Red Hat JBoss Enterprise Application Platform 7.4

配置消息传递

面向希望为红帽 JBoss 企业应用平台开发和部署消息传递应用程序的开发人员和管理员提供的说明和信息。

Last Updated: 2024-02-10

Red Hat JBoss Enterprise Application Platform 7.4 配置消息传递

面向希望为红帽 JBoss 企业应用平台开发和部署消息传递应用程序的开发人员和管理员提供的说明和信息。

法律通告

摘要

本文档为希望使用红帽 JBoss 企业应用平台开发和部署消息传递应用程序的开发人员和管理员提供了信息。

目录

提供有关 JBOSS EAP 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 请包含 **文档 URL、章节编号 并描述问题**。
3. 在 **Summary** 中输入问题的简短描述。
4. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
5. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 CTO Chris Wright 信息。

部分 I. 关于消息传递和 JBOSS EAP 7

JBoss EAP 6 中的消息传递代理名为 HornetQ，这是一个 JBoss 社区项目。HornetQ 代码库已贡献到 Apache ActiveMQ 项目，HornetQ 社区加入该项目，以增强贡献的代码库并创建下一代消息传递代理。其结果是 JBoss EAP 7 的消息传递代理 Apache ActiveMQ Artemis，提供消息整合和 JBoss EAP 6 的向后兼容性。虽然 ActiveMQ Artemis 在 JBoss EAP 6 中保持了与 HornetQ 代理的协议兼容性，但它还包含一些明智的新功能。本指南将探讨 JBoss EAP 7.4 中提供的 ActiveMQ Artemis 代理的许多功能，并提供有用的示例。

第 1 章 消息传递概念

1.1. 消息传递系统

消息传递系统允许您松散耦合异构系统，同时提高可靠性。与基于远程过程调用(RPC)模式的系统不同，消息传递系统主要使用异步消息传递模式，请求和响应之间没有紧密关系。大多数消息传递系统足够灵活，足以在需要时支持请求响应模式，但这不是消息传递系统的主要功能。

消息传递系统将消息的发件人与其消费者分离。事实上，消息的发送者和使用者完全独立，彼此不了解，这使得您可以创建灵活、松散耦合的系统。大型企业通常使用消息传递系统来实施松散耦合异构系统的消息总线。消息总线可形成企业服务总线(ESB)的核心。利用消息总线分离分散分散的系统，让系统能够更加轻松地发展并适应。它还能够更加灵活地添加新系统或停用旧系统，因为它们之间没有相互依赖的混乱。

消息传递系统还可以纳入诸如以下概念：确保可靠消息传递、将发送或消费多条消息聚合为一个工作单元，以及允许消息避免服务器故障或重新启动的持久性。

1.2. 消息传递样式

大多数消息传递系统支持两种消息传递样式： *点对点* 模式和 *发布订阅模式*。

- 点到点模式
  点对点模式涉及向侦听队列的单个消费者发送消息。进入队列后，消息通常变为永久消息，以保证传输。消息经过队列后，消息传递系统会将消息传送给消费者。消费者确认消息一旦被处理即可发送。对于同一消息，可以有多个消费者侦听同一队列，但只有一位使用者会收到每条消息。

- publish-Subscribe Pattern
  发布-订阅模式允许发送者利用单一目的地向多个消费者发送消息。此目的地通常称为 *主题*。每个主题都有多个消费者或订阅者，与点对点消息不同，每个订阅者会收到发布到该主题的任何消息。

  另一个有趣的区别在于，订阅者可以是持久化的。持久订阅在连接时传递服务器的唯一标识符，允许服务器识别并发送自订阅者上次连接以来发布到该主题的任何消息。此类消息通常在重新启动后由服务器保留。

1.3. JAKARTA MESSAGING

Jakarta Messaging 2.0 在 Jakarta Messaging 中定义。Jakarta Messaging 是一种 Java API，提供点对点和发布订阅者消息传递样式。雅加达消息传递还包含事务的使用。Jakarta Messaging 没有定义标准线路格式，因此 Jakarta 消息传递提供商的供应商可能都使用标准 API，他们可能会使用不同的内部线路协议在客户端和服务器之间进行通信。

1.4. JAKARTA 消息传递目标

Jakarta 消息传递目的地以及 Jakarta 消息传递连接工厂是管理对象。目的地供 Jakarta 消息传递客户端用于生成和使用消息。目标允许客户端在生成消息时使用消息时指定目标以及消息源。使用发布订阅模式时，目的地称为主题。使用点对点模式时，目的地称为队列。

应用程序可能使用许多不同的 Jakarta 消息传递目的地，这些目的地在服务器端配置，通常通过 JNDI 访问。

第 2 章 集成 ACTIVEMQ ARTEMIS MESSAGING BROKER

2.1. ACTIVEMQ ARTEMIS

Apache ActiveMQ Artemis 是一个用于异步消息传递系统的开源项目。它是高性能、嵌入式、集群化并支持多种协议。JBoss EAP 7 将 Apache ActiveMQ Artemis 用作其 Jakarta 消息传递代理，并使用 **messaging-activemq** 子系统进行配置。这完全取代了 HornetQ 代理，但仍保持了与 JBoss EAP 6 的协议兼容性。

核心 ActiveMQ Artemis 是 Jakarta Messaging-agnostic，提供非 Jakarta 消息 API，称为 *核心 API*。ActiveMQ Artemis 还提供 Jakarta 消息传递客户端 API，它使用学术层在核心 API 基础上实施 Jakarta 消息传递语义。基本上，Jakarta 消息传递交互使用 Jakarta 消息传递客户端 API 在客户端上转换为核心 API 操作。从那里，所有操作都使用核心客户端 API 和 Apache ActiveMQ Artemis wire 格式发送。服务器本身仅使用 core API。如需有关核心 API 及其概念的更多详细信息，请参阅 ActiveMQ Artemis 文档。

2.2. APACHE ACTIVEMQ ARTEMIS CORE API 和 JAKARTA MESSAGING DESTINATIONS

让我们简要讨论 Jakarta 消息目的地如何映射到 Apache ActiveMQ Artemis 地址。

Apache ActiveMQ Artemis 内核是 Jakarta Messaging-agnostic.它没有任何雅加达消息传递主题的概念。Jakarta Messaging 主题作为地址（主题名称）在核心中实施，队列绑定为零个或多个队列。绑定到该地址的每个队列都代表一个主题订阅。同样，雅加达消息传递队列实施为一个地址（雅加达消息传递队列名称），其中有一个队列绑定到它，表示 Jakarta 消息传递队列。

按照惯例，所有 Jakarta 消息队列映射到核心队列，其中核心队列名称具有 string **jms.queue.** 前缀为它。例如，名称为 Order **.europe** 的 Jakarta Messaging 队列将映射到带有 name **jms.queue.orders.europe** 的核心队列。绑定核心队列的地址也由核心队列名称提供。

对于 Jakarta Messaging 主题，通过将 string **jms.topic.** 添加到 Jakarta Messaging 主题的名称前，提供代表订阅的队列的绑定地址。例如，名称为 **news.europe** 的 Jakarta Messaging 主题将映射到核心 address **jms.topic.news.europe**。

换句话说，如果您使用名称 **Order .europe 向 Jakarta 消息传递队列发送 Jakarta 消息**，它将被路由到绑定到 address **jms.queue.orders.europe** 的任何核心队列。如果您发送 **Jakarta 消息** 到名称为 **news.europe** 的 Jakarta Messaging 主题，它会在服务器上路由到绑定到 address **jms.topic.news.europe** 的任何核心队列。

如果要使用名称 Order **.europe** 为 Jakarta Messaging 队列配置设置，您需要配置对应的内核队列 **jms.queue.orders.europe** ：

```
<!-- expired messages in JMS Queue "orders.europe" will be sent to the JMS Queue "expiry.europe" -->
<address-setting match="jms.queue.orders.europe">
   <expiry-address>jms.queue.expiry.europe</expiry-address>
   ...
</address-setting>
```

部分 II. 配置单节点消息传递系统

第 II 部分从使用 **helloworld-mdb** 快速入门的 JBoss EAP 7 消息传递入门指南开始。以下为任何安装可用的配置选项包括安全性和持久性等主题。有关多个 JBoss EAP 7 安装的相关配置，包括群集、高可用性和连接其他服务器等主题，请参阅配置多节点消息传递系统 第三部分。

第 3 章 开始使用

3.1. 使用 HELLOWORLD-MDB QUICKSTART

helloworld-mdb quickstart 使用一个简单的消息驱动 Bean 来演示基本的 Jakarta EE 消息传递功能。在回顾 基本配置 时，快速入门和运行是向自己介绍 JBoss EAP 消息传递服务器所含功能的一种好方法。

构建和部署 helloworld-mdb Quickstart

有关构建和部署 **helloworld-mdb** 快速启动的说明，请参阅由快速入门提供的 **README.md** 文件中的说明。您将需要启动 JBoss EAP 服务器，指定包含 **messaging-activemq** 子系统 **的完整** 配置。有关使用不同配置文件启动 JBoss EAP 的详细信息，请参阅 **README.md** 文件或 JBoss EAP *配置指南*。

3.2. MESSAGING 子系统配置概述

在使用 full 或 **full - ha 配置**启动 **JBoss EAP 服务器时，包含 messaging- activemq** 子系统的默认配置。**full-ha** 选项包括 集群和 高可用性 等功能的高级配置。

虽然不需要，但 建议您使用 **helloworld-mdb** 快速入门 作为有效的示例，以便与配置概述一起运行。

如需有关 **messaging-activemq** 子系统中所有可用设置的信息，请参见 ***EAP_HOME*/docs/schema/** 目录中的架构定义，或者从管理 CLI 对子系统运行 **read-resource-description** 操作，如下所示：

```
/subsystem=messaging-activemq:read-resource-description(recursive=true)
```

服务器配置文件中的以下扩展名指示 JBoss EAP 将 **messaging-activemq** 子系统作为其运行时的一部分：

```
<extensions>
  ...
  <extension module="org.wildfly.extension.messaging-activemq"/>
  ...
</extensions>
```

messaging-activemq 子系统的配置包含在 **<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">** 元素中。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    <server name="default">
        <cluster password="${jboss.messaging.cluster.password:CHANGE ME!!}"/>
        <security-setting name="#">
            <role name="guest" send="true" consume="true" create-non-durable-queue="true" delete-
non-durable-queue="true"/>
        </security-setting>
        <address-setting name="#" dead-letter-address="jms.queue.DLQ" expiry-
address="jms.queue.ExpiryQueue" max-size-bytes="10485760" page-size-bytes="2097152"
message-counter-history-day-limit="10" redistribution-delay="1000"/>
        <http-connector name="http-connector" socket-binding="http" endpoint="http-acceptor"/>
        <http-connector name="http-connector-throughput" socket-binding="http" endpoint="http-
acceptor-throughput">
            <param name="batch-delay" value="50"/>
        </http-connector>
        <in-vm-connector name="in-vm" server-id="0"/>
        <http-acceptor name="http-acceptor" http-listener="default"/>
```

```
        <http-acceptor name="http-acceptor-throughput" http-listener="default">
            <param name="batch-delay" value="50"/>
            <param name="direct-deliver" value="false"/>
        </http-acceptor>
        <in-vm-acceptor name="in-vm" server-id="0"/>
        <broadcast-group name="bg-group1" connectors="http-connector" jgroups-cluster="activemq-
cluster"/>
        <discovery-group name="dg-group1" jgroups-cluster="activemq-cluster"/>
        <cluster-connection name="my-cluster" address="jms" connector-name="http-connector"
discovery-group="dg-group1"/>
        <jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>
        <jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>
        <connection-factory name="InVmConnectionFactory" connectors="in-vm"
entries="java:/ConnectionFactory"/>
        <connection-factory name="RemoteConnectionFactory" ha="true" block-on-acknowledge="true"
reconnect-attempts="-1" connectors="http-connector"
entries="java:jboss/exported/jms/RemoteConnectionFactory"/>
        <pooled-connection-factory name="activemq-ra" transaction="xa" connectors="in-vm"
entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory"/>
    </server>
</subsystem>
```

连接事实
消息传递客户端使用 Jakarta Messaging **ConnectionFactory** 对象来连接服务器。默认 JBoss EAP 配置定义多种连接工厂：请注意，有 **<connection-factory>** 用于 in-vm、http 和池化连接。

```
  <connection-factory name="InVmConnectionFactory" connectors="in-vm"
entries="java:/ConnectionFactory"/>
  <connection-factory name="RemoteConnectionFactory" ha="true" block-on-acknowledge="true"
reconnect-attempts="-1" connectors="http-connector"
entries="java:jboss/exported/jms/RemoteConnectionFactory"/>
  <pooled-connection-factory name="activemq-ra" transaction="xa" connectors="in-vm"
entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory"/>
```

如需了解更多详细信息，请参阅 配置连接事实部分。

连接器和 Acceptors
每个 Jakarta 消息传递连接工厂使用连接器启用来自客户端或消费者与消息传递服务器的 Jakarta 消息通信。连接器对象定义用于连接消息传递服务器的传输和参数。其对应对象是接收器对象，用于标识消息传递服务器接受的连接类型。

默认的 JBoss EAP 配置定义了多个连接器和接收器。

示例：默认连接器

```
<http-connector name="http-connector" socket-binding="http" endpoint="http-acceptor"/>
<http-connector name="http-connector-throughput" socket-binding="http" endpoint="http-acceptor-
throughput">
  <param name="batch-delay" value="50"/>
</http-connector>
<in-vm-connector name="in-vm" server-id="0"/>
```

示例：默认接受器

```
<http-acceptor name="http-acceptor" http-listener="default"/>
```

```
<http-acceptor name="http-acceptor-throughput" http-listener="default">
  <param name="batch-delay" value="50"/>
  <param name="direct-deliver" value="false"/>
</http-acceptor>
```

详情请查看 Acceptors 和 Connectors 部分。

套接字绑定组
默认连接器 **的 socket-binding** 属性引用名为 **http** 的套接字绑定。使用 http 连接器，因为 JBoss EAP 可以在标准 Web 端口上多路复用请求。

您可以在配置文件的其他位置找到此 **socket-binding** 作为 **<socket-binding-group>** 部分的一部分。请注意 http 和 https 套接字绑定的配置如何出现在 **<socket-binding-groups>** 元素中：

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
   ...
  <socket-binding name="http" port="${jboss.http.port:8080}"/>
  <socket-binding name="https" port="${jboss.https.port:8443}"/>
   ...
</socket-binding-group>
```

如需有关套接字绑定的信息，请参阅 JBoss EAP *配置指南中的配置* 套接字绑定。

消息传递安全性
在首次安装 JBoss EAP 时，**messaging -activemq 子系统包含一个 security-setting** 元素：

```
<security-setting name="#">
  <role name="guest" delete-non-durable-queue="true" create-non-durable-queue="true"
consume="true" send="true"/>
</security-setting>
```

这声明，任何具有 **guest** 角色的用户都可以访问服务器上的任何地址，如通配符 **# 所述**。有关通配符语法的更多信息，请参阅配置地址设置。

有关保护目的地和远程连接的更多信息，请参阅配置消息传递安全性。

消息传递目标
完整和全 ha 配置包括两个有帮助的队列，JBoss EAP 可以使用这些队列存放已过期或无法路由到正确目的地的消息。

```
<jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>
<jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>
```

您可以使用以下方法之一在 JBoss EAP 中添加自己的消息传递目的地：

- 使用管理 CLI
  使用以下管理 CLI 命令添加队列：

  ```
  jms-queue add --queue-address=testQueue  --
  entries=queue/test,java:jboss/exported/jms/queue/test
  ```

  使用以下管理 CLI 命令添加主题：

```
jms-topic add --topic-address=testTopic  --
entries=topic/test,java:jboss/exported/jms/topic/test
```

- 使用管理控制台
  可以通过导航到 **Configuration → Subsystems → Messaging(ActiveMQ) → Server**，从管理控制台中配置消息传递目的地，选择 **Destinations** 并点 **View**。选择 **JMS Queue** 选项卡来配置队列，再选择 **JMS 主题** 来配置主题。

- 使用 Jakarta EE 部署描述符或注释来定义您的目的地。
  在 Jakarta EE 8 中，部署描述符可以包含队列和主题的配置。以下是来自 Jakarta EE 描述符文件的代码片段，该文件定义了 Jakarta 消息传递队列。

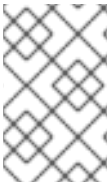
```
...
<jms-destination>
  <name>java:global/jms/MyQueue</name>
  <interfaceName>javax.jms.Queue</interfaceName>
  <destinationName>myQueue</destinationName>
</jms-destination>
...
```

例如，helloworld **-mdb** quickstart 中的消息驱动型 Bean 包含定义运行应用所需的队列和主题的注释。以这种方式创建的目的地将显示在运行时队列列表中。使用管理 CLI 显示运行时队列的列表。部署 Quickstart 后，它创建的运行时队列将如下所示：

```
/subsystem=messaging-activemq/server=default/runtime-queue=*:read-resource
{
    "outcome" => "success",
    "result" => [
        ...
        {
            "address" => [
                ("subsystem" => "messaging-activemq"),
                ("server" => "default"),
                ("runtime-queue" => "jms.queue.HelloWorldMDBQueue")
            ],
            "outcome" => "success",
            "result" => {"durable" => undefined}
        },
        ...
        {
            "address" => [
                ("subsystem" => "messaging-activemq"),
                ("server" => "default"),
                ("runtime-queue" => "jms.topic.HelloWorldMDBTopic")
            ],
            "outcome" => "success",
            "result" => {"durable" => undefined}
        },
        ...
    ]
}
```

如需了解更多详细信息，请参阅配置消息传递目标。

第 4 章 配置消息传递目标



注意

请记住，配置消息传递目的地要求 JBoss EAP 启用了消息传递。使用 standalone-full.xml 或 **standalone- full-ha.xml** 配置文件运行时，此功能会被默认启用。**domain.xml** 配置文件也启用了消息传递。

4.1. 添加队列

要添加 Jakarta Messaging 队列，请使用管理 CLI 中的 **jms-queue** 命令：

```
jms-queue add --queue-address=myQueue --entries=[queue/myQueue jms/queue/myQueue
java:jboss/exported/jms/queue/myQueue]
```

注意 **entries** 属性如何是一个包含由单个空格分隔的多个 JNDI 名称的列表。另请注意，使用方括号 **[]** 来括起 JNDI 名称列表。**queue-address** 提供路由配置，**条目** 提供 JNDI 名称列表，客户端可用于查找队列。

读取队列的属性

您可以在管理 CLI 中使用 **jms-queue** 命令读取队列的配置。

```
jms-queue read-resource --queue-address=myQueue
```

或者，您可以使用管理 CLI 访问 **messaging-activemq** 子系统来读取队列配置：

```
/subsystem=messaging-activemq/server=default/jms-queue=myQueue:read-resource()
{
    "outcome" => "success",
    "result" => {
        "durable" => true,
        "entries" => ["queue/myQueue jms/queue/myQueue java:jboss/exported/jms/queue/myQueue"],
        "legacy-entries" => undefined,
        "selector" => undefined
    }
}
```

a jms-queue的属性

输入以下命令时，管理 CLI 显示 **jms-queue** 配置元素的所有属性：

```
/subsystem=messaging-activemq/server=default/jms-queue=*:read-resource-description()
```

下表提供了 a **jms-queue** 的所有属性：

属性	描述
consumer-count	此队列中消息的使用者数量。在运行时可用.
dead-letter-address	将死信发送到的地址。如需更多信息，请参阅配置 Dead Letter 地址。

属性	描述
delivery-count	此队列当前传送到其使用者的消息数量。在运行时可用.
Durable	队列是否持久。有关持久订阅的更多信息，请参阅 消息样式。
条目	队列将绑定到的 JNDI 名称的列表。必需。
expiry-address	将接收过期邮件的地址。详情请参阅 配置消息过期。
legacy-entries	队列将绑定到的 JNDI 名称。
message-count	此队列中当前消息的数量。在运行时可用.
添加消息	自该队列创建以来添加到此队列的消息数。在运行时可用.
paused	队列是否暂停。在运行时可用.
queue-address	队列地址定义用于路由消息的地址。有关 地址设置的详情，请参阅配置 地址设置。必需。
scheduled-count	此队列中调度的消息数量。在运行时可用.
selector	队列选择器。有关选择器的更多信息，请参阅 Filter Expressions 和 Message Selectors。
临时	队列是否为临时队列。如需更多信息，请参阅 临时队列和运行时队列。

4.2. 添加主题

添加或读取主题与添加队列非常类似：

```
jms-topic add --topic-address=myTopic --entries=[topic/myTopic jms/topic/myTopic
java:jboss/exported/jms/topic/myTopic]
```

阅读主题的属性

读取主题属性的语法也类似于用于队列的语法：

```
jms-topic read-resource --topic-address=myTopic

entries
  topic/myTopic jms/topic/myTopic java:jboss/exported/jms/topic/myTopic
legacy-entries=n/a
```

```
/subsystem=messaging-activemq/server=default/jms-topic=myTopic:read-resource
{
    "outcome" => "success",
    "result" => {
        "entries" => ["topic/myTopic jms/topic/myTopic java:jboss/exported/jms/topic/myTopic"],
```

```
        "legacy-entries" => undefined
    }
}
```

a jms-topic的属性

输入以下命令时，管理 CLI 显示 **jms-topic** 配置元素的所有属性：

```
/subsystem=messaging-activemq/server=default/jms-topic=*:read-resource-description()
```

下表列出了 a **jms-topic** 的属性：

属性	描述
delivery-count	此队列当前传送到其使用者的消息数量。在运行时可用.
durable-message-count	本主题的所有持久订阅者的消息数量。在运行时可用.
durable-subscription-count	本主题的持久订户数量。在运行时可用.
条目	该主题的 JNDI 名称将绑定到.必需。
legacy-entries	该主题将绑定到的传统 JNDI 名称。
message-count	此队列中当前消息的数量。在运行时可用.
添加消息	自该队列创建以来添加到此队列的消息数。在运行时可用.
non-durable-message-count	本主题的所有不可持久性订阅者的消息数量。在运行时可用.
non-durable-subscription-count	本主题的非可用订阅数量。在运行时可用.
subscription-count	本主题的（可持久性和不可持久性）订阅者的数量。在运行时可用.
临时	主题是否为临时主题。
topic-address	主题指向的地址。必需。

4.3. JNDI 条目和客户端

队列或主题必须绑定到 **java:jboss/exported** 命名空间，以便远程客户端能够进行查找。在进行查找时，客户端必须使用 **java:jboss/exported/** 后的文本。例如，名为 **testQueue** 的队列为其 **条目** 的 list **jms/queue/test java:jboss/exported/jms/queue/test**.希望将消息发送到 **testQueue** 的远程客户端将使用 string **jms/queue/test** 查找队列。另一方面，本地客户端可以通过 **java:jboss/exported/jms/queue/test**、**java:jms/queue/test** 或更多 simple **jms/queue/test 进行** 查找。

管理 CLI 帮助

您可以使用 **--help --command s 标志找到有关 jms-queue 和jms -topic** 命令的更多信息：

```
jms-queue --help --commands
```

```
jms-topic --help --commands
```

4.4. 使用管理 API 暂停 JAKARTA MESSAGING 主题的方法

您可以通过暂停所有消费者来暂停一个主题。如果在主题暂停时该主题注册了任何新订阅，则也暂停。

该主题的订阅者不会从暂停的主题收到新消息。但是，暂停的主题仍然可以接收发送给它的消息。恢复该主题时，排队的消息将传送到订阅者。

您可以使用 **persist** 参数来存储主题的状态，这样即使您重启代理，主题也会保持暂停。

其他资源

- 有关暂停某一主题的详情，请参考 暂停一个主题。
- 有关恢复某个主题的详情，请参考 假定一个主题。

4.5. 暂停一个主题

您可以暂停一个主题，以便所有主题订阅者停止从暂停的主题接收新消息。

流程

- 如以下示例所示，暂停该主题：

```
/subsystem=messaging-activemq/server=default/jms-topic=topic:pause()
{
    "outcome" => "success",
    "result" => undefined
}
```

暂停的主题如下例所示：

```
/subsystem=messaging-activemq/server=default/jms-topic=topic:read-
attribute(name=paused)
{
    "outcome" => "success",
    "result" => true
}
```

其他资源

- 有关主题的暂停方法的信息，请参阅使用 管理 API 为 Jakarta Messaging 主题暂停方法 。
- 有关恢复某个主题的详情，请参考 假定一个主题。

4.6. 恢复一个主题

您可以恢复暂停的主题。当您恢复该主题时，在暂停时收到的消息将传送给订阅者。

流程

- 恢复该主题，如下例所示：

```
/subsystem=messaging-activemq/server=default/jms-topic=topic:resume()
{
    "outcome" => "success",
    "result" => undefined
}
```

恢复的主题如下例所示：

```
/subsystem=messaging-activemq/server=default/jms-topic=topic:read-
attribute(name=paused)
{
    "outcome" => "success",
    "result" => false
}
```

其他资源

- 有关主题的暂停方法的信息，请参阅使用 管理 API 为 Jakarta Messaging 主题暂停方法 。
- 有关暂停某一主题的详情，请参考 暂停一个主题。

第 5 章 配置日志记录

您可以通过在 JBoss EAP **logging** 子系统中添加 **org.apache. activemq** 的日志类别并设置所需的日志级别，为 **messaging-activemq** 配置日志记录。您还可以为类别配置日志处理程序，以配置日志消息的记录方式。

要在有关 XA 事务的日志中查看更多信息，请将 **com.arjuna** 类别的日志级别更改为更为详细的设置，如 **TRACE 或 DEBUG**。

有关日志记录的更多信息，包括类别配置和其他选项的配置，请参见《JBoss EAP *配置指南*》中关于 登录 的章节。

表 5.1. 日志记录类别

如果要日志用于...	使用此类别...
XA 事务	com.arjuna
所有消息传递活动	org.apache.activemq
只调用消息传递日志	org.apache.activemq.artemis.journal
仅 Jakarta 消息传递调用	org.apache.activemq.artemis.jms
仅消息传递 utils 调用	org.apache.activemq.artemis.utils
仅限消息传递核心服务器	org.apache.activemq.artemis.core.server

配置客户端以进行日志

按照以下步骤配置消息传递客户端：

1. 将依赖性下载到 JBoss Jakarta 消息传递客户端和日志管理器。
   如果使用 Maven，请在 **pom.xml** 文件中添加以下依赖项：

```
<dependencies>
  ...
  <dependency>
    <groupId>org.jboss.logmanager</groupId>
    <artifactId>jboss-logmanager</artifactId>
    <version>1.5.3.Final</version>
  </dependency>
  <dependency>
     <groupId>org.jboss.eap</groupId>
     <artifactId>wildfly-jms-client-bom</artifactId>
     <type>pom</type>
  </dependency>
  ...
</dependencies>
```

   如需更多信息，请参见《JBoss EAP *开发指南*》中有关 *将 Maven 与 JBoss EAP 搭配使用* 的章节。

2. 创建日志记录器的属性文件。将它命名为 **logging.properties** 并将其保存到已知位置。以下是属性文件示例：有关在客户端上配置 日志记录 选项的更多信息，请参阅有关在 JBoss EAP *开发指南中* 记录的章节。

```
# Root logger option
loggers=org.jboss.logging,org.apache.activemq.artemis.core.server,org.apache.activemq.artemis.utils,org.apache.activemq.artemis.journal,org.apache.activemq.artemis.jms,org.apache.activemq.artemis.ra

# Root logger level
logger.level=INFO
# Apache ActiveMQ Artemis logger levels
logger.org.apache.activemq.artemis.jms.level=INFO
logger.org.apache.activemq.artemis.journal.level=INFO
logger.org.apache.activemq.artemis.utils.level=INFO
logger.org.apache.activemq.artemis.core.server.level=INFO

# Root logger handlers
logger.handlers=FILE

# File handler configuration
handler.FILE=org.jboss.logmanager.handlers.FileHandler
handler.FILE.level=FINE
handler.FILE.properties=autoFlush,fileName
handler.FILE.autoFlush=true
handler.FILE.fileName=activemq.log
handler.FILE.formatter=PATTERN

# Formatter pattern configuration
formatter.PATTERN=org.jboss.logmanager.formatters.PatternFormatter
formatter.PATTERN.properties=pattern
formatter.PATTERN.pattern=%d{HH:mm:ss,SSS} %-5p [%c] %s%E%n
```

3. 使用预期参数启动客户端。使用 **java** 命令启动客户端代码时，添加以下参数：

   a. 将 JBoss 客户端和日志记录器 JAR 添加到类路径中：

   ```
   -cp /PATH/TO/jboss-client.jar:/PATH/TO/jboss-logmanager.jar
   ```

   b. 启用 JBoss 日志记录管理器：

   ```
   -Djava.util.logging.manager=org.jboss.logmanager.LogManager
   ```

   c. 设置日志记录属性文件的位置：

   ```
   -Dlogging.configuration=/PATH/TO/logging.properties
   ```

   启动客户端的完整命令将类似以下示例：

```
$ java -Djava.util.logging.manager=org.jboss.logmanager.LogManager -Dlogging.configuration=/PATH/TO/logging.properties -cp /PATH/TO/jboss-client.jar:/PATH/TO/jboss-logmanager.jar org.example.MyClient
```

第 6 章 地址设置

messaging-activemq 子系统具有多个可配置的选项，这些选项控制消息的发送方式和时间、应进行的尝试数以及消息过期的时间。这些配置选项都存在于 **<address-setting>** 配置元素中。您可以使用通配符语法将 JBoss EAP 配置为将单个 **<address-setting>** 应用到多个目的地。

6.1. 通配符语法

通配符可用于将相似的地址与单个语句匹配，很像有多少系统使用星号字符 * 将多个文件或字符串与单个查询匹配。下表列出了可用于定义 **<address-setting>** 的特殊字符。

表 6.1. Jakarta Messaging Wildcard Syntax

字符	描述
.（单个句点）	表示通配符表达式中词语之间的空格.
#（井号或井号）	匹配由零个或多个词语组成的任意序列。
*（星号）	匹配单个单词。

下表中的示例说明了如何使用通配符匹配一组地址。

表 6.2. Jakarta Messaging Wildcard 示例

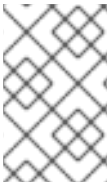
示例	描述
news.europe.#	匹配 **news.europe**、**news.europe.sport**、**news.europe.politics.fr**，但不会出现 **news.usa** 或 **europe**。
news.*	匹配 **news.europe** 和 **news.usa**，但不匹配 **news.europe.sport**。
news.*.sport	匹配 **news.europe.sport** 和 **news.usa.sport**，但不匹配 **news.europe.fr.sport**。

6.2. 默认 ADDRESS-SETTING

开箱即用，JBoss EAP 包含了单一 **address-setting** 元素，作为 **messaging-activemq** 子系统配置的一部分：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <address-setting
      name="#"
      dead-letter-address="jms.queue.DLQ"
      expiry-address="jms.queue.ExpiryQueue"
      max-size-bytes="10485760"
      page-size-bytes="2097152"
      message-counter-history-day-limit="10" />
```

```
    ...
  </server>
</subsystem>
```



注意

对 **name** 属性使用单个 **#** 使此默认 **address-setting** 的配置用于所有目的地，因为 **#** 匹配任何地址。您可以继续将此概括性配置应用到所有地址，或者您可以为每个需要设置自己配置的地址组添加新 **address-setting**。

使用管理 CLI 配置地址设置

配置地址设置可使用管理 CLI 或管理控制台进行，但管理 CLI 会公开更多用于编辑的配置属性。有关 属性的完整列表，请参阅本指南附录中的地址设置 属性。

添加新 address-setting

如果需要，使用 **add** 操作创建新的地址设置。您可以从管理 CLI 会话的根目录运行此命令，以下示例中创建名为 的新模式。您可以包含 **address-setting** 的配置属性。下面是创建一个新的 **address-setting** 匹配 **news.europe.#**，其 **dead-letter-address** 属性设置为队列 **DLQ.news**，该属性是提前创建的。单机服务器和受管域都使用 **full** 配置文件的示例分别显示。

```
/subsystem=messaging-activemq/server=default/address-setting=news.europe.#/:add(dead-letter-address=DLQ.news)
```

```
/profile=full/subsystem=messaging-activemq/server=default/address-setting=news.europe.#/:add(dead-letter-address=DLQ.news)
```

编辑 address-setting 属性

使用 **write-attribute** 操作，将新值写入到属性。您可以使用 Tab 补全来帮助完成键入的命令字符串，以及公开可用的属性。以下示例将 **max-delivery-attempts** 值更新为 **10**。

```
/subsystem=messaging-activemq/server=default/address-setting=news.europe.#/:write-attribute(name=max-delivery-attempts,value=10)
```

```
/profile=full/subsystem=messaging-activemq/server=default/address-setting=news.europe.#/:write-attribute(name=max-delivery-attempts,value=10)
```

读取 address-setting 属性

使用 include- **runtime=true** 参数运行 **read-resource** 操作来公开服务器模式中的所有当前值，以确认值已更改。

```
/subsystem=messaging-activemq/server=default/address-setting=news.europe.#/:read-resource(include-runtime=true)
```

```
/profile=full/subsystem=messaging-activemq/server=default/address-setting=news.europe.#/:read-resource(include-runtime=true)
```

使用管理控制台配置地址设置

您可以按照以下步骤使用管理控制台创建和查看地址设置：

1. 登录管理控制台。
2. 选择屏幕顶部的 **Configuration** 选项卡。运行受管域时，选择要更新的配置集。

3. 选择 **Messaging(ActiveMQ) → Server**。
4. 选择消息传递服务器。在默认配置中，仅显示一个名为 **default** 的服务器。
5. 选择 **Destinations** 并单击 **View**。
6. 选择 **地址设置** 选项卡来配置地址设置。

请记住，添加新模式时，如 **news.europe.#**，Pattern 字段引用 **address-setting** 元素的 **name** 属性。在使用管理 CLI 读取或写入属性时，您可以输入这个值。

在使用管理控制台时，您只能编辑 **dead-letter- address**、re**delivery -delay 和 max-delivery-attempts** 属性。其他属性必须通过管理 CLI 进行配置。

为消息传递服务器配置全局资源使用量

address-setting 元素中的三个 属性可帮助您控制消息传递服务器的全局资源使用情况：

属性	描述
global-max-memory-size	控制 Artemis 在将消息视为满之前用于存储其地址的最大内存量，并且其 **address-full-policy** 开始应用。默认值为 **-1**，这表示没有限制。
global-max-disk-usage	控制 Artemis 可用于在文件系统中存储数据的最大空间。达到限制时，任何新消息都会被阻断。此属性以磁盘上可用空间的百分比表示。最小值为 **0%**，最大值为 **100%**。默认值为 **100%**。
disk-scan-period	控制 Artemis 检查文件系统中可用空间的频率。默认值为 **5000 毫秒**。

6.3. LAST-VALUE QUEUES

最后值队列是特殊队列，当将值相同的新消息放入队列中时，丢弃任何消息。换句话说，最后一个值队列仅保留最后一个值。最后值队列的典型应用可能涉及股票价格，您仅对特定股票的最新价格感兴趣。

重要

如果队列启用了分页，则最后值队列将无法正常工作。务必在使用最后值队列之前 禁用分页。

配置 Last-value Queues

last-value 队列在 **address-setting** 配置元素中定义：

```
<address-setting name="jms.queue.lastValueQueue" last-value-queue="true" />
```

使用管理 CLI 为给定 **address -setting 读取 last-value- queue** 的值：

```
/subsystem=messaging-activemq/server=default/address-setting=news.europe.#:read-
attribute(name=last-value-queue)
{
```

```
    "outcome" => "success",
    "result" => false
}
```

last-value-queue 的可接受的值为 **true** 或 **false**。使用管理 CLI 设置任一值，如下所示：

```
/subsystem=messaging-activemq/server=default/address-setting=news.europe.#:write-attribute(name=last-value-queue,value=true)

/subsystem=messaging-activemq/server=default/address-setting=news.asia.#:write-attribute(name=last-value-queue,value=false)
```

使用 Last-value 属性

用于标识最后一个值的属性名称为 **_AMQ_LVQ_NAME** （或核心 API 中的恒定 **Message.HDR_LAST_VALUE_NAME** ）。让以下 Java 代码演示了如何使用 last-value 属性。

- 首先，发布者向最后一个值队列发送消息

```
TextMessage message = session.createTextMessage("My 1st message with the last-value property set");
message.setStringProperty("_AMQ_LVQ_NAME", "MY_MESSAGE");
producer.send(message);
```

- 然后，它会使用相同的最后值向队列发送另一条消息

```
message = session.createTextMessage("My 2nd message with the last-value property set");
message.setStringProperty("_AMQ_LVQ_NAME", "MY_MESSAGE");
producer.send(message);
```

- 接下来，消费者收到带有最后值的消息

```
TextMessage messageReceived = (TextMessage)messageConsumer.receive(5000);
System.out.format("Received message: %s\n", messageReceived.getText());
```

在上例中，客户端的输出将是 **"My 2nd 消息，并设置了 last-value 属性"**，因为两条消息都 设置为 **"MY_MESSAGE"**，在队列中在第一个消息后收到第二条消息。

第 7 章 配置安全性

7.1. 保护远程连接

7.1.1. 使用传统安全子系统

您可以使用 JBoss EAP 中的传统 **安全** 子系统来保护 **messaging-activemq** 子系统。传统的 **security** 子系统使用传统安全域和域。有关安全域 *和安全 域 的更多信息，请参阅 JBoss EAP 安全架构指南*。**messaging-activemq** 子系统已预先配置，以使用名为 **ApplicationRealm** 的安全域和名为 **other** 的安全域。

注意

旧版 **安全性** 子系统方法是 JBoss EAP 7.0 中的默认配置。

ApplicationRealm 在配置文件顶部附近定义。

```
<management>
  <security-realms>
     ...
     <security-realm name="ApplicationRealm">
       <authentication>
          <local default-user="$local" allowed-users="*" skip-group-loading="true"/>
          <properties
            path="application-users.properties"
            relative-to="jboss.server.config.dir" />
       </authentication>
       <authorization>
          <properties
            path="application-roles.properties"
            relative-to="jboss.server.config.dir" />
       </authorization>
     </security-realm>
  </security-realms>
  ...
</management>
```

顾名思义，**ApplicationRealm** 是 JBoss EAP 中以应用为焦点的所有子系统（如 **messaging-activemq**、**undertow** 和 **ejb3** 子系统）的默认安全域。**ApplicationRealm** 使用本地文件系统存储用户名和哈希密码。为方便起见，JBoss EAP 包含一个可用于添加用户到 **ApplicationRealm** 的脚本。详情请参阅 JBoss EAP *如何 配置服务器安全指南中的默认 用户* 配置。

其他 安全域是与应用相关的子系统（如 **messaging-activemq** ）的默认安全域。它在配置中没有被显式声明，但您可以使用以下管理 CLI 命令确认 **messaging-activemq** 子系统使用了哪一个安全域：

```
/subsystem=messaging-activemq/server=default:read-attribute(name=security-domain)
{
    "outcome" => "success",
    "result" => "other"
}
```

您还可以更新使用哪个安全域：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=security-domain,
value=mySecurityDomain)
```

JBoss EAP *如何配置服务器安全* 指南包含有关如何创建新安全域和域的更多信息。目前，需要注意 **其他** 域如何出现在配置中：

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
    <security-domains>
        <security-domain name="other" cache-type="default">
            <authentication>
                <login-module code="Remoting" flag="optional">
                    <module-option name="password-stacking" value="useFirstPass"/>
                </login-module>
                <login-module code="RealmDirect" flag="required">
                    <module-option name="password-stacking" value="useFirstPass"/>
                </login-module>
            </authentication>
        </security-domain>
        ...
    <security-domains>
 </subsystem>
```

"other"域使用两个登录模块作为其身份验证方式。第一个模块 **Remoting** 验证远程 Jakarta Enterprise Beans 调用，而 **RealmDirect** 模块使用给定域中定义的信息存储来验证用户身份。本例中使用了默认的 realm **ApplicationRealm**，因为没有声明 realm。每个模块 **的 password-stacking** 选项设置为 **使用 FirstPass**，它会告诉 login-module 存储经过身份验证的用户的主体名称和密码。有关登录 *模块及其选项的更多详细信息，请参阅 JBoss EAP 登录模块参考*。

基于角色的访问权限在地址级别上配置，请参阅基于角色的安全性以了解地址。

7.1.2. 使用 Elytron 子系统

您也可以使用 **elytron** 子系统来保护 **messaging-activemq** 子系统。您可以在" *配置身份管理* 指南》中的"使用 **elytron** 子系统"一节中找到有关使用 elytron 子系统以及创建和 Elytron 安全域的更多信息。

使用 Elytron 安全域：

1. 取消定义传统安全域。

```
/subsystem=messaging-activemq/server=default:undefine-attribute(name=security-domain)
```

2. 设置 Elytron 安全域。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=elytron-domain,
value=myElytronSecurityDomain)

reload
```

7.1.2.1. 使用管理控制台设置 Elytron 安全域

使用管理控制台设置 Elytron 安全域：

1. 访问管理控制台。如需更多信息，请参阅 JBoss EAP 配置指南中的 管理控制台。

2. 导航到 **Configuration → Subsystems → Messaging(ActiveMQ) → Server → default**，然后点 **View**。
3. 导航到 **Security** 选项卡，再单击 **Edit**。
4. 添加或编辑 **Elytron Domain** 的值。
5. 单击 **Save** 以保存更改。
6. 重新加载服务器以使更改生效。

注意

您只能定义 **security-domain** 或 **elytron-domain**，但不能同时定义这两个域。如果未定义，JBoss EAP 将使用 **其他 的安全域** 默认值，它们映射到 **其他** 传统安全域。

7.1.3. 保护传输

默认情况下，JBoss EAP 消息传递捆绑的默认 **http-connector** 不受保护。您可以按照说明为 SSL/TLS 保护消息传输和启用 Web 流量，从而 针对 *如何为 JBoss EAP 配置服务器安全性的应用配置单向* 和双向 SSL/TLS。

注意

以上保护消息传输的方法也可用于保护 **http-acceptor**。

在按上述方式配置传输时，您必须执行以下步骤：

- 默认情况下，所有 HTTP 接收器都配置为使用默认的 **http-listener**，它侦听 HTTP 端口。您必须将 HTTP 接收器配置为使用 **https-listener**，它侦听 HTTPS 端口。
- 必须更新所有 HTTP 连接器的 **socket-binding** 元素，以使用 **https** 而不是 **http**。
- 通过 SSL/TLS 通信的每个 **http-connector** 都必须将 **启用ssl** 的参数设置为 **true**。
- 如果使用 HTTP 连接器连接到其他服务器，您必须配置相关参数，如 **trust-store** 和 **key-store**。保护 **http-connector** 要求您配置与 **远程连接器相同的参数，** 如 保护远程连接器所述。

如需有关 为消息传递传输配置接收器和连接器的信息，请参阅配置消息传递传输。

7.1.4. 保护远程连接器

如果您没有使用默认的 **http-connector**，而是创建了自己的 **remote-connector** 和 **remote-acceptor** 进行 TCP 通讯，您可以使用下表中的属性为 SSL/TLS 配置 SSL/TLS。属性显示在配置中，作为接收器或连接器的子 **<param> 元素** 的一部分。

通常，服务器拥有其专用 SSL/TLS 密钥，并与客户端共享其公钥。在这种情况下，服务器在 **远程接收器中定义 key- store-path 和 key-store-password 参数**。由于每个客户端都可以位于不同的位置，并且使用不同的密码加密，因此不建议在 **remote -connector 上指定 trust-store-path 和 trust- store-password** 属性。取而代之，使用系统属性 **javax.net.ssl.trustStore 和 javax.net .ssl.trustStorePassword** 在客户端上配置这些参数。为 **远程连接器配置的参数是 ssl-enabled=true，并使用DefaultSslContext=true**。但是，如果服务器 **使用远程连接器** 连接到另一服务器，那么在这种情况下，为**远程** 连接器设置 **trust-store-path** 和 **trust-store-password** 参数有意义。

在以上用例中，**remote-acceptor** 会使用以下管理 CLI 命令创建：

```
/subsystem=messaging-activemq/server=default/remote-acceptor=mySslAcceptor:add(socket-binding=netty,params={ssl-enabled=true, key-store-path=PATH/TO/server.jks, key-store-password=${VAULT::server-key::key-store-password::sharedKey}})
```

要根据以上用例创建 **remote-connector**，请使用以下管理 CLI 命令：

```
/subsystem=messaging-activemq/server=default/remote-connector=mySslConnector:add(socket-binding=netty,params={ssl-enabled=true, useDefaultSslContext=true})
```

管理 CLI 还允许您在已存在的 **remote-acceptor** 或 **remote-connector** 中添加参数：

```
/subsystem=messaging-activemq/server=default/remote-connector=myOtherSslConnector:map-put(name=params,key=ssl-enabled,value=true)
```

请注意，**remote-acceptor** 和 **remote-connector** 都引用 **套接字绑定来声明要** 用于通信的端口。如需有关套接字绑定及其与接收器和连接器关系的更多信息，请参阅 消息传递子系统配置概述。

表 7.1. NettyConnectorFactory的 SSL/TLS 相关配置属性

属性	描述
enabled-cipher-suites	可用于配置接收器或连接器。这是用逗号分开的密码套件列表，用于 SSL/TLS 通信。默认值为 null，表示将使用 JVM 的默认值。
enabled-protocols	可用于配置接收器或连接器。这是用于 SSL/TLS 通信的以逗号分隔的协议列表。默认值为 null，表示将使用 JVM 的默认值。
key-store-password	在接收器上使用时，这是服务器端密钥存储的密码。 在连接器上使用时，这是客户端密钥存储的密码。只有在使用双向 SSL/TLS 时，这才与连接器相关。虽然可以在服务器上配置此值，但它会被客户端下载和使用。 如果客户端需要使用与服务器上设置的不同密码不同的密码，它可以使用标准的 **javax.net.ssl.keyStorePassword** 系统属性覆盖服务器端设置。如果客户端上的另一个组件已在使用标准系统属性，则使用 **org.apache.activemq.ssl.keyStorePassword** 属性。
key-store-path	在接收器上使用时，这是服务器上保存服务器证书的 SSL/TLS 密钥存储的路径。对证书使用自签名或由颁发机构签名的证书。 在连接器中使用时，这是客户端 SSL/TLS 密钥存储的路径，该存储存放客户端证书。只有在使用双向 SSL/TLS 时，这才与连接器相关。 虽然此值在服务器上配置，但它会被客户端下载和使用。如果客户端需要使用与服务器上设置的其他路径不同的路径，它可以通过使用标准 **javax.net.ssl.keyStore** 系统属性来覆盖服务器端设置。如果客户端上的另一个组件已在使用 **standard 属性，则使用 org.apache.activemq.ssl.keyStore** 系统属性。
key-store-provider	定义在其中存储密钥的文件格式，例如 PKCS11 或 PKCS12。接受的值是 JDK 特定的。

属性	描述
needs-client-auth	此属性仅适用于接收器。它告知客户端连接到此接收器的客户端，需要双向 SSL/TLS。有效值为 **true** 或者 **false**。默认为 **false**。
启用 SSL	必须为 **true** 以启用 SSL/TLS。默认为 **false**。
trust-store-password	在接收器上使用时，这是服务器端信任存储的密码。只有在使用双向 SSL/TLS 时，这才与接收器相关。 在连接器中使用时，这是客户端信任存储的密码。虽然可以在服务器上配置此值，但它会被客户端下载和使用。 如果客户端需要使用与服务器上设置的不同密码，则可以使用标准的 **javax.net.ssl.trustStorePassword** 系统属性来覆盖服务器端设置。如果客户端上的另一个组件已在使用 **standard 属性，则使用 org.apache.activemq.ssl.trustStorePassword** 系统属性。
trust-store-path	在接收器上使用时，这是服务器端 SSL/TLS 密钥存储的路径，该存储保存服务器信任的所有客户端的密钥。只有在使用双向 SSL/TLS 时，这才与接收器相关。 在连接器中使用时，这是客户端 SSL/TLS 密钥存储的路径，其中包含客户端信任的所有服务器的公钥。虽然可以在服务器上配置此值，但它会被客户端下载和使用。 如果客户端需要使用与服务器上该集合不同的路径，则可以使用标准的 **javax.net.ssl.trustStore** 系统属性来覆盖服务器端设置。如果客户端上的另一个组件已在使用标准系统属性，则使用 **org.apache.activemq.ssl.trustStore** 系统属性。
trust-store-provider	定义在其中存储密钥的文件格式，例如 PKCS11 或 PKCS12。接受的值是 JDK 特定的。

7.2. 保护目标

除了保护远程连接到消息传递服务器外，您还可以配置特定目的地的安全性。这可以通过使用 **security-setting** 配置元素添加安全约束来实现。JBoss EAP 消息传递默认配置有 **security-setting**，如以下管理 CLI 命令的输出所示：

```
/subsystem=messaging-activemq/server=default:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        ....
        "security-setting" => {"#" => {"role" => {"guest" => {
            "consume" => true,
            "create-durable-queue" => false,
            "create-non-durable-queue" => true,
            "delete-durable-queue" => false,
            "delete-non-durable-queue" => true,
            "manage" => false,
```

```
            "send" => true
        }}}}
    }
}
```

security-setting 选项在 **name** 字段中使用通配符来处理要应用安全性约束的目的地。单个 **#** 的值将匹配任何地址。有关在安全约束中使用通配符的更多信息，请参阅基于角色的安全性。

7.2.1. 地址基于角色的安全性

JBoss EAP 消息传递包含一种灵活的基于角色的安全模型，用于根据队列的地址将安全性应用到队列中。

核心 JBoss EAP 消息传递服务器主要由绑定至地址的队列集组成。将消息发送到地址时，服务器首先查找绑定到该地址的一组队列，然后将消息路由到绑定队列。

JBoss EAP 消息传递具有一组权限，可以根据队列的地址对其应用。可以使用地址上的完全字符串匹配，也可以使用通配符字符 **#** 和 ***。有关如何使用通配符语法的更多信息，请参阅 地址设置。

您可以为每个 **security-setting** 创建多个角色，并且有 7 个权限设置可应用于角色。以下是可用权限的完整列表：

- **create-durable-queue** 允许角色在匹配的地址下创建持久队列。
- **delete-durable-queue** 允许角色删除匹配地址下的持久队列。
- **create-non-durable-queue** 允许角色在匹配地址下创建不可处理队列。
- **delete-non-durable-queue** 允许角色删除匹配地址下的不可解析队列。
- **send** 允许角色发送消息到匹配的地址。
- **使用** 时，该角色可以使用绑定到匹配地址的队列中的消息。
- **管理** 允许角色通过发送管理消息到管理地址来调用管理操作。

配置基于角色的安全性
要开始将基于角色的安全性用于 **security-setting**，您必须首先创建一个：例如，下方创建了一个 **security-setting of news.europe.#**。它适用于从 **news.europe 开始的任何目的地，** 如 **news.europe.fr 或 news.europe.tech.uk**。

```
/subsystem=messaging-activemq/server=default/security-setting=news.europe.#:add()
{"outcome" => "success"}
```

接下来，您要将角色添加到您创建的 **security-setting** 中并为它声明权限。在以下示例中，创建 **dev** 角色并授予使用和发送到队列的权限，以及创建和删除不可调度队列的权限。由于默认值为 **false**，因此您必须仅告知 JBoss EAP 要打开的权限。

```
/subsystem=messaging-activemq/server=default/security-
setting=news.europe.#/role=dev:add(consume=true,delete-non-durable-queue=true,create-non-
durable-queue=true,send=true)
{"outcome" => "success"}
```

为了进一步说明权限的使用，以下示例创建了 **管理员** 角色，并允许它通过切换管理权限来发送 **管理** 消息。另外，也会切换和删除持久队列的权限：

```
/subsystem=messaging-activemq/server=default/security-
setting=news.europe.#/role=admin:add(manage=true,create-durable-queue=true,delete-durable-
queue=true)
{"outcome" => "success"}
```

若要确认 **security-setting** 的配置，可使用管理 CLI。记得使用 **recursive=true** 选项来完全显示权限：

```
/subsystem=messaging-activemq/server=default:read-children-resources(child-type=security-
setting,recursive=true)
{
    "outcome" => "success",
    "result" => {
        "#" => {"role" => {"guest" => {
            "consume" => true,
            "create-durable-queue" => false,
            "create-non-durable-queue" => true,
            "delete-durable-queue" => false,
            "delete-non-durable-queue" => true,
            "manage" => false,
            "send" => true
        }}},
        "news.europe.#" => {"role" => {
            "dev" => {
                "consume" => true,
                "create-durable-queue" => false,
                "create-non-durable-queue" => true,
                "delete-durable-queue" => false,
                "delete-non-durable-queue" => true,
                "manage" => false,
                "send" => true
            },
            "admin" => {
                "consume" => false,
                "create-durable-queue" => true,
                "create-non-durable-queue" => false,
                "delete-durable-queue" => true,
                "delete-non-durable-queue" => false,
                "manage" => true,
                "send" => false
            }
        }}
    }
```

上面，管理 CLI 完全 显**示以字符串 news.europe**. 开头的地址的权限。总之，只有具备 **admin 角色的用**户 才能创建或删除持久队列，而只有具有 **dev** 角色的用户才能创建或删除不可覆盖的队列。此外，具有 **dev 角色的用**户 可以发送或消耗消息，但 **管理**员**用**户 不能。但是，他们可以发送管理消息，因为其 **管理**权限已设为 **true**。

如果多个匹配项适用于一组地址，则更具体的匹配项具有优先权。例如，地址 **news.europe.tech.uk.#** 比 **news.europe.tech.#** 更具体。由于不会继承权限，因此您可以通过不指定权限来有效地拒绝更具体的 **security-setting** 块中的权限。否则，将无法拒绝地址子组的权限。

用户之间的映射和他们拥有的角色之间的映射由安全管理器处理。JBoss EAP 附带了一个用户经理，从磁盘上的文件读取用户凭据，也可插入 JAAS 或 JBoss EAP 安全性。

有关配置安全管理器的更多信息，请参阅 JBoss EAP *安全架构指南*。

7.2.1.1. 使用传统安全子系统授予未经身份验证的客户端角色

如果您希望 JBoss EAP 自动向未经身份验证的客户端授予客户端，**客户机** 角色将进行以下两项更改：

1. 将新 **模块选项** 添加到 **其他** 安全域。新选项(**unauthenticatedIdentity**)将告知 JBoss EAP 向未经身份验证的客户端授予 **guest** 访问权限。推荐的做法是使用管理 CLI:

```
/subsystem=security/security-domain=other/authentication=classic/login-
module=RealmDirect:map-put(name=module-
options,key=unauthenticatedIdentity,value=guest)
{
  "outcome" => "success",
  "response-headers" => {
      "operation-requires-reload" => true,
      "process-state" => "reload-required"
  }
}
```

请注意，在发出 命令后，服务器需要重新加载。您可以使用以下管理 CLI 命令确认新选项：

```
/subsystem=security/security-domain=other/authentication=classic/login-
module=RealmDirect:read-resource()
{
   "outcome" => "success",
   "result" => {
       "code" => "RealmDirect",
       "flag" => "required",
       "module" => undefined,
       "module-options" => {
           "password-stacking" => "useFirstPass",
           "unauthenticatedIdentity" => "guest"
       }
   }
}
```

另外，在执行该命令后，您的服务器配置文件应类似如下：

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
 <security-domains>
  <security-domain name="other" cache-type="default">
   <authentication>
    ...
    <login-module code="RealmDirect" flag="required">
     ...
     <module-option name="unauthenticatedIdentity" value="guest"/>
     ...
    </login-module>
    ...
   </authentication>
  </security-domain>
 ...
 </security-domains>
</subsystem>
```

2. 通过删除 **#** 字符，取消注释 **application-roles.properties** 文件中的以下行。该文件位于 ***EAP_HOME*/standalone/configuration/** 或 ***EAP_HOME*/domain/configuration/** 中，具体取决于您是分别使用单机服务器还是域控制器。

```
#guest=guest
```

远程客户端现在应当能够访问服务器，而无需进行身份验证。他们将获得与 **guest** 角色相关联的权限。

7.3. 控制 JAKARTA MESSAGING OBJECTMESSAGE DESERIALIZATION

由于 **ObjectMessage** 可以包含潜在的危险对象，ActiveMQ Artemis 提供了一个简单的类过滤机制，以控制哪些软件包和类值得信任且不被信任。您可以将其类从可信软件包中的对象添加到白名单，以指明它们可以无问题地进行非序列化。您可以将其类从不受信任的软件包添加到黑列表中的对象，以防止它们被反序列化。

ActiveMQ Artemis 过滤对象以进行反序列化，如下所示：

- 如果白名单和黑名单都为空（默认值），则允许任何可序列化的对象进行反序列化。
- 如果对象的类或软件包与黑名单中的某一条目匹配，则不允许进行反序列化。
- 如果对象的类或软件包与白名单中的条目匹配，则允许进行反序列化。
- 如果对象的类或软件包与黑名单和白名单中的条目匹配，则黑名单中的条目具有优先权，这意味着不允许进行反序列化。
- 如果对象的类或软件包都不匹配黑名单或白名单，则拒绝对象降序化，除非白名单为空，否则没有指定白名单。

如果对象全名与列表中的某一条目完全匹配，或者其软件包与列表中某一条目匹配，或者是列表中某一条目的子软件包，则对象被视为匹配项。

您可以使用 deserialization- **white-list 和 deserialization- black-list 属性，在 connection-factory 和 pooled-connection - factory 中指定哪些对象可以被反序列化。deserialization-white-list** 属性用于定义允许进行反序列化的类或软件包的列表。**deserialization-black-list** 属性用于定义不允许进行反序列化的类或软件包的列表。

以下命令为 **RemoteConnectionFactory** 连接工厂创建一个黑名单，并为默认服务器创建 **activemq-ra** 池连接工厂的白名单。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=deserialization-black-list,value=
[my.untrusted.package,another.untrusted.package])
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra:write-
attribute(name=deserialization-white-list,value=[my.trusted.package])
```

这些命令在 **messaging-activemq** 子系统中生成以下配置：

```
<connection-factory name="RemoteConnectionFactory"
entries="java:jboss/exported/jms/RemoteConnectionFactory" connectors="http-connector" ha="true"
block-on-acknowledge="true" reconnect-attempts="-1" deserialization-black-
list="my.untrusted.package another.untrusted.package"/>
<pooled-connection-factory name="activemq-ra" entries="java:/JmsXA
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm" deserialization-white-
list="my.trusted.package" transaction="xa"/>
```

有关连接工厂和池连接工厂的详情，请参考本指南中的 配置连接工厂。

您还可以通过配置激活属性在 MDB 中指定哪些对象可以被反序列化。**deserializationWhiteList** 属性用于定义允许进行反序列化的类或软件包的列表。**deserializationBlackList** 属性用于定义不允许进行反序列化的类或软件包列表。如需有关激活属性的更多信息，请参阅为 JBoss EAP *开发 Jakarta 企业 Bean 应用* 时使用 部署描述符配置 MDB。

7.4. 授权无效管理

messaging **- activemq 子系统中服务器上的 security- invalidation-interval** 属性决定了在必须重新授权前缓存授权的时长。

系统授权用户在地址上执行操作时，将缓存授权。下次同一用户在同一地址上执行相同的操作时，系统会将缓存的授权用于操作。

例如，用户 **admin** 尝试向地址 **新闻** 发送消息。系统授权操作，并缓存授权。下一次 **管理员** 尝试向 news 发送消息时， 系统会使用缓存的授权。

如果缓存的授权没有在无效间隔指定的时间内再次使用，则会从缓存中清除授权。系统必须重新授权用户，以便在请求的地址执行请求的操作。

安装后，JBoss EAP 假定默认值为 10000 毫秒（10 秒）。

```
/subsystem=messaging-activemq/server=default:read-attribute(name=security-invalidation-interval)
{
    "outcome" => "success",
    "result" => 10000L
}
```

security-invalidation-interval 属性可以配置。例如，以下命令将间隔更新为 60000 毫秒（60 秒或一分钟）。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=security-invalidation-
interval,value=60000)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

您必须重新加载服务器才能使配置修改生效。

读取 属性会显示新结果。

```
/subsystem=messaging-activemq/server=default:read-attribute(name=security-invalidation-interval)
{
    "outcome" => "success",
    "result" => 60000L
}
```

第 8 章 配置消息传递传输

本节介绍理解 JBoss EAP 消息传递传输的关键概念，特别是连接器和接收器。服务器上使用接收器来定义其如何接受连接，而客户端则使用连接器来定义其如何连接到服务器。逐个讨论每个概念，一个实用示例演示了客户端如何使用 JNDI 或核心 API 连接 JBoss EAP 消息传递服务器。

8.1. 接收器和连接器类型

JBoss EAP 配置中定义了三种主要类型的接收器和连接器：

in-vm ：In-vm 是 Intra 虚拟机的缩写。如果客户端和服务器都在同一个 JVM 中运行，例如在同一 JBoss EAP 实例中运行的 Message Driven Beans(MDB)，可使用这种连接器类型。

HTTP ：在不同 JVM 中运行客户端和服务器时使用。使用 **undertow** 子系统的默认端口 **8080**，因此能够通过 HTTP 进行多路消息传递通信。由于端口管理等注意事项，红帽建议在客户端和服务器在不同 JVM 中运行 **http** 连接器，特别是在云环境中。

远程： **远程** 传输是基于 Netty 的组件，当客户端和服务器在不同 JVM 中运行时，用于原生 TCP 通信。在无法使用 **http** 时替代 http。

客户端必须使用与服务器接收器之一兼容的连接器。例如，只有 **in-vm-connector** 可以连接到 **in-vm-acceptor**，并且只有 **http-connector** 可以连接到 **http-acceptor** 等。

您可以使用 **read-children-attributes** 操作，显示管理 CLI 列表给定接收器或连接器类型的属性。例如，要查看您将输入的默认消息传递服务器的所有 **http-connectors** 属性：

```
/subsystem=messaging-activemq/server=default:read-children-resources(child-type=http-connector,include-runtime=true)
```

所有 **http-acceptors** 的属性都使用类似的命令读取：

```
/subsystem=messaging-activemq/server=default:read-children-resources(child-type=http-acceptor,include-runtime=true)
```

其他接收器和连接器类型遵循相同的语法。仅向 **子类型** 提供接收器或连接器类型，如 **remote-connector 或 in -vm-acceptor**。

8.2. 接收器

接收器定义 JBoss EAP 集成消息传递服务器接受哪些类型的连接。您可以定义每台服务器任意数量的接收器。以下示例配置是从默认的 **full-ha** 配置文件修改的，并提供了每个接收器类型的示例。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <http-acceptor name="http-acceptor" http-listener="default"/>
    <remote-acceptor name="legacy-messaging-acceptor" socket-binding="legacy-messaging"/>
    <in-vm-acceptor name="in-vm" server-id="0"/>
    ...
  </server>
</subsystem>
```

在上述配置中，http **-acceptor** 使用 Undertow 的默认 **http-listener**，它侦听 JBoss EAP 的默认 http 端口 8080。**http-listener** 在 **undertow** 子系统中定义：

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0">
  ...
  <server name="default-server">
    <http-listener name="default" redirect-socket="https" socket-binding="http"/>
    ...
  </server>
  ...
</subsystem>
```

另请注意，上面的 **remote-acceptor** 如何使用名为 **legacy -messaging 的 socket-binding**，这会在稍后配置中作为服务器的默认 **socket-binding-group** 的一部分进行定义。

```
<server xmlns="urn:jboss:domain:8.0">
  ...
  <socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
      ...
      <socket-binding name="legacy-messaging" port="5445"/>
      ...
  </socket-binding-group>
</server>
```

在本例中，传统**-messaging socket-binding** 将 JBoss EAP 绑定到端口 **5445**，上面的远程 接收器则声明代表 **messaging-activemq** 子系统供传统客户端使用的端口。

最后，**in-vm-acceptor** 对 **server-id** 属性使用唯一值，以便此服务器实例可以和可能在同一 JVM 中运行的其他服务器区分开来。

8.3. 连接器

连接器定义如何连接集成 JBoss EAP 消息传递服务器，供客户端用于连接。

您可能会知道为什么服务器上定义了连接器（当客户端实际使用时）。原因包括：

- 在某些情况下，当服务器连接到其他服务器时，它可能充当客户端。例如，一个服务器可能充当另一台服务器的桥梁，或者它可能希望参与到群集中。在这种情况下，服务器需要了解如何连接到其他服务器，并通过连接器定义。
- 服务器可以使用 **ConnectionFactory** 提供连接器，该连接器由客户端使用 JNDI 进行查找，因此创建服务器连接更加简单。

您可以定义每台服务器任意数量的连接器。以下示例配置基于 **full-ha** 配置配置文件，并包含每种类型的连接器。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <http-connector name="http-connector" endpoint="http-acceptor" socket-binding="http" server-
name="messaging-server-1"/>
    <remote-connector name="legacy-remoting-connector" socket-binding="legacy-remoting"/>
    <in-vm-connector name="in-vm" server-id="0"/>
    ...
  </server>
</subsystem>
```

与 **full -ha 配置文件中的 http- acceptor** 一样，**http-connector** 使用 **undertow** 子系统定义的默认 **http-listener**。**endpoint** 属性声明要连接的 **http-acceptor**。在这种情况下，连接器将连接到默认的 **http-acceptor**。

JBoss EAP 7.1 为 **http -connector 引入了一个新的 server- name** 属性。此新属性为可选，但需要能够在运行多个 ActiveMQ Artemis 实例的远程服务器上连接正确的 **http-acceptor**。如果未定义此属性，则该值在运行时解析为定义了连接器的父 ActiveMQ Artemis 服务器的名称。

另外，请注意 **remote-connector** 引用的 **socket-binding 与其远程接收器相同。**最后，**in-vm-connector** 对 **server-id** 使用与 **in-vm-acceptor** 相同的值，因为它们在同一服务器实例中运行。

注意

如果公共接口的绑定地址设为 **0.0.0.0**，您会在启动 JBoss EAP 服务器时看到以下警告信息：

```
AMQ121005: Invalid "host" value "0.0.0.0" detected for "connector" connector.
Switching to <HOST_NAME>. If this new address is incorrect please manually
configure the connector to use the proper one.
```

这是因为远程连接器无法使用 **0.0.0.0** 地址连接到服务器，并且 **messaging-activemq** 子系统会尝试将它替换为服务器的主机名。管理员应将远程连接器配置为使用不同的接口地址进行套接字绑定。

8.4. 配置接受器和连接器

连接器和接收器有许多配置选项。它们在配置中显示为子 **<param>** 元素。每个 **<param>** 元素包含一个 **名称和 值** 属性对，由默认的基于 Netty 的工厂类理解和使用，该类负责实例化连接器或接收器。

在管理 CLI 中，每个远程连接器或接收器元素都包含参数名称和值对的内部映射。例如，要将新的 **param** 添加到 **名为 myRemote 的远程** 连接器中，请使用以下命令：

```
/subsystem=messaging-activemq/server=default/remote-connector=myRemote:map-
put(name=params,key=foo,value=bar)
```

使用类似语法检索参数值。

```
/subsystem=messaging-activemq/server=default/remote-connector=myRemote:map-
get(name=params,key=foo)
{
    "outcome" => "success",
    "result" => "bar"
}
```

您还可以在创建接收器或连接器时包含参数，如下例所示。

```
/subsystem=messaging-activemq/server=default/remote-connector=myRemote:add(socket-
binding=mysocket,params={foo=bar,foo2=bar2})
```

表 8.1. 传输配置属性

属性	描述
batch-delay	在将数据包写入传输之前，可以将消息传递服务器配置为批处理写入，最多以毫秒为 **单位进行批处理-delay**。通过增加消息传输的平均延迟，这会增加非常小消息的总吞吐量。默认值为 0。
direct-deliver	当消息到达服务器并传送到等待的消费者时，默认情况下会在邮件到达的同一线程上执行传输。这为具有相对较小的消息和少量消费者的环境提供了良好的延迟，但降低了吞吐量和延迟。对于最高吞吐量，您可以将此属性设为 **false**。默认值为 **true**。
http-upgrade-enabled	http-connector 使用 **http-connector** 指定它使用 HTTP 升级，因此要通过 HTTP 对消息传递流量进行多路复用。当 **http-connector** 创建且不需要管理员时，JBoss EAP 会自动将此属性设置为 **true**。
http-upgrade-endpoint	指定 **http-acceptor** 在 http-**connector 将连接的服务器端的 http-acceptor**。连接器将通过 HTTP 进行多路复用，并且需要此信息在 HTTP 升级后查找相关的 **http-acceptor**。当 **http-connector** 创建且不需要管理员时，JBoss EAP 会自动设置此属性。
local-address	对于 http 或远程连接器，这用于指定在连接到远程地址时客户端将使用的本地地址。如果未指定本地地址，则连接器将使用任何可用的本地地址。
local-port	对于 http 或远程连接器，这用于指定客户端在连接到远程地址时使用的本地端口。如果使用 local-port 默认值(0)，则连接器将允许系统采用临时端口。有效端口值为 **0** 到 **65535**。
nio-remoting-threads	如果配置为使用 NIO，则消息将默认使用等于 **Runtime.getRuntime（）.availableProcessors（）报告的内核数（或超线程）** 的三倍线程。要覆盖这个值，您可以为线程数量设置自定义值。默认值为 **-1**。
tcp-no-delay	如果 **情况如此**，将启用 Nagle 的算法。此算法通过减少通过网络发送的数据包数量，帮助提高 TCP/IP 网络的效率。默认值为 **true**。
tcp-send-buffer-size	此参数以字节为单位确定 TCP 发送缓冲区的大小。默认值为 **32768**。
tcp-receive-buffer-size	此参数以字节为单位确定 TCP 接收缓冲区的大小。默认值为 **32768**。
use-nio-global-worker-pool	此参数确保所有 Jakarta 消息传递连接共享一个 Java 线程池，而不是每个连接都有自己的池。这有助于避免耗尽操作系统中进程的最大数量。默认值为 **true**。

8.5. 连接到服务器

如果要将客户端连接到服务器，您必须有一个正确的连接器。有两种方法可以做到这一点。您可以使用服务器上配置的 ConnectionFactory，可通过 JNDI 查找获取。或者，也可以使用 ActiveMQ Artemis 核心 API，并在客户端上配置整个 **ConnectionFactory**。

8.5.1. Jakarta Messaging Connection Factories

客户端可以使用 JNDI 来查找提供服务器连接的 ConnectionFactory 对象。连接事实可以公开三种连接器类型中的每种类型：

远程客户端可以使用 **引用 remote-connector** 的连接事实来向服务器发送消息或从服务器接收消息（假设 connection-factory 具有适当的导出的条目）。**远程连接器与 socket-binding 关联，该** 绑定使用 **connection-factory** 在哪里连接来告知客户端。

引用 in- vm-connector 的连接事实适合供本地客户端用于向本地服务器发送消息或接收消息。**in-vm-connector** 与 **server-id** 关联，后者使用 **connection-factory** 进行连接的位置告知客户端，因为多个消息传递服务器可以在单个 JVM 中运行。

在升级到消息传递协议之前，**引用 http-connector 的连接** 事实适合供远程客户端用于通过连接到服务器的 HTTP 端口来发送消息或从服务器接收消息。**http-connector** 与代表 HTTP **套接字的 socket-binding** 关联，默认为 **http**。

自 Jakarta Messaging 2.0 起，默认的 Jakarta Messaging 连接工厂可供 JNDI 名称 **java:comp/DefaultJMSConnectionFactory** 下的 Jakarta EE 应用访问。 **messaging-activemq** 子系统定义一个 **pooled-connection-factory**，用于提供此默认连接工厂。

以下是 JBoss EAP **完整** 配置配置文件中所含的默认连接器和连接工厂：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    [...]
    <http-connector name="http-connector" socket-binding="http" endpoint="http-acceptor" />
    <http-connector name="http-connector-throughput" socket-binding="http" endpoint="http-acceptor-
throughput">
      <param name="batch-delay" value="50"/>
    </http-connector>
    <in-vm-connector name="in-vm" server-id="0"/>
    [...]
    <connection-factory name="InVmConnectionFactory" connectors="in-vm"
entries="java:/ConnectionFactory" />
    <pooled-connection-factory name="activemq-ra" transaction="xa" connectors="in-vm"
entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory"/>
    [...]
    </server>
</subsystem>
```

工厂的 **entries** 属性指定将工厂公开的 JNDI 名称。只有 **java:jboss/exported** 命名空间内绑定的 JNDI 名称可用于远程客户端。如果 **connection-factory** 在 **java:jboss/exported** 命名空间中绑定了条目，远程客户端将在 **java:jboss/exported** 之后使用文本查找 **connection-factory**。例如，**RemoteConnectionFactory** 默认绑定到 **java:jboss/exported/jms/RemoteConnectionFactory**，这意味着远程客户端将使用 **jms/RemoteConnectionFactory** 来查找此连接事实。**pooled-connection-factory** 应当不会在 **java:jboss/exported** 命名空间内绑定任何条目，因为 **pooled-connection-factory** 不适合于远程客户端。

8.5.2. 使用 JNDI 连接到服务器

如果客户端驻留在与服务器相同的 JVM 中，它可以使用 **InVmConnectionFactory** 提供的 **in-vm** 连接器。以下是如何配置 **InvmConnectionFactory**，如 **standalone-full.xml** 中找到。

```
<connection-factory
```

```
  name="InVmConnectionFactory"
  entries="java:/ConnectionFactory"
  connectors="in-vm" />
```

注意 **entries** 属性的值。使用 **InVmConnectionFactory** 的客户端应在查找过程中丢弃前导 **java:/**，如下例所示：

```
InitialContext ctx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)ctx.lookup("ConnectionFactory");
Connection connection = cf.createConnection();
```

远程客户端使用 **RemoteConnectionFactory**，它通常配置如下：

```
<connection-factory
  name="RemoteConnectionFactory"
  scheduled-thread-pool-max-size="10"
  entries="java:jboss/exported/jms/RemoteConnectionFactory"
  connectors="http-connector"/>
```

远程客户端应忽略 **条目** 值的前导 **java:jboss/exported/**，如下代码片段示例：

```
final Properties env = new Properties();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
env.put(Context.PROVIDER_URL, "http-remoting://remotehost:8080");
InitialContext remotingCtx = new InitialContext(env);
ConnectionFactory cf = (ConnectionFactory) remotingCtx.lookup("jms/RemoteConnectionFactory");
```

注意 **PROVIDER_URL** 属性的值，以及客户端如何使用 JBoss EAP http-remoing 协议。另请注意客户端如何使用 **org.wildfly.naming.client.WildFlyInitialContextFactory**，这意味着客户端拥有此类及其包含的客户端 JAR 在类路径中的某处。对于 maven 项目，这可以通过包含以下依赖项来实现：

```
<dependencies>
  <dependency>
    <groupId>org.wildfly</groupId>
    <artifactId>wildfly-jms-client-bom</artifactId>
    <type>pom</type>
  </dependency>
</dependencies>
```

8.5.3. 使用 Core API 连接到服务器

您可以使用 Core API 进行客户端连接，而无需 JNDI 查找。使用核心 API 的客户端在其类路径中要求客户端 JAR，就像基于 JNDI 的客户端一样。

ServerLocator
客户端使用 **ServerLocator** 实例来创建 **ClientSessionFactory** 实例。顾名思义，**ServerLocator** 实例用于查找服务器并创建与服务器的连接。

在 Jakarta Messaging 术语中，对 **ServerLocator** 的思考方式与 Jakarta Messaging Connection Factory 相同。

ServerLocator 实例是使用 **ActiveMQClient** 工厂类创建的。

```
ServerLocator locator = ActiveMQClient.createServerLocatorWithoutHA(new
TransportConfiguration(InVMConnectorFactory.class.getName()));
```

ClientSessionFactory
客户端使用 **ClientSessionFactory** 创建 **客户端Session** 实例，这些实例基本上是与服务器的连接。在 Jakarta Messaging 术语中，它们被视为 Jakarta Messaging 连接。

ClientSessionFactory 实例使用 **ServerLocator** 类创建。

```
ClientSessionFactory factory =  locator.createClientSessionFactory();
```

ClientSession
客户端使用客户端 **Session** 来使用和生成消息，并将其分组到交易中。**ClientSession** 实例可以同时支持事务和非事务性语义，还提供 XAResource 接口，使得消息传递操作可以作为 Jakarta Transactions 操作的一部分来执行。

ClientSession Instance group **ClientConsumers** 和 **ClientProducers**.

```
ClientSession session = factory.createSession();
```

以下简单示例重点介绍了刚刚讨论的一些内容：

```
ServerLocator locator = ActiveMQClient.createServerLocatorWithoutHA(
  new TransportConfiguration( InVMConnectorFactory.class.getName()));

// In this simple example, we just use one session for both
// producing and consuming
ClientSessionFactory factory =  locator.createClientSessionFactory();
ClientSession session = factory.createSession();

// A producer is associated with an address ...
ClientProducer producer = session.createProducer("example");
ClientMessage message = session.createMessage(true);
message.getBodyBuffer().writeString("Hello");

// We need a queue attached to the address ...
session.createQueue("example", "example", true);

// And a consumer attached to the queue ...
ClientConsumer consumer = session.createConsumer("example");

// Once we have a queue, we can send the message ...
producer.send(message);

// We need to start the session before we can -receive- messages ...
session.start();
ClientMessage msgReceived = consumer.receive();

System.out.println("message = " + msgReceived.getBodyBuffer().readString());

session.close();
```

8.6. 通过负载均衡器的消息传递

将 JBoss EAP 用作负载平衡器时，客户端可以调用静态 Undertow HTTP 负载平衡器后或 mod_cluster 负载平衡器后面的消息传递服务器。

要支持通过静态负载均衡器调用消息传递服务器的消息传递客户端的配置必须满足以下要求：

- 将 JBoss EAP 用作负载平衡器时，您必须使用 HTTP 或 HTTPS 配置负载平衡器。AJP 不支持消息传递负载平衡器。
  - 有关将 Undertow 配置为静态负载平衡器的详细信息，请参阅《JBoss EAP *配置指南*》中的将 Undertow 配置为静态负载平衡器。
- 如果在负载平衡器后面的消息传递服务器上进行 JNDI 查找，您必须 配置后端消息传递工作程序。
- 连接到负载平衡器的客户端必须重复使用与负载平衡器的初始连接，以确保它们与同一服务器通信。连接到负载均衡器的客户端不能使用集群拓扑连接到负载均衡器。使用集群拓扑可能会导致消息发送到其他服务器，这可能会导致事务处理中断。

有关使用 mod_cluster 将 Undertow 配置为负载平衡器的详细信息，请将 Undertow 配置为使用 mod_cluster 的负载平衡器 。

配置消息客户端以通过负载均衡器进行通信

连接到负载均衡器的客户端必须配置为重复使用初始连接，而不是使用集群拓扑连接到负载均衡器。

重新使用初始连接可确保客户端连接到同一服务器。使用集群拓扑可能会导致消息定向到不同的服务器，这可能会导致事务处理中断。

用于连接负载平衡器的连接工厂或池式连接工厂必须通过属性 **use-topology-for-load-balancing** 设置为 false 来配置。以下示例演示了如何在 CLI 中定义此配置：

```
/subsystem=messaging-activemq/pooled-connection-factory=remote-artemis:write-attribute(name=use-topology-for-load-balancing, value=false)
```

配置后端 worker

只有在负载均衡器后计划进行 JNDI 查找时，您必须配置后端消息传递工作程序。

1. 创建一个新的出站套接字绑定，以指向负载平衡服务器。

   ```
   /socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=balancer-binding:add(host=load_balance.example.com,port=8080)
   ```

2. 创建一个 HTTP 连接器来引用负载平衡服务器套接字绑定。

   ```
   /subsystem=messaging-activemq/server=default/http-connector=balancer-connector:add(socket-binding=balancer-binding, endpoint=http-acceptor)
   ```

3. 将 HTTP 连接器添加到客户端使用的连接工厂。

   ```
   /subsystem=messaging-activemq/server=default/connection-factory=RemoteConnectionFactory:write-attribute(name=connectors,value=[balancer-connector])
   ```

   确保您将客户端配置为重复使用初始连接：

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=use-topology-for-load-
balancing,value=false)
```

第 9 章 配置连接事实

默认情况下，JBoss EAP **messaging-activemq** 子系统提供 **InVmConnectionFactory** 和 **RemoteConnectionFactory** 连接工厂，以及 **activemq-ra 池** 连接工厂。

基本连接事实

InVmConnectionFactory 引用 **in-vm-connector**，并可用于在同一 JVM 中运行客户端和服务器时发送和接收消息。**RemoteConnectionFactory** 引用 **http-connector**，并可用于在不同 JVM 中运行客户端和服务器时通过 HTTP 发送和接收消息。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <connection-factory name="InVmConnectionFactory" connectors="in-vm"
entries="java:/ConnectionFactory"/>
    <connection-factory name="RemoteConnectionFactory" connectors="http-connector"
entries="java:jboss/exported/jms/RemoteConnectionFactory"/>
    ...
  </server>
</subsystem>
```

有关不同类型的连接器的更多信息，请参阅 Acceptors 和 Connectors 部分。

添加连接事实

您可以使用以下管理 CLI 命令添加新连接工厂：在添加连接工厂时，您必须提供 **连接器** 和 JNDI **条目**。

```
/subsystem=messaging-activemq/server=default/connection-
factory=MyConnectionFactory:add(entries=[java:/MyConnectionFactory],connectors=[in-vm])
```

配置连接事实

您可以使用管理 CLI 更新连接工厂的设置。

```
/subsystem=messaging-activemq/server=default/connection-factory=MyConnectionFactory:write-
attribute(name=thread-pool-max-size,value=40)
```

有关连接工厂可用属性的信息，请参阅 Connection Factory Attributes。

删除连接事实

您可以使用管理 CLI 删除连接工厂。

```
/subsystem=messaging-activemq/server=default/connection-factory=MyConnectionFactory:remove
```

池的连接事实

JBoss EAP **messaging-activemq** 子系统提供池式连接工厂，允许您配置集成 ActiveMQ Artemis 资源适配器的入站和出站连接器。有关配置 **pooled-connection-factory** 以连接到远程 ActiveMQ Artemis 服务器的更多信息，请参阅 使用集成资源适配器进行远程连接。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <pooled-connection-factory name="activemq-ra" transaction="xa" entries="java:/JmsXA
```

```
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm"/>
  </server>
</subsystem>
```

池连接工厂有几个独特的特征：

- 虽然它可以配置为指向远程服务器，但只能供本地客户端使用。有关连接到远程 ActiveMQ Artemis 服务器的更多信息，请参阅 使用集成 Artemis 资源适配器进行远程连接。
- 只有在查找 JNDI 或注入时，它才应用于发送消息。
- 它可以配置为使用安全凭据，这在指向安全远程服务器时非常有用。
- 从它获取的资源将自动加入任何正在进行的 Jakarta Transactions。

添加池连**接事**实
您可以使用以下管理 CLI 命令添加新池连接工厂：在添加连接工厂时，您必须提供 **连接器** 和 JNDI **条目**。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=MyPooledConnectionFactory:add(entries=[java:/MyPooledConnectionFactory],connectors=[in-vm])
```

配置池连**接事**实
您可以使用管理 CLI 更新池连接工厂的设置。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=MyPooledConnectionFactory:write-attribute(name=max-retry-interval,value=3000)
```

如需有关池连接工厂可用属性的信息，请参阅池 式连接工厂属性。

您可以通过将 enlistment **-trace** 属性设置为 **false**，使用管理 CLI 禁用此池连接工厂的条目记录。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=MyPooledConnectionFactory:write-attribute(name=enlistment-trace,value=false)
```

警告

禁用加入跟踪会使在事务注册期间跟踪错误变得更加困难。

您还可以配置池化连接工厂使用的受管理连接池实施。如需更多信息，请参阅 JBoss EAP 配置 *指南中的配置管理* 连接池 一节。

删**除池的**连**接事**实
您可以使用管理 CLI 删除池连接工厂。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=MyPooledConnectionFactory:remove
```

第 10 章 配置持久性

10.1. 关于 JBOSS EAP 7 MESSAGING 中的持久性

JBoss EAP 随附两个持久化选项，用于存储绑定数据和消息：

- 您可以使用默认的 基于文件的日志，该日志 针对消息传递用例进行了高度优化，可提供出色的性能。默认情况下会提供这个选项，如果您不进行任何附加配置，则使用此选项。
- 您可以将数据存储在 JDBC 数据存储 中，该数据存储使用 JDBC 连接到您选择的数据库。此选项要求在服务器配置文件中配置 **数据源和 messaging-activemq** 子系统。

10.2. 使用默认文件日志的消息传递日志持久性

JBoss EAP 消息传递附带了高性能、基于文件的日志，针对消息传递进行了优化。

JBoss EAP 消息传递日志具有可配置的文件大小且仅附加，这可通过启用单写操作来提高性能。它由磁盘上的一组文件组成，这些文件最初会预先创建成固定大小并填写 padding。在执行添加消息、删除消息和更新消息等服务器操作时，操作的记录将附加到日志中，直到日志文件已满，此时使用下一个日志文件。

复杂的垃圾收集算法决定了在所有数据都被删除后是否可以回收和重新使用日志文件。紧凑算法从日志文件中删除死空间并压缩数据。

日志还完全支持本地和 XA 事务。

10.2.1. 消息传递日志文件系统实施

大多数日志用 Java 编写，但已提取了与文件系统的交互，以允许不同的可插拔实施。JBoss EAP 消息传递随附两种实施为：

Java 新 I/O(NIO)

此实施使用标准 Java NIO 与文件系统接口。它提供了极其良好的性能，并在具有 Java 6 或更高版本运行时的任何平台上运行。请注意，JBoss EAP 7 需要 Java 8。JBoss EAP 支持的任何操作系统上均支持使用 NIO。

Linux 异步 IO(ASYNCIO)

此实施使用原生代码打包程序与 Linux 异步 IO 库(ASYNCIO)对话。此实施消除了显式同步的需求。ASYNCIO 通常比 Java NIO 提供更好的性能。
要检查使用哪些日志类型，请发出以下 CLI 请求：

```
/subsystem=messaging-activemq/server=default:read-attribute(name=runtime-journal-type)
```

系统返回以下值之一：

表 10.1. 日志类型返回值

返回值	描述
NONE	持久性被禁用
NIO	Java NIO 正在使用

返回值	描述
ASYNCIO	使用 **libaio 的 ASY** NCIO
数据库	使用了 JDBC 持久性

使用 **libaio** 原生时，以下文件系统已经过测试，且仅在 Red Hat Enterprise Linux 6、Red Hat Enterprise Linux 7 和 Red Hat Enterprise Linux 8 中被支持。它们未经测试，在其他操作系统中不受支持。

- EXT4
- XFS
- NFSv4
- GFS2

下表列出了已经测试的 HA 共享存储文件系统，包括 **libaio** 原生和未测试的 HA 共享存储文件系统，以及是否被支持。

操作系统	文件系统	支持使用 libaio 原生选项？ (journal-type="ASYNCIO")	不支持使用 libaio Natives？ (journal-type="NIO")
Red Hat Enterprise Linux 6	NFSv4	是	是
Red Hat Enterprise Linux 7 及更新的版本	NFSv4	是	是
Red Hat Enterprise Linux 6	GFS2	是	否
Red Hat Enterprise Linux 7 及更新的版本	GFS2	是	否

10.2.2. 标准消息日志文件系统实例

标准 JBoss EAP 消息传递核心服务器使用以下日志实例：

绑定日志

此日志用于存储绑定相关的数据，包括服务器上部署的队列集合及其属性。它还存储 id 序列计数器等数据。
绑定日志始终是 NIO 日志，因为它的吞吐量通常低于消息日志。

此日志中的文件前缀为 activemq-bindings。每个文件都有一个绑定扩展名。文件大小为 1048576，位于 bindings 文件夹中。

Jakarta Messaging Journal

此日志实例存储与 Jakarta 消息传递相关的所有数据，如任何 Jakarta 消息传递队列、topics、连接工厂和这些资源的任何 JNDI 绑定。
通过管理 API 创建的任何 Jakarta 消息传递资源都将保留在此日志中。通过配置文件配置的任何资源都不会。只有在使用 Jakarta 消息传递时，才会创建 Jakarta 消息传递日志。

此日志中的文件前缀为 activemq-jms。每个文件都有 a **jms** 扩展名。文件大小为 1048576，位于 bindings 文件夹中。

消息日志

此日志实例存储与所有消息相关的数据，包括消息本身和重复 ID 缓存。
默认情况下，JBoss EAP 消息传递将尝试使用 ASYNCIO 日志。如果 ASYNCIO 不可用，例如，没有安装正确的内核版本或 ASYNCIO 的 Linux，它将自动回退到使用任何 Java 平台上提供的 Java NIO。

此日志中的文件前缀为 activemq-data。每个文件都有一个 amq 扩展名。文件大小默认为 10485760（可配置），它位于 journal 文件夹中。

对于大型消息，JBoss EAP 消息传递将其保留在消息日志之外。这在有关 大型消息 的章节中讨论。

也可以将 JBoss EAP 消息传递配置为在内存不足的情况下将消息分页到磁盘。Paging 部分中将对此进行探讨。

如果根本不需要持久性，JBoss EAP 消息传递也可以配置为完全不持久保留任何数据到存储，如为 Zero Persistence 配置 JBoss EAP 消息传递 一节中所述。

10.2.3. 配置绑定和 Jakarta 消息传递日志

由于绑定日志与 Jakarta 消息传递日志共享其配置，因此您可以使用下面的单一管理 CLI 命令读取这两者的当前配置。也包含 输出以突出显示默认配置。

```
/subsystem=messaging-activemq/server=default/path=bindings-directory:read-resource

{
    "outcome" => "success",
    "result" => {
        "path" => "activemq/bindings",
        "relative-to" => "jboss.server.data.dir"
    }
}
```

请注意，默认情况下日志 **的路径** 为 **activemq/bindings**。您可以使用以下管理 CLI 命令更改 **路径** 位置：

```
/subsystem=messaging-activemq/server=default/path=bindings-directory:write-
attribute(name=path,value=PATH_LOCATION)
```

另请注意上面输出中的 **relative-to** 属性。使用 **relative-to** 时，**path** 属性的值被视为 **relative-to** 指定的文件路径的相对。默认情况下，此值为 JBoss EAP 属性 **jboss.server.data.dir**。对于单机服务器，**jboss.server.data.dir** 位于 ***EAP_HOME*/standalone/data**。对于域，每一服务器自己的 **serverX/data/activemq** 目录位于 ***EAP_HOME*/domain/servers** 下。您可以使用以下管理 CLI 命令更改 **relative-to** 的值：

```
/subsystem=messaging-activemq/server=default/path=bindings-directory:write-
attribute(name=relative-to,value=RELATIVE_LOCATION)
```

默认情况下，JBoss EAP 配置为自动创建绑定目录（如果不存在）。使用以下管理 CLI 命令切换此行为：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=create-bindings-dir,value=TRUE/FALSE)
```

将 **值设** 为 **true** 将启用自动创建目录。将 **值设** 为 **false** 将禁用它。

10.2.4. 配置消息日志位置

您可以使用下面的管理 CLI 命令读取消息日志的位置信息。也包含 输出以突出显示默认配置。

```
/subsystem=messaging-activemq/server=default/path=journal-directory:read-resource
{
    "outcome" => "success",
    "result" => {
        "path" => "activemq/journal",
        "relative-to" => "jboss.server.data.dir"
    }
}
```

请注意，默认情况下，日志 **的路径** 为 **activemq/journal**。您可以使用以下管理 CLI 命令更改 **路径** 位置：

```
/subsystem=messaging-activemq/server=default/path=journal-directory:write-attribute(name=path,value=PATH_LOCATION)
```

注意

为了获得最佳性能，红帽建议日志位于其自身物理卷中，以便最大程度减少磁盘头移动。如果日志位于与其他可能写入其他文件（如绑定日志、数据库或交易协调器）共享的卷上，那么磁盘头在文件写入时可能会快速移动，从而显著降低性能。

另请注意上面输出中的 **relative-to** 属性。使用 **relative-to** 时，**path** 属性的值被视为 **relative-to** 指定的文件路径的相对。默认情况下，此值为 JBoss EAP 属性 **jboss.server.data.dir**。对于单机服务器，**jboss.server.data.dir** 位于 ***EAP_HOME*/standalone/data**。对于域，每一服务器自己的 **serverX/data/activemq** 目录位于 ***EAP_HOME*/domain/servers** 下。您可以使用以下管理 CLI 命令更改 **relative-to** 的值：

```
/subsystem=messaging-activemq/server=default/path=journal-directory:write-attribute(name=relative-to,value=RELATIVE_LOCATION)
```

默认情况下，JBoss EAP 配置为自动创建日志目录（如果不存在）。使用以下管理 CLI 命令切换此行为：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=create-journal-dir,value=TRUE/FALSE)
```

将 **值设** 为 **true** 将启用自动创建目录。将 **值设** 为 **false** 将禁用它。

10.2.5. 配置消息日志属性

下列属性是消息传递服务器的所有子属性。因此，使用管理 CLI 获取和设置值的命令语法分别相同。

要读取给定属性的当前值，其语法如下：

```
/subsystem=messaging-activemq/server=default:read-attribute(name=ATTRIBUTE_NAME)
```

编写属性值的语法遵循对应的模式。

```
/subsystem=messaging-activemq/server=default:write-
attribute(name=ATTRIBUTE_NAME,value=NEW_VALUE)
```

- **create-journal-dir**
  如果设置为 **true**，日志目录将在 **journal-directory** 中指定的位置（如果尚不存在）自动创建。默认值为 **true**。

- **journal-file-open-timeout**
  此属性修改打开日志文件的超时值。默认值为 **5** 秒。

- **journal-buffer-timeout**
  我们不需要刷新每个需要刷新的写入操作，而是维护一个内部缓冲区，而是在缓冲区已满时或超时过期时清空整个缓冲区，以以较快者为准。这可用于 NIO 和 ASYNCIO，允许系统通过许多需要清空的并发写入更好地扩展。

  此参数控制在尚未填充缓冲区时清空缓冲区的超时时间。ASYNCIO 通常可以应对比 NIO 更高的清空率，因此系统在 NIO 和 ASYNCIO 中保持不同的默认值。NIO 的默认值为 **3333333** 纳秒，每秒为 300 倍。ASYNCIO 的默认值为 **500000** 纳秒，每秒为 2000 倍。

  注意

  通过增大超时时间，您可以提高系统吞吐量，但会牺牲延迟，选择默认参数以在吞吐量和延迟之间达到合理的平衡。

- **journal-buffer-size**
  ASYNCIO 上定时缓冲区的大小（以字节为单位）。**journal-buffer-size** 和 **journal-file-size** 必须大于 **min-large-message-size**。否则，消息不会写入日志中。如需更多信息，请参阅配置 大型消息。

- **journal-compact-min-files**
  在考虑压缩日志前，最少的文件数量。只有至少有 **journal-compact-min-files** 才会启动压缩算法。

  将此设置为 **0** 将禁用完全压缩功能。虽然日志可能会无限增长，但这可能会很危险。明智地使用它！

  此参数的默认值为 **10**

- **journal-compact-percentage**
  开始紧凑的阈值。当将少于这个百分比视为实时数据时，我们开始缩减。另请注意，在 **日志中至少有 journal-compact-min-files** 数据文件前，压缩不会启动

  此参数的默认值为 **30**。

- **journal-file-size**
  各个日志文件的大小，以字节为单位。此默认值为 **10485760** 字节，即 10MB。**journal-file-size** 和 **journal-buffer-size** 必须设置为大于 **min-large-message-size**。否则，消息不会写入日志中。如需更多信息，请参阅配置 大型消息。

- **journal-max-io**
  在将写入请求提交到系统以执行之前，写入请求会被排队。此参数控制任意时间上 IO 队列中的最大写入请求数。如果队列已满，则写入将阻止，直到空间被释放。

  系统会保留这个参数的不同默认值，具体取决于它是 NIO 还是 ASYNCIO。NIO 的默认值为 **1**，ASYNCIO 的默认值为 **500**。

  存在一个限制，最大 ASYNCIO 的值不能超过在 OS 级别配置时，在 /proc/sys/fs/aio-max-nr（通常为 **65536** ）中配置的 ASYNCIO。

- **journal-min-files**
  日志将维护的最少文件数。当 JBoss EAP 启动且没有初始消息数据时，JBoss EAP 将预先创建 **journal-min-files** 号文件数。默认值为 **2**。

  创建日志文件并使用 padding 填充它们是一个相当昂贵的操作，我们希望在文件填写时尽可能避免这样做。通过预创建文件，在填写后，日志可以立即恢复下一个日志，而不必暂停创建日志。

  根据您期望队列以稳定状态包含的数据量，您应该调整这一数量的文件以匹配该数据总量。

- **journal-pool-files**
  可重复使用的日志文件数量。虽然在回收文件时，ActiveMQ 将根据需要创建任意数量的文件，但是，在回收文件时，它将回滚到 值。默认值为 **-1**，即没有限制。

- **journal-sync-transactional**
  如果设置为 true，则 JBoss EAP 将确保所有事务数据在事务边界上清空到磁盘，如提交、准备或回滚。默认值为 **true**。

- **journal-sync-non-transactional**
  如果设置为 true，则 JBoss EAP 将确保每次都清空到磁盘，如发送和确认等非事务消息数据。默认值为 **true**。

- **journal-type**
  有效值为 **NIO** 或 **ASYNCIO**。

  选择 **NIO** 可让 JBoss EAP 使用 Java NIO 日志。 **ASYNCIO** 告诉它使用 Linux 异步 IO 日志。如果您选择 **ASYNCIO** 但没有运行 Linux，或者您尚未安装 libaio，那么 JBoss EAP 将使用 Java NIO 日志。

10.2.6. 禁用磁盘写入缓存中的备注

无论您是从操作系统执行 **fsync（）** 还是从 Java 程序内部正确同步数据，都会出现这种情况！

默认情况下，许多系统都启用了磁盘写入缓存。这意味着即使在从操作系统同步之后，仍无法保证数据实际将其传输到磁盘，因此在出现故障时，关键数据可能会丢失。

有些更昂贵的磁盘具有非易失性或电池支持的写入缓存，这些写入缓存不一定会在发生故障时丢失数据，但您需要对其进行测试！

如果您的磁盘没有昂贵的非易失性或受电池支持的缓存，且它不是某种冗余阵列的一部分，例如 RAID，并且您需要确保禁用磁盘写入缓存的数据完整性。

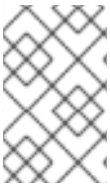
请注意，禁用磁盘写入缓存可让您明智地获取性能。如果您使用的是使用在默认设置中启用了写入缓存的磁盘，则不知道您的数据完整性可能会受到破坏，那么禁用它将使您了解磁盘在真正可靠的操作时能执行的速度。

在 Linux 上，您可以使用 IDE 磁盘的工具 **hdparm 或 sdparm 或 s ginfo** 检查或更改磁盘的写入缓存设置。

在 Windows 上，您可以通过右键单击磁盘并单击 **属性** 来检查和更改设置。

10.2.7. 安装 libaio

Java NIO 日志性能较高，但是如果您使用 Linux 内核 2.6 或更高版本运行 JBoss EAP 消息传递，红帽强烈建议您使用 ASYNCIO 日志，以获取最佳的持久性性能。



注意

JBoss EAP 仅支持在红帽企业 Linux 第 6、7 或 8 版上安装时，且仅在使用 ext4、xfs、gfs2 或 nfs4 文件系统时支持 ASYNCIO。不可能在其他操作系统或早期版本的 Linux 内核下使用 ASYNCIO 日志。

您将需要安装 **libaio**，才能使用 ASYNCIO 日志。要安装，请使用以下命令：

- Red Hat Enterprise Linux 6 和 7:

```
yum install libaio
```

- Red Hat Enterprise Linux 8：

```
dnf install libaio
```

警告

请勿将消息传递日志放在 tmpfs 文件系统上，例如用于 **/tmp** 目录。如果 ASYNCIO 日志使用 tmpfs，JBoss EAP 将无法启动。

10.2.8. 配置 NFS 共享存储以进行消息传递

在使用专用的共享存储、数据复制高可用性时，您必须将实时服务器和备份服务器配置为使用 NFS 客户端上的共享目录。如果您将一台服务器配置为使用 NFS 服务器上的共享目录，另一个服务器使用 NFS 客户端上的共享目录，则备份服务器无法识别实时服务器启动或正在运行时。因此，为了正常工作，两台服务器必须在 NFS 客户端上指定共享目录。

您还必须为 NFS 客户端挂载配置以下选项：

- **sync** ：此选项指定所有更改都会立即刷新到磁盘。
- **intr** ：此选项允许在服务器停机或无法访问时中断 NFS 请求。
- **noac** ：此选项禁用属性缓存，这是在多个客户端之间实现属性缓存一致性所必需的。
- **soft** ：此选项指定服务于导出的文件系统的主机不可用，应报告错误，而不是等待服务器恢复在线。
- **lookupcache=none** ：此选项禁用查找缓存。

- **timeo=*n*** ：NFS 客户端在重试 NFS 请求前等待响应，以毫秒为单位的时间（秒数十）。对于 NFS over TCP，默认的 **timeo** 值为 **600** （60 秒）。对于 NFS over UDP，客户端使用自适应算法为常用请求类型（如读取和写入请求）估算适当的超时值。
- **retrans=*n*** ：NFS 客户端在尝试进一步恢复操作前重试请求的次数。如果没有指定 **retrans** 选项，NFS 客户端会尝试每个请求三次。

重要

在配置 **timeo 和重新传输** 选项时，务必要使用合理的值。默认的 **timeo** 等待时间为 **600** 分钟（60 秒），组合回 **转** 值 **5** 次尝试可等待 ActiveMQ Artemis 检测 NFS 断开连接。

有关如何使用 共享文件系统以实现高可用性的更多信息，请参阅本指南中的共享存储部分。

10.3. 使用 JDBC 数据库的消息传递日志持久性

要使用 JDBC 将消息和绑定数据保存到数据库，而不是使用默认的基于文件的日志，您必须配置 JBoss EAP 7 消息传递。

为此，您必须首先在 **datasource s** 子系统中配置数据源元素，然后在 messaging **- activemq 子系统中的 server 元素上定**义 **journal- datasource** 属性来使用该数据源。存在 **journal-datasource** 属性通知 messaging 子系统将日志条目持久保存到数据库，而非基于文件的日志。**messaging-activemq** 子系统中的 **server** 资源上的 **journal-database** 属性定义用于与数据库通信的 SQL 情况。此属性使用数据源元数据自动配置。

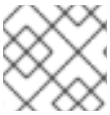
当将消息永久保留到基于文件的日志时，较大的消息大小仅受磁盘大小的限制。但是，当持久保留消息到数据库时，较大的消息大小仅限于该数据库的 **BLOB** 数据类型的最大大小。

重要

JBoss EAP 7.4 目前仅支持 Oracle 12c 和 IBM DB2 Enterprise 数据库。

10.3.1. 配置数据库持久性存储的注意事项

为提高可靠性，JBoss EAP 通过连接池实现消息调用，该池提供一组可在多个应用之间共享的指定数据库的开放连接。这意味着，如果 JBoss EAP 丢弃连接，池中的另一个连接会替换该故障的连接以避免失败。



注意

以前的 JBoss EAP 版本只支持池中的一个连接。

当您在 datasources 子系统中配置数据库持久性存储或池时，请考虑以下点：

- 将 **min-pool-size** 属性的值设置为至少 4，使连接专用于以下每个用法：
  - 一个用于绑定
  - 用于消息日志
  - 一个用于租期锁定，如果使用高可用性(HA)
  - 如果使用 HA，则节点管理器共享状态一个

- 根据执行分页或大型消息流操作的并发线程数量，设置 **max-pool-size** 属性的值。没有定义用于配置 **max-pool-size** 属性的规则，因为线程数量和连接数量之间的关系不是一对一的。

连接数取决于进程分页和大型消息操作的线程数量，以及 **阻止-timeout-wait-millis** 的属性块，用于定义等待连接所需时间。

在专用线程中发生新的大型消息或分页操作，需要连接。这些专用的线程会排队，直到连接就绪或获取连接超时的时间，这会导致失败。

您可以根据需求自定义池配置，并测试环境中配置的池。

10.3.2. 配置消息传递日志 JDBC 持久性存储

按照以下步骤将 JBoss EAP 7 消息传递配置为使用 JDBC 将消息并将数据绑定到数据库：

1. 在 datasources 子系统中配置数据源，供 **messaging-activemq** 子系统使用。有关如何创建和配置数据源的信息，请参阅 JBoss EAP *配置指南中的* 数据源 管理。

2. 配置 **messaging-activemq** 子系统，以使用新的数据源。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=journal-
datasource,value="MessagingOracle12cDS")
```

这会在服务器配置文件的 **messaging-activemq** 子系统中创建以下配置：

```
<server name="default">
  <journal datasource="MessagingOracle12cDS"/>
  ...
</server>
```

JBoss EAP 消息传递现在已配置为使用数据库存储消息传递数据。

10.3.3. 配置消息传递日志表名称

JBoss EAP 7 消息传递使用单独的 JDBC 表来存储绑定信息、消息、大消息和分页信息。这些表的名称可以使用 **服务器** 配置文件 messaging **- activemq 子系统中的 journal -bindings-table、journal-jms - bindings-table、journal -messages -table 和 journal-page- store-table** 属性来配置。

下表列出了表名称限制：

- JBoss EAP 7 消息传递使用模式 *TABLE_NAME + GENERATED_ ID* 为分页表生成标识符，其中 *GENERATED_ID* 最多可包含 20 个字符。由于 Oracle Database 12c 中的最大表名称长度为 30 个字符，因此您必须将表名称限制为 10 个字符。否则，您可能会看到 error **ORA-00972: 标识符太长**，分页将无法工作。

- 不遵循 Oracle Database 12c 的架构对象命名规则 的表名称必须使用双引号括起。带引号的标识符可以任何字符开头，并且可以包含任何字符和标点符号以及空格。但是，带引号和非引号的标识符不能包含双引号或空字符(\0)。务必要注意，带引号的标识符区分大小写。

- 如果多个 JBoss EAP 服务器实例使用相同的数据库来持久保留消息和绑定数据，则表名称对于每个服务器实例都必须是唯一的。多个 JBoss EAP 服务器无法访问相同的表。

以下是管理 CLI 命令的示例，它使用带引号的标识符配置 **journal-page-store-table** 名称：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=journal-page-store-table,value="\"PAGE_DATA\"")
```

这会在服务器配置文件的 **messaging-activemq** 子系统中创建以下配置：

```
<server name="default">
  <journal datasource="MessagingOracle12cDS" journal-page-store-table="&amp;quot;PAGED_DATA&amp;quot;"/>
  ...
</server>
```

10.3.4. 在受管域中配置消息传递日志

如 配置消息传递日志表名称 中所述，在使用 JDBC 持久消息并将数据绑定到数据库时，多个 JBoss EAP 服务器无法访问同一数据库表。在受管域中，服务器组中的所有 JBoss EAP 服务器实例共享同一配置文件配置，因此您必须使用表达式来配置消息传递日志名称或数据源。

如果所有服务器都配置为使用同一个数据库来存储消息传递数据，则表名称必须为每个服务器实例唯一。以下是管理 CLI 命令的示例，它通过使用名称中包含唯一节点标识符 **的表达式为服务器组中的每个服务器创建唯一的 journal-page-store-table** 表名称：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=journal-page-store-table,value="${env.NODE_ID}_page_store")
```

如果每个服务器实例访问不同的数据库，您可以使用表达式来允许各服务器的消息传递配置连接到不同的数据源。以下管理 CLI 命令使用 **connection-url** 中的 **DB_CONNECTION_URL** 环境变量连接到不同的数据源：

```
data-source add --name=messaging-journal --jndi-name=java:jboss/datasources/messaging-journal --driver-name=oracle12c  --connection-url=${env.DB_CONNECTION_URL}
```

10.3.5. 配置消息传递日志网络超时

您可以配置 JDBC 连接将等待数据库回复请求的最长时间，以毫秒为单位。如果网络因任何原因停机或 JBoss EAP 消息传递和数据库之间的连接已关闭，这很有用。发生这种情况时，客户端会被阻止，直到超时发生为止。

您可以通过更新 **journal-jdbc-network-timeout** 属性来配置超时。默认值为 **20000** 毫秒，即 **20** 秒。

以下是管理 CLI 命令的示例，它将 **journal-jdbc-network-timeout** 属性值设置为 **10000** 毫秒，或 **10** 秒：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=journal-jdbc-network-timeout,value=10000)
```

10.3.6. 为消息传递 JDBC Persistence 存储配置 HA

当代理配置了数据库存储类型时，JBoss EAP **messaging-activemq** 子系统将激活 JDBC HA 共享存储功能。然后，代理使用共享数据库表来确保实时和备份服务器通过共享 JDBC 日志存储协调操作。

您可以使用以下属性为 JDBC 持久存储配置 HA：

- **journal-node-manager-store-table** ：JDBC 数据库表的名称，以存储节点管理器。

- **journal-jdbc-lock-expiration** ：JDBC 锁定被视为有效的时间，而不保持其正常运行。以秒为单位指定这个属性值。默认值为 **20** 秒。
- **journal-jdbc-lock-renew-period** ：JDBC 锁定服务的保持活动期间。以秒为单位指定这个属性值。默认值为 **2** 秒。

根据服务器的 **ha-policy 和 journal- datasource** 属性的值，考虑这些默认值。

为了向后兼容，您还可以使用相应的 Artemis 特定的系统属性指定它们的值：

- **brokerconfig.storeConfiguration.nodeManagerStoreTableName**
- **brokerconfig.storeConfiguration.jdbcLockExpirationMillis**
- **brokerconfig.storeConfiguration.jdbcLockRenewPeriodMillis**

配置后，这些 element 特定的系统属性的优先级高于对应的属性的默认值。

10.4. 管理消息传递日志准备的事务

您可以使用以下管理 CLI 命令管理消息传递日志准备的事务：

- 提交准备的事务：

```
/subsystem=messaging-activemq/server=default:commit-prepared-transaction(transaction-as-base-64=XID)
```

- 回滚准备的事务：

```
/subsystem=messaging-activemq/server=default:rollback-prepared-transaction(transaction-as-base-64=XID)
```

- 显示所有准备的事务的详情：

```
/subsystem=messaging-activemq/server=default:list-prepared-transactions
```

注意

您还可以使用 **list-prepared-transaction-details-as-html 操作以 HTML 格式显示准备的事务详情，或者使用 list- prepared-transaction-details-as-json** 操作以 JSON 格式显示。

10.5. 为 ZERO PERSISTENCE 配置 JBOSS EAP 消息传递

在某些情况下，消息传递系统需要零持久性。零持久性意味着不应保留绑定数据、消息数据、大消息数据、重复 ID 缓存或分页数据。

若要配置 **messaging-activemq** 子系统来执行零持久性，请将 **启用 persistence** 的参数设置为 **false**。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=persistence-enabled,value=false)
```

重要

请注意，如果禁用了持久性，但启用了分页，页面文件将继续存储在 **分页元素** 指定的位置。当 **address-full-policy** 属性设置为 **PAGE** 时启用分页。如果需要完全零持久性，请确保配置 **address- setting 元素的 address-full- policy** 属性以使用 **BLOCK**、**DROP** 或 **FAIL**。

10.6. 导入和导出日志数据

有关导入和导出日志数据的信息，请参阅 JBoss EAP 7 迁移指南。

第 11 章 配置分页

11.1. 关于分页

JBoss EAP 消息传递支持许多消息队列，每个队列包含数百万条消息。JBoss EAP 消息传递服务器运行的内存有限，因此很难一次性将所有消息队列存储到内存中。

分页是 JBoss EAP 消息传递服务器用于按需透明地对内存中和内存不足的消息分页的机制，以便在有限内存中适应大型消息队列。

当特定地址的消息大小超过配置的最大消息大小时，JBoss EAP 消息传递开始分页到磁盘。

注意

JBoss EAP 消息传递分页功能默认为启用。

11.2. 页面文件

文件系统中的每个地址都有一个单独的文件夹，可将消息存储在多个文件中。存储消息的文件称为页面文件。每一文件包含的消息最多由 **page-size-bytes** 属性设置的最大配置的消息大小。

系统根据需要导航页面文件，并在页面中的所有消息都由客户端收到后立即删除页面文件。

警告

出于性能考虑，JBoss EAP 消息传递不会扫描分页消息。因此，您应该禁用在配置为分组消息或提供最后一个值的队列上的分页。此外，启用分页的队列中消息优先级和消息选择器的行为不会如预期。您必须禁用分页，才能使这些功能按预期工作

例如，如果使用者具有从队列读取消息的消息选择器，则只会将与选择器匹配的内存中的消息传送给消费者。当使用者确认发送这些消息时，新消息将取消页并加载到内存中。页面文件中可能存在与使用者对磁盘选择器匹配的消息，但 JBoss EAP 消息传递不会将它们加载到内存中，直到另一使用者读取内存中的消息并提供可用空间。如果可用空间不可用，则使用选择器的消费者可能不会收到任何新消息。

11.3. 配置分页目录

您可以使用下面的管理 CLI 命令读取分页目录的配置。在本例中，输出中显示默认配置。

```
/subsystem=messaging-activemq/server=default/path=paging-directory:read-resource
{
    "outcome" => "success",
    "result" => {
        "path" => "activemq/paging",
        "relative-to" => "jboss.server.data.dir"
    }
}
```

分 页目录 配置元素指定要存储页面文件的文件系统中的位置。JBoss EAP 为此分页目录中的每个分页地址创建一个文件夹，页面文件则存储在这些文件夹中。默认情况下，此路径为 **activemq/paging/**。您可以使用以下管理 CLI 命令更改路径位置：

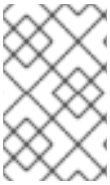
```
/subsystem=messaging-activemq/server=default/path=paging-directory:write-attribute(name=path,value=PATH_LOCATION)
```

另请注意上述示例输出中的 **relative-to** 属性。指定 **relative-to** 时，**path** 属性的值被视为 **relative-to** 属性指定的文件路径的相对。默认情况下，这个值是 JBoss EAP **jboss.server.data.dir** 属性。对于单机服务器，**jboss.server.data.dir** 位于 ***EAP_HOME*/standalone/data/**。对于受管域，每一服务器将拥有自己的 **serverX/data/activemq/** 目录，位于 ***EAP_HOME*/domain/servers/** 下。您可以使用以下管理 CLI 命令更改 **relative-to** 的值：

```
/subsystem=messaging-activemq/server=default/path=paging-directory:write-attribute(name=relative-to,value=RELATIVE_LOCATION)
```

11.4. 配置分页模式

当发送到地址的消息超过配置的大小时，该地址将进入 *分页模式*。



注意

分页为每个地址单独执行。如果您为一个地址配置 **max-size-bytes**，这意味着每个匹配地址将具有您指定的最大大小。但并不意味着所有匹配地址的总大小都仅限于 **max-size-bytes**。

即使使用 页**面** 模式，服务器也可能会因为内存不足而崩溃。JBoss EAP 消息传递保留对磁盘上每个页面文件的引用。在有数百万页面文件的情况下，JBoss EAP 消息传递可能会面临内存耗尽的情况。要最小化此风险，务必要将属性 **page-size-bytes** 设置为适当的值。您必须将 JBoss EAP 消息传递服务器的内存配置为目标数量为 **max-size-bytes** 的两倍，否则可能会出现内存不足错误。

您可以使用以下管理 CLI 命令，为地址读取当前的最大**字节大小（最大字节数）**：

```
/subsystem=messaging-activemq/server=default/address-setting=ADDRESS_SETTING:read-attribute(name=max-size-bytes)
```

您可以使用以下管理 CLI 命令，为地址配置**最大字节大小**（最大字节数）。

```
/subsystem=messaging-activemq/server=default/address-setting=ADDRESS_SETTING:write-attribute(name=max-size-bytes,value=MAX_SIZE)
```

在读取或写入地址设置的其他分页相关属性的值时，请使用类似语法。下表列出了各个属性，以及描述和默认值。

下表描述了地址设置中的参数：

表 11.1. 地址设置的分页配置

元素	描述

元素	描述
address-full-policy	此属性值用于分页决策。有效值如下所列。 **页面** 启用分页和页面信息（超过设定限制）到磁盘。 **DROP** 静默丢弃超过设置限制的消息。 **FAIL** 丢弃消息并向客户端消息制作者发送异常。 **BLOCK** 发送消息超过设定限制时，阻止客户端消息制作者。 默认值为 **PAGE**。
max-size-bytes	这用于指定在进入分页模式前地址可以具有的最大内存大小。默认值为 **10485760**。
page-max-cache-size	系统会将页面文件保持在内存中最多 **page-max-cache-size**，以便在分页导航期间优化输入/输出。默认值为 **5**。
page-size-bytes	这用于指定分页系统中使用的每个页面文件的大小。默认值为 **2097152**。

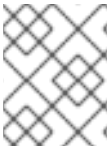
重要

默认情况下，所有地址都在地址到达 **max-size-bytes** 后配置为页面消息。如果您不想在达到最大大小时换页消息，您可以配置地址来丢弃消息，丢弃消息（客户端一侧除外）或阻止生产者通过分别将 **address-full-policy** 设置为 **DROP**、**FAIL** 和 **BLOCK** 来发送其他消息。

请注意，如果在任何目的地启动到页面消息后，将 **address-full-policy** 从 **PAGE** 更改为 **BLOCK**，则使用者将不再能够使用分页消息。

使用多个队列的地址

当消息路由到绑定有多个队列的地址时，内存中仅有一个消息副本。每个队列仅处理对该消息的原始副本的引用，因此仅当引用原始消息的所有队列都发送消息时，内存才会被释放。



注意

单个 lazy 队列/订阅可以降低整个地址的输入/输出性能，因为所有队列都将通过分页系统上的额外存储发送消息。

第 12 章 使用大消息

JBoss EAP 消息传递支持大型消息，即使客户端或服务器的内存有限。大消息可以按照原样流传输，也可以进一步压缩，以实现更高效的传输。用户可以通过在邮件正文中设置 **InputStream** 来发送大消息。发送消息时，JBoss EAP 消息传递将读取此 **输入流，** 并将数据传输到片段中的服务器。

客户端或服务器均未将大消息的完整正文存储在内存中。使用者最初收到一条包含空正文的大消息，随后在邮件上设置一个 **OutputStream**，以将它流传输到磁盘文件。

警告

在处理大型消息时，服务器不会像消息正文一样处理消息属性。例如，某个属性设为大于 **journal-buffer-size** 的字符串的消息无法被服务器处理，因为它会过填充日志缓冲区。

12.1. 流传输大消息

如果您以标准方式发送大信息，发送它们所需的堆大小可以是消息大小的四倍或多倍，这意味着 1 GB 的消息可能需要 4 GB 的堆内存中。因此，JBoss EAP 消息传递支持使用 java.io. InputStream 和 **java.io**.OutputStream 类设置消息正文，它们需要的内存要少得多。输入流直接用于发送消息，输出流用于接收消息。

在接收信息时，可以通过两种方式处理输出流：

- 您可以使用 **ClientMessage.saveToOutputStream(OutputStream out)** 方法阻断输出流。
- 您可以使用 **ClientMessage.setOutputstream(OutputStream out)** 方法异步将消息写入流。此方法要求使用者保持活动状态，直到消息完全收到。

您可以使用您喜欢的任何流，例如文件 JDBC Blobs 或 SocketInputStream，只要它实施 **java.io.InputStream 来** 发送消息，而 **java.io.OutputStream** 用于接收消息。

使用核心 API 流传输大型消息

下表显示了使用对象属性通过 Jakarta Messaging 提供的 **ClientMessage** 类中的方法。

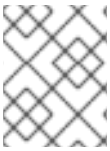
ClientMessage 方法	描述	Jakarta 消息传递同等属性
setBodyInputStream(InputStream)	设置 **用于在邮件发送时读取邮件正文的输入流。**	**JMS_AMQ_InputStream**
setOutputStream(OutputStream)	设置 **将接收邮件正文的输出流。** 这个方法不会阻止。	**JMS_AMQ_OutputStream**
saveOutputStream(OutputStream)	将邮件的正文保存到 **输出流。** 它将阻止整个内容传输到 **输出流。**	**JMS_AMQ_SaveStream**

以下代码示例在接收核心消息时设置输出流。

```
ClientMessage firstMessage = consumer.receive(...);

// Block until the stream is transferred
firstMessage.saveOutputStream(firstOutputStream);

ClientMessage secondMessage = consumer.receive(...);

// Do not wait for the transfer to finish
secondMessage.setOutputStream(secondOutputStream);
```

以下代码示例在发送内核信息时设置输入流：

```
ClientMessage clientMessage = session.createMessage();
clientMessage.setInputStream(dataInputStream);
```



注意

对于大于 2GiB 的消息，您必须使用 **_AMQ_LARGE_SIZE** 消息属性。这是因为 **getBodySize（）** 方法将返回无效值，因为它仅限于最大整数值。

流处理 Jakarta 消息传递的大型消息

使用 Jakarta Messaging 时，JBoss EAP 消息传递通过设置对象属性来映射核心 API 流方式。您可以使用 **Message.setObjectProperty(String name, Object value)** 方法来设置输入和输出流。

在发送的消息上，**使用 JMS_AMQ_InputStream 属性来设置 Input** Stream。

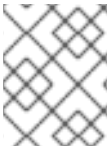
```
BytesMessage bytesMessage = session.createBytesMessage();
FileInputStream fileInputStream = new FileInputStream(fileInput);
BufferedInputStream bufferedInput = new BufferedInputStream(fileInputStream);
bytesMessage.setObjectProperty("JMS_AMQ_InputStream", bufferedInput);
someProducer.send(bytesMessage);
```

OutputStream 使用 **JMS_AMQ_SaveStream** 属性来设置，对以阻止方式接收的消息使用 JMS_AMQ_SaveStream 属性。

```
BytesMessage messageReceived = (BytesMessage) messageConsumer.receive(120000);
File outputFile = new File("huge_message_received.dat");
FileOutputStream fileOutputStream = new FileOutputStream(outputFile);
BufferedOutputStream bufferedOutput = new BufferedOutputStream(fileOutputStream);

// This will block until the entire content is saved on disk
messageReceived.setObjectProperty("JMS_AMQ_SaveStream", bufferedOutput);
```

也可以通过使用 **JMS_AMQ_Output Stream 属性以非阻塞方式设置 Output Stream**。

```
// This does not wait for the stream to finish. You must keep the consumer active.
messageReceived.setObjectProperty("JMS_AMQ_OutputStream", bufferedOutput);
```



注意

使用 Jakarta Messaging 流传输大型消息时，仅支持 **StreamMessage** 和 **BytesMessage** 对象。

12.2. 配置大消息

12.2.1. 配置大型消息位置

您可以使用下面的管理 CLI 命令读取大消息目录的配置。也包含 输出以突出显示默认配置。

```
/subsystem=messaging-activemq/server=default/path=large-messages-directory:read-resource
{
    "outcome" => "success",
    "result" => {
        "path" => "activemq/largemessages",
        "relative-to" => "jboss.server.data.dir"
    }
}
```

重要

为获得最佳性能，建议将大型消息目录存储在来自消息日志或分页目录的不同物理卷上。

large-messages-directory 配置元素用于指定在文件系统中存储大型消息的位置。请注意，默认情况下路径为 **activemq/largemessages**。您可以使用以下管理 CLI 命令更改路径位置：

```
/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-
attribute(name=path,value=PATH_LOCATION)
```

另请注意上面输出中的 **relative-to** 属性。使用 **relative-to** 时，path 属性的值被视为 **relative-to** 指定的文件路径的相对。默认情况下，此值为 JBoss EAP 属性 **jboss.server.data.dir**。对于单机服务器，**jboss.server.data.dir** 位于 ***EAP_HOME*/standalone/data**。对于域，每一服务器自己的 **serverX/data/activemq** 目录位于 ***EAP_HOME*/domain/servers** 下。您可以使用以下管理 CLI 命令更改 **relative-to** 的值：

```
/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-
attribute(name=relative-to,value=RELATIVE_LOCATION)
```

配置大型消息大小
使用管理 CLI 查看大型消息的当前配置。请注意，此配置是 **connection-factory** 元素的一部分。例如，要读取包含的默认 **RemoteConnectionFactory** 的当前配置，请使用以下命令：

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:read-attribute(name=min-large-message-size)
```

使用类似语法设置 属性。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=min-large-message-
size,value=NEW_MIN_SIZE)
```

注意

属性 **min-large-message-size** 的值应当以字节为单位。

配置大型消息压缩
您可以选择压缩大型消息，实现快速、高效的传输。所有压缩/解压缩操作都在客户端端处理。如果压缩的消息小于 **min-large-message 大小**，它将作为常规消息发送到服务器。使用管理 CLI，将 boolean 属性 **compress-large-messages** 设置为 **true**，以压缩大型消息。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=compress-large-messages,value=true)
```

12.2.2. 使用核心 API 配置大型消息大小

如果您在客户端使用核心 API，则需要使用 **setMinLargeMessageSize** 方法来指定大型信息的最小大小。默认情况下，大消息(**min-large-message-size**)的最小大小设置为 100KB。

```
ServerLocator locator = ActiveMQClient.createServerLocatorWithoutHA(new
TransportConfiguration(InVMConnectorFactory.class.getName()))

locator.setMinLargeMessageSize(25 * 1024);

ClientSessionFactory factory = ActiveMQClient.createClientSessionFactory();
```

第 13 章 计划消息

您可以尽早为发送消息指定时间。这可以通过在发送消息前设置 **_AMQ_SCHED_DELIVERY** 调度发送属性来完成。

指定的值必须是正 长，对应于消息被发送的时间（毫秒）。以下是使用 Jakarta Messaging API 发送调度消息的示例。

```
// Create a message to be delivered in 5 seconds
TextMessage message = session.createTextMessage("This is a scheduled message message that will be delivered in 5 sec.");
message.setLongProperty("_AMQ_SCHED_DELIVERY", System.currentTimeMillis() + 5000);
producer.send(message);

...

// The message will not be received immediately, but 5 seconds later
TextMessage messageReceived = (TextMessage) consumer.receive();
```

也可以使用核心 API 发送计划消息，在发送消息前设置 **_AMQ_SCHED_DELIVERY** 属性。

第 14 章 临时队列和运行时队列

在设计一个请求reply 模式时，客户端发送请求并等待回复时，您必须考虑客户端的每个运行时实例是为其回复需要一个专用队列，或者运行时实例是否可以访问共享队列，根据适当的属性选择其特定的回复消息。

如果需要多个队列，客户端需要能够动态创建队列。Jakarta Messaging 利用临时队列的概念提供此设施。**Session** 根据要求创建一个 **TemporaryQueue**。它在连接的生命周期内存在，例如，连接关闭或临时队列被删除为止。这意味着，尽管临时队列由特定的会话创建，但它可以被从同一连接创建的任何其他会话重复利用。

使用共享队列和回复个别临时队列之间的权衡受活动客户端实例数量的影响。采用共享队列方法时，在某些特定于提供商的阈值上，对队列访问的争用可能会成为问题。这必须与提供商在运行时创建队列存储相关的额外开销以及托管大量临时队列的计算机内存影响进行对比。

以下示例在启动时为每个客户端创建一个临时队列和使用者。它将每条消息上的 **JMSReplyTo** 属性设置为临时队列，然后在每条消息上设置关联 ID，以将请求消息与响应消息相关联。这可避免为每个请求创建和关闭消费者的开销，而这非常昂贵。相同的生产者和消费者可以在多个线程之间共享或共用。在消费者下次访问队列时，收到的任何消息（在会话终止时尚未确认）都会保留并重新传送。

示例：临时队**列代**码

```
...
// Create a temporary queue, one per client
Destination temporaryQueue = session.createTemporaryQueue();
MessageConsumer responseConsumer = session.createConsumer(temporaryQueue);

// This class handles messages to the temporary queue
responseConsumer.setMessageListener(this);

// Create the message to send
TextMessage textMessage = session.createTextMessage();
textMessage.setText("My new message!");

// Set the reply to field and correlation ID
textMessage.setJMSReplyTo(temporaryQueue);
textMessage.setJMSCorrelationID(myCorrelationID);

producer.send(textMessage);
...
```

类似地，使用 **Session.createTemporaryTopic（）** 方法创建临时主题。

第 15 章 FILTER EXPRESSIONS AND MESSAGE SELECTORS

JBoss EAP 中的 **messaging-activemq** 子系统基于 SQL 92 表达式语法的子集提供了强大的过滤器语言。

它与用于 Jakarta 消息传递选择器的语法相同，但预定义的标识符是不同的。有关 Jakarta 消息传递选择器语法的文档，请参考 javax.jms.Message 接口 Javadoc。

filter 属性可以在配置中的多个位置找到。

- 预定义队列.在预先定义队列时，可以为其定义过滤器表达式。只有与过滤器表达式匹配的消息才会进入队列。以下配置片段显示了包括过滤器的队列定义：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  ...
  <queue
    name="myQueue"
    filter="FILTER_EXPRESSION"
    ...
  />
  ...
</subsystem>
```

  要在管理 CLI 中使用选择器创建队列，请使用以下命令：

```
jms-queue add --queue-address=QUEUE_ADDRESS --selector=FILTER_EXPRESSION
```

- 核心网桥可以通过可选过滤器表达式定义，只有匹配的消息才会桥接。以下是来自示例配置文件的代码片段，其中 **messaging-activemq** 子系统包含具有过滤器的网桥。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  ...
  <bridge
    name="myBridge"
    filter="FILTER_EXPRESSION"
    ...
  />
  ...
</subsystem>
```

- 可以通过可选的过滤器表达式来定义转换，只有匹配的消息才会被移出。如需更多信息，请参阅 Diverts。以下示例片段显示了使用过滤器的 Divert：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  ...
  <divert
    name="myDivert"
    filter="FILTER_EXPRESSION"
    ...
  />
  ...
</subsystem>
```

Jakarta 消息传递选择器表达式和 JBoss EAP 消息传递核心过滤器表达式之间存在一些区别。Jakarta 消息传递选择器表达式在 Jakarta 消息传递消息上运行，而 JBoss EAP 消息传递核心过滤器表达式则在核心消息上运行。

在核心过滤器表达式中可以使用以下标识符来指代核心消息的属性：

- **AMQPriority**.来引用消息的优先级。消息优先级是具有来自 **0** 到 **9** 的有效值的整数。**0** 是最低优先级，**9** 是最高优先级。例如，A **MQPriority** = 3 AND isolated = 'aardvark'。
- **AMQ 扩展.**参考邮件的过期时间：该值是一个长整数。
- **AMQDurable**。参考消息是否持久。该值是一个具有有效值的字符串：DUR **ABLE** 或 **NON_DURABLE**。
- **AMQTimestamp**.创建消息时的时间戳。该值是一个长整数。
- **AMQSize** ：消息的大小，以字节为单位。该值是一个整数。

核心过滤器表达式中使用的任何其他标识符将被假定为消息的属性。

第 16 章 配置消息过期

如果已发送的消息在指定时间后未传送给消费者，则可在服务器上将其设置为在服务器上过期。这些过期的消息可在以后使用，以进行进一步检查。

使用 Core API 设置消息过期

使用核心 API，您可以使用 **setExpiration** 方法在消息上设置过期时间。

```
// The message will expire 5 seconds from now
message.setExpiration(System.currentTimeMillis() + 5000);
```

使用 Jakarta Messaging 设置消息过期

您可以将发送消息时使用的 Jakarta Messaging **MessageProducer** 的时间设置为 live。您可以使用 **setTimeToLive** 方法以毫秒为单位指定这个值。

```
// Messages sent by this producer will be retained for 5 seconds before expiring
producer.setTimeToLive(5000);
```

您还可以通过设置生产者的 **发送** 方法的生存时间，根据每个消息指定消息到期时间。

```
// The last parameter of the send method is the time to live, in milliseconds
producer.send(message, DeliveryMode.PERSISTENT, 0, 5000)
```

从到期地址使用的消息具有以下属性：

- **_AMQ_ORIG_ADDRESS**
  包含已过期消息的原始地址 **的字符串** 属性。
- **_AMQ_ACTUAL_EXPIRY**
  包含过期消息实际到期时间的 **Long** 属性。

16.1. 到期地址

您可以通过设置到期地址来指定发送过期消息的位置。如果消息过期且未指定到期地址，则消息将从队列中删除并丢弃。

您可以使用管理 CLI 为 **address -setting 设置 expiration-address**。在以下示例中，**jms.queue.exampleQueue** 队列中的过期消息将发送到 **jms.queue.expiryQueue** 到期地址。

```
/subsystem=messaging-activemq/server=default/address-setting=jms.queue.exampleQueue:write-attribute(name=expiry-address,value=jms.queue.expiryQueue)
```

16.2. 到期 REAPER 线程

恢复器线程定期检查队列，以检查消息是否已过期。您可以使用管理 CLI 为回收器线程设置扫描周期和线程优先级。

设置到期恢复器线程的扫描周期，也就是以毫秒为单位扫描队列的频率，以检测过期的消息。默认值为 **30000**。您可以将其设置为 **-1**，以禁用回收器线程。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=message-expiry-scan-period,value=30000)
```

设置到期的线程优先级。可能的值从 **0** 到 **9**，其中 **9** 是最高优先级。默认值为 **3**。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=message-expiry-thread-priority,value=3)
```

第 17 章 配置转发的 REDELIVERY

将延迟重新传送到地址通过 address **- setting 配置元素的 redelivery- delay** 属性来定义。如果指定了重新传送延迟，JBoss EAP 将等待这一延迟的持续时间，然后再重新传送消息。如果将 **redelivery-delay** 设为 **0**，则不会重新传送延迟。若要为给定 **address-setting 获取 redelivery-delay** 的当前值，可使用以下管理 CLI 命令作为示例：

```
/subsystem=messaging-activemq/server=default/address-
setting=YOUR_ADDRESS_SETTING:read-attribute(name=redelivery-delay)
```

下表列出了 **address-setting** 的配置属性，可用于配置消息重新传送：以下列管理 CLI 命令为例，设置给定属性的值：

```
/subsystem=messaging-activemq/server=default/address-
setting=YOUR_ADDRESS_SETTING:write-attribute(name=ATTRIBUTE,value=NEW_VALUE)
```

表 17.1. 地址设置的相关属性

属性	描述
max-delivery-attempts	定义在发送到 **死信地址** 之前可以重新传送取消的消息的时间。默认值为 **10**。
max-redelivery-delay	以毫秒为单位 **redelivery-delay** 的最大值。您可以设置 **max-redelivery-delay** 参数，以防止延迟太大。默认值为 **redelivery-delay * 10**。
redelivery-delay	定义在尝试重新传送已取消消息前，以毫秒为单位等待多长时间。默认为 **0**。
redelivery-multiplier	应用到 **redelivery-delay** 参数的倍数。每次重新传送消息时，延迟时间将等同于之前的 **redelivery-delay** * **redelivery-multiplier**。默认值为 **1.0**。

有关配置 **address-setting** 的详细信息，请参阅**地址设置**。

第 18 章 配置 DEAD LETTER 地址

messaging-activemq 子系统配置的 **address-setting** 元素中定义了死信地址。若要读取给定 **address-setting** 的当前配置，可使用以下管理 CLI 命令作为示例：

```
/subsystem=messaging-activemq/server=default/address-setting=ADDRESS_SETTING:read-attribute(name=dead-letter-address)
```

如果没有指定 **dead-letter-address**，会在尝试提供 **max-delivery-attempts** 时间后删除消息。默认情况下，会尝试发送消息 10 次。将 **max-delivery-attempts** 设置为 **-1** 可进行无限重新传送尝试。以下示例管理 CLI 命令演示了如何为给定 **address -setting 设置 dead-letter -address 和 max-delivery-attempts** 属性。

```
/subsystem=messaging-activemq/server=default/address-setting=ADDRESS_SETTING:write-attribute(name=dead-letter-address,value=NEW_VALUE)

/subsystem=messaging-activemq/server=default/address-setting=ADDRESS_SETTING:write-attribute(name=max-delivery-attempts,value=NEW_VALUE)
```

例如，可以为一组匹配地址设置一个死信，您可以为特定地址设置 **max-delivery-attempts** 设为 **-1**，以便只针对这个地址进行无限重新传送尝试。地址通配符也可用于为一组地址配置死信设置。

有关创建和配置 **address-setting** 的详细信息，请参阅**地址设置**。

第 19 章 流控制

流程控制可用于限制客户端和服务器之间的消息传递数据流，从而使消息传递参与者不会感到不堪重负。您可以从消费方和制作者一侧管理数据流。

19.1. 消费者流控制

JBoss EAP 消息传递包括的配置，用于定义预先为使用者获取的数据量，以及控制使用者能够使用消息的速度。

基于窗口的流控制

JBoss EAP 消息预先填充到每位消费者的缓冲区中。缓冲区的大小由 connection- **factory 的 consumer-window- size** 属性决定。以下示例配置显示了一个明确设置了 **consumer- window-size 属性的 connection- factory**。

```
<connection-factory name="MyConnFactory" ... consumer-window-size="1048576" />
```

使用管理 CLI，读取和写入给定 **connection -factory 的 consumer-window- size** 属性的值。以下示例演示了如何使用 **InVmConnectionFactory** 连接工厂完成此操作，对于驻留在同一虚拟机（例如本地 **MessageDrivenBean** ）的消费者而言，这是默认设置。

- 从管理 CLI 阅读 **InVmConnectionFactory** 的 **consumer-window-size** 属性

```
/subsystem=messaging-activemq/server=default/connection-factory=InVmConnectionFactory:read-attribute(name=consumer-window-size)
{
    "outcome" => "success",
    "result" => 1048576
}
```

- 从管理 CLI 编写 **consumer-window-size** 属性

```
/subsystem=messaging-activemq/server=default/connection-factory=InVmConnectionFactory:write-attribute(name=consumer-window-size,value=1048576)
{"outcome" => "success"}
```

consumer-window-size 的值必须是整数。下表中指出一些值具有特殊含义。

表 19.1. consumer-window-size 的值

值	描述
n	用于将缓冲区大小设置为 **n** 字节的整数值。默认值为 **1048576**，这在大部分情况下都是正常的。如果默认值不够，基准测试将帮助您找到窗口大小的最佳值。
0	关闭缓冲区。这有助于适应较慢的消费者，并能跨多个消费者确定分布情况。
-1	创建未绑定缓冲区。这有助于非常快速的消费者在收到消息时尽快调取和处理消息。

警告

如果使用者无法在接收消息时尽快处理消息，则将 **consumer-window-size** 设置为 **-1** 可溢出客户端内存。

如果您使用核心 API，则使用者窗口大小可以使用其 **setConsumerWindowSize（）** 方法从 **ServerLocator** 设置。

如果使用 Jakarta Messaging，客户端可以使用实例化 **ConnectionFactory** 的 **setConsumerWindowSize（）** 方法指定消费者窗口大小。

速率限制流控制

JBoss EAP 消息传递可以规范每秒使用的消息速率，这种流控制方法称为节流。使用适当 **connection-factory 的 consumer-max- rate** 属性，确保使用者不会以比指定的速度更快地使用消息。

```
<connection-factory name="MyConnFactory" ... consumer-max-rate="10" />
```

默认值为 **-1，它会** 禁用速率限制流控制。

管理 CLI 是读取和写入 **consumer-max-rate** 属性的建议方法。以下示例演示了如何使用 **InVmConnectionFactory** 连接工厂完成此操作，这对于驻留在同一虚拟机（例如本地 **MessageDrivenBean** ）的消费者而言是默认设置。

- 使用管理 CLI 读取 **consumer-max-rate** 属性

```
/subsystem=messaging-activemq/server=default/connection-factory=InVmConnectionFactory:read-attribute(name=consumer-max-rate)
{
    "outcome" => "success",
    "result" => -1
}
```

- 使用管理 CLI 编写 **consumer-max-rate** 属性：

```
/subsystem=messaging-activemq/server=default/connection-factory=InVmConnectionFactory:write-attribute(name=consumer-max-rate,value=100)
{"outcome" => "success"}
```

如果您使用 Jakarta Messaging，可使用实例化 **ConnectionFactory** 的 setConsumerMaxRate(int consumerMaxRate)方法设置最大速率大小。

如果您使用 Core API，可使用 **ServerLocator.setConsumerMaxRate(int consumerMaxRate)** 方法设置速率。

19.2. 制作者流控制

JBoss EAP 消息传递还可以限制从客户端发送的数据量，以防止服务器接收过多的消息。

基于窗口的流控制

JBoss EAP 消息传递利用信用交换规范消息制作者。只要有足够的学分，生产者就可以发送消息到地址。

发送邮件所需的信用金额由邮件大小决定。由于生产者的学分较低，他们必须从服务器请求更多。在服务器配置中，生产者一次请求的信用量称为 **producer-window-size**，这是 **connection-factory** 元素的属性：

```
<connection-factory name="MyConnFactory" ... producer-window-size="1048576" />
```

窗口大小决定了任意时间可以内亮的字节数量，从而防止远程连接过载服务器。

使用管理 CLI 读取和写入给定连接工厂的 **producer-window-size** 属性。以下示例使用 **RemoteConnectionFactory**，它包含在默认配置中并供远程客户端使用。

- 使用管理 CLI 读取 **producer-window-size** 属性：

```
subsystem=messaging-activemq/server=default/connection-factory=RemoteConnectionFactory:read-attribute(name=producer-window-size)
{
    "outcome" => "success",
    "result" => 65536
}
```

- 使用管理 CLI 编写 **producer-window-size** 属性：

```
/subsystem=messaging-activemq/server=default/connection-factory=RemoteConnectionFactory:write-attribute(name=producer-window-size,value=65536)
{"outcome" => "success"}
```

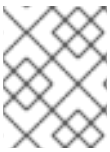
如果使用 Jakarta Messaging，客户端可以调用 **ConnectionFactory** 的 **setProducerWindowSize(int producerWindowSize)** 方法以直接设置窗口大小。

如果使用 core API，可以使用 **ServerLocator** 的 **setProducerWindowSize(int producerWindowSize)** 方法设置窗口大小。

阻塞基于 Producer Window 的流控制

通常，消息传递服务器始终提供与请求相同的分数。但是，可以限制服务器发送的信用量，这可以防止因为生产者发送超过一次可处理的消息而耗尽内存。

例如，如果您有一个名为 **myqueue** 的 Jakarta 消息队列，并且您将最大内存大小设置为 10MB，服务器将规定队列中的消息数量，使其大小永不会超过 10MB。当地址变满时，生产者将在客户端阻止，直到地址上有足够的空间可用。



注意

阻止制作者流控制是分页的一种替代方法，它不会阻止生产者，而是将信息页到存储。如需更多信息，请参阅 关于分页。

address-setting 配置元素包含用于管理阻止制作者流控制的配置。**address-setting** 用于将一组配置应用到该地址注册的所有队列。有关如何进行此操作的更多信息，请参阅配置地址设置。

对于需要阻止制作者流控制的每个 **address-setting**，您必须包含 **max-size-bytes** 属性的值。绑定到该地址的所有队列的总内存不能超过 **max-size-bytes**。对于 Jakarta Messaging 主题，这意味着主题中所有订阅的总内存不能超过 **最大大小。**

您还必须将 **address-full-policy** 属性设置为 **BLOCK**，以便服务器知道如果达到 **max-size-bytes，** 则应阻止制作者。以下是设置了这两个属性 **的 address-setting** 示例：

-

```
<address-setting ...
    name="myqueue"
    address-full-policy="BLOCK"
    max-size-bytes="100000" />
```

上例会将 Jakarta Messaging 队列"myqueue"的最大大小设置为 **100000** 字节。达到最大大小后，生产者将被阻止发送到该地址。

使用管理 CLI 设置这些属性，如下例所示：

- 为指定 **address -setting** 设置 **max-size- bytes**

```
/subsystem=messaging-activemq/server=default/address-setting=myqueue:write-
attribute(name=max-size-bytes,value=100000)
{"outcome" => "success"}
```

- 为指定的 **address- setting** 设置 **address-full-policy**

```
/subsystem=messaging-activemq/server=default/address-setting=myqueue:write-
attribute(name=address-full-policy,value=BLOCK)
{"outcome" => "success"}
```

速率限制流控制
如果您为它所使用的 **connection-factory** 指定制作者/最大值，请限制生产者 每秒可发送的消息数量，如下例所示：

```
<connection-factory name="MyConnFactory" producer-max-rate="1000" />
```

默认值为 **-1，它会** 禁用速率限制流控制。

使用管理 CLI 读取和写入 **producer-max-rate** 的值。以下示例使用 **RemoteConnectionFactory**，它包含在默认配置中并供远程客户端使用。

- 阅读 **producer-max-rate** 属性的值：

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:read-attribute(name=producer-max-rate)
{
    "outcome" => "success",
    "result" => -1
}
```

- 编写 **producer-max-rate** 属性的值：

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=producer-max-rate,value=100)
{"outcome" => "success"}
```

如果使用 core API，请使用方法 **ServerLocator.setProducerMaxRate(int producerMaxRate)** 来设置速率。

如果您使用 JNDI 来实例化和查找连接工厂，则可以使用实例化连接工厂的 **setProducerMaxRate(int producerMaxRate)** 方法在客户端上设置最大速率。

第 20 章 配置预确认

Jakarta Messaging 指定三种确认模式：

- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE

在某些情况下，您可以在失败时丢失消息，因此在将消息发送到客户端之前确认服务器上的消息会有意义。JBoss EAP 消息支持这种额外模式，称为预确认模式。

在传输前在服务器上预确认的缺点是，如果服务器的系统在确认消息后崩溃，但在将消息传送到客户端之前，该消息将会丢失。在这种情况下，消息将丢失，系统重启时将不会恢复。

根据您的消息传递情况，预先确认模式可以避免额外的网络流量和 CPU 使用量，代价是处理消息丢失。

预认证的一个用例是股价更新消息。通过这些消息，在崩溃时丢失消息可能是合理的，因为下一个价格更新消息将很快到达，从而覆盖了先前的价格。

注意

如果您使用预确认模式，您将丢失被使用消息的事务语义，因为它们是首先在服务器上被确认，而不是在您提交事务时确认。

20.1. 配置服务器

可以使用管理 CLI 将预确认 **属性设置为 true，从而将连接工厂配置为使用预** 确认模式：

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=pre-acknowledge,value=true)
```

20.2. 配置客户端

预确认模式可以在客户端的 JNDI 上下文环境中配置，例如在 **jndi.properties** 文件中：

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connection.ConnectionFactory=tcp://localhost:8080?preAcknowledge=true
```

另外，若要使用使用 Jakarta 消息传递 API 的预确认模式，请创建一个具有 **ActiveMQSession.PRE_ACKNOWLEDGE** 常量的 Jakarta 消息会话。

```
// messages will be acknowledge on the server *before* being delivered to the client
Session session = connection.createSession(false, ActiveMQJMSConstants.PRE_ACKNOWLEDGE);
```

第 21 章 拦截器

JBoss EAP 消息传递支持拦截器来拦截进入和退出服务器的数据包。对于分别进入或退出服务器的每个数据包，会调用传入和传出拦截器。这允许执行自定义代码，如用于审核或过滤数据包。拦截器可以修改它们拦截的数据包。这使得拦截器功能强大，但也具有危险性。

21.1. 实施 INTERCEPTORS

拦截器必须实现拦截器接口：

```
package org.apache.artemis.activemq.api.core.interceptor;

public interface Interceptor
{
   boolean intercept(Packet packet, RemotingConnection connection) throws ActiveMQException;
}
```

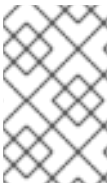
返回的布尔值很重要：

- 如果返回 true，进程通常会继续
- 如果返回 false，则进程将被中止，不会调用其他拦截器，服务器也不会进一步处理数据包。

拦截器类应作为模块添加到 JBoss EAP 中。如需更多信息，请参阅 JBoss EAP *配置指南* 中的 创建自定义模块。

21.2. 配置 INTERCEPTORS

在将模块添加到 JBoss EAP 作为自定义模块后，传入和传出拦截器都通过管理 CLI 添加到消息传递子系统配置中。



注意

您必须 *仅以管理员* 模式启动 JBoss EAP，然后才能接受新的拦截器配置。有关详细信息，请参阅 JBoss EAP *配置指南* 中的仅管理模式运行 JBoss EAP。处理新配置后，以正常模式重新启动服务器。

每个拦截器都应该根据以下示例管理 CLI 命令添加。示例假定每个拦截器已作为自定义模块添加到 JBoss EAP 中。

```
/subsystem=messaging-activemq/server=default:list-add(name=incoming-interceptors, value={name
=> "foo.bar.MyIncomingInterceptor", module=>"foo.bar.interceptors"})
```

添加传出拦截器遵循类似语法，如下例所示：

```
/subsystem=messaging-activemq/server=default:list-add(name=outgoing-interceptors, value={name
=> "foo.bar.MyOutgoingInterceptor", module=>"foo.bar.interceptors"})
```

第 22 章 消息分组

邮件组是一组共享特定特征的消息：

- 消息组中的所有消息都分组到一个通用组 ID 下。这意味着，可以通过通用的组属性来识别它们。
- 消息组中的所有消息都由同一消费者序列处理和使用，无论队列中的客户数量如何。这意味着，当消费者打开时，具有唯一组 id 的特定消息组始终由一位消费者处理。如果使用者关闭该消息组，则整个消息组将定向到队列中的另一消费者。

当需要单一使用者序列处理具有特定值的 属性（如组 ID）的消息时，消息组特别有用。

重要

如果队列启用了分页，消息分组将无法正常工作。务必先 禁用分页， 然后为消息分组配置队列。

有关在消息传递服务器群集中配置 消息分组的详情，请参考 配置多消息传递系统第 III 部分中的 集群消息分组。

22.1. 使用核心 API 配置消息组

属性 **_AMQ_GROUP_ID** 用于使用客户端端的核心 API 标识消息组。要选择随机的唯一消息组标识符，您可以在 **SessionFactory** 上将 **auto-group** 属性设置为 **true**。

22.2. 使用 JAKARTA MESSAGING 配置消息组

属性 **JMSXGroupID** 用于识别 Jakarta 消息客户端的消息组。如果您要向一个消费者发送具有不同消息的消息组，您可以为不同的消息设置相同的 **JMSXGroupID**。

```
Message message = ...
message.setStringProperty("JMSXGroupID", "Group-0");
producer.send(message);

message = ...
message.setStringProperty("JMSXGroupID", "Group-0");
producer.send(message);
```

另一种方法是使用下列 **connection-factory** 属性之一供客户端使用： **auto-group** 或 **group-id**。

当 **auto-group** 设为 **true** 时， **connection-factory** 将开始对通过它发送的所有消息使用随机唯一的消息组标识符。您可以使用管理 CLI 来设置 **auto-group** 属性。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=auto-group,value=true)
```

group-id 属性会将属性 **JMSXGroupID** 设置为通过连接工厂发送的所有消息的指定值。要在连接工厂上设置特定的 **group-id**，请使用管理 CLI。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=group-id,value="Group-0")
```

第 23 章 转变

转换是 JBoss EAP 消息中配置的对象，有助于将消息从一个地址转移到另一个地址。转换可归类为以下类型：

专用

邮件仅转用到新地址，从未发送到旧地址。

非独占性

将向旧地址发送邮件，并且也会将邮件的副本发送到新地址。非独占移动可用于分割消息流。

转换只会将邮件转移到同一服务器上的地址。如果您要转用其他服务器上的地址，常见的模式是转为使用本地存储和转发队列，然后设置一个从该队列消耗并转发到不同服务器上的地址的网桥。

因此，转义是一个非常复杂的概念。与网桥结合使用时，转换可用于创建有趣而复杂的路由。可以将其视为消息的路由表类型。通过将转换与网桥相结合，您可以在多个地理分布式服务器之间创建可靠路由连接的分布式网络，从而创建全局消息传递网格。有关如何使用 网桥的更多信息，请参阅配置 核心网桥。

可以对转换进行配置，以应用 Transformer **和** 可选消息过滤器。可选的消息过滤器有助于仅引发与指定过滤器匹配的消息。有关过滤器的更多信息，请参阅 Filter Expressions 和 Message Selectors 。

转换器用于将消息转换为另一种形式。当指定转换器时，所有转换的消息都会由 Transformer 进行转换。所有转换器都必须实施 **org.apache.activemq.artemis.core.server.cluster.Transformer** 接口：

```
package org.apache.activemq.artemis.core.server.cluster;
import org.apache.activemq.artemis.core.server.ServerMessage;

public interface Transformer {
   ServerMessage transform(ServerMessage message);
}
```

若要使 JBoss EAP 消息传递能够实例化您的转换器实施实例，您必须将它包含在 JBoss EAP 模块中，然后将该模块作为导出的依赖关系添加到 **org.apache.activemq.artemis** 模块。如需有关如何 创建自定义模块 的信息，请参阅 JBoss EAP *配置指南中* 的自定义模块。要将依赖项添加到 **org.apache.activemq.artemis** 模块，打开文件 ***EAP_HOME*/modules/system/layers/base/org/apache/activemq/artemis/main/module.xml**，并将您的 **<module>** 添加到 **<dependencies>** 列表中，如下例所示。

```
<module xmlns="urn:jboss:module:1.3" name="org.apache.activemq.artemis">
   <resources>
     ...
   </resources>
   <dependencies>
     ...
     <module name="YOUR_MODULE_NAME" export="true"/>
   </dependencies>
</module>
```

23.1. 独占转义

以下是在配置中可能出现排他性的问题示例：

```
<divert
  name="prices-divert"
```

```
  address="jms.topic.priceUpdates"
  forwarding-address="jms.queue.priceForwarding"
  filter="office='New York'"
  transformer-class-
name="org.apache.activemq.artemis.jms.example.AddForwardingTimeTransformer"
  exclusive="true"/>
```

在本例中，配置了一个名为 price **-divert** 的转换，它将任何发送到 address **jms.topic. priceUpdates** 的消息映射到名为 priceUpdates 的 Jakarta Messaging 主题，它对应于名为 **priceUpdates** 的本地 Jakarta Messaging 队列，它对应于名为 **priceForwarding 的本地 Jakarta Messaging**队列。

我们还指定消息过滤器字符串，以便仅 丢弃 值为 **New York** 的消息。所有其他消息将继续路由到正常地址。若未指定，过滤器字符串为可选，则所有消息都将被视为匹配项。

请注意，指定了转换器类。在这个示例中，转换器只是添加一个标头来记录发生移动的时间。

此示例实际将消息转移到本地存储和转发队列，该队列配置了将消息转发到另一服务器上的地址的网桥。如需了解更多详细信息，请参阅配置 核心网桥。

23.2. 非独占性转变

以下是非排他性转变的示例。非排他性转移的配置方式与使用可选的过滤器和转换器独占式转换法相同。

```
<divert
  name="order-divert"
  address="jms.queue.orders"
  forwarding-address="jms.topic.spytopic"
  exclusive="false"/>
```

以上转换采用发送到 address **jms.queue.orders** 的每个消息的副本，该副本映射到名为 订单 的 Jakarta 消息传递队列，并将其发送到名为 **jms.topic 的本地地址。SpyTopic** 对应于名为 **SpyTopic** 的 Jakarta Messaging 主题。

创建转变

使用管理 CLI 创建您想要的转变类型：

```
/subsystem=messaging-activemq/server=default/divert=my-divert:add(divert-
address=news.in,forwarding-address=news.forward)
```

默认情况下会创建非排他性转换。要创建专用转变，请使用 **exclusive** 属性：

```
/subsystem=messaging-activemq/server=default/divert=my-exclusive-divert:add(divert-
address=news.in,forwarding-address=news.forward,exclusive=true)
```

下表捕获了转换的属性及其描述。您可以使用以下命令显示管理 CLI：

```
/subsystem=messaging-activemq/server=default/divert=*:read-resource-description()
```

属性	描述
divert-address	地址，从.必需。

属性	描述
exclusive	转用是否排他性，意味着消息被转移到新地址上，而根本不使用旧地址。默认值为 false。
filter	可选的过滤器字符串。如果指定，则仅会引发与过滤器表达式匹配的消息。
forwarding-address	地址转向.必需。
routing-name	转变的路由名称.
transformer-class-name	用于转换邮件正文或属性的类的名称，然后再转换邮件的正文或属性。

第 24 章 线程管理

每个 JBoss EAP 消息传递服务器维护一个线程池供一般使用，并且计划线程池以供计划使用。Java 计划的线程池不能配置为使用标准线程池，否则我们可以将单个线程池用于已调度和非计划的活动。

请注意，JBoss EAP 使用的是全新的非阻塞 NIO。默认情况下，JBoss EAP 消息传递使用的线程数等于内核数或超线程数的三倍，如 **.getRuntime（）.availableProcessors（）报告的，用于处理** 传入的数据包。要覆盖这个值，请在传输配置中指定 the **nio-remoting-threads** 参数来设置线程数。如需更多信息，请参阅配置消息传递传输。

24.1. 服务器调度线程池

服务器调度线程池用于需要定期运行或有延迟的服务器端的大部分活动。它在内部映射到 java.util.concurrent.ScheduledThreadPoolExecutor 实例。

此池使用的最大线程数通过使用 **scheduled-thread-pool-max-size 参数来** 配置。默认值为 5 个线程。通常，少量线程足以满足这个池的需要。要为默认的 JBoss EAP 消息传递服务器更改此值，请使用以下命令：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=scheduled-thread-pool-max-size,value=10)
```

24.2. 服务器通用线程池

线程池通常用于服务器端的大部分异步操作。它在内部映射到 **java.util.concurrent.ThreadPoolExecutor** 实例。

此池使用的最大线程数通过使用 **thread-pool-max-size** 属性来配置。

如果将 **thread-pool-max-size** 设为 **-1**，则线程池没有上限，并在没有足够的线程来满足请求时按需创建新的线程。如果稍后的活动细分，则线程超时并关闭。

如果将 **thread-pool-max-size** 设置为大于零的正整数，则线程池将被绑定。如果请求进入并且池中没有可用线程，请求将停止，直到线程可用为止。建议谨慎使用有界线程池，因为如果上限配置过低，可能会导致死锁情况。

thread-pool-max-size 的默认值为 **30**。若要为默认的 JBoss EAP 消息传递服务器设置一个新值，可使用以下管理 CLI 命令：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=thread-pool-max-size,value=40)
```

如需有关未绑定(cached)和绑定（固定）线程池的更多信息，请参阅 **ThreadPoolExecutor** Javadoc。

24.3. 监控服务器线程使用

检查线程利用率，以确保您已正确配置线程池的大小。

要检查线程使用率，请发出以下 CLI 命令：

```
/subsystem=messaging-activemq:read-resource(include-runtime)
```

系统返回的结果类似如下

```
{
    "outcome" => "success",
    "result" => {
        "global-client-scheduled-thread-pool-active-count" => 0,
        "global-client-scheduled-thread-pool-completed-task-count" => 0L,
        "global-client-scheduled-thread-pool-current-thread-count" => 0,
        "global-client-scheduled-thread-pool-keepalive-time" => 10000000L,
        "global-client-scheduled-thread-pool-largest-thread-count" => 0,
        "global-client-scheduled-thread-pool-max-size" => undefined,
        "global-client-scheduled-thread-pool-task-count" => 0L,
        "global-client-thread-pool-active-count" => 0,
        "global-client-thread-pool-completed-task-count" => 2L,
        "global-client-thread-pool-current-thread-count" => 2,
        "global-client-thread-pool-keepalive-time" => 60000000000L,
        "global-client-thread-pool-largest-thread-count" => 2,
        "global-client-thread-pool-max-size" => undefined,
        "global-client-thread-pool-task-count" => 2L,
        "connection-factory" => undefined,
        "connector" => undefined,
        "discovery-group" => undefined,
        "external-jms-queue" => undefined,
        "external-jms-topic" => undefined,
        "http-connector" => undefined,
        "in-vm-connector" => undefined,
        "jms-bridge" => undefined,
        "pooled-connection-factory" => undefined,
        "remote-connector" => undefined,
        "server" => {"default" => undefined}
    }
}
```

表 24.1. 线程利用率数据

使用率属性	描述
global-client-scheduled-thread-pool-active-count	当前执行任务的所有 ActiveMQ 客户端所使用的调度池线程数量。
global-client-scheduled-thread-pool-completed-task-count	服务器引导以来，所有 ActiveMQ 客户端已执行的已调度池线程的任务数量。
global-client-scheduled-thread-pool-current-thread-count	调度池中当前供所有 Active MQ 客户端使用的线程数。
global-client-scheduled-thread-pool-keepalive-time	当空闲时，在调度线程池中保持线程运行的时间长度。
global-client-scheduled-thread-pool-largest-thread-count	已为所有 Active MQ 客户端同时使用的调度池中最多的线程数。
global-client-scheduled-thread-pool-max-size	调度池中供此服务器内运行的所有 ActiveMQ 客户端使用的最多线程数。

使用率属性	描述
global-client-scheduled-thread-pool-task-count	已计划线程池中由所有 Active MQ 客户端调度的任务总数。
global-client-thread-pool-active-count	当前执行任务的所有 ActiveMQ 客户端使用的通用池线程数量。
global-client-thread-pool-completed-task-count	使用所有 ActiveMQ 客户端已执行的通用池线程的任务数量。
global-client-thread-pool-current-thread-count	通用池中当前供所有 Active MQ 客户端使用的线程数。
global-client-thread-pool-keepalive-time	当空闲时，在通用线程池中应保持运行的时间。
global-client-thread-pool-largest-thread-count	通用池中最大数量的线程，这些线程过去已由所有 Active MQ 客户端同时使用。
global-client-thread-pool-max-size	此服务器内运行的所有 ActiveMQ 客户端使用的通用池中的最大线程数。
global-client-thread-pool-task-count	所有 Active MQ 客户端调度的通用线程池中的任务总数。

24.4. 到期 REAPER 线程

在服务器端也使用一个线程扫描队列中过期的消息。我们无法为此使用任一线程池，因为此线程需要以自己的可配置优先级运行。

24.5. 异步 IO

异步 IO 有一个线程池，用于从原生层接收和分配事件。它位于带有前缀 **ArtemisMQ-AIO-poller-pool** 的线程转储中。JBoss EAP 消息传递在日志中为每个打开的文件使用一个线程（通常为一个线程）。

还有一个线程用于在 libaio 上调用写入。其目的在于避免在 libaio 上切换会导致性能问题。此线程可在带有前缀 **ArtemisMQ-AIO-writer-pool** 的线程转储中找到。

24.6. 客户端线程管理

JBoss EAP 包括用于创建客户端连接的客户端线程池。这个池与本章前面提到的静态池分开，在 JBoss EAP 的行为类似于客户端时使用它。例如，客户端线程池客户端作为与同一集群中其他节点的群集连接创建，或者当 Artemis 资源适配器连接到集成于 JBoss EAP 远程实例的远程 Apache ActiveMQ Artemis 消息传递服务器时。还有一个用于调度的客户端线程的池。

注意

随着 JBoss EAP 7.1 的发布，客户端线程现在会在 60 秒后超时，没有活动。

使用管理 CLI 设置客户端线程池大小

使用管理 CLI 配置客户端线程池和客户端调度线程池的大小。使用管理 CLI 设置的池大小优先于使用系统属性设置的大小。

以下命令设置客户端线程池。

```
/subsystem=messaging-activemq:write-attribute(name=global-client-thread-pool-max-size,value=POOL_SIZE)
```

此属性没有定义默认值。如果未定义 属性，则池的最大大小确定为 CPU 内核处理器的八(8)倍。

若要查看当前的设置，可使用以下命令：

```
/subsystem=messaging-activemq:read-attribute(name=global-client-thread-pool-max-size)
```

使用以下命令设置客户端计划线程池大小：

```
/subsystem=messaging-activemq:write-attribute(name=global-client-scheduled-thread-pool-max-size,value=POOL_SIZE)
```

以下将显示客户端计划线程池的当前池大小：默认值为 **5**。

```
/subsystem=messaging-activemq:read-attribute(name=global-client-scheduled-thread-pool-max-size)
```

使用系统属性设置客户端线程池大小

以下系统属性可用于分别设置客户端的全局和全局调度线程池的大小：

- **activemq.artemis.client.global.thread.pool.max.size**
- **activemq.artemis.client.global.scheduled.thread.pool.core.size**

然后，可以在 XML 配置中引用系统属性，如下例所示。

注意

使用管理 CLI 设置的池大小将优先于系统属性所设置的大小。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <global-client thread-pool-max-size="${activemq.artemis.client.global.thread.pool.max.size}"
    scheduled-thread-pool-max-size="${activemq.artemis.client.global.scheduled.thread.pool.core.size}" />
  <server ...>
  </server>
  ...
</subsystem>
```

配置客户端以使用其拥有的线程池

客户端可以配置其每个 **ClientSessionFactory** 实例，使其不使用 JBoss EAP 提供的池，而是使用自己的客户端线程池。从该 **客户端SessionFactory** 创建的任何会话都将使用新创建的池。

要将 **ClientSessionFactory** 实例配置为使用其自己的池，请在创建工厂后立即调用适当的 setter 方法。例如：

```
ServerLocator locator = ActiveMQClient.createServerLocatorWithoutHA(transportConfiguration);
locator.setUseGlobalPools(true);
locator.setThreadPoolMaxSize(-1);
locator.setScheduledThreadPoolMaxSize(10);
ClientSessionFactory myFactory = locator.createSessionFactory();
```

如果使用 Jakarta Messaging API，您可以在 **ClientSessionFactory** 上设置相同的参数。例如：

```
ActiveMQConnectionFactory myConnectionFactory =
ActiveMQJMSClient.createConnectionFactory(url, "ConnectionFactoryName");
myConnectionFactory.setUseGlobalPools(false);
myConnectionFactory.setScheduledThreadPoolMaxSize(10);
myConnectionFactory.setThreadPoolMaxSize(-1);
```

如果您使用 JNDI 来实例化 **ActiveMQConnectionFactory** 实例，您也可以使用管理 CLI 来设置这些参数，如下例所示，适用于 JBoss EAP 单机实例。

```
/subsystem=messaging-activemq/server=default/connection-factory=myConnectionFactory:write-
attribute(name=use-global-pools,value=false)

/subsystem=messaging-activemq/server=default/connection-factory=myConnectionFactory:write-
attribute(name=scheduled-thread-pool-max-size,value=10)

/subsystem=messaging-activemq/server=default/connection-factory=myConnectionFactory:write-
attribute(name=thread-pool-max-size,value=1)
```

请注意，管理 CLI 将提醒您在执行上述每个命令后需要重新加载实例。

第 25 章 配置重复数据删除消息检测

当发送者向另一服务器发送消息时，目标服务器或连接在发送消息后会失败，但在向发件人发送响应之前，指示进程成功之前。在这些情况下，发件人很难确定消息是否已成功发送到预期的接收器。如果发送方决定重新发送最后一个邮件，则可能会导致向地址发送重复的消息。

您可以在 JBoss EAP 消息传递中配置重复的消息检测，以便您的应用无需提供过滤重复消息的逻辑。

25.1. 将重复数据删除消息检测用于发送消息

要为发送的消息启用重复的消息检测，您需要将 **org.apache.activemq.artemis.api.core.Message.HDR_DUPLICATE_DETECTION_ID** 属性的值解析为 **_AMQ_DUPL_ID** 属性。当目标服务器收到消息时，如果设置了 **_AMQ_DUPL_ID** 属性，它将检查其内存缓存，以查看它是否收到了具有该标头值的消息。如果存在，则忽略此消息。如需更多信息，请参阅配置重复 ID 缓存。

如果您使用核心 API，**_AMQ_DUPL_ID** 属性的值可以是 **字节[]** 或 **SimpleString**。如果您使用的是 Jakarta Messaging，它必须是一个 **String**。

以下示例演示了如何为核心 API 设置 属性。

```
SimpleString myUniqueID = "This is my unique id";  // Can use a UUID for this

ClientMessage message = session.createMessage(true);
message.setStringProperty(HDR_DUPLICATE_DETECTION_ID, myUniqueID);
```

以下示例演示了如何为 Jakarta 消息传递客户端设置 属性。

```
String myUniqueID = "This is my unique id";  // Can use a UUID for this

Message jmsMessage = session.createMessage();
message.setStringProperty(HDR_DUPLICATE_DETECTION_ID.toString(), myUniqueID);
```

重要

当使用 **HDR_DUPLICATE_DETECTION_ID** 属性在同一事务中发送重复消息时，*不会* 检测到重复消息。

25.2. 配置重复数据删除 ID 缓存

服务器维护发送到每个地址的 **_AMQ_DUPL_ID** 属性的接收值缓存。每个地址维护自己的地址缓存。

缓存的大小是固定的。缓存的最大大小使用 **id-cache-size** 属性来配置。此参数的默认值为 **20000** 元素。如果缓存的最大值为 **n** 个元素，则存储的(**n + 1)th**ID 将覆盖缓存中的元素 **0**。该值使用以下管理 CLI 命令设置：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=id-cache-size,value=SIZE)
```

缓存也可以配置为永久保留到磁盘。这可以通过使用以下命令设置 **persist-id-cache** 属性来配置：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=persist-id-cache,value=true)
```

如果此值设为 **true**，则每个 ID 在收到时都会保留为永久存储。此参数的默认值为 **true**。

注意

将重复 ID 缓存的大小设置为较大的大小，以确保消息重新发送不会覆盖之前存储在缓存中的消息。

第 26 章 处理 LOW CONSUMERS

具有服务器端队列较慢的使用者可能会对服务器性能造成严重问题。随着消息在消费者的服务器端队列中不断积累，内存使用量会增加。因此，服务器可能会进入分页模式，这可能会对性能造成负面影响。另一个重要问题是，如果 **consumer-windows-size** 属性大于 **0**，则发送到客户端消息缓冲区的消息可以继续等待被使用。可以设置条件，以便不会足够快速地使用消息的使用者可以与服务器断开连接。

在速度较慢的消费者是 MDB 时，JBoss EAP 服务器管理连接。MDB 断开连接后，消息将返回到使用 MDB 的队列，MDB 自动重新连接，此时，消息将再次负载平衡到队列上的所有 MDB。此过程适用于侦听持久主题的 MDB。对于 Jakarta 消息使用者，如果速度慢，则断开连接，并且只有在重新连接 **-attempts** 设为 **-1 或大于 0 时才会重新** 连接。

如果是不可升级的 Jakarta 消息传递订阅者，或者带有不可持久性订阅的 MDB，连接将断开连接，从而导致订阅被删除。如果不可达目的订阅者重新连接，则会创建一个新的不可持久性订阅，它开始只消耗发送到该主题的新消息。

确定消费者是否慢慢检查特定消费者已确认的消息数量的计算。它不考虑是否在消费者上启用了流控制，或者消费者是否在流传输大型消息，例如：在配置缓慢的使用者检测时请记住这一点。

使用调度的线程池执行慢速消费者检查。服务器上启用了消费者检测缓慢的队列将在内部 **java.util.concurrent.ScheduledThreadPoolExecutor** 实例中产生一个新条目。如果队列数量较多，并且 **慢消费-检查-period** 相对较低，那么执行某些检查时可能会延迟。但是，这不会影响检测算法使用的计算准确性。有关这个 线程池的详情，请参阅线程管理。

每个 **地址设置** 较慢的消费者处理速度较慢。有关配置 **address-setting 的更多信息，请参阅地址设置，** 并参阅附录中的 **address-setting** 属性 列表。有三个属性可用于配置较慢使用者的处理：它们是：

slow-consumer-check-period

慢速消费者的检查频率（秒数）。默认值为 **5**。

slow-consumer-policy

决定识别慢速消费者时发生的情况。有效选项为 **KILL** 或 **NOTIFY** ：

- **KILL** 将终止消费者的连接，这将影响使用同一连接的任何客户端线程。
- **NOTIFY** 将向客户端发送 **CONSUMER_SLOW** 管理通知。

默认值为 **NOTIFY**。

slow-consumer-threshold

在消费者被视为缓慢之前允许的最小消息消耗率。默认值为 **-1**，它没有绑定。

使用管理 CLI 来读取任何属性的当前值：例如，使用以下命令来读取 **address-setting 的当前 slow-consumer- policy**，名称为 **myAddress** ：

```
/subsystem=messaging-activemq/server=default/address-setting=myAddress:read-attribute(name=slow-consumer-policy)
```

同样，使用以下示例来设置相同的 **slow-consumer-policy** ：

```
/subsystem=messaging-activemq/server=default/address-setting=myAddress:write-attribute(name=slow-consumer-policy,value=<NEW_VALUE>)
```

部分 III. 配置多节点消息传递系统

第 27 章 配置 JAKARTA 消息传递网桥

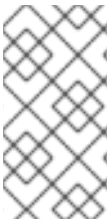
JBoss EAP 消息传递包含 Jakarta 消息传递网桥，该网桥从源目的地获取消息，并将它们发送到目标目的地，通常是在不同服务器上。

Jakarta 消息传递网桥支持目标映射，其中每个链接都由以下定义的源和目标组成：

- 该源定义 Jakarta 消息传递网桥从中接收消息的目的地。源由连接工厂组成，用于创建与 Jakarta 消息传递提供程序的连接，以及该提供程序中的消息源目的地。
- 目标定义 Jakarta 消息传递网桥将消息从源发送到的目的地。目标由连接工厂组成，用于创建与 Jakarta 消息传递提供程序的连接，以及该提供程序中的消息目标目的地。

如果源目的地是一个主题，Jakarta Messaging 网桥会为其创建订阅。如果为 Jakarta Messaging 网桥配置了 **client -id 和 subscription-name** 属性，则订阅具有持久性。这意味着，如果 Jakarta 消息传递网桥停止然后重新启动，不会遗漏任何消息。

源和目标服务器不必位于同一集群中，这使得桥接适合从一个集群将消息可靠发送到另一个集群，例如跨 WAN 以及连接不可靠的地方。



注意

不要将 Jakarta 消息传递网桥与核心网桥混淆。Jakarta 消息传递网桥可用于桥接任何与 Jakarta Messaging-1.1 兼容的提供程序，并使用 Jakarta Messaging API。配置核心网桥 用于桥接任意两个 JBoss EAP 消息传递实例，并使用核心 API。在可能的情况下，使用核心网桥而不是 Jakarta 消息传递网桥。

JBoss EAP Jakarta Messaging 网桥配置示例

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
  ...
  </server>
  <jms-bridge name="my-jms-bridge" module="org.apache.activemq.artemis" max-batch-time="100"
max-batch-size="10" max-retries="1" failure-retry-interval="500" quality-of-
service="AT_MOST_ONCE">
    <source destination="jms/queue/InQueue" connection-factory="ConnectionFactory">
      <source-context/>
    </source>
    <target destination="jms/queue/OutQueue" connection-factory="jms/RemoteConnectionFactory">
      <target-context>
        <property name="java.naming.factory.initial"
value="org.wildfly.naming.client.WildFlyInitialContextFactory"/>
        <property name="java.naming.provider.url" value="http-remoting://192.168.40.1:8080"/>
      </target-context>
    </target>
  </jms-bridge>
  ...
</subsystem>
```

在上例配置中，Jakarta Messaging 网桥使用 **connection-factory** 属性来创建以下两个连接：

- source-context，用于定义收到的消息的原始目的地。
- target-context，用于定义接收消息的目标目的地。

您可以使用 Apache ActiveMQ Artemis 或 Red Hat AMQ 提供的默认实施，通过使用 Java 命名和目录接口(JNDI)搜索连接工厂。对于其他应用服务器或 Jakarta 消息传递提供程序，您可以通过实施接口 **org.apache.activemq.artemis.jms.bridge.ConnectionFactoryFactory** 来提供新的实施。

使用管理 CLI 添加 Jakarta 消息传递网桥

可以使用以下管理 CLI 命令来添加 Jakarta 消息传递网桥：请注意，源和目标目的地必须在配置中定义。有关可配置属性的完整列表，请参见 附录中的表。

```
/subsystem=messaging-activemq/jms-bridge=my-jms-bridge:add(quality-of-
service=AT_MOST_ONCE,module=org.apache.activemq.artemis,failure-retry-interval=500,max-
retries=1,max-batch-size=10,max-batch-time=100,source-connection-
factory=ConnectionFactory,source-destination=jms/queue/InQueue,source-context={},target-
connection-factory=jms/RemoteConnectionFactory,target-destination=jms/queue/OutQueue,target-
context=
{java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory,java.naming.provider.u
=http-remoting://192.168.40.1:8080})
```

您可以在管理 CLI 中使用 **read-resource 命令**检查 Jakarta Messaging 网桥的配置，如下例所示。

```
/subsystem=messaging-activemq/jms-bridge=my-jms-bridge:read-resource
```

使用 **write-attribute** 将配置添加到 Jakarta 消息传递网桥，如本例中所示：

```
/subsystem=messaging-activemq/jms-bridge=my-jms-bridge:write-
attribute(name=ATTRIBUTE,value=VALUE)
```

使用管理控制台添加 Jakarta 消息传递网桥

您还可以按照下列步骤使用管理控制台添加 Jakarta 消息传递网桥。

1. 在浏览器中打开管理控制台，再导航到 **Configuration → Subsystems → Messaging(ActiveMQ) → JMS Bridge**。
2. 单击添加(+)按钮，然后在出现提示时提供所需的信息。
3. 完成后，单击 **Add**。

27.1. 服务质量

在 JBoss EAP **中，服务质量是一种** 可配置的属性，决定了消息的使用和确认方式。**服务质量及其** 描述的有效值如下所示。有关 Jakarta Messaging 网桥属性的完整列表，请参见 附录中的表。

AT_MOST_ONCE

消息将在最多一次从来源到达目的地。消息从来源使用，并在发送到目的地之前确认。因此，如果在从源中删除消息到到达目的地之间发生故障，可能会丢失消息。这个模式是默认值。
此模式适用于持久和非持久消息。

DUPLICATES_OK

消息从源使用，然后在成功发送到目的地后进行确认。因此，如果在发送初始消息之后但在确认之前发生错误，则有可能再次发送消息。
此模式适用于持久和非持久消息。

ONCE_AND_ONLY_ONCE

消息将从源一次且仅一次到达目的地。如果源和目标在同一服务器实例上，可以通过发送和确认同一本地事务中的消息来实现。如果来源和目标位于不同的服务器上，则可通过在 Jakarta Transactions 发起发送和消费会话来实现。事务由交易管理器控制，需要使用网桥上的 **setTransactionManager（）** 方法设置。
此模式仅适用于持久消息。

警告

在关闭部署了 Jakarta 消息传递网桥的服务器 时，**确保首先使用 Jakarta 消息传递网桥将 服务器关闭，以避免意外错误。**

通过使用 DUPLICATES_OK 模式而不是 ONCE_AND_ ONLY_ONCE，然后检查目的地是否有重复项并丢弃它们，可以提供一次且仅一次的语义。如需更多信息，请参阅配置重复消息检测。但是，缓存仅在一定时间内有效。因此，这种方法不再像使用 ONCE_AND_ONLY_ONCE 一样好，但根据您的特定应用需求，这可能是一个不错的选择。

27.2. 超时和 JAKARTA 消息传递网桥

目标或源服务器可能在特定时间点上不可用。如果出现这种情况，网桥将尝试重新连接等于 max-retries 值的多次。尝试之间的等待通过 failure-retry-interval 设置。

警告

因为每个 Jakarta Messaging 网桥都有自己的 max-retries 参数，所以您应该使用一个没有设置 reconnect -attempts 参数的连接工厂，或者将其设置为 0。这将避免潜在的冲突，进而可能导致较长的重新连接时间。另请注意，Jakarta 消息传递网桥引用的任何连接工厂都需要 将服务质量设置为 ONCE_AND_ONLY_ONCE，将 工厂类型 设置为 XA_GENERIC、XA_TOPIC 或 XA_QUEUE。

27.3. 解决 REMOTECONNECTIONFACTORY 异常

在配置 Jakarta 消息传递消息网桥时，您可能会获得 RemoteConnectionFactory 异常。要解决这个问题，请执行以下步骤之一或所有步骤：

- 如果需要，添加 URL 前缀 java:/jboss/exported，取决于您的来自 Java 2 平台 Enterprise Edition (J2EE)组件的客户端连接，一个非J2EE 组件或远程组件。

- 为您的连接工厂使用两个 Java 命名和目录接口名称，如下所示：

```
<jms-connection-factories>
 <connection-factory name="RemoteConnectionFactory">
  <connectors>
    <connector-ref connector-name="netty"/>
  </connectors>
  <entries>
    <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/> <entry
name="jms/RemoteConnectionFactory"/>
  </entries>
 </connection-factory>
</jms-connection-factories>
```

- 在 spring bean 配置文件中使用以下代码片段：

```
<jee:jndi-lookup id="jmsConnectionFactory" jndi-name="java:/ConnectionFactory" expected-
type="javax.jms.ConnectionFactory" lookup-on-startup="false" />
```

注意

lookup-on-startup 和 jndi-name 属性的值可能会根据应用程序更改。

第 28 章 配置核心网桥

网桥的功能是使用来自一个目的地的消息并将它们转发到另一个目的地，通常在不同的 JBoss EAP 消息传递服务器上。

源和目标服务器不必位于同一集群中，这样可使桥接适合从一个集群将消息可靠发送到另一个集群，例如跨 WAN 或互联网，连接可能不可靠。

网桥内置了故障恢复能力，因此当目标服务器连接丢失时（例如，由于网络失败），网桥将重试连接到目标，直到它恢复在线。恢复在线后，它将正常恢复运行。

网桥是将两个单独的 JBoss EAP 消息传递服务器可靠连接在一起的一种方式。利用核心网桥，源和目标服务器必须是 JBoss EAP 7 消息传递服务器。

注意

不要将核心网桥与 Jakarta 消息传递网桥混淆。核心网桥用于桥接任何两个 JBoss EAP 消息传递实例并使用核心 API。Jakarta 消息传递网桥可用于桥接与 Jakarta 消息传递 2.0 兼容的任意两个 Jakarta 消息提供商，并使用 Jakarta 消息传递 API。最好尽可能使用核心网桥，而不是 Jakarta 消息传递网桥。

以下是 JBoss EAP 消息传递核心网桥的示例配置。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <bridge name="my-core-bridge" static-connectors="bridge-connector" queue-
name="jms.queue.InQueue"/>
    ...
  </server>
</subsystem>
```

可以使用以下管理 CLI 命令添加此核心网桥：请注意，在定义核心网桥时，您必须定义 queue-name 和 static-connectors 或 discovery-group。有关可配置属性的完整列表，请参见 附录中的表。

```
/subsystem=messaging-activemq/server=default/bridge=my-core-bridge:add(static-connectors=
[bridge-connector],queue-name=jms.queue.InQueue)
```

28.1. 配置内核桥接进行重复检测

如果消息中还没有一个，则核心网桥可以自动添加唯一的重复 ID 值，然后再将消息转发到目标。若要为重复消息检测配置核心网桥，可将 **use-duplicate-detection 属性** 设为 **true**，这是默认值。

```
/subsystem=messaging-activemq/server=default/bridge=my-core-bridge:write-attribute(name=use-duplicate-detection,value=true)
```

第 29 章 集群概述

JBoss EAP 消息传递群集允许将 JBoss EAP 消息传递服务器组分组在一起，以便共享消息处理负载。群集中的每个活动节点都是活跃的 JBoss EAP 消息传递服务器，用于管理自己的消息并处理自己的连接。

警告

messaging-activemq 子系统不支持由不同版本的 JBoss EAP 组成的混合集群。构成消息传递群集的服务器必须全部使用相同的 JBoss EAP 版本。

群集由每个节点组成，每个节点声明到 JBoss EAP 配置文件中其他节点的群集连接。当节点组成到另一节点的集群连接时，它在内部创建 其自身与其他节点之间的核心桥接 连接。这在幕后透明执行，您不必为每个节点声明一个明确的桥接。这些集群连接允许集群节点之间流动的消息来平衡负载。

集群的一个重要部分是 服务器发现，服务器 可以广播其连接详细信息，以便客户端或其他服务器可以通过最小配置进行连接。

本节还讨论 客户端侧负载平衡、平衡 群集节点之间的客户端连接和 消息重新 发布，其中 JBoss EAP 消息传递将在节点之间重新分发消息，以避免出现缺失。

警告

配置了群集节点后，通常只需将该配置复制到其他节点，即可生成对称集群。

实际上，集群中的每个节点都必须共享以下元素的相同配置，以避免意外错误：

- cluster-connection
- broadcast-group
- discovery-group
- address-settings，包括队列和主题

但是，在复制 JBoss EAP 消息文件时必须小心。不要将消息传递数据、绑定、日志和大型消息目录从一个节点复制到另一个节点。当首次启动集群节点并初始化其日志文件时，它会持久保留日志目录的特殊标识符。节点之间的标识符必须是唯一的，集群才能正确组成。

29.1. 服务器发现

服务器发现是一种机制，服务器可以通过这个机制将其连接详情传播到：

- 消息传递客户端

  消息传递客户端希望能够连接到群集的服务器，而不必了解群集中哪些服务器在任何一个时间上可用。

- 其他服务器

群集中的服务器希望在之前了解群集中的所有其他服务器的情况下能够互相创建群集连接。

此信息或群集拓扑围绕通过群集连接与客户端和其他服务器的正常 JBoss EAP 消息传递连接发送。但是，您需要一种方法来建立初始第一个连接。这可以通过 UDP 和 JGroups 等动态发现技术或提供初始连接器 的静态列表 来实现。

29.1.1. 广播组

广播组是服务器通过网络广播连接器的方式。连接器 定义客户端或其他服务器可以与服务器进行连接的方式。

广播组取一组连接器并在网络上广播它们。根据您配置群集的广播技术，它使用 UDP 或 JGroups 来广播连接器对信息。

广播组在服务器配置的 messaging-activemq 子系统中定义。每个 JBoss EAP 消息传递服务器可以有多个广播组。

使用 UDP 配置广播组

以下是定义 UDP 广播组的消息传递服务器配置示例。请注意，此配置依赖于 messaging-group 套接字绑定。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <broadcast-group name="my-broadcast-group" connectors="http-connector" socket-
binding="messaging-group"/>
    ...
  </server>
</subsystem>
...
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
  ...
  <socket-binding name="messaging-group" interface="private" port="5432" multicast-
address="231.7.7.7" multicast-port="9876"/>
  ...
</socket-binding-group>
```

此配置可通过以下管理 CLI 命令实现：

1. **添加 messaging-group 套接字绑定。**

```
/socket-binding-group=standard-sockets/socket-binding=messaging-group:add(interface=private,port=5432,multicast-address=231.7.7.7,multicast-port=9876)
```

2. **添加广播组。**

```
/subsystem=messaging-activemq/server=default/broadcast-group=my-broadcast-group:add(socket-binding=messaging-group,broadcast-period=2000,connectors=[http-connector])
```

使用 JGroups 配置广播组

以下是定义广播组的消息传递服务器的示例配置，它使用默认的 JGroups 广播组，后者使用 UDP。请注意，若要使用 JGroups 进行广播，您必须设置 jgroups-channel。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <broadcast-group name="my-broadcast-group" connectors="http-connector" jgroups-cluster="activemq-cluster"/>
    ...
  </server>
</subsystem>
```

这可以通过以下管理 CLI 命令进行配置：

```
/subsystem=messaging-activemq/server=default/broadcast-group=my-broadcast-group:add(connectors=[http-connector],jgroups-cluster=activemq-cluster)
```

广播组属性

下表列出了广播组的可配置属性。

属性	描述
broadcast-period	连续广播之间相隔的时间，以毫秒为单位。
connectors	要广播的连接器的名称。
jgroups-channel	**jgroups 子系统中定义的频道名称，它与 jgroups-cluster** 属性结合使用以组成集群。如果未定义，将 **使用默认** 频道。请注意，不同逻辑组之间的 **jgroups-channel** 多路复用组通信由 **jgroups-cluster** 属性标识。

属性	描述
jgroups-cluster	用于广播组与发现组之间通信的逻辑名称。希望从特定广播组接收消息的发现组必须使用与广播组相同的群集名称。
jgroups-stack	**jgroups** 子系统中定义的堆栈名称，用于组成集群。此属性已弃用。使用 **jgroups-channel** 组成集群。由于每个 **jgroups-stack** 都已与 **jgroups-channel** 关联，您可以使用该频道，或者您可以创建新的 **jgroups-channel**，并将它与 **jgroups-stack** 关联。  **重要** 如果同时指定了 **jgroups-stack** 和 **jgroups-channel**，则会生成一个新的 **jgroups-channel**，并且与 **jgroups-stack 和 jgroups- channel** 在同一命名空间中注册。因此，**jgroups-stack** 和 **jgroup-channel** 名称必须唯一。
socket-binding	广播组套接字绑定。

29.1.2. 发现组

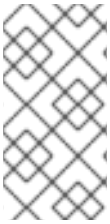
广播组定义如何从服务器广播连接器信息，但发现组定义如何从广播端点接收连接器信息，如 UDP 多播地址或 JGroup 通道。

发现组维护一个连接器列表，每个连接器由不同的服务器广播。当它从特定服务器在广播端点上接收广播时，它会更新该服务器的列表中的条目。如果在一段时间内没有从特定服务器收到广播，它将从列表中删除该服务器的条目。

发现组用于 JBoss EAP 消息传递的两个位置：

- **通过集群连接，它们知道如何获取初始连接来下载拓扑。**
- **通过消息传递客户端，他们知道如何获取用于下载拓扑的初始连接。**

尽管发现组将始终接受广播，但其当前可用的实时和备份服务器列表仅在进行初始连接时使用。从起，服务器发现通过普通的 JBoss EAP 消息传递连接进行。



注意

每一发现组必须配置有与其广播组对应位置的广播端点（UDP 或 JGroups）。例如，如果使用 UDP 配置广播组，发现组还必须使用 UDP 和相同的多播地址。

29.1.2.1. 在服务器上配置发现组

发现组在服务器配置的 messaging-activemq 子系统中定义。每个 JBoss EAP 消息传递服务器可以有许多发现组。

使用 UDP 配置发现组

以下是定义 UDP 发现组的消息传递服务器配置示例。请注意，此配置依赖于 messaging-group 套接字绑定。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <discovery-group name="my-discovery-group" refresh-timeout="10000" socket-
binding="messaging-group"/>
    ...
  </server>
</subsystem>
...
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
  ...
  <socket-binding name="messaging-group" interface="private" port="5432" multicast-
address="231.7.7.7" multicast-port="9876"/>
  ...
</socket-binding-group>
```

此配置可通过以下管理 CLI 命令实现：

1. 添加 messaging-group 套接字绑定。

```
/socket-binding-group=standard-sockets/socket-binding=messaging-
group:add(interface=private,port=5432,multicast-address=231.7.7.7,multicast-port=9876)
```

2. 添加发现组。

```
/subsystem=messaging-activemq/server=default/discovery-group=my-discovery-
group:add(socket-binding=messaging-group,refresh-timeout=10000)
```

使用 JGroups 配置发现组

以下是定义 JGroups 发现组的消息传递服务器配置示例。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
 <server name="default">
  ...
  <discovery-group name="my-discovery-group" refresh-timeout="10000" jgroups-cluster="activemq-cluster"/>
  ...
 </server>
</subsystem>
```

这可以通过以下管理 CLI 命令进行配置：

```
/subsystem=messaging-activemq/server=default/discovery-group=my-discovery-group:add(refresh-timeout=10000,jgroups-cluster=activemq-cluster)
```

发现组属性

下表列出了发现组的可配置属性。

属性	描述
initial-wait-timeout	以毫秒为单位等待初始广播为我们提供集群中的至少一个节点。
jgroups-channel	**jgroups** 子系统中定义的频道名称，它与 **jgroups-cluster** 属性结合使用以组成集群。如果未定义，将**使用默认**频道。请注意，不同逻辑组之间的 **jgroups-channel** 多路复用组通信由 **jgroups-cluster** 属性标识。
jgroups-cluster	用于广播组与发现组之间通信的逻辑名称。希望从特定广播组接收消息的发现组必须使用与广播组相同的群集名称。
jgroups-stack	**jgroups** 子系统中定义的堆栈名称，用于组成集群。此属性已弃用。使用 **jgroups-channel** 组成集群。由于每个 **jgroups-stack** 都已与 **jgroups-channel** 关联，您可以使用该频道，或者您可以创建新的 **jgroups-channel**，并将它与 **jgroups-stack** 关联。  **重要** 如果同时指定了 **jgroups-stack** 和 **jgroups-channel**，则会生成一个新的 **jgroups-channel**，并且与 **jgroups-stack 和 jgroups- channel** 在同一命名空间中注册。因此，**jgroups-stack** 和 **jgroup-channel** 名称必须唯一。

属性	描述
refresh-timeout	在从特定服务器接收最后一次广播后，发现组等待一次，然后从列表中删除该服务器的连接器对条目。
socket-binding	发现组套接字绑定。

警告

上面描述的 JGroups 属性和 UDP 相关属性是相互排斥的。在发现组配置中只能指定一个设置。

29.1.2.2. 在客户端幻灯片上配置发现组

您可以使用 Jakarta Messaging 或核心 API 配置 JBoss EAP 消息传递客户端，以发现其可以连接的服务器列表。

使用 Jakarta Messaging 配置客户端发现

使用 Jakarta Messaging 的客户端可以通过 JNDI 查找相关的 ConnectionFactory。connection-factory 或 pooled- connection-factory 的 entries 属性指定将在其中公开工厂的 JNDI 名称。以下是为远程客户端配置的 ConnectionFactory 配置为使用 JNDI 进行查找的示例：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <connection-factory name="RemoteConnectionFactory"
entries="java:jboss/exported/jms/RemoteConnectionFactory" connectors="http-connector"/>
    ...
  </server>
</subsystem>
```

注意

请记住，只有 java:jboss/exported 命名空间内绑定的 JNDI 名称才对远程客户端可用。如果 connection-factory 在 java:jboss/exported 命名空间中绑定了条目，远程客户端将使用 java:jboss/exported 后面的文本来查找 connection-factory。例如，RemoteConnectionFactory 默认绑定到 java:jboss/exported/jms/RemoteConnectionFactory，这意味着远程客户端 将使用 jms/RemoteConnectionFactory 来查找此连接 事实。pooled-connection-factory 应当不会在 java:jboss/exported 命名空间内绑定任何条目，因为 pooled-connection-factory 不适合于远程客户端。

自 Jakarta Messaging 2.0 起，默认的 Jakarta Messaging 连接工厂可供 JNDI 名称 java:comp/DefaultJMSConnectionFactory 下的 Jakarta EE 应用访问。JBoss EAP messaging-activemq 子系统定义一个 pooled-connection-factory，用于提供此默认连接工厂。任何 Jakarta EE 应用在 JNDI 名称 java:comp/DefaultJMSConnectionFactory 下查找默认的 Jakarta 消息提供商，都将考虑此 pooled-connection-factory 的任何参数更改。以下是 *-full 和 *- full- ha 配置集中定义的默认池连接工厂。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <pooled-connection-factory name="activemq-ra" transaction="xa" entries="java:/JmsXA
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm"/>
    ...
  </server>
</subsystem>
```

使用核心 API 配置客户端发现

如果您使用核心 API 直接实例化 ClientSessionFactory 实例，您可以在创建会话工厂时直接指定发现组参数。例如：

```
final String groupAddress = "231.7.7.7";
final int groupPort = 9876;

ServerLocator factory =
  ActiveMQClient.createServerLocatorWithHA(new DiscoveryGroupConfiguration(
     groupAddress,
     groupPort,
     new UDPBroadcastGroupConfiguration(groupAddress, groupPort, null, -1)));
ClientSessionFactory factory = locator.createSessionFactory();
ClientSession session1 = factory.createSession();
ClientSession session2 = factory.createSession();
```

您可以使用 DiscoveryGroupConfiguration 上的 setDiscoveryRefreshTimeout（） setter 方法设置 refresh-timeout 值，默认值为 10000 毫秒。

您还可以在 DiscoveryGroupConfiguration 上使用 setDiscoveryInitialWaitTimeout（） setter 方法设置 initial-wait-timeout 值，它决定了会话工厂在创建第一个会话前将等待的时长。默认值为 10000 毫秒。

29.1.3. 静态发现

如果您不能或不想在网络上使用 UDP，您可以使用一个或多个服务器的初始列表配置连接。

这并不表示您必须知道要托管所有服务器的位置。您可以将这些服务器配置为连接到可靠的服务器，并通过该服务器传播其连接详细信息。

配置集群连接

对于这里的集群连接，您只需要确保以常规方式定义任何 连接器。然后，集群连接配置会引用它们。

配置客户端连接

客户端也可以使用可能的服务器的静态列表。

使用 Jakarta Messaging 配置客户端发现

在 Jakarta Messaging 中使用静态发现的建议方法是通过多个连接器（ 分别指向集群中的一个唯一节点）配置连接 事实，并让客户端使用 JNDI 查找 ConnectionFactory。以下是仅显示此类 连接因素的配置片段 ：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <connection-factory name="MyConnectionFactory" entries="..." connectors="http-connector http-
node1 http-node2"/>
    ...
  </server>
</subsystem>
```

在上例中， http -connector 是一个指向本地服务器的 HTTP 连接器(<http-connector>)， http -node1 是一个指向服务器 node1 的 HTTP 连接器，以此类推。有关在服务器配置中配置连接器的信息，请参阅 Connectors 和 Acceptors 部分。

使用核心 API 配置客户端发现

如果您使用核心 API，请为集群中的每台服务器创建一个唯一的 TransportConfiguration，并将它们传递给负责创建 ServerLocator 的方法，如下例所示。

```
HashMap<String, Object> map = new HashMap<String, Object>();
```

```
map.put("host", "myhost");
map.put("port", "8080");

HashMap<String, Object> map2 = new HashMap<String, Object>();
map2.put("host", "myhost2");
map2.put("port", "8080");

TransportConfiguration server1 = new
TransportConfiguration(NettyConnectorFactory.class.getName(), map);
TransportConfiguration server2 = new
TransportConfiguration(NettyConnectorFactory.class.getName(), map2);

ServerLocator locator = ActiveMQClient.createServerLocatorWithHA(server1, server2);
ClientSessionFactory factory = locator.createSessionFactory();
ClientSession session = factory.createSession();
```

29.1.4. 默认Groups 值

在以前的版本中，您必须查看 jgroups-defaults.xml 文件来查找默认的 JGroups 值，这非常耗时。为方便起见，红帽列出了以下默认的 JGroups 值：

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:org:jgroups">
   <UDP
      ip_ttl="2"
      mcast_recv_buf_size="25m"
      mcast_send_buf_size="1m"
      ucast_recv_buf_size="20m"
      ucast_send_buf_size="1m"
      port_range="0"
   />
   <TCP
      send_buf_size="640k"
      sock_conn_timeout="300"
      port_range="0"
   />
   <TCP_NIO2
      send_buf_size="640k"
      sock_conn_timeout="300"
      port_range="0"
   />
   <TCPPING port_range="0" num_discovery_runs="4"/>
   <MPING ip_ttl="2"/>
   <kubernetes.KUBE_PING port_range="0"/>
   <MERGE3
      min_interval="10000"
      max_interval="30000"
   />
   <FD max_tries="5"
      msg_counts_as_heartbeat="false"
      timeout="3000"
   />
   <FD_ALL
```

```
        interval="15000"
        timeout="60000"
        timeout_check_interval="5000"/>
    <FD_SOCK/>
    <VERIFY_SUSPECT timeout="1000"/>
    <pbcast.NAKACK2
        xmit_interval="100"
        xmit_table_num_rows="50"
    />
    <UNICAST3
        xmit_interval="100"
        xmit_table_num_rows="50"
    />
    <pbcast.STABLE
        stability_delay="500"
        desired_avg_gossip="5000"
        max_bytes="1m"
    />
    <pbcast.GMS print_local_addr="false"/>
    <UFC max_credits="2m"/>
    <MFC max_credits="2m"/>
    <FRAG2 frag_size="30k"/>
</config>
```

29.2. 服务器端消息负载平衡

如果在群集的节点之间定义了群集连接，则 JBoss EAP 消息传递将对客户端到达特定节点的消息进行负载平衡。

消息传递集群连接可以配置为以轮循方式对消息进行负载平衡，无论其他节点上是否存在匹配的使用者。也可以将其配置为仅在存在匹配的消费者时分发到其他节点。如需更多信息，请参阅 Message Redistribution 部分。

配置集群连接

群集连接将服务器分组到集群中，以便在群集的节点之间平衡消息。群集连接在 JBoss EAP 服务器配置中利用 cluster-connection 元素定义。

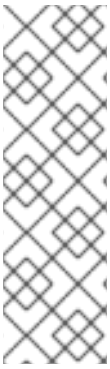
警告

红帽支持在 messaging -activemq 子系统内仅使用一个 cluster- connection。

以下是 * -full 和 *-full- ha 配置集中定义的默认 cluster- connection。如需 完整的属性列表，请参阅 群集连接 属性。

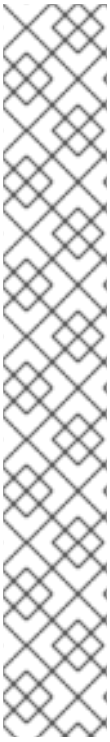
```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <cluster-connection name="my-cluster" discovery-group="dg-group1" connector-name="http-
connector" address="jms"/>
    ...
  </server>
</subsystem>
```

在上面显示的情况下，集群连接将加载发送到以"jms"开头的地址的消息负载平衡。此群集连接实际上将应用到所有 Jakarta 消息传递队列和主题，因为它们映射到以子字符串"jms"开头的核心队列。



注意

当使用集群连接发送数据包并处于受阻状态并等待确认时，call-timeout 属性指定在引发异常前等待回复的时长。默认值为 30000。在某些情况下，例如，如果远程 Jakarta 消息传递代理与网络断开连接且交易不完整，则线程可能会停留在连接重新建立前。为避免这种情况，建议使用 call-failover-timeout 属性和 call-timeout 属性。在故障转移尝试期间发出调用时使用 call-failover-timeout 属性。默认值为 -1，即没有超时。有关客户端故障的更多信息，请参阅 自动客户端故障切换。



注意

或者，如果您希望集群连接使用静态服务器列表进行发现，您可以使用 static-connectors 属性。例如：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <cluster-connection name="my-cluster" static-connectors="server0-connector
server1-connector" .../>
    ...
  </server>
</subsystem>
```

在本例中，我们定义了两台服务器，其中至少一台服务器可用。群集中可能有更多服务器，但初始连接后，将使用其中一个连接器来发现这些服务器。

为重复数据删除配置集群连接

集群连接内部使用 核心网桥 在集群节点之间移动消息。若要为重复的消息检测配置集群连接，请将 use-duplicate-detection 属性 设置为 true，这是默认值。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:write-attribute(name=use-duplicate-detection,value=true)
```

集群用户凭证

在集群节点之间创建连接以组成群集连接时，JBoss EAP 消息使用群集用户和密码。

您可以使用以下管理 CLI 命令来设置集群用户和密码。

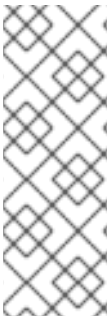
```
/subsystem=messaging-activemq/server=default:write-attribute(name=cluster-user,value="NewClusterUser")
/subsystem=messaging-activemq/server=default:write-attribute(name=cluster-password,value="NewClusterPassword123")
```

这会将下列 XML 内容添加到 JBoss EAP 配置文件中的 messaging-activemq 子系统：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <cluster user="NewClusterUser" password="NewClusterPassword123"/>
    ...
  </server>
</subsystem>
```

警告

cluster-user 的默认值为 ACTIVEMQ.CLUSTER.ADMIN.USER，cluster-password 的默认值为 CHANGE ME!!。这些值必须从默认值更改，否则远程客户端将能够使用默认值连接服务器。如果未从默认值更改它们，JBoss EAP 消息传递将检测到这一点并在每次启动时显示警告。



注意

您还可以使用 cluster-credential-reference 属性引用 凭证存储，而不设置集群密码。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=cluster-credential-reference,value={clear-text=SecretStorePassword})
```

29.3. 客户端负载均衡

借助 JBoss EAP 消息客户端负载平衡，使用单一会话工厂创建的后续会话可以连接到群集的不同节点。这允许会话在群集的节点间均匀分布，而不会在任何特定节点上断开。

声明客户端工厂使用的负载平衡策略的建议方法是设置 <connection -factory> 资源的 connection-load-balancing-policy- class-name 属性。JBoss EAP 消息传递提供下列出厂负载平衡策略，您也可以自行实施：

round robin

采用此策略时，会随机选择第一个节点，然后按相同顺序按顺序选择每个后续节点。

例如，节点可能按 B、C、D、A、B、C、D、A、B 或 D、A、B、C、D、A、B、B、A、B、C、A 、 B、C 的顺序选择。

使用 org.apache.activemq.artemis.api.core.client.loadbalance.RoundRobinConnectionLoadBalancingPolicy 作为 connection-load-balancing-policy-class-name。

随机

采用此策略时，每个节点都会随机选择。

使用 org.apache.activemq.artemis.api.core.client.loadbalance.RandomConnectionLoadBalancingPolicy 作为 connection-load-balancing-policy-class-name。

随机粘滞

采用此策略时，会随机选择第一个节点，然后重新用于后续连接。

使用 org.apache.activemq.artemis.api.core.client.loadbalance.RandomStickyConnectionLoadBalancingPolicy 作为 connection-load-balancing-policy-class-name。

第一个元素

采用此策略时，将始终返回第一个或 0th 节点。

使用 org.apache.activemq.artemis.api.core.client.loadbalance.FirstElementConnectionLoadBalancingPolicy 作为 connection-load-balancing-policy-class-name。

您还可以通过实施接口 org.apache.activemq.artemis.api.core.client.loadbalance.ConnectionLoadBalancingPolicy来实施自己的策略

29.4. 消息 REDISTRIBUTION

通过消息重新发布，JBoss EAP 消息传递可以配置为从队列自动重新分发消息，这些队列中没有任何使用者返回到群集中具有匹配使用者的其他节点。要启用此功能，集群连接的 message-load-balancing-type 必须设置为 ON_DEMAND，这是默认值。您可以使用以下管理 CLI 命令设置此项：

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:write-attribute(name=message-load-balancing-type,value=ON_DEMAND)
```

可以将消息重新分发配置为在队列上的最后一个使用者关闭后立即启动，或者在队列上的最后一个使用者关闭后等待可配置的延迟再分发。这使用 reload -delay 属性进行配置。

您可以使用 重新发布-delay 属性设置在队列中最后一个消费者关闭后等待多少毫秒，然后再将来自该队列的消息重新分发到群集中匹配消费者的其他节点。值为 -1 （默认值）表示永远不会重新分发消息。值 0 表示将立即重新分发消息。

默认 JBoss EAP 配置中的 address-setting 设置 re istribute-delay 值 1000，这意味着它将等待 1000 毫秒后再分发消息。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <address-setting name="#" redistribution-delay="1000" message-counter-history-day-limit="10"
page-size-bytes="2097152" max-size-bytes="10485760" expiry-address="jms.queue.ExpiryQueue"
dead-letter-address="jms.queue.DLQ"/>
    ...
  </server>
</subsystem>
```

在重新分发之前引入延迟通常比较适当，因为消费者关闭，但在同一队列中快速创建另一个延迟的情况通常比较适当。在这种情况下，您可能不想立即重新分发，因为新消费者很快会到达。

以下是 address-setting 的示例，它为绑定到 以"jms"开头的地址的任何队列或主题设置重新分配-delay 0。在这种情况下，消息将立即重新分发。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <address-setting name="jms.#" redistribution-delay="0"/>
    ...
  </server>
</subsystem>
```

此地址设置可通过以下管理 CLI 命令添加：

```
/subsystem=messaging-activemq/server=default/address-setting=jms.#:add(redistribution-delay=1000)
```

29.5. 集群消息分组

重要

不支持此功能。

群集分组遵循与正常 消息分组相关的不同方法。在群集中，具有特定组 ID 的消息组可以到达任何节点。节点务必要确定哪些组 ID 绑定到哪个节点上哪个消费者。无论消息组默认到达何处，每个节点负责将消息组路由到拥有消费者处理这些组 ID 的节点。当带有给定组 id 的消息发送到连接到集群中给定节点的特定消费者后，即使消费者断开连接，这些消息也不会发送到另一个节点。

这种情形可通过分组处理程序来解决。每个节点都有一个分组处理程序，此分组处理程序（以及其他处理程序）负责将消息组路由到正确的节点。有两种类型的分组处理程序：LOC AL 和 REMOTE。

本地处理程序负责确定消息组应采用的路由。远程处理程序与本地处理程序通信，并相应地工作。每个群集应当选择具有本地分组处理程序的特定节点，所有其他节点都应具有远程处理程序。

警告

如果群集中使用了消息分组，则配置为远程分组处理程序的节点失败时，它将中断。为远程分组处理程序设置备份将不正确。

最初收到消息组的节点根据常规集群路由条件（循环队列可用性）做出路由决定。节点向对应的分组处理程序提出此决定，后者将消息路由到建议队列（如果它接受该请求）。

如果分组处理程序拒绝该提议，它会提出其他路由，相应地进行路由。其他节点跟随 Suite，并将消息组转发到所选队列。消息到达队列中后，它将固定到该队列上的客户。

您可以使用管理 CLI 配置分组处理程序。以下命令添加带有地址 news.europe.# 的 LOCAL 分组处理程序。

```
/subsystem=messaging-activemq/server=default/grouping-handler=my-group-handler:add(grouping-handler-address="news.europe.#",type=LOCAL)
```

这将需要重新加载服务器。

```
reload
```

下表列出了 grouping -handler 的可配置 属性：

属性	描述
group-timeout	使用 **REMOTE** 处理程序时，此值指定 **REMOTE** 将通知 **LOCAL** 使用路由的频率。使用 **LOCAL** 处理程序时，如果路由没有用于指定的时间，它将被删除，并且需要建立新的路径。该值以毫秒为单位。
grouping-handler-address	对集群连接及其使用的地址的引用。
Reaper-period	回收器运行的频率，以检查是否有超时的组绑定（仅适用于 **LOCAL** 处理程序）。
timeout	等待多久才能做出处理决定；如果达到此超时，则在发送期间引发异常，确保保持严格排序。

属性	描述
type	处理程序是否为群集的单一本地处理程序，用于做出处理决策，或者是违反了本地处理程序的远程处理程序。可能的值有 **LOCAL** 或 **REMOTE**。

29.5.1. 集群消息分组的最佳做法

集群分组的一些最佳实践如下：

- 如果您定期创建和关闭消费者，请确保您的使用者在不同节点上均匀分布。固定队列后，消息将自动传输到该队列，无论客户从队列中移除。
- 如果您要删除绑定了消息组的队列，请确保正在发送消息的会话将删除队列。这样做可确保其他节点在移除后不会尝试将消息路由到此队列。
- 作为故障转移机制，始终复制具有本地分组处理程序的节点。

29.6. 启动和停止消息传递集群

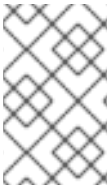
当您将 JBoss EAP 7.4 服务器配置为组成 ActiveMQ Artemis 集群时，可能还有其他服务器和客户端连接到正在运行的集群服务器。建议先关闭连接的客户端和服务器，再关闭群集中运行的 JBoss EAP 7.4 服务器。这必须按顺序执行，而不能并行执行，以便服务器有足够的时间关闭所有连接并避免在关闭可能会导致不一致状态时发生故障。ActiveMQ Artemis 不支持自动缩减群集节点，并且希望重新启动所有群集节点。

启动服务器时也是如此。您必须首先在 ActiveMQ Artemis 集群中启动 JBoss EAP 7.4 服务器。启动完成后，您可以启动其他连接到集群的服务器和客户端。

第 30 章 高可用性

高可用性是系统能够在一个或多个服务器出现故障后继续正常工作的能力。

高可用性的一部分是故障转移，后者是客户端连接能够在服务器出现故障时从一台服务器迁移到另一台服务器，以便客户端应用程序可以继续运行。



注意

只有持久消息数据才能保留故障切换。故障转移后任何非持久消息数据都将不可用。

30.1. 实时/备份对

JBoss EAP 7 消息传递使得服务器能够作为实时链接在一起 - 每个现场服务器都有备份对备份对。实时服务器从客户端接收消息，而备份服务器在故障转移发生之前不运行。备份服务器只能归一台现场服务器所有，它将保持被动模式，等待实时服务器工作。

注意

活动服务器和备份服务器之间有一个一对一的关系。实时服务器只能 *有一个* 备份服务器，并且备份服务器只能归 *一台* 实时服务器所有。

当实时服务器崩溃或进入正确的模式时，当前处于被动模式的备份服务器将成为新的实时服务器。如果新实时服务器配置为允许自动故障恢复，它将检测到旧的实时服务器重新启动并自动停止，从而使旧的实时服务器可以再次开始接收消息。

注意

如果您只部署一对实时/备份服务器，则无法在对前有效使用负载平衡器，因为备份实例没有主动处理消息。此外，JNDI 和 Undertow Web 服务器等服务也不会在备份服务器上处于活动状态。因此，不支持将 JEE 应用部署到用作备份消息传递服务器的 JBoss EAP 实例。

30.1.1. 日志同步

使用复制日志配置 HA 时，备份必须与实时服务器同步。

要检查同步是否完成，在 CLI 中提交以下命令：

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:read-
attribute(name=synchronized-with-backup)
```

如果结果为 true，则同步已完成。

要检查关闭实时服务器是否安全，请在 CLI 中提交以下命令：

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-slave:read-
attribute(name=synchronized-with-live)
```

如果结果为 true，可以安全地关闭实时服务器。

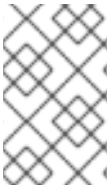
30.2. HA 策略

JBoss EAP 消息传递支持两种不同的策略来备份服务器：复制和共享存储。使用 服务器配置 元素的 ha-policy 属性，将您选择的策略分配给给定的服务器。ha-policy 有四个有效值：

- replication-master
- replication-slave
- shared-store-master
- shared-store-slave

如您所见，值指定服务器使用 数据复制 还是 共享存储 ha 策略，以及它是否取 master 还是 slave 角色。

使用管理 CLI 将 ha-policy 添加到您选择的服务器。



注意

以下示例假定您将使用 **standalone-full-ha** 配置配置文件运行 **JBoss EAP**：

```
/subsystem=messaging-activemq/server=SERVER/ha-policy=POLICY:add
```

例如，使用以下命令将 **replication-master** 策略 添加到 默认 服务器：

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:add
```

replication-master 策略使用默认值进行配置。添加策略时，可以包含覆盖默认配置的值。管理 **CLI** 命令使用以下基本语法来读取当前配置：

```
/subsystem=messaging-activemq/server=SERVER/ha-policy=POLICY:read-resource
```

例如，使用以下命令来读取上方添加到 默认 服务器的 **replication-master** 策略的当前配置：也包含输出以突出显示默认配置。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:read-resource

{
    "outcome" => "success",
    "result" => {
        "check-for-live-server" => true,
        "cluster-name" => undefined,
        "group-name" => undefined,
        "initial-replication-sync-timeout" => 30000L
    }
}
```

如需了解每个策略可用的配置选项 的详细信息，请参阅数据 复制和 共享存储。

30.3. 数据复制

在使用复制时，实时和备份服务器对不会共享相同的数据目录，所有数据同步都是通过网络进行的。因此，实时服务器接收的所有（永久）数据都将复制到备份中。

如果实时服务器已彻底关闭，备份服务器将激活，并且客户端将故障转移到备份。这个行为是预先确定的，因此在使用数据复制时不可配置。

备份服务器首先需要同步来自活动服务器的所有现有数据，然后才能进行替换。因此，与共享存储不同，副本备份在启动后不会立即完全运行。同步的发生时间取决于要同步的数据量和网络速度。另请注意，当备份启动时，客户端会在 初始复制-sync-timeout 期间被阻止。在此超时后，客户端将被取消阻塞，即使没有完成同步。

成功故障转移后，备份的日志将开始保存比实时服务器上的数据更新的数据。您可以将原始的实时服务器配置为执行故障恢复，并在重新启动后成为实时服务器。故障恢复将在实时服务器恢复在线之前同步备份与实时服务器之间的数据。

如果两个服务器都已关闭，管理员必须确定哪些服务器的日志具有最新的数据。如果备份日志具有最新的数据，请将该日志复制到实时服务器。否则，每当它再次激活时，备份将从实时服务器复制陈旧的日志数据并删除自己的日志数据。如果实时服务器的数据是最新数据，则不需要任何操作，并且可以正常启动服务器。

重要

由于数据中心之间的延迟较高，数据中心之间可能不可靠，因此不建议配置和使用复制日志以实现数据中心之间的高可用性。

复制实时和备份对必须是集群的一部分。cluster-connection 配置元素定义备份服务器如何找到其实时匹配项。

复制至少需要三个实时/备份对，以减少网络隔离的风险，但您无法消除此风险。如果您使用至少三个 live/backup 对，集群可以使用仲裁投票来避免使用两个 live 代理。

配置 cluster-connection 时，请记住以下详情：

- 实时和备份服务器必须属于同一群集。请注意，即使简单的实时/备份复制对也需要集群配置。
- 集群用户和密码必须与对中每台服务器上匹配。

通过在 < master> 和 <slave> 元素中配置 group-name 属性来指定 一对 live/backup 服务器。备份服务器仅连接到共享同一 组名的实时服务器。

作为使用 group-name 的示例，假设您有三个实时服务器和三个备份服务器。因为每个实时服务器都必须带有自己的备份对，所以请分配以下组名称：

- live1 和 backup1 使用 pair1 的 group-name。
- live2 和 backup2 使用 pair2 的 group-name。
- live3 和 backup3 使用 pair3 的 group-name。

在本例中，服务器 backup1 会搜索具有相同 组名称对 1 的实时服务器，本例中为服务器 live1。

与共享存储案例中非常相似，当实时服务器停止或崩溃时，其复制和配对备份将变为活动状态并接管其职责。具体来说，配对备份将在丢失与其实时服务器的连接时激活。这可能有问题，因为可能会因为临时网络问题而发生。为解决这个问题，配对备份将尝试确定它是否仍然可以连接到群集中的其他服务器。如果它可以连接到超过一半的服务器，它将变为活动状态。如果无法与其实时服务器通信，并且群集中的其他服务器超过一半，配对备份将等待并尝试重新与实时服务器连接。这可降低"脑裂"情况的风险，即备份和实时服务器正在处理消息，而其他人不知道该情况。

重要

这是共享存储备份的重要区别，在共享存储备份中，如果备份找不到实时服务器并且日志中的文件锁定已被释放，则备份将激活并开始为客户端请求提供服务。另请注意，在复制时，备份服务器不知道其是否具有最新数据，因此它实际上无法决定自动激活。要使用其所拥有的数据激活备份服务器副本，管理员必须通过将从卷更改为主服务器来更改其配置，使它成为实时服务器。

其他资源

- 配置集群连接

30.3.1. 配置数据复制

以下是两个示例，显示了驻留在名为 my-cluster 的群集和名为 group1 的备份组中的实时和备份服务器的基本配置。

以下步骤使用管理 CLI 为驻留于名为 my-cluster 的群集和名为 group1 的备份组中的实时和备份服务器提供基本配置。

注意

以下示例假定您将使用 standalone-full-ha 配置配置文件运行 JBoss EAP：

管理 CLI 命令以配置用于数据复制的实时服务器

1. **将 ha-policy 添加到 Live Server**

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(check-for-live-server=true,cluster-name=my-cluster,group-name=group1)
```

check-for-live-server 属性告知 live 服务器检查以确保没有其他服务器在群集中具有其给定 ID。此属性的默认值在 JBoss EAP 7.0 中为 false。在 JBoss EAP 7.1 及更高版本中，默认值为 true。

2. **将 ha-policy 添加到备份服务器**

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-slave:add(cluster-name=my-cluster,group-name=group1)
```

3. **确认共享的 cluster-connection 是否存在。**

实时和备份服务器之间的正确通信需要 群集连接。使用以下管理 CLI 命令，确认实时和备份服务器上配置了相同的 cluster-connection ：该示例使用 standalone -full-ha 配置配置文件中找到的默认 cluster- connection，这应该足以满足大多数用例的需要。如需有关如何 配置集群连接的详细信息，请参阅 配置集群连接。

使用以下管理 CLI 命令，确认实时和备份服务器使用相同的 cluster-connection：

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:read-resource
```

如果存在 cluster-connection，输出将提供当前的配置。否则将显示错误消息。

有关 所有配置属性的详情，请参阅所有复制 配置。

30.3.2. 所有复制配置

在添加配置后，您可以使用管理 CLI 将配置添加到策略中。要执行此操作的命令遵循以下基本语法：

```
/subsystem=messaging-activemq/server=default/ha-policy=POLICY:write-
attribute(name=ATTRIBUTE,value=VALUE)
```

例如，若要将 restart-backup 属性的值设为 true，可使用以下命令：

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-slave:write-
attribute(name=restart-backup,value=true)
```

下表提供 replication -master 节点和 replication- slave 配置元素的 HA 配置属性。

表 30.1. replication-master的属性

属性	描述
check-for-live-server	设置为 **true** 可告知此服务器在启动时使用相同服务器 ID 检查另一台服务器的集群。JBoss EAP 7.0 的默认值为 **false**。JBoss EAP 7.1 及更高版本的默认值为 **true**。
cluster-name	用于复制的集群的名称。
group-name	如果设置，备份服务器将仅对具有匹配 **group-name** 的实时服务器。
initial-replication-sync-timeout	在同步发起复制前，以毫秒为单位等待的时间。默认值为 **30000**。
synchronized-with-backup	指明实时服务器上的日志是否已同步。

表 30.2. replication-slave的属性

属性	描述
allow-failback	当另一服务器发出接管其位置的请求时，该服务器是否会自动停止。典型的用例是实时服务器请求，请求在重新启动或故障恢复后恢复活动处理。当实时服务器重新加入群集并请求恢复处理后，其 **allow-failback** 设为 **true** 的备份服务器将返回到该实时服务器。默认值为 **true**。

属性	描述
cluster-name	用于复制的集群的名称。
group-name	如果设置，备份服务器将仅对具有匹配 **group-name** 的实时服务器。
initial-replication-sync-timeout	在同步发起复制前，以毫秒为单位等待的时间。默认值为 **30000**。
max-saved-replicated-journal-size	指定复制备份服务器在开始移动其文件后可以重新启动的次数。达到最大值后，如果 出现故障，服务器将在 之后永久停止。默认值为 **2**。
restart-backup	设置为 **true**，告知此备份服务器在因为故障恢复而停止后重新启动。默认值为 **true**。
sync-with-live	指明复制服务器上的日志是否与实时服务器同步，这意味着可以安全地关闭实时服务器。

30.3.3. 防止集群连接超时

每个实时和备份对使用 集群连接 进行通信。cluster -connection 的 call- timeout 属性设置服务器在向群集中另一台服务器发出调用后等待响应的时间。call-timeout 的默认值为 30 秒，这足以满足大多数用例的需要。但在某些情况下，备份服务器可能无法处理来自实时服务器的复制数据包。例如，由于磁盘操作较慢或 journal- min-file 的值较大，日志文件的初始预创建时间可能会发生。如果出现如此超时的情况，您将在日志中看到与以下类似的一行。

```
AMQ222207: The backup server is not responding promptly introducing latency beyond the limit.
Replication server being disconnected now.
```

警告

如果诸如上面的行出现在日志中，这意味着复制过程已停止。您必须重新启动备份服务器才能重新初始化复制。

要防止集群连接超时，请考虑以下选项：

- 增加 cluster -connection 的 call- timeout。如需更多信息，请参阅配置集群连接。

- 降低 journal-min-files 的值。如需更多信息，请参阅配置持久性。

30.3.4. 删除旧日志目录

当备份服务器开始与实时服务器同步时，备份服务器会将其日志移到新位置。默认情况下，日志目录位于 EAP_HOME /standalone 下的 data/ activemq 目录中。对于域，每一服务器自己的 serverX/data/activemq 目录位于 EAP_HOME/domain/servers 下。目录命名为 bindings、日志、大型消息和 分页。有关这些目录的更多信息，请参阅 配置持久性和 配置 Paging。

移动之后，新目录将重命名为 旧replica.X，其中 X 是数字后缀。如果由于新的故障转移而启动另一个同步，则"moved"目录的后缀将由 1 增加。例如，在第一次同步时，日志目录将移到 旧replica.1，第二个是 旧的replica.2，等等。原始目录将存储从实时服务器同步的数据。

默认情况下，备份服务器配置为管理出现两次故障切换和回退故障。在此之后，清理过程会触发移除旧replica.X 目录。您可以使用备份服务器上的 max-saved-replicated-journal-size 属性来更改触发清理过程的故障转移数量。

注意

实时服务器将 max-saved-replicated-journal-size 设置为 2。此值无法更改

30.3.5. 更新 Dedicated Live 和备份服务器

如果实时和备份服务器部署在专用拓扑中（每个服务器都在自己的 JBoss EAP 实例中运行），请按照以下步骤确保集群的平稳更新和重新启动。

1. 彻底关闭备份服务器。

2. 彻底关闭实时服务器。

3. 更新实时和备份服务器的配置。

4. 启动实时服务器。

5. 启动备份服务器。

30.3.6. 检测代理的网络隔离

要检测代理的网络隔离，您可以 ping 可配置的主机列表。使用以下参数之一配置如何检测网络上的代理状态：

- network-check-NIC ：注意要在 InetAddress.isReachable 方法中使用的网络接口控制器 (NIC)来检查网络可用性。
- network-check-period: （以毫秒为单位），它定义检查网络状态的频率。
- network-check-timeout ：在网络连接过期之前，请注意等待的时间段。
- network-check-list ：注意要检测网络状态的 ping 的 IP 地址列表。
- network-check-URL-list ：注意用于验证网络的 http URI 列表。
- network-check-ping-command ：注意 ping 命令及其用于检测 IPv4 网络上的网络状态的参数。
- network-check-ping6-command ：注意 ping 命令及其用于检测 IPv6 网络上的网络状态的参数。

流程

- 使用以下命令 ping 可配置的主机列表，以检测代理的网络隔离：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=<parameter-name>,
value="<ip-address>")
```

示例

要通过 ping IP 地址 10.0.0.1 检查网络状态，请发出以下命令：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=network-check-list,
value="10.0.0.1")
```

30.3.7. 数据复制的限制：脑裂处理

当实时服务器及其备份同时处于活动状态时，会发生"脑裂"情况。两个服务器都可以提供客户端和进程消息，而其他人不知道它。在这种情况下，实时和备份服务器之间不再有任何消息复制。如果这两个服务器之间进行网络故障，则会出现分割情况。

例如，如果实时服务器和网络路由器之间的连接中断，备份服务器将丢失与实时服务器的连接。但是，备份仍可以连接到集群中超过一半的服务器，因此它变为活跃状态。请记住，如果只有一个实时备份对，并且备份服务器丢失与实时服务器的连接，则备份也会激活。当两个服务器都在集群中处于活跃状态时，可能会发生两个不合预期的情况：

1. 远程客户端故障转移到备份服务器，但 MDB 等本地客户端将使用实时服务器。两个节点都有完全不同的日志，从而导致脑裂处理。

2. 在远程客户端切换到备份服务器后，与 live 服务器的连接已被修复。当旧客户端继续使用备份时，任何新客户端都将连接到实时服务器，这也会导致脑裂情景。

客户应在每对实时和备份服务器之间实施可靠的网络，以减少使用数据复制时脑裂处理的风险。例如，使用重复的网络接口卡和其他网络冗余。

30.4. 共享存储

这种高可用性方式不同于数据复制，因为它需要可由实时和备份节点访问的共享文件系统。这意味着服务器对在其配置中使用相同的位置来 分页、 消息日志 、绑定日志和 大型消息。

注意

Windows 上不支持使用共享存储。当使用红帽 GFS2 或 NFSv4 版本时，Red Hat Enterprise Linux 中支持它。另外，只有 ASYNCIO 日志类型支持 GFS2，而 ASYNCIO 和 NIO 日志类型都支持 NFSv4。

此外，对中的每个参与服务器（实时和备份 ）也需要定义群集连接，即使不是群集的一部分，因为 群

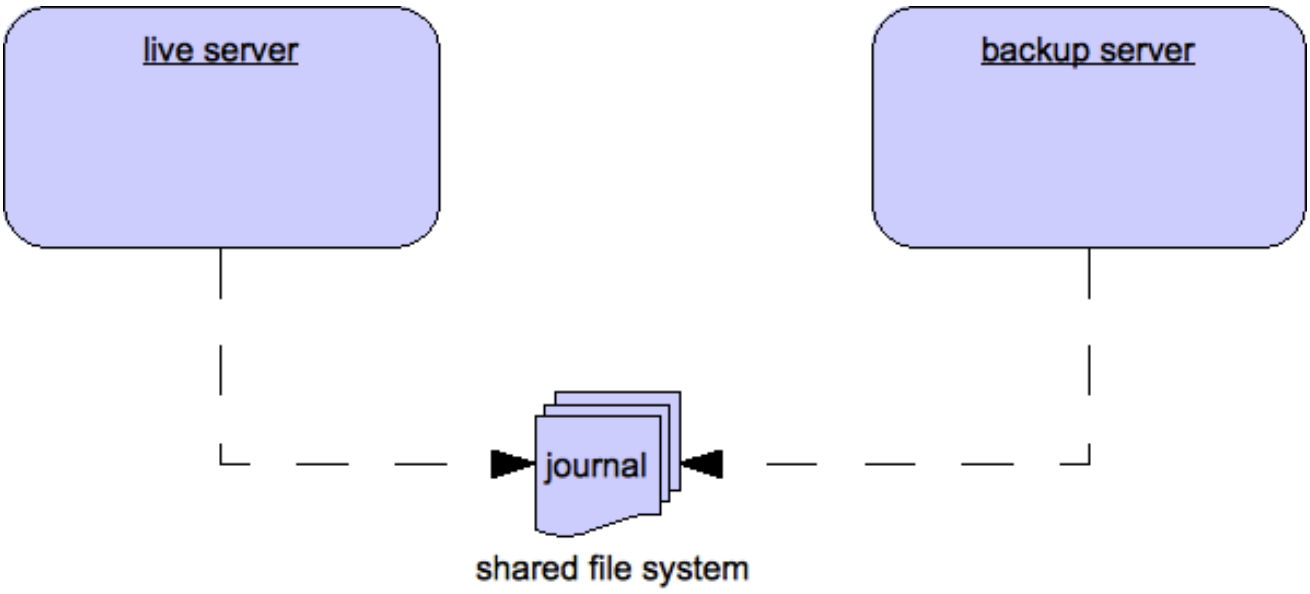
集连接 定义备份服务器如何将其存在声明到其实时服务器和其他任何节点。有关如何完成此操作的详情，请参阅配置集群连接。

发生故障转移并且备份服务器接管时，它将需要从共享文件系统加载持久存储，然后才能客户端连接它。这种高可用性方式不同于数据复制，因为它需要可由实时和备份对访问的共享文件系统。这通常是某种高性能存储区域网络或 SAN。红帽不推荐将网络附加存储（称为 NAS）用于您的存储解决方案。

共享存储高可用性的优势在于实时节点和备份节点之间不会发生复制，这意味着它不会因为正常操作期间复制的开销而受到性能偏差。

共享存储复制的缺点在于，当备份服务器激活它需要从共享存储加载日志时，这可能需要一些时间，具体取决于存储中的数据量。此外，它还需要 JBoss EAP 支持的共享存储解决方案。

如果您在正常操作过程中要求最高性能，红帽建议访问高性能 SAN，并接受稍慢的故障转移成本。确切的成本将取决于数据量。



30.4.1. 配置共享存储

注意

以下示例假定您将使用 standalone-full-ha 配置配置文件运行 JBoss EAP：

1. 将 ha-policy 添加到 Live Server。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-master:add
```

2. **将 ha-policy 添加到备份服务器。**

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:add
```

3. **确认共享的 cluster-connection 是否存在。**

实时和备份服务器之间的正确通信需要 群集连接。使用以下管理 CLI 命令，确认实时和备份服务器上配置了相同的 cluster-connection ：该示例使用 standalone -full-ha 配置配置文件中找到的默认 cluster- connection，这应该足以满足大多数用例的需要。如需有关如何 配置集群连接的详细信息，请参阅 配置集群连接。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:read-
resource
```

如果存在 cluster-connection，输出将提供当前的配置。否则将显示错误消息。

有关 共享存储策略的所有配置 属性的详情，请参阅所有共享存储配置。

30.4.2. 所有共享存储配置

在添加配置后，使用管理 CLI 将配置添加到策略中。要执行此操作的命令遵循以下基本语法：

```
/subsystem=messaging-activemq/server=default/ha-policy=POLICY:write-
attribute(name=ATTRIBUTE,value=VALUE)
```

例如，若要将 restart-backup 属性的值设为 true，可使用以下命令：

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-
attribute(name=restart-backup,value=true)
```

表 30.3. shared-store-master 配置元素的属性

属性	描述
failover-on-server-shutdown	设置为 **true**，告知此服务器正常关机时进行故障转移。默认为 **false**。

表 30.4. shared-store-slave 配置元素的属性

属性	描述
allow-failback	设置为 **true** 可让该服务器在另一个请求接管其位置时自动停止。当常规服务器停止并且其备份接管其职责时，该用例为主服务器重新启动并请求服务器（前备份）停止运行。默认值为 **true**。
failover-on-server-shutdown	设置为 **true**，告知此服务器正常关机时进行故障转移。默认为 **false**。
restart-backup	设置为 **true** 以指示此服务器在因为故障恢复或缩减而停止后重新启动。默认值为 **true**。

30.5. 回退到实时服务器失败

在实时服务器出现故障并且备份接管了自己的职责后，您可能希望重新启动实时服务器，并让客户端恢复运行。

如果是共享存储，只需重新启动原先的实时服务器并通过终止进程本身终止新的实时服务器。或者，您可以在从 卷上将 allow-fail-back 设置为 true，它会强制其在主服务器恢复在线后自动停止。设置 allow-fail-back 的管理 CLI 命令类似如下：

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-
attribute(name=allow-fail-back,value=true)
```

在复制 HA 模式中，您需要确保在主配置中 将 check-for-live-server 属性设置为 true。从 JBoss EAP 7.1 开始，这是默认值。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:write-
attribute(name=check-for-live-server,value=true)
```

如果设置为 true，则 实时服务器将在启动过程中使用 nodeID 搜索另一个服务器。如果找到一个服务器，它将联系此服务器并尝试"失败"。由于这是一种远程复制方案，原始的实时服务器必须将其数据与运行的备份与其 ID 同步。同步后，它将请求备份服务器关闭，以便它可以接管活动的处理。此行为可让原始的实时服务器确定是否存在故障转移，如果发生故障，则承担其职责的服务器是否仍在运行。

警告

请注意，如果在发生故障转移到备份后重新启动实时服务器，则 **check-for-live-server** 属性必须设为 **true**。如果没有，则实时服务器将立即启动，而不检查其备份服务器正在运行。这会导致实时和备份同时运行，从而导致将重复消息传送到所有新连接的客户端。

对于共享存储，也可以在正常服务器关闭时导致故障转移发生，以便在主或从设备上的 HA 配置中将此 **failover-on-server-shutdown** 设置为 **true** ：

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-attribute(name=failover-on-server-shutdown,value=true)
```

您还可以强制运行的备份服务器在原始实时服务器恢复时关闭，从而允许原始的实时服务器自动接管，方法是将 **allow-failback** 设置为 **true**。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-attribute(name=allow-failback,value=true)
```

30.6. 并置备份服务器

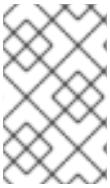
JBoss EAP 还使得备份消息传递服务器可以和另一台实时服务器在同一 JVM 中并置。例如，一个简单的双节点单机服务器集群，每个实时服务器将备份一起分配给另一个服务器。在以这种方式接合服务器时，您可以使用共享存储或复制 HA 策略。在配置消息传递服务器进行并置时，需要记住两个重要事项。

首先，配置中的每个 **server** 元素都需要自己的 **remote-connector** 和 **remote-acceptor** 或 **http-connector** 和 **http-acceptor**。例如，具有 远程接收器的实时服务器可以 配置为侦听端口 **5445**，而 来自并置备份的远程 接收器则使用端口 **5446**。端口在 **socket-binding** 元素中定义，必须添加到默认的 **socket-binding-group**。如果是 **http-acceptors**，则实时和并置备份可以共享相同的 **http-listener**。每个服务器配置中与集群相关的配置元素将使用 服务器 使用的 **remote-connector** 或 **http-connector**。相关的配置包含在下面的每个示例中。

其次，请记得为日志相关目录正确配置路径。例如，在共享存储中，实时服务器及其备份必须配置为共享 绑定 和 消息 日志的目录位置，以用于 大型消息，以及进行 分页。

30.6.1. 配置并置 HA 拓扑的手动创建

以下步骤中使用的管理 CLI 命令示例演示了如何使用并置拓扑配置简单的两个节点集群。这个示例配置了一个两个节点并置群集。实时服务器和备份服务器将存在于每个节点上。在 *节点 1 上的并置备份* 与 live 服务器并置在 *节点 2* 上，并且 *节点 2* 上的备份服务器与 *节点 1* 上的实时服务器配对。共享存储和数据复制 HA 策略的示例均包含在内。



注意

以下示例假定您将使用 full-ha 配置文件运行 JBoss EAP。

1. 修改每个实例上的默认服务器，以使用 HA 策略。每个节点上的默认服务器将成为实时服务器。您所遵循的说明取决于您是否已配置共享存储策略或数据复制策略。

   - *共享存储策略的说明：* 使用以下管理 CLI 命令添加首选 HA 策略：

     ```
     /subsystem=messaging-activemq/server=default/ha-policy=shared-store-master:add
     ```

   - *数据复制策略说明：* 每个节点中的默认服务器应配置一个唯一的 group-name。在以下示例中，第一个 命令在 *节点 1* 上执行，第二个命令在 *节点 2* 上执行。

     ```
     /subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(cluster-name=my-cluster,group-name=group1,check-for-live-server=true)

     /subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(cluster-name=my-cluster,group-name=group2,check-for-live-server=true)
     ```

2. 将新的备份服务器与每个实时服务器并置。

   a. 向 JBoss EAP 的每个实例添加新服务器，以与默认的实时服务器并置。新服务器将在另一节点上备份默认服务器。使用以下管理 CLI 命令，创建名为 backup 的新服务器：

      ```
      /subsystem=messaging-activemq/server=backup:add
      ```

   b. 接下来，将新服务器配置为使用首选 HA 策略。您所遵循的说明取决于您是否已配置共享存储策略或数据复制策略。

      - *共享存储策略的说明：* 使用以下管理 CLI 命令添加 HA 策略：

        -

```
/subsystem=messaging-activemq/server=backup/ha-policy=shared-store-slave:add
```

- *数据复制策略的说明：*将备份服务器配置为在其他节点上使用实时服务器的 **group-name**。在以下示例中，第一个 命令在 *节点 1* 上执行，第二个命令则在 *节点 2* 上执行。

```
/subsystem=messaging-activemq/server=backup/ha-policy=replication-slave:add(cluster-name=my-cluster,group-name=group2)

/subsystem=messaging-activemq/server=backup/ha-policy=replication-slave:add(cluster-name=my-cluster,group-name=group1)
```

3. 配置所有服务器的目录位置。

为 HA 配置服务器后，您必须配置 绑定 日志、消息 日志和 大型消息 目录的位置。如果您计划使用分页，还必须 配置分页 目录。您所遵循的说明取决于您是否已配置共享存储策略或数据复制策略。

- *共享存储策略说明：*实时服务器在 *节点 1* 的路径 值应指向支持的文件系统中与 *节点 2* 上的备份服务器相同的位置。节点 *2 上的实时服务器以及节点 1* 上的备份也是如此。

a. 使用以下管理 CLI 命令为 *节点配置目录位置* ：

```
/subsystem=messaging-activemq/server=default/path=bindings-directory:write-attribute(name=path,value=/PATH/TO/shared/bindings-A)

/subsystem=messaging-activemq/server=default/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-A)

/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-A)

/subsystem=messaging-activemq/server=default/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-A)

/subsystem=messaging-activemq/server=backup/path=bindings-directory:write-attribute(name=path,value=/PATH/TO/shared/bindings-B)

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-B)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-B)

/subsystem=messaging-activemq/server=backup/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-B)
```

b.

使用以下管理 CLI 命令为 *节点 2* 配置目录位置：

```
/subsystem=messaging-activemq/server=default/path=bindings-directory:write-attribute(name=path,value=/PATH/TO/shared/bindings-B)

/subsystem=messaging-activemq/server=default/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-B)

/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-B)

/subsystem=messaging-activemq/server=default/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-B)

/subsystem=messaging-activemq/server=backup/path=bindings-directory:write-attribute(name=path,value=/PATH/TO/shared/bindings-A)

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-A)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-A)

/subsystem=messaging-activemq/server=backup/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-A)
```

- ***数据复制策略说明：*** **每台服务器使用自己的目录，不将它们与任何其他服务器共享。在以下示例命令中，将 路径 位置的每个值假定为文件系统中的唯一位置。无需更改实时服务器的目录位置，因为它们将使用默认位置。但是，备份服务器仍然必须配置有唯一位置。**

a.

使用以下管理 CLI 命令为 *节点配置目录位置* ：

```
/subsystem=messaging-activemq/server=backup/path=bindings-directory:write-attribute(name=path,value=activemq/bindings-B)

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=activemq/journal-B)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=activemq/largemessages-B)

/subsystem=messaging-activemq/server=backup/path=paging-directory:write-attribute(name=path,value=activemq/paging-B)
```

b.

使用以下管理 CLI 命令为 *节点 2* 配置目录位置：

```
/subsystem=messaging-activemq/server=backup/path=bindings-directory:write-
```

```
attribute(name=path,value=activemq/bindings-B)

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=activemq/journal-B)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=activemq/largemessages-B)

/subsystem=messaging-activemq/server=backup/path=paging-directory:write-attribute(name=path,value=activemq/paging-B)
```

4. **添加新的接收器和连接器到备份服务器。**

每一备份服务器必须配置有 http-connector 以及使用默认 http- listener 的 http-acceptor。这使得服务器能够通过 HTTP 端口接收和发送通信。以下示例将 http-acceptor 和 http-connector 添加到备份服务器。

```
/subsystem=messaging-activemq/server=backup/http-acceptor=http-acceptor:add(http-listener=default)

/subsystem=messaging-activemq/server=backup/http-connector=http-connector:add(endpoint=http-acceptor,socket-binding=http)
```

5. **为备份 服务器配置 cluster-connection。**

每一消息传递服务器需要一个 群集连接、广播组和 发现组才能 正确通信。使用以下管理 CLI 命令配置这些元素：

```
/subsystem=messaging-activemq/server=backup/broadcast-group=bg-group1:add(connectors=[http-connector],jgroups-cluster=activemq-cluster)

/subsystem=messaging-activemq/server=backup/discovery-group=dg-group1:add(jgroups-cluster=activemq-cluster)

/subsystem=messaging-activemq/server=backup/cluster-connection=my-cluster:add(connector-name=http-connector,cluster-connection-address=jms,discovery-group=dg-group1)
```

并置服务器配置现已完成。

30.7. 故障切换模式

JBoss EAP 消息传递定义两种类型的客户端故障转移：

- 自动客户端故障切换
- 应用程序级别的客户端故障切换

JBoss EAP 消息传递还提供 100% 透明自动重新连接同一服务器（例如，当出现瞬态网络问题时）。这与故障转移类似，除了重新连接到同一服务器并在 客户端重新连接和会话附加中讨论。

在故障转移过程中，如果客户端在任何非持久或临时队列上拥有消费者，则在备份节点上故障转移期间将自动重新创建这些队列，因为备份节点将不知道非持久队列。

30.7.1. 自动客户端故障

JBoss EAP 消息传递客户端可以配置为接收所有实时和备份服务器的知识，这样，在客户端（实时服务器连接）连接发生连接故障时，客户端将检测故障并重新连接到备份服务器。然后备份服务器会在故障转移前自动重新创建每个连接上存在的任何会话和消费者，从而让用户不必手动编码手动重新连接逻辑。

JBoss EAP 消息传递客户端在 client-failure-check-period 期限内未从服务器接收到数据包时，会检测到连接失败，如 检测检测连接中所述。

如果客户端没有在分配的时间内接收数据，它将假定连接失败并尝试故障转移。如果套接字由操作系统关闭，则服务器进程可能会被终止，而不是服务器硬件本身崩溃，客户端将直接故障转移。

JBoss EAP 消息传递客户端可以通过多种不同方式进行配置，以发现实时备份服务器对的列表。例如，它们可以被配置为显式端点，但最常见的方法是客户端在第一次连接到集群时接收关于集群拓扑的信息。如需更多信息，请参阅服务器发现。

默认 HA 配置包括使用 推荐的 http-connector 进行集群通信的集群连接。这与使用默认的 RemoteConnectionFactory 进行服务器连接时远程客户端使用的 http-connector 相同。虽然不建议这样做，但您可以使用不同的连接器。如果使用自己的连接器，请确保它作为配置的一部分被远程客户端和集群节点使用的 cluster -factory 使用。如需有关 连接器和 集群连接 的更多信息，请参阅配置消息传递传输 和集群连接。

警告

用于 Jakarta 消息传递客户端的 connection-factory 中定义的 连接器 必须与集群使用的 cluster-connection 中定义的连接器相同。否则，客户端将无法更新其底层实时/备份对拓扑，因此不知道备份服务器的位置。

使用 CLI 命令检查 connection-factory 和 cluster- connection 的配置。例如，若要读取名为 RemoteConnectionFactory 的 connection-factory 的当前配置，可使用以下命令：

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:read-resource
```

类似地，以下命令读取名为 my-cluster 的 cluster-connection 的配置。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:read-resource
```

要启用自动客户端故障转移，客户端必须配置为允许非零重新连接尝试。如需更多信息，请参阅客户端重新连接和会话附加。默认情况下，只有在与实时服务器至少建立了一条连接后才会进行故障转移。换句话说，当客户端无法与实时服务器进行初始连接时，不会发生故障转移。如果初始尝试失败，客户端会只根据 reconnect -attempts 属性尝试连接到 live 服务器，并在配置完尝试次数后失败。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=reconnect-attempts,value=<NEW_VALUE>)
```

此规则的一个例外是，只有一对实时 - 备份服务器，并且没有其他实时服务器，并且远程 MDB 在完全关闭时连接到实时服务器。如果 MDB 配置了 @ActivationConfigProperty（propertyName = "rebalanceConnections", propertyValue = "true"，它将 尝试重新平衡它与另一个实时服务器的连接，并且不会故障转移到备份。

在初始连接中覆盖失败

由于客户端在进行第一次连接之前不会了解完整的拓扑，因此有一个时间窗口，它不知道备份。如果此时发生故障，客户端只能尝试重新连接到原始的实时服务器。若要配置客户端尝试次数，您可以设置 ClientSessionFactoryImpl 或 ActiveMQConnectionFactory 上属性 initialConnectAttempts。

或者，在服务器配置中，您可以设置客户端使用的连接工厂的 initial-connect-attempts 属性。默认值为 0，即，仅尝试一次。尝试次数后，将引发异常。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=initial-connect-attempts,value=
<NEW_VALUE>)
```

关于服务器复制

JBoss EAP 消息传递不会在实时服务器和备份服务器之间复制完整的服务器状态。当在备份上自动重新创建新会话时，将不会知晓该会话中已发送或确认的消息。故障切换时的任何内部发送或确认也可能丢失。

通过复制完整的服务器状态，JBoss EAP 消息传递理论上可提供 100% 透明无缝故障转移，从而避免任何丢失的消息或确认。但是，这样做需要高昂的成本：复制完整的服务器状态，包括队列和会话。这需要复制整个服务器状态机。也就是说，实时服务器上的每个操作都必须以完全相同的全局顺序复制到副本服务器上，以确保副本状态一致。这极其难以以高性能和可扩展的方式执行，特别是考虑到多个线程同时更改了实时服务器状态。

可以使用虚拟同步等技术提供完整的状态机复制，但这不能良好扩展并有效地将所有操作序列化到单个线程，从而显著降低并发性。存在用于多线程主动复制的其他技术，如复制锁定状态或复制线程调度，但这在 Java 级别上很难实现。

因此，出于 100% 透明故障转移的目的，无需降低性能和并发性。即使没有 100% 透明故障转移，也很容易通过将重复检测和重试交易结合使用，保证一次并仅一次交付一次（即便在故障的情况下也是如此）。但是，这对客户端代码不是 100% 透明的。

30.7.1.1. 在故障转移期间处理阻塞调用

如果客户端代码正在阻止服务器调用，例如，它正在等待响应以便在故障转移期间继续执行，新的会话将不知道正在进行的调用。否则被阻止的调用可能会永远挂起，等待永远不会出现响应。

为防止这种情况，JBoss EAP 消息传递将通过使生成 javax.jms.JMSException（如果使用 Jakarta Messaging）或带有错误代码 ActiveMQException 的 ActiveMQException.UNBLOCKED （如果使用核心 API）来取消阻塞任何在故障转移时正在进行的中断调用。客户端代码可以捕捉到这个异常，并在需要时重试任何操作。

如果被阻塞的方法是对 commit（）或 prepare（）的调用，则事务将被自动回滚，并且 JBoss EAP 消息传递将引发 javax.jms.TransactionRolledBackException，如果使用 Jakarta Messaging，或者带有错误代码 ActiveMQException 的 ActiveMQException. TRANSACTION_ROLLED_BACK（如果使用核心 API），则将引发 javax. jms.TransactionRolledBackException。

30.7.1.2. 使用事务处理失败

如果会话为事务性并且消息已在当前事务中发送或确认，则服务器无法确定故障转移期间是否丢失了消息或确认。

因此，事务将被标记为仅回滚，且后续提交的任何尝试都将引发 javax.jms.TransactionRolledBackException，如果使用 Jakarta Messaging。或者带有错误代码 ActiveMQException 的 ActiveMQException.TRANSACTION_ROLLED_BACK （如果使用核心 API）。

警告

应注意此规则是何时通过 Jakarta Messaging 或核心 API 使用 XA。如果使用了两个阶段提交，并且已调用 prepare（），则回滚可能会导致 HeuristicMixedException。因此，提交将抛出 XAException.XA_RETRY 异常。这告知交易管理器，它在稍后某个时间点重试提交，其副作用是将丢失任何非持久消息。为避免发生这种情况，在使用 XA 时务必使用持久的消息。通过确认这一点并不是个问题，因为它们在调用 prepare （） 之前被刷新到服务器。

用户可以捕捉异常，并根据需要执行任何客户端的本地回滚代码。不需要手动回滚会话，因为它已经回滚。然后，用户只需在同一会话中再次重试事务操作。

如果在执行提交调用时发生故障转移，服务器（如前文所述）将取消阻断调用以防止挂起，因为没有响应将返回。在这种情况下，客户端很难在出现故障前确定事务提交是否真正在实时服务器上处理。

注意

如果通过 Jakarta Messaging 或核心 API 使用 XAException.XA_RETRY，则会抛出 XAException.XA_RETRY。这是为了通知交易经理，应在某一点进行重试。稍后的某个时候，交易管理器将重试提交。如果原始提交尚未发生，它将仍然存在并提交。如果不存在，则假定它已提交，尽管事务管理器可能会记录警告。

为补救这一点，客户端可以在事务中启用重复检测，并在调用未阻塞后重试事务。如需有关如何在服务器上配置 检测的信息，请参阅重复消息 检测。如果事务在故障转移之前确实在实时服务器上进行了成功提交，则重复检测将确保服务器上重新定向的任何持久消息都会被忽略，以防止在事务重试时它们多次发送。

30.7.1.3. 通知连接失败

Jakarta Messaging 提供了一种标准机制，用于发送连接失败的异步通知：java.jms.ExceptionListener。有关本课程的更多信息，请参阅 Jakarta Messaging javadoc。核心 API 也以类 org.apache.activemq.artemis.core.client.SessionFailureListener 的形式提供类似的功能。

如果连接失败，则 JBoss EAP 始终会调用任何 Exception Listener 或 SessionFailureListener 实例，无论连接是成功失败、重新连接还是重新连接。但是，您可以通过检查 传入 到 SessionfailureListener 上的 connectionFailed（） 的值或 javax.jms.JMSException 的 javax.jms.JMSException 的值来找出重新连接或重新附加是否发生了：

JMSException 错误代码

错误代码	描述
故障切换	故障转移已经发生，我们已成功重新连接或重新连接。
断开连接	没有发生故障转移，我们断开连接。

30.7.2. 应用程序级别的故障切换

在某些情况下，您可能不希望自动客户端故障转移，更喜欢自己处理任何连接失败，并在自己的故障处理程序中编码您自己的手动重新连接逻辑。我们将此定义为应用级别的故障转移，因为故障转移是在用户应用级别上处理的。

如果您正在使用 Jakarta 消息传递在 Jakarta 消息传递连接上设置了 ExceptionListener 类，则实施应用程序级别的故障转移。当检测到连接失败时，JBoss EAP 消息传递将调用 ExceptionListener。在 ExceptionListener 中，您将关闭旧的 Jakarta Messaging 连接，可能会从 JNDI 查找新的连接工厂实例并创建新连接。

如果您使用核心 API，则流程非常相似： 您可以在核心 ClientSession 实例上设置 FailureListener。

30.8. 检测拒绝连接

本节讨论到实时连接时间(TTL)，并解释 JBoss EAP 消息传递如何处理已退出但未完全关闭资源的崩溃客户端和客户端。

清理服务器上的拒绝连接资源

在 JBoss EAP 客户端应用退出之前，它应当使用 最终 块以受控的方式关闭其资源。

以下是在最终 块中正确关闭其会话和会话工厂的核心客户端的示例 ：

```
ServerLocator locator = null;
ClientSessionFactory sf = null;
ClientSession session = null;

try {
   locator = ActiveMQClient.createServerLocatorWithoutHA(..);

   sf = locator.createClientSessionFactory();;

   session = sf.createSession(...);

   ... do some stuff with the session...
}
finally {
   if (session != null) {
      session.close();
   }

   if (sf != null) {
      sf.close();
   }

   if(locator != null) {
      locator.close();
   }
}
```

以下是功能良好的 Jakarta 消息传递客户端应用程序的示例：

```
Connection jmsConnection = null;

try {
   ConnectionFactory jmsConnectionFactory =
ActiveMQJMSClient.createConnectionFactoryWithoutHA(...);

   jmsConnection = jmsConnectionFactory.createConnection();

   ... do some stuff with the connection...
}
finally {
   if (connection != null) {
      connection.close();
   }
}
```

不幸的是，客户端有时崩溃，无法清理其资源。如果出现这种情况，它可以使服务器端资源挂起在服务器上。如果未移除这些资源，则会导致服务器上的资源泄漏，而且随着时间推移，可能会导致服务器内存或其他资源不足。

在查看清除死的客户端资源时，务必要清楚有时客户端和服务器之间的网络可能会出现故障，然后返回，允许客户端重新连接。由于 JBoss EAP 支持客户端重新连接，因此务必不要太快清理"dead"服务器端资源，否则将阻止任何客户端重新连接并恢复服务器上的旧会话。

JBoss EAP 使所有这些可配置。对于配置的每个 ClientSessionFactory （ Time-To-Live 或 TTL），可以使用 属性来设置服务器在没有客户端的任何数据时保持连接的时长（以毫秒为单位）。客户端将定期自动发送"ping"数据包，以防止服务器关闭其连接。如果服务器在 TTL 时间内没有接收任何数据包，它将自动关闭与该连接相关的服务器中的所有会话。

如果您使用 Jakarta Messaging，连接 TTL 由 ActiveMQConnectionFactory 实例上的 ConnectionTTL 属性定义，或者如果您部署 Jakarta Messaging 连接工厂实例直接在服务器端 JNDI 中，您可以使用 connectionTtl 参数在 xml 配置中指定它。

基于网络的连接（如 http-connector） 上的 ConnectionTTL 的默认值为 60000，即 1 分钟。内部连接上的连接 TTL 的默认值（如 in-vm 连接）为 -1。ConnectionTTL 的值为 -1 表示服务器端的连接永远不会超时。

如果您不希望客户端指定自己的连接 TTL，您可以在服务器端设置全局值。这可以通过在服务器配置中指定 connection-ttl-override 属性来完成。connection-ttl-override 的默认值为 -1，表示"不覆盖"，即允许客户端使用自己的值。

关闭核心会话或 Jakarta 消息传递连接

所有核心客户端会话和 Jakarta 消息传递连接在您使用结束时始终在 最后一个 块中明确关闭非常重要。

如果您未能这样做，JBoss EAP 将在垃圾收集时检测到这一点。然后，它将关闭连接并记录类似如下的警告：

```
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession] I'm closing a ClientSession you left open. Please make sure you close all ClientSessions explicitly before let
ting them go out of scope!
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession] The session you didn't close was created here:
java.lang.Exception
```

```
   at org.apache.activemq.artemis.core.client.impl.DelegatingSession.<init>
(DelegatingSession.java:83)
   at org.acme.yourproject.YourClass (YourClass.java:666)
```

请注意，如果您使用 Jakarta 消息传递，则警告将涉及 Jakarta 消息传递连接，而不是客户端会话。另外，日志还会告诉您没有关闭的 Jakarta Messaging 连接或核心客户端会话被实例化的代码行。这样，您可以找出代码中的错误并相应地进行更正。

从客户端幻灯片检测故障

只要客户端从服务器接收数据，它将认为连接处于活动状态。如果客户端没有接收任何 客户端-failure-check-period 毫秒的任何数据包，它将考虑连接失败，并将启动故障转移，或调用任何 Failure Listener 实例，如果您使用 Jakarta Messaging，则代表 ExceptionListener 实例。

如果您使用的是 Jakarta Messaging，行为由 ActiveMQConnectionFactory 实例上的 ClientFailureCheckPeriod 属性定义。

网络连接中的客户端失败检查周期（如 HTTP 连接）的默认值为 30000 或 30 秒。in-vm 连接中的客户端失败检查期间的默认值为 -1。值 -1 表示客户端端的连接永远不会失败（如果没有从服务器收到任何数据）。无论连接类型如何，检查周期通常都比服务器上连接 TTL 的值小得多，以便客户端能够在发生暂时故障时重新连接。

配置异步连接执行

服务器端收到的大多数数据包都在 远程线程 中执行。这些数据包代表短期运行的操作，出于性能的原因，始终在远程 线程 中执行。

但是，默认情况下，某些类型的数据包是使用来自线程池中的线程来执行的，以便远程 线程 不会太长时间关联。请注意，在另一个线程上异步处理操作会增加延迟。这些数据包是：

```
org.apache.activemq.artemis.core.protocol.core.impl.wireformat.RollbackMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionCloseMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionCommitMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionXACommitMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionXAPrepareMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionXARollbackMessage
```

要禁用异步连接执行，请将 async-connection-execution-enabled 参数设置为 false。默认值为 true。

30.9. 客户端重新连接和会话附加

JBoss EAP 消息传递客户端可以配置为在客户端和服务器之间的连接中检测到故障时自动重新连接或重新连接到服务器。

透明会话附加

如果故障是由于临时网络中断等临时原因造成，并且目标服务器没有重新启动，则该会话仍将存在于该服务器上，假设客户端未断开连接，超过 connection-ttl 的值。请参阅 检测检测连接。

在这种情况下，在进行重新连接时，JBoss EAP 将自动将客户端会话重新关联到服务器会话。100% 透明地完成此操作，客户可以完全像没有发生任何情况一样继续操作。

当 JBoss EAP 消息传递客户端向服务器发送命令时，它们将各个发送的命令存储在内存中缓冲区中。当连接失败并且客户端随后尝试重新连接到同一服务器时，作为重新附加协议的一部分，服务器会为客户端提供其成功收到的最后一个命令的 id。

如果客户端发送的命令超过故障转移前收到的命令，它可以重播其缓冲区中发送的任何命令，以便客户端和服务器可以协调其状态。

此缓冲区的大小由firm WindowSize 属性设置。当服务器收到 Confirm WindowSize bytes 字节并处理后，它会向客户端发回一个命令确认，客户端随后可以在缓冲区中释放空间。

如果您使用服务器上的 Jakarta Messaging 服务将 Jakarta Messaging 连接工厂实例加载到 JNDI 中，可以通过设置所选 connection-factory 的firm -window-size 属性在服务器配置中配置此属性。如果您使用 Jakarta Messaging 但不使用 JNDI，则您可以使用适当的 setter 方法直接在 ActiveMQConnectionFactory 实例上设置这些值，setConfirmationWindowSize。如果您使用核心 API，Server Locator 实例也公开了 setConfirmationWindowSize 方法。

将 confirmWindowSize 设置为 -1 （也是默认设置）会禁用任何缓冲，并防止发生任何重新附加，从而迫使重新连接。

会话重新连接

或者，服务器可能已在崩溃后实际重新启动，或者可能已被停止。在这种情况下，服务器上不再存在任何会话，且无法透明地重新连接它们。

在这种情况下，JBoss EAP 将自动重新连接连接，并在服务器上重新创建与客户端上的会话和使用者对应的任何会话和使用者。此过程与故障转移到备份服务器时所发生的情况完全相同。

客户端重新连接也由核心网桥等组件在内部使用，以允许它们重新连接到其目标服务器。

请参阅 自动客户端故障部分，以深入了解如何在重新连接期间重新连接转换 和非转换会话，以及维护一次且仅一次交付保证所需的操作。

配置重新连接属性

通过设置以下属性来配置客户端重新连接：

- retryInterval.如果与目标服务器的连接失败，此可选参数设置后续重新连接尝试之间以毫秒为单位的句点。默认值为 2000 毫秒。

- retryIntervalMultiplier.此可选参数设置一个倍数，以应用到上次重试到下一次重试的时间。这可让您在重试尝试之间实施指数回退。

  例如，如果您将 retryInterval 设置为 1000 毫秒，并将 retryIntervalMultiplier 设置为 2.0，那么如果第一次重新连接尝试失败，客户端将等待 1000 毫秒，然后在后续重新连接尝试之间等待 4000 毫秒。

  默认值为 1.0 表示 每次重新连接尝试的间隔相等。

- maxRetryInterval.此可选参数设置要使用的最大重试间隔。在设置 retryIntervalMultiplier 时，后续重试可能会按指数增加，从而劫持巨大的值。通过设置此参数，您可以为该值设置上限。默认值为 2000 毫秒。

- reconnectAttempts.此可选参数设置在放弃和关闭前重新连接尝试的总数。值 -1 表示无限数量的尝试。默认值为 0。

如果您在客户端上使用 Jakarta Messaging 和 JNDI 来查找 Jakarta Messaging 连接工厂实例，则可以在 JNDI 上下文环境中指定这些参数。例如，您的 jndi.properties 文件可能类似如下：

```
java.naming.factory.initial = ActiveMQInitialContextFactory
connection.ConnectionFactory=tcp://localhost:8080?
retryInterval=1000&retryIntervalMultiplier=1.5&maxRetryInterval=60000&reconnectAttempts=1000
```

如果您使用 Jakarta Messaging，但直接实例化 Jakarta Messaging 连接工厂，您可以在创建后立即在 ActiveMQConnectionFactory 上使用适当的 setter 方法指定参数。

如果您使用核心 API 并直接实例化 ServerLocator 实例，您也可以在创建后立即使用 ServerLocator 上的适当 setter 方法指定参数。

如果您的客户端能够成功重新连接，但会话在服务器上不再可用，例如，如果服务器重新启动或超时，客户端将无法重新连接，并且将调用连接或会话中注册的任何 Exception Listener 或 FailureListener 实例。

ExceptionListeners 和 SessionFailureListeners

请注意，当客户端重新连接或重新连接时，将调用任何注册的 Jakarta Messaging ExceptionListener 或 core API SessionFailureListener。

第 31 章 资源适配器

Jakarta Connectors 资源适配器使您的应用程序能够与任何消息传递提供商通信。它配置 Jakarta EE 组件（如 MDB 和其他 Jakarta Enterprise Beans，甚至 Servlets）如何发送或接收消息。

31.1. 关于集成 ARTEMIS 资源适配器

JBoss EAP 7 包括一个集成的 Artemis 资源适配器，它使用 pooled-connection-factory 元素来配置资源适配器的出站和入站连接。

出站连接

出站连接使用 pooled-connection-factory 元素来定义，然后由 Jakarta Enterprise Beans 和 servlets 在 Jakarta EE 部署中使用该元素将消息发送到队列或主题并接收消息。因为从连接工厂创建的连接是在应用服务器范围内创建的，所以它们可以使用类似如下的应用服务器功能：

- 连接池
- 使用应用服务器定义的安全域进行身份验证
- 使用事务管理器参与 XA 交易

这是 pooled-connection-factory 的主要区别，因为这些功能不适用于 InVmConnectionFactory 或 RemoteConnectionFactory 等基本连接 事实。另外，请注意，通过使用 pooled-connection-factory 定义的连接工厂，无法从外部独立 Jakarta 消息传递客户端使用 JNDI 进行查找。

入站连接

入站连接仅由消息驱动型 Bean(MDB)用于从队列或主题接收消息。MDB 是侦听队列或主题的无状态会话 Bean。它们必须实施公共 onMessage（消息） 方法，该方法在消息发送到队列或主题时调用。Artemis 资源适配器负责从队列或主题接收消息，并将它传递到 onMessage（Message 消息） 方法。为此，它会配置入站连接，定义集成 Artemis 服务器的位置和其他一些元素。

每个 MDB 会话 bean 使用来自客户端线程池的线程来消耗来自目的地的消息。如果未定义最大池大小，则将其确定为 CPU 内核处理器的八(8)倍。对于有许多 MDB 会话的系统，如测试套件，这可能会导致线程耗尽，并强制 MDB 从池中等待一个空闲的线程。您可以使用管理 CLI 增加客户端线程池的最大池大小。以下命令将最大客户端线程池大小设置为 128。

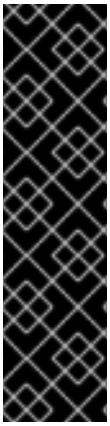
```
/subsystem=messaging-activemq:write-attribute(name=global-client-thread-pool-max-size,value=128)
```

-

有关如何配置客户端线程池大小的详情，请参考 客户端线程管理。如需有关 MDB 的更多信息，*请参阅 JBoss EAP 开发 Jakarta 企业 Bean 的消息驱动* Bean。

31.2. 使用集成 ARTEMIS 资源适配器远程连接

JBoss EAP 包括一个资源适配器，可用于连接其集成的 ActiveMQ Artemis 消息服务器。默认情况下，messaging- activemq 子系统中定义的 pooled- connection-factory 使用适配器来进行连接。但是，您也可以使用相同的资源适配器来连接在 JBoss EAP 远程实例中运行的 Artemis 服务器。



重要

在 messaging-activemq 子系统中默认配置的 activemq- ra 池连接工厂分配有 java:jboss/DefaultJMSConnectionFactory 条目。messaging-activemq 子系统需要此条目。如果您决定删除 activemq-ra 池连接工厂，您必须将此条目分配到不同的连接工厂。否则，您将在服务器登录部署中看到以下错误：

```
WFLYCTL0412: Required services that are not installed:" =>
["jboss.naming.context.java.jboss.DefaultJMSConnectionFactory"]
```

若要连接在 JBoss EAP 远程实例中运行的 Artemis 服务器，请 按照以下步骤创建新的 pooled-connection-factory ：

1. 创建一个指向远程消息传递服务器的出站套接字绑定：

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-
binding=remote-server:add(host=<server host>, port=8080)
```

2. 创建远程连接器，引用在第 1 步中创建的 outbound-socket-binding。

```
/subsystem=messaging-activemq/server=default/http-connector=remote-http-
connector:add(socket-binding=remote-server,endpoint=http-acceptor)
```

3. 创建一个 pooled-connection-factory，引用在第 2 步中创建的 remote-connector。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=remote-
artemis:add(connectors=[remote-http-connector], entries=[java:/jms/remoteCF])
```

注意

Artemis 1.x 需要目标名称上的前缀（jms.topic 用于主题，jms.queue 用于队列）。Artemis 2.x 不需要前缀，但为了与 Artemis 1.x 兼容，EAP 仍然添加前缀并指示 Artemis 在兼容性模式下运行。如果您连接到远程 Artemis 2.x 服务器，它可能不会处于兼容模式，您可能不需要前缀。在不使用前缀的目的地时，您可以通过将属性 enable-amq1-prefix 设置为 false，将连接工厂配置为不包含前缀。

将 MDB 配置为使用 pooled-connection-factory

在将 pooled-connection-factory 配置为连接到远程 Artemis 服务器后，必须先使用 pooled-connection-factory 资源的名称为来自远程服务器的 Message-Driven Beans (MDB)注释掉 @ResourceAdapter 注释。

```
import org.jboss.ejb3.annotation.ResourceAdapter;

@ResourceAdapter("remote-artemis")
@MessageDriven(name = "MyMDB", activationConfig = { ... })
public class MyMDB implements MessageListener {
    public void onMessage(Message message) {
        ...
    }
}
```

如果 MDB 需要发送消息到远程服务器，则必须使用其中一个 JNDI 条目 查找 pooled-connection-factory 来注入 pooled-connection-factory。

```
@Inject
@JMSConnectionFactory("java:/jms/remoteCF")
private JMSContext context;
```

配置 Jakarta 消息传递目的地

MDB 还必须指定它将使用消息的目的地。执行此操作的标准方法是定义与本地服务器上的 JNDI 查找对应的 destinationLookup 激活配置属性。

```
@ResourceAdapter("remote-artemis")
@MessageDriven(name = "MyMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue =
"myQueue"),
    ...
})
public class MyMDB implements MessageListener {
    ...
}
```

如果本地服务器不为远程 Artemis 服务器包含 JNDI 绑定，请指定目的地名称，如远程 Artemis 服务器中配置，使用 目标 激活配置属性，并将 useJNDI 激活配置属性设置为 false。这指示 Artemis 资源适配器自动创建 Jakarta 消息目的地，而无需查找 JNDI。

```
@ResourceAdapter("remote-artemis")
@MessageDriven(name = "MyMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "myQueue"),
    ...
})
public class MyMDB implements MessageListener {
    ...
}
```

在上例中，激活配置属性将 MDB 配置为使用来自远程 Artemis 服务器上托管的名为 myQueue 的 Jakarta Messaging Queue 的消息。在大多数情况下，MDB 不需要查找其他目的地来处理已使用的消息，如果消息上定义了 JMSReplyTo 目标，它也可使用它。

如果 MDB 需要远程服务器上定义的任何其他 Jakarta 消息传递目的地，则必须使用客户端 JNDI。如需更多信息，请参阅 连接到服务器。

31.3. 配置 ARTEMIS 资源适配器以连接到红帽 AMQ

您可以配置集成的 Artemis 资源适配器，以连接到红帽 AMQ 7 的远程安装，然后成为 JBoss EAP 7.4 应用的 Jakarta 消息传递提供商。这使得 JBoss EAP 能够成为远程红帽 AMQ 7 服务器的客户端。

如果需要支持其他消息传递协议，如 AMQP 或 STOMP，您必须将 Red Hat AMQ 7 配置为您的消息传递代理。然后，JBoss EAP 服务器中集成的 Artemis 资源适配器可用于处理已部署应用的消息。

集成资源适配器的限制

队列和主题的动态创建

请注意，集成在 JBoss EAP 7.4 中的 Artemis 资源适配器不支持在红帽 AMQ 7 代理中动态创建队列和主题。您必须直接在远程 Red Hat AMQ 7 服务器上配置所有队列和主题目的地。

创建连接事实

尽管红帽 AMQ 允许同时使用 pooled-connection-factory 和 external- context 配置连接工厂，但每个连接工厂的创建方式会有差别。当使用 external-context 创建连接工厂时，它会创建 Jakarta Messaging 连接工厂，如 Jakarta Messaging 规范 中所定义。新创建的连接工厂等同于 RemoteConnectionFactory，它在 messaging-activemq 子系统中默认定义。此连接工厂独立于应用服务器中的其他组件，这意味着它不知道并且无法使用其他组件，如交易管理器或安全管理器。因此，只有 pooled-connection-factory 可用于在 JBoss EAP 7 中创建连接工厂。external-context 只能用于将已

在远程 AMQ 7 代理上配置的 Jakarta 消息传递目的地注册到 JBoss EAP 7 服务器的 JNDI 树中，以便本地部署可以对其进行查找或注入。

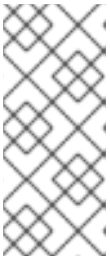
通过配置 external-context 或 connection- factory 元素所创建的连接工厂无法用于连接到远程 AMQ 7 代理，因为它们不使用 Artemis 资源适配器。在连接远程 AMQ7 代理时，仅支持配置 pooled-connection-factory 元素所创建的连接工厂。

配置 JBoss EAP 以使用远程红帽 AMQ 服务器

您可以按照以下步骤使用管理 CLI 将 JBoss EAP 配置为使用 Red Hat AMQ 7 的远程安装作为消息传递提供程序：

1. 在 Red Hat AMQ 7 broker.xml 部署描述符文件中配置队列。

```
<configuration xmlns="urn:activemq"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="urn:activemq /schema/artemis-configuration.xsd">

   <core xmlns="urn:activemq:core" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="urn:activemq:core ">
         ...
         <acceptors>
            <acceptor name="netty-acceptor">tcp://localhost:61616?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.
            </acceptor>
         </acceptors>
         <addresses>
            <address name="MyQueue">
               <anycast>
                  <queue name="MyQueue" />
               </anycast>
            </address>
            <address name="MyOtherQueue">
               <anycast>
                  <queue name="MyOtherQueue" />
               </anycast>
            </address>
            <address name="MyTopic">
               <multicast/>
            </address>
         <addresses>
         ...
   </core>
</configuration>
```



注意

JBoss EAP 随附的 Artemis 资源适配器使用 ActiveMQ Artemis Jakarta Messaging Client 2.x。此客户端需要地址上的 anycastPrefix 和多 播Prefix 前缀。它还预期队列名称与地址名称相同。

2. 创建远程连接器。

```
/subsystem=messaging-activemq/remote-connector=netty-remote-throughput:add(socket-binding=messaging-remote-throughput)
```

这会在 **messaging -activemq** 子系统中创建以下 **remote- connector** ：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    ...
    <remote-connector name="netty-remote-throughput" socket-binding="messaging-remote-throughput"/>
    ...
</subsystem>
```

3. 添加远程目的地出站套接字绑定。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=messaging-remote-throughput:add(host=localhost, port=61616)
```

这会在 **outbound- socket -binding** 元素配置中创建以下 **remote- destination** ：

```
<outbound-socket-binding name="messaging-remote-throughput">
    <remote-destination host="localhost" port="61616"/>
</outbound-socket-binding>
```

4. 为远程连接器添加池连接工厂。

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-remote:add(transaction=xa,entries=[java:/RemoteJmsXA, java:jboss/RemoteJmsXA],connectors=[netty-remote-throughput])
```

这会在 **messaging -activemq** 子系统中创建以下 **pooled-connection- factory** ：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    ...
    <pooled-connection-factory name="activemq-ra-remote" entries="java:/RemoteJmsXA
java:jboss/RemoteJmsXA" connectors="netty-remote-throughput"/>
    ...
</subsystem>
```

注意

Artemis 1.x 需要目标名称上的前缀（jms.topic 用于主题，jms.queue 用于队列）。Artemis 2.x 不需要前缀，但为了与 Artemis 1.x 兼容，EAP 仍然添加前缀并指示 Artemis 在兼容性模式下运行。如果您连接到远程 Artemis 2.x 服务器，它可能不会处于兼容模式，您可能不需要前缀。在不使用前缀的目的地时，您可以通过将属性 enable-amq1-prefix 设置为 false，将连接工厂配置为不包含前缀。

5. 为队列和主题创建 外部上下文 绑定。

```
/subsystem=naming/binding=java\:global\/remoteContext:add(binding-type=external-context,
class=javax.naming.InitialContext, module=org.apache.activemq.artemis, environment=
[java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory,
java.naming.provider.url=tcp://127.0.0.1:61616, queue.MyQueue=MyQueue,
queue.MyOtherQueue=MyOtherQueue, topic.MyTopic=MyTopic])
```

这会在 **naming** 子系统 中创建以下外部上下文 绑定：

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    ...
    <bindings>
        <external-context name="java:global/remoteContext"
module="org.apache.activemq.artemis" class="javax.naming.InitialContext">
            <environment>
                <property name="java.naming.factory.initial"
value="org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory"/>
                <property name="java.naming.provider.url" value="tcp://127.0.0.1:61616"/>
                <property name="queue.MyQueue" value="MyQueue"/>
                <property name="queue.MyOtherQueue" value="MyOtherQueue"/>
                <property name="topic.MyTopic" value="MyTopic"/>
            </environment>
        </external-context>
    </bindings>
    ...
</subsystem>
```

6. 通过将 JNDI 名称设置为红帽 AMQ 7 地址名称值，为 Jakarta Messaging 队列和主题创建查找条目。这会在 JNDI 名称和 Red Hat AMQ 7 地址名称之间创建一个映射。

```
/subsystem=naming/binding=java\:\/MyQueue:add(lookup=java:global/remoteContext/MyQueue
,binding-type=lookup)
/subsystem=naming/binding=java\:\/MyOtherQueue:add(lookup=java:global/remoteContext/My
OtherQueue,binding-type=lookup)
/subsystem=naming/binding=java\:\/MyTopic:add(lookup=java:global/remoteContext/MyTopic,bi
nding-type=lookup)
```

这会在 naming 子系统中创建以下 查找 配置：

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    ...
    <lookup name="java:/MyQueue" lookup="java:global/remoteContext/MyQueue"/>
    <lookup name="java:/MyOtherQueue"
lookup="java:global/remoteContext/MyOtherQueue"/>
    <lookup name="java:/MyTopic" lookup="java:global/remoteContext/MyTopic"/>
    ...
</subsystem>
```

另外，也可定义 /subsystem=messaging-activemq/external-jms-queue 或 /subsystem=messaging-activemq/external-jms-topic 资源，而不配置 naming 子系统。例如：

```
/subsystem=messaging-activemq/external-jms-queue=MyQueue:add(entries=
[java:/MyQueue])
```

这会创建以下资源：

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    ...
    <external-jms-queue name="MyQueue" entries="java:/MyQueue"/>
    ...
</subsystem>
```

注意

external-jms-queue 资源不提供队列管理和统计数据的操作。

JBoss EAP 现已配置为使用红帽 AMQ 7 的远程安装作为消息传递提供商。

31.4. 适用于远程 ARTEMIS 的远程代理的 JAKARTA 消息传递资源配置

在管理 CLI 中，您可以使用 @JMSConnectionFactoryDefinition 注释或

@JMSDestinationDefinition 注释，为基于 Artemis 的远程代理（如红帽 AMQ 7）配置 Jakarta 消息传递资源。您也可以从管理控制台配置资源。

远程 ActiveMQ 服务器资源不需要本地实例 Artemis。这有助于减少 JBoss EAP 映像的内存占用和 CPU 占用。

31.4.1. 使用 JMSConnectionFactoryDefinition 注解配置的 Jakarta 消息传递资源配置

JBoss EAP 使用 @JMSConnectionFactoryDefinition 注释来定义连接工厂。此连接工厂可以连接到本地或远程 Artemis 代理。@JMSConnectionFactoryDefinition 注释的 resourceAdapter 元素随后引用 messaging- subsystem 中定义的 pooled-connection-factory 的名称，该名称可以连接到远程 Artemis 代理。resourceAdapter 元素定义用于创建连接工厂的资源适配器，或者定义连接工厂的资源适配器。

当 @JMSConnectionFactoryDefinition 注释中未定义 resourceAdapter 元素时，messaging-activemq 子系统将默认使用连接工厂的 JNDI 名称。这称为默认绑定。默认绑定使用 /subsystem=ee/service=default -bindings 中的jms-connection- factory 属性来定义。如果指定了 resourceAdapter 元素，或者可以从 jms-connection-factory 的默认绑定定义 resourceAdapter 元素，如果 它是一个 pooled-connection-factory 到远程代理，您可以使用它连接到远程代理。

如果 messaging-activemq 子系统中没有定义 resourceAdapter，或者无法从 jms-connection-factory 的默认绑定获取资源，则创建 Jakarta Messaging 资源的任务将委派给 resource-adapters 子系统，该 子系统基于资源适配器的 admin-objects 和 connection-definitions 资源。

以下小节提供了如何配置和使用 @JMSConnectionFactoryDefinition 注释的示例。

使用默认资源适配器配置 @JMSConnectionFactory 定义

1. 创建到远程实例的套接字绑定：

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=messaging-remote-throughput:add(host=127.0.0.2, port=5445)
```

2. 创建连接器：

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-binding="messaging-remote-throughput")
```

3. 创建池连接工厂：

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-
remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-
amq"]
```

4. 为 ee 子系统定义默认的 Jakarta Messaging 连接工厂：

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory,
value="java:/jms/remote-amq/JmsConnectionFactory")
```

5. 在应用程序代码中使用 @JMSConnectionFactoryDefinition 注释：

```
@JMSConnectionFactoryDefinition(name="java:/jms/remote-
amq/JmsConnectionFactory")
```

使用远程 Artemis Broker 配置 @JMSConnectionFactory 定义

1. 创建连接器：

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-
binding="messaging-remote-throughput")
```

2. 创建池连接工厂：

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-
remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-
amq"])
```

3. 为 ee 子系统定义默认的 Jakarta Messaging 连接工厂：

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory,
value="java:/jms/remote-amq/JmsConnectionFactory")
```

4. 在应用程序代码中使用 @JMSConnectionFactoryDefinition 注释：

```
@JMSConnectionFactoryDefinition(
    name="java:app/myCF"
    resourceAdapter="myPCF"
)
```

使用第三方 JMS 资源适配器配置 @JMSConnectionFactory 定义

1. 创建连接器：

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-
binding="messaging-remote-throughput")
```

2. 创建池连接工厂：

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-
remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-
amq"])
```

3. 为 ee 子系统定义默认的 Jakarta Messaging 连接工厂：

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory,
value="java:/jms/remote-amq/JmsConnectionFactory")
```

4. 在应用程序代码中使用 @JMSConnectionFactoryDefinition 注释：

```
@JMSConnectionFactoryDefinition(
    name="java:app/myCF"
    resourceAdapter="wsmq"
)
```

31.4.2. 使用 JMSDestinationDefinition 注解配置 Jakarta 消息传递资源配置

您可以使用 server 资源为本地代理 pooled-connection-factory 创建所需的目的地。

如果 resourceAdapter 元素指向 pooled-connection-factory 名称，并且它在本地代理中定义，例如 /subsystem/messaging-activemq/server=default，则它会在本地 Artemis 代理中创建目的地。

注意

如果您需要在基于 Artemis 的远程代理中创建目的地，则必须在 messaging - activemq 子系统中定义 pooled-connection- factory。

如果 @JMSDestinationDefinition 注释 中设置 的 resourceAdapter 元素与 messaging-activemq

子系统中为 服务器 定义的 resourceAdapter 元素匹配，则在此代理中创建目的地，无论 pooled-connection-factory 中的连接器指向本地还是远程 Artemis 代理。

使用 JMSDestinationDefinition 注解配置 Jakarta 消息传递资源

1. 创建连接器：

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-binding="messaging-remote-throughput")
```

2. 创建池连接工厂：

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-amq"])
```

3. 为 ee 子系统定义默认的 Jakarta Messaging 连接工厂：

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory, value="java:/jms/remote-amq/JmsConnectionFactory")
```

4. 在应用程序代码中使用 @JMSDestinationDefinition 注释：

```
@JMSDestinationDefinition(
    name = "java:/jms/queue/MessageBeanQueue",
    interfaceName = "javax.jms.Queue",
    destinationName = "MessageBeanQueue"
    properties= {
        "management-address=remote-activemq.management"
    }
)
```

31.4.3. 使用管理控制台配置远程 ActiveMQ 服务器资源

您可以从管理控制台配置以下远程 ActiveMQ 服务器资源：

- 通用连接器
- 在 VM Connector 中

- HTTP 连接器
- 远程连接器
- 发现组
- 连接事实
- 池的连接事实
- 外部 JMS Queue
- 外部 JMS 主题

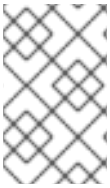
从管理控制台配置远程 ActiveMQ 服务器资源：

1. 访问管理控制台，再导航到 Configuration → Subsystems → Messaging(ActiveMQ) → Remote ActiveMQ Server 并点 View。
2. 在导航窗格中，点击您要配置的资源。

31.5. 部署红帽 JBOSS A-MQ 资源适配器

您可以部署红帽 JBoss A-MQ 产品提供的资源适配器，例如红帽 JBoss A-MQ 6.3.0，已成为 JBoss EAP 的外部 Jakarta 消息传递提供商。

有关如何部署和配置红帽 *JBoss A-MQ 资源适配器的详细信息，请参阅红帽 JBoss A-MQ 文档套件中的 JBoss 企业应用平台 安装 ActiveMQ* 资源适配器。



注意

请注意，在 7.x 版本中，6.x 版本中的产品名称从红帽 JBoss A-MQ 改为红帽 AMQ。

31.5.1. 红帽 JBoss A-MQ 6 资源适配器的问题

- JBoss EAP 将跟踪和监控应用程序，寻找未关闭的资源。尽管在很多情况下，此类监控可能会在应用尝试以单一方法重新使用已关闭的 UserTransaction 实例时造成意外行为。如果应用程序以这种方式重新使用连接，在配置 Red Hat JBoss A -MQ 资源适配器时，将属性 tracking="false" 添加到 <connection-definition/> 元素 中。

```
<connection-definition class-name="..." tracking="false" ... />
```

- 红帽 JBoss A-MQ 6 资源适配器不从 Narayana API 实施 XAResourceWrapper，后者由 JBoss EAP 使用。因此，当交易管理器向所有 XA 事务参与者发送提交并在等待回复时崩溃时，它将无限期地记录警告，直到从对象存储中删除提交的事务的记录。

- 当提交方法协议调用期间发生错误（如网络断开连接）时，红帽 JBoss A-MQ 6 资源适配器返回代码 XAER_RMERR。此行为会破坏 XA 规范，因为正确的返回代码应当是 XAER_RMFAIL 或 XAER_RETRY。因此，在消息代理端，事务处于未知状态，在某些情况下可能会导致数据不一致。当返回意外的错误代码时，将记录类似于以下的消息：

```
WARN [com.arjuna.ats.jtax] ...: XAResourceRecord.rollback caused an XA error:
ARJUNA016099: Unknown error code:0 from resource ... in transaction ...:
javax.transaction.xa.XAException: Transaction ... has not been started.
```

- 红帽 JBoss A-MQ 6.x 支持 Java EE 6 中包含的 JMS 1.1 规范。它不支持 Jakarta EE 8 中引入的 Jakarta Messaging 2.0 规范，并在 JBoss EAP 7 中受到支持。如果您需要发送消息到远程红帽 JBoss A-MQ 代理，您必须在应用程序代码中使用 JMS 1.1 API。有关红帽 JBoss A-MQ 6.x 支持标准的更多信息，请参阅 Red Hat JBoss A-MQ 支持的标准和协议。

31.6. 部署 IBM MQ 资源适配器

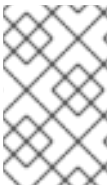
关于 IBM MQ

IBM MQ 是来自 IBM 的消息传递导向型中间件(MOM)产品，允许分布式系统上的应用程序相互通信。这通过使用消息和消息队列来完成。IBM MQ 负责发送消息到消息队列，以及使用消息通道将数据传输到其他队列管理器。有关 IBM MQ 的更多信息，请参阅 IBM 产品网站上的 IBM MQ。

概述

IBM MQ 可以配置为 JBoss EAP 7.4 的外部 Jakarta 消息传递提供商。本节介绍在 JBoss EAP 中部署和配置 IBM MQ 资源适配器的步骤。此部署和配置可使用管理 CLI 工具或基于 Web 的管理控制台来完

成。有关 IBM MQ 支持 配置的最新信息，请参阅 JBoss EAP 支持的配置。



注意

您必须在配置 IBM MQ 资源适配器后重启您的系统，使配置更改生效。

先决条件

开始之前，您必须验证 IBM MQ 资源适配器的版本并了解其配置属性。

- IBM MQ 资源适配器作为名为 wmq.jmsra.rar 的 Resource Archive(RAR)文件提供。您可以从 /opt/ mqm/java/lib/jca/wmq.jmsra.rar 获取 w mq.jmsra.rar 文件。如需有关每个 JBoss EAP 版本支持的特定版本的信息，请参阅 JBoss EAP 支持的配置。
- 您必须知道以下 IBM MQ 配置值：有关这些值的详情，请参阅 IBM MQ 产品文档。
  - *MQ_QUEUE_MANAGER* ：IBM MQ 队列管理器的名称
  - *MQ_HOST_NAME* ：用于连接到 IBM MQ 队列管理器的主机名
  - *MQ_CHANNEL_NAME* ：用于连接到 IBM MQ 队列管理器的服务器频道
  - *MQ_QUEUE_NAME* ：目标队列的名称
  - *MQ_TOPIC_NAME* ：目标主题的名称
  - *MQ_PORT* ：用于连接到 IBM MQ 队列管理器的端口
  - *MQ_CLIENT* ：传输类型
- 对于出站连接，还必须熟悉以下配置值：

- ***MQ_CONNECTIONFACTORY_NAME*** **：提供与远程系统连接的连接工厂实例的名称**

流程

注意

以下是 IBM 提供的默认配置，可能随时更改。如需更多信息，请参阅 IBM MQ 文档。

1. **首先，通过将 wmq.jmsra.rar 文件复制到 *EAP_HOME*/standalone/deployments/ 目录中来手动部署资源适配器。**

2. **接下来，使用管理 CLI 添加资源适配器并进行配置。**

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar:add(archive=wmq.jmsra.rar,
transaction-support=XATransaction)
```

请注意，transaction-support 元素被设置为 XATransaction。在使用事务时，请务必提供 XA 恢复数据源的安全域，如下例所示。

```
/subsystem=resource-adapters/resource-adapter=test/connection-definitions=test:write-
attribute(name=recovery-security-domain,value=myDomain)
```

有关 XA 恢复的更多信息，请参阅 JBoss EAP *配置指南中的配置* XA 恢复。

对于非事务部署，将 事务支持 的值更改为 NoTransaction。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar:add(archive=wmq.jmsra.rar,
transaction-support=NoTransaction)
```

3. **现在创建了资源适配器，您可以在其中添加必要的配置元素。**

   a. **为队列添加一个 admin-object，并配置其属性。**

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=queue-
ao:add(class-name=com.ibm.mq.connector.outbound.MQQueueProxy, jndi-
name=java:jboss/MQ_QUEUE_NAME)
```

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueName:add(value=MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueManagerName:add(value=MQ_QUEUE_MANAGER)
```

b. **为主题添加一个 admin-object，并配置其属性。**

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=topic-ao:add(class-name=com.ibm.mq.connector.outbound.MQTopicProxy, jndi-name=java:jboss/MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=topic-ao/config-properties=baseTopicName:add(value=MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=topic-ao/config-properties=brokerPubQueueManager:add(value=MQ_QUEUE_MANAGER)
```

c. **为受管连接工厂添加连接定义，并配置其属性。**

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd:add(class-name=com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl, jndi-name=java:jboss/MQ_CONNECTIONFACTORY_NAME, tracking=false)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=hostName:add(value=MQ_HOST_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=port:add(value=MQ_PORT)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=channel:add(value=MQ_CHANNEL_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=transportType:add(value=MQ_CLIENT)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=queueManager:add(value=MQ_QUEUE_MANAGER)
```

4. **如果要将 JBoss EAP 中 EJB3 消息传递系统的默认提供程序从 JBoss EAP 7 消息传递更改为 IBM MQ，请使用管理 CLI 来修改 ejb3 子系统，如下所示：**

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-name,value=wmq.jmsra.rar)
```

5. **在 MDB 代码 中配置 @ActivationConfigProperty 和 @ResourceAdapter 注释，如下所**

示：

```
@MessageDriven(name="IbmMqMdb", activationConfig = {
   @ActivationConfigProperty(propertyName = "destinationType",propertyValue =
"javax.jms.Queue"),
   @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
   @ActivationConfigProperty(propertyName = "hostName", propertyValue =
"MQ_HOST_NAME"),
   @ActivationConfigProperty(propertyName = "port", propertyValue = "MQ_PORT"),
   @ActivationConfigProperty(propertyName = "channel", propertyValue =
"MQ_CHANNEL_NAME"),
   @ActivationConfigProperty(propertyName = "queueManager", propertyValue =
"MQ_QUEUE_MANAGER"),
   @ActivationConfigProperty(propertyName = "destination", propertyValue =
"MQ_QUEUE_NAME"),
   @ActivationConfigProperty(propertyName = "transportType", propertyValue =
"MQ_CLIENT")
})

@ResourceAdapter(value = "wmq.jmsra-VERSION.rar")
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class IbmMqMdb implements MessageListener {
}
```

务必将 @ResourceAdapter 值中的 *VERSION* 替换为 RAR 名称中的实际版本。

6. 激活资源适配器：

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar:activate()
```

31.6.1. IBM MQ 资源适配器的限制和已知问题

下表列出了 IBM MQ 资源适配器的已知问题。version 列中的复选标记(Auditor)表示该资源适配器版本存在问题。

表 31.1. IBM MQ 资源适配器已知问题

JIRA	问题描述	IBM MQ 8	IBM MQ 9

JIRA	问题描述	IBM MQ 8	IBM MQ 9
JBEAP-503	IBM MQ 资源适配器返回 **Queue.toString（）和 Queue Browser.getQueue（）.toString（）** 方法的不同 String 值。**队列**是 **com.ibm.mq.connector.outbound.MQQueueProxy** 类的实例，它与 **QueueBrowser .htmlQueueBrowser.getQueue（）方法返回的 com.ibm.mq.jms.MQ Queue** 类不同。这些类包含 **toString（）** 方法的不同实施。请注意，您不能依赖这些 **toString（）** 方法来返回相同的值。	✔	✔
JBEAP-511,JBEAP-550, JBEAP-3686	以下限制适用于 IBM MQ 的消息属性名称。 • 在部署描述符的 **activation-config** 部分中，您不能使用 **_、& 或 |** 等特殊字符配置 **destinationName** 属性。使用这些字符会导致 MDB 部署失败，并显示 **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 异常。 • 在部署描述符的 **activation-config** 部分中，您不能使用 **java:/** 前缀配置 **destinationName** 属性。使用这个前缀会导致 MDB 部署失败，并显示 **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 异常。 • 属性不能以"JMS"或"usr.JMS"开头，因为它们被 IBM MQ Jakarta 消息传递类保留使用。IBM 知识库网站上会记录例外。 请参阅 IBM MQ、版本 8.0 和 属性名称限制(IBM MQ)、IBM 知识库网站上版本 9.0 的限制，以了解每个资源适配器的消息属性名称限制的完整列表。	✔	✔
JBEAP-549	在使用 **@ActivationConfigProperty** 注释为 MDB 指定 **目的地** 属性名称值时，您必须使用所有大写字母。例如： `@ActivationConfigProperty(propertyName = "destination", propertyValue = "QUEUE")`	✔	✔

JIRA	问题描述	IBM MQ 8	IBM MQ 9

JBEAP-624	如果使用 IBM MQ 资源适配器，以使用 **@JMSConnectionFactoryDefinition** 注释在 Jakarta EE 部署中创建连接工厂，您必须指定 **resourceAdapter** 属性。否则，部署将失败。
	
	`@JMSConnectionFactoryDefinition(`
	`  name = "java:/jms/WMQConnectionFactory",`
	`  interfaceName = "javax.jms.ConnectionFactory",`
	`  resourceAdapter = "wmq.jmsra",`
	`  properties = {`
	`  "channel=<channel>",`
	`  "hostName=<hostname_wmq_broker>",`
	`  "transportType=<transport_type>",`
	`  "queueManager=<queue_manager>"`
	`  }`
	`)`
		✔	✔

JBEAP-2339	IBM MQ 资源适配器可以在连接开始前从队列和主题读取消息。这意味着使用者可以在连接启动之前使用消息。要避免出现这个问题，请使用 **外部上下文（而非 IBM MQ 资源适配器）** 创建的远程 IBM MQ 代理创建的连接工厂。	✔	✔

JBEAP-3685	且设置了 **<transaction-support>XATransaction</transaction-support>**，**JMSContext** 始终为 **JMSContext.SESSION_TRANSACTED**，无论是使用注入还是手动创建。
	
	在下面的代码示例中，@ **JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)将** 被忽略，**JMSContext** 则保留在 **JMSContext.SESSION_TRANSACTED 中**。
	
	`@Inject`
	`@JMSConnectionFactory("jms/CF")`
	`@JMSPasswordCredential(userName="myusername",`
	`password="mypassword")`
	`@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLED`
	`GE)`
	`transient JMSContext context3;`
		✔	✔

JBEAP-14633	根据 Jakarta Messaging 规范，Queue **Session** 接口无法用于创建特定于发布/订阅域的对象，以及从 **Session** 继承的某些方法应抛出 **javax.jms.IllegalStateException**。其中一种方法是 **QueueSession.createTemporaryTopic( )**。IBM MQ 资源适配器抛出 **java. lang.NullPointerException，而不是抛出 javax.jms.IllegalState Exception**。	✔	✔

JIRA	问题描述	IBM MQ 8	IBM MQ 9
JBEAP-14634	**MQTopicProxy.getTopicName（）** 返回与 IBM MQ 代理设置的不同主题名称。例如，如果将主题名称设置为 **topic://MYTOPIC?XMSC_WMQ_BROKER_PUBQ_QMGR=QM**，**MQTopicProxy** 会返回 **topic://MYTOPIC**。	✔	✔
JBEAP-14636	**JMSContext** 的默认 **autoStart** 设置为 **false**，表示 **JMSContext** 使用的底层连接在创建使用者时不会自动启动。此设置应默认为 **true**。	✔	✔
JBEAP-14640	在使用无效凭证时，IBM MQ 资源适配器抛出 **DetailedJMSException 而不是 JMSSecurityException**，并将以下错误记录到服务器控制台。 > WARN [org.jboss.jca.core.connectionmanager.pool.strategy.PoolByCri] (EJB default - 7) IJ000604: Throwable while attempting to get a new connection: null: com.ibm.mq.connector.DetailedResourceException: MQJCA1011: Failed to allocate a {JMS} connection., error code: MQJCA1011 An internal error caused an attempt to allocate a connection to fail. See the linked exception for details of the failure. 以下是可能导致此问题的代码示例： > QueueConnection qc = queueConnectionFactory.createQueueConnection("invalidUserName", "invalidPassword");	✔	✔
JBEAP-14642	由于 **MQMessageProducer.send(Destination destination, Message message)** 和 **MQMessageProducer.send(Destination destination, Message message, int deliveryMode), int 优先级, 长 timeToLive, CompletionListener completionListener）** 方法，IBM MQ 资源适配器抛出 **JMSException**，并将以下错误消息记录到服务器控制台。 > SVR-ERROR: Expected JMSException, received com.ibm.mq.connector.outbound.MQQueueProxy cannot be cast to com.ibm.mq.jms.MQDestination 这是因为队列或主题查找中使用的 JNDI 名称为 **com.ibm.mq.connector.outbound.MQQueueProxy/MQTopicProxy**。	✔	✔
JBEAP-14643	**JMSProducer** 接口上的 **setDeliveryDelay(expDeliveryDelay)** 方法不会更改设置。调用此方法后，它将保留为 **0** 的默认设置。	✔	✔

JIRA	问题描述	IBM MQ 8	IBM MQ 9
JBEAP-14670	如果在 **UserTransaction.begin（）** 之前创建的 **QueueSession** 上完成了工作，则该工作不被视为事务的一部分。这意味着，任何使用此会话发送到队列的消息都不是由 **UserTransaction.commit（）提交**，在 **UserTransaction.rollback（）** 之后，消息仍会保留在队列中。	✔	✔
JBEAP-14675	如果您关闭了连接，然后立即创建具有相同 **clientID** 的 **JMSContext**，IBM MQ 资源适配器有时会将以下错误记录到服务器控制台。 > ERROR [io.undertow.request] (default task-1) UT005023: Exception handling request to /jmsServlet-1.0-SNAPSHOT/: com.ibm.msg.client.jms.DetailedJMSRuntimeException: MQJCA0002: An exception occurred in the IBM MQ layer. See the linked exception for details. > A call to IBM MQ classes for Java(tm) caused an exception to be thrown. 当与同一 **clientID** 连接关闭后创建新的 **JMSContext** 时，不会出现此问题。	✔	✔
JBEAP-15535	如果有状态会话 Bean 尝试在容器管理事务(CMT)中同时发送消息到主题，则发送的消息会失败并显示以下消息： > SVR-ERROR: com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a message to destination '*MDB_NAME TOPIC_NAME*' 堆栈追踪显示它是由以下异常导致的： > com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED') reason '2072' ('MQRC_SYNCPOINT_NOT_AVAILABLE')	✔	✔

JIRA	问题描述	IBM MQ 8	IBM MQ 9
JBEAP-20758	当您在 JBoss EAP 上部署 IBM MQ 8 或 IBM MQ 9 资源适配器 **wmq.jmsra.rar** 时，服务器控制台中会显示以下错误消息： WARN [org.jboss.as.connector.deployers.RADeployer] (MSC service thread 1-8) IJ020017: Invalid archive: file:/<path-to-jboss>/jboss-eap-7.4/standalone/tmp/vfs/temp/tempa02bdd5ee254e590/content-135e13d4f38704fc/contents/ IBM MQ v9.0.0.4 资源适配器已作为 JBoss EAP 7.4 的 Jakarta 消息传递提供商测试的一部分进行测试。您可以选择忽略此警告消息，或者您可以通过将 **enabled** 属性设置为 **false** 来禁用存档验证。例如： /subsystem=jca/archive-validation=archive-validation:write-attribute(name=enabled, value=false)	✔	✔

31.7. 部署 GENERIC JAKARTA 消息传递资源适配器

JBoss EAP 可以配置为与第三方 Jakarta 消息传递提供商合作；然而，并非所有 Jakarta 消息传递提供商都生产用于与 Jakarta 应用平台集成的 Jakarta 消息传递器资源适配器。此流程涵盖配置 JBoss EAP 中包含的通用 Jakarta 消息传递资源适配器以连接 Jakarta 消息传递提供商所需的步骤。在此流程中，Tibco EMS 8 用作 Jakarta 消息传递供应商的示例。其他 Jakarta 消息传递提供商可能需要不同的配置。

重要

在使用通用 Jakarta 消息传递资源适配器之前，请检查 Jakarta 消息传递提供商，了解他们自己的资源适配器可用于 JBoss EAP。只有在 Jakarta 消息传递供应商不提供自己的资源适配器时，才应使用通用 Jakarta Messaging Jakarta Connectors 资源适配器。

在配置通用资源适配器前，您需要执行以下操作：

- **您的 Jakarta 消息传递供应商服务器必须已经配置好并可供使用。提供商 Jakarta Messaging 实施所需的任何二进制文件都需要。**
- **您将需要了解下列 Jakarta 消息传递提供商属性的值，才能查找其 Jakarta 消息传递资源，如连接工厂、队列或主题。**

- java.naming.factory.initial
- java.naming.provider.url
- java.naming.factory.url.pkgs

在此流程中使用的 XML 示例中，这些参数分别写为 PROVIDER_FACTORY_INITIAL、PROVIDER_URL 和 PROVIDER_CONNECTION_FACTORY。将这些占位符替换为您的环境的 Jakarta 消息传递供应商值。

31.7.1. 配置 Generic Jakarta 消息传递资源适配器，以便第三方 Jakarta 消息传递提供商使用

1. 创建并配置资源适配器模块。

   创建 JBoss EAP 模块，其中包含连接和与 Jakarta 消息传递提供商通信所需的所有库。此模块将命名为 *org.jboss.genericjms.provider*。

   - 创建以下目录结构： *EAP_HOME*/modules/org/jboss/genericjms/provider/main
   - 将提供商 Jakarta 消息实施所需的二进制文件复制到 *EAP_HOME*/modules/org/jboss/genericjms/provider/main。

     注意

     对于 Tibco EMS，所需的二进制文件是来自 Tibco 安装的 lib 目录的 tibjms.jar 和 tibcrypt.jar。

   - 在 *EAP_HOME*/modules/org/jboss/genericjms/provider/main 中创建 module.xml 文件，以资源形式列出前面步骤中的 JAR 文件：

```
<module xmlns="urn:jboss:module:1.5" name="org.jboss.genericjms.provider">
  <resources>
    <!-- all jars required by the Jakarta Messaging provider, in this case Tibco -->
    <resource-root path="tibjms.jar"/>
    <resource-root path="tibcrypt.jar"/>
  </resources>
```

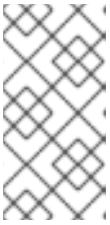
```
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.jms.api"/>
  </dependencies>
</module>
```

- 使用以下命令在 ee 子系统中添加模块：

```
/subsystem=ee:list-add(name=global-modules, value={"name" =>
"org.jboss.genericjms.provider", "slot" =>"main"}
```

2. 为 Jakarta 消息传递提供商创建和配置 Java 命名和目录接口外部上下文。

Jakarta 消息传递资源（如连接工厂和目的地）在 Jakarta 消息传递提供商中查找。在 JBoss EAP 实例中添加外部上下文，以便对此资源的任何 *本地* 查找将自动在远程 Jakarta 消息传递提供程序上查找资源。



注意

在此过程中，EAP _*HOME*/standalone/configuration/standalone-full.xml 用作 JBoss EAP 配置文件。

使用管理 CLI 创建外部 Java 命名和目录接口上下文，并包含其配置属性。以下示例中的属性应当替换为正确的值，以连接远程 Jakarta 消息传递提供程序。例如，一些 Jakarta 消息提供商（如 Tibco EMS）不支持 Java 命名和目录接口 查找(Name) 方法。在这些情况下，添加 org.jboss.as.naming.lookup.by.string 属性，值设为 true 以解决这个问题。检查适配器文档中有关必要属性及其值的信息。

```
/subsystem=naming/binding="java:global/remoteJMS":add(binding-type=external-
context,module=org.jboss.genericjms.provider,class=javax.naming.InitialContext,environment=
[java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory,java.naming.prc
vider.url=tcp://<hostname>:7222,org.jboss.as.naming.lookup.by.string=true])
```

在正确配置了外部上下文后，将对从 java:global/remoteJMS/ 起的资源的任何 Java 命名和目录接口查找在远程 Jakarta 消息传递提供商上完成。例如，如果消息驱动型 Bean 为 java:global/remoteJMS/Queue1 执行 Java 命名和目录接口查找，则外部上下文将连接到远程 Jakarta 消息传递提供程序，并且对 Queue1 资源执行查询。

或者，您也可以在查找 Java 命名和目录接口名称时，无需 使用外部上下文即可向 远程服务器发出 Java 命名和目录接口查找。为此，可使用 CLI 创建一个引用 external-context 的新绑定，如下例所示：

```
/subsystem=naming/binding=java\:\/jms\/queue\/myQueue:add(binding-type=lookup,
lookup=java:global/remoteJMS/jms/queue/myQueue)
```

在上例中，对 **java:/jms/queue/myQueue** 进行 **Java** 命名和目录接口查找的应用将定位到远程服务器上名为 **myQueue** 的队列。

3. 创建通用 **Jakarta** 消息传递资源适配器。

使用管理 **CLI** 创建资源适配器

```
/subsystem=resource-adapters/resource-adapter=generic-
ra:add(module=org.jboss.genericjms,transaction-support=XATransaction)
```

4. 配置通用 **Jakarta** 消息传递资源适配器。

使用管理 **CLI** 配置资源适配器 的连接定义 和其他元素。

```
/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
cd:add(class-name=org.jboss.resource.adapter.jms.JmsManagedConnectionFactory, jndi-
name=java:/jms/XAQCF)

/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
cd/config-properties=ConnectionFactory:add(value=XAQCF)

/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
cd/config-
properties=JndiParameters:add(value="java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory;java.naming.provider.url=tcp://<hostname>:7222")

/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
cd:write-attribute(name=security-application,value=true)
```

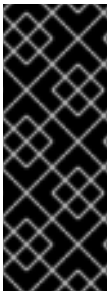
5. 配置 **ejb3** 子系统中默认的消息驱动型 **Bean** 池，以使用通用资源适配器。

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-name, value=generic-ra)
```

现在，通用 **Jakarta** 消息传递资源适配器已被配置并可供使用。以下是在创建新的消息驱动 **Bean** 时使用资源适配器的示例。

示例：使用通用资源适配器代码

```
@MessageDriven(name = "HelloWorldQueueMDB", activationConfig = {
  // The generic Jakarta Messaging resource adapter requires the  Java Naming and Directory
Interface bindings
  // for the actual remote connection factory and destination
  @ActivationConfigProperty(propertyName = "connectionFactory", propertyValue =
"java:global/remoteJMS/XAQCF"),
  @ActivationConfigProperty(propertyName = "destination", propertyValue =
"java:global/remoteJMS/Queue1"),
  @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),
  @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge") })
  public class HelloWorldQueueMDB implements MessageListener {
  public void onMessage(Message message) {
  // called every time a message is received from the _Queue1_ queue on the Jakarta
Messaging provider.
  }
}
```



重要

使用通用 Jakarta 消息传递资源适配器时，请确保您将会话设置为被转换，以避免出现潜在的 NullPointerException 错误。发生错误的原因是，当 Jakarta EE 规格指出 *未被* 处理它们时，通用 Jakarta 消息传递资源适配器尝试处理参数。这可以通过执行以下操作来实现： connection.createSession(true, Session.SESSION_TRANSACTED);

您还可以使用资源适配器的池连接工厂：

```
@Resource(lookup = "java:/jms/XAQCF")
private ConnectionFactory cf;
```

无法直接从外部上下文注入资源，但可以注入外部上下文并执行查找。例如，Tibco EMS 代理中部署的队列的查找如下所示：

```
@Resource(lookup = "java:global/remoteJMS")
private Context context;
...
Queue queue = (Queue) context.lookup("Queue1")
```

31.8. 使用资源注解

利用 @Resource 注释，Jakarta Enterprise Beans 可以直接注入 Jakarta 消息资源或连接工厂。您可以使用 @Resource 注释指定以下参数：

- lookup
- name
- mappedName

若要注入资源，您必须在其中一个参数中指定资源的 Java 命名和目录接口(JNDI)名称。

31.8.1. 注入 Jakarta 消息传递资源

1. 按如下所示定义队列：

```
<jms-queue name="OutQueue" entries="jms/queue/OutQueue
java:jboss/exported/jms/queue/OutQueue"/>
```

2. 通过在 @Resource 注释的 lookup、name 或 mapping Name 参数中指定其 Java 命名和目录接口名称 来注入此队列。例如：

```
@Resource(lookup = "java:jboss/exported/jms/queue/OutQueue")
public Queue myOutQueue;
```

31.8.2. 注入连接事实

1. 如下方所示，定义您的连接工厂。示例显示了 JmsXA 池 化连接工厂。

```
<pooled-connection-factory name="activemq-ra" entries="java:/JmsXA
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm" transaction="xa"/>
```

2. 注入默认 activemq-ra 池连接工厂，如下所示：

```
@Resource(lookup = "java:/JmsXA")
private ConnectionFactory cf;
```

31.8.3. Generic Jakarta 消息传递资源适配器的限制和已知问题

- Jakarta Messaging API 不提供创建 Jakarta 消息资源的程序化方式，仅支持 Jakarta Messaging 2.0 规范中定义的功能。有关规范的更多信息，请参阅 Jakarta Messaging 2.0 规范。

  - EE.5.18.4 Jakarta Messaging Connection Factory 资源定义

    这使得应用能够定义 Jakarta Messaging ConnectionFactory 资源。

  - EE.5.18.5 Jakarta 消息传递目标定义

    这使得应用程序能够定义 Jakarta Messaging Destination 资源。

第 32 章 后向和转发兼容性

JBoss EAP 支持与将 HornetQ 用作消息传递代理（如 JBoss EAP 6）的传统 JBoss EAP 版本向后兼容。这两种兼容性模式由 JBoss EAP 内置消息传递服务器 ActiveMQ Artemis 提供，其支持 HornetQ 的核心协议。

- 向前兼容性 ：使用 HornetQ 的 Legacy Jakarta 消息传递客户端可以连接到运行 ActiveMQ Artemis 的 JBoss EAP 7 服务器。
- 向后兼容性 ：使用 JBoss EAP 消息传递的 JBoss EAP 7 Jakarta 消息传递客户端可以连接运行 HornetQ 的传统 JBoss EAP 6 服务器。

32.1. FORWARD COMPATIBILITY

向前兼容性不需要对旧版 JBoss EAP 6 Jakarta 消息传递客户端的代码更改。JBoss EAP messaging-activemq 子系统及其资源提供支持。为了支持向前兼容性，需要对 JBoss EAP 7 服务器的配置进行以下更改：每个步骤都提供了单机服务器的管理 CLI 命令示例：

- 创建 socket-binding，侦听远程传统客户端的端口 4447。

```
/socket-binding-group=standard-sockets/socket-binding=legacy-remoting:add(port=4447)
```

- 创建一个传统的 remote-connector，它将使用上一步中创建的 socket-binding。JNDI 查找需要此项。

```
/subsystem=remoting/connector=legacy-remoting-connector:add(socket-binding=legacy-remoting)
```

- 设置一个传统的消息传递 套接字绑定，侦听端口 5445。

```
/socket-binding-group=standard-sockets/socket-binding=legacy-messaging:add(port=5445)
```

- 在 messaging -activemq 子系统中设置 remote-connector 和 remote- acceptor，该子系统使用上一步中的绑定。

```
/subsystem=messaging-activemq/server=default/remote-connector=legacy-messaging-connector:add(socket-binding=legacy-messaging)
```

```
/subsystem=messaging-activemq/server=default/remote-acceptor=legacy-messaging-acceptor:add(socket-binding=legacy-messaging)
```

- 在 **messaging- activemq** 子系统的 **legacy-connection-factory** 元素中创建传统的 **HornetQ Jakarta Messaging ConnectionFactory**。

```
/subsystem=messaging-activemq/server=default/legacy-connection-factory=legacy-discovery:add(entries=[java:jboss/exported/jms/LegacyRemoteConnectionFactory], connectors=[legacy-messaging-connector])
```

- 创建传统的 **HornetQ Jakarta** 消息目的地，并将 **legacy-entries** 属性包含在 **jms-queue** 或 **jms-topic** 资源。

```
jms-queue add --queue-address=myQueue --entries=[java:jboss/exported/jms/myQueue-new] --legacy-entries=[java:jboss/exported/jms/myQueue]

jms-topic add --topic-address=myTopic --entries=[java:jboss/exported/jms/myTopic-new] --legacy-entries=[java:jboss/exported/jms/myTopic]
```

您可以按照以下示例所示，在现有队列或主题中添加 **legacy-entries**。

```
/subsystem=messaging-activemq/server=default/jms-queue=myQueue:write-attribute(name=legacy-entries,value=[java:jboss/exported/jms/myQueue])
```

虽然这些 条目 属性由 **JBoss EAP** 消息 **Jakarta** 消息传递客户端使用，但 **legacy-entries** 供旧版 **HornetQ Jakarta** 消息传递客户端使用。旧版 **Jakarta** 消息传递客户端查找此传统的 **Jakarta** 消息传递资源，以便与 **JBoss EAP 7** 通信。

注意

为避免旧版 **Jakarta** 消息传递客户端中任何代码更改，**messaging-activemq** 子系统中配置的旧 **JNDI** 条目必须与传统的 **Jakarta** 消息传递客户端预期的查找相匹配。

管理 CLI 迁移操作

当您运行管理 **CLI** 迁移 操作以更新 消息传递 子系统配置时，如果布尔值参数 **add-legacy-entries** 设为 **true**，**delay -activemq** 子系统会创建 **legacy-connection-factory** 资源，并将 **legacy-entries** 添加到 **jms-queue** 和 **jms-topic** 资源。迁移的 **messaging-activemq** 子系统中的传统条目将与传统 消息传递 子系统中指定的条目对应，而常规条目则使用 **-new** 后缀创建。

如果在运行 迁移 操作时将布尔值参数 **add-legacy-entries** 设置为 **false**，则 **messaging-activemq** 子

系统中不会创建传统资源，并且传统的 Jakarta 消息传递客户端将无法与 JBoss EAP 7 服务器通信。

32.2. 向后兼容性

向后兼容性不需要更改旧版 JBoss EAP 7 服务器中的配置更改。JBoss EAP 7 Jakarta 消息传递客户端不查找传统服务器上的资源，而是使用客户端 JNDI 创建 Jakarta 消息传递资源。JBoss EAP 7 Jakarta 消息传递客户端可使用这些资源使用 HornetQ 核心协议与传统服务器通信。

警告

目前不支持与 JBoss EAP 5 服务器的 JBoss EAP 7 客户端连接。

JBoss EAP 消息传递支持客户端 JNDI 创建 Jakarta Messaging ConnectionFactory 和 Destination 资源。

例如，如果 JBoss EAP 7 Jakarta 消息传递客户端希望使用名为"myQueue"的 Jakarta 消息传递队列与传统服务器通信，则必须使用以下属性来配置其 JNDI InitialContext ：

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connectionFactory.jms/ConnectionFactory=tcp://<legacy server address>:5445? \

protocolManagerFactoryStr=org.apache.activemq.artemis.core.protocol.hornetq.client.HornetQClientProtocolManagerFactory
queue.jms/myQueue=myQueue
```

然后，客户端可以使用 jms/ConnectionFactory 名称来创建 Jakarta Messaging ConnectionFactory，并使用 jms/myQueue 来创建 Jakarta Messaging Queue。请注意，在指定传统连接工厂的 URL 时，属性 protocolManagerFactoryStr=org.apache.activemq.artemis.core.protocolq.client.HornetQClientProtocolManagerFactory 是必需的。这使得 JBoss EAP 消息传递 Jakarta 消息传递客户端能够与传统服务器中的 HornetQ 代理通信。

部分 IV. 性能调优

第 33 章 监控消息传递统计

在 messaging-activemq 子系统中为消息传递服务器启用了统计信息集合后，您可以查看消息传递服务器上资源的运行时统计信息。

33.1. 启用消息传递统计

由于这可能会对性能造成负面影响，因此默认情况下 *不会* 启用 messaging-activemq 子系统的统计数据收集。您不需要启用队列统计信息来获取基本信息，如队列中的消息数或添加到队列的消息数量。这些统计数据可使用队列属性提供，而无需将 启用了统计信息 设置为 true。

您可以使用 管理 CLI 或 管理控制台 启用其他统计信息收集。

使用管理 CLI 启用消息传递统计信息

以下管理 CLI 命令启用 默认 消息传递服务器的统计信息集合：

```
/subsystem=messaging-activemq/server=default:write-attribute(name=statistics-enabled,value=true)
```

池连接工厂统计数据是独立于其他消息传递服务器统计数据启用的。使用以下命令，启用池连接工厂的统计数据。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra:write-attribute(name=statistics-enabled,value=true)
```

重新加载服务器以使更改生效。

使用管理控制台启用消息传递统计信息

使用以下步骤，通过管理控制台为消息传递服务器启用统计信息集合。

1. 导航到 Configuration → Subsystems → Messaging(ActiveMQ) → Server。
2. 选择服务器并单击" 查看 "。
3. 单击 Statistics 选项卡下的 Edit。

4. 将 Statistics Enabled 字段设置为 ON，再单击 Save。

池连接工厂统计数据是独立于其他消息传递服务器统计数据启用的。使用以下步骤为池式连接工厂启用统计信息集合。

1. 导航到 Configuration → Subsystems → Messaging(ActiveMQ) → Server。
2. 选择 server，选择 Connections，再单击 View。
3. 选择 Pooled Connection Factory 选项卡。
4. 选择池式连接工厂，再单击 Attributes 选项卡下的 Edit。
5. 将 Statistics Enabled 字段设置为 ON，再单击 Save。
6. 重新加载服务器以使更改生效。

33.2. 查看消息传递统计信息

您可以使用 管理 CLI 或 管理控制台 查看消息传递服务器的运行时统计信息。

使用管理 CLI 查看消息传递统计信息

您可以使用以下管理 CLI 命令查看消息传递统计信息：务必包含 include-runtime=true 参数，因为统计信息是运行时信息。

- 查看队列的统计信息.

```
/subsystem=messaging-activemq/server=default/jms-queue=DLQ:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "consumer-count" => 0,
        "dead-letter-address" => "jms.queue.DLQ",
```

```
        "delivering-count" => 0,
        "durable" => true,
        ...
    }
}
```

- 查看主题的统计信息。

```
/subsystem=messaging-activemq/server=default/jms-topic=testTopic:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "delivering-count" => 0,
        "durable-message-count" => 0,
        "durable-subscription-count" => 0,
        ...
    }
}
```

- 查看池化连接工厂的统计信息。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra/statistics=pool:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => 1,
        "AvailableCount" => 20,
        "AverageBlockingTime" => 0L,
        "AverageCreationTime" => 13L,
        "AverageGetTime" => 14L,
        ...
    }
}
```

注意

池连接工厂统计数据是独立于其他消息传递服务器统计数据启用的。具体步骤请参阅 启用消息传递统计信息。

使用管理控制台查看消息传递统计信息

若要从管理控制台查看消息传递统计信息，可从 **Runtime** 选项卡导航到 **Messaging(ActiveMQ)** 子系统，再选择服务器。选择目标以查看其统计信息。

注意

准备的事务 页面可以查看、提交和回滚准备的事务。如需更多信息，请参阅管理消息传递日志准备事务。

如需了解 所有可用统计数据 的详细列表，请参阅消息传递统计信息。

33.3. 配置消息计数器

您可以为消息传递服务器配置以下消息计数器属性：

- **message-counter-max-day-history** ：消息计数器历史记录保留的天数。
- **message-counter-sample-period** ：队列以毫秒为单位，以毫秒为单位。

管理 CLI 命令使用这些语法来配置这些选项：务必将 *STATISTICS_NAME* 和 *STATISTICS_VALUE* 替换为您要配置的统计名称和值。

```
/subsystem=messaging-activemq/server=default::write-
attribute(name=STATISTICS_NAME,value=STATISTICS_VALUE)
```

例如，使用以下命令将 **message-counter-max-day-history** 设为 **5** 天，**message-counter-sample-period** 设为 **2** 秒。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=message-counter-max-day-
history,value=5)
/subsystem=messaging-activemq/server=default:write-attribute(name=message-counter-sample-
period,value=2000)
```

33.4. 查看队列的消息计数器和历史

您可以使用以下管理 CLI 操作查看队列的消息计数器和消息计数器历史记录。

- **list-message-counter-as-json**

- **list-message-counter-as-html**
- **list-message-counter-history-as-json**
- **list-message-counter-history-as-html**

用于显示这些值的管理 CLI 命令使用以下语法：务必将 *QUEUE_NAME* 和 *OPERATION_NAME* 替换为您要使用的队列名称和操作。

```
/subsystem=messaging-activemq/server=default/jms-queue=QUEUE_NAME:OPERATION_NAME
```

例如，使用以下命令，以 JSON 格式查看 TestQueue 队列的消息计数器：

```
/subsystem=messaging-activemq/server=default/jms-queue=TestQueue:list-message-counter-as-
json
{
    "outcome" => "success",
    "result" => "
{\"destinationName\":\"TestQueue\",\"destinationSubscription\":null,\"destinationDurable\":true,\"count\":
0,\"countDelta\":0,\"messageCount\":0,\"messageCountDelta\":0,\"lastAddTimestamp\":\"12/31/69
7:00:00 PM\",\"updateTimestamp\":\"2/20/18 2:24:05 PM\"}"
}
```

33.5. 为队列重置消息计数器

您可以使用 reset-message-counter 管理 CLI 操作重置队列的消息计数器。

```
/subsystem=messaging-activemq/server=default/jms-queue=TestQueue:reset-message-counter
{
    "outcome" => "success",
    "result" => undefined
}
```

33.6. 使用管理控制台的运行时操作

使用管理控制台，您可以：

- 执行强制故障转移到另一消息传递服务器
- 重置消息传递服务器的所有消息计数器
- 重置消息传递服务器的所有消息计数器历史记录
- 查看与消息传递服务器相关的信息
- 消息传递服务器的关闭连接
- 回滚事务
- 提交事务

执行强制故障到其他消息传递服务器

1. 访问管理控制台并使用以下任一方法进入 Server ：
   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server
2. 点 Messaging ActiveMQ → Server
3. 单击 View 旁边的箭头按钮，再单击 force Failover。
4. 在 force Failover 窗口上，单击 Yes。

为消息传递服务器重置所有消息计数器

1. 访问管理控制台并使用以下任一方法进入 Server ：
   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server
2. 点 Messaging ActiveMQ → Server
3. 单击 View 旁边的箭头按钮，再单击 重置。
4. 在 Reset 窗口上，单击 Reset all message counters 旁边的切换按钮来启用 该功能。

   该按钮现在 以 蓝色背景显示。
5. 点 Reset。

为消息传递服务器重置消息计数器历史

1. 访问管理控制台并使用以下任一方法进入 Server ：
   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server
2. 点 Messaging ActiveMQ → Server
3. 单击 View 旁边的箭头按钮，再单击 重置。
4. 在 Reset 窗口上，单击 Reset all message counters history 旁边的切换按钮来启用该功能。

该按钮现在 以 蓝色背景显示。

5. 点 Reset。

查看与消息传递服务器相关的信息

使用管理控制台，您可以查看与消息传递服务器相关的以下信息列表：

- 连接
- 用户
- producer
- 连接器
- 角色
- 事务

查看与消息传递服务器相关的信息：

1. 访问管理控制台并使用以下任一方法进入 Server ：
   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server
2. 点 Messaging ActiveMQ → Server，然后点 View。

3. 单击导航窗格上的相应项目，以查看右侧窗格上的项目列表。

关闭消息传递服务器的连接

您可以通过提供 IP 地址、ActiveMQ 地址匹配或用户名来关闭连接。

关闭消息传递服务器的连接：

1. 访问管理控制台并使用以下任一方法进入 Server ：
   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server
2. 点 Messaging ActiveMQ → Server，然后点 View。
3. 在导航窗格上，单击 Connections。
4. 在 Close 窗口中，点击您要关闭的连接的适当选项卡。
5. 根据您的选择，输入 IP 地址、ActiveMQ 地址匹配或用户名，然后单击 Close。

为消息传递服务器滚动后端事务

1. 访问管理控制台并使用以下任一方法进入 Server ：
   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server

2. 点 Messaging ActiveMQ → Server，然后点 View。

3. 在导航窗格上，单击 Transactions。

4. 选择您要回滚的事务并点击 Rollback。

为消息传递服务器提交事务

1. 访问管理控制台并使用以下任一方法进入 Server ：

   - runtime→ Browse By → Hosts → Host → Server
   - runtime→ Browse By → Server Groups → Server Group → Server

2. 点 Messaging ActiveMQ → Server，然后点 View。

3. 在导航窗格上，单击 Transactions。

4. 选择您要提交的事务，然后单击 Commit。

第 34 章 调整 JAKARTA MESSAGING

如果使用 Jakarta Messaging API，请查看以下有关如何提高性能的提示：

- 禁用消息 ID。

  如果您不需要消息 ID，请使用 MessageProducer 类上的 setDisableMessageID（） 方法禁用它们。将值设为 true 可消除创建唯一 ID 的额外开销，并减小消息的大小。

- 禁用消息时间戳。

  如果您不需要消息时间戳，请使用 MessageProducer 类上的 setDisableMessageTimeStamp（） 方法禁用它们。将值设为 true 可消除创建时间戳的开销并减小消息的大小。

- 避免使用 ObjectMessage。

  ObjectMessage 用于发送包含序列化对象（即消息正文或载荷）的消息，作为字节流通过线路发送。即便是小型对象的 Java 序列化形式也非常大，占用线路上的大量空间。与自定义托管技术相比，它也很慢。只有在您无法使用其他一种消息类型时才使用 ObjectMessage，例如，如果您不知道有效负载的类型，直到运行时为止。

- 避免 AUTO_ACKNOWLEDGE.

  选择确认模式会影响消费者的性能，因为通过网络发送确认消息会导致额外的开销和流量。AUTO_ACKNOWLEDGE 会产生这种开销，因为它需要针对客户端上收到的每一消息从服务器发送确认。如果可以，请使用 DUPS_OK_ACKNOWLEDGE （以 lazy 方式确认消息），即 CLIENT_ACKNOWLEDGE 表示客户端代码将调用一种方法来确认消息，或者通过在转换的会话中提交来批量进行许多确认。

- 避免持久消息。

  默认情况下，Jakarta 消息传递消息具有持久性。如果您不需要持久消息，请将它们设置为 不可持久。持久化的消息会产生大量开销，因为它们会保留在存储中。

- 使用 TRANSACTED_SESSION 模式在单个事务中发送和接收消息。

通过在单个交易中批处理消息，JBoss EAP 中集成的 ActiveMQ Artemis 服务器仅要求提交一次网络往返，而不是每次发送或接收。

第 35 章 调优持久性

- 将消息日志放在其自己的物理卷中。

  仅附加日志的一个优点是最小化磁盘头移动。如果共享磁盘，则这一优势会丢失。当交易协调员、数据库和其他日志等多个进程从同一磁盘读取和写入时，性能会受到影响，因为磁盘头在不同的文件之间必须跳过。如果您使用分页或大消息，请确保它们也放在单独的卷中。

- 调优 journal-min-files 值。

  将 journal-min-files 参数设置为适合您平均可持续发展率的文件数。如果您经常看到在日志数据目录中创建新文件，这意味着许多数据将被保留，您需要增加最少的文件数量。这使得日志能够重复使用，而不是创建新的数据文件。

- 优化日志文件大小。

  日志文件大小必须与磁盘上柱面的容量一致。大多数系统中的默认值 10MB 应当足够。

- 使用 AIO 日志类型。

  对于 Linux 操作系统，请将您的日志类型保留为 AIO。AIO 扩展优于 Java NIO.

- 调优 journal-buffer-timeout 值。

  增加 journal-buffer-timeout 值会导致吞吐量增加，但会牺牲延迟。

- 调优 journal-max-io 值。

  如果使用 AIO，可以通过增大 journal-max-io 参数值来 提高性能。如果您使用 NIO，请不要更改这个值。

第 36 章 其他调整选项

本节介绍 JBoss EAP 消息传递中可以调优的其他位置。

- 使用异步发送确认.

  如果您需要发送非事务的持久化消息，且不需要保证它们已在 调用 send（） 返回时到达服务器，请不要将它们设置为阻止发送。相反，应使用异步发送确认在单独的流中获取您的发送确认。但是，在服务器崩溃时，一些消息可能会丢失。

- 使用 预确认模式。

  使用 预确认模式 时，会在消息发送到客户端之前进行确认。这可减少线路上的确认流量量。但是，如果该客户端崩溃，如果客户端重新连接，则不会重新传送消息。

- 禁用安全.

  当您通过将 security-enabled 属性设置为 false 来禁用安全性时，性能会很小提高。

- 禁用持久性。

  您可以通过将 persistence-enabled 设置为 false 来完全关闭消息持久性。

- 轻松同步事务.

  将 journal-sync-transactional 设置为 false 可提供更好的事务持久性性能，牺牲了在故障时丢失事务的可能性。

- 以非事务方式同步.

  将 journal-sync-non-transactional 设置为 false 可提供更好的非事务持久性能，但会牺牲在故障时丢失持久消息的可能性。

- 发送消息非阻塞.

为避免等待发送的每一邮件的网络往返，如果您使用 Jakarta Messaging 和 JNDI，请调用 setBlockOnDur ableSend（）方法，将 block-on-durable-send 和 block-on-durable-send（）设置为 false，或者通过调用 setBlockOnDurableSend（）方法 直接将其 设置在 ServerLocator 上。

- 优化 consumer-window-size。

  如果您的消费者速度非常快，您可以增大使用者 -窗口大小 以有效地禁用消费者流控制。

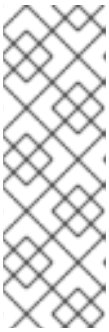
- 使用核心 API 而不是 Jakarta Messaging API。

  Jakarta 消息传递操作必须转换为核心操作，然后服务器才能处理它们，比使用核心 API 时性能更低。使用核心 API 时，请尝试使用尽可能使用 SimpleString 的方法。SimpleString 与 java.lang.String 不同，在写入到线路之前不需要复制；因此，如果您在调用之间重复使用 SimpleString 实例，您可以避免一些不必要的复制。请注意，核心 API 无法移植到其他代理。

第 37 章 避免冲突(PATTERN)

- 尽可能重复利用连接、会话、使用者和制作者。

最常见的消息传递反模式是为发送或使用的每条消息创建一个新的连接、会话和制作者。这些对象需要时间来创建，并且可能涉及多次网络往返，因此资源使用较差。始终重复使用它们。



注意

Spring Messaging 模板等一些常用的库使用这些反模式。如果您使用的是 Spring 消息传递模板，则性能可能会较差。Spring Messaging Template 只能在缓存 Jakarta 消息会话（例如使用 Jakarta Connectors）的应用服务器中安全地使用，然后仅用于发送消息。它不能安全地用于同步使用消息，即使在应用服务器中也是如此。

- 避免空消息.

XML 等详细格式占用了线上的大量空间，因此性能会受到影响。如果可以，避免在消息正文中使用 XML.

- 不要为每个请求创建临时队列。

这种常见的反模式涉及临时队列请求响应模式。使用临时队列请求模式时，消息将发送到目标，并且使用本地临时队列的地址设置回复标头。当接收者收到邮件时，它们将处理邮件，然后发回回复到回复标题中指定的地址。使用这种模式的常见错误是在每个发送的消息上创建一个新的临时队列，这会大大降低性能。相反，应当为许多请求重复使用临时队列。

- 不要使用消息驱动型 Bean，除非有此必要。

使用 MDB 来使用消息的速度要比使用简单的 Jakarta 消息使用者来使用消息的速度慢。

附录 A. 参考资料

A.1. 地址设置属性

表 A.1. 地址设置属性

Name	描述
address-full-policy	决定当指定 **max-size-bytes** 的地址已满时会发生什么。可接受的值有 **PAGE**、**DROP**、**FAIL** 或 **BLOCK**。如果值为 **PAGE**，则会将进一步的消息分页到磁盘。如果值为 **DROP**，则后续消息将被静默丢弃。如果值为 **FAIL**，则消息将被丢弃，客户端消息制作者将收到异常。如果值为 **BLOCK**，则客户端消息制作者会在尝试发送更多消息时阻止它们。**PAGE** 是默认设置。有关分页的详情，请参阅 关于分页。
auto-create-jms-queues	决定 JBoss EAP 是否应在 Jakarta 消息制作者或消费者尝试使用此类队列时自动创建与地址设置匹配的 Jakarta 消息传递队列。默认值为 **false**。*deprecated：改为使用* ***auto-create-queues***。
auto-create-jms-topics	决定 JBoss EAP 是否应在 Jakarta 消息制作者或消费者尝试使用此类队列时自动创建与地址设置匹配的 Jakarta Messaging 主题。默认值为 **false**。*deprecated：改为使用* ***auto-create-addresses***。
auto-create-addresses	确定代理在发送消息时应自动创建地址，还是尝试连接到名称 **与地址匹配的**队列。自动创建的队列具有持久性、非临时和非临时的。默认值为 **true**。
auto-create-queues	确定代理是否应在发送消息时自动创建队列，或者使用者是否尝试连接到名称 **与地址匹配的** 队列。自动创建的队列具有持久性、非临时和非临时的。默认值为 **true**。
auto-delete-jms-queues	决定 JBoss EAP 是否应该在没有消费者且无消息时自动删除自动创建的 Jakarta 消息传递队列。默认值为 **false**。*deprecated：改为使用* ***auto-delete-queues***。
auto-delete-jms-topics	决定 JBoss EAP 是否应该在没有消费者且无消息时自动删除自动创建的 Jakarta 消息主题。默认值为 **false**。*deprecated：改为使用* ***auto-delete-addresses***。
auto-delete-addresses	决定代理是否应在地址不再有任何队列后自动删除自动创建的地址。默认值为 **true**。
auto-delete-queues	决定代理是否应在具有 0 个消费者和 **0** 消息时自动删除自动创建的队列。默认值为 **true**。
dead-letter-address	将死信发送到的地址。如需更多信息，请参阅配置 Dead Letter 地址。

Name	描述
expiry-address	将接收过期邮件的地址。详情请参阅 配置消息过期。
expiry-delay	定义过期时间（以毫秒为单位），它将用于使用默认到期时间的消息。默认值为 **-1**。
last-value-queue	定义队列是否仅使用最后一个值。如需更多信息，请参阅 Last-value Queues。
max-delivery-attempts	定义在发送到死信地址之前可以重新传送取消的消息的时间。默认值为 **10**。
max-redelivery-delay	redelivery-delay 的最大值，以毫秒为单位。默认值为 **0**。
max-size-bytes	此地址的最大大小，以字节为单位。默认值为 **-1**。
message-counter-history-day-limit	消息计数器历史记录的天限值.默认值为 **0**。
page-max-cache-size	在分页导航期间保留在内存中的页面文件数，以优化 IO。默认值为 **5**。
page-size-bytes	分页大小，以字节为单位。默认值为 **10485760**。
redelivery-delay	定义在尝试重新传送已取消消息前需要等待的时间（以毫秒为单位）。默认值为 **0**。如需更多信息，请参阅配置延迟的 Redelivery。
redelivery-multiplier	应用到 redelivery-delay 参数的倍数。默认值为 **1.0**。
redistribution-delay	定义在队列中最后一个消费者关闭后等待的时间（以毫秒为单位），然后再重新分发任何消息。默认值为 **-1**。
send-to-dla-on-no-route	当设置为 **true** 时，如果无法路由到任何队列，消息将发送到配置的死信地址。默认为 **false**。
slow-consumer-check-period	慢速消费者的检查频率（秒数）。默认值为 **5**。
slow-consumer-policy	决定识别慢速消费者时发生的情况。有效选项为 **KILL** 或 **NOTIFY**。**KILL** 将终止消费者的连接，这将影响使用同一连接的任何客户端线程。**NOTIFY** 将向客户端发送 **CONSUMER_SLOW** 管理通知。默认为 **NOTIFY**。
slow-consumer-threshold	在消费者被视为缓慢之前允许的最小消息消耗率。默认值为 **-1**。

A.2. 连接事实属性

表 A.2. 连接事实属性

属性	描述
auto-group	是否自动使用消息分组.
block-on-acknowledge	是否在确认时阻止.
block-on-durable-send	是否阻止在持久发送上.
block-on-non-durable-send	是否阻止非持久发送.
cache-large-message-client	是否缓存大型消息.
call-failover-timeout	故障转移时要使用的超时时间（以毫秒为单位）。
call-timeout	调用超时，以毫秒为单位。
client-failure-check-period	客户端失败检查周期，以毫秒为单位。
client-id	客户端 ID。
press-large-messages	是否应压缩大型消息。
confirmation-window-size	确认窗口大小，以字节为单位。
connection-load-balancing-policy-class-name	实施客户端负载平衡策略的类的名称，客户端可以使用该策略在集群中的不同节点之间负载平衡会话。
connection-ttl	生存连接时间，以毫秒为单位。
connectors	定义连接器，通过连接器名称（具有未定义值）存储在映射中。在编写此属性时，可以传递连接器名称的列表。
consumer-max-rate	每秒消费者的最大率。
consumer-window-size	使用者窗口大小，以字节为单位。
deserialization-black-list	定义不允许进行反序列化的类或软件包的列表。
deserialization-white-list	定义允许被反序列化的类或包的列表。
discovery-group	发现组名称。
dups-ok-batch-size	dups ok 批处理大小。
enable-amq1-prefix	指定是否在 Jakarta Messaging 目的地名称中包含前缀。如果值为 false，则不包含前缀。

属性	描述
条目	连接工厂应绑定到 JNDI 名称。
factory-type	连接工厂的类型。有效值为： • **GENERIC** • **主题** • **QUEUE** • **XA_GENERIC** • **XA_QUEUE** • **XA_TOPIC** 使用 **GENERIC** 与代理的一般连接，而 **TOPIC** 和 **QUEUE** 则应用于连接到其各自的 Jakarta Messaging 类型。XA 对等点应用于事务性消息传递。
failover-on-initial-connection	是否在初始连接时故障转移.
group-id	组 ID。
ha	连接工厂是否支持高可用性。
max-retry-interval	最大重试间隔，以毫秒为单位。
min-large-message-size	最小大消息大小，以字节为单位。
pre-acknowledge	是否预先确认.
producer-max-rate	制作者每秒的最大率。
producer-window-size	制作者窗口大小，以字节为单位。
protocol-manager-factory	此连接工厂使用的协议管理器工厂。
reconnect-attempts	重新连接尝试。
retry-interval	重试间隔，以毫秒为单位。
retry-interval-multiplier	重试间隔倍数。
scheduled-thread-pool-max-size	调度线程池最大大小。
thread-pool-max-size	线程池最大大小。

属性	描述
transaction-batch-size	事务批量大小。
use-global-pools	是否使用全局池.
use-topology-for-load-balancing	指定消息传递客户端将使用集群拓扑连接到集群，还是使用初始连接器进行所有连接。

A.3. 池连接事实属性

表 A.3. 池连接事实属性

属性	描述
allow-local-transactions	是否允许对出站 Jakarta 消息传递会话进行本地事务. 当设置为 **true** 时，这可让 Jakarta 消息传递制作者在以下情况下发送消息： ● 转换的会话中的 servlet 中。 ● 从已转换会话中的 MDB，事务类型属性设为 **NOT_SUPPORTED**。 此属性不适用于 **JMSContext**，后者明确禁止它。
auto-group	是否自动使用消息分组.
block-on-acknowledge	是否在确认时阻止.
block-on-durable-send	是否阻止在持久发送上.
block-on-non-durable-send	是否阻止非持久发送.
cache-large-message-client	是否缓存大型消息.
call-failover-timeout	故障转移时要使用的超时时间（以毫秒为单位）。
call-timeout	调用超时，以毫秒为单位。
client-failure-check-period	客户端失败检查周期，以毫秒为单位。
client-id	客户端 ID.
press-large-messages	是否应压缩大型消息。
confirmation-window-size	确认窗口大小，以字节为单位。

属性	描述
connection-load-balancing-policy-class-name	实施客户端负载平衡策略的类的名称，客户端可以使用该策略在集群中的不同节点之间负载平衡会话。
connection-ttl	生存连接时间，以毫秒为单位。
connectors	定义连接器，通过连接器名称（具有未定义值）存储在映射中。在编写此属性时，可以传递连接器名称的列表。
consumer-max-rate	每秒使用者最大率。
consumer-window-size	使用者窗口大小，以字节为单位。
credential-reference	凭据（来自凭据存储），用于验证池连接工厂。
deserialization-black-list	定义不允许进行反序列化的类或软件包的列表。
deserialization-white-list	定义允许被反序列化的类或包的列表。
discovery-group	发现组名称。
dups-ok-batch-size	dups ok 批处理大小。
enable-amq1-prefix	指定是否在 Jakarta Messaging 目的地名称中包含前缀。如果值为 false，则不包含前缀。
enlistment-trace	启用 IronJacamar 记录此池连接工厂的加入跟踪。此属性默认未定义，行为是由 **ironjacamar.disable_enlistment_trace** 系统属性所驱动的。
条目	连接工厂应绑定到的 JNDI 名称。
failover-on-initial-connection	是否在初始连接时故障转移.
group-id	组 ID。
ha	连接工厂是否支持高可用性。
initial-connect-attempts	最初尝试与此工厂连接的次数。
initial-message-packet-size	通过该工厂创建的消息的初始大小。
jndi-params	用于查找传入连接目的地的 JNDI 参数。
managed-connection-pool	此池连接工厂使用的受管连接池的类名称。

属性	描述
max-pool-size	池的最大大小。
max-retry-interval	最大重试间隔，以毫秒为单位。
min-large-message-size	最小大消息大小，以字节为单位。
min-pool-size	池的最小大小。
password	此连接工厂要使用的默认密码。这只在将连接工厂指向远程主机时才需要。
pre-acknowledge	是否预先确认.
producer-max-rate	制作者的最大率，每秒。
producer-window-size	制作者窗口大小，以字节为单位。
protocol-manager-factory	这个池连接工厂使用的协议管理器工厂。
reconnect-attempts	重新连接尝试。默认情况下，池化连接工厂将尝试无限地重新连接到消息传递服务器。
retry-interval	重试间隔，以毫秒为单位。
retry-interval-multiplier	重试间隔倍数。
scheduled-thread-pool-max-size	计划线程池最大大小。
setup-attempts	设置 MDB 端点的次数。
setup-interval	设置 MDB 端点尝试之间的间隔，以毫秒为单位。
启用了统计	是否启用运行时统计数据。
thread-pool-max-size	线程池最大大小。
事务	事务模式。
transaction-batch-size	事务批量大小。
use-auto-recovery	是否使用自动恢复.
use-global-pools	是否使用全局池.

属性	描述
use-jndi	使用 JNDI 查找传入连接的目的地。
use-local-tx	将本地事务用于传入会话.
use-topology-for-load-balancing	指定消息传递客户端将使用集群拓扑连接到集群，还是使用初始连接器进行所有连接。
user	此连接工厂使用的默认用户名。这只在将连接工厂指向远程主机时才需要。

A.4. CORE BRIDGE ATTRIBUTES

表 A.4. Core Bridge Attributes

属性	描述
call-timeout	内核网桥发出的块调用的响应时间（以毫秒为单位）。当没有配置重复检测时，核心网桥使用阻塞调用。默认值为 **30000**，或 30 秒。
check-period	客户端失败检查间隔期间，以毫秒为单位。
confirmation-window-size	用于将消息转发到目标节点的连接的大小。
connection-ttl	网桥使用的连接被视为活动时间（以毫秒为单位），但没有心跳。
credential-reference	凭据（来自凭据存储），用于验证网桥。
discovery-group	此网桥使用的发现组的名称。如果定义了 **static-connectors**，则无法设置此属性。
filter	可选的过滤器字符串。如果指定，则仅转发与指定的过滤器表达式匹配的消息。
forwarding-address	消息要转发到的目标服务器上的地址。如果没有指定转发地址，则会保留邮件的原始目的地。
ha	此网桥是否应该支持高可用性。如果为 **true**，它将连接到群集中的任何可用服务器并支持故障切换。默认值为 **false**。
initial-connect-attempts	最初尝试与此网桥连接的次数。
max-retry-interval	用于重试连接的最长时间间隔。
min-large-message-size	消息被视为大消息前的最小大小，以字节为单位。

属性	描述
password	创建与远程服务器的网桥连接时要使用的密码。如果未指定，则将使用 messaging- **activemq** 子系统 **资源中 cluster-password** 属性指定的默认集群密码。
cluster-credential-reference	用于向集群进行身份验证的凭据存储引用。这可以使用，而不使用 **password**。
queue-name	网桥从其消耗的本地队列的唯一名称。当网桥在启动时实例化时，队列必须已存在。
reconnect-attempts	在放弃和关闭之前网桥进行的重新连接尝试总数。默认值为 **-1**，表示无限数量的尝试。
reconnect-attempts-on-same-node	在放弃和关闭前，网桥在放弃和关闭前在同一个节点上重新连接尝试的总数。值 **-1** 表示无限数量的尝试。默认值为 **10**。
retry-interval	如果与目标服务器的连接失败，则后续重新连接尝试之间的周期以毫秒为单位。
retry-interval-multiplier	倍数应用到自上次重试以来的时间，以计算时间到下一次重试的时间。这可让您在重试尝试之间实施指数回退。
static-connectors	此网桥使用的静态定义的连接器列表。如果定义了 **discovery-group**，则可能不会设置此属性。
transformer-class-name	实施 **org.apache.activemq.artemis.core.server.cluster.Transformer** 接口的用户定义的类的名称。
use-duplicate-detection	网桥是否会自动将重复的 ID 属性插入到它转发的每一消息中。
user	创建与远程服务器的网桥连接时要使用的用户名。如果未指定，则将使用 messaging- **activemq 子系统资源中 cluster- user** 属性指定的默认集群用户。

A.5. JAKARTA 消息传递网桥属性

表 A.5. Jakarta 消息传递网桥属性

属性	描述
add-messageID-in-header	如果设置为 **true**，则原始消息的消息 ID 将在标头 **AMQ_BRIDGE_MSG_ID_LIST** 中发送到目的地的消息中附加。如果消息多次桥接一次，则将附加每个消息 ID。

属性	描述
client-id	如果订阅持久且源目的地为主题，则在创建和查找订阅时要使用的 Jakarta 消息传递客户端 ID。
failure-retry-interval	当网桥检测到失败时，尝试重新创建源服务器或目标服务器连接之间的时间，以毫秒为单位。
max-batch-size	从源目的地使用的最大消息数，再将它们批量发送到目标目的地。该值必须大于或等于 **1**。
max-batch-time	在向目标发送批量消息前等待的最大毫秒数，即使使用的消息数未达到 max-batch-size。值 **-1** 表示永久等待。
max-retries	当网桥检测到失败时，尝试重新创建与源或目标服务器的连接的次数。尝试这一次数后，网桥将放弃。值为 **-1** 表示永久尝试。
module	包含查找来源并以 Jakarta 消息资源为目标的 JBoss EAP 模块的名称。
paused	种只读属性，报告 Jakarta Messaging 网桥是否暂停。
quality-of-service	所需服务质量模式.可能的值有 **AT_MOST_ONCE**、**DUPLICATES_OK** 或 **ONCE_AND_ONLY_ONCE**。有关不同模式的详情，请参阅服务质量。
selector	Jakarta 消息传递选择器表达式，用于使用来自源目的地的消息。只有与选择器表达式匹配的消息才会从源桥接到目标目的地。
subscription-name	如果订阅持久且源目的地是一个主题，则订阅的名称。
source-connection-factory	在源消息传递服务器上查找的源连接工厂的名称。
source-context	用于配置源 JNDI 初始上下文的属性。
source-credential-reference	凭据存储引用，用于验证源连接。这可以使用，而不使用 **source-password**。
source-destination	要在源消息传递服务器上查找的源目的地名称。
source-password	用于创建源连接的密码。
source-user	用于创建源连接的用户名称。
target-connection-factory	在目标消息传递服务器上查找的目标连接工厂的名称。
target-context	用于配置目标 JNDI 初始上下文的属性。

属性	描述
target-credential-reference	凭据存储引用，用于验证目标连接。可以使用此选项而不是 **target-password**。
target-destination	要在目标消息传递服务器上查找的目标目的地的名称。
target-password	用于创建目标连接的密码。
target-user	用于创建目标连接的用户名称。

A.6. 集群连接属性

表 A.6. 集群连接属性

属性	描述
allow-direct-connections-only	如果设为 **true，则** 此节点 **不会** 创建到集群中另一节点的连接（如果它驻留于 1 个跃点之外）。仅在定义了 **static-connectors** 属性时使用。默认值为 **false**。
call-failover-timeout	当故障转移正在处理由集群连接发出的远程调用时，可以使用的超时时间为毫秒。默认值为 **-1**，它没有绑定。
call-timeout	集群连接发出远程调用的超时时间，以毫秒为单位。默认值为 **30000** 或 30 秒。
check-period	客户端失败检查间隔期间，以毫秒为单位。默认值为 **30000** 或 30 秒。
cluster-connection-address	每个集群连接都只适用于发送到以这个值开头的地址的消息。
confirmation-window-size	用于将消息转发到目标节点的连接的窗口大小（以字节为单位）。默认值为 **1048576**。
connection-ttl	在没有心跳的情况下，群集连接使用的连接将被视为活动时间（以毫秒为单位）。默认值为 **60000** 或 60 秒。
connector-name	用于集群连接的连接器名称。
discovery-group	用于获取集群中进行连接的群集中其他服务器列表的发现组。如果定义了 **static-connectors**，则必须未定义(null)。
initial-connect-attempts	最初使用此集群连接的尝试次数。默认值为 **-1**，它没有绑定。
max-hops	消息可以转发的最大次数。JBoss EAP 可以配置为同时平衡可能仅间接与其他 ActiveMQ Artemis 消息服务器连接的节点之间的负载平衡消息，作为链中的中间产品。默认值为 **1**。

属性	描述
max-retry-interval	用于重试连接的最大时间间隔，以毫秒为单位。默认值为 **2000 或** 两秒钟。
message-load-balancing-type	此参数决定消息如何在群集中的其他节点之间分发。替换已弃用的 **forward-when-no-consumers**。有效值为 **OFF**、S**TRICT** 或 **ON_DEMAND**。 **OFF** 消息永远不会转发到群集中的另一节点。 **STRICT** 消息将以轮循方式分发，即使群集中其他节点上的相同队列可能没有任何使用者，或者它们可能具有不匹配的消息过滤器或选择器。请注意，如果其他节点上没有相同名称的队列，JBoss EAP 也不会将消息转发到其他节点，即使此参数设为 **STRICT**。使用 **STRICT** 类似于将旧的 **forward-when-no-consumers** 参数设置为 **true**。 **ON_DEMAND** 如果转发地址具有拥有使用者的队列，则消息将转发到集群的其他节点。如果这些使用者具有消息过滤器或选择器，则至少其中一个选择器必须与消息匹配。使用 **ON_DEMAND** 就像将旧的 **forward-when-no-consumers** 参数设置为 **false**。 默认值为 **ON_DEMAND**。
min-large-message-size	消息被视为大消息前的最小大小，以字节为单位。默认值为 **102400**。
node-id	此集群连接使用的节点 ID。此属性是只读的。
notification-attempts	集群连接将自行广播的次数。默认值为 **2**。
notification-interval	通知之间的间隔，以毫秒为单位。默认值为 **10000 或** 10 秒。
reconnect-attempts	在放弃和关闭之前，网桥将尝试重新连接的总数。默认值为 **-1**，表示无限数量的尝试。
retry-interval	如果与目标服务器的连接失败，则后续尝试重新连接到目标服务器之间的周期（以毫秒为单位）。默认值为 **500**。
retry-interval-multiplier	倍数应用到自上次重试以来的时间，以计算时间到下一次重试的时间。这可让您在重试尝试之间实施指数回退。默认值为 **1.0**。
static-connectors	此集群连接将进行静态定义的连接器列表。如果定义了 **discovery-group-name，则必须** 未定义。
topology	此集群连接了解的节点拓扑。此属性是只读的。
use-duplicate-detection	网桥是否会自动将重复的 ID 属性插入到它转发的每一消息中。默认值为 **true**。

A.7. 消息传递统计信息

队列统计信息

表 A.7. 队列统计信息

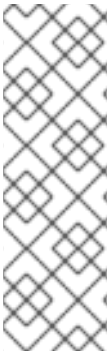
统计	描述
消费者数	此队列中消息的使用者数量。
Message Count	此队列中当前消息的数量。
添加 Count 的消息	自该队列创建以来添加到此队列的消息数。
调度数	此队列中调度的消息数量。

主题统计

表 A.8. 主题统计

统计	描述
提供计数	本主题正在向其消费者传送的消息数量。
持久消息计数	本主题的所有持久订阅者的消息数量。
持久订阅数	本主题的持久订户数量。
Message Count	当前在此主题中的消息数。
添加消息	自创建以来添加到此主题的消息数。
订阅数	本主题的持久和不可持久性订阅用户数量。

池连接事实统计信息



注意

池连接工厂的统计信息集合与为消息传递服务器收集的其他统计数据分开启用。使用以下管理 CLI 命令，为池化连接工厂启用统计信息集合：

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra:write-attribute(name=statistics-enabled,value=true)
```

表 A.9. 池连接事实统计信息

统计	描述
ActiveCount	活跃数量。
AvailableCount	可用数量。
AverageBlockingTime	池的平均阻塞时间.
AverageCreationTime	创建物理连接的平均时间。
AverageGetTime	获取物理连接的平均时间。
AveragePoolTime	池中物理连接的平均时间。
AverageUsageTime	使用物理连接的平均时间。
BlockingFailureCount	试图获得物理连接的失败数量。
CreatedCount	创建的数量。
DestroyedCount	已销毁数。
IdleCount	当前空闲的物理连接数。
InUseCount	当前使用的物理连接数。
MaxCreationTime	创建物理连接的最长时间。
MaxGetTime	获取物理连接的最长时间。
MaxPoolTime	池中物理连接的最长时间。
MaxUsageTime	使用物理连接的最长时间。
MaxUsedCount	使用的最大连接数。
MaxWaitCount	等待连接的最大线程数。
MaxWaitTime	连接的最长时间。
timedOut	超时时间。
TotalBlockingTime	总阻塞时间.
TotalCreationTime	创建物理连接所花费的总时间。
TotalGetTime	获取物理连接所花费的总时间。

统计	描述
TotalPoolTime	池中物理连接花费的总时间。
TotalUsageTime	使用物理连接所花费的总时间。
WaitCount	必须等待的请求数以获取物理连接。
XACommitAverageTime	XAResource 提交调用的平均时间。
XACommitCount	XAResource 提交调用的数量。
XACommitMaxTime	XAResource 提交调用的最长时间。
XACommitTotalTime	所有 XAResource 提交调用的总时间。
XAEndAverageTime	XAResource 结束调用的平均时间。
XAEndCount	XAResource 端点调用的数量。
XAEndMaxTime	XAResource 结束调用的最长时间。
XAEndTotalTime	所有 XAResource 结束调用的总时间。
XAForgetAverageTime	XAResource 的平均时间忘记调用。
XAForgetCount	XAResource 忘记调用的数量。
XAForgetMaxTime	XAResource 忘记调用的最长时间。
XAForgetTotalTime	所有 XAResource 都忘记调用的总时间。
XAPrepareAverageTime	XAResource 准备调用的平均时间。
XAPrepareCount	XAResource 准备调用的数量。
XAPrepareMaxTime	XAResource 准备调用的最长时间。
XAPrepareTotalTime	所有 XAResource 准备调用的总时间。
XARecoverAverageTime	XAResource 恢复调用的平均时间。
XARecoverCount	XAResource 恢复调用的数量。
XARecoverMaxTime	XAResource 恢复调用的最长时间。

统计	描述
XARecoverTotalTime	所有 XAResource 恢复调用的总时间。
XARollbackAverageTime	XAResource 回滚调用的平均时间。
XARollbackCount	XAResource 回滚调用数量。
XARollbackMaxTime	XAResource 回滚调用的最长时间。
XARollbackTotalTime	所有 XAResource 回滚调用的总时间。
XAStartAverageTime	XAResource 开始调用的平均时间。
XAStartCount	XAResource start 调用的数量。
XAStartMaxTime	XAResource 开始调用的最长时间。
XAStartTotalTime	所有 XAResource 启动调用的总时间。

更新于 2024-02-08