建的 tombstone 记录的默认行为。
route.by.field		要使用行数据来确定要将记录路由到的主题，请将此选项设置为 **after** 字段属性。SMT 将记录路由到名称与指定 **after** 字段属性值匹配的主题。对于 **DELETE** 操作，将此选项设置为 **before** 字段属性。 例如，配置 **route.by.field=destination** 将记录路由到 name 是 **after.destination** 的值的主题。默认行为是，Debezium 连接器会将每个更改事件记录发送到一个主题，其名称是从数据库的名称以及进行更改的表的名称。 如果要在 sink 连接器上配置事件扁平化 SMT，当目标主题名称指定使用简化的更改事件记录更新的数据库表的名称时，设置这个选项可能很有用。如果您的用例主题名称不正确，您可以配置 **route.by.field** 来重新路由事件。
add.fields.prefix	__ (double-underscore)	将此可选字符串设为前缀字段。

选项	默认	描述
add.fields		将这个选项设置为以逗号分隔的列表，没有空格，使用 metadata 字段添加到简化的 Kafka 记录值中。当存在重复的字段名称时，若要为其中一个字段添加元数据，请指定 struct 以及字段，如 **source.ts_ms**。 另外，您可以通过 <field name&**gt;:<new 字段 name&** gt; 覆盖字段名称，例如：新字段名称，如 **version:VERSION, connector:CONNECTOR, source.ts_ms:EVENT_TIMESTAMP**。请注意，**新字段名称**是区分大小写的。 当 SMT 将元数据字段添加到简化的记录值中时，它会使用双下划线前缀每个 metadata 字段名称。对于 struct 规格，SMT 也在 struct 名称和字段名称之间插入下划线。 如果您指定了不在更改事件记录中的字段，则 SMT 仍然会将字段添加到记录的值中。
add.headers.prefix	__ (double-underscore)	将此可选字符串设为前缀标头。
add.headers		将这个选项设置为以逗号分隔的列表，没有空格，使用 metadata 字段添加到简化的 Kafka 记录标头中。当存在重复的字段名称时，若要为其中一个字段添加元数据，请指定 struct 以及字段，如 **source.ts_ms**。 另外，您可以通过 <field name&**gt;:<new 字段 name&** gt; 覆盖字段名称，例如：新字段名称，如 **version:VERSION, connector:CONNECTOR, source.ts_ms:EVENT_TIMESTAMP**。请注意，**新字段名称**是区分大小写的。 当 SMT 将元数据字段添加到简化的记录标头中时，它会使用双下划线前缀每个 metadata 字段名称。对于 struct 规格，SMT 也在 struct 名称和字段名称之间插入下划线。 如果您指定了不在更改事件记录中的字段，则 SMT 不会将字段添加到标头中。
drop.fields.header.name		用于列出您要从输出消息丢弃的源消息中的字段名称的 Kafka 消息标头名称。
drop.fields.from.key	**false**	指定是否希望 SMT 从事件键中删除 **drop.fields.header.name** 中列出的字段。

选项	默认	描述
drop.fields.keep.schema.compatible	true	指定是否希望 SMT 删除 drop.fields.header.name 配置属性中包含的非可选字段。 默认情况下，SMT 只删除标记为可选的字段。

13.8. 在 DEBEZIUM MONGODB 更改事件状态后提取源文档

Debezium MongoDB 连接器会发出数据更改信息来代表 MongoDB 集合中发生的每个操作。这些事件消息的复杂结构是原始数据库事件的详细信息。但是，一些下游用户可能无法以其原始格式处理消息。例如，为了代表数据收集中的嵌套文档，连接器以包含嵌套字段的格式发出事件消息。要支持接收器连接器，或者无法处理原始消息的分层格式的其他消费者，您可以使用 Debezium MongoDB 事件扁平化 (ExtractNewDocumentState)单一消息转换(SMT)。SMT 简化了原始消息的结构，并可以其他方式修改消息，以便更轻松地处理数据。

事件扁平化转换是一个 Kafka Connect SMT。

注意

本章中的信息描述了 Debezium MongoDB 连接器的事件扁平化单一消息转换(SMT)。有关用于关系数据库的对等 SMT 的详情，请查看新记录状态扩展 SMT 的文档。

以下主题提供详情：

- 第 13.8.1 节 “Debezium MongoDB 更改事件结构的描述”
- 第 13.8.2 节 “Debezium MongoDB 事件扁平化转换的行为”
- 第 13.8.3 节 “配置 Debezium MongoDB 事件扁平化转换”
- 第 13.8.4 节 “MongoDB 事件消息中编码数组的选项”

13.8.1. Debezium MongoDB 更改事件结构的描述

Debezium MongoDB 连接器生成具有复杂结构的更改事件。每个事件信息包括以下部分：

源元数据

包括但不仅限于以下字段：

- 更改集合中数据的操作类型(create/insert、update 或 delete)。
- 发生更改的数据库和集合的名称。
- 在进行更改时标识的时间戳。
- 可选的事务信息。

文档数据

数据 前

当 Debezium 连接器的 capture.mode 设置为以下值之一时，此字段存在于运行 MongoDB 6.0 及之后的版本的环境中：

- *change_streams_with_pre_image.*
- *change_streams_update_full_with_pre_image.*

如需更多信息，请参阅 MongoDB pre-image 支持

在 数据后

代表当前操作后文档中存在的值的 JSON 字符串。事件消息中存在 after 字段取决于事件类型和连接器配置。MongoDB 插入 操作的 create 事件始终包含一个 after 字段，而不考虑 capture.mode 设置。对于 更新 事件，只有在 capture.mode 设置为以下值之一时才会出现 after 字段：

- *change_streams_update_full*
- *change_streams_update_full_with_pre_image.*

注意

更改事件消息中的 after 值不一定在事件后立即代表文档的状态。该值不会动态计算；在连接器捕获更改事件后，它会查询集合来检索文档的当前值。

例如，假设一种情况，它有多个操作 a, b, 和 c 在快速成功中修改文档。当连接器处理时，修改 a，它会查询集合以获取完整的文档。同时，更改 b 和 c 发生。当连接器收到对更改 a 的完整文档的响应时，可能会收到一个基于 b 或 c 后续更改的文档版本。如需更多信息，请参阅 capture.mode 属性的文档。

以下片段显示了在 MongoDB 插入 操作后连接器发出的 create change 事件的基本结构：

```
{
  "op": "c",
  "after": "{\"field1\":\"newvalue1\",\"field2\":\"newvalue1\"}",
  "source": { ... }
}
```

上例中的 after 字段的复杂格式提供有关源数据库中更改的详细信息。但是，有些使用者无法处理包含嵌套值的消息。要将原始消息的复杂嵌套字段转换为更简单、更通用的兼容结构，请为 MongoDB 使用事件扁平化 SMT。SMT 扁平化消息中嵌套字段的结构，如下例所示：

```
{
  "field1" : "newvalue1",
  "field2" : "newvalue2"
}
```

有关 Debezium MongoDB 连接器生成的默认消息结构的更多信息，请参阅 连接器文档。

13.8.2. Debezium MongoDB 事件扁平化转换的行为

MongoDB 的事件扁平化 SMT 会从 Debezium MongoDB 连接器发送的 create 或 update 更改事件消息中拉取 after 字段。在 SMT 处理原始更改事件消息后，它会生成一个简化的版本，该版本仅包含 after 字段的内容。

根据您的用例，您可以将 ExtractNewDocumentState SMT 应用到 Debezium MongoDB 连接器，或应用到消耗 Debezium 连接器生成的消息的接收器连接器。如果您将 SMT 应用到 Debezium MongoDB 连接器，则 SMT 会修改连接器在发送到 Apache Kafka 之前发送的消息。为确保 Kafka 以原始格式保留完整的 Debezium 更改事件信息，请将 SMT 应用到接收器连接器。

当您使用事件扁平化 SMT 处理从 MongoDB 连接器发送的消息时，SMT 会将原始消息中的记录结构转换为正确输入的 Kafka Connect 记录，这些记录可以被典型的 sink 连接器使用。例如，SMT 将原始消息中的 after 信息转换为任何消费者可以处理的模式结构的 JSON 字符串。

另外，您可以为 MongoDB 配置事件扁平化 SMT，以便在处理过程中以其他方式修改消息。如需更多信息，请参阅配置 主题。

13.8.3. 配置 Debezium MongoDB 事件扁平化转换

为 MongoDB 配置事件扁平化(ExtractNewDocumentState) SMT，用于使用 Debezium MongoDB 连接器发送的消息。

以下主题提供详情：

- 第 13.8.3.1 节 “示例： Debezium MongoDB event flattening-transformation 的基本配置”
- 第 13.8.4 节 “MongoDB 事件消息中编码数组的选项”
- 第 13.8.5 节 “在 MongoDB 事件消息中扁平化嵌套结构”
- 第 13.8.6 节 “Debezium MongoDB 连接器如何报告由 $unset 操作删除的字段名称”
- 第 13.8.7 节 “确定原始数据库操作的类型”
- 第 13.8.8 节 “使用 MongoDB 事件扁平化 SMT 将 Debezium 元数据添加到 Kafka 记录”
- 第 13.8.9 节 “用于有选择地应用 MongoDB 提取新文档状态转换的选项”
- 第 13.8.10 节 “MongoDB 的 Debezium 事件扁平化转换的配置选项”

13.8.3.1. 示例： Debezium MongoDB event flattening-transformation 的基本配置

要获取 SMT 的默认行为，请在没有指定任何选项的情况下将 SMT 添加到接收器连接器配置中，如下例所示：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.connector.mongodb.transforms.ExtractNewDocumentState
```

与任何 Kafka Connect 连接器配置一样，您可以将 transformations= 设置为多个、以逗号分隔的 SMT 别名。Kafka Connect 应用您在列出的顺序中指定的转换。

您可以为使用 MongoDB 事件扁平化 SMT 的连接器设置多个选项。以下示例显示了为连接器设置 drop.tombstones、delete.handling.mode 和 add.headers 选项的配置：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.connector.mongodb.transforms.ExtractNewDocumentState
transforms.unwrap.drop.tombstones=false
transforms.unwrap.delete.handling.mode=drop
transforms.unwrap.add.headers=op
```

有关上例中的配置选项的更多信息，请参阅 配置主题。

自定义配置

连接器可能会发出许多类型的事件信息（如 heartbeat 消息、tombstone 消息或有关事务的元数据信息）。要将转换应用到事件子集，您可以定义一个 SMT predicate 语句，该语句有选择地将转换 应用到特定的事件。

13.8.4. MongoDB 事件消息中编码数组的选项

默认情况下，事件扁平化 SMT 将 MongoDB 阵列转换为与 Apache Kafka Connect 或 Apache Avro 模式兼容的数组。虽然 MongoDB 数组可以包含多个类型元素，但 Kafka 数组中的所有元素都必须是相同的类型。

要确保 SMT 对数组的满足环境需求的方式进行编码，您可以指定 array.encoding 配置选项。以下示例显示了设置数组编码的配置：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.connector.mongodb.transforms.ExtractNewDocumentState
transforms.unwrap.array.encoding=<array|document>
```

根据配置，SMT 通过使用以下编码方法之一处理源消息中数组的每个实例：

阵列编码

如果将 array.encoding 设置为 数组 （默认），则 SMT encodes 使用 数组 datatype 在原始消息中对数组进行编码。为确保处理正确，数组实例中的所有元素都必须相同类型。这个选项是一个限制，但它可让下游客户端轻松处理数组。

文档编码

如果将 array.encoding 设置为 document，则 SMT 会将源中的每个阵列转换为 structs 中的一个 struct，类似于 BSON 序列化。main struct 包含名为_0、 _1、_2 等的字段，其中每个字段名称代表原始阵列中元素的索引。SMT 使用为源数组中等同元素获取的值填充这些 index 字段。索引名称的前缀为下划线，因为 Avro 编码禁止以数字字符开头的字段名称。

以下示例显示了 Debezium MongoDB 连接器如何代表一个数据库文档，其中包含一个包含异构数据类型的数组：

例 13.2. 示例：记录包含多个数据类型的数组编码

```
{
   "_id": 1,
   "a1": [
      {
         "a": 1,
         "b": "none"
      },
      {
         "a": "c",
         "d": "something"
      }
   ]
}
```

如果将 array.encoding 设置为 document，则 SMT 会将前面的文档转换为以下格式：

```
{
   "_id": 1,
   "a1": {
      "_0": {
         "a": 1,
         "b": "none"
      },
      "_1": {
         "a": "c",
         "d": "something"
      }
   }
}
```

文档 编码选项可让 SMT 处理由异构元素组成的任意数组。但是，在使用这个选项前，请始终验证 sink 连接器和其他下游用户是否可以处理包含多个数据类型的数组。

13.8.5. 在 MongoDB 事件消息中扁平化嵌套结构

当数据库操作涉及嵌入式文档时，Debezium MongoDB 连接器会发出一个 Kafka 事件记录，其结构反映了原始文档的分级结构。也就是说，事件消息将嵌套文档表示为一组嵌套字段结构。在下游连接器无法处理包含嵌套结构的消息的环境中，您可以将事件扁平化 SMT 配置为在消息中的扁平化分级结构。扁平消息结构更适合类似表格的存储。

要将 SMT 配置为扁平化嵌套结构，请将 flatten.struct 配置选项设置为 true。在转换的消息中，字段名称被构建为与文档源一致。SMT 通过将父文档字段的名称与嵌套文档字段的名称连接来重命名每个扁平化字段。flatten.struct.delimiter 选项定义的分隔符会将名称的组件分开来。struct.delimiter 的默认值是一个下划线字符(_)。

以下示例显示了指定 SMT flattens 嵌套结构的配置：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.connector.mongodb.transforms.ExtractNewDocumentState
transforms.unwrap.flatten.struct=<true|false>
transforms.unwrap.flatten.struct.delimiter=<string>
```

以下示例显示了由 MongoDB 连接器发出的事件消息。消息包括了一个文档 a 的字段，其中包含两个嵌套文档 (b 和 c) 的字段：

```
{
   "_id": 1,
   "a": {
         "b": 1,
         "c": "none"
   },
   "d": 100
}
```

以下示例中的消息显示了 MongoDB flattens 在以上消息中嵌套结构的 SMT 后的输出：

```
{
   "_id": 1,
   "a_b": 1,
   "a_c": "none",
   "d": 100
}
```

在生成的消息中，原始消息中嵌套的 b 和 c 字段将被扁平化并重命名。重命名的字段将父文档 a 的名称与嵌套文档的名称连接在一起： a_b 和 a_c 。新字段名称的组件通过下划线字符分隔，具体由 struct.delimiter 配置属性的定义。

13.8.6. Debezium MongoDB 连接器如何报告由 $unset 操作删除的字段名称

在 MongoDB 中，$unset operator 和 $rename operator 都从文档中删除字段。因为 MongoDB 集合是无模式的，因此在更新从文档中删除了字段后，无法推断更新文档中缺失字段的名称。为了支持接收器连接器或其他可能需要有关删除字段信息的消费者，Debezium 会发出更新消息，其中包含一个 removedFields 元素，其中列出了已删除字段的名称。

以下示例显示一个操作的一个更新消息的部分，这导致删除了 a 字段：

```
"payload": {
  "op": "u",
  "ts_ms": "...",
  "before": "{ ... }",
  "after": "{ ... }",
  "updateDescription": {
    "removedFields": ["a"],
    "updatedFields": null,
    "truncatedArrays": null
  }
}
```

在上例中，before 和 after 代表文档更新以前和以后的源文档的状态。只有在为连接器设置了 capture.mode 时，连接器才会发出这些字段，如以下列表中所述：

before 字段

提供更改前文档的状态。只有在 capture.mode 设置为以下值之一时才会出现此字段：

- change_streams_with_pre_image
- change_streams_update_full_with_pre_image.

after 字段

在更改后提供文档的完整状态。只有在 capture.mode 设置为以下值之一时才会出现此字段：

- change_streams_update_full
- change_streams_update_full_with_pre_image.

假设配置为捕获完整文档的连接器，当 ExtractNewDocumentState SMT 收到 $unset 事件 的更新 消息时，通过代表删除的字段具有 null 值来重新编码消息，如下例所示：

```
{
   "id": 1,
```

```
    "a": null
}
```

对于没有配置为捕获完整文档的连接器，当 SMT 收到 $unset 操作的 update 事件时，它会生成以下输出信息：

```
{
  "a": null
}
```

13.8.7. 确定原始数据库操作的类型

在 SMT flattens 事件消息后，生成的消息不再指示生成事件是 create,update 或 initial snapshot read 的操作。通常，您可以通过将连接器配置为公开与 删除 附带的 tombstone 或 rewrite 事件的信息来识别删除操作。有关将连接器配置为公开有关 tombstones 并重写事件信息的更多信息，请参阅 drop.tombstones 和 delete.handling.mode 属性。

要在事件消息中报告数据库操作类型，SMT 可以在以下元素之一中添加 op 字段：

- 事件消息正文。
- 消息标头。

例如，要添加一个标头属性来显示原始操作的类型，添加转换，然后将 add.headers 属性添加到连接器配置中，如下例所示：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.connector.mongodb.transforms.ExtractNewDocumentState
transforms.unwrap.add.headers=op
```

根据前面的配置，SMT 通过向消息添加 op 标头来报告事件类型，并为它分配一个字符串值来标识操作类型。分配的字符串值基于原始 MongoDB 更改事件消息 中的 op 字段值。

13.8.8. 使用 MongoDB 事件扁平化 SMT 将 Debezium 元数据添加到 Kafka 记录

MongoDB 的事件扁平化 SMT 可将原始更改事件消息中的 metadata 字段添加到简化的消息中。添加的元数据字段的前缀为双下划线("__")。在事件记录中添加元数据可以包括内容，如发生更改事件的集合

名称，或包含特定于连接器的字段，如副本集名称。目前，SMT 只能从以下更改事件子结构中添加字段：source, transaction 和 updateDescription。

有关 MongoDB 更改事件结构的更多信息，请参阅 MongoDB 连接器文档。

例如，您可以指定以下配置，将副本集名称(rs)和更改事件的集合名称添加到最终扁平化事件记录中：

```
transforms=unwrap,...
transforms.unwrap.type=io.debezium.connector.mongodb.transforms.ExtractNewDocumentState
transforms.unwrap.add.fields=rs,collection
```

前面的配置会导致以下内容添加到扁平化的记录中：

```
{ "__rs" : "rs0", "__collection" : "my-collection", ... }
```

如果您希望 SMT 添加 metadata 字段 来删除 事件，请将 delete.handling.mode 选项的值设置为 rewrite。

13.8.9. 用于有选择地应用 MongoDB 提取新文档状态转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。

有关如何有选择地应用 SMT 的更多信息，请参阅为转换配置 SMT predicate。

13.8.10. MongoDB 的 Debezium 事件扁平化转换的配置选项

下表描述了 MongoDB 事件扁平化 SMT 的配置选项。

属性	默认	描述

属性	默认	描述
array.encoding	**数组**	指定 SMT 在从原始事件信息读取的数组时使用的格式。设置以下选项之一： **数组** SMT 使用 **数组** datatype 将 MongoDB 阵列编码为与 Apache Kafka Connect 或 Apache Avro 模式兼容的格式。如果设置了这个选项，请验证每个阵列实例中元素是否使用相同的类型。虽然 MongoDB 允许数组包含多个数据类型，但一些下游客户端无法处理数组。 **文档** SMT 将每个 MongoDB 数组转换为 **structs** 的一个 **struct**，其方式与 BSON 序列化 类似。main **struct** 包含名为 **_0**、 **_1**、 **_2** 等的字段。为了遵守 Avro 命名标准，SMT 为每个 index 字段的数字名称添加下划线前缀。每个数字字段名称代表原始阵列中元素的索引。SMT 使用从指定数组元素的源文档获取的值填充这些 index 字段的值。 有关 **array.coding** 选项的更多信息，请参阅 MongoDB 事件消息中编码数组的选项。
flatten.struct	**false**	通过串联消息中嵌套属性的名称（由可配置的分隔符分隔）来组成简单的字段名称，原始事件消息中的 SMT flattens 结构(structs)。
flatten.struct.delimiter	**_**	当 **flatten.struct** 设为 **true** 时，指定转换在从输入记录串联的字段名称之间插入的分隔符，以便在输出记录中生成字段名称。
drop.tombstones	**true**	Debezium 为每个 **delete** 操作生成一个 tombstone 记录。默认的行为是事件扁平化 SMT 从流中删除 tombstone 记录。要在流中保留 tombstone 记录，请指定 **drop.tombstones=false**。

属性	默认	描述
delete.handling.mode	**drop**	指定 SMT 如何处理 Debezium 为 **删除操作** 生成的更改事件记录。设置以下选项之一： **drop** SMT 从事件流 **中删除** 删除操作的记录。 **none** SMT 从事件流中保留原始更改事件记录。记录仅包含 **"value": "null** "。 **rewrite** SMT 从流中保留更改事件记录的修改版本。为了提供另一种指示已删除记录的方法，修改后的记录包含一个 **value** 字段，其中包含来自原始记录的键/值对，并将 **__deleted: true** 添加到 **值** 中。 如果您设置了 **rewrite** 选项，您可能会发现，对 **DELETE** 操作的更新简化记录足以跟踪已删除的记录。在这种情况下，您可能需要 SMT 丢弃 tombstone 记录。
add.headers.prefix	__ (double-underscore)	将此可选字符串设为前缀标头。

add.headers	没有默认值	指定以逗号分隔的列表，没有空格，这是您希望 SMT 添加到简化消息的标头中的元数据字段。当原始消息包含重复字段名称时，您可以通过提供 struct 的名称以及字段的名称来识别要修改的特定字段，如 **source.ts_ms**。
		
		另外，您还可以覆盖字段的原始名称，并通过在列表中添加以下格式的条目来为其分配新名称：
		
		<field_name>:<new_field_name>.
		
		例如：
		
		`version:VERSION,`
		`connector:CONNECTOR,`
		`source.ts_ms:EVENT_TIMESTAMP`
		
		您指定的新名称值区分大小写。
		
		当 SMT 将 metadata 字段添加到简化消息的标头中时，它会使用双下划线前缀每个 metadata 字段名称。对于 struct 规格，SMT 也在 struct 名称和字段名称之间插入下划线。
		
		如果您指定了不在更改事件原始消息中的字段，则 SMT 不会将字段添加到标头中。

add.fields.prefix	__ (double-underscore)	指定作为字段名称前缀的可选字符串。

属性	默认	描述

add.fields	没有默认值	将这个选项设置为以逗号分隔的列表，没有空格，使用 metadata 字段添加到简化的 Kafka 消息的 **value** 元素中。当原始消息包含重复字段名称时，您可以通过提供 struct 的名称以及字段的名称来识别要修改的特定字段，如 **source.ts_ms**。
		另外，您还可以覆盖字段的原始名称，并通过在列表中添加以下格式的条目来为其分配新名称：
		<field_name>:<new_field_name>.
		例如：
		`version:VERSION,`
		`connector:CONNECTOR,`
		`source.ts_ms:EVENT_TIMESTAMP`
		您指定的新名称值区分大小写。
		当 SMT 将 metadata 字段添加到简化消息的 **value** 元素中时，它会使用双下划线前缀每个 metadata 字段名称。对于 struct 规格，SMT 也在 struct 名称和字段名称之间插入下划线。
		如果您指定了原始更改事件消息中不存在的字段，则 SMT 仍然会将指定的字段添加到修改消息的 **value** 元素中。

已知限制

- ***因为 MongoDB 是一个无模式数据库，所以在使用 Debezium 流更改模式的数据相关数据库时，以确保列定义的一致性，所以具有相同名称的集合中的字段必须存储相同类型的数据。***

- ***配置 SMT 以格式生成与接收器连接器兼容的信息。如果接收器(sink)连接器需要"flat"消息结构，但它收到一个将源 MongoDB 文档中的一个数组编码为 structs 的消息，则接收器连接器无法处理该消息。***

13.9. 配置 DEBEZIUM 连接器以使用 OUTBOX 模式

outbox 模式是在多个(micro)服务之间安全可靠地交换数据的方法。开箱即用模式实施可避免服务内部状态（通常在数据库中保留）和需要相同数据的服务所消耗的事件之间的不一致。

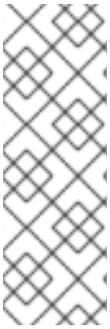
要在 Debezium 应用程序中实施 outbox 模式，请将 Debezium 连接器配置为：

- 捕获 outbox 表中的更改
- 应用 Debezium outbox 事件路由器单一消息转换(SMT)

配置为应用 outbox SMT 的 Debezium 连接器应该只捕获 outbox 表中发生的更改。如需更多信息，请参阅 有选择地应用转换的选项。

只有在每个 outbox 表具有相同的结构时，连接器才可以捕获多个 outbox 表中的更改。

请参阅 带有 Outbox Pattern 的 Reliable microservicess Data Exchange，以了解 outbox 模式很有用的原因及其工作方式。



注意

outbox 事件路由器 SMT 与 MongoDB 连接器不兼容。

MongoDB 用户可以运行 MongoDB outbox 事件路由器 SMT。

以下主题提供详情：

- 第 13.9.1 节 “Debezium outbox 消息示例”
- 第 13.9.2 节 “Debezium outbox 事件路由器 SMT 期望的 outbox 表结构”
- 第 13.9.3 节 “基本 Debezium outbox 事件路由器 SMT 配置”
- 第 13.9.4 节 “用于有选择地应用 Outbox 事件路由器转换的选项”
- 第 13.9.5 节 “在 Debezium outbox 消息中使用 Avro 作为有效负载格式”

- 第 13.9.6 节 “在 Debezium outbox 消息中发出其他字段”
- 第 13.9.7 节 “扩展转义的 JSON 字符串作为 JSON”
- 第 13.9.8 节 “用于配置 outbox 事件路由器转换的选项”

13.9.1. Debezium outbox 消息示例

要了解 Debezium outbox 事件路由器 SMT 是如何配置的，请查看以下 Debezium outbox 消息示例：

```
# Kafka Topic: outbox.event.order
# Kafka Message key: "1"
# Kafka Message Headers: "id=4d47e190-0402-4048-bc2c-89dd54343cdc"
# Kafka Message Timestamp: 1556890294484
{
  "{\"id\": 1, \"lineItems\": [{\"id\": 1, \"item\": \"Debezium in Action\", \"status\": \"ENTERED\",
\"quantity\": 2, \"totalPrice\": 39.98}, {\"id\": 2, \"item\": \"Debezium for Dummies\", \"status\":
\"ENTERED\", \"quantity\": 1, \"totalPrice\": 29.99}], \"orderDate\": \"2019-01-31T12:13:01\",
\"customerId\": 123}"
}
```

配置为应用 outbox 事件路由器 SMT 的 Debezium 连接器通过转换 Debezium 原始消息来生成上述消息，如下所示：

```
# Kafka Message key: "406c07f3-26f0-4eea-a50c-109940064b8f"
# Kafka Message Headers: ""
# Kafka Message Timestamp: 1556890294484
{
  "before": null,
  "after": {
    "id": "406c07f3-26f0-4eea-a50c-109940064b8f",
    "aggregateid": "1",
    "aggregatetype": "Order",
    "payload": "{\"id\": 1, \"lineItems\": [{\"id\": 1, \"item\": \"Debezium in Action\", \"status\":
\"ENTERED\", \"quantity\": 2, \"totalPrice\": 39.98}, {\"id\": 2, \"item\": \"Debezium for
Dummies\", \"status\": \"ENTERED\", \"quantity\": 1, \"totalPrice\": 29.99}], \"orderDate\":
\"2019-01-31T12:13:01\", \"customerId\": 123}",
    "timestamp": 1556890294344,
    "type": "OrderCreated"
  },
  "source": {
    "version": "2.3.4.Final",
    "connector": "postgresql",
    "name": "dbserver1-bare",
```

```
    "db": "orderdb",
    "ts_usec": 1556890294448870,
    "txId": 584,
    "lsn": 24064704,
    "schema": "inventory",
    "table": "outboxevent",
    "snapshot": false,
    "last_snapshot_record": null,
    "xmin": null
  },
  "op": "c",
  "ts_ms": 1556890294484
}
```

此 Debezium outbox 消息示例基于 默认的 outbox 事件路由器配置，它假定基于聚合的 outbox 表结构和事件路由。要自定义行为，outbox 事件路由器 SMT 提供了大量 配置选项。

13.9.2. Debezium outbox 事件路由器 SMT 期望的 outbox 表结构

要应用 default outbox 事件路由器 SMT 配置，您的 outbox 表被认为具有以下列：

```
Column       |          Type          | Modifiers
-------------+------------------------+-----------
id           | uuid                   | not null
aggregatetype | character varying(255) | not null
aggregateid   | character varying(255) | not null
type          | character varying(255) | not null
payload       | jsonb                  |
```

表 13.8. 预期的 outbox 表列的描述

column	效果
id	包含事件的唯一 ID。在 outbox 消息中，这个值是一个标头。例如，您可以使用此 ID 删除重复的消息。 要从不同的 outbox 表列中获取事件的唯一 ID，请在连接器配置中设置 **table.field.event.id** SMT 选项。

column	效果
aggregatetype	包含 SMT 附加到连接器发出 outbox 消息的主题名称中的值。默认行为是，这个值替换了 **route.topic.replacement** SMT 选项中的默认 **${routedByValue}** 变量。 例如，在默认配置中，**route.by.field** SMT 选项被设置为 **aggregatetype**，**route.topic.replacement** SMT 选项被设置为 **outbox.event.${routedByValue}**。假设您的应用程序在 outbox 表中添加两个记录。在第一条记录中，**aggregatetype** 列中的值是 **customers**。在第二条记录中，**aggregatetype** 列中的值是 **orders**。连接器将第一个记录发送到 **outbox.event.customers** 主题。连接器向 **outbox.event.orders** 主题发送第二个记录。 要从不同的 outbox 表列中获取这个值，请在连接器配置中设置 **route.by.field** SMT 选项。
aggregateid	包含事件密钥，它为有效负载提供 ID。SMT 使用这个值作为 emitted outbox 消息中的键。这对于在 Kafka 分区中维护正确顺序非常重要。 要从不同的 outbox 表列中获取事件键，请在连接器配置中设置 **table.field.event.key** SMT 选项。
payload	outbox 更改事件的表示。默认结构是 JSON。默认情况下，Kafka message 值只由 **有效负载** 值组成。但是，如果 outbox 事件被配置为包含其他字段，Kafka 消息值包含信封封装器和附加字段，并且每个字段都单独表示。如需更多信息，请参阅使用其他字段传输消息。 要从不同的 outbox 表列中获取事件有效负载，请在连接器配置中设置 **table.field.event.payload** SMT 选项。
其他自定义列	outbox 表中的任何其他列 可以添加到 outbox 事件， 可以在 payload 部分或作为消息标头中添加。 一个示例可以是列 **eventType**，它传达用户定义的值，有助于分类或组织事件。

13.9.3. 基本 Debezium outbox 事件路由器 SMT 配置

要将 Debezium 连接器配置为支持 outbox 模式，请配置 outbox.EventRouter SMT。要获取 SMT 的默认行为，请在没有指定任何选项的情况下将其添加到连接器配置中，如下例所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
```

自定义配置

连接器可能会发出许多类型的事件信息（例如： heartbeat 消息、tombstone 消息或有关事务或模式

更改的元数据信息）。要将转换应用到源自 outbox 表中的事件，请定义一个 有选择地将转换应用到这些事件的 SMT predicate 语句。

13.9.4. 用于有选择地应用 Outbox 事件路由器转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。您可以使用以下任一方法配置连接器来有选择地应用 SMT：

- 为转换配置 SMT predicate。
- 对 SMT 使用 route.topic.regex 配置选项。

13.9.5. 在 Debezium outbox 消息中使用 Avro 作为有效负载格式

outbox 事件路由器 SMT 支持任意有效负载格式。outbox 表中的 payload 列值以透明的方式传递。使用 JSON 的替代方法是使用 Avro。这对消息格式监管很有用，并确保 outbox 事件模式以向后兼容的方式发展。

源应用程序如何为 outbox 消息有效负载生成 Avro 格式内容超出了本文档的范围。可以利用 KafkaAvroSerializer 类序列化 GenericRecord 实例。要确保 Kafka message 值是准确的 Avro 二进制数据，请将以下配置应用到连接器：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
value.converter=io.debezium.converters.BinaryDataConverter
```

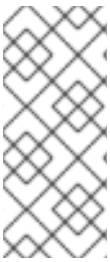
默认情况下，有效负载 列值(Avro 数据)是唯一的消息值。配置 BinaryDataConverter，因为值转换器将 payload 列值按原样传播到 Kafka 消息值中。

Debezium 连接器可以配置为发送心跳、事务元数据或模式更改事件（支持因连接器而异）。这些事件无法通过 BinaryDataConverter 序列化，因此必须提供额外的配置，因此转换器知道如何序列化这些事件。例如，以下配置演示了使用没有模式的 Apache Kafka JsonConverter ：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
value.converter=io.debezium.converters.BinaryDataConverter
value.converter.delegate.converter.type=org.apache.kafka.connect.json.JsonConverter
value.converter.delegate.converter.type.schemas.enable=false
```

-

delegate Converter 实现通过 delegate.converter.type 选项指定。如果转换器需要任何其他配置选项，也可以指定它们，如使用 schemas.enable=false 显示的模式禁用。



注意

从 Debezium 版本 1.9 开始，converter io.debezium.converters.ByteBufferConverter 已被弃用，并在 2.0 中删除。另外，在使用 Kafka Connect 时，必须在升级到 Debezium 2.x 前更新连接器的配置。

13.9.6. 在 Debezium outbox 消息中发出其他字段

您的 outbox 表可能包含您要添加到发出出出出出消息的值的列。例如，一个 outbox 表，它在 aggregatetype 列中具有 购买顺序 的值，另一个列, eventType，其可能的值是 order-created 和 order-shipped。可以使用语法 column:placement:alias 来添加其他字段。

放置 允许的值有： - header - envelope - partition

要在 outbox 消息标头中发出 eventType 列值，请配置 SMT，如下所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.table.fields.additional.placement=eventType:header:type
```

结果将是 Kafka 消息的标头，类型 作为其键，eventType 列的值作为其值。

要在 outbox 消息 envelope 中发出 eventType 列值，请配置 SMT，如下所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.table.fields.additional.placement=eventType:envelope:type
```

要控制在其上生成 outbox 消息的分区，请配置 SMT，如下所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.table.fields.additional.placement=partitionColumn:partition
```

请注意，对于 分区 放置，添加别名将无效。

13.9.7. 扩展转义的 JSON 字符串作为 JSON

您可能注意到，Debezium outbox 消息包含以 String 表示 的有效负载。因此，当这个字符串实际上是 JSON 时，它会出现在结果 Kafka 信息中，如下所示：

```
# Kafka Topic: outbox.event.order
# Kafka Message key: "1"
# Kafka Message Headers: "id=4d47e190-0402-4048-bc2c-89dd54343cdc"
# Kafka Message Timestamp: 1556890294484
{
  "{\"id\": 1, \"lineItems\": [{\"id\": 1, \"item\": \"Debezium in Action\", \"status\": \"ENTERED\",
  \"quantity\": 2, \"totalPrice\": 39.98}, {\"id\": 2, \"item\": \"Debezium for Dummies\", \"status\":
  \"ENTERED\", \"quantity\": 1, \"totalPrice\": 29.99}], \"orderDate\": \"2019-01-31T12:13:01\",
  \"customerId\": 123}"
}
```

outbox 事件路由器允许您将此消息内容扩展为 "real" JSON，使用从 JSON 文档本身分离的编译模式。这样，Kafka 信息的结果类似如下：

```
# Kafka Topic: outbox.event.order
# Kafka Message key: "1"
# Kafka Message Headers: "id=4d47e190-0402-4048-bc2c-89dd54343cdc"
# Kafka Message Timestamp: 1556890294484
{
  "id": 1, "lineItems": [{"id": 1, "item": "Debezium in Action", "status": "ENTERED", "quantity":
  2, "totalPrice": 39.98}, {"id": 2, "item": "Debezium for Dummies", "status": "ENTERED",
  "quantity": 1, "totalPrice": 29.99}], "orderDate": "2019-01-31T12:13:01", "customerId": 123
}
```

要启用此转换，您必须将 table.expand.json.payload 设置为 true，并使用如下所示的 JsonConverter ：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.table.expand.json.payload=true
value.converter=org.apache.kafka.connect.json.JsonConverter
```

13.9.8. 用于配置 outbox 事件路由器转换的选项

下表描述了您可以为 outbox 事件路由器 SMT 指定的选项。在表格中，Group 列指示 Kafka 的配置选项分类。

表 13.9. outbox 事件路由器 SMT 配置选项的描述

选项	默认	组	描述
table.op.invalid.behavior	**warn**	表	决定在 outbox 表中有 **UPDATE** 操作时 SMT 的行为。可能的设置有： • **warn** - SMT 会记录警告并继续下一个 outbox 表记录。 • **error** - SMT 记录错误，并继续到下一个 outbox 表记录。 • **fatal** - SMT 日志错误，连接器会停止处理。 outbox 表中的所有更改都应该是 **INSERT** 操作。也就是说，不允许将 outbox 表作为队列；不允许对 outbox 表中的记录进行更新。SMT 自动过滤出 out out a outbox 表中的 **DELETE** 操作。
table.field.event.id	**id**	表	指定包含唯一事件 ID 的 outbox 表列。此 ID 将存储在 **id** 键下发出的事件标头中。
table.field.event.key	**aggregateid**	表	指定包含 event 键的 outbox 表列。当此列包含值时，SMT 会使用该值作为 emitted outbox 信息中的键。这对于在 Kafka 分区中维护正确的顺序非常重要。
table.field.event.timestamp		表	默认情况下，emitted outbox 消息中的时间戳是 Debezium 事件时间戳。要在 outbox 消息中使用不同的时间戳，请将此选项设置为包含您要发出出出消息的时间戳的 outbox 表列。
table.field.event.payload	**payload**	表	指定包含事件有效负载的 outbox 表列。
table.expand.json.payload	**false**	表	指定是否应进行 String 有效负载的 JSON 扩展。如果没有找到内容，或者在解析错误时保留内容，则会保留"as is"。 如需了解更多详细信息，请参阅 扩展转义的 json 部分。

选项	默认	组	描述
table.json.payload.null.behavior	**ignore**	表	当启用 JSON 扩展属性 **table.expand.json.payload** 时，请确定 json 有效负载的行为，其中包括 outbox 表中的 **null** 值。可能的设置有： • **ignore** - 忽略 null 值。 • **optional_bytes** - 保持 null 值，并将 null 视为可选的 connect 字节。
table.fields.additional.placement		table, Envelope	指定您要添加到 outbox 消息标头或信封的一个或多个 outbox 表列。指定以逗号分隔的对列表。在每个对中，指定列的名称，以及是否希望该值在标头中还是信封。使用冒号分隔对中的值，例如： **id:header,my-field:envelope** 要为列指定别名，请将别名指定为 trio，例如： **id:header,my-field:envelope:my-alias** 第二个值是放置，它必须始终是 **标头** 或 **信封**。 配置示例包括在 emitting additional fields in Debezium outbox messages 中。
table.field.event.schema.version		table, Schema	设置后，这个值将用作 schema 版本，如 Kafka Connect Schema Javadoc 所述。
route.by.field	**aggregatetype**	路由器	指定 outbox 表中的列名称。默认行为是此列中的值成为连接器发出 outbox 消息的主题名称的一部分。示例位于 预期 outbox 表的描述 中。
route.topic.regex	**(?<routedByValue>.*)**	路由器	指定 outbox SMT 在 RegexRouter 中应用到 outbox 表记录的正则表达式。这个正则表达式是设置 **route.topic.replacement** SMT 选项的一部分。 默认的行为是 SMT 将 **route.topic.replacement** SMT 选项的设置中的默认 **${routedByValue}** 变量替换为 **route.by.field** outbox SMT 选项的设置。

选项	默认	组	描述
route.topic.replacement	outbox.event.${routedByValue}	路由器	指定连接器发出 outbox 消息的主题名称。默认主题名称为 outbox.event.，后跟 outbox 表记录中的 aggregatetype 列值。例如，如果 aggregatetype 值是 customers，则主题名称为 outbox.event.customers。 要更改主题名称，您可以： ● 将 route.by.field 选项设置为不同的列。 ● 将 route.topic.regex 选项设置为不同的正则表达式。
route.tombstone.on.empty.payload	false	路由器	指明一个空或 null 有效负载会导致连接器发出 tombstone 事件。

13.10. 配置 DEBEZIUM MONGODB 连接器以使用 OUTBOX 模式

注意

此 SMT 仅用于 Debezium MongoDB 连接器。有关将 outbox 事件路由器 SMT 用于关系数据库的详情，请参考 Outbox 事件路由器。

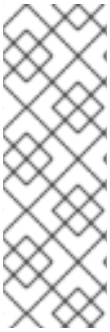
outbox 模式是在多个(micro)服务之间安全可靠地交换数据的方法。开箱即用模式实施可避免服务内部状态（通常在数据库中保留）和需要相同数据的服务所消耗的事件之间的不一致。

要在 Debezium 应用程序中实施 outbox 模式，请将 Debezium 连接器配置为：

- *捕获 outbox 集合中的更改*
- *应用 Debezium MongoDB outbox 事件路由器单个消息转换(SMT)*

配置为应用 MongoDB outbox SMT 的 Debezium 连接器应该只捕获 outbox 集合中发生的更改。如需更多信息，请参阅 有选择地应用转换的选项。

只有在每个 outbox 集合具有相同的结构时，连接器才可以捕获多个 outbox 集合中的更改。



注意

要使用这个 SMT，在实际业务集合和插入 outbox 集合中的操作必须是多文档事务的一部分，自 MongoDB 4.0 开始被支持，以防止商业集合和 outbox 集合之间潜在的数据不一致。对于将来的更新，要在没有多文档事务的情况下在 ACID 事务中启用更新现有数据并插入 outbox 事件，我们计划支持以现有集合的子文档的形式存储 outbox 事件的额外配置，而不是独立的 outbox 集合。

有关 outbox 模式的更多信息，请参阅使用 Outbox Pattern 的 Reliable microservicess Data Exchange。

以下主题提供详情：

- 第 13.10.1 节 “Debezium MongoDB outbox 消息示例”
- 第 13.10.2 节 “Debezium mongodb outbox 事件路由器 SMT 期望的 outbox 集合结构”
- 第 13.10.3 节 “基本 Debezium MongoDB outbox 事件路由器 SMT 配置”
- 第 13.10.5 节 “在 Debezium MongoDB outbox 消息中使用 Avro 作为有效负载格式”
- 第 13.10.6 节 “在 Debezium MongoDB outbox 信息中发出其他字段”
- 第 13.10.8 节 “用于配置 outbox 事件路由器转换的选项”

13.10.1. Debezium MongoDB outbox 消息示例

要了解如何配置 Debezium MongoDB outbox 事件路由器 SMT，请考虑以下 Debezium outbox 消息示例：

```
# Kafka Topic: outbox.event.order
```

```
# Kafka Message key: "b2730779e1f596e275826f08"
# Kafka Message Headers: "id=596e275826f08b2730779e1f"
# Kafka Message Timestamp: 1556890294484
{
  "{\"id\": {\"$oid\": \"da8d6de63b7745ff8f4457db\"}, \"lineItems\": [{\"id\": 1, \"item\":
\"Debezium in Action\", \"status\": \"ENTERED\", \"quantity\": 2, \"totalPrice\": 39.98}, {\"id\": 2,
\"item\": \"Debezium for Dummies\", \"status\": \"ENTERED\", \"quantity\": 1, \"totalPrice\":
29.99}], \"orderDate\": \"2019-01-31T12:13:01\", \"customerId\": 123}"
}
```

配置为应用 MongoDB outbox 事件路由器 SMT 的 Debezium 连接器通过转换原始 Debezium 更改事件信息来生成前面的消息，如下例所示：

```
# Kafka Message key: { "id": "{\"$oid\": \"596e275826f08b2730779e1f\"}" }
# Kafka Message Headers: ""
# Kafka Message Timestamp: 1556890294484
{
  "patch": null,
  "after": "{\"_id\": {\"$oid\": \"596e275826f08b2730779e1f\"}, \"aggregateid\": {\"$oid\":
\"b2730779e1f596e275826f08\"}, \"aggregatetype\": \"Order\", \"type\": \"OrderCreated\",
\"payload\": {\"_id\": {\"$oid\": \"da8d6de63b7745ff8f4457db\"}, \"lineItems\": [{\"id\": 1,
\"item\": \"Debezium in Action\", \"status\": \"ENTERED\", \"quantity\": 2, \"totalPrice\": 39.98},
{\"id\": 2, \"item\": \"Debezium for Dummies\", \"status\": \"ENTERED\", \"quantity\": 1,
\"totalPrice\": 29.99}], \"orderDate\": \"2019-01-31T12:13:01\", \"customerId\": 123}}",
  "source": {
    "version": "2.3.4.Final",
    "connector": "mongodb",
    "name": "fulfillment",
    "ts_ms": 1558965508000,
    "snapshot": false,
    "db": "inventory",
    "rs": "rs0",
    "collection": "customers",
    "ord": 31,
    "h": 1546547425148721999
  },
  "op": "c",
  "ts_ms": 1556890294484
}
```

此 Debezium outbox 消息示例基于 默认的 outbox 事件路由器配置，它假定基于聚合的 outbox 集合结构和事件路由。要自定义行为，outbox 事件路由器 SMT 提供了大量 配置选项。

13.10.2. Debezium mongodb outbox 事件路由器 SMT 期望的 outbox 集合结构

要应用默认的 MongoDB outbox 事件路由器 SMT 配置，您的 outbox 集合被假定为具有以下字段：

```
{
  "_id": "objectId",
```

```
  "aggregatetype": "string",
  "aggregateid": "objectId",
  "type": "string",
  "payload": "object"
}
```

表 13.10. 预期的 outbox 集合字段的描述

字段	效果
id	包含事件的唯一 ID。在 outbox 消息中，这个值是一个标头。例如，您可以使用此 ID 删除重复的消息。 要从不同的 outbox 集合字段获取事件的唯一 ID，请在连接器配置中设置 **collection.field.event.id** SMT 选项。
aggregatetype	包含 SMT 附加到连接器发出 outbox 消息的主题名称中的值。默认行为是，这个值替换了 **route.topic.replacement** SMT 选项中的默认 **${routedByValue}** 变量。 例如，在默认配置中，**route.by.field** SMT 选项被设置为 **aggregatetype**，**route.topic.replacement** SMT 选项被设置为 **outbox.event.${routedByValue}**。假设您的应用程序在 outbox 集合中添加两个文档。在第一个文档中，**aggregatetype** 字段中的值 **是客户**。在第二个文档中，**aggregatetype** 字段的值是 **orders**。连接器将第一个文档发送到 **outbox.event.customers** 主题。连接器将第二个文档发送到 **outbox.event.orders** 主题。 要从不同的 outbox 集合字段获取这个值，请在连接器配置中设置 **route.by.field** SMT 选项。
aggregateid	包含事件密钥，它为有效负载提供 ID。SMT 使用这个值作为 emitted outbox 消息中的键。这对于在 Kafka 分区中维护正确顺序非常重要。 要从不同的 outbox 集合字段获取事件键，请在连接器配置中设置 **collection.field.event.key** SMT 选项。
payload	outbox 更改事件的表示。默认结构是 JSON。默认情况下，Kafka message 值只由 **有效负载** 值组成。但是，如果 outbox 事件被配置为包含其他字段，Kafka 消息值包含信封封装器和附加字段，并且每个字段都单独表示。如需更多信息，请参阅使用其他字段传输消息。 要从不同的 outbox 集合字段获取事件有效负载，请在连接器配置中设置 **collection.field.event.payload** SMT 选项。
其他自定义字段	outbox 集合中的任何其他字段都可以 添加到 outbox 事件，可以在 payload 部分或作为消息标头中添加。 一个示例可以是字段 **eventType**，它传达用户定义的值，有助于分类或组织事件。

13.10.3. 基本 Debezium MongoDB outbox 事件路由器 SMT 配置

要将 Debezium MongoDB 连接器配置为支持 outbox 模式，请配置 outbox.MongoEventRouter SMT。要获取 SMT 的默认行为，请在没有指定任何选项的情况下将其添加到连接器配置中，如下例所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.connector.mongodb.transforms.outbox.MongoEventRouter
```

自定义配置

连接器可能会发出许多类型的事件信息（如 heartbeat 消息、tombstone 消息或有关事务的元数据信息）。要将转换应用到源自 outbox 集合中的事件，请定义一个 有选择地将转换应用到这些事件的 SMT predicate 语句。

13.10.4. 用于有选择地应用 MongoDB Outbox 事件路由器转换的选项

除了 Debezium 连接器在数据库更改时发出的更改事件消息外，连接器还会发出其他类型的信息，包括心跳消息，以及有关 schema 更改和事务的元数据消息。由于这些其他消息的结构与 SMT 设计的更改事件消息的结构不同，因此最好将连接器配置为有选择地应用 SMT，以便它只处理预期的数据更改消息。您可以使用以下任一方法配置连接器来有选择地应用 SMT：

- 为转换配置 SMT predicate。
- 对 SMT 使用 route.topic.regex 配置选项。

13.10.5. 在 Debezium MongoDB outbox 消息中使用 Avro 作为有效负载格式

MongoDB outbox 事件路由器 SMT 支持任意有效负载格式。outbox 集合中 有效负载 字段值以透明的方式传递。使用 JSON 的替代方法是使用 Avro。这对消息格式监管很有用，并确保 outbox 事件模式以向后兼容的方式发展。

源应用程序如何为 outbox 消息有效负载生成 Avro 格式内容超出了本文档的范围。可以利用 KafkaAvroSerializer 类序列化 GenericRecord 实例。要确保 Kafka message 值是准确的 Avro 二进制数据，请将以下配置应用到连接器：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.connector.mongodb.transforms.outbox.MongoEventRouter
value.converter=io.debezium.converters.ByteArrayConverter
```

默认情况下，payload 字段值(Avro 数据)是唯一消息值。配置 ByteArrayConverter，因为值转换器将有效负载 字段值按原样传播到 Kafka 消息值。

请注意，这与其他 SMT 建议的 BinaryDataConverter 不同。这是因为 MongoDB 在内部存储字节阵列的不同方法。

Debezium 连接器可以配置为发送心跳、事务元数据或模式更改事件（支持因连接器而异）。这些事件无法通过 ByteArrayConverter 序列化，因此必须提供额外的配置，因此转换器知道如何序列化这些事件。例如，以下配置演示了使用没有模式的 Apache Kafka JsonConverter ：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.connector.mongodb.transforms.outbox.MongoEventRouter
value.converter=io.debezium.converters.ByteArrayConverter
value.converter.delegate.converter.type=org.apache.kafka.connect.json.JsonConverter
value.converter.delegate.converter.type.schemas.enable=false
```

delegate Converter 实现通过 delegate.converter.type 选项指定。如果转换器需要任何其他配置选项，也可以指定它们，如使用 schemas.enable=false 显示的模式禁用。

13.10.6. 在 Debezium MongoDB outbox 信息中发出其他字段

outbox 集合可能会包含您要添加到发出出出出出消息的值的字段。例如，假设 outbox 集合的 aggregatetype 字段中的值为 purchase-order，另一个字段 eventType，其可能的值是 order-created 和 order-shipped。可以使用语法 field:placement:alias 来添加其他字段。

放置 允许的值有： - header - envelope - partition

要在 outbox 消息标头中发出 eventType 字段值，请配置 SMT，如下所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.collection.fields.additional.placement=eventType:header:type
```

结果将是 Kafka 消息的标头，其 type 作为其键，eventType 字段的值为其值。

要在 outbox 消息 envelope 中发出 eventType 字段值，请配置 SMT，如下所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.collection.fields.additional.placement=eventType:envelope:type
```

要控制在其上生成 outbox 消息的分区，请配置 SMT，如下所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.transforms.outbox.EventRouter
transforms.outbox.collection.fields.additional.placement=partitionField:partition
```

请注意，对于 分区 放置，添加别名将无效。

13.10.7. 扩展转义的 JSON 字符串作为 JSON

默认情况下，Debezium outbox 消息 的有效负载 以字符串表示。当字符串的原始源采用 JSON 格式时，生成的 Kafka 消息使用转义序列来代表字符串，如下例所示：

```
# Kafka Topic: outbox.event.order
# Kafka Message key: "1"
# Kafka Message Headers: "id=596e275826f08b2730779e1f"
# Kafka Message Timestamp: 1556890294484
{
  "{\"id\": {\"$oid\": \"da8d6de63b7745ff8f4457db\"}, \"lineItems\": [{\"id\": 1, \"item\":
\"Debezium in Action\", \"status\": \"ENTERED\", \"quantity\": 2, \"totalPrice\": 39.98}, {\"id\": 2,
\"item\": \"Debezium for Dummies\", \"status\": \"ENTERED\", \"quantity\": 1, \"totalPrice\":
29.99}], \"orderDate\": \"2019-01-31T12:13:01\", \"customerId\": 123}"
}
```

您可以配置 outbox 事件路由器以扩展消息内容，将转义的 JSON 转换为其原始的未转义 JSON 格式。在转换的字符串中，companion 模式会从原始 JSON 文档中分离。以下示例显示生成的 Kafka 信息中展开的 JSON：

```
# Kafka Topic: outbox.event.order
# Kafka Message key: "1"
# Kafka Message Headers: "id=596e275826f08b2730779e1f"
# Kafka Message Timestamp: 1556890294484
{
  "id": "da8d6de63b7745ff8f4457db", "lineItems": [{"id": 1, "item": "Debezium in Action",
"status": "ENTERED", "quantity": 2, "totalPrice": 39.98}, {"id": 2, "item": "Debezium for
Dummies", "status": "ENTERED", "quantity": 1, "totalPrice": 29.99}], "orderDate": "2019-01-
31T12:13:01", "customerId": 123
}
```

要在转换中启用字符串转换，将 collection.expand.json.payload 的值设置为 true，并使用 StringConverter，如下例所示：

```
transforms=outbox,...
transforms.outbox.type=io.debezium.connector.mongodb.transforms.outbox.MongoEventRouter
```

```
transforms.outbox.collection.expand.json.payload=true
value.converter=org.apache.kafka.connect.storage.StringConverter
```

13.10.8. 用于配置 outbox 事件路由器转换的选项

下表描述了您可以为 **outbox 事件路由器 SMT** 指定的选项。在表格中，**Group** 列指示 **Kafka** 的配置选项分类。

表 13.11. outbox 事件路由器 SMT 配置选项的描述

选项	默认	组	描述
collection.op.invalid.behavior	**warn**	集合	决定在 outbox 集合上有更新操作时 SMT 的行为。可能的设置有： • **warn** - SMT 会记录警告并继续下一个 outbox 集合文档。 • **Error** - SMT 会记录错误并继续下一个 outbox 集合文档。 • **fatal** - SMT 日志错误，连接器会停止处理。 outbox 集合中的所有更改都应该是插入或删除操作。也就是说，outbox 集合作为队列的功能；不允许对 outbox 集合中的文档进行更新。SMT 自动过滤掉的删除操作（用于删除开箱即用的事件）。
collection.field.event.id	**_id**	集合	指定包含唯一事件 ID 的 outbox 集合字段。此 ID 将存储在 **id** 键下发出的事件标头中。
collection.field.event.key	**aggregateid**	集合	指定包含 event 键的 outbox 集合字段。当此字段包含值时，SMT 会使用该值作为 emitted outbox 信息中的键。这对于在 Kafka 分区中维护正确的顺序非常重要。
collection.field.event.timestamp		集合	默认情况下，emitted outbox 消息中的时间戳是 Debezium 事件时间戳。要在 outbox 消息中使用不同的时间戳，请将这个选项设置为包含您要发出出出消息的时间戳的 outbox 集合字段。
collection.field.event.payload	**payload**	集合	指定包含事件有效负载的 outbox 集合字段。

选项	默认	组	描述
collection.expand.json.payload	**false**	集合	指定是否应进行 String 有效负载的 JSON 扩展。如果没有找到内容，或者在解析错误时保留内容，则会保留"as is"。 如需了解更多详细信息，请参阅 扩展转义的 json 部分。
collection.fields.additional.placement		collection, Envelope	指定您要添加到 outbox 消息标头或信封的一个或多个 outbox 集合字段。指定以逗号分隔的对列表。在每个对中，指定字段的名称，以及是否希望该值在标头中还是 envelope。使用冒号分隔对中的值，例如： **id:header,my-field:envelope** 要为字段指定别名，请将别名指定为 trio，例如： **id:header,my-field:envelope:my-alias** 第二个值是放置，它必须始终是 **标头** 或 **信封**。 配置示例包括在 emitting additional fields in Debezium outbox messages 中。
collection.field.event.schema.version		collection, Schema	设置后，这个值将用作 schema 版本，如 Kafka Connect Schema Javadoc 所述。
route.by.field	**aggregatetype**	路由器	指定 outbox 集合中字段的名称。默认情况下，此字段中指定的值成为连接器发出 outbox 消息的主题名称的一部分。有关示例，请查看预期 outbox 集合的描述。
route.topic.regex	**(?<routedByValue>.*)**	路由器	指定 outbox SMT 在 RegexRouter 中应用到 outbox 集合文档的正则表达式。这个正则表达式是设置 **route.topic.replacement** SMT 选项的一部分。 + 默认的行为是 SMT 将 **route.topic.replacement** SMT 选项的设置中的默认 **${routedByValue}** 变量替换为 **route.by.field** outbox SMT 选项的设置。

选项	默认	组	描述
route.topic.replacement	**outbox.event.${routedByValue}**	路由器	指定连接器发出 outbox 消息的主题名称。默认主题名称为 **outbox.event.**，后跟 outbox 集合文档中的 **aggregatetype** 字段值。例如，如果 **aggregatetype** 值是 **customers**，则主题名称为 **outbox.event.customers**。 + 要更改主题名称，您可以： ● 将 route.by.field 选项设置为不同的字段。 ● 将 route.topic.regex 选项设置为不同的正则表达式。
route.tombstone.on.empty.payload	**false**	路由器	指明一个空或 **null** 有效负载会导致连接器发出 tombstone 事件。

13.11. 根据有效负载字段将记录路由到分区

默认情况下，当 Debezium 检测到数据收集的变化时，它会发出的更改事件发送到使用单个 Apache Kafka 分区的主题。如 自定义 Kafka Connect 自动主题创建 中所述，您可以根据主密钥的哈希值自定义默认配置，将事件路由到多个分区。

然而，在某些情况下，您可能还希望 Debezium 将事件路由到特定的主题分区。分区路由 SMT 可让您根据一个或多个指定有效负载字段的值将事件路由到特定的目标分区。要计算目标分区，Debebe 使用指定字段值的哈希值。

13.11.1. 示例： Debezium 分区路由 SMT 的基本配置

您可以在 Debezium 连接器的 Kafka Connect 配置中配置分区路由转换。配置指定以下参数：

partition.payload.field

指定 SMT 用来计算目标分区的事件有效负载中的字段。您可以使用点表示法来指定嵌套有效负载字段。

partition.topic.num

指定目标主题中的分区数量。

partition.hash.function

指定用来决定目标分区数的字段哈希函数。

默认情况下，Debezium 将配置的数据收集的所有更改事件记录路由到单个 Apache Kafka 主题。连接器不会将事件记录定向到主题中的特定分区。

要将 Debezium 连接器配置为将事件路由到特定分区，请在 Debezium 连接器的 Kafka Connect 配置中配置 PartitionRouting SMT。

例如，您可以在连接器配置中添加以下配置。

```
...
topic.creation.default.partitions=2
topic.creation.default.replication.factor=1
...

topic.prefix=fulfillment
transforms=PartitionRouting
transforms.PartitionRouting.type=io.debezium.transforms.partitions.PartitionRouting
transforms.PartitionRouting.partition.payload.fields=change.name
transforms.PartitionRouting.partition.topic.num=2
transforms.PartitionRouting.predicate=allTopic
predicates=allTopic
predicates.allTopic.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
predicates.allTopic.pattern=fulfillment.*
...
```

根据前面的配置，当 SMT 收到一个名称以前缀开头的主题绑定的消息时，它会将消息重定向到特定的主题分区。

SMT 从消息有效负载中的 name 字段的值的哈希值计算目标分区。通过指定'allTopic' predicate，配置有选择地应用 SMT。更改 前缀是一个特殊关键字，它允许 SMT 自动引用有效负载中的元素，该元素描述数据的 before 或 after 状态。如果事件消息中没有指定的字段，则 SMT 会忽略它。如果消息中没有字段，则转换将完全忽略事件消息，并将消息的原始版本传送到默认目的地主题。SMT 配置中的 topic.num 设置指定的分区数量必须与 Kafka Connect 配置指定的分区数量匹配。例如，在前面的配置示例中，由 Kafka Connect 属性 topic.creation.default.partitions 指定的值与 SMT 配置中的 topic.num 值匹配。

根据 这个 产品表

表 13.12. 产品表

id	name	description	weight

101	scooter	small 2-wheel scooter	3.14
102	car battery	12v car battery	8.1
103	12-pack 深入位	12-pack ofdepth位，大小范围从 #40 到 #3	0.8
104	hammer	12Oz carpenter 的 hammer	0.75
105	hammer	14Oz carpenter 的 hammer	0.875
106	hammer	16Oz carpenter 的 hammer	1.0
107	rocks	分类的原则的方框	5.3
108	jacket	Water resistent black wind breaker	0.1
109	spare tire	24 个空闲备份	22.2

根据配置，SMT 将字段名称 hammer 的记录更改事件到同一分区。也就是说，id 值 104、105 和 106 的项目会被路由到同一分区。

13.11.2. 示例： Debezium 分区路由 SMT 的高级配置

假设您想要将来自两个数据收集(t1、t2)的事件路由到相同的主题（例如，my_topic），并且您希望使用字段 f1 从数据收集 t1 分区事件，并使用字段 f2 从数据收集 t2 分区事件。

您可以应用以下配置：

```
transforms=PartitionRouting
transforms.PartitionRouting.type=io.debezium.transforms.partitions.PartitionRouting
transforms.PartitionRouting.partition.payload.fields=change.f1,change.f2
transforms.PartitionRouting.partition.topic.num=2
transforms.PartitionRouting.predicate=myTopic

predicates=myTopic
predicates.myTopic.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
predicates.myTopic.pattern=my_topic
```

以上配置没有指定如何重新路由事件，以便它们发送到特定的目标主题。有关如何将事件发送到其默

认目标主题以外的主题的详情，请查看 主题路由 SMT。https://access.redhat.com/documentation/zh-cn/red_hat_integration/2023.q4/html-single/debezium_user_guide/index

13.11.3. 从 Debezium ComputePartition SMT 迁移

Debezium ComputePartition SMT 将在以后的发行版本中停用。下面的部分的信息描述了如何从 ComputePartition SMT 迁移到新的 PartitionRouting SMT。

假设配置为所有主题设置相同分区数量，请将以下 ComputePartition'configuration 替换为"PartitionRouting SMT"。以下示例提供了两个配置的比较。

示例： Legacy ComputePartition 配置

```
...
topic.creation.default.partitions=2
topic.creation.default.replication.factor=1
...
topic.prefix=fulfillment
transforms=ComputePartition
transforms.ComputePartition.type=io.debezium.transforms.partitions.ComputePartition
transforms.ComputePartition.partition.data-
collections.field.mappings=inventory.products:name,inventory.orders:purchaser
transforms.ComputePartition.partition.data-
collections.partition.num.mappings=inventory.products:2,inventory.orders:2
...
```

将前面的 ComputePartition 替换为以下 分区 配置。示例：替换早期 ComputePartition 配置的分区 Routing 配置

```
...
topic.creation.default.partitions=2
topic.creation.default.replication.factor=1
...

topic.prefix=fulfillment
transforms=PartitionRouting
transforms.PartitionRouting.type=io.debezium.transforms.partitions.PartitionRouting
transforms.PartitionRouting.partition.payload.fields=change.name,change.purchaser
transforms.PartitionRouting.partition.topic.num=2
transforms.PartitionRouting.predicate=allTopic
predicates=allTopic
```

```
predicates.allTopic.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
predicates.allTopic.pattern=fulfillment.*
...
```

如果 SMT 将事件发送到没有共享相同分区的主题，您必须为每个主题指定唯一的 partition.num.mappings 值。例如，在以下示例中，旧 产品 集合的主题配置了 3 个分区，而 orders 数据收集的主题被配置为 2 个分区：

示例：为不同主题设置唯一分区值的 Legacy ComputePartition 配置

```
...
topic.prefix=fulfillment
transforms=ComputePartition
transforms.ComputePartition.type=io.debezium.transforms.partitions.ComputePartition
transforms.ComputePartition.partition.data-
collections.field.mappings=inventory.products:name,inventory.orders:purchaser
transforms.ComputePartition.partition.data-
collections.partition.num.mappings=inventory.products:3,inventory.orders:2
...
```

将前面的 ComputePartition 配置替换为以下 PartitionRouting 配置：.PartitionRouting 配置，为不同的主题设置唯一的 partition.topic.num 值

```
...
topic.prefix=fulfillment

transforms=ProductsPartitionRouting,OrdersPartitionRouting
transforms.ProductsPartitionRouting.type=io.debezium.transforms.partitions.PartitionRouting
transforms.ProductsPartitionRouting.partition.payload.fields=change.name
transforms.ProductsPartitionRouting.partition.topic.num=3
transforms.ProductsPartitionRouting.predicate=products

transforms.OrdersPartitionRouting.type=io.debezium.transforms.partitions.PartitionRouting
transforms.OrdersPartitionRouting.partition.payload.fields=change.purchaser
transforms.OrdersPartitionRouting.partition.topic.num=2
transforms.OrdersPartitionRouting.predicate=products

predicates=products,orders
predicates.products.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
predicates.products.pattern=fulfillment.inventory.products
predicates.orders.type=org.apache.kafka.connect.transforms.predicates.TopicNameMatches
predicates.orders.pattern=fulfillment.inventory.orders
...
```

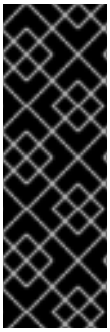
13.11.4. 用于配置分区路由转换的选项

下表列出了您可以为分区路由 SMT 设置的配置选项。

表 13.13. 分区路由 SMT（分区）配置选项

属性	默认	描述
partition.payload.fields		指定 SMT 用来计算目标分区的事件有效负载中的字段。如果您希望 SMT 将原始有效负载中的字段添加到输出数据结构中的特定级别，请使用点表示法。要访问与数据收集相关的字段，您可以使用：**在、之前 或更改**。'change' 字段是一个特殊字段，它会根据操作类型在 'after' 或 'before' 元素中自动填充内容。如果记录中没有指定字段，则 SMT 会跳过它。例如, **after.name,source.table,change.name**
partition.topic.num		此 SMT 行为的主题分区数量。使用 **TopicNameMatches** predicate 按主题过滤记录。
partition.hash.function	**java**	在计算要确定目标分区数目的字段的计算哈希时，要使用的哈希函数。可能的值有： **java** - 标准 Java **Object::hashCode** function **murmur** - latest version ofMurmurHash 功能, MurmurHash3 this configuration is optional.如果没有指定或无效值，则使用默认值。

第 14 章 开发 DEBEZIUM 自定义数据类型转换器



重要

使用自定义开发的转换器只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅 https://access.redhat.com/support/offerings/techpreview。

Debezium 更改事件记录中的每个字段代表源表或数据收集中的字段或列。当连接器向 Kafka 发送更改事件记录时，它会将源中的每个字段的数据类型转换为 Kafka Connect 模式类型。列值同样转换为与目的地字段的 schema 类型匹配。对于每个连接器，默认映射指定连接器如何转换每个数据类型。这些默认映射在每种连接器的数据类型文档中描述。

虽然默认映射通常就足够了，但对于某些应用程序，您可能需要应用备用映射。例如，如果默认映射导出自 UNIX epoch 以来的毫秒格式，您可能需要自定义映射，但下游应用只能以格式化的字符串形式消耗列值。您可以通过开发和部署自定义转换器来自定义数据类型映射。您可以将自定义转换器配置为对某个类型的所有列执行操作，或者您可以缩小其范围，使其仅适用于特定的表列。转换程序功能会截获与指定条件匹配的列的数据类型转换请求，然后执行指定的转换。转换程序会忽略与指定条件不匹配的列。

自定义转换器是实现 Debezium 服务供应商接口(SPI)的 Java 类。您可以通过在连接器配置中设置 converters 属性来启用和配置自定义转换器。converters 属性指定可用于连接器的转换器，可以包含进一步修改转换行为的子属性。

启动连接器后，连接器配置中启用的转换器将被实例化并添加到 registry 中。registry 将每个转换器与要处理的列或字段相关联。每当 Debezium 处理一个新的更改事件时，它会调用配置的转换器来转换其注册的列或字段。

14.1. 创建 DEBEZIUM 自定义数据类型转换器

以下示例显示了实现接口 io.debezium.spi.converter.CustomConverter 的 Java 类的转换器实现：

```
public interface CustomConverter<S, F extends ConvertedField> {

  @FunctionalInterface
  interface Converter { (1)
    Object convert(Object input);
  }

  public interface ConverterRegistration<S> { (2)
```

```
        void register(S fieldSchema, Converter converter); 3
    }

    void configure(Properties props);

    void converterFor(F field, ConverterRegistration<S> registration); 4
}
```

1

将数据从一个类型转换为另一个类型的功能。

2

注册转换器的回调。

3

为当前字段注册给定模式和转换器。对于同一字段，不应多次调用。

4

注册自定义值和模式转换器，以用于特定字段。

自定义转换器方法

CustomConverter 接口的实现必须包括以下方法：

configure()

将连接器配置中指定的属性传递给转换器实例。configure 方法在连接器初始化时运行。您可以使用带有多个连接器的转换器，并根据连接器的属性设置修改其行为。
configure 方法接受以下参数：

Proprops

包含传递到转换器实例的属性。每个属性指定转换特定类型的列值的格式。

converterFor()

注册转换器以处理数据源中的特定列或字段。Debezium 调用 converterFor （） 方法，以提示输入转换器 来调用转换的注册。converterFor 方法为每个列运行一次。
这个方法接受以下参数：

field

一个对象，用于传递有关所处理字段或列的元数据。列元数据可以包括列或字段的名称、表或集合的名称、数据类型、大小等。

注册

一个类型为 io.debezium.spi.converter.CustomConverterRegistration 的对象，它提供目标模式定义以及转换列数据的代码。当源列与应处理的类型匹配时，转换器调用 注册 参数。调用 register 方法，以定义 schema 中各个列的转换器。模式使用 Kafka Connect SchemaBuilder API 表示。

14.1.1. Debezium 自定义转换器示例

以下示例实现了一个简单的转换器，它执行以下操作：

- 运行 configure 方法，它根据连接器配置中指定的 schema.name 属性的值配置转换器。转换程序配置特定于每个实例。
- 运行 converterFor 方法，它会在 data type 设置为 isbn 的源列中注册转换器来处理值。
  - 根据为 schema.name 属性指定的值标识目标 STRING 模式。
  - 将源列中的 ISBN 数据转换为 String 值。

例 14.1. 简单的自定义转换器

```
public static class IsbnConverter implements CustomConverter<SchemaBuilder,
RelationalColumn> {

  private SchemaBuilder isbnSchema;

  @Override
  public void configure(Properties props) {
    isbnSchema = SchemaBuilder.string().name(props.getProperty("schema.name"));
  }

  @Override
  public void converterFor(RelationalColumn column,
      ConverterRegistration<SchemaBuilder> registration) {

    if ("isbn".equals(column.typeName())) {
      registration.register(isbnSchema, x -> x.toString());
```

```
        }
    }
}
```

14.1.2. Debezium 和 Kafka Connect API 模块依赖项

自定义转换器 Java 项目对 Debezium API 和 Kafka Connect API 库模块有编译依赖项。这些编译依赖项必须包含在项目的 pom.xml 中，如下例所示：

```
<dependency>
    <groupId>io.debezium</groupId>
    <artifactId>debezium-api</artifactId>
    <version>${version.debezium}</version> (1)
</dependency>
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>connect-api</artifactId>
    <version>${version.kafka}</version> (2)
</dependency>
```

(1) ${version.debezium} 代表 Debezium 连接器的版本。

(2) ${version.kafka} 代表您环境中的 Apache Kafka 版本。

14.2. 使用带有 DEBEZIUM 连接器的自定义转换器

自定义转换器对源表中的特定列或列类型进行操作，以指定如何将源中的数据类型转换为 Kafka Connect 模式类型。要将自定义转换器与连接器搭配使用，您可以将转换器 JAR 文件与连接器文件一起部署，然后将连接器配置为使用转换器。

14.2.1. 部署自定义转换器

先决条件

- 您有一个自定义转换器 Java 程序。

流程

-

要将自定义转换器与 Debezium 连接器搭配使用，请将 Java 项目导出到 JAR 文件，并将文件复制到包含您要使用它的每个 Debezium 连接器的 JAR 文件的目录中。

例如，在典型的部署中，Debezium 连接器文件存储在 Kafka Connect 目录的子目录中（/kafka/connect），每个连接器 JAR 在其自己的子目录中(/kafka/connect/debezium-connector-db2、 /kafka/connect/debezium-connector-mysql 等)。要将转换器与连接器搭配使用，请将转换器 JAR 文件添加到连接器的子目录中。

注意

要将转换器与多个连接器搭配使用，您必须将转换器 JAR 文件的副本放在每个连接器子目录中。

14.2.2. 将连接器配置为使用自定义转换器

要启用连接器使用自定义转换器，您可以在连接器配置中添加指定转换器名称和类的属性。如果转换器需要更多信息才能自定义特定数据类型的格式，您也可以定义其他缓解选项以提供该信息。

流程

- 通过在连接器配置中添加以下强制属性，为连接器实例启用转换器：

```
converters: <converterSymbolicName> 1
<converterSymbolicName>.type: <fullyQualifiedConverterClassName> 2
```

1

converters 属性是必需的，枚举一个以逗号分隔的、用来与连接器一起使用的转换器实例的符号链接名称列表。在您为转换器指定的其他属性名称中，为此属性列出的值充当前缀。

2

& lt;converterSymbolicName > .type 属性是必需的，并指定实现转换器的类名称。例如，对于以前的 自定义转换器 示例，您可以在连接器配置中添加以下属性：

```
converters: isbn
isbn.type: io.debezium.test.IsbnConverter
```

- 要将其他属性与自定义转换器关联，请为属性名称加上转换器的符号名称作为前缀，后跟一个点(.)。符号名称是您指定为 converters 属性的值的标签。例如，要为前面的 isbn converter 添加属性，以指定要传递给转换器代码中配置方法的 schema.name，请添加以下属性：

isbn.schema.name: io.debezium.postgresql.type.Isbn

更新于 2024-01-09