Red Hat AMQ Broker 7.12

配置 AMQ Broker

用于 AMQ Broker 7.12

Last Updated: 2024-05-22

Red Hat AMQ Broker 7.12 配置 AMQ Broker

用于 AMQ Broker 7.12

法律通告

摘要

本指南论述了如何配置 AMQ Broker。

目录

使开源包含更多 5

第 1 章 概述 6
1.1. AMQ BROKER 配置文件和位置 6
1.2. 了解默认代理配置 6
1.3. 重新加载配置更新 10
1.4. 模块化代理配置文件 11
1.5. 扩展 JAVA 类路径 13
1.6. 文档惯例 13

第 2 章 在网络连接中配置接收器和连接器 14
2.1. 关于接收器 14
2.2. 配置接收器 15
2.3. 关于连接器 15
2.4. 配置连接器 16
2.5. 配置 TCP 连接 16
2.6. 配置 HTTP 连接 17
2.7. 配置安全网络连接 18
2.8. 配置 IN-VM 连接 18

第 3 章 在网络连接中配置消息传递协议 20
3.1. 将网络连接配置为使用消息传递协议 20
3.2. 使用带有网络连接的 AMQP 23
3.3. 使用带有网络连接的 MQTT 24
3.4. 使用带有网络连接的 OPENWIRE 27
3.5. 使用带有网络连接的 STOMP 27

第 4 章 配置地址和队列 33
4.1. 地址、队列和路由类型 33
4.2. 将地址设置应用到一组地址 34
4.3. 为点到点消息传递配置地址 37
4.4. 为发布订阅消息传递配置地址 40
4.5. 为点到点和发布订阅消息传递配置地址 42
4.6. 在接受器配置中添加路由类型 44
4.7. 配置订阅队列 45
4.8. 自动创建和删除地址和队列 48
4.9. 指定完全限定的队列名称 51
4.10. 配置分片队列 51
4.11. 配置最后一个值队列 53
4.12. 将过期的消息移到过期地址 57
4.13. 将未发送的消息移到死信地址 62
4.14. 在过期或未提供 AMQP 消息上注解和属性 66
4.15. 禁用队列 67
4.16. 限制连接到队列的消费者数量 68
4.17. 配置专用队列 69
4.18. 将特定地址设置应用到临时队列 70
4.19. 配置环队列 71
4.20. 配置 RETROACTIVE 地址 75
4.21. 为内部管理的地址和队列禁用公告消息 76
4.22. 联邦地址和队列 77

第 5 章 保护代理 110

5.1. 保护连接 110
5.2. 验证客户端 113
5.3. 授权客户端 123
5.4. 使用 LDAP 进行身份验证和授权 136
5.5. 使用 KERBEROS 进行身份验证和授权 148
5.6. 指定安全管理器 155
5.7. 禁用安全性 160
5.8. 跟踪来自验证的用户的消息 161
5.9. 在配置文件中加密密码 162
5.10. 配置身份验证和授权缓存 165

第 6 章 持久性消息数据 167
6.1. 在日志中保留消息数据 167
6.2. 在数据库中持久保留消息数据 178
6.3. 禁用持久性 184

第 7 章 为地址配置内存用量 186
7.1. 配置消息分页 186
7.2. 配置消息丢弃 197
7.3. 配置消息阻塞 197
7.4. 了解多播地址上的内存用量 199

第 8 章 处理大型消息 201
8.1. 为大型消息处理配置代理 201
8.2. 为大型消息处理配置 AMQP 接受器 203
8.3. 配置 STOMP ACCEPTOR 用于处理大量信息 204
8.4. 大消息和 JAVA 客户端 205

第 9 章 检测死连接 207
9.1. CONNECTION TIME-TO-LIVE 207
9.2. 禁用异步连接执行 208

第 10 章 检测重复消息 210
10.1. 配置重复的 ID 缓存 210
10.2. 为集群连接配置重复检测 211

第 11 章 截获消息 213
11.1. 创建拦截器 213
11.2. 将 BROKER 配置为使用 INTERCEPTORS 215

第 12 章 分离消息和分割消息流 217
12.1. MESSAGE DIVERTS 如何工作 217
12.2. 配置消息 DIVERTS 217

第 13 章 过滤消息 221
13.1. 将队列配置为使用过滤器 221
13.2. 过滤 JMS 消息属性 222
13.3. 根据注释上的属性过滤 AMQP 消息 222
13.4. 过滤 XML 消息 224

第 14 章 设置代理集群 227
14.1. 了解代理集群 227
14.2. 创建代理集群 232
14.3. 实施高可用性 244
14.4. 启用消息重新发布 270
14.5. 配置集群消息分组 272

14.6. 将客户端连接到代理集群 275
14.7. 对客户端连接进行分区 275

第 15 章 使用 CEPH 配置多站点、容错消息传递系统 284
15.1. RED HAT CEPH STORAGE 集群如何工作 285
15.2. 安装 RED HAT CEPH STORAGE 286
15.3. 配置 RED HAT CEPH STORAGE 集群 288
15.4. 在代理服务器上挂载 CEPH 文件系统 293
15.5. 在多站点、容错消息传递系统中配置代理 294
15.6. 在多站点、容错消息传递系统中配置客户端 297
15.7. 在数据中心停机过程中验证存储集群健康状况 300
15.8. 在数据中心中断期间保持消息传递连续 300
15.9. 重启之前失败的数据中心 303

第 16 章 使用代理连接配置多站点、容错消息传递系统 306
16.1. 关于代理连接 306
16.2. 配置代理镜像 308

第 17 章 桥接代理 312
17.1. 代理连接的发件人和接收器配置 312
17.2. 代理连接的对等配置 314

第 18 章 日志记录 316
18.1. 更改日志记录级别 316
18.2. 启用审计日志记录 316
18.3. 客户端或嵌入式服务器日志记录 317
18.4. AMQ BROKER 插件支持 318

附录 A. ACCEPTOR 和 CONNECTOR 配置参数 322

附录 B. 地址设置配置元素 328

附录 C. 集群连接配置元素 332

附录 D. 命令行工具 335

附录 E. 消息传递日志配置元素 337

附录 F. 其他复制高可用性配置元素 339

附录 G. 代理属性 340
G.1. BRIDGECONFIGURATIONS 352
G.2. AMQPCONNECTIONS 358
G.3. DIVERTCONFIGURATION 360
G.4. ADDRESSSETTINGS 362
G.5. FEDERATIONCONFIGURATIONS 375
G.6. CLUSTERCONFIGURATIONS 389
G.7. CONNECTIONROUTERS 394

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 概述

AMQ Broker 配置文件为代理实例定义重要设置。通过编辑代理的配置文件，您可以控制代理在环境中如何工作。

1.1. AMQ BROKER 配置文件和位置

所有代理的配置文件都存储在 < ***broker_instance_dir***> /**etc** 中。您可以通过编辑这些配置文件中的设置来配置代理。

每个代理实例使用以下配置文件：

broker.xml

主配置文件。您可以使用此文件配置代理的大多数方面，如网络连接、安全设置、消息地址等。

bootstrap.xml

AMQ Broker 用来启动代理实例的文件。您可以使用它来更改 **broker.xml** 的位置，配置 web 服务器，并设置一些安全设置。

logging.properties

您可以使用此文件为代理实例设置日志记录属性。

artemis.profile

您可以使用此文件设置代理实例运行时使用的环境变量。

login.config,artemis-users.properties,artemis-roles.properties

与安全相关的文件。您可以使用这些文件为用户访问代理实例设置身份验证。

1.2. 了解默认代理配置

您可以通过编辑 **broker.xml** 配置文件来配置大多数代理功能。此文件包含默认设置，足以启动和操作代理。但是，您可能需要更改一些默认设置，并添加新设置来为您的环境配置代理。

默认情况下，**broker.xml** 包含以下功能的默认设置：

- 消息持久性
- acceptors
- 安全性
- 消息地址

默认消息持久性设置
默认情况下，AMQ Broker 持久性使用仅附加的文件日志，该日志由磁盘上的一组文件组成。日志保存消息、事务和其他信息。

```
<configuration ...>
  <core ...>
    ...
    <persistence-enabled>true</persistence-enabled>
    <!-- this could be ASYNCIO, MAPPED, NIO
```

```
        ASYNCIO: Linux Libaio
        MAPPED: mmap files
        NIO: Plain Java Files
     -->
    <journal-type>ASYNCIO</journal-type>

    <paging-directory>data/paging</paging-directory>

    <bindings-directory>data/bindings</bindings-directory>

    <journal-directory>data/journal</journal-directory>

    <large-messages-directory>data/large-messages</large-messages-directory>

    <journal-datasync>true</journal-datasync>

    <journal-min-files>2</journal-min-files>

    <journal-pool-files>10</journal-pool-files>

    <journal-file-size>10M</journal-file-size>

    <!--
     This value was determined through a calculation.
     Your system could perform 8.62 writes per millisecond
     on the current journal configuration.
     That translates as a sync write every 115999 nanoseconds.

     Note: If you specify 0 the system will perform writes directly to the disk.
          We recommend this to be 0 if you are using journalType=MAPPED and journal-
datasync=false.
    -->
    <journal-buffer-timeout>115999</journal-buffer-timeout>

    <!--
      When using ASYNCIO, this will determine the writing queue depth for libaio.
     -->
    <journal-max-io>4096</journal-max-io>

    <!-- how often we are looking for how many bytes are being used on the disk in ms -->
    <disk-scan-period>5000</disk-scan-period>

    <!-- once the disk hits this limit the system will block, or close the connection in certain protocols
         that won't support flow control. -->
    <max-disk-usage>90</max-disk-usage>

    <!-- should the broker detect dead locks and other issues -->
    <critical-analyzer>true</critical-analyzer>

    <critical-analyzer-timeout>120000</critical-analyzer-timeout>

    <critical-analyzer-check-period>60000</critical-analyzer-check-period>

    <critical-analyzer-policy>HALT</critical-analyzer-policy>

    ...
```

```
  </core>

</configuration>
```

默认接收器设置

代理通过使用 **acceptor** 配置元素来侦听传入的客户端连接，以定义客户端可用于进行连接的端口和协议。默认情况下，AMQ Broker 包括每个支持的消息传递协议的接收器，如下所示。

```
<configuration ...>

  <core ...>

    ...

    <acceptors>

      <!-- Acceptor for every supported protocol -->
      <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,HORNE
TQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- AMQP Acceptor. Listens on default AMQP port for AMQP traffic -->
      <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCre
dits=1000;amqpLowCredits=300</acceptor>

      <!-- STOMP Acceptor -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true</acce
ptor>

      <!-- HornetQ Compatibility Acceptor. Enables HornetQ Core and STOMP for legacy HornetQ
clients. -->
      <acceptor name="hornetq">tcp://0.0.0.0:5445?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.;protocols=HORNETQ,STOMP;useEpoll=true</a
cceptor>

      <!-- MQTT Acceptor -->
      <acceptor name="mqtt">tcp://0.0.0.0:1883?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</accept
or>

    </acceptors>

    ...

  </core>

</configuration>
```

默认安全设置

AMQ Broker 包含灵活的基于角色的安全模型，可根据它们的地址将安全性应用到队列。默认配置使用通配符将 **amq** 角色应用到所有地址（以数字符号 **#** 表示）。

```
<configuration ...>

  <core ...>

    ...

    <security-settings>
      <security-setting match="#">
        <permission type="createNonDurableQueue" roles="amq"/>
        <permission type="deleteNonDurableQueue" roles="amq"/>
        <permission type="createDurableQueue" roles="amq"/>
        <permission type="deleteDurableQueue" roles="amq"/>
        <permission type="createAddress" roles="amq"/>
        <permission type="deleteAddress" roles="amq"/>
        <permission type="consume" roles="amq"/>
        <permission type="browse" roles="amq"/>
        <permission type="send" roles="amq"/>
        <!-- we need this otherwise ./artemis data imp wouldn't work -->
        <permission type="manage" roles="amq"/>
      </security-setting>
    </security-settings>

    ...

  </core>

</configuration>
```

默认消息地址设置
AMQ Broker 包含一个默认地址，它建立一组要应用到任何创建的队列或主题的默认配置设置。

此外，默认配置定义了两个队列： **DLQ** (Dead Letter Queue)处理到达没有已知目的地的消息， **Expiry Queue** 保管其过期消息，因此不应路由到其原始目的地。

```
<configuration ...>

  <core ...>

    ...

    <address-settings>
      ...
      <!--default for catch all-->
      <address-setting match="#">
        <dead-letter-address>DLQ</dead-letter-address>
        <expiry-address>ExpiryQueue</expiry-address>
        <redelivery-delay>0</redelivery-delay>
        <!-- with -1 only the global-max-size is in use for limiting -->
        <max-size-bytes>-1</max-size-bytes>
        <message-counter-history-day-limit>10</message-counter-history-day-limit>
        <address-full-policy>PAGE</address-full-policy>
        <auto-create-queues>true</auto-create-queues>
        <auto-create-addresses>true</auto-create-addresses>
        <auto-create-jms-queues>true</auto-create-jms-queues>
        <auto-create-jms-topics>true</auto-create-jms-topics>
```

```
    </address-setting>
  </address-settings>

  <addresses>
    <address name="DLQ">
      <anycast>
        <queue name="DLQ" />
      </anycast>
    </address>
    <address name="ExpiryQueue">
      <anycast>
        <queue name="ExpiryQueue" />
      </anycast>
    </address>
  </addresses>

 </core>

</configuration>
```

1.3. 重新加载配置更新

默认情况下，代理会每 5000 毫秒检查配置文件中的更改。如果代理检测到配置文件的"last modified"时间戳的变化，代理会决定配置更改发生。在这种情况下，代理会重新载入配置文件来激活更改。

当代理重新载入 **broker.xml** 配置文件时，它会重新载入以下模块：

- 地址设置和队列
  重新加载配置文件后，地址设置决定了如何处理从配置文件中删除的地址和队列。您可以使用 **config-delete-addresses** 和 **config-delete-queues** 属性设置它。更多信息请参阅 附录 B, *地址设置配置元素*。

- 安全设置
  可以重新加载 SSL/TLS 密钥存储和信任存储在现有接收器上建立新证书，而无需对现有客户端产生任何影响。连接的客户端（即使有旧证书或不同证书）可以继续发送和接收信息。

证书撤销列表文件（使用 **crlPath** 参数配置）也可以重新加载。

- diverts
  配置重新加载会部署您添加的 **任何新的** divert。但是，在重启代理前，从配置中删除对 < **divert&** gt; 元素中的子元素的更改不会生效。

以下步骤演示了如何更改代理检查对 **broker.xml** 配置文件更改的时间间隔。

流程

1. 打开 & ***lt;broker_instance_dir&*gt; /etc/broker.xml** 配置文件。

2. 在 &lt **;core&** gt; 元素中，添加 **<configuration-file-refresh-period** > 元素并设置刷新周期（以毫秒为单位）。
   这个示例将配置刷新周期设置为 60000 毫秒：

```
<configuration>
  <core>
    ...
```

```
      <configuration-file-refresh-period>60000</configuration-file-refresh-period>
      ...
   </core>
</configuration>
```

如果出于某种原因无法访问配置文件，也可以使用 Management API 或控制台强制重新加载配置文件。可以使用 **ActiveMQServerControl** 上的管理操作 **reloadConfigurationFile （）** 重新加载配置文件（使用 **ObjectName org.apache.activemq.artemis:broker="*BROKER_NAME*"** 或资源 **名称服务器**）

其他资源

- 要了解如何使用管理 API，请参阅管理 AMQ Broker 中的 使用管理 API

1.4. 模块化代理配置文件

如果您有多个共享通用配置设置的代理，您可以在单独的文件中定义通用配置，然后在每个代理的 **broker.xml** 配置文件中包括这些文件。

您可以在代理间共享的最常见配置设置包括：

- addresses
- 地址设置
- 安全设置

流程

1. 为您要共享的每个 **broker.xml** 部分创建一个单独的 XML 文件。
   每个 XML 文件只能包含来自 **broker.xml** 的单个部分（例如，地址或地址设置，但不能同时包含两者）。顶级元素还必须定义元素命名空间(**xmlns="urn:activemq:core"**)。

   本例演示了 **my-security-settings.xml** 中定义的安全设置配置：

   my-security-settings.xml

   ```
   <security-settings xmlns="urn:activemq:core">
     <security-setting match="a1">
       <permission type="createNonDurableQueue" roles="a1.1"/>
     </security-setting>
     <security-setting match="a2">
       <permission type="deleteNonDurableQueue" roles="a2.1"/>
     </security-setting>
   </security-settings>
   ```

2. 为每个应使用通用配置设置的代理打开 & ***lt;broker_instance_dir>*** **/etc/broker.xml** 配置文件。
3. 对于您打开的每个 **broker.xml** 文件，请执行以下操作：
   a. 在 **broker.xml** 开头的 < **configuration** > 元素中，验证是否出现以下行：

      ```
      xmlns:xi="http://www.w3.org/2001/XInclude"
      ```

   b. 为每个包含共享配置设置的 XML 文件添加 XML 包含。

这个示例包括 **my-security-settings.xml** 文件。

broker.xml

```
<configuration ...>
  <core ...>
    ...
    <xi:include href="/opt/my-broker-config/my-security-settings.xml"/>
    ...
  </core>
</configuration>
```

c. 如果需要，验证 **broker.xml** 以验证 XML 对 schema 是否有效。
您可以使用任何 XML 验证器程序。本例使用 **xmllint** 根据 **artemis-server.xsl** 模式验证 **broker.xml**。

```
$ xmllint --noout --xinclude --schema /opt/redhat/amq-broker/amq-broker-7.2.0/schema/artemis-server.xsd /var/opt/amq-broker/mybroker/etc/broker.xml
/var/opt/amq-broker/mybroker/etc/broker.xml validates
```

其他资源

- 有关 XML 包含(XIncludes)的详情请参考 https://www.w3.org/TR/xinclude/。

1.4.1. 重新加载模块配置文件

当代理定期检查配置更改（根据 **config-file-refresh-period** 指定的频率），它**不会**自动检测通过 **xi:include** 在 **broker.xml** 配置文件中包含的配置文件的变化。例如，如果 **broker.xml** 包含 **my-address-settings.xml**，并且对 **my-address-settings.xml** 进行配置更改，代理不会自动检测 **my-address-settings.xml** 中的更改并重新载入配置。

要强制 重新载入 **broker.xml** 配置文件以及其中包含的任何修改的配置文件，您必须确保 **broker.xml** 配置文件的"last modified"时间戳已更改。您可以使用标准 Linux **touch** 命令更新 **broker.xml** 的最后修改时间戳，而无需进行任何其他更改。例如：

```
$ touch -m <broker_instance_dir>/etc/broker.xml
```

另外，您可以使用管理 API 来强制重新载入代理。可以使用 **ActiveMQServerControl** 上的管理操作 **reloadConfigurationFile （）** 重新加载配置文件（使用 **ObjectName org.apache.activemq.artemis:broker="*BROKER_NAME*"** 或资源 **名称服务器**）

其他资源

- 要了解如何使用管理 API，请参阅管理 AMQ Broker 中的 使用管理 API

1.4.2. 禁用外部 XML 实体(XXE)处理

如果您不想将代理配置放在 **broker.xml** 文件中包含的独立文件中，您可以禁用 XXE 处理来保护 AMQ Broker 免受 XXE 安全漏洞。如果您没有模块化代理配置，红帽建议您禁用 XXE 处理。

流程

1. 打开 &**lt;broker_instance_dir>/etc/artemis.profile** 文件。

2. 将新参数 **-Dartemis.disableXxe** 添加到 Java 系统参数的 **JAVA_ARGS** 列表中。

```
-Dartemis.disableXxe=true
```

3. 保存 **artemis.profile** 文件。

1.5. 扩展 JAVA 类路径

默认情况下，< ***broker_instance_dir>*/lib** 目录中的 JAR 文件会在运行时加载，因为目录是 Java 类路径的一部分。如果您希望 AMQ Broker 从 < ***broker_instance_dir>* /lib** 以外的目录中加载 JAR 文件，您必须将该目录添加到 Java 类路径中。

要在 Java 类路径中添加目录，您可以使用以下方法之一：

- 在 &**lt;broker_instance_dir>/etc/artemis.profile** 文件中，将新属性 **artemis.extra.libs** 添加到系统属性的 **JAVA_ARGS** 列表中。
- 设置 **ARTEMIS_EXTRA_LIBS** 环境变量。

以下是使用两种方法添加到 Java 类路径中的以逗号分隔的目录列表示例：

```
-Dartemis.extra.libs=/usr/local/share/java/lib1,/usr/local/share/java/lib2
```

```
export ARTEMIS_EXTRA_LIBS=/usr/local/share/java/lib1,/usr/local/share/java/lib2
```

注意

如果在 < **broker_instance_dir>/etc/artemis.profile 文件中配置了 artemis.extra.libs Java 系统属性，则 ARTEMIS_ EXTRA_LIBS** 环境变量会被忽略。

1.6. 文档惯例

本文档对 **sudo** 命令、文件路径和可替换值使用以下惯例：

sudo 命令
在本文档中，**sudo** 用于任何需要 root 特权的命令。使用 **sudo** 时，您应始终谨慎操作，因为任何更改都可能会影响整个系统。

有关使用 **sudo** 的更多信息，请参阅管理 sudo 访问。

关于在此文档中使用文件路径
在这个文档中，所有文件路径都对 Linux、UNIX 和类似操作系统（例如 **/home/...**）有效。如果您使用的是 Microsoft Windows，则应使用等效的 Microsoft Windows 路径（例如，**C:\Users\...**）。

可替换值
本文档有时会使用可替换值，您必须将这些值替换为特定于环境的值。可替换的值为小写，以尖括号 (**<>**)括起，样式则使用斜体和 **monospace** 字体。用下划线(_)分隔多个词语。

例如，在以下命令中，将 ***<install_dir>*** 替换为您自己的目录名称。

```
$ <install_dir>/bin/artemis create mybroker
```

第 2 章 在网络连接中配置接收器和连接器

AMQ Broker 中使用两种连接：网络连接和 *in-VM* 连接。当双方位于不同的虚拟机（无论是位于同一服务器还是物理远程）中时，会使用网络连接。当客户端（应用程序或服务器）位于与代理相同的虚拟机上时，会使用 in-VM 连接。

网络连接使用 Netty。Netty 是一个高性能的低级别网络库，它允许以几种不同的方式配置网络连接；使用 Java IO 或 NIO、TCP 套接字、SSL/TLS 或通过 HTTP 或 HTTPS 隧道。Netty 还允许单个端口用于所有消息传递协议。代理将自动检测正在使用的协议，并将传入的消息定向到适当的处理程序，以进行进一步处理。

网络连接的 URI 确定其类型。例如，在 URI 中指定 **vm** 会创建一个 in-VM 连接：

```
<acceptor name="in-vm-example">vm://0</acceptor>
```

或者，在 URI 中指定 **tcp** 会创建一个网络连接。例如：

```
<acceptor name="network-example">tcp://localhost:61617</acceptor>
```

以下章节描述了网络连接和内连接所需的两个重要配置元素： *acceptors* 和 *connectors*。这些部分演示了如何为 TCP、HTTP 和 SSL/TLS 网络连接配置 acceptors 和连接器，以及 in-VM 连接。

2.1. 关于接收器

acceptors 定义如何将连接连接到代理。每个接收器都定义了客户端可用于建立连接的端口和协议。简单的接收器配置如下所示：

```
<acceptors>
  <acceptor name="example-acceptor">tcp://localhost:61617</acceptor>
</acceptors>
```

您在代理配置中定义的每个 **acceptor** 元素都包含在一个 **acceptors** 元素中。您可以为代理定义的接收器数量没有上限。默认情况下，AMQ Broker 包括每个支持的消息传递协议的接收器，如下所示：

```
<configuration ...>
  <core ...>
    ...
    <acceptors>
      ...
      <!-- Acceptor for every supported protocol -->
      <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- AMQP Acceptor. Listens on default AMQP port for AMQP traffic -->
      <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- STOMP Acceptor -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true</acceptor>
```

```
        <!-- HornetQ Compatibility Acceptor. Enables HornetQ Core and STOMP for legacy HornetQ
clients. -->
        <acceptor name="hornetq">tcp://0.0.0.0:5445?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.;protocols=HORNETQ,STOMP;useEpoll=true</a
cceptor>

        <!-- MQTT Acceptor -->
        <acceptor name="mqtt">tcp://0.0.0.0:1883?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</accept
or>
      </acceptors>
      ...
  </core>
</configuration>
```

2.2. 配置接收器

以下示例演示了如何配置 acceptor。

流程

1. 打开 & ***lt;broker_instance_dir>*** **/etc/broker.xml** 配置文件。
2. 在 **acceptors** 元素中，添加新的 **acceptor** 元素。指定协议，以及代理上的端口。例如：

```
<acceptors>
  <acceptor name="example-acceptor">tcp://localhost:61617</acceptor>
</acceptors>
```

   前面的示例为 TCP 协议定义了一个 acceptor。代理在端口 61617 上侦听使用 TCP 的客户端连接。

3. 将键值对附加到为接收器定义的 URI。使用分号(**;**)来分隔多个键值对。例如：

```
<acceptor name="example-acceptor">tcp://localhost:61617?sslEnabled=true;key-store-
path=</path/to/key_store></acceptor>
```

   配置现在定义一个使用 TLS/SSL 的接收器，并定义所需密钥存储的路径。

其他资源

- 有关接收器和连接器的可用配置选项的详情，请参考 附录 A, *acceptor 和 Connector 配置参数*。

2.3. 关于连接器

acceptors 定义代理如何*接受*连接，客户端使用的连接器定义它们如何*连接到*一个代理。

当代理本身作为客户端时，会在代理上配置连接器。例如：

- 当代理桥接到另一个代理时
- 当代理在集群中使用部分时

简单的连接器配置如下所示。

```
<connectors>
  <connector name="example-connector">tcp://localhost:61617</connector>
</connectors>
```

2.4. 配置连接器

以下示例演示了如何配置连接器。

流程

1. 打开 &*lt;broker_instance_dir>* **/etc/broker.xml** 配置文件。
2. 在 **连接器** 元素中，添加新的 **连接器** 元素。指定协议，以及代理上的端口。例如：

```
<connectors>
  <connector name="example-connector">tcp://localhost:61617</connector>
</connectors>
```

前面的示例为 TCP 协议定义了连接器。客户端可以使用连接器配置来通过 TCP 协议连接到端口 61617 上的代理。代理本身也可以将此连接器用于出站连接。

3. 将键值对附加到为连接器定义的 URI。使用分号(**;**)来分隔多个键值对。例如：

```
<connector name="example-connector">tcp://localhost:61616?tcpNoDelay=true</connector>
```

配置现在定义了一个连接器，将 **tcpNoDelay** 属性的值设置为 **true**。将此属性的值设置为 **true** 可为连接关闭 Nagle 的算法。Nagle 的算法是通过延迟小数据数据包的传输，并将它们整合到大型数据包中来提高 TCP 连接效率的算法。

其他资源

- 有关接收器和连接器的可用配置选项的详情，请参考 附录 A, *acceptor 和 Connector 配置参数*。
- 要了解如何在 AMQ 核心协议 JMS 客户端中配置代理连接器，请参阅 AMQ Core Protocol JMS 文档中的 配置代理连接器。

2.5. 配置 TCP 连接

AMQ Broker 使用 Netty 提供基于 TCP 的基本连接，可以配置为使用阻止 Java IO 或较新的非阻塞 Java NIO。首选 Java NIO 通过多个并发连接实现更好的可扩展性。但是，当您担心支持数千个并发连接时，使用旧的 IO 有时可以为您提供比 NIO 更高的延迟。

如果您在不受信任的网络中运行连接，您应该注意 TCP 网络连接是未加密的。如果安全性为优先级，您可能需要考虑使用 SSL 或 HTTPS 配置来加密通过此连接发送的消息。详情请查看 第 5.1 节 “保护连接”。

在使用 TCP 连接时，所有连接都由客户端启动。代理不会启动到客户端的任何连接。这适用于强制从一个方向启动连接的防火墙策略。

对于 TCP 连接，主机和连接器 URI 端口定义用于连接的地址。

以下示例演示了如何配置 TCP 连接。

先决条件

- 您应该熟悉配置接收器和连接器。如需更多信息，请参阅：
  - 第 2.2 节 “配置接收器”
  - 第 2.4 节 “配置连接器”

流程

1. 打开 &*lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。
2. 添加新接受者或修改现有的接收器。在连接 URI 中，指定 **tcp** 作为协议。在代理上包括 IP 地址或主机名以及端口。例如：

```
<acceptors>
  <acceptor name="tcp-acceptor">tcp://10.10.10.1:61617</acceptor>
  ...
</acceptors>
```

   根据上例，代理接受来自连接到 IP 地址 **10.10.10.1** 的端口 **61617** 的客户端的 TCP 通信。

3. （可选）您可以以类似的方式配置连接器。例如：

```
<connectors>
  <connector name="tcp-connector">tcp://10.10.10.2:61617</connector>
  ...
</connectors>
```

   在向指定的 IP 和端口 **10.10.10.2:61617** 进行 TCP 连接时，上例中的连接器由客户端甚至代理本身引用。

其他资源

- 有关 TCP 连接可用配置选项的详情，请参考 附录 A, *acceptor 和 Connector 配置参数*。

2.6. 配置 HTTP 连接

HTTP 连接通过 HTTP 协议隧道数据包，在防火墙只允许 HTTP 流量的情况下非常有用。AMQ Broker 自动检测是否使用 HTTP，因此为 HTTP 配置网络连接与为 TCP 配置连接相同。

先决条件

- 您应该熟悉配置接收器和连接器。如需更多信息，请参阅：
  - 第 2.2 节 “配置接收器”
  - 第 2.4 节 “配置连接器”

流程

1. 打开 &*lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。
2. 添加新接受者或修改现有的接收器。在连接 URI 中，指定 **tcp** 作为协议。在代理上包括 IP 地址或主机名以及端口。例如：

```
<acceptors>
```

```
  <acceptor name="http-acceptor">tcp://10.10.10.1:80</acceptor>
  ...
</acceptors>
```

根据前面的示例，代理接受来自连接到 IP 地址 **10.10.10.1** 的客户端的 HTTP 通信。**代理自动检测 HTTP 协议正在使用，并相应地与客户端通信。**

3. （可选）您可以以类似的方式配置连接器。例如：

```
<connectors>
  <connector name="http-connector">tcp://10.10.10.2:80</connector>
  ...
</connectors>
```

使用上例中显示的连接器，代理在 IP 地址 10.10.10.2 的端口 80 上创建一个出站 HTTP 连接。

其他资源

- HTTP 连接使用与 TCP 相同的配置参数，但它也具有自己的一些配置参数。有关 HTTP 连接的所有可用配置选项的详情，请参考 附录 A, *acceptor 和 Connector 配置参数*。
- 有关演示如何使用 HTTP 的完整工作示例，请参阅 JMS HTTP 示例。

2.7. 配置安全网络连接

您可以使用 TLS/SSL 保护网络连接。更多信息请参阅 第 5.1 节 “保护连接”。

2.8. 配置 IN-VM 连接

当多个代理位于同一虚拟机上时，您可以使用 in-VM 连接，例如作为高可用性(HA)配置的一部分。在虚拟机内连接也可以由与代理在同一 JVM 中运行的本地客户端使用。

先决条件

- 您应该熟悉配置接收器和连接器。如需更多信息，请参阅：

- 第 2.2 节 “配置接收器”
- 第 2.4 节 “配置连接器”

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加新接受者或修改现有的接收器。在连接 URI 中，指定 vm 作为协议。例如：

```
<acceptors>
  <acceptor name="in-vm-acceptor">vm://0</acceptor>
  ...
</acceptors>
```

根据上例中的 acceptor，代理接受 ID 为 0 的代理的连接。其他代理必须在同一虚拟机中运行。

3. （可选）您可以以类似的方式配置连接器。例如：

```
<connectors>
  <connector name="in-vm-connector">vm://0</connector>
  ...
</connectors>
```

上例中的连接器定义了客户端如何建立与 ID 为 0 的代理的连接，该代理与客户端在同一虚拟机中运行的 ID 为 0。客户端可以是应用程序或其他代理。

第 3 章 在网络连接中配置消息传递协议

AMQ Broker 具有可插拔协议架构，以便您可以轻松为网络连接启用一个或多个协议。

代理支持以下协议：

- AMQP
- MQTT
- OpenWire
- STOMP

注意

除了上述协议外，代理还支持自己的原生协议，称为 "Core"。此协议的过去版本被称为 "HornetQ"，并被 Red Hat JBoss Enterprise Application Platform 使用。

3.1. 将网络连接配置为使用消息传递协议

在使用协议前，您必须将协议与网络连接关联。（请参阅 第 2 章 *在网络连接中配置接收器和连接器* 以了解有关如何创建和配置网络连接的更多信息。）位于 < *broker_instance_dir*> /etc/broker.xml 文件中的默认配置包括已定义了几个连接。为方便起见，AMQ Broker 包括每个支持的协议的接收器，以及支持所有协议的默认接收器。

默认接收器概述

下面显示了在 broker.xml 配置文件中默认包括的 acceptors。

```
<configuration>
  <core>
    ...
    <acceptors>

      <!-- All-protocols acceptor -->
      <acceptor name="artemis">tcp://0.0.0.0:61616?
```

```
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- AMQP Acceptor. Listens on default AMQP port for AMQP traffic -->
      <acceptor name="amqp">tcp://0.0.0.0:5672?tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- STOMP Acceptor -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true</acceptor>

      <!-- HornetQ Compatibility Acceptor. Enables HornetQ Core and STOMP for legacy HornetQ clients. -->
      <acceptor name="hornetq">tcp://0.0.0.0:5445?anycastPrefix=jms.queue.;multicastPrefix=jms.topic.;protocols=HORNETQ,STOMP;useEpoll=true</acceptor>

      <!-- MQTT Acceptor -->
      <acceptor name="mqtt">tcp://0.0.0.0:1883?tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</acceptor>

    </acceptors>
    ...
  </core>
</configuration>
```

在给定网络连接器中启用协议的唯一要求是，将 protocol 参数添加到接受者的 URI 中。参数的值必须是用逗号分开的协议名称列表。如果 URI 中省略了 protocol 参数，则会启用所有协议。

例如，要使用 AMQP 协议在端口 3232 上创建接收消息的接收器，请按照以下步骤操作：

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 在 < acceptors& gt; 小节中添加以下行：

```
<acceptor name="ampq">tcp://0.0.0.0:3232?protocols=AMQP</acceptor>
```

默认 acceptors 中的附加参数

在最小的 acceptor 配置中，您可以将协议指定为连接 URI 的一部分。但是，broker.xml 配置文件中的默认接收器配置了一些额外的参数。下表详细介绍了为默认接收器配置的附加参数。

acceptor (s)	参数	描述
all-protocols acceptor AMQP STOMP	tcpSendBufferSize	TCP 发送缓冲区的大小（以字节为单位）。默认值为 **32768**。
	tcpReceiveBufferSize	TCP 接收缓冲区的大小（以字节为单位）。默认值为 **32768**。 应根据网络的带宽和延迟调整 TCP 缓冲区大小。 总的 TCP 发送/接收缓冲区大小应计算为： buffer_size = bandwidth * RTT. 其中带宽每秒字节数，网络往返用时(RTT)以秒为单位。可以使用 **ping** 实用程序轻松测量 RTT。 对于快速网络，您可能想要从默认值增加缓冲区大小。
all-protocols acceptor AMQP STOMP HornetQ MQTT	useEpoll	如果使用支持它的系统(Linux)，请使用 Netty epoll。Netty 原生传输比 NIO 传输提供更好的性能。此选项的默认值为 **true**。如果将选项设置为 **false**，则使用 NIO。
all-protocols acceptor AMQP	amqpCredits	AMQP 生成者可以发送的最大消息数，无论消息总数如何。默认值为 **1000**。 如需了解更多有关如何使用信用来阻止 AMQP 信息的信息，请参阅 第 7.3.2 节 “阻塞 AMQP 生成者”。
all-protocols acceptor AMQP	amqpLowCredits	代理 replenishes producer 信用时的较低阈值。默认值为 **300**。当生成者达到这个阈值时，代理会发送生成者有足够的信来恢复 **amqpCredits** 值。 如需了解更多有关如何使用信用来阻止 AMQP 信息的信息，请参阅 第 7.3.2 节 “阻塞 AMQP 生成者”。
HornetQ 兼容性接收器	anycastPrefix	当连接到一个使用 **anycast** 和 **multicast** 地址时，客户端用来指定 **anycast** 路由类型的前缀。默认值为 **jms.queue**。 有关配置前缀以便客户端在连接到地址时指定路由类型的详情，请参考 第 4.6 节 “在接受器配置中添加路由类型”。
	multicastPrefix	当连接到一个使用 **anycast** 和 **multicast** 地址时，客户端用来指定 **multicast** 路由类型的前缀。默认值为 **jms.topic**。 有关配置前缀以便客户端在连接到地址时指定路由类型的详情，请参考 第 4.6 节 “在接受器配置中添加路由类型”。

其他资源

- 有关您可以为 Netty 网络连接配置的其他参数的详情，请参考 附录 A, *acceptor 和 Connector 配置参数*。

3.2. 使用带有网络连接的 AMQP

代理支持 AMQP 1.0 规格。AMQP 链接是源与目标之间消息的单向协议，即客户端和代理。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加或配置一个 acceptor 来接收 AMQP 客户端，方法是包括一个值为 AMQP 的 protocols 参数作为 URI 的一部分。如下例所示：

```
<acceptors>
  <acceptor name="amqp-acceptor">tcp://localhost:5672?protocols=AMQP</acceptor>
  ...
</acceptors>
```

在上例中，代理接受端口 5672 上的 AMQP 1.0 客户端，这是默认的 AMQP 端口。

AMQP 链接有两个端点，即发送者和接收器。当发送者传输消息时，代理将其转换为内部格式，以便将其转发到代理上的目的地。接收器连接到代理中的目的地，并在发送前将消息转换为 AMQP。

如果 AMQP 链接是动态的，则会创建一个临时队列，并将远程源或远程目标地址设置为临时队列的名称。如果链接不是动态的，则使用远程目标或源的地址用于队列。如果远程目标或源不存在，则会发送异常。

链接目标也可以是协调器，用于将底层会话作为事务处理，可以是回滚或提交它。

注意

AMQP 允许每个会话使用多个事务，amqp:multi-txns-per-ssn，但当前版本的 AMQ Broker 将仅支持每个会话的单一事务。

注意

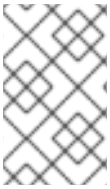
AMQP 中的分布式事务(XA)详情在规格的 1.0 版本中没有提供。如果您的环境需要支持分布式事务，建议您使用 AMQ 核心协议 JMS。

有关协议及其功能的更多信息，请参阅 AMQP 1.0 规格。

3.2.1. 使用 AMQP 链接作为主题

与 JMS 不同，AMQP 协议不包括主题。但是，仍然可以将 AMQP 用户或接收器视为订阅，而不只是队列中的用户。默认情况下，任何附加到前缀 jms.topic. 的地址的链接都被视为订阅，并创建一个订阅队列。订阅队列是持久的或易失性，具体取决于如何配置Terminus Durability，如下表中捕获：

为仅多播队列... 创建这类订阅	将 Terminus Durability 设置为 this...
durable	UNSETTLED_STATE 或 CONFIGURATION
非持久性	NONE



注意

持久队列的名称由容器 ID 和链接名称组成，如 my-container-id:my-link-name。

AMQ Broker 还支持 qpid-jms 客户端，并将遵守其主题的使用，而不考虑用于地址的前缀。

3.2.2. 配置 AMQP 安全性

代理支持 AMQP SASL 身份验证。有关如何在代理上配置基于 SASL 的身份验证的更多信息，请参阅安全性。

3.3. 使用带有网络连接的 MQTT

代理支持 MQTT v3.1.1 和 v5.0 （以及旧的 v3.1 代码消息格式）。MQTT 是一个轻量级、客户端到服务器，发布/订阅消息传递协议。MQTT 减少了消息传递开销和网络流量，以及客户端的代码占用。因此，MQTT 非常适合受限设备，如传感器和参与者，并快速成为物联网(IoT)的事实标准通信协议。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加启用了 MQTT 协议的接收器。例如：

```
<acceptors>
  <acceptor name="mqtt">tcp://localhost:1883?protocols=MQTT</acceptor>
  ...
</acceptors>
```

MQTT 带有一些有用的功能，其中包括：

服务质量

每个消息都可以定义与其关联的服务质量。代理将尝试在定义最高服务质量级别向订阅者发送信息。

保留消息

可以为特定地址保留消息。该地址的新订阅者在任何其他消息之前收到最后的保留消息，即使客户端连接之前发送了保留的消息。

保留的消息会存储在名为 sys.mqtt.<*topic name*& gt; 并保留在队列中的队列中，直到客户端删除保留的消息，或者如果配置了到期，直到消息过期为止。当队列为空时，队列不会被删除，直到您显式删除它。例如，以下配置会删除队列：

```
<address-setting match="$sys.mqtt.retain.#">
   <auto-delete-queues>true</auto-delete-queues>
   <auto-delete-addresses>true</auto-delete-addresses>
</address-setting>
```

通配符订阅

MQTT 地址是分级的，类似于文件系统的层次结构。客户端可以订阅特定主题或整个层次结构分支。

将消息

客户端可以设置"will 消息"作为其连接数据包的一部分。如果客户端正常断开连接，代理会将消息发布到指定的地址。其他订阅者收到该消息，并可相应地做出反应。

有关 MQTT 协议的更多信息，请参阅规范。

- MQTT 3.11 规格
- MQTT 5.0 规格

3.3.1. 配置 MQTT 属性

您可以将键值对附加到 MQTT acceptor 来配置连接属性。例如：

```
<acceptors>
  <acceptor name="mqtt">tcp://localhost:1883?
protocols=MQTT;receiveMaximum=50000;topicAliasMaximum=50000;maximumPacketSize;13
4217728;
serverKeepAlive=30;closeMqttConnectionOnPublishAuthorizationFailure=false</acceptor>
  ...
</acceptors>
```

receiveMaximum

通过在需要确认前指定代理可以从客户端接收的最大 QoS 1 和 2 消息数来启用流控制。默认值为 65535。值 -1 禁用从客户端到代理的 flow-control。这与将值设置为 0 的影响相同，但会减少 CONNACK 数据包的大小。

topicAliasMaximum

为客户端指定代理支持的最大别名数。默认值为 65535。值 -1 会阻止代理告知客户端主题别名限制。这与将值设置为 0 的效果相同，但会减少 CONNACK 数据包的大小。

maximumPacketSize

指定代理可以从客户端接受的最大数据包大小。默认值为 268435455。值 -1 会阻止代理告知客户端的最大数据包大小，这意味着在传入的数据包的大小中不会强制实施任何限制。

serverKeepAlive

指定代理保持不活跃客户端连接打开的持续时间。只有在为客户端配置了 keep-alive 值或为客户端配置的值时，配置的值才会应用到连接。默认值为 60 秒。-1 表示代理总是接受客户端的保留值（即使该值为 0）。

closeMqttConnectionOnPublishAuthorizationFailure

默认情况下，如果因为缺少授权而导致 PUBLISH 数据包失败，代理会关闭网络连接。如果您希望代理发送正确认而不是关闭网络连接，请将 closeMqttConnectionOnPublishAuthorizationFailure 设置为 false。

3.4. 使用带有网络连接的 OPENWIRE

代理支持 OpenWire 协议，它允许 JMS 客户端直接与代理通信。使用此协议与旧版本的 AMQ Broker 通信。

目前，AMQ Broker 支持只使用标准 JMS API 的 OpenWire 客户端。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加或修改接受 者，使其包含 OPENWIRE 作为 protocol 参数的一部分，如下例所示：

```
<acceptors>
  <acceptor name="openwire-acceptor">tcp://localhost:61616?
protocols=OPENWIRE</acceptor>
  ...
</acceptors>
```

在前面的示例中，代理将监听端口 61616 用于传入的 OpenWire 命令。

如需了解更多详细信息，请参阅 Openwire 示例。

3.5. 使用带有网络连接的 STOMP

STOMP 是一个基于文本的线协议，允许 STOMP 客户端与 STOMP Broker 进行通信。代理支持 STOMP 1.0、1.1 和 1.2。STOMP 客户端可用于多种语言和平台，使其成为可互操作性的理想选择。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 配置现有的 acceptor 或创建一个新的，其中包含值为 STOMP 的 protocol 参数，如下所示：

```
<acceptors>
```

```
  <acceptor name="stomp-acceptor">tcp://localhost:61613?protocols=STOMP</acceptor>
  ...
</acceptors>
```

在上例中，代理接受端口 61613 上的 STOMP 连接，这是默认设置。

有关如何使用 STOMP 配置代理的示例，请参阅 STOMP 示例。

3.5.1. STOMP 限制

使用 STOMP 时，会有以下限制：

1. 代理目前不支持虚拟主机，这意味着 CONNECT 帧中的 host 标头被忽略。

2. 消息确认不是事务性。ACK 框架不能是事务的一部分，如果设置了其 事务 标头，则忽略该帧。

3.5.2. 为 STOMP 消息提供 ID

当通过 JMS 消费者或 QueueBrowser 接收 STOMP 消息时，消息不包含任何 JMS 属性，如 JMSMessageID。但是，您可以使用代理参数参数为每个传入的 STOMP 消息设置消息 ID。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 对于用于 STOMP 连接的 acceptor，将 stompEnableMessageId 参数设置为 true，如下例所示：

```
<acceptors>
  <acceptor name="stomp-acceptor">tcp://localhost:61613?
protocols=STOMP;stompEnableMessageId=true</acceptor>
  ...
</acceptors>
```

通过使用 stompEnableMessageId 参数，使用此接受者发送的每个 stomp 消息都添加了额外的属性。属性键是 amq-message-id，值是带有 "STOMP" 前缀的内部消息 id 的 String 表示，如下例所示：

```
amq-message-id : STOMP12345
```

如果配置中没有指定 **stompEnableMessageId**，则默认值为 **false**。

3.5.3. 将连接时间设置为 live

STOMP 客户端必须在关闭连接前发送 **DISCONNECT** 帧。这允许代理关闭任何服务器端资源，如会话和消费者。但是，如果 **STOMP** 客户端在没有发送 **DISCONNECT** 帧的情况下退出，或者如果失败，代理将无法立即知道客户端是否处于活动状态。因此，**STOMP** 连接被配置为 **1** 分钟的"生存时间" **(TTL)**。这意味着，如果代理闲置超过 **1** 分钟，代理会停止与 **STOMP** 客户端的连接。

流程

1. 打开 **&** ***lt;broker_instance_dir>*** **/etc/broker.xml** 配置文件。

2. 将 **connectionTTL** 参数添加到用于 **STOMP** 连接的 **acceptor** 的 **URI** 中，如下例所示：

```
<acceptors>
  <acceptor name="stomp-acceptor">tcp://localhost:61613?
protocols=STOMP;connectionTTL=20000</acceptor>
  ...
</acceptors>
```

在前面的示例中，使用 **stomp-acceptor** 的任何 **stomp** 连接都将其 **TTL** 设置为 **20** 秒。

注意

STOMP 协议的版本 **1.0** 不包含任何心跳帧。因此，用户的责任可确保数据在 **connection-ttl** 中发送，否则代理会假定客户端死机并清理服务器端资源。对于 **1.1** 版本，您可以使用 **core-beats** 来维护 **stomp** 连接的生命周期。

将代理默认时间覆盖为 live

如前文所述，**STOMP** 连接的默认 **TTL** 为一分钟。您可以通过在代理配置中添加 **connection-ttl-override** 属性来覆盖这个值。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加 connection-ttl-override 属性，并为新默认值提供一个值（毫秒）。它属于 < core> 小节，如下所示。

```
<configuration ...>
  ...
  <core ...>
    ...
    <connection-ttl-override>30000</connection-ttl-override>
    ...
  </core>
<configuration>
```

在前面的示例中，STOMP 连接的默认生存时间(TTL)被设置为 30 秒，30000 毫秒。

3.5.4. 从 JMS 发送和使用 STOMP 消息

STOMP 主要是一个基于文本的协议。为了便于与 JMS 互操作，STOMP 实施检查是否存在 content-length 标头，以确定如何将 STOMP 消息映射到 JMS。

如果您希望 STOMP 消息映射到 ...	消息应该....。
JMS TextMessage	不包含 content-length 标头。
JMS BytesMessage	包括一个 content-length 标头。

将 JMS 消息映射到 STOMP 时应用相同的逻辑。STOMP 客户端可以确认存在 content-length 标头来确定消息正文的类型（字符串或字节数）。

有关消息标头的更多信息，请参阅 STOMP 规格。

3.5.5. 将 STOMP 目的地映射到 AMQ Broker 地址和队列

在发送消息和订阅时，STOMP 客户端通常包含 一个目标 标头。目的地名称是字符串值，它们映射到代理上的目的地。在 AMQ Broker 中，这些目的地映射到 addresses 和 queues。有关目标帧的更多信息，请参阅 STOMP 规格。

例如，一个发送以下消息的 STOMP 客户端（包括headers 和 body）：

```
SEND
destination:/my/stomp/queue

hello queue a
^@
```

在这种情况下，代理会将消息转发到与地址 /my/stomp/queue 关联的任何队列。

例如，当 STOMP 客户端使用 SEND 帧发送消息时，指定的目的地映射到地址。

当客户端发送 SUBSCRIBE 或 UNSUBSCRIBE 帧时，它的工作方式相同，但在本例中为 AMQ Broker 将 目的地 映射到一个队列。

```
SUBSCRIBE
destination: /other/stomp/queue
ack: client

^@
```

在前面的示例中，代理会将 目的地映射 到队列 /other/stomp/queue。

将 STOMP 目的地映射到 JMS 目的地

JMS 目的地也映射到代理地址和队列。如果要使用 STOMP 将消息发送到 JMS 目的地，STOMP 目标必须遵循相同的约定：

- 通过 jms.queue 添加队列名称来发送或订阅 JMS Queue。例如，要将信息发送到 orders JMS Queue，STOMP 客户端需要发送帧：

  ```
  SEND
  destination:jms.queue.orders
  hello queue orders
  ^@
  ```

- 通过添加主题名称作为 jms.topic. 前，发送或订阅 JMS 主题。例如，要订阅 库存 JMS 主题，STOMP 客户端必须发送类似如下的帧：

```
SUBSCRIBE
destination:jms.topic.stocks
^@
```

第 4 章 配置地址和队列

4.1. 地址、队列和路由类型

在 AMQ Broker 中，寻址模型包含三个主要概念： *地址*、*队列*和 *路由类型*。

地址 代表消息传递端点。在配置中，为典型的地址指定唯一名称、一个或多个队列，以及路由类型。

队列与 地址关联。每个地址可以有多个队列。传入的消息与地址匹配后，消息将发送到一个或多个队列，具体取决于配置的路由类型。队列可以配置为自动创建和删除。您还可以配置一个地址（以及其关联队列）作为 *durable*。持久队列中的消息可以保留崩溃或重启代理，只要队列中的消息也持久保留。相反，非持久队列中的消息在崩溃或代理重启后不会保留，即使消息本身是持久的。

路由类型决定了 消息如何发送到与地址关联的队列。在 AMQ Broker 中，您可以使用两个不同的路由类型配置地址，如表中所示。

如果您希望消息路由到...	使用此路由类型...
匹配地址中的单个队列，以点到点的方式	**anycast**
匹配地址中的每个队列，采用发布订阅方式	**multicast**

注意

地址必须至少有一个定义的路由类型。

每个地址 *可以*定义多个路由类型，但不推荐这样做。

如果一个地址 *同时*定义了两个路由类型，并且客户端没有其首选项，则代理会默认使用 multicast 类型。

其他资源

- 有关配置的更多信息：
  - 点到点的消息使用 anycast 路由类型，请参阅 第 4.3 节 “为点到点消息传递配置地址”

-

- 使用 multicast 路由类型发布订阅消息传递，请参阅 第 4.4 节 “为发布订阅消息传递配置地址”

4.1.1. 地址和队列命名要求

在配置地址和队列时请注意以下要求：

- 为确保客户端可以连接到队列，无论客户端使用的有线协议，您的地址和队列名称 不应包含以下任意字符：

  & :: , ? >

- 数字符号（#）和星号（*）字符被保留为通配符表达式，在地址和队列名称中不应使用它们。更多信息请参阅 第 4.2.1 节 “AMQ Broker 通配符语法”。

- 地址和队列名称不应包含空格。

- 要分隔地址或队列名称中的词语，请使用配置的分隔符字符。默认分隔符字符是句点(.)。更多信息请参阅 第 4.2.1 节 “AMQ Broker 通配符语法”。

4.2. 将地址设置应用到一组地址

在 AMQ Broker 中，您可以使用通配符表达式来将 address-setting 元素中指定的配置应用到 *一组* 地址，来表示匹配的地址名称。

以下小节介绍了如何使用通配符表达式。

4.2.1. AMQ Broker 通配符语法

AMQ Broker 使用特定语法在地址设置中代表通配符。通配符也可以在安全设置中使用，也可在创建消费者时使用。

- 通配符表达式包含以句点分隔的词语(.)。
- 数字符号 (#) 和星号 (*) 字符还有特殊的用处，可以替代一个词，例如：
  - 数字符号字符表示"匹配任何 0 个或更多连续的词"。在表达式的末尾使用此内容。
  - 星号字符表示"匹配单个单词"。在您的表达式内使用此内容。

匹配不是按照字符进行的，而是根据分割的字符段进行的。例如，一个 address-setting 项被配置为匹配名称中带有 my 的队列将不匹配名为 myqueue 的队列。

当多个 address-setting 元素与地址匹配时，代理覆盖配置使用最低特定匹配项的配置作为基准。表达式中的字面字符比通配符更具体，星号 (*) 比数字符号更具体 (#)。例如，my.destination 和 my.* 都会匹配地址 my.destination。在这种情况下，代理首先应用 my memcached 下找到的配置，因为通配符表达式比字面更具体。接下来，代理覆盖 my.destination address 设置元素的配置，该配置会覆盖与 my constraint 共享的任何配置。例如，如果进行了以下配置，与 my.destination 关联的队列将 max-delivery-attempts 设置为 3，并将 last-value-queue 设置为 false。

```
<address-setting match="my.*">
   <max-delivery-attempts>3</max-delivery-attempts>
   <last-value-queue>true</last-value-queue>
</address-setting>
<address-setting match="my.destination">
   <last-value-queue>false</last-value-queue>
</address-setting>
```

下表中的示例说明了如何使用通配符匹配一组地址。

示例	描述
#	broker.xml 中使用的默认 address-setting。匹配每个地址。您可以继续应用此 catch-all，或者您可以根据需要为每个地址或一组地址添加新的 address-setting。
news.europe.#	匹配 news.europe,news.europe.sport,news.europe.politics.fr, but not news.usa 或 europe。
news.*	匹配 news.europe 和 news.usa，但不匹配 news.europe.sport。

示例	描述
news.*.sport	匹配 **news.europe.sport** 和 **news.usa.sport**，但不匹配 **news.europe.fr.sport**。

4.2.2. 配置字面匹配

在字面匹配中，通配符字符被视为字面字符，以匹配包含通配符的地址。例如，字面匹配中的 hash (#)字符可以匹配一个 订购地址。 8." 没有匹配的地址，如 orders.retail 或 orders.wholesale。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在配置字面匹配前，请使用 literal-match-markers 参数来定义限制字面匹配的字符。在以下示例中，括号用于限制字面匹配。

```
<core>
  ...
  <literal-match-markers>()</literal-match-markers>
  ...
</core>
```

3. 在定义了限制字面匹配的标记后，在 address 设置 match 参数指定匹配项 （包括标记）。以下示例为名为 orders. 的地址配置字面匹配，以启用该特定地址的指标。

```
<address-settings>
  <address-setting match="(orders.#)">
    <enable-metrics>true</enable-metrics>
  </address-setting>
</address-settings>
```

4.2.3. 配置代理通配符语法

以下流程演示了如何自定义用于通配符地址的语法。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在配置中添加 < wildcard-addresses > 部分，如下例所示。

```
<configuration>
 <core>
  ...
  <wildcard-addresses> //
   <enabled>true</enabled> //
   <delimiter>,</delimiter> //
   <any-words>@</any-words> //
   <single-word>$</single-word>
  </wildcard-addresses>
  ...
 </core>
</configuration>
```

enabled

当设置为 **true** 时，指示代理使用您的自定义设置。

delimiter

提供自定义字符作为 **分隔符** 替代默认字符（.）。

any-words

作为 **any-words** 的值提供的字符代表“匹配 0 个或更多个序列”，并将替换默认的 **#**。在表达式的末尾使用此字符。

single-word

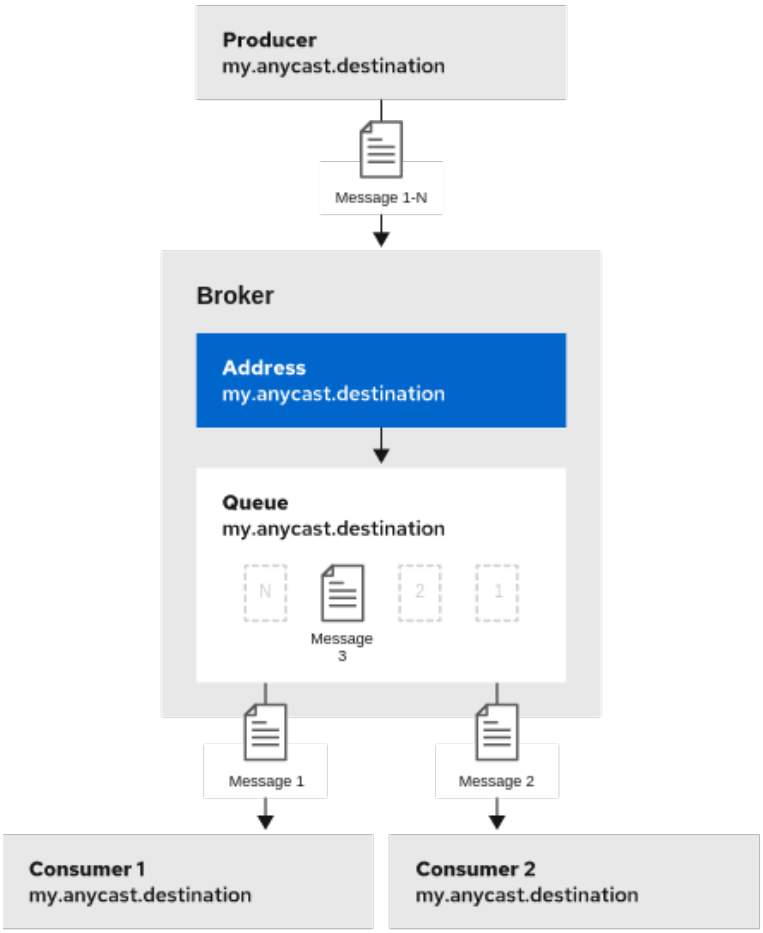
作为 **single-word** 的值提供的字符代表“匹配一个词”，并将替换默认的 ***。在您的表达式内使用此字符。

4.3. 为点到点消息传递配置地址

点到点的消息传递是一个常见情况，其中由生产者发送的消息只有一个消费者。AMQP 和 JMS 消息生产者和消费者可以利用点到点消息传递队列，例如：为确保在点到点的方式中队列与一个接收消息的地址相关联，在您的代理配置中为给定的 **address**项定义一个 **anycast** 路由类型。

当消息在一个使用 **anycast** 的地址上接收时，代理会找到与地址关联的队列，并将消息路由到它。然后，消费者可能会请求使用来自该队列的消息。如果多个消费者连接到同一队列，则消息会在消费者间平均分配，只要使用者可以平等地处理它们。

下图显示了点对点消息传递的示例。



4.3.1. 配置基本点对点消息传递

以下流程演示了如何使用单一队列为点到点消息传递配置地址。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 使用一个 address 所选的 queue 项环绕一个 anycast 配置项。确保 address 和 queue 元素的 name 属性的值都相同。例如：

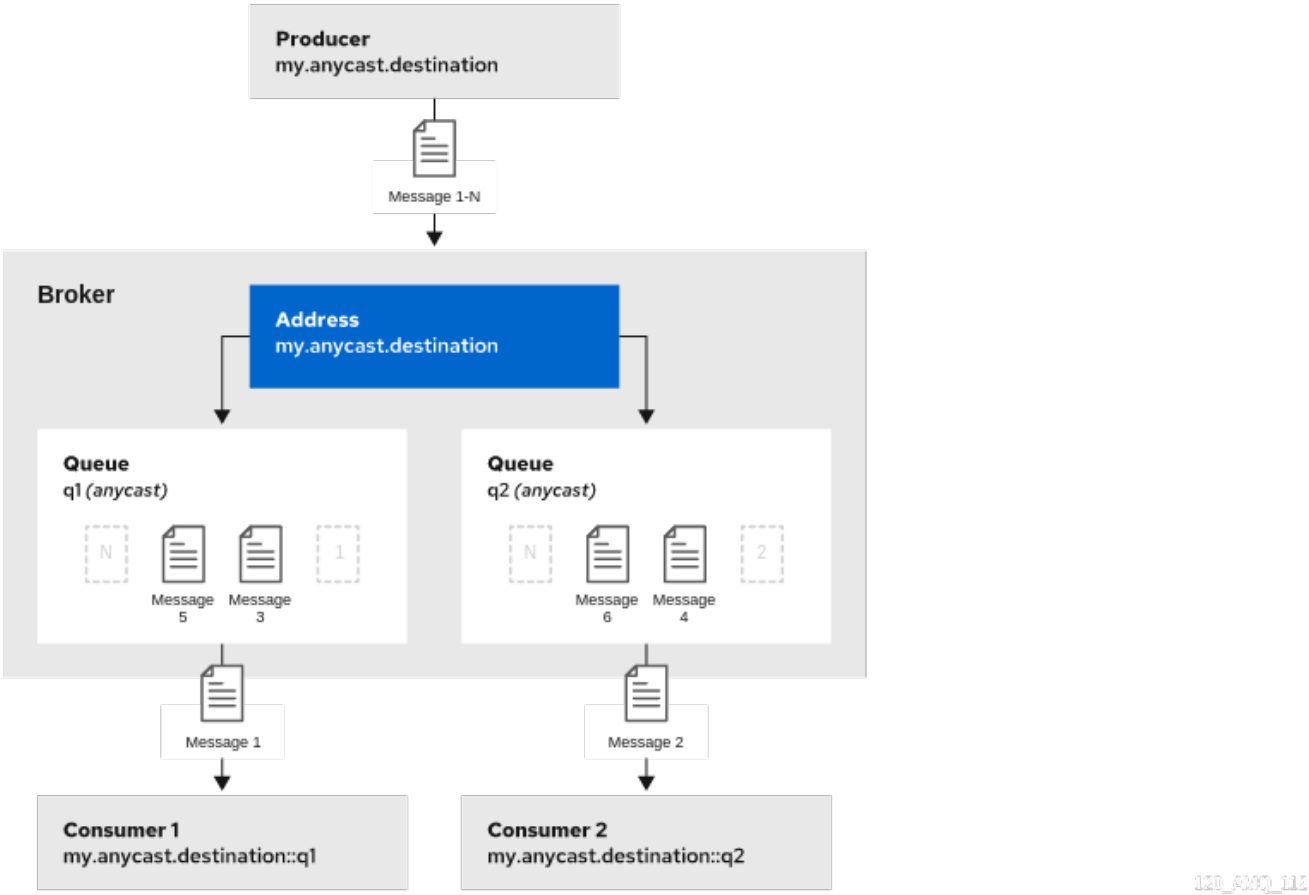
```
<configuration ...>
  <core ...>
    ...
    <address name="my.anycast.destination">
      <anycast>
        <queue name="my.anycast.destination"/>
      </anycast>
    </address>
  </core>
</configuration>
```

-

4.3.2. 为多个队列配置点对点消息传递

您可以在使用 anycast 路由类型的地址上定义多个队列。代理会在所有相关队列中平均分发发送到 anycast 地址的消息。通过指定一个 *完全限定域名* (FQQN)，您可以将客户端连接到特定的队列。如果多个消费者连接到同一队列，代理会在消费者之间均匀分发消息。

下图显示了使用两个队列的点对点消息传递的示例。



以下流程演示了如何为具有多个队列的地址配置点对点消息传递。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 使用在 address 项中的 queue 项环绕一个 anycast 配置项。例如：

```
<configuration ...>
```

```
<core ...>
  ...
  <address name="my.anycast.destination">
    <anycast>
      <queue name="q1"/>
      <queue name="q2"/>
    </anycast>
  </address>
</core>
</configuration>
```

如果您有一个配置，如上面在集群中的多个代理间镜像的配置，集群可以以不透明生产者和消费者的方式对点进行负载均衡。确切的行为取决于如何为集群配置消息负载均衡策略。

其他资源

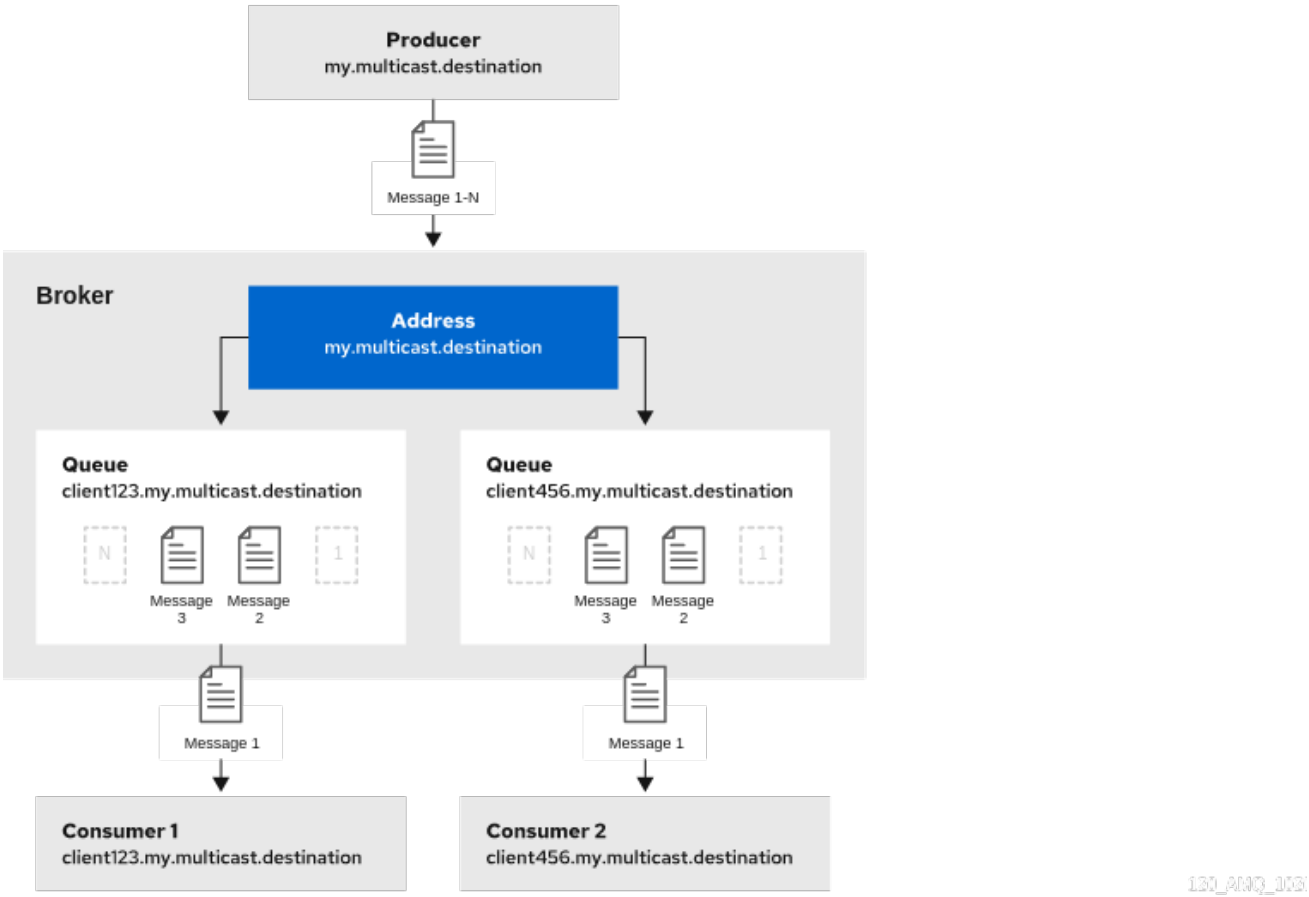
- 有关以下内容的更多信息：
  - 指定完全限定域名，请参阅 第 4.9 节 “指定完全限定的队列名称”。
  - 如何为代理集群配置消息负载均衡，请参阅 第 14.1.1 节 “代理集群如何平衡消息负载”。

4.4. 为发布订阅消息传递配置地址

在发布订阅场景中，信息会发送到订阅地址的每个消费者。JMS 主题和 MQTT 订阅是发布订阅消息传递的两个示例。为确保与地址关联的队列以发布订阅方式接收消息，您可以在代理配置中为给定 address 元素定义 多播 路由类型。

当消息在带有 multicast 路由类型的地址上收到时，代理会将消息的副本路由到与地址关联的每个队列。为减少复制开销，每个队列仅发送对消息 的引用，而不是完整副本。

下图显示了发布订阅消息传递的示例。



以下流程演示了如何为发布订阅消息传递配置地址。

流程

1. **打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。**

2. **向地址添加一个空的 multicast 配置元素。**

```
<configuration ...>
  <core ...>
    ...
    <address name="my.multicast.destination">
      <multicast/>
    </address>
  </core>
</configuration>
```

3. **（可选）将一个或多个 queue 项添加到地址中，并在它们间环绕 multicas 项。通常不需要这一步，因为代理会自动为客户端请求的每个订阅创建一个队列。**

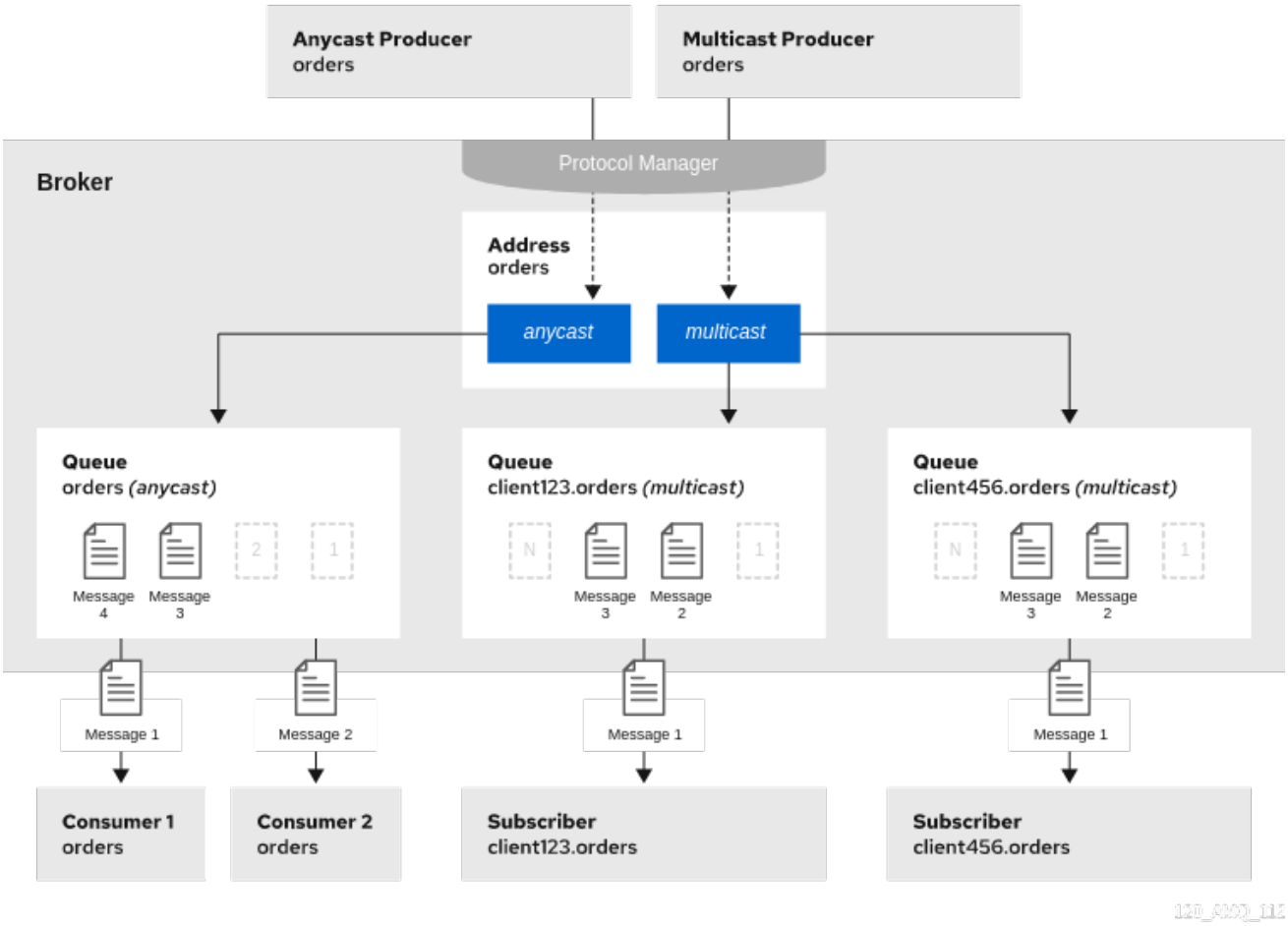
```
<configuration ...>
  <core ...>
    ...
    <address name="my.multicast.destination">
      <multicast>
        <queue name="client123.my.multicast.destination"/>
        <queue name="client456.my.multicast.destination"/>
      </multicast>
    </address>
  </core>
</configuration>
```

4.5. 为点到点和发布订阅消息传递配置地址

您还可以使用点到点 和发布 订阅语义配置地址。

通常不建议将地址配置为同时使用点到点和发布订阅模式。但是，如果需要，它 *可能会*有用。例如，一个名为 **orders** 的 **JMS** 队列，以及名为 **order** 的 **JMS** 主题。不同的路由类型使地址显示为不同的客户端连接。在这种情况下，由 **JMS** 队列生成者发送的消息使用 **anycast** 路由类型。**JMS** 主题制作者发送的消息使用 **multicast** 路由类型。当 **JMS** 主题消费者连接到代理时，它将连接到自己的订阅队列。但是，一个 **JMS** 队列消费者附加到 **anycast** 队列。

下图显示了点到点和发布订阅消息传递的示例。



以下流程演示了如何为点到点和发布订阅消息传递配置地址。

注意

这种情境的行为取决于所使用的协议。对于 JMS，主题和队列生产者和消费者之间有一个明显的区别，即逻辑简单。AMQP 等其他协议不会区分此区别。通过 AMQP 发送的消息以 anycast 和 multicast 路由，消费者默认为 anycast。更多信息请参阅 第 3 章 *在网络连接中配置消息传递协议*。

流程

1. 打开 & *lt;broker_instance_dir>* /etc/broker.xml 配置文件。

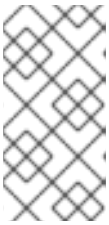
2. 使用在 address 项中的 queue 项环绕一个 anycast 配置项。例如：

```
<configuration ...>
  <core ...>
    ...
    <address name="orders">
```

```
      <anycast>
        <queue name="orders"/>
      </anycast>
    </address>
  </core>
</configuration>
```

3. 向地址添加一个空的 multicast 配置元素。

```
<configuration ...>
  <core ...>
    ...
    <address name="orders">
      <anycast>
        <queue name="orders"/>
      </anycast>
      <multicast/>
    </address>
  </core>
</configuration>
```



注意

通常，代理会根据需求创建订阅队列，因此不需要列出 multicast 元素中的特定队列元素。

4.6. 在接受器配置中添加路由类型

通常，如果一个消息由同时使用 anycast 和 multicast 的地址接收，一个 anycast 队列会接收消息，以及所有 multicast 队列。但是，当连接到地址时，客户端可以指定一个特定的前缀，以指定是否使用 anycast 或 multicast 连接。前缀是自定义值，使用代理配置中接受器的 URL 中的 anycastPrefix 和 multicastPrefix 参数指定。

以下流程演示了如何为给定接受器配置前缀。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 对于给定的接收器，若要配置 anycast 前缀，将 anycastPrefix 添加到配置的 URL。设置自定义值。例如：

```
<configuration ...>
  <core ...>
    ...
      <acceptors>
        <!-- Acceptor for every supported protocol -->
        <acceptor name="artemis">tcp://0.0.0.0:61616?
protocols=AMQP;anycastPrefix=anycast://</acceptor>
      </acceptors>
    ...
  </core>
</configuration>
```

根据前面的配置，acceptor 被配置为使用 anycast://（anycast 前缀）。如果客户端需要将信息只发送到 anycast队列，客户端可以指定 anycast://<my.destination>/。

3. 对于给定的接收器，若要配置 多播 前缀，请将 multicastPrefix 添加到配置的 URL 中。设置自定义值。例如：

```
<configuration ...>
  <core ...>
    ...
      <acceptors>
        <!-- Acceptor for every supported protocol -->
        <acceptor name="artemis">tcp://0.0.0.0:61616?
protocols=AMQP;multicastPrefix=multicast://</acceptor>
      </acceptors>
    ...
  </core>
</configuration>
```

根据前面的配置，acceptor 被配置为使用 multicast:// 作为 多播 前缀。如果客户端需要仅发送到多播队列的消息，客户端代码可以指定 multicast://<my.destination & gt;/。

4.7. 配置订阅队列

在大多数情况下，不需要手动创建订阅队列，因为协议管理器会在客户端第一次请求订阅地址时自动创建订阅队列。请参阅 第 4.8.3 节 “协议管理器和地址” 了解更多信息。对于持久订阅，生成的队列名称通常是客户端 ID 和地址的串联。

以下小节演示了如何根据需要手动创建订阅队列。

4.7.1. 配置持久订阅队列

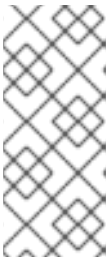
当队列配置为持久订阅时，代理会保存任何不活跃订阅者的消息，并在订阅时将它们传送到订阅者。因此，客户端保证在订阅后接收发送到队列的每个消息。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 将 **durable** 配置元素添加到所选队列中。将值设为 **true**。

```
<configuration ...>
  <core ...>
    ...
    <address name="my.durable.address">
      <multicast>
        <queue name="q1">
          <durable>true</durable>
        </queue>
      </multicast>
    </address>
  </core>
</configuration>
```



注意

因为队列默认是持久的，包括 **durable** 元素，并将值设为 **true** 并不严格需要创建持久队列。但是，明确包含该元素可让您稍后将队列的行为更改为非持久（如果需要）。

4.7.2. 配置非共享的持久订阅队列

代理可以被配置为阻止多个消费者在任何时间点上连接到队列。因此，以这种方式配置队列的订阅被视为"非共享"。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 将 **durable configuration** 元素添加到每个所选队列。将值设为 **true**。

```
<configuration ...>
  <core ...>
```

```
    ...
    <address name="my.non.shared.durable.address">
     <multicast>
      <queue name="orders1">
       <durable>true</durable>
      </queue>
      <queue name="orders2">
       <durable>true</durable>
      </queue>
     </multicast>
    </address>
   </core>
</configuration>
```

注意

因为队列默认是持久的，包括 durable 元素，并将值设为 true 并不严格需要创建持久队列。但是，明确包含该元素可让您稍后将队列的行为更改为非持久（如果需要）。

3. 将 max-consumers 属性添加到每个所选队列。设置值 1。

```
<configuration ...>
 <core ...>
  ...
  <address name="my.non.shared.durable.address">
   <multicast>
    <queue name="orders1" max-consumers="1">
     <durable>true</durable>
    </queue>
    <queue name="orders2" max-consumers="1">
     <durable>true</durable>
    </queue>
   </multicast>
  </address>
 </core>
</configuration>
```

4.7.3. 配置非持久订阅队列

非持久订阅通常由相关协议管理器管理，后者创建和删除临时队列。

但是，如果要手动创建类似非持久订阅队列的队列，您可以在队列上使用 purge-on-no-consumers 属性。当 purge-on-no-consumers 设为 true 时，队列不会开始接收消息，直到消费者连接为止。另外，当最后一个消费者与队列断开连接时，队列将被*清除*（即，其消息会被删除）。在新消费者连接到队列前，队列不会接收任何进一步的消息。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 为每个所选队列添加 purge-on-no-consumers 属性。将值设为 true。

```
<configuration ...>
  <core ...>
    ...
    <address name="my.non.durable.address">
      <multicast>
        <queue name="orders1" purge-on-no-consumers="true"/>
      </multicast>
    </address>
  </core>
</configuration>
```

4.8. 自动创建和删除地址和队列

您可以将代理配置为自动创建地址和队列，并在不再使用它们后删除它们。这样，在客户端可以连接到这个地址前，需要预先配置每个地址。

4.8.1. 自动创建和删除的配置选项

下表列出了配置 address-setting 元素时可用的配置元素，以自动创建和删除队列和地址。

如果您希望 address-setting 变为...	添加此配置...
当客户端发送消息到或尝试使用来自映射的队列的消息时，创建地址。	**auto-create-addresses**
当客户端发送消息到或尝试使用来自队列的消息时，创建队列。	**auto-create-queues**
当它不再有任何队列时，删除自动创建的地址。	**auto-delete-addresses**
当队列有 0 个消费者和 0 消息时，删除自动创建的队列。	**auto-delete-queues**
如果客户端没有指定，使用一个特定的路由类型。	**default-address-routing-type**

4.8.2. 配置自动创建和删除地址和队列

以下流程演示了如何配置自动创建和删除地址和队列。

流程

1. 打开 *<broker_instance_dir>* /etc/broker.xml 配置文件。

2. 配置 address-setting 以自动创建和删除。以下示例使用上表中提到的所有配置元素。

```
<configuration ...>
 <core ...>
  ...
  <address-settings>
    <address-setting match="activemq.#">
      <auto-create-addresses>true</auto-create-addresses>
      <auto-delete-addresses>true</auto-delete-addresses>
      <auto-create-queues>true</auto-create-queues>
      <auto-delete-queues>true</auto-delete-queues>
      <default-address-routing-type>ANYCAST</default-address-routing-type>
    </address-setting>
  </address-settings>
  ...
 </core>
</configuration>
```

address-setting

address-setting 元素的配置应用于与通配符地址 activemq. 4.1 匹配的任何地址或队列。

auto-create-addresses

当客户端请求连接到尚不存在的地址时，代理会创建地址。

auto-delete-addresses

当自动创建的地址不再关联任何队列时，会删除自动创建的地址。

auto-create-queues

当客户端请求连接到尚未存在的队列时，代理会创建队列。

auto-delete-queues

当自动创建的队列不再有任何消费者或消息时，将删除自动创建的队列。

default-address-routing-type

如果客户端在连接时没有指定路由类型，代理会在向地址传递消息时使用 ANYCAST。默认值为 MULTICAST。

其他资源

- 有关以下内容的更多信息：
  - 在配置地址时可以使用的通配符语法，请参阅 第 4.2 节 “将地址设置应用到一组地址”。
  - 路由类型，请参阅 第 4.1 节 “地址、队列和路由类型”。

4.8.3. 协议管理器和地址

称为 *协议管理器* 的组件将特定于协议的概念映射到 AMQ Broker 地址模型、队列和路由类型中使用的概念。在某些情况下，协议管理器可能会在代理上自动创建队列。

例如，当客户端发送带有地址 /house/room1/lights 和 /house/ room2/lights 的 MQTT 订阅数据包时，MQTT 协议管理器了解这两个地址需要 多播 语义。因此，协议管理器首先查找以确保两个地址都启用了 多播。如果没有，它会尝试动态创建它们。如果成功，协议管理器会为客户端请求的每个订阅创建特殊的订阅队列。

每个协议的行为略有不同。下表描述了在请求将帧订阅到各种 队列时 通常发生的情况。

如果队列是此类型...	协议管理器的典型操作是 ...
持久化订阅队列	查找适当的地址并确保启用了 **多播** 语义。然后，它会创建一个特殊的订阅队列，客户端 ID 和地址作为其名称，**multicast** 作为路由类型。 特殊名称允许协议管理器快速识别所需的客户端订阅队列，应稍后断开连接和重新连接。 当客户端取消订阅队列时。
临时订阅队列	查找适当的地址并确保启用了 **多播** 语义。然后，它会在这个地址下创建一个带有 **multicast** 路由类型的随机（读取 UUID）名称的队列。 当客户端断开队列时，

如果队列是此类型...	协议管理器的典型操作是 ...
点到点队列	查找适当的地址并确保启用了 **anycast** 路由类型。如果是，它旨在查找名称与地址相同的队列。如果不存在，它会查找第一个可用的队列。它不存在，然后它会自动创建队列（启用自动创建）。队列消费者绑定到此队列。 如果队列是自动创建的，则当没有消费者且没有消息后，它会自动删除。

4.9. 指定完全限定的队列名称

在内部，代理将客户端对地址的请求映射到特定的队列。代理决定客户端将消息发送到哪个队列，或从哪个队列接收消息。但是，更高级的用例可能需要客户端直接指定队列名称。在这些情况下，客户端可以使用 *完全限定的队列名称* (FQQN)。FQQN 包含地址名称和队列名称，用 :: 分隔。

以下流程演示了如何在连接到带有多个队列的地址时指定 FQQN。

先决条件

- 您已配置了两个或多个队列的地址，如下例所示。

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="my.address">
        <anycast>
          <queue name="q1" />
          <queue name="q2" />
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

流程

- 在客户端代码中，在从代理请求连接时使用地址名称和队列名称。使用两个冒号来分隔名称。例如：

```
String FQQN = "my.address::q1";
Queue q1 session.createQueue(FQQN);
MessageConsumer consumer = session.createConsumer(q1);
```

4.10. 配置分片队列

在队列间处理消息的常用模式，其中只有部分顺序是使用 *队列分片*。这意味着您可以定义一个作为单一逻辑队列的 anycast 地址，但它由多个底层物理队列支持。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加一个 address 元素并设置 name 属性。例如：

```
<configuration ...>
 <core ...>
  ...
  <addresses>
    <address name="my.sharded.address"></address>
  </addresses>
 </core>
</configuration>
```

3. 添加 anycast 路由类型，并包含所需的分片队列数量。在以下示例中，队列 q1, q2, 和 q3 添加为 anycast 的目的地。

```
<configuration ...>
 <core ...>
  ...
  <addresses>
    <address name="my.sharded.address">
      <anycast>
        <queue name="q1" />
        <queue name="q2" />
        <queue name="q3" />
      </anycast>
    </address>
  </addresses>
</core>
</configuration>
```

根据上述配置，发送到 my.sharded.address 的消息会在 q1, q2 和 q3 之间均匀分布。在使用完全限定域名(FQQN)时，客户端可以直接连接到特定的物理队列，并接收仅发送到该特定队列的消息。

要将特定消息绑定到特定的队列，客户端可以为每个消息指定一个消息组。代理将消息分组到同一队列，一个消费者全部处理它们。

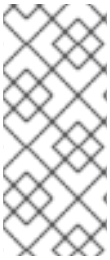
其他资源

- 有关以下内容的更多信息：
  - 完全限定的队列名称，请参阅 第 4.9 节 “指定完全限定的队列名称”
  - 消息分组，请参阅 AMQ Core Protocol JMS 文档中的使用消息组。https://access.redhat.com/documentation/zh-cn/red_hat_amq_clients/2023.q4/html-single/using_the_amq_core_protocol_jms_client/index.html#using_message_groups

4.11. 配置最后一个值队列

最后一个值队列 是一种队列类型，当将具有相同最后值键值的较新的消息放入队列中时，丢弃队列中的消息。通过此行为，最后一个值队列只为同一键的消息保留最后一个值。

最后一个值队列的简单用例是监控股票价格，其中只有特定库存的最新价值值得关注。



注意

如果没有配置的最后值键的消息被发送到最后一个值队列，代理会将这个消息作为 "normal" 信息处理。当带有配置的最后一个值键的新消息到达时，此类消息不会从队列中清除。

您可以单独配置最后一个值队列，或配置与一组地址关联的所有队列。

以下流程演示了如何以这些方式配置最后的值队列。

4.11.1. 单独配置最后一个值队列

以下流程演示了如何单独配置最后的值队列。

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。
2. 对于给定队列，添加 last-value-key 键并指定自定义值。例如：

```
<address name="my.address">
  <multicast>
    <queue name="prices1" last-value-key="stock_ticker"/>
  </multicast>
</address>
```

3. 或者，您可以配置使用默认值最后的值键名称 _AMQ_LVQ_NAME 的最后一个值队列。为此，请将 last-value 键添加到给定的队列中。将值设为 true。例如：

```
<address name="my.address">
  <multicast>
    <queue name="prices1" last-value="true"/>
  </multicast>
</address>
```

4.11.2. 为地址配置最后一个值队列

以下流程演示了如何为地址 *或一组* 地址配置最后的值队列。

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 address-setting 元素中，用于匹配地址，添加 default-last-value-key。指定自定义值。例如：

```
<address-setting match="lastValue">
  <default-last-value-key>stock_ticker</default-last-value-key>
</address-setting>
```

根据上述配置，与 lastValue 地址关联的所有队列都使用 stock_ticker 的最后值。默认情况下，不设置 default-last-value-key 的值。

3. 要为 *一组* 地址配置最后的值队列，您可以指定地址通配符。例如：

```
<address-setting match="lastValue.*">
  <default-last-value-key>stock_ticker</default-last-value-key>
</address-setting>
```

4. 或者，您可以将与地址或地址 *组* 关联的所有队列配置为使用默认值 _AMQ_LVQ_NAME。为此，请添加 default-last-value-queue 而不是 default-last-value-key。将值设为 true。例如：

```
<address-setting match="lastValue">
  <default-last-value-queue>true</default-last-value-queue>
</address-setting>
```

其他资源

- 有关配置地址时可以使用的通配符语法的更多信息，请参阅 第 4.2 节 “将地址设置应用到一组地址”。

4.11.3. 最后值队列行为示例

本例显示了最后的值队列的行为。

在 broker.xml 配置文件中，假设您添加了类似于以下内容的配置：

```
<address name="my.address">
   <multicast>
      <queue name="prices1" last-value-key="stock_ticker"/>
   </multicast>
</address>
```

上述配置会创建一个名为 prices1 的队列，其最后的值为 stock_ticker。

现在，假设客户端发送两个信息。每个信息对于属性 stock_ticker 有相同的 ATN 值。每个消息对于名为 stock_price 的属性都有不同的值。每个消息都发送到同一队列 prices1。

```
TextMessage message = session.createTextMessage("First message with last value property set");
message.setStringProperty("stock_ticker", "ATN");
message.setStringProperty("stock_price", "36.83");
producer.send(message);
```

```
TextMessage message = session.createTextMessage("Second message with last value property set");
message.setStringProperty("stock_ticker", "ATN");
message.setStringProperty("stock_price", "37.02");
producer.send(message);
```

当有两个对于 stock_ticker 最后值键（这里是 ATN）具有相同值的信息到达 prices1 queue 时，最有最后的信息会保留在队列中，第一个信息会被删除。在命令行中输入以下行来验证此行为：

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

在本例中，您看到的输出是第二个消息，因为两个消息都使用最后一个值键的相同值，并且第一个后在队列中收到第二个消息。

4.11.4. 为最后一个值队列强制使用非破坏性消耗

当消费者连接到队列时，发送到该消费者的正常行为会被消费者单独获取。当消费者确认接收消息时，代理会从队列中删除消息。

作为常规消耗的替代选择，您可以将队列配置为强制实施 *非破坏性* 消耗。在这种情况下，当队列向消费者发送消息时，其他消费者仍然可以接收该消息。此外，即使消费者消耗了它，消息也会保留在队列中。当您强制这种非破坏性消耗行为时，用户被称为队列 *浏览器*。

强制非破坏性消耗是最后一个值队列的有用配置，因为它可确保队列始终包含特定最后的值键的最新值。

以下流程演示了如何对最后一个值队列强制使用非破坏性。

先决条件

- 您已单独配置了最后一个值队列，或针对与地址 *或一组* 地址关联的所有队列。如需更多信息，请参阅：
  - 第 4.11.1 节 “单独配置最后一个值队列”
  - 第 4.11.2 节 “为地址配置最后一个值队列”

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。
2. 如果您之前将队列单独配置为最后一个值队列，请添加 非破坏性 键。将值设为 true。例如：

```
<address name="my.address">
  <multicast>
    <queue name="orders1" last-value-key="stock_ticker" non-destructive="true" />
  </multicast>
</address>
```

3. 如果您之前为最后一个值队列配置了地址 *或一组* 地址，请添加 **default-non-destructive** 键。将值设为 **true**。例如：

```
<address-setting match="lastValue">
  <default-last-value-key>stock_ticker </default-last-value-key>
  <default-non-destructive>true</default-non-destructive>
</address-setting>
```

注意

默认情况下，**default-non-destructive** 的值为 **false**。

4.12. 将过期的消息移到过期地址

对于除最后一个值队列以外的队列，如果您只有非破坏性消费者，代理永远不会从队列中删除消息，从而导致队列大小随时间增加。要防止这种不受限的队列大小增长，您可以在消息过期时配置，并指定代理将过期消息的地址。

4.12.1. 配置消息到期

以下流程演示了如何配置消息到期。

流程

1. 打开 **<broker_instance_dir> /etc/broker.xml** 配置文件。

2. 在 **core** 元素中，设置 **message-expiry-scan-period** 以指定代理扫描过期消息的频率。

```
<configuration ...>
  <core ...>
    ...
    <message-expiry-scan-period>1000</message-expiry-scan-period>
    ...
```

根据前面的配置，代理会每 1000 毫秒扫描已过期消息的队列。

3. 在匹配地址 *或一组* 地址的 address-setting 元素中，指定到期地址。另外，设置消息过期时间。例如：

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        <expiry-delay>10</expiry-delay>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

expiry-address

匹配地址或地址的到期地址。在前面的示例中，代理会将 库存 地址的过期消息发送到名为 ExpiryAddress 的到期地址。

expiry-delay

代理应用到 使用默认 过期时间的消息，以毫秒为单位。默认情况下，消息的过期时间为 0， 这意味着它们不会过期。对于过期时间大于默认值的消息，expiry-delay 无效。

例如，假设您在地址上将 expiry-delay 设置为 10，如上例中所示。如果默认过期时间为 0 的消息到达这个地址，则代理会将消息的过期时间从 0 改为 10。但是，如果另一个使用过期时间为 20 的消息到达，则其过期时间不会改变。如果将 expiry-delay 设为 -1， 则此功能将被禁用。默认情况下，expiry-delay 设置为 -1。

4. 另外，您可以指定最小和最大到期延迟值，而不是为 expiry-delay 指定值。例如：

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        <min-expiry-delay>10</min-expiry-delay>
```

```
            <max-expiry-delay>100</max-expiry-delay>
            ...
          </address-setting>
          ...
        <address-settings>
    <configuration ...>
```

min-expiry-delay

代理适用于消息的最短过期时间（以毫秒为单位）。

max-expiry-delay

代理应用到消息的最大过期时间（以毫秒为单位）。

代理应用 **min-expiry-delay** 和 **max-expiry-delay** 的值，如下所示：

- 对于默认过期时间为 **0** 的消息，代理会将过期时间设置为指定的值 **max-expiry-delay**。如果您还没有为 **max-expiry-delay** 指定一个值，代理会将过期时间设置为指定的值 **min-expiry-delay**。如果您还没有为 **min-expiry-delay** 指定一个值，代理不会更改消息的过期时间。
- 对于 **max-expiry-delay** 值高于过期时间的消息，代理会将过期时间设置为指定的值 **max-expiry-delay**。
- 对于值为 **min-expiry-delay** 的过期时间的消息，代理会将过期时间设置为指定的值 **min-expiry-delay**。
- 对于在 **min-expiry-delay** 和 **max-expiry-delay** 值之间带有过期的消息，代理不会更改消息的过期时间。
- 如果您为 **expiry-delay** 指定一个值（即与默认值 **-1** 不同的值），这将覆盖您为 **min-expiry-delay** 和 **max-expiry-delay** 指定的值。
- **min-expiry-delay** 和 **max-expiry-delay** 的默认值为 **-1**（即 **disabled**）。

5. 在配置文件的 **addresses** 元素中，配置之前为 **expiry-address** 指定的地址。在此地址上定义队列。例如：

```
<addresses>
   ...
   <address name="ExpiryAddress">
      <anycast>
         <queue name="ExpiryQueue"/>
      </anycast>
   </address>
   ...
</addresses>
```

前面的示例配置将到期队列 **ExpiryQueue** 与到期地址 **ExpiryAddress** 相关联。

4.12.2. 自动创建到期资源

常见用例是根据其原始地址隔离过期的消息。例如，您可以选择将过期的消息从名为来自名为 **stocks** 的地址路由到一个名为 **EXP.stocks** 的过期队列。同样，您可以将过期的消息从名为 **order** 的地址路由到名为 **EXP.orders** 的过期队列。

这种类型的路由模式可让您轻松跟踪、检查和管理过期的消息。但是，很难在主要自动创建的地址和队列的环境中实施这种模式。在这种环境中，管理员不希望额外努力手动创建地址和队列来保存过期的消息。

作为解决方案，您可以将代理配置为自动创建资源（即地址和队列）以处理给定地址 *或一组* 地址的过期消息。以下流程显示了一个示例。

先决条件

- 您已为给定地址 *或一组* 地址配置了到期地址。更多信息请参阅 第 4.12.1 节 “配置消息到期”。

流程

1. 打开 **& *lt;broker_instance_dir*> /etc/broker.xml** 配置文件。

2. 找到您之前添加到配置文件的 **& lt;address-setting > 元素，以定义匹配地址 *或一组* 地址的到期地址。例如：**

```
<configuration ...>

  <core ...>
    ...
```

```
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

3. 在 & lt;address-setting > 元素中，添加指示代理自动创建到期资源（即地址和队列）的配置项以及如何命名这些资源。例如：

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        <auto-create-expiry-resources>true</auto-create-expiry-resources>
        <expiry-queue-prefix>EXP.</expiry-queue-prefix>
        <expiry-queue-suffix></expiry-queue-suffix>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

auto-create-expiry-resources

指定代理是否自动创建到期地址和队列来接收过期的消息。默认值为 **false**。

如果参数值设为 **true**，代理会自动创建一个 **<address > 元素，用于定义到期地址和相关到期队列。自动创建的 <address& gt; 元素的 name 值与为 指定的值匹配 <expiry-address>。**

自动创建的过期队列有 **multicast 路由类型。默认情况下，代理将到期队列命名为 expiry 队列，以匹配最初发送过期消息的地址，如 库存。**

代理还为使用 **_AMQ_ORIG_ADDRESS 属性的到期队列定义过滤器。此过滤器可确保到期队列仅接收发送到对应原始地址的消息。**

expiry-queue-prefix

代理应用到自动创建过期队列的名称的前缀。默认值为 EXP。

当您定义前缀值或保留默认值时，到期队列的名称是前缀和原始地址的串联，如 EXP.stocks。

expiry-queue-suffix

代理应用到自动创建的过期队列的后缀。默认值没有被定义（即，代理不会应用后缀）。

您可以使用自己的队列（例如，使用 AMQ Broker 核心协议 JMS 客户端）或使用完全限定的队列名称（例如，使用其他 JMS 客户端）直接访问到期队列。

注意

由于会自动创建到期地址和队列，因此与删除自动创建的地址和队列相关的任何地址设置也适用于这些到期资源。

其他资源

- 有关配置自动创建的地址设置和队列的详情，请参考 第 4.8.2 节 “配置自动创建和删除地址和队列”。

4.13. 将未发送的消息移到死信地址

如果向客户端发送消息失败，您可能不希望代理持续尝试发送消息。为防止无限的尝试，您可以定义一个*死信地址*，以及一个或多个相关的*死信队列*。在进行指定数量的发送尝试后，代理会从其原始队列中删除未发送的消息，并将消息发送到配置的死信地址。系统管理员稍后可能会消耗从死信队列中未发送的消息来检查消息。

如果您没有为给定队列配置死信地址，代理会在指定次数发送尝试后从队列永久删除未发送的消息。

从死信队列使用未发送的消息具有以下属性：

_AMQ_ORIG_ADDRESS

指定消息的原始地址的字符串属性

_AMQ_ORIG_QUEUE

指定消息的原始队列的字符串属性

4.13.1. 配置死信地址

以下步骤演示了如何配置死信地址和关联的死信队列。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在与您的队列名匹配的 < address-setting > 元素中，为死信地址名称和发送尝试的最大数量设置值。例如：

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="exampleQueue">
        <dead-letter-address>DLA</dead-letter-address>
        <max-delivery-attempts>3</max-delivery-attempts>
      </address-setting>
    ...
    <address-settings>
<configuration ...>
```

match

代理在此 address-setting 部分中应用配置的地址。您可以为 < address-setting> 元素的 match 属性指定一个通配符表达式。如果您希望将 <address-setting> 项中配置的死信设置与一*组*地址相匹配，可以使用通配符表达式。

dead-letter-address

死信地址的名称。在本例中，代理将未发送的消息从队列 *exampleQueue* 移到死信地址 *DLA*。

max-delivery-attempts

在将未发送的消息移到配置的死信地址前，代理发出的最大发送尝试次数。在本例中，代理会在 3 次失败发送尝试后将未发送的消息移到死信地址。默认值为 10。如果您希望代理进行无限的重新发送尝试数，请指定 -1 的值。

3. 在 addresses 部分中，为死信地址 *DLA* 添加一个 address 元素。要将死信队列与死信地址关联，请为 队列 指定 name 值。例如：

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="DLA">
        <anycast>
          <queue name="DLQ" />
        </anycast>
      </address>
    ...
    </addresses>
  </core>
</configuration>
```

在前面的配置中，您将名为 *DLQ* 的死信队列与死信地址 *DLA* 关联。

其他资源

- 有关在地址设置中使用通配符的更多信息，请参阅 第 4.2 节 “将地址设置应用到一组地址”。

4.13.2. 自动创建死信队列

常见用例是根据其原始地址隔离未发送的消息。例如，您可以选择将来自 stocks 的未被成功发送的消息路由到一个名为 DLA.stocks 的队列，它有一个相关的名为 DLQ.stocks 的死信队列。同样，您可以将未发送的消息从名为 orders 的地址路由到一个名为 DLA.orders的死信地址。

这种类型的路由模式可让您轻松跟踪、检查和管理未发送的消息。但是，很难在主要自动创建的地址和队列的环境中实施这种模式。系统管理员可能不希望手动创建地址和队列来保存未发送的消息所需的额外工作量。

作为解决方案，您可以将代理配置为自动创建地址和队列来处理未发送的消息，如以下步骤所示。

先决条件

- 您已为队列或一组队列配置了死信地址。更多信息请参阅 第 4.13.1 节 “配置死信地址”。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 找到您之前添加的 &lt ;address-setting > 元素，为匹配的队列或一组队列定义死信地址。例如：

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="exampleQueue">
        <dead-letter-address>DLA</dead-letter-address>
        <max-delivery-attempts>3</max-delivery-attempts>
      </address-setting>
    ...
    <address-settings>
<configuration ...>
```

3. 在 <address-setting > 元素中，添加指示代理自动创建死信资源（即地址和队列）的配置项以及如何命名这些资源。例如：

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="exampleQueue">
        <dead-letter-address>DLA</dead-letter-address>
        <max-delivery-attempts>3</max-delivery-attempts>
        <auto-create-dead-letter-resources>true</auto-create-dead-letter-resources>
        <dead-letter-queue-prefix>DLQ.</dead-letter-queue-prefix>
        <dead-letter-queue-suffix></dead-letter-queue-suffix>
      </address-setting>
    ...
    <address-settings>
<configuration ...>
```

auto-create-dead-letter-resources

指定代理是否自动创建死信地址和队列来接收未发送的消息。默认值为 false。

如果将 auto-create-dead-letter-resources 设置为 true，代理会自动创建定义死信 地址和 关联的死信队列的 <address> 元素。自动创建的 < address> 元素的名称与 您为 < dead-letter-address> 指定的 name 值匹配。

代理在自动创建的 <address> 元素 中定义的 死信队列具有 multicast 路由类型。默认

情况下，代理将死信队列命名为死信队列，以匹配未 传输消息的原始地址，如库存。

代理还定义了使用 _AMQ_ORIG_ADDRESS 属性的死信队列的过滤器。此过滤器可确保死信队列仅接收发送到对应原始地址的消息。

dead-letter-queue-prefix

代理应用到自动创建的死信队列的名称的前缀。默认值为 DLQ。

当您定义前缀值或保留默认值时，死信队列的名称是前缀和原始地址的串联，如 DLQ.stocks。

dead-letter-queue-suffix

代理应用于自动创建的死信队列的后缀。默认值没有被定义（即，代理不会应用后缀）。

4.14. 在过期或未提供 AMQP 消息上注解和属性

在将过期或未提供 AMQP 消息移动到您配置的到期或死信队列之前，代理会将注解和属性应用到消息。客户端可以根据这些属性或注解创建过滤器，从到期或死信队列中选择要使用的特定消息。

注意

代理应用的属性是 *内部* 属性。这些属性不公开给客户端供常规使用，但 可以被 过滤器中的客户端指定。

下表显示了代理应用到过期或未提供 AMQP 消息的 annotations 和 internal 属性。

注解名称	内部属性名称	描述
x-opt-ORIG-MESSAGE-ID	_AMQ_ORIG_MESSAGE_ID	原始消息 ID，在消息被移到到期或死信队列之前。
x-opt-ACTUAL-EXPIRY	_AMQ_ACTUAL_EXPIRY	消息到期时间，指定自上次 epoch 启动后的毫秒数。
x-opt-ORIG-QUEUE	_AMQ_ORIG_QUEUE	过期或未发送邮件的原始队列名称。

注解名称	内部属性名称	描述
x-opt-ORIG-ADDRESS	_AMQ_ORIG_ADDRESS	过期或未发送邮件的原始地址名称。

其他资源

- 有关配置 AMQP 客户端以根据注解过滤 AMQP 消息的示例，请参考 第 13.3 节 “根据注释上的属性过滤 AMQP 消息”。

4.15. 禁用队列

如果在代理配置中手动定义队列，则队列会被默认启用。

但是，在有些情况下，您要定义队列，以便客户端可以订阅它，但不准备好将队列用于消息路由。或者，在有些情况下，您要停止消息流到队列，但仍然保持客户端绑定到队列。在这些情况下，您可以禁用队列。

以下示例演示了如何禁用您在代理配置中定义的队列。

先决条件

- 您应该熟悉如何在代理配置中定义地址和相关队列。更多信息请参阅 第 4 章 *配置地址和队列*。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。
2. 对于您之前定义的队列，添加 enabled 属性。要禁用队列，请将此属性的值设置为 false。例如：

```
<addresses>
  <address name="orders">
    <multicast>
      <queue name="orders" enabled="false"/>
    </multicast>
  </address>
</addresses>
```

enabled 属性的默认值为 true。当您将值设为 false 时，到队列的消息路由将被禁用。

注意

如果您禁用地址 上的所有 队列，则发送到该地址的任何消息都会静默丢弃。

4.16. 限制连接到队列的消费者数量

使用 max-consumers 属性限制连接到特定队列的消费者数量。通过将 max-consumers 标志设置为 1 来创建专用消费者。默认值为 -1， 它设置无限数量的消费者。

以下流程演示了如何对可连接到队列的用户数量设置限制。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 对于给定队列，添加 max-consumers 键并设置值。

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="my.address">
        <anycast>
          <queue name="q3" max-consumers="20"/>
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

根据上述配置，只有 20 个消费者可以同时连接到队列 q3。

3. 要创建专用消费者，请将 max-consumers 设置为 1。

```
<configuration ...>
  <core ...>
    ...
```

```
    <address name="my.address">
     <anycast>
      <queue name="q3" max-consumers="1"/>
     </anycast>
    </address>
   </core>
</configuration>
```

4. 若要允许无限数量的消费者，请将 **max-consumers** 设置为 **-1**。

```
<configuration ...>
 <core ...>
  ...
  <address name="my.address">
   <anycast>
    <queue name="q3" max-consumers="-1"/>
   </anycast>
  </address>
 </core>
</configuration>
```

4.17. 配置专用队列

专用队列是特殊的队列，可将所有消息一次仅路由到一个消费者。当您希望所有消息被同一消费者按顺序处理时，此配置很有用。如果一个队列有多个消费者，则只有一个消费者接收消息。如果该消费者与队列断开连接，则会选择另一个消费者。

4.17.1. 单独配置专用队列

以下步骤演示了如何将给定队列单独配置为 **exclusive**。

流程

1. 打开 **<broker_instance_dir> /etc/broker.xml** 配置文件。

2. 对于给定队列，添加 **exclusive** 密钥。将值设为 **true**。

```
<configuration ...>
 <core ...>
  ...
  <address name="my.address">
   <multicast>
    <queue name="orders1" exclusive="true"/>
   </multicast>
```

```
    </address>
  </core>
</configuration>
```

4.17.2. 为地址配置专用队列

以下流程演示了如何配置地址 *或一组* 地址，以便所有关联的队列都是独占的。

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 address-setting 元素中，用于匹配地址，添加 default-exclusive-queue 键。将值设为 true。

```
<address-setting match="myAddress">
  <default-exclusive-queue>true</default-exclusive-queue>
</address-setting>
```

根据前面的配置，与 myAddress 地址关联的所有队列都是专用的。默认情况下，default-exclusive-queue 的值为 false。

3. 要为 一组 地址配置专用队列，您可以指定地址通配符。例如：

```
<address-setting match="myAddress.*">
  <default-exclusive-queue>true</default-exclusive-queue>
</address-setting>
```

其他资源

- 有关配置地址时可以使用的通配符语法的更多信息，请参阅 第 4.2 节 “将地址设置应用到一组地址”。

4.18. 将特定地址设置应用到临时队列

例如，在使用 JMS 时，代理通过将通用唯一标识符(UUID)分配为地址名称和队列名称来创建 *临时队列*。

默认的 <address-setting match="#"> 会将配置的地址设置应用到*所有*队列，包括临时队列。如果只想将特定的地址设置应用到临时队列，您可以选择指定一个 ephemeral -queue-namespace，如下所

述。然后，您可以指定与命名空间匹配的地址设置，代理会将这些设置应用到所有临时队列。

当临时队列被创建并存在一个临时队列命名空间时，代理会将 temporary-queue-namespace 值和配置的分隔符（默认 .）添加到地址名称。它使用它来引用匹配的地址设置。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加 temporary-queue-namespace 值。例如：

```
<temporary-queue-namespace>temp-example</temporary-queue-namespace>
```

3. 添加一个 address-setting 元素，其 match 值与临时队列命名空间对应。例如：

```
<address-settings>
  <address-setting match="temp-example.#">
    <enable-metrics>false</enable-metrics>
  </address-setting>
</address-settings>
```

这个示例禁用代理创建的所有临时队列中的指标。

注意

指定临时队列命名空间不会影响临时队列。例如，命名空间不会更改临时队列的名称。命名空间用于引用临时队列。

其他资源

- 有关在地址设置中使用通配符的更多信息，请参阅 第 4.2 节 “将地址设置应用到一组地址”。

4.19. 配置环队列

通常，AMQ Broker 中的队列使用先出(FIFO)语义。这意味着代理向队列的尾部添加信息，并将它们从头中删除。环队列是一种特殊类型的队列，包含指定的、固定数量的消息。当新消息到达但队列已包含指定数量的消息时，代理通过删除队列头来维护固定队列大小。

例如，假设环队列配置大小为 3，生成者按顺序发送消息 A、B、C 和 D。消息 C 到达队列后，队列中的消息数已达到配置的环大小。此时，消息 A 位于队列的头头，而消息 C 处于尾部。当消息 D 到达队列时，代理会将消息添加到队列的尾部。为了维护固定队列大小，代理会删除队列头的消息（即消息 A）。消息 B 现在位于队列的头。

4.19.1. 配置环队列

以下步骤演示了如何配置环队列。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

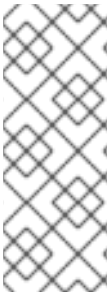
2. 要在没有设置显式环大小的匹配地址上为所有队列定义默认环大小，请在 address-setting 元素中为 default-ring-size 指定一个值。例如：

```
<address-settings>
  <address-setting match="ring.#">
    <default-ring-size>3</default-ring-size>
  </address-setting>
</address-settings>
```

default-ring-size 参数对于定义自动创建队列的默认大小特别有用。default-ring-size 的默认值为 -1 （即无大小限制）。

3. 要在特定队列上定义环大小，请将 ring-size 键添加到 queue 元素。指定一个值。例如：

```
<addresses>
  <address name="myRing">
    <anycast>
      <queue name="myRing" ring-size="5" />
    </anycast>
  </address>
</addresses>
```



注意

您可以在代理运行时更新 ring-size 的值。代理会动态应用更新。如果新的 ring-size 值小于前面的值，代理不会立即从队列头中删除消息，以强制实施新大小。发送到队列的新消息仍然强制删除旧的消息，但队列不会达到其新的大小，直到客户端通过正常方式使用消息。

4.19.2. 环队列故障排除

本节介绍环队列的行为与其配置不同的情况。

In-delivery 消息和回滚

当消息发送到消费者时，消息处于"in-between"状态，其中消息在技术上不再在队列中，但尚未确认。消息会一直处于 in-delivery 状态，直到被消费者确认。处于 in-delivery 状态的消息无法从环队列中删除。

由于代理无法删除 in-delivery 消息，因此客户端可以将比环队列所允许的更多消息发送到环队列。例如，请考虑这种情况：

1. **制作者将三个消息发送到配置了 ring-size="3" 的环队列。**

2. **所有消息都会立即分配给消费者。**

   此时，messageCount= 3 和 deliver Count= 3。

3. **生产者向队列发送另一个消息。然后，消息被分配给消费者。**

   现在，messageCount = 4 和 deliverCount = 4。消息计数 4 大于配置的环大小 3。但是，代理需要允许这种情况，因为它无法从队列中删除发送的消息。

4. **现在，假设消费者已关闭，而不会确认任何消息。**

   在这种情况下，四个 in-delivery，未确认的消息会取消到代理，并以相反的顺序添加到队列的头。此操作将队列置于其配置的环大小上。由于环队列更喜欢在头上队列尾部的消息，因此队列丢弃了生产者发送的第一个消息，因为这是最后一个消息添加到队列的头头。事务或核心会话回滚的方式相同。

如果您直接使用核心客户端，或使用 AMQ Core Protocol JMS 客户端，您可以通过减少 consumerWindowSize 参数的值（默认为 1024 * 1024 字节）来最小化传输的消息数量。

调度的消息

当调度的消息发送到队列时，该消息不会立即添加到队列的尾部，如正常消息。相反，代理会将调度的消息保存在中间缓冲区中，并根据消息的详情调度消息以发送到队列的头。但是，调度的消息仍然反映在队列的消息计数中。与内发送的消息一样，此行为可能会显示代理没有强制执行环队列大小。例如，请考虑这种情况：

1. 在 12:00 时，生成者发送消息 A 给使用 ring-size="3" 配置的环队列。该消息被调度为 12:05。

   此时，messageCount= 1 和 scheduledCount= 1。

2. at 12:01，生成者将消息 B 发送到同一环队列。

   现在，messageCount= 2 和 scheduledCount= 1。

3. at 12:02，生成者将消息 C 发送到同一环队列。

   现在，messageCount= 3 和 scheduledCount= 1。

4. at 12:03，生成者将消息 D 发送到同一环队列。

   现在，messageCount= 4 和 scheduledCount= 1。

   队列的消息数现在是 4，大于配置的环大小为 3。但是，调度的消息还没有在队列中（即，它位于代理上，并被调度到队列中）。在计划的 12:05 交付时，代理会将消息放在队列的头头上。但是，由于环队列已达到其配置的大小，因此调度的消息 A 会立即被删除。

页面消息

与发送中调度的消息和消息类似，页面的消息不会计入代理所强制执行的环队列大小，因为消息实际上被记录在地址级别，而不是队列级别。一个 paged 消息在队列中没有技术，但它反映在队列的 messageCount 值中。

建议您不要将分页用于环队列的地址。相反，请确保整个地址可以容纳在内存中。或者，将 address-full-policy 参数配置为 DROP, BLOCK 或 FAIL。

其他资源

- 当您配置被动地址时，代理会创建环队列的内部实例。如需了解更多相关信息，请参阅 第 4.20 节 “配置 retroactive 地址”。

4.20. 配置 RETROACTIVE 地址

将地址配置为 *retroactive* 可让您保留发送到该地址的消息，包括当还没有绑定到地址的队列时。当队列稍后创建并绑定到地址时，代理会重新主动将消息分发到这些队列。如果地址 *没有配置为* retroactive，且还没有绑定到它，代理会丢弃发送到该地址的消息。

当您配置被动地址时，代理会创建一个类型的队列的内部实例，称为 *环队列*。环队列是一种特殊类型的队列，包含指定的、固定数量的消息。队列达到指定大小后，到达队列的下一个消息会强制从队列中最旧的消息。当您配置 retroactive 地址时，您间接指定内部环队列的大小。默认情况下，内部队列使用 multicast 路由类型。

retroactive 地址使用的内部环队列通过管理 API 公开。您可以检查指标并执行其他常见管理操作，如清空队列。环队列还贡献地址的总内存用量，这会影响消息分页等行为。

以下流程演示了如何将地址配置为 retroactive。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 为 address-setting 元素中的 retroactive-message-count 参数指定一个值。您指定的值定义了您要保留的消息数量。例如：

```
<configuration>
 <core>
  ...
  <address-settings>
    <address-setting match="orders">
      <retroactive-message-count>100</retroactive-message-count>
    </address-setting>
  </address-settings>
```

```
...
  </core>
</configuration>
```

注意

您可以在代理运行时更新 retroactive-message-count 的值，可以是 broker.xml 配置文件或管理 API。但是，如果您 *减少* 这个参数的值，则需要一个额外的步骤，因为 retroactive 地址是通过环队列实现的。减少了 ring-size 参数的环队列不会自动从队列中删除消息，以实现新的 ring-size 值。这个行为可防止意外消息丢失。在这种情况下，您需要使用管理 API 来手动减少环队列中的消息数量。

其他资源

- **有关环队列的更多信息，请参阅 第 4.19 节 “配置环队列”。**

4.21. 为内部管理的地址和队列禁用公告消息

默认情况下，AMQ Broker 会在 OpenWire 客户端连接到代理时创建关于地址和队列的公告消息。公告消息发送到代理创建的内部管理地址。这些地址会出现在与用户部署的地址和队列相同的显示中。虽然它们提供了有用的信息，但当代理管理大量目的地时，公告消息可能会导致不必要的后果。例如，消息可能会增加内存用量或入站连接资源。另外，当尝试显示为发送公告消息创建的所有地址时，AMQ Management Console 可能会变得混乱。要避免这些情况，您可以使用以下参数来配置代理上公告消息的行为。

supportAdvisory

将这个选项设置为 true 以启用创建公告信息或 false 来禁用它们。默认值为 true。

suppressInternalManagementObjects

将这个选项设置为 true，将公告信息公开给管理服务，如 JMX registry 和 AMQ Management Console，或 false 来公开它们。默认值为 true。

以下流程演示了如何在代理中禁用公告信息。

流程

1. **打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。**

2.

对于 OpenWire 连接器，将 supportAdvisory 和 suppressInternalManagementObjects 参数添加到配置的 URL 中。按照本节前面所述设置值。例如：

```
<acceptor name="artemis">tcp://127.0.0.1:61616?
protocols=CORE,AMQP,OPENWIRE;supportAdvisory=false;suppressInternalManagem
entObjects=false</acceptor>
```

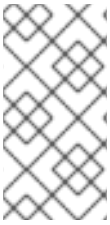
4.22. 联邦地址和队列

联邦支持在代理间传输信息，而无需代理位于一个通用集群中。代理可以是独立的，也可以在单独的集群中。另外，源和目标代理可以位于不同的管理域中，这意味着代理可能具有不同的配置、用户和安全设置。代理甚至可能使用不同版本的 AMQ Broker。

例如，联邦适合可靠地将信息从一个集群发送到另一个集群。这种传输可能会跨越广域网(WAN)、云基础架构的*区域*或互联网。如果从源代理连接到目标代理丢失（例如，由于网络失败），则源代理会尝试重新建立连接，直到目标代理重新上线为止。当目标代理重新上线时，消息传输会恢复。

管理员可以使用地址和队列策略来管理联邦。策略配置可以匹配到特定的地址或队列，或者策略可以包含与一*组*地址或队列匹配的通配符表达式。因此，联邦可以动态应用，作为队列或地址被添加到匹配集合中，或从匹配的集合中删除。策略 可以包含多个 表达式，包括和/或排除特定地址和队列。另外，多个策略也可以应用到代理或代理集群。

在 AMQ Broker 中，两个主要的联邦选项是 ***address federation*** 和 ***queue federation***。这些选项在后面的部分中描述。



注意

代理可以包含联邦 和 仅限本地组件的配置。也就是说，如果您在代理上配置联邦，则不需要联合该代理上的所有内容。

4.22.1. 关于地址联邦

地址联邦类似于连接的代理之间的完整多播分布模式。例如，发送到 BrokerA 上地址的每个消息都会传送到该代理上的每个队列。另外，每个消息都传送到 BrokerB，以及所有附加的队列。

地址联邦动态将代理链接到远程代理中的地址。例如，如果本地代理希望从远程代理上的地址获取信息，则会在远程地址上自动创建队列。然后，远程代理上的消息会消耗到这个队列。最后，消息会复制到本地代理上的对应地址，就像它们最初直接发布到本地地址一样。

远程代理不需要重新配置，以允许联邦在其上创建地址。但是，*本地* 代理需要向远程地址授予权限。

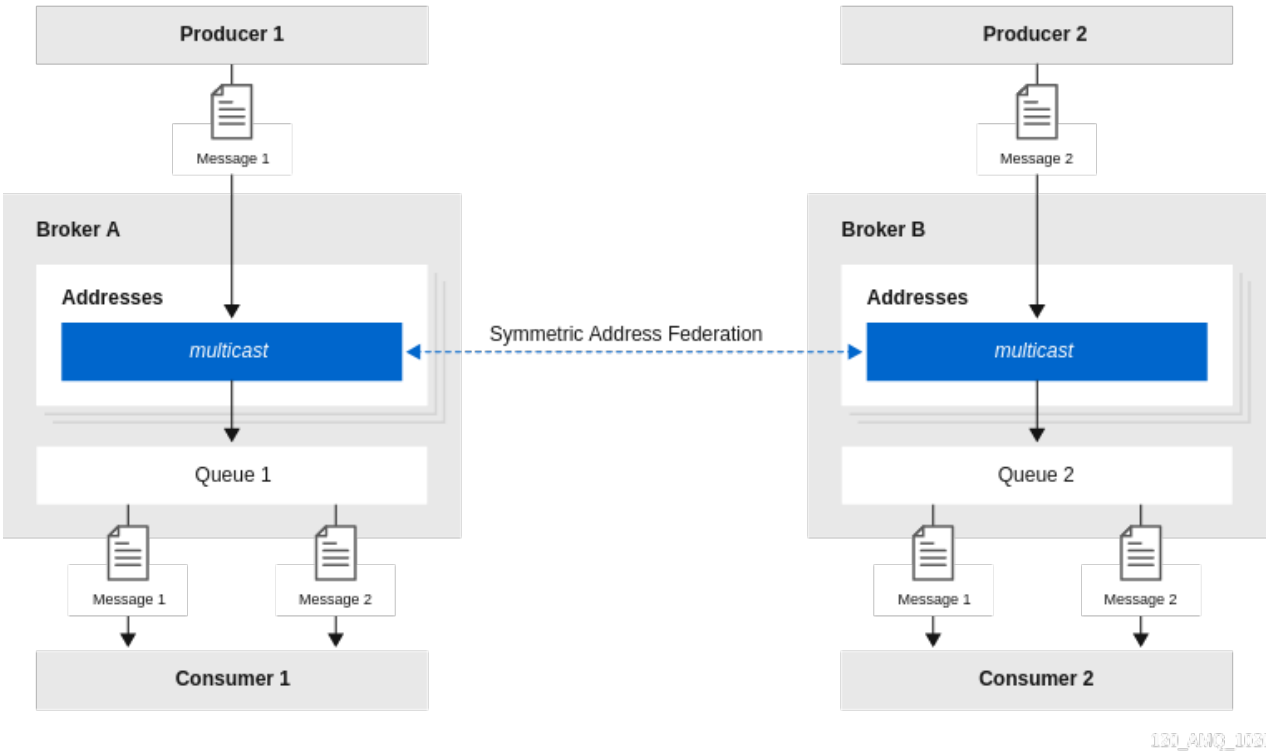
4.22.2. 用于地址联邦的通用拓扑

下面介绍了一些使用地址联邦的拓扑。

对称拓扑

在对称拓扑中，生成者和消费者连接到每个代理。队列及其使用者可以接收由任一制作者发布的消息。对称拓扑示例如下所示：

图 4.1. 对称拓扑中的地址联邦

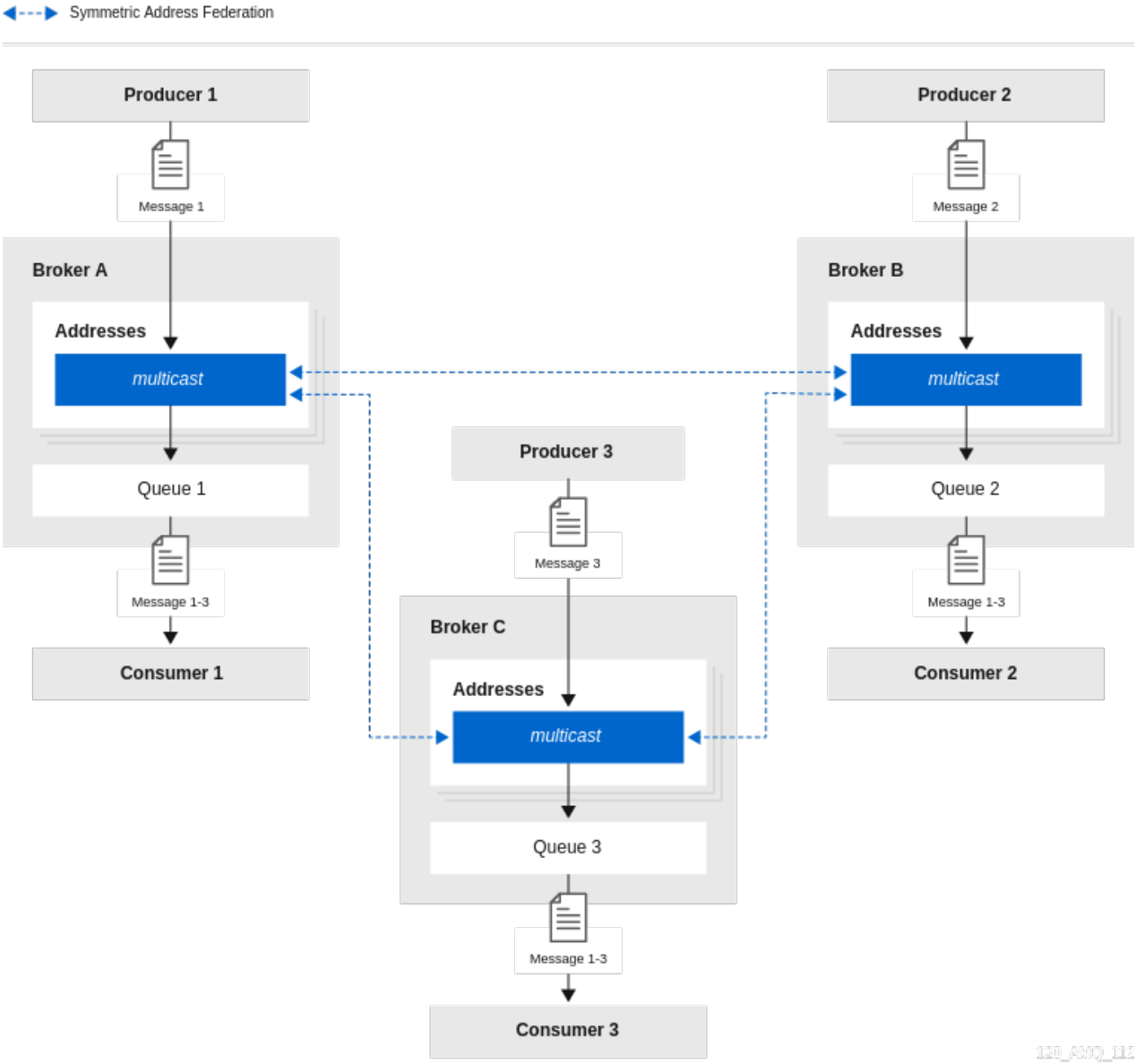


为对称拓扑配置地址联邦时，务必要将 address 策略的 max-hops 属性的值设置为 1。这样可确保消息仅复制一次，避免 cyclic 复制。如果此属性设置为较大的值，则消费者将接收同一消息的多个副本。

完整网格拓扑

完整网格拓扑与对称设置类似。三个或更多的代理会相互独立处理，从而创建一个完整的网格。在此设置中，生成者和消费者连接到每个代理。队列及其使用者可以接收任何制作者发布的消息。此拓扑的示例如下所示。

图 4.2. 在完整网格拓扑中处理联邦

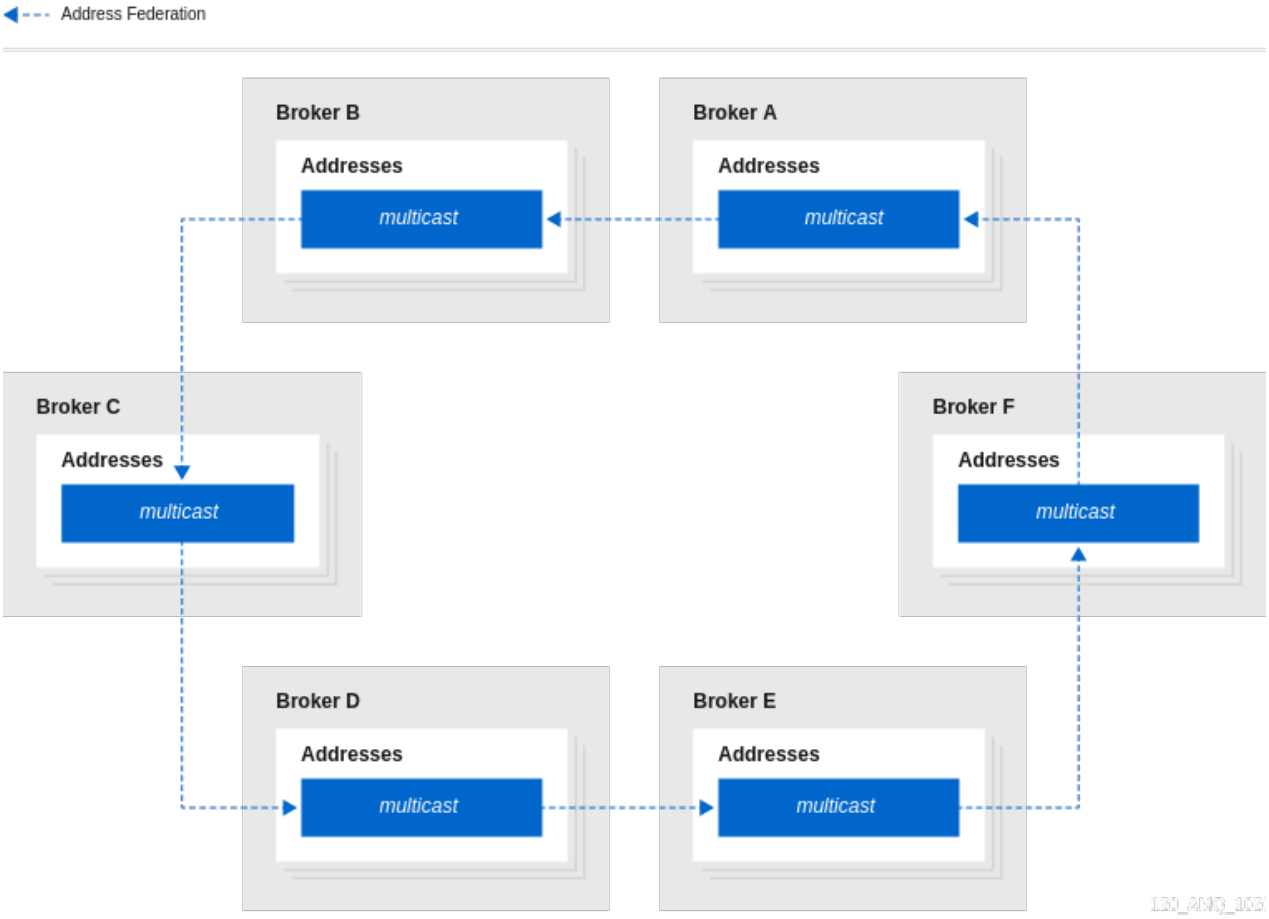


与对称设置一样，在为完整网格拓扑配置地址联邦时，务必要将地址策略的 max-hops 属性的值设置为 1。这样可确保消息仅复制一次，避免 cyclic 复制。

Ring topology

在代理环中，每个联邦地址都上游到环中另一个地址。此拓扑的示例如下所示。

图 4.3. 环拓扑中的地址联邦



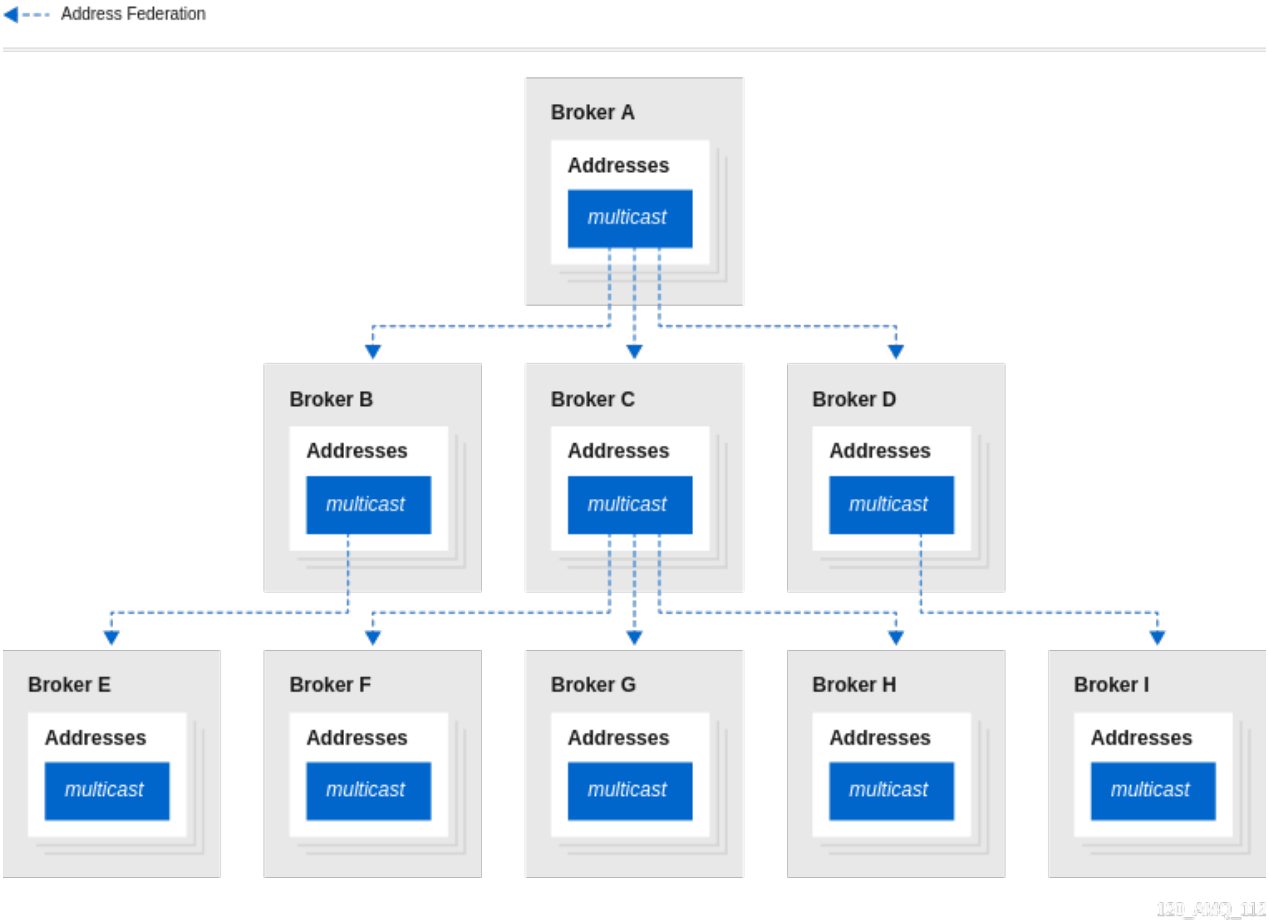
当您为环拓扑配置联邦时，为了避免 cyclic 复制，务必要将地址策略的 max-hops 属性设为 n-1，其中 *n* 是环中的节点数。例如，在上面的环拓扑中，max-hops 的值设为 5。这可确保环中的每个地址都 准确看到消息一次。

环路拓扑的一个优点是，根据您需要进行的物理连接数量，可以更便宜设置。但是，这种类型的拓扑的一个缺陷在于，如果单个代理失败，整个环会失败。

fan-out 拓扑

在 fan-out 拓扑中，单个 master 地址通过联邦地址树链接至。发布到 master 地址的任何消息都可由与树中任何代理连接的消费者接收。树可以被配置为任何深度。也可以扩展树，无需在树中重新配置现有代理。此拓扑的示例如下所示。

图 4.4. 在 fan-out 拓扑中地址联邦



当您为 fan-out 拓扑配置联邦时，请确保将地址策略的 max-hops 属性设为 n-1 的值，其中 *n* 是树中的级别数量。例如，在上面的 fan-out 拓扑中，max-hops 的值设置为 2。这样可确保树中的每个地址都 准确看到消息一次。

4.22.3. 支持在地址联邦配置中使用 divert 绑定

在配置地址联邦时，您可以在地址策略配置中添加对分离绑定的支持。添加此支持可让联邦响应分离绑定，以便在远程代理上为给定地址创建联邦消费者。

例如，假设地址策略中包含了一个名为 test.federation.source 的地址，并且不包含另一个名为 test.federation.target 的地址。通常，当一个队列在 test.federation.target 上创建，不会导致创建联邦消费者，因为地址不是地址策略的一部分。但是，如果您创建一个 divert 绑定，使得 test.federation.source 是源地址，test.federation.target 是转发地址，则会在转发地址处创建一个持久化消费者。源地址仍然必须使用 multicast 路由类型，但目标地址可以使用 multicast 或 anycast。

示例用例是一个 divert，它将一个 JMS 主题(多播地址)重定向到 JMS 队列（任播 地址）。这可实现对传统消费者的消息进行负载平衡，以便不支持 JMS 2.0 和共享订阅。

4.22.4. 配置联邦

您可以使用 Core 协议或从 7.12、AMQP 开始配置地址和队列联邦。使用 AMQP 进行联邦提供以下优点：

- 如果客户端使用 AMQP 进行消息传递，使用 AMQP 进行联邦无需在 AMQP 和核心协议之间进行转换，反之亦然，如果联邦使用 Core 协议，则需要它。
- AMQP 联邦支持通过单一传出连接进行双向联邦。这消除了远程代理连接到本地代理的需求，这是您使用 Core 协议进行联邦时的要求，这可能被网络策略阻止。

4.22.4.1. 使用 AMQP 配置联邦

您可以使用以下策略使用 AMQP 配置地址和队列联邦：

- 本地地址策略将本地代理配置为监视地址的需求，以及当该需求存在时，在远程代理上的匹配地址上创建一个联合消费者，以联合消息到本地代理。
- 远程地址策略将远程代理配置为监视地址的需求，并且当该需求存在时，在本地代理上的匹配地址上创建一个联合消费者，以便将消息发送到远程代理。
- 本地队列策略将本地代理配置为监视队列上的需求，并且当该需求存在时，在远程代理上的匹配队列上创建一个联合消费者，以联合消息到本地代理。
- 远程队列策略将远程代理配置为监视队列上的需求，并且当该需求存在时，在本地代理上的匹配队列上创建一个联合消费者，以联合消息到远程代理。

远程地址和队列策略发送到远程代理，并成为远程代理上的本地策略，以提供反向联邦连接。为反向联邦连接应用策略，接收策略的代理是本地代理，发送策略的代理是远程代理。通过在本地代理上配置远程地址和队列策略，您可以在单个代理上保留所有联邦配置，例如，对于 hub-and-spoke 拓扑，这可能是一个有用的方法。

4.22.4.1.1. 使用 AMQP 配置地址联邦

使用 & lt;broker-connections& gt; 元素来配置使用 AMQP 的地址联邦。

前提条件

< amqp-connection> 元素中指定的用户具有与远程代理上地址和队列匹配的读写权限。

流程

1. **打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。**

2. **添加一个 &lt ;broker connections> 元素，其中包含 &lt ;amqp-connection> 元素。在 & lt;amqp-connection > 元素中，指定远程代理的连接详情，并为联邦配置分配一个名称。例如：**

```
<broker-connections>
  <amqp-connection uri="tcp://<__HOST__>:<__PORT__>" user="federation_user"
password="federation_pwd" name="queue-federation-example">
  </amqp-connection>
</broker-connections>
```

3. **添加 & lt;federation > 元素，并包括以下一个或多个项：**

   - **一个 <local-address-policy > 元素，将消息从远程代理到本地代理。**
   - **一个 <remote-address-policy > 元素，将消息从本地代理到远程代理。**

   以下示例显示了一个带有本地和远程地址策略的 federation 元素。

```
<broker-connections>
  <amqp-connection uri="tcp://<__HOST__>:<__PORT__>" user="federation_user"
password="federation_pwd" name="queue-federation-example">
    <federation>
      <local-address-policy name="example-local-address-policy" auto-delete="true" auto-
delete-delay="1" auto-delete-message-count="2" max-hops="1" enable-divert-
bindings="true">
        <include address-match="queue.news.#" />
     <include address-match="queue.bbc.news" />
       <exclude address-match="queue.news.sport.#" />
      </local-address-policy>
      <remote-address-policy name="example-remote-address-policy">
        <include address-match="queue.usatoday" />
      </remote-address-policy>
```

```
    </federation>
  </amqp-connection>
</broker-connections>
```

相同的参数可在本地和远程地址策略中配置。有效参数有：

name

地址策略的名称。所有地址策略名称必须在 < broker-connections > 元素中的 < federation > 元素中唯一。

max-hops

在联邦过程中可以发出的最大跃点数。默认值 0 适用于大多数简单的联邦部署。但是，在某些拓扑中，可能需要一个更大的值来防止消息进行循环。

auto-delete

对于地址联邦，会在消息联邦的代理上创建 durable 队列。将此参数设置为 true，以在启动代理断开连接并满足延迟和消息计数参数后为自动删除标记队列。如果要自动清理动态创建的队列，则此选项是一个非常有用的选项。默认值为 false，这意味着队列不会被自动删除。

auto-delete-delay

在创建队列前的时间（以毫秒为单位），在启动代理断开连接后具有自动删除的时间（毫秒）。默认值为 0。

auto-delete-message-count

队列的消息计数必须小于或等于，然后才能自动删除队列。默认值为 0。

enable-divert-bindings

设置为 true 可允许根据需求侦听 divert 绑定。如果地址与地址策略的包含地址匹配，则任何与 divert 的转发地址匹配的队列绑定都会创建需求。默认值为 false。

Include

与策略中要包含的地址匹配地址的地址匹配模式。您可以指定多个模式。如果没有指定模式，则所有地址都包含在策略中。

您可以指定一个准确的地址，如 queue.bbc.news。或者，您可以使用数字符号(#)通配符字符来指定匹配的地址集合。在前面的示例中，本地地址策略还包括以 queue.news 字符串开头的所有地址。

exclude

匹配要从策略中排除的地址的 address-match 模式。您可以指定多个模式。如果没有指定模式，策略中没有排除地址。

您可以指定一个准确的地址，如 queue.bbc.news。或者，您可以使用数字符号(#)通配符字符来指定匹配的地址集合。在前面的示例中，本地地址策略排除以 queue.news.sport 字符串开头的所有地址。

4.22.4.1.2. 使用 AMQP 配置队列联邦

流程

使用 & lt;broker connections& gt; 元素为 AMQP 配置队列联邦。

前提条件

< amqp-connection> 元素中指定的用户具有与远程代理上地址和队列匹配的读写权限。

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 添加一个 &lt ;broker connections> 元素，其中包含 &lt ;amqp-connection> 元素。在 & lt;amqp-connection > 元素中，指定远程代理的连接详情，并为联邦配置分配一个名称。例如：

```
<broker-connections>
  <amqp-connection uri="tcp://<__HOST__>:<__PORT__>" user="federation_user"
password="federation_pwd" name="queue-federation-example">
  </amqp-connection>
</broker-connections>
```

3. 添加 & lt;federation > 元素，并包括以下一个或多个项：

   - 一个 <local-queue-policy > 元素，将消息从远程代理到本地代理。
   - 一个 <remote-queue-policy > 元素，将消息从本地代理到远程代理。

以下示例显示了包含本地队列策略的 federation 元素。

-

```
<broker-connections>
 <amqp-connection uri="tcp://HOST:PORT" name="federation-example">
  <federation>
   <local-queue-policy name="example-local-queue-policy">
    <include address-match="#" queue-match="#.remote" />
    <exclude address-match="#" queue-match="#.local" />
   </local-queue-policy>
  </federation>
 </amqp-connection>
</broker-connections>
```

name

队列策略的名称。所有队列策略名称必须在 < broker-connections > 元素的 <federation > 元素内唯一。

Include

匹配地址 和队列匹配模式的地址匹配，以匹配 这些地址上的特定队列，以包含在策略中。与 address-match 参数一样，您可以指定 queue-match 参数的确切名称，也可以使用通配符表达式来指定一组队列。在上例中，包含与所有地址中的 .remote 字符串匹配的队列（由 address-match 值表示）。

exclude

匹配地址 和队列匹配模式的地址匹配，以匹配 这些地址上的特定队列，以便从策略中排除。与 address-match 参数一样，您可以指定 queue-match 参数的确切名称，也可以使用通配符表达式来指定一组队列。在前面的示例中，排除在所有地址中与 .local 字符串匹配的队列，由 address-match 值表示。

priority-adjustment

调整联邦消费者的值，以确保它们的优先级值低于同一队列中其他本地用户的值。默认值为 -1， 这样可确保本地消费者优先于联邦消费者。

include-federated

当此参数的值设为 false 时，配置不会重新联邦一个已存在的消费者（即联邦队列上的消费者）。这可避免在对称或关闭的拓扑中，没有非联邦消费者和系统的消息流。

如果您没有 closed-loop 拓扑，则将此参数设置为 true。例如，假设您有一个包含三个代理(BrokerA、BrokerB 和 BrokerC)的链，以及 BrokerA 和消费者 at BrokerC 的制作者。在这种情况下，您希望 BrokerB 重新联邦消费者到 BrokerA。

4.22.4.2. 使用 Core 协议配置联邦

您可以配置消息和队列联邦以使用 Core 协议。

4.22.4.2.1. 为代理集群配置联邦

下面的部分的示例演示了如何配置 独立 本地和远程代理之间的地址和队列联邦。对于在独立代理间联邦、联邦配置的名称以及任何地址和队列策略的名称，在本地和远程代理之间必须是唯一的。

但是，如果您要在集群中为代理配置联邦，则需要额外的要求。对于集群代理，联邦配置的名称以及该配置中的任何地址和队列策略的名称 对于该集群中的每个代理都必须相同。

确保同一集群中的代理使用相同的联邦配置和地址和队列策略名称避免消息重复。例如，如果同一集群中的代理 有不同的联邦配置名称，这可能会导致为同一地址创建多个不同的 转发队列，从而导致下游用户的消息重复。相反，如果同一集群中的代理 使用相同的 联邦配置名称，这基本上会创建与下游用户进行负载均衡的复制的集群转发队列。这可避免消息重复。

4.22.4.2.2. 配置上游地址联邦

以下示例演示了如何在独立代理之间配置上游地址联邦。在本例中，您要将本地（即 *下游*）代理中的联邦配置为一些远程（即 *上游*）代理。

先决条件

- 以下示例演示了如何配置独立代理间的地址联邦。但是，您还应熟悉为代理 *集群配置* 联邦的要求。更多信息请参阅 第 4.22.4.2.1 节 “为代理集群配置联邦”。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 添加新的 < federations> 元素，其中包含 < federation> 元素。例如：

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
  </federation>
</federations>
```

name

联邦配置的名称。在本例中，名称对应于本地代理的名称。

user

用于连接到上游代理的共享用户名。

password

用于连接到上游代理的共享密码。

注意

如果用户和密码凭证与远程代理不同，您可以在将代理添加到配置中时单独指定这些代理的凭证。此流程稍后将对此进行描述。

3. 在 **federation** 元素中，添加一个 **< address-policy>** 元素。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">
      </address-policy>

   </federation>
</federations>
```

name

地址策略的名称。代理上配置的所有地址策略必须具有唯一的名称。

auto-delete

在地址联邦过程中，本地代理会在远程地址动态创建持久队列。**auto-delete** 属性的值指定在本地代理断开连接后队列是否应该被删除， **auto-delete-delay** 和 **auto-delete-message-count** 属性的值也已达到。如果要自动清理动态创建的队列，则此选项是一个非常有用的选项。如果您要防止在远程代理长时间断开连接时，在远程代理上构建消息，它也是一个有用的选项。但是，如果您希望消息在本地代理断开连接时始终保持排队，则可能会将此选项设置为 **false**，从而避免本地代理上的消息丢失。

auto-delete-delay

在本地代理断开连接后，此属性的值指定了动态创建的远程队列有资格自动删除的时间（以毫秒为单位）。

auto-delete-message-count

在本地代理断开连接后，此属性的值指定了在该队列被自动删除前仍然可以位于动态

创建的远程队列中的最大消息数。

enable-divert-bindings

将此属性设置为 true 可根据需要侦听分离绑定。如果存在与地址策略包含的地址匹配的解析绑定，则任何与 divert 的转发地址匹配的队列绑定都会创建需求。默认值为 false。

max-hops

在联邦过程中可以发出的最大跃点数。特定的拓扑需要此属性的特定值。要了解更多有关这些要求的信息，请参阅 第 4.22.2 节 “用于地址联邦的通用拓扑”。

transformer-ref

转换程序配置的名称。如果要在联邦消息传输过程中转换消息，您可以添加转换器配置。此流程稍后会描述转换器配置。

4. 在 &lt ;address-policy& gt; 元素中，添加 address-matching 模式，以便从地址策略中包含和排除地址策略。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
         <include address-match="queue.news.#" />

         <exclude address-match="queue.news.sport.#" />
      </address-policy>

   </federation>
</federations>
```

Include

此元素的 address-match 属性的值指定要包含在地址策略中的地址。您可以指定一个准确的地址，如 queue.bbc.new 或 queue.usatoday。或者，您可以使用通配符表达式来指定匹配的地址 *集合*。在前面的示例中，地址策略还包括以字符串 queue.news 开头的所有 地址名称。

exclude

此元素的 address-match 属性的值指定要从地址策略中排除的地址。您可以指定一个准确的地址名称，或使用通配符表达式来指定匹配的地址 *集合*。在前面的示例中，地址策略

排除以字符串 queue.news.sport 开头的所有 地址名称。

5. （可选）使用 federation 元素，添加一个 transformer 元素来引用自定义转换器。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
         <include address-match="queue.news.#" />

         <exclude address-match="queue.news.sport.#" />
      </address-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
</federations>
```

name

转换程序配置的名称。此名称在本地代理中必须是唯一的。这是您指定为地址策略的 transformer-ref 属性的值的名称。

class-name

实施 org.apache.activemq.artemis.core.server.transformer.Transformer 接口的用户定义的类的名称。

在传输消息前，会使用消息调用转换器的 transform （） 方法。这可让您在被联邦前转换消息标头或正文。

属性

用于保存特定转换器配置的键值对。

6. 在 federation 元素内，添加一个或多个 上游 元素。每个 upstream 元素都定义了与远程代理的连接，以及应用到该连接的策略。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <upstream name="eu-east-1">
         <static-connectors>
            <connector-ref>eu-east-connector1</connector-ref>
         </static-connectors>
         <policy ref="news-address-federation"/>
      </upstream>

      <upstream name="eu-west-1" >
         <static-connectors>
            <connector-ref>eu-west-connector1</connector-ref>
         </static-connectors>
         <policy ref="news-address-federation"/>
      </upstream>

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
         <include address-match="queue.news.#" />

         <exclude address-match="queue.news.sport.#" />
      </address-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
</federations>
```

static-connectors

包含 connector-ref 元素列表，引用本地代理的 broker.xml 配置文件中定义的 连接器元素。连接器定义用于出站连接的传输(TCP、SSL、HTTP 等)和服务器连接参数（主机、端口等）。此流程的下一步演示了如何添加 static-connectors 元素中引用的连接器。

policy-ref

在应用于上游代理的下游代理上配置的地址策略名称。

您可以为 上游 元素指定的附加选项如下所述：

name

上游代理配置的名称。在本例中，名称对应于名为 **eu-east-1** 和 **eu-west-1** 的上游代理。

user

创建到上游代理的连接时使用的用户名。如果没有指定，则使用在配置 **federation** 元素中指定的共享用户名。

password

创建到上游代理的连接时使用的密码。如果没有指定，则使用在配置 **federation** 元素中指定的共享密码。

call-failover-timeout

与 **call-timeout** 类似，但在故障转移尝试期间调用时使用。默认值为 **-1**， 这意味着禁用超时。

call-timeout

时间，以毫秒为单位，联邦连接在传输作为阻塞调用的数据包时等待来自远程代理的回复。如果这个时间过，连接会抛出异常。默认值为 **30000**。

check-period

周期（以毫秒为单位），在本地代理发送到远程代理连续的"keep-alive"消息之间，用于检查联邦连接的健康状态。如果联邦连接处于健康状态，远程代理会响应每个 **keep-alive** 消息。如果连接不健康，则下游代理无法接收来自上游代理的响应，则使用称为 ***断路器*** 的机制来阻止联邦消费者。如需更多信息，请参阅 **circuit-breaker-timeout** 参数的描述。**check-period** 参数的默认值为 **30000**。

circuit-breaker-timeout

下游和上游代理之间的单个连接可由许多联邦队列和地址消费者共享。如果代理之间的连接丢失，每个联邦消费者可能会同时尝试重新连接。为避免这种情况，称为 ***断路器*** 的机制会阻止使用者。当指定的超时值经过后，断路器会重新记录连接。如果成功，消费者将取消阻塞。否则，断路器再次应用。

connection-ttl

以毫秒为单位，如果联邦连接停止从远程代理接收信息，则联邦连接会保持活跃状态。默认值为 **60000**。

discovery-group-ref

作为定义连接到上游代理的静态连接器的替代选择，此元素可用于指定在 **broker.xml** 配置文件中已经配置的发现组。具体来说，您可以将现有的发现组指定为此元素的

discovery-group-name 属性的值。有关发现组的详情，请参考 第 14.1.6 节 “代理发现方法”。

ha

指定是否为与上游代理的连接启用了高可用性。如果此参数的值设为 true，则本地代理可以连接到上游集群中任何可用代理，并在 live upstream 代理关闭时自动切换到备份代理。默认值为 false。

initial-connect-attempts

下游代理将连接到上游代理的初始尝试次数。如果在不建立连接的情况下达到这个值，则上游代理将被视为永久离线。downstream 代理不再将信息路由到上游代理。默认值为 -1， 这意味着没有限制。

max-retry-interval

与远程代理连接时，后续重新连接尝试之间的最长时间（以毫秒为单位）。默认值为 2000。

reconnect-attempts

如果连接失败，下游代理将尝试重新连接到上游代理的次数。如果在没有重新建立连接的情况下达到这个值，则上游代理被视为永久离线。downstream 代理不再将信息路由到上游代理。默认值为 -1， 这意味着没有限制。

retry-interval

与远程代理的连接失败，在后续重新连接尝试之间周期（以毫秒为单位）。 é»˜è®¤å€¼ä¸º 500ã€,

retry-interval-multiplier

对应用到 retry-interval 参数的值的多重因素。默认值为：1。

share-connection

如果同一代理配置了一个下游和上游连接，则只要下游和上游配置的值设置为 true，就会共享同一连接。默认值为 false。

7. 在本地代理中，将连接器添加到远程代理。这些是联邦地址配置的 static-connectors 元素中引用的连接器。例如：

```
<connectors>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>
```

4.22.4.2.3. 配置下游地址联邦

以下示例演示了如何为独立代理配置下游地址联邦。

下游地址联邦允许您在一个或多个远程代理用来连接到本地代理的本地代理中添加配置。这种方法的优点是您可以在单一代理上保持所有联邦配置。这可能是 hub 和 spoke 拓扑的有用方法，例如：

注意

下游地址联邦会反转联邦连接方向，而不是上游地址配置。因此，当您在配置中添加远程代理时，这些代理被视为 *下游* 代理。下游代理使用配置中的连接信息来连接回本地代理，这现在被视为上游。本例稍后会演示，当您为远程代理添加配置时。

先决条件

- 您应该熟悉上游地址联邦的配置。请参阅 第 4.22.4.2.2 节 “配置上游地址联邦”。
- 以下示例演示了如何配置独立代理间的地址联邦。但是，您还应熟悉为代理 *集群配置* 联邦的要求。更多信息请参阅 第 4.22.4.2.1 节 “为代理集群配置联邦”。

流程

1. 在本地代理上，打开 &lt ;*broker_instance_dir&gt*; /etc/broker.xml 配置文件。
2. 添加包括 <federation> 项的 <federations> 项。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
   </federation>
</federations>
```

3. 添加地址策略配置。例如：

```
<federations>
   ...
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
```

```
      <address-policy name="news-address-federation" max-hops="1" auto-delete="true"
auto-delete-delay="300000" auto-delete-message-count="-1" transformer-ref="news-
transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
         <include address-match="queue.news.#" />

         <exclude address-match="queue.news.sport.#" />
      </address-policy>

   </federation>
 ...
</federations>
```

4. **如果要在传输前转换信息，请添加转换器配置。例如：**

```
<federations>
   ...
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" max-hops="1" auto-delete="true"
auto-delete-delay="300000" auto-delete-message-count="-1" transformer-ref="news-
transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
         <include address-match="queue.news.#" />

         <exclude address-match="queue.news.sport.#" />
      </address-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
 ...
</federations>
```

5. **为每个远程代理添加 downstream 项。例如：**

```
<federations>
   ...
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <downstream name="eu-east-1">
```

```
            <static-connectors>
               <connector-ref>eu-east-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
         </downstream>

         <downstream name="eu-west-1" >
            <static-connectors>
               <connector-ref>eu-west-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
         </downstream>

         <address-policy name="news-address-federation" max-hops="1" auto-delete="true"
auto-delete-delay="300000" auto-delete-message-count="-1" transformer-ref="news-
transformer">
            <include address-match="queue.bbc.new" />
            <include address-match="queue.usatoday" />
            <include address-match="queue.news.#" />

            <exclude address-match="queue.news.sport.#" />
         </address-policy>

         <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
         </transformer>

      </federation>
     ...
</federations>
```

如前面的配置所示，远程代理现在被视为本地代理的下游。下游代理使用配置中的连接信息连接到本地（即 *上游*）代理。

6. **在本地代理中，添加供本地和远程代理用来建立联邦连接的连接器和接收器。例如：**

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61616</connector>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>

<acceptors>
  <acceptor name="netty-acceptor">tcp://localhost:61616</acceptor>
</acceptors>
```

connector name="netty-connector"

本地代理发送到远程代理的连接器配置。远程代理使用此配置连接到本地代理。

connector name="eu-west-1-connector", connector name="eu-east-1-connector"

远程代理连接器。本地代理使用这些连接器来连接到远程代理，并共享远程代理需要连接到本地代理的配置。

acceptor name="netty-acceptor"

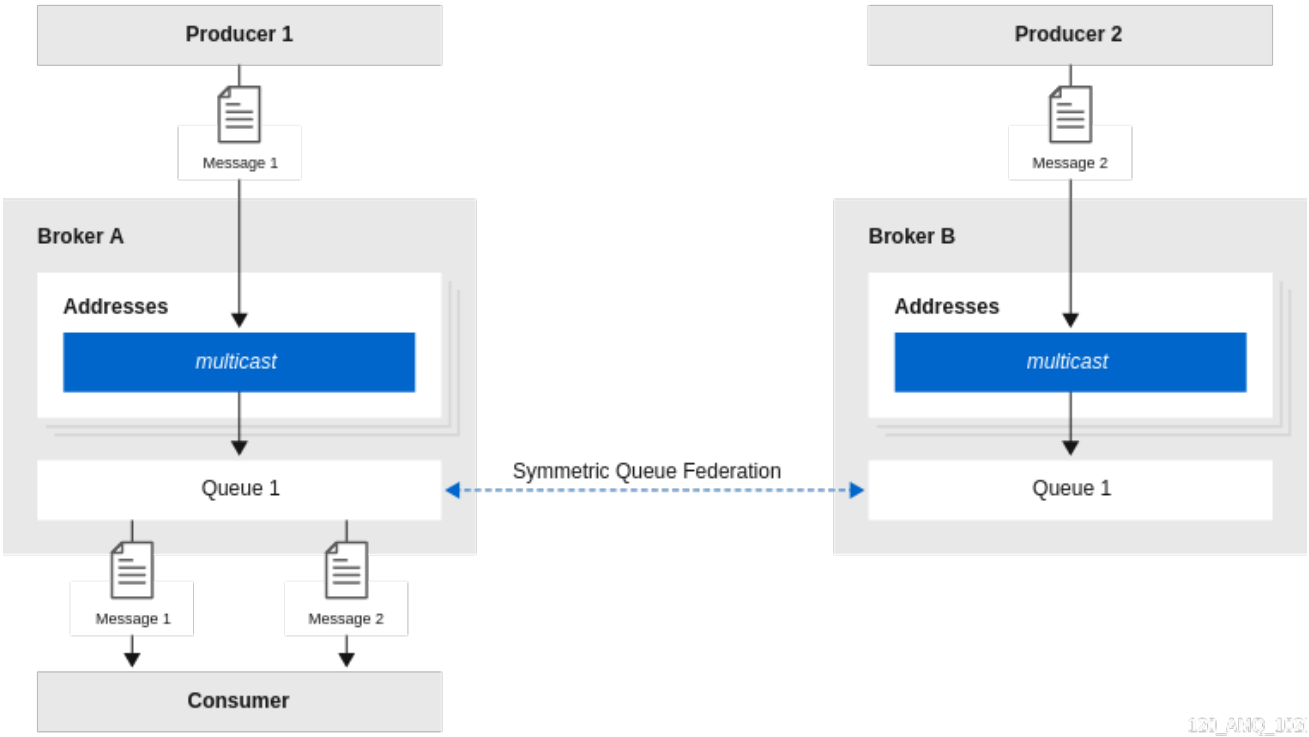
在本地代理上与远程代理使用的连接器对应的接收器来重新连接到本地代理。

4.22.4.2.4. 关于队列联邦

队列联邦提供了一种方式，在本地代理间平衡单个队列的负载。

为实现负载平衡，本地代理从远程队列检索消息，以满足来自本地使用者的消息的需求。下面是一个示例。

图 4.5. 对称队列联邦



远程队列不需要重新配置，且不必位于同一代理或同一集群中。建立远程链接和联邦队列所需的所有配置都位于本地代理中。

4.22.4.2.4.1. 队列联邦的优点

以下是您可能选择配置队列联邦的一些原因。

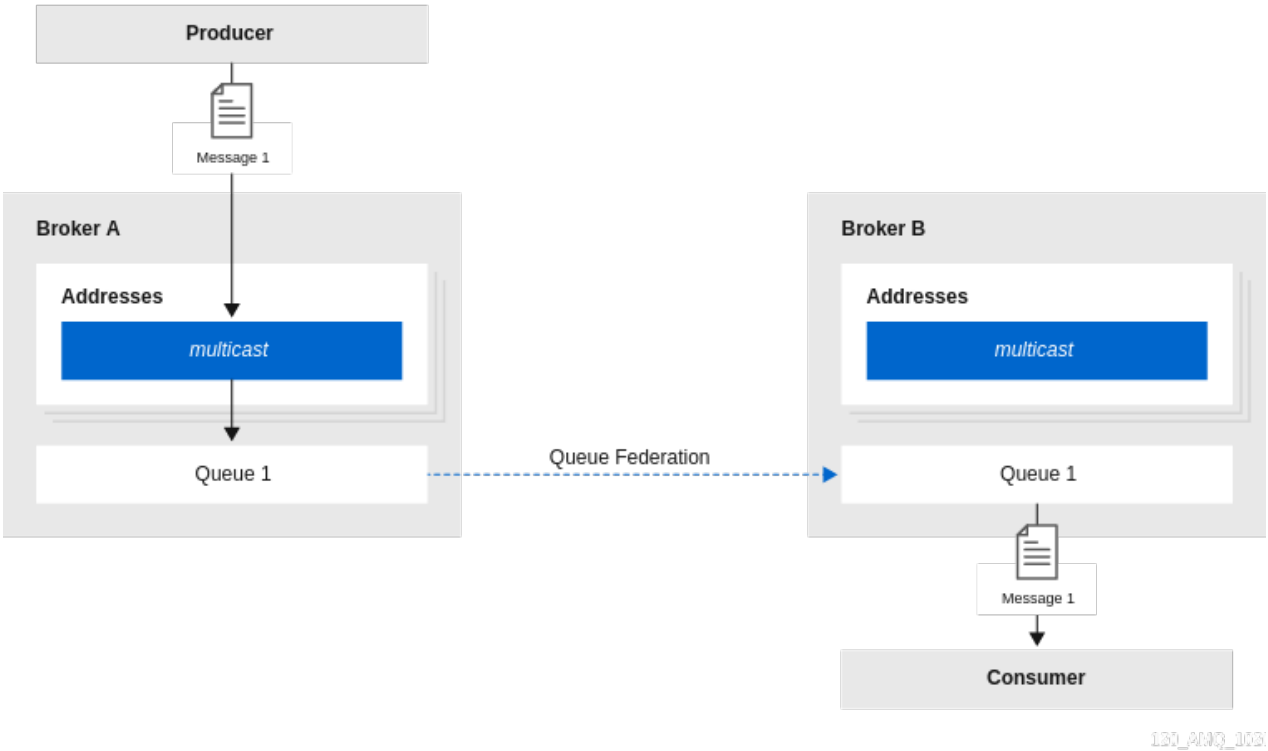
增加容量

队列联邦可以创建在多个代理上分发的"逻辑"队列。这个逻辑分布式队列的容量比单个代理上的单个队列要高。在这个设置中，尽可能多的信息会从最初发布到的代理中消耗。只有在需要负载平衡时，系统才会在联邦中移动消息。

部署多区域设置

在多区域设置中，您可能在一个地区或场合中都有一个消息制作者，并在另一个地区和消费者之间有消息生成者。但是，您应该最好将生成者和消费者连接保持本地到给定区域。在这种情况下，您可以在生成者和消费者的每个区域中部署代理，并使用队列联邦在区域间移动消息(WAN)。下面是一个示例。

图 4.6. 多区域队列联邦



安全企业 LAN 和 DMZ 间的通信

在网络安全中， *demilitarized 区域* (DMZ)是一个物理或逻辑子网，其中包含并公开面向企业的外部服务到不受信任的服务，通常是更大的网络，如互联网。企业的本地区域网络(LAN)的其余部分在防火墙后面保持与这个外部网络隔离。

如果很多消息制作者位于 DMZ 中，且安全企业 LAN 中的许多消费者，则可能不适合生产者连接到安全企业 LAN 中的代理。在这种情况下，您可以在 DMZ 中部署代理，以便生成者可以发布消

息。然后，企业 LAN 中的代理可以连接到 DMZ 中的代理，并使用联邦队列从 DMZ 中的代理接收信息。

4.22.4.2.5. 配置上游队列联邦

以下示例演示了如何为独立代理配置上游队列联邦。在本例中，您要将本地（即 *下游*）代理中的联邦配置为一些远程（即 *上游*）代理。

先决条件

- 以下示例演示了如何配置独立代理间的队列联邦。但是，您还应熟悉为代理 *集群配置* 联邦的要求。更多信息请参阅 第 4.22.4.2.1 节 “为代理集群配置联邦”。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在新的 < federations> 元素中，添加一个 < federation> 元素。例如：

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
  </federation>
</federations>
```

name

联邦配置的名称。在本例中，名称对应于下游代理的名称。

user

用于连接到上游代理的共享用户名。

password

用于连接到上游代理的共享密码。

注意

- 如果用户和密码凭证与上游代理不同，您可以在将代理添加到配置中时单独指定这些代理的凭证。此流程稍后将对此进行描述。

3. 在 **federation** 元素中，添加一个 **< queue-policy>** 元素。为 **< queue-policy& gt;** 元素的属性指定值。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">
      </queue-policy>

   </federation>
</federations>
```

name

队列策略的名称。代理上配置的所有队列策略必须具有唯一的名称。

include-federated

当此属性的值设为 **false** 时，配置不会重新联邦一个已联邦消费者（即联邦队列上的消费者）。这可避免在对称或关闭的拓扑中，没有非联邦消费者，且系统无消息流。

如果您没有 **closed-loop** 拓扑，则可以将此属性的值设置为 **true**。例如，假设您有以下三个代理链： **BrokerA**、**BrokerB** 和 **BrokerC**，以及 **BrokerA** 和消费者位于 **BrokerC** 的制作者。在这种情况下，您将希望 **BrokerB** 重新联邦消费者到 **BrokerA**。

priority-adjustment

当消费者连接到队列时，其优先级在创建上游（即 *联邦*）消费者时使用。联邦消费者的优先级由 **priority-adjustment** 属性的值调整。此属性的默认值为 **-1**， 可确保本地消费者在负载均衡期间优先于联邦消费者。但是，您可以根据需要更改优先级调整的值。

如果优先级调整不足，以防止太多消息移至联邦消费者，这可能会导致消息在代理之间移动，您可以限制移到联邦消费者的消息批处理大小。要限制批处理大小，请在联邦消费者的连接 **URI** 上将 **consumerWindowSize** 值设置为 **0**。

```
tcp://<host>:<port>?consumerWindowSize=0
```

将 consumerWindowSize 值设置为 0 时，AMQ Broker 使用匹配地址的地址设置中的 defaultConsumerWindowSize 参数的值来确定可以在代理之间移动的批处理信息。defaultConsumerWindowSize 属性的默认值为 1048576 字节。

transformer-ref

转换程序配置的名称。如果要在联邦消息传输过程中转换消息，您可以添加转换器配置。此流程稍后会描述转换器配置。

4. 在 &lt ;queue-policy& gt; 元素中，添加 address-matching 模式，使其包含和排除来自队列策略的地址。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />

         <exclude queue-match="#.local" address-match="#" />

      </queue-policy>

   </federation>
</federations>
```

Include

此元素的 address-match 属性的值指定要包含在队列策略中的地址。您可以指定一个准确的地址，如 queue.bbc.new 或 queue.usatoday。或者，您可以使用通配符表达式来指定匹配的地址 *集合*。在前面的示例中，队列策略还包含以字符串 queue.news 开头的所有 地址名称。

与 address-match 属性结合使用，您可以使用 queue-match 属性在队列策略中包含这些地址的特定队列。与 address-match 属性一样，您可以指定准确的队列名称，也可以使用通配符表达式来指定 *一组* 队列。在前面的示例中，数字符号 (#) 通配符代表在每个地址或包括在队列策略中的一组地址中的所有队列。

exclude

此元素的 address-match 属性的值指定要从队列策略中排除的地址。您可以指定一个准确的地址，或使用通配符表达式来指定匹配的地址 *集合*。在前面的示例中，数字符号(#)通

配符表示排除任何匹配跨越 所有地址中的 queue-match 属性的队列。在这种情况下，任何以字符串 .local 结尾的队列都会被排除。这表示某些队列保持为本地队列，而不是联合。

5. 在 federation 元素中，添加一个 transformer 元素来引用自定义转换器实现。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />

         <exclude queue-match="#.local" address-match="#" />

      </queue-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
</federations>
```

name

转换程序配置的名称。此名称在有问题的代理上必须是唯一的。您可以将此名称指定为地址策略的 transformer-ref 属性的值。

class-name

实施 org.apache.activemq.artemis.core.server.transformer.Transformer 接口的用户定义的类的名称。

在传输消息前，会使用消息调用转换器的 transform （） 方法。这可让您在被联邦前转换消息标头或正文。

属性

用于保存特定转换器配置的键值对。

6. 在 federation 元素内，添加一个或多个 上游 元素。每个 upstream 元素都定义了一个上游

代理连接，以及用于应用到该连接的策略。例如：

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <upstream name="eu-east-1">
         <static-connectors>
            <connector-ref>eu-east-connector1</connector-ref>
         </static-connectors>
         <policy ref="news-queue-federation"/>
      </upstream>

      <upstream name="eu-west-1" >
         <static-connectors>
            <connector-ref>eu-west-connector1</connector-ref>
         </static-connectors>
         <policy ref="news-queue-federation"/>
      </upstream>

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />

         <exclude queue-match="#.local" address-match="#" />

      </queue-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
</federations>
```

static-connectors

包含 connector-ref 元素列表，引用本地代理的 broker.xml 配置文件中定义的 连接器元素。连接器定义用于出站连接的传输(TCP、SSL、HTTP 等)和服务器连接参数（主机、端口等）。此步骤的以下步骤演示了如何添加联邦队列配置的 static-connectors 元素引用的连接器。

policy-ref

在应用于上游代理的下游代理上配置的队列策略名称。

您可以为 上游 元素指定的附加选项如下所述：

name

上游代理配置的名称。在本例中，名称对应于名为 **eu-east-1** 和 **eu-west-1** 的上游代理。

user

创建到上游代理的连接时使用的用户名。如果没有指定，则使用在配置 **federation** 元素中指定的共享用户名。

password

创建到上游代理的连接时使用的密码。如果没有指定，则使用在配置 **federation** 元素中指定的共享密码。

call-failover-timeout

与 **call-timeout** 类似，但在故障转移尝试期间调用时使用。默认值为 **-1**， 这意味着禁用超时。

call-timeout

时间，以毫秒为单位，联邦连接在传输作为阻塞调用的数据包时等待来自远程代理的回复。如果这个时间过，连接会抛出异常。默认值为 **30000**。

check-period

周期（以毫秒为单位），在本地代理发送到远程代理连续的"keep-alive"消息之间，用于检查联邦连接的健康状态。如果联邦连接处于健康状态，远程代理会响应每个 **keep-alive** 消息。如果连接不健康，则下游代理无法接收来自上游代理的响应，则使用称为 *断路器* 的机制来阻止联邦消费者。如需更多信息，请参阅 **circuit-breaker-timeout** 参数的描述。**check-period** 参数的默认值为 **30000**。

circuit-breaker-timeout

下游和上游代理之间的单个连接可由许多联邦队列和地址消费者共享。如果代理之间的连接丢失，每个联邦消费者可能会同时尝试重新连接。为避免这种情况，称为 *断路器* 的机制会阻止使用者。当指定的超时值经过后，断路器会重新记录连接。如果成功，消费者将取消阻塞。否则，断路器再次应用。

connection-ttl

以毫秒为单位，如果联邦连接停止从远程代理接收信息，则联邦连接会保持活跃状态。默认值为 **60000**。

discovery-group-ref

作为定义连接到上游代理的静态连接器的替代选择，此元素可用于指定在 broker.xml 配置文件中已经配置的发现组。具体来说，您可以将现有的发现组指定为此元素的 discovery-group-name 属性的值。有关发现组的详情，请参考 第 14.1.6 节 “代理发现方法”。

ha

指定是否为与上游代理的连接启用了高可用性。如果此参数的值设为 true，则本地代理可以连接到上游集群中任何可用代理，并在 live upstream 代理关闭时自动切换到备份代理。默认值为 false。

initial-connect-attempts

下游代理将连接到上游代理的初始尝试次数。如果在不建立连接的情况下达到这个值，则上游代理将被视为永久离线。downstream 代理不再将信息路由到上游代理。默认值为 -1， 这意味着没有限制。

max-retry-interval

与远程代理连接时，后续重新连接尝试之间的最长时间（以毫秒为单位）。默认值为 2000。

reconnect-attempts

如果连接失败，下游代理将尝试重新连接到上游代理的次数。如果在没有重新建立连接的情况下达到这个值，则上游代理被视为永久离线。downstream 代理不再将信息路由到上游代理。默认值为 -1， 这意味着没有限制。

retry-interval

与远程代理的连接失败，在后续重新连接尝试之间周期（以毫秒为单位）。 é»˜è®¤å€¼ä¸º 500ã€,

retry-interval-multiplier

对应用到 retry-interval 参数的值的多重因素。默认值为：1。

share-connection

如果同一代理配置了一个下游和上游连接，则只要下游和上游配置的值设置为 true，就会共享同一连接。默认值为 false。

7. 在本地代理中，将连接器添加到远程代理。这些是联邦地址配置的 static-connectors 元素中引用的连接器。例如：

```
<connectors>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
```

```
    <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
  </connectors>
```

4.22.4.2.6. 配置下游队列联邦

以下示例演示了如何配置下游队列联邦。

下游队列联邦允许您在一个或多个远程代理用来连接到本地代理的本地代理中添加配置。这种方法的优点是您可以在单一代理上保持所有联邦配置。这可能是 hub 和 spoke 拓扑的有用方法，例如：

注意

下游队列联邦联邦联邦连接与上游队列配置方向相反。因此，当您在配置中添加远程代理时，这些代理被视为 *下游* 代理。下游代理使用配置中的连接信息来连接回本地代理，这现在被视为上游。本例稍后会演示，当您为远程代理添加配置时。

先决条件

- 您应该熟悉上游队列联邦的配置。请参阅 第 4.22.4.2.5 节 “配置上游队列联邦”。
- 以下示例演示了如何配置独立代理间的队列联邦。但是，您还应熟悉为代理 *集群配置* 联邦的要求。更多信息请参阅 第 4.22.4.2.1 节 “为代理集群配置联邦”。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。
2. 添加包括 <federation> 项的 <federations> 项。例如：

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
  </federation>
</federations>
```

3. 添加队列策略配置。例如：

```
<federations>
```

```
    ...
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <queue-policy name="news-queue-federation" priority-adjustment="-5" include-
federated="true" transformer-ref="new-transformer">

            <include queue-match="#" address-match="queue.bbc.new" />
            <include queue-match="#" address-match="queue.usatoday" />
            <include queue-match="#" address-match="queue.news.#" />

            <exclude queue-match="#.local" address-match="#" />

        </queue-policy>

    </federation>
  ...
</federations>
```

4. **如果要在传输前转换信息，请添加转换器配置。例如：**

```
<federations>
    ...
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <queue-policy name="news-queue-federation" priority-adjustment="-5" include-
federated="true" transformer-ref="news-transformer">

            <include queue-match="#" address-match="queue.bbc.new" />
            <include queue-match="#" address-match="queue.usatoday" />
            <include queue-match="#" address-match="queue.news.#" />

            <exclude queue-match="#.local" address-match="#" />

        </queue-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
  ...
</federations>
```

5. **为每个远程代理添加 downstream 项。例如：**

```
<federations>
    ...
```

```
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <downstream name="eu-east-1">
            <static-connectors>
                <connector-ref>eu-east-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
        </downstream>

        <downstream name="eu-west-1" >
            <static-connectors>
                <connector-ref>eu-west-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
        </downstream>

        <queue-policy name="news-queue-federation" priority-adjustment="-5" include-
federated="true" transformer-ref="new-transformer">

            <include queue-match="#" address-match="queue.bbc.new" />
            <include queue-match="#" address-match="queue.usatoday" />
            <include queue-match="#" address-match="queue.news.#" />

            <exclude queue-match="#.local" address-match="#" />

        </queue-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
  ...
</federations>
```

如前面的配置所示，远程代理现在被视为本地代理的下游。下游代理使用配置中的连接信息连接到本地（即 *上游*）代理。

6. **在本地代理中，添加供本地和远程代理用来建立联邦连接的连接器和接收器。例如：**

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61616</connector>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>
```

```
<acceptors>
  <acceptor name="netty-acceptor">tcp://localhost:61616</acceptor>
</acceptors>
```

connector name="netty-connector"

本地代理发送到远程代理的连接器配置。远程代理使用此配置连接到本地代理。

connector name="eu-west-1-connector" , connector name="eu-east-1-connector"

远程代理连接器。本地代理使用这些连接器来连接到远程代理，并共享远程代理需要连接到本地代理的配置。

acceptor name="netty-acceptor"

在本地代理上与远程代理使用的连接器对应的接收器来重新连接到本地代理。

第 5 章 保护代理

5.1. 保护连接

当代理连接到消息传递客户端或代理连接到其他代理时，您可以使用传输层安全(TLS)保护这些连接。

您可以使用两个 TLS 配置：

- 单向 TLS，其中只有代理提供证书。这是最常见的配置。
- 双向（或 *mutual*）TLS，代理和客户端（或其他代理）都提供证书。

5.1.1. 配置单向 TLS

以下流程演示了如何为单向 TLS 配置给定接受者。

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。
2. 对于给定的 acceptor，添加 sslEnabled 键并将值设为 true。另外，添加 keyStorePath 和 keyStorePassword 密钥。设置与代理密钥存储对应的值。例如：

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!</accept
or>
```

5.1.2. 配置双向 TLS

以下步骤演示了如何配置双向 TLS。

先决条件

- 您必须已经为单向 TLS 配置了给定接受者。更多信息请参阅 第 5.1.1 节 “配置单向 TLS”。

流程

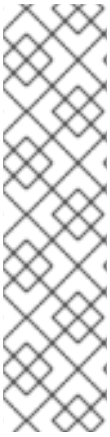
1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 对于您之前为单向 TLS 配置的 acceptor，请添加 needClientAuth 密钥。将值设为 true。例如：

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!;needCl
ientAuth=true</acceptor>
```

3. 上一步中的配置假定客户端证书由可信提供程序签名。如果客户端证书 不是由 可信供应商（例如，是自签名的）签名，则代理需要将客户端证书导入到信任存储中。在这种情况下，添加 trustStorePath 和 trustStorePassword 密钥。设置与代理信任存储对应的值。例如：

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!;needCl
ientAuth=true;trustStorePath=../etc/client.truststore;trustStorePassword=5678!</accept
or>
```



注意

AMQ Broker 支持多种协议，每个协议和平台有不同的方法来指定 TLS 参数。但是，对于使用核心协议（一个网桥）的客户端，TLS 参数在连接器 URL 上配置，与代理接受器上类似。

如果在 Java 虚拟机(JVM)信任存储中被列为可信证书，则 JVM 不会验证证书的到期日期。在生产环境中，红帽建议您使用由证书颁发机构签名的证书。

5.1.3. TLS 配置选项

下表显示了所有可用的 TLS 配置选项。

选项	备注
sslEnabled	指定是否为连接启用 SSL。必须设置为 **true** 以启用 TLS。默认值为 **false**。

选项	备注
keyStorePath	**当在包含代理证书的** 代理（无论是自签名还是由颁发机构签名）上的代理上到 TLS 密钥存储的路径时。 **在连接器中使用时：** 包含客户端证书的客户端上的 TLS 密钥存储的路径。这只有在您使用双向 TLS 时与连接器相关。虽然您可以在代理上配置这个值，但被下载并供客户端使用。如果客户端需要使用与代理上设置的不同路径，它可以使用标准 **javax.net.ssl.keyStore** 系统属性或特定于 AMQ 的 **org.apache.activemq.ssl.keyStore** 系统属性来覆盖代理设置。如果客户端上的另一个组件已使用标准 Java 系统属性，则特定于 AMQ 的系统属性非常有用。
keyStorePassword	**当在 acceptor: Password 中使用** 时，代理上的密钥存储。 **在连接器中使用时：** 客户端上密钥存储的密码。这只有在您使用双向 TLS 时与连接器相关。虽然您可以在代理上配置这个值，但被下载并供客户端使用。如果客户端需要使用与代理上设置的不同密码，那么它可以使用标准 **javax.net.ssl.keyStorePassword** 系统属性或特定于 AMQ 的 **org.apache.activemq.ssl.keyStorePassword** 系统属性来覆盖代理设置。如果客户端上的另一个组件已使用标准 Java 系统属性，则特定于 AMQ 的系统属性非常有用。
trustStorePath	**当在接受器上使用时：** 包含代理信任的所有客户端密钥的代理上 TLS 信任存储的路径。这只有在您使用双向 TLS 时与接受者相关。 **在连接器中使用时：** 包含客户端信任的所有代理的公钥的客户端上 TLS 信任存储的路径。虽然您可以在代理上配置这个值，但被下载并供客户端使用。如果客户端需要使用与服务器上设置的不同路径，那么可以使用标准的 **javax.net.ssl.trustStore** 系统属性或特定于 AMQ 的 **org.apache.activemq.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使用标准 Java 系统属性，则特定于 AMQ 的系统属性非常有用。

选项	备注
trustStorePassword	**当在 acceptor: Password 中使用**时，代理上的信任存储的密码。这只有在您使用双向 TLS 时与接受者相关。 **在连接器中使用时：** 客户端上信任存储的密码。虽然您可以在代理上配置这个值，但被下载并供客户端使用。如果客户端需要使用与代理上设置的不同密码，那么它可以使用标准 **javax.net.ssl.trustStorePassword** 系统属性或特定于 AMQ 的 **org.apache.activemq.ssl.trustStorePassword** 系统属性来覆盖代理设置。如果客户端上的另一个组件已使用标准 Java 系统属性，则特定于 AMQ 的系统属性非常有用。
enabledCipherSuites	以逗号分隔的密码套件列表，用于接受器或连接器的 TLS 通信。 指定客户端应用程序支持的最安全密码套件。如果您指定了代理和客户端通用的、以逗号分隔的密码套件列表，或者您没有指定任何密码套件、代理和客户端相互协商要使用的密码套件。如果您不知道要指定哪个密码套件，您可以首先与以 debug 模式运行的客户端建立 broker-client 连接，以验证代理和客户端通用的密码套件。然后，在代理上配置 **enabledCipherSuites**。 可用的密码套件取决于代理和客户端使用的 TLS 协议版本。如果在升级代理后默认 TLS 协议版本有变化，您可能需要选择早期的 TLS 协议版本，以确保代理和客户端可以使用通用密码套件。如需更多信息，请参阅 **enabledProtocols**。
enabledProtocols	无论是在接收器还是连接器中使用，这是用于 TLS 通信的、以逗号分隔的协议列表。如果没有指定 TLS 协议版本，代理将使用 JVM 的默认版本。 如果代理使用 JVM 的默认 TLS 协议版本，且升级代理后该版本会改变，代理和客户端使用的 TLS 协议版本可能会不兼容。虽然建议您使用更新的 TLS 协议版本，但您可以在 **enabledProtocols** 中指定较早的版本，以便与不支持较新的 TLS 协议版本的客户端进行交互。
needClientAuth	此属性仅适用于 acceptor。它指示客户端连接接受者需要双向 TLS。有效值为 **true** 或者 **false**。默认值为 **false**。

5.2. 验证客户端

5.2.1. 客户端验证方法

要在代理中配置客户端身份验证，您可以使用以下方法：

基于用户名和密码的身份验证

使用以下选项之一直接验证用户凭证：

- 针对本地存储在代理中的一组属性文件检查凭证。您还可以配置允许有限访问代理 *的客户机* 帐户，并组合登录模块来支持更复杂的用例。
- 配置 *轻量级目录访问协议* (LDAP)登录模块，以针对存储在中央 X.500 目录服务器中的用户数据检查客户端凭证。

基于证书的验证

配置双向 *传输层安全* (TLS)，要求代理和客户端同时提供 mutual 身份验证的证书。管理员还必须配置定义批准的客户端用户和角色的属性文件。这些属性文件存储在代理中。

基于 Kerberos 的身份验证

配置代理，以使用 *Simple Authentication and Security Layer* (SASL)框架中的 GSSAPI 机制为客户端验证 Kerberos 安全凭证。

以下章节描述了如何配置基于用户和密码的身份验证。

其他资源

- 要了解 LDAP *和* Kerberos 的完整身份验证和授权工作流，请参阅：
  - 第 5.4 节 “使用 LDAP 进行身份验证和授权”
  - 第 5.5 节 “使用 Kerberos 进行身份验证和授权”

5.2.2. 根据属性文件配置用户和密码身份验证

AMQ Broker 支持灵活的基于角色的安全模型，以便根据其地址将安全性应用到队列。队列绑定到一

对点（用于点到点消息传递）或多对一（用于发布订阅消息传递）的地址。当消息发送到地址时，代理会查找绑定到该地址的队列集合，并将消息路由到该一组队列。

当您需要基本用户和密码身份验证时，请使用 PropertiesLoginModule 定义它。这个登录模块会根据保存在代理中的以下配置文件检查用户凭证：

artemis-users.properties

用于定义用户和对应的密码

artemis-roles.properties

用于定义角色并将用户分配给这些角色

login.config

用于为用户和密码身份验证和客户机访问配置登录模块

artemis-users.properties 文件可以包含哈希密码，以提高安全性。

以下部分演示了如何配置：

- 基本用户和密码身份验证
- 包括客户机访问的用户和密码验证

5.2.2.1. 配置基本用户和密码身份验证

以下步骤演示了如何配置基本用户和密码身份验证。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/login.config 配置文件。默认情况下，新 AMQ Broker 7.12 实例中的此文件包括以下行：

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
    debug=false
```

```
    reload=true
    org.apache.activemq.jaas.properties.user="artemis-users.properties"
    org.apache.activemq.jaas.properties.role="artemis-roles.properties";
     };
```

ActiveMQ

配置的别名。

org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule

实施类。

sufficient

指定 PropertiesLoginModule 所需的成功级别的标志。您可以设置的值有：

- 必需 ：需要登录模块才能成功。身份验证将继续关闭给定别名下配置的登录模块列表，而不考虑成功或失败。
- 先决条件 ：成功需要登录模块。故障会立即向应用程序返回控制。身份验证不会执行在给定别名下配置的登录模块列表。
- sufficient: 登录模块不需要成功。如果成功，则控制会返回到应用程序，并且身份验证不会进一步操作。如果失败，身份验证会尝试关闭在给定别名下配置的登录模块列表。
- 可选 登录模块不是必需的。身份验证将继续关闭给定别名下配置的登录模块列表，而不考虑成功或失败。

org.apache.activemq.jaas.properties.user

指定为登录模块实施定义一组用户和密码的属性文件。

org.apache.activemq.jaas.properties.role

指定属性文件，将用户映射到登录模块实施的定义的角色。

2. 打开 & *lt;broker_instance_dir&*gt; /etc/artemis-users.properties 配置文件。

3. 添加用户并为用户分配密码。例如：

```
user1=secret
user2=access
user3=myPassword
```

4. 打开 & *lt;broker_instance_dir&*gt; /etc/artemis-roles.properties 配置文件。

5. 将角色名称分配给您之前添加到 artemis-users.properties 文件中的用户。例如：

```
admin=user1,user2
developer=user3
```

6. 打开 & *lt;broker_instance_dir&*gt; /etc/bootstrap.xml 配置文件。

7. 如有必要，将您的安全域别名（在这个实例中，*activemq*）添加到该文件中，如下所示：

```
<jaas-security domain="activemq"/>
```

5.2.2.2. 配置客户机访问

对于没有登录凭证或其凭证失败的身份验证的用户，您可以使用客户机帐户授予代理的有限访问权限。

您可以使用命令行参数创建带有启用客户机访问权限的代理实例；-- allow-anonymous （代表 -- require-login）。

以下步骤演示了如何配置客户机访问。

先决条件

- 此流程假设您已配置了基本用户和密码身份验证。如需了解更多相关信息，请参阅 第 5.2.2.1 节 “配置基本用户和密码身份验证”。

流程

1.

打开之前为基本用户和密码身份验证配置的 & *lt;broker_instance_dir>* /etc/login.config 配置文件。

2. 在您之前添加的属性登录模块配置后，添加客户机登录模块配置。例如：

```
activemq {
 org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
    debug=true
    org.apache.activemq.jaas.properties.user="artemis-users.properties"
    org.apache.activemq.jaas.properties.role="artemis-roles.properties";

 org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule sufficient
    debug=true
    org.apache.activemq.jaas.guest.user="guest"
    org.apache.activemq.jaas.guest.role="restricted";
};
```

org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule

实施类。

org.apache.activemq.jaas.guest.user

分配给匿名用户的用户名。

org.apache.activemq.jaas.guest.role

分配给匿名用户的角色。

根据前面的配置，如果用户提供凭据，则会激活用户和密码身份验证模块。如果用户没有提供凭证，或者提供的凭证不正确，则会激活客户机身份验证。

5.2.2.2.1. 客户机访问示例

以下示例显示了为用例配置客户机访问，其中只有没有凭证的用户作为客户机登录。在本例中，观察登录模块的顺序与前面的配置过程相反。另外，附加到属性登录模块的 标志也会更改 为必需。

```
activemq {
   org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule sufficient
      debug=true
     credentialsInvalidate=true
     org.apache.activemq.jaas.guest.user="guest"
     org.apache.activemq.jaas.guest.role="guests";

   org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule requisite
      debug=true
```

```
        org.apache.activemq.jaas.properties.user="artemis-users.properties"
        org.apache.activemq.jaas.properties.role="artemis-roles.properties";
};
```

根据上述配置，如果没有提供登录凭据，则会激活 guest 身份验证模块。

对于这个用例，在配置客户机登录模块时，credentialsInvalidate 选项必须设置为 true。

如果提供了凭证，则会激活属性登录模块。凭据必须有效。

其他资源

- 有关 *Java 身份验证和授权服务* (JAAS)的更多信息，请参阅 Java 供应商中的文档。例如，有关配置 login.config 的 Oracle 指南，请参阅 Oracle Java 文档中的 JAAS 登录 配置文件。
- 要了解如何配置 LDAP 登录模块以验证客户端凭证，请参阅 第 5.4.1 节 “配置 LDAP 以验证客户端”。
- 有关在配置文件中加密密码的详情请参考 第 5.9.2 节 “在配置文件中加密密码”。

5.2.3. 配置基于证书的身份验证

Java 身份验证和授权服务 (JAAS)证书登录模块处理使用传输层安全(TLS)的客户端的身份验证和授权。模块需要使用双向 *传输层安全* (TLS)，并使用自己的证书配置客户端。身份验证在 TLS 握手期间执行，而不是直接由 JAAS 证书登录模块执行。

证书登录模块的角色是：

- 限制可接受的用户集合。只有相关属性文件中明确列出的用户 *可辨识名称* (DN)才有资格进行身份验证。
- 将组列表与接收的用户身份相关联。这有助于授权。
- 需要存在传入的客户端证书（默认情况下，TLS 层配置为将客户端证书存在视为可选）。

certificate 登录模块将证书 DN 的集合存储在一对平面文本文件中。文件将用户名和组群 ID 列表与每个 DN 关联。

证书登录模块由 org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule 类实现。

5.2.3.1. 将代理配置为使用基于证书的身份验证

以下流程演示了如何将代理配置为使用基于证书的身份验证。

先决条件

- 您必须将代理配置为使用双向传输层安全(TLS)。更多信息请参阅 第 5.1.2 节 “配置双向 TLS”。

流程

1. 从之前导入到代理密钥存储的用户证书获取 Subject *Distinguished Names* (DN)。

   a. 将密钥存储文件的证书导出到临时文件中。例如：

   ```
   keytool -export -file <file_name> -alias broker-localhost -keystore broker.ks -storepass
   <password>
   ```

   b. 输出导出的证书的内容：

   ```
   keytool -printcert -file <file_name>
   ```

   输出结果类似如下：

   ```
   Owner: CN=localhost, OU=broker, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
   Issuer: CN=localhost, OU=broker, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
   Serial number: 4537c82e
   Valid from: Thu Oct 19 19:47:10 BST 2006 until: Wed Jan 17 18:47:10 GMT 2007
   Certificate fingerprints:
           MD5:  3F:6C:0C:89:A8:80:29:CC:F5:2D:DA:5C:D7:3F:AB:37
           SHA1: F0:79:0D:04:38:5A:46:CE:86:E1:8A:20:1F:7B:AB:3A:46:E4:34:5C
   ```

Owner 条目是主题 DN。用于输入主题 DN 的格式取决于您的平台。以上字符串也可以表示为;

```
Owner: `CN=localhost,\ OU=broker,\ O=Unknown,\ L=Unknown,\ ST=Unknown,\
C=Unknown`
```

2. 配置基于证书的身份验证。

a. 打开 & *lt;broker_instance_dir&*gt; /etc/login.config 配置文件。添加 certificate 登录模块并引用用户和角色属性文件。例如：

```
activemq {

org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule
      debug=true
      org.apache.activemq.jaas.textfiledn.user="artemis-users.properties"
      org.apache.activemq.jaas.textfiledn.role="artemis-roles.properties";
};
```

org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule

实施类。

org.apache.activemq.jaas.textfiledn.user

指定为登录模块实施定义一组用户和密码的属性文件。

org.apache.activemq.jaas.textfiledn.role

指定属性文件，将用户映射到登录模块实施的定义的角色。

b. 打开 & *lt;broker_instance_dir&*gt; /etc/artemis-users.properties 配置文件。在此文件中定义用户及其对应的 DN。例如：

```
system=CN=system,O=Progress,C=US
user=CN=humble user,O=Progress,C=US
guest=CN=anon,O=Progress,C=DE
```

例如，根据上述配置，名为 system 的用户映射到 CN=system,O=Progress,C=US Subject DN。

c. 打开 & *lt;broker_instance_dir&*gt; /etc/artemis-roles.properties 配置文件。此文件

中定义包含这些角色的可用角色和用户。例如：

```
admins=system
users=system,user
guests=guest
```

在前面的配置中，对于 users 角色，您将以逗号分隔列表列出多个用户。

d. 请确保您的安全域别名（在本例中是 *activemq*）在 bootstrap.xml 中引用，如下所示：

```
<jaas-security domain="activemq"/>
```

5.2.3.2. 为 AMQP 客户端配置基于证书的身份验证

在 *连接到代理时，使用简单身份验证和安全层* (SASL) EXTERNAL 机制配置参数，为基于证书的身份验证配置 AQMP 客户端。

代理会以验证任何证书的相同方式验证您的 AMQP 客户端的 *Transport Layer Security* (TLS)/*Secure Sockets Layer* (SSL) 证书：

1. 代理读取客户端的 TLS/SSL 证书，以从证书主体获取身份。
2. 证书主题通过 certificate 登录模块映射到代理身份。然后，代理会根据用户的角色授权用户。

以下流程演示了如何为 AMQP 客户端配置基于证书的身份验证。要让 AMQP 客户端能够使用基于证书的身份验证，您必须将配置参数添加到客户端用来连接到代理的 URI 中。

先决条件

- 您必须已配置：
  - 双向 TLS.更多信息请参阅 第 5.1.2 节 “配置双向 TLS”。
  -

使用基于证书的验证的代理。更多信息请参阅 第 5.2.3.1 节 “将代理配置为使用基于证书的身份验证”。

流程

1. 打开包含用于编辑的 URI 的资源：

```
amqps://localhost:5500
```

2. 添加参数 sslEnabled=true 来为连接启用 TSL/SSL ：

```
amqps://localhost:5500?sslEnabled=true
```

3. 添加与客户端信任存储和密钥存储相关的参数，以启用代理与 TSL/SSL 证书交换：

```
amqps://localhost:5500?
sslEnabled=true&trustStorePath=<trust_store_path>&trustStorePassword=<trust_store_
password>&keyStorePath=<key_store_path>&keyStorePassword=<key_store_password>
```

4. 添加参数 saslMechanisms=EXTERNAL，以使用其 TSL/SSL 证书中找到的身份来请求代理验证客户端：

```
amqps://localhost:5500?
sslEnabled=true&trustStorePath=<trust_store_path>&trustStorePassword=<trust_store_
password>&keyStorePath=<key_store_path>&keyStorePassword=<key_store_password>
&saslMechanisms=EXTERNAL
```

其他资源

- 有关在 AMQ Broker 中基于证书的身份验证的更多信息，请参阅 第 5.2.3.1 节 “将代理配置为使用基于证书的身份验证”。

- 有关配置 AMQP 客户端的更多信息，请访问红帽客户门户网站 以获取特定于您的客户端产品文档。

5.3. 授权客户端

5.3.1. 客户端授权方法

要授权客户端在代理上执行操作，如创建和删除地址和队列，以及发送和使用消息，您可以使用以下方法：

基于用户和基于角色的授权

为经过身份验证的用户和角色配置代理安全设置。

配置 LDAP 以授权客户端

配置 *轻量级目录访问协议* (LDAP)登录模块，以处理身份验证和授权。LDAP 登录模块针对存储在中央 X.500 目录服务器中的用户数据检查传入的凭证，并根据用户角色设置权限。

配置 Kerberos 以授权客户端

配置 *Java 认证和授权服务* (JAAS) Krb5LoginModule 登录模块，将凭证传递给 PropertiesLoginModule 或 LDAPLoginModule 登录模块，它将 Kerberos 验证的用户映射到 AMQ Broker 角色。

5.3.2. 配置基于用户和基于角色的授权

5.3.2.1. 设置权限

通过 broker.xml 配置文件中的 < security-setting> 元素，根据队列（基于其地址）定义权限。您可以在配置文件的 <security-settings> 项中定义多个 <security-setting> 实例。您可以指定一个准确的地址匹配，或者使用数字符号(#)和星号(*)通配符字符来定义通配符匹配。

可以为与地址匹配的队列集授予不同的权限。这些权限显示在下表中。

允许用户进行...	使用此参数...
创建地址	**createAddress**
删除地址	**deleteAddress**
在匹配地址下创建持久队列	**createDurableQueue**
删除匹配地址下的持久队列	**deleteDurableQueue**
在匹配地址下创建非持久队列	**createNonDurableQueue**
删除匹配地址下的非持久队列	**deleteNonDurableQueue**
发送消息到匹配地址	**send**

允许用户进行...	使用此参数...
使用绑定到匹配地址的队列中的消息	**consume**
通过将管理消息发送到管理地址来调用管理操作	**管理**
浏览绑定到匹配地址的队列	**browse**
具有对管理操作子集的只读访问权限	**view**
访问变异管理操作，这是任何没有通过 **view** 权限授予访问权限的操作。	**edit**

对于每个权限，您可以指定授予该权限的角色列表。如果给定用户具有任何角色，则它们被授予该地址集合的权限。

以下章节显示了权限的一些配置示例。

5.3.2.1.1. 为单个地址配置消息 production

以下步骤演示了如何为单个地址配置消息生产权限。

流程

1. **打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。**

2. **在 <security-settings> 项中添加一个单一的 <security-setting> 项。对于 match 键，请指定地址。例如：**

```
<security-settings>
   <security-setting match="my.destination">
      <permission type="send" roles="producer"/>
   </security-setting>
</security-settings>
```

根据上述配置，制作者 角色的成员具有地址 my.destination 的 发送 权限。

5.3.2.1.2. 为单个地址配置消息消耗

以下流程演示了如何为单个地址配置消息消耗权限。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 <security-settings> 项中添加一个单一的 <security-setting> 项。对于 match 键，请指定地址。例如：

```
<security-settings>
   <security-setting match="my.destination">
      <permission type="consume" roles="consumer"/>
   </security-setting>
</security-settings>
```

根据上述配置，consumer 角色的成员具有地址 my.destination 的 consume 权限。

5.3.2.1.3. 在所有地址上配置完整的访问

以下步骤演示了如何配置对所有地址和相关队列的完整访问权限。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 <security-settings> 项中添加一个单一的 <security-setting> 项。对于 match 键，若要配置为对所有地址的访问，使用数字符号 (#) 通配符字符。例如：

```
<security-settings>
   <security-setting match="#">
      <permission type="createDurableQueue" roles="guest"/>
      <permission type="deleteDurableQueue" roles="guest"/>
      <permission type="createNonDurableQueue" roles="guest"/>
      <permission type="deleteNonDurableQueue" roles="guest"/>
      <permission type="createAddress" roles="guest"/>
      <permission type="deleteAddress" roles="guest"/>
      <permission type="send" roles="guest"/>
      <permission type="browse" roles="guest"/>
      <permission type="consume" roles="guest"/>
      <permission type="manage" roles="guest"/>
   </security-setting>
</security-settings>
```

-

根据上述配置，所有权限都授予所有队列的 *guest* 角色的成员。这在将匿名身份验证配置为为每个用户分配 guest 角色时非常有用。

其他资源

- 要了解配置更复杂的用例的信息，请参阅 第 5.3.2.1.4 节 “配置多个安全设置”。

5.3.2.1.4. 配置多个安全设置

以下示例步骤演示了如何为匹配的地址集合单独配置多个安全设置。这与本节中前面的示例相反，它演示了如何为 *所有*地址授予 *完全*访问权限

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 <security-settings> 项中添加一个单一的 <security-setting> 项。对于 match 键，请包含数字符号(#)通配符字符，以将设置应用到匹配的地址 *集合*。例如：

```
<security-setting match="globalqueues.europe.#">
  <permission type="createDurableQueue" roles="admin"/>
  <permission type="deleteDurableQueue" roles="admin"/>
  <permission type="createNonDurableQueue" roles="admin, guest, europe-users"/>
  <permission type="deleteNonDurableQueue" roles="admin, guest, europe-users"/>
  <permission type="send" roles="admin, europe-users"/>
  <permission type="consume" roles="admin, europe-users"/>
</security-setting>
```

match=globalqueues.europe.#

数字符号(#)通配符字符由代理解释为"任意单词序列"。单词用句点(.)分隔。在本例中，安全设置适用于以字符串 *globalqueues.europe*开头的任何地址。

permission type="createDurableQueue"

只有具有 admin 角色的用户才能创建或删除绑定到字符串 *globalqueues.europe*的地址的持久队列。

permission type="createNonDurableQueue"

具有 admin、guest 或 europe-users 角色的任何用户可以创建或删除绑定到字符串 *globalqueues.europe*的地址的临时队列。

permission type="send"

具有 admin 或 europe-users 角色的任何用户可以发送消息到绑定到字符串 *globalqueues.europe*的地址的队列。

permission type="consume"

具有 admin 或 europe-users 角色的任何用户都可以使用绑定到以字符串 *globalqueues.europe*开头的队列的消息。

3. （可选） 要将不同的安全设置应用到更严格的地址集合，请添加另一个 < security-setting> 元素。对于 match 键，请指定更具体的文本字符串。例如：

```
<security-setting match="globalqueues.europe.orders.#">
  <permission type="send" roles="europe-users"/>
  <permission type="consume" roles="europe-users"/>
</security-setting>
```

在第二个 security-setting 元素中，globalqueues.europe.orders.# 的匹配比第一个 security-setting 元素中的 globalqueues.europe.# 匹配更加具体。对于与 globalqueues.europe.orders.# 匹配的任何地址，权限 createDurableQueue, deleteDurableQueue, createNonDurableQueue, deleteNonDurableQueue 不是从文件的第一个 security-setting 元素继承的。例如，对于地址 globalqueues.europe.orders.plastics，唯一存在的权限为 europe-users 发送 和使用。

因此，因为一个 security-setting 块中指定的权限不会被另一个继承，因此您可以通过不指定这些权限，实际上可以有效地拒绝更具体的 security-setting 块中的权限。

5.3.2.1.5. 使用用户配置队列

自动创建队列时，会为队列分配连接客户端的用户名。此用户名作为元数据包含在队列中。名称由 JMX 和 AMQ Broker 管理控制台公开。

以下流程演示了如何将用户名添加到您在代理配置中手动定义的队列中。

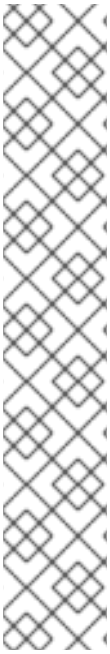
流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 对于给定队列，添加用户 密钥。分配一个值。例如：

```
<address name="ExampleQueue">
  <anycast>
    <queue name="ExampleQueue" user="admin"/>
  </anycast>
</address>
```

根据前面的配置，admin 用户分配到队列 ExampleQueue。



注意

- 在队列中配置用户不会更改该队列的任何安全语义 - 它仅用于该队列的元数据。
- 用户之间的映射及其已由称为 *安全管理器* 的组件处理的角色。安全管理器从存储在代理的属性文件中读取用户凭证。默认情况下，AMQ Broker 使用 org.apache.activemq.artemis.spi.core.security.ActiveMQJAASSecurityManager 安全管理器。此默认安全管理器提供与 JAAS 和 Red Hat JBoss Enterprise Application Platform (JBoss EAP)安全性的集成。

  要了解如何 *使用自定义* 安全管理器，请参阅 第 5.6.2 节 “指定自定义安全管理器”。

5.3.2.2. 配置基于角色的访问控制

基于角色的访问控制 (RBAC)用于限制对 MBeans 属性和方法的访问。Mbeans 是 AMQ Broker 公开管理 API 的方式，以支持管理操作。

您可以使用以下方法之一限制对 MBeans 的访问：

- 在 management.xml 文件中配置 authorisation 元素，这是默认方法。
- 在 broker.xml 文件中配置安全设置。

与更新 management.xml 文件时，您不需要在更改 broker.xml 文件中的安全设置后重启代理。

5.3.2.2.1. 在 management.xml 文件中配置基于角色的访问控制。

为管理操作配置基于角色访问控制的默认方法是，在 management.xml 文件中配置 authorisation 元素。

以下示例步骤演示了如何将角色映射到特定的 MBeans 及其属性和方法。

先决条件

- 您定义的用户和角色。更多信息请参阅 第 5.2.2.1 节 “配置基本用户和密码身份验证”。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/management.xml 配置文件。
2. 搜索 role-access 元素并编辑配置。例如：

```
<role-access>
  <match domain="org.apache.activemq.artemis">
    <access method="list*" roles="view,update,amq"/>
    <access method="get*" roles="view,update,amq"/>
    <access method="is*" roles="view,update,amq"/>
    <access method="set*" roles="update,amq"/>
    <access method="*" roles="amq"/>
  </match>
</role-access>
```

- 在本例中，匹配应用于具有域名 org.apache.activemq.apache 的任何 MBean 属性。
- 访问 view, update, 或 amq 角色来匹配 MBean 属性由您添加到角色的 list*, get*, set*, is*, 和 * 访问方法控制。method="*" (通配符) 语法作为一个 catch-all 的方式，适用于没有在配置中列出的所有其他方法。配置中的每个访问方法都会转换为 MBean 方法调用。
- 调用的 MBean 方法与配置中列出的方法匹配。例如，如果您在带有 org.apache.activemq.artemis 域的 MBean 上调用名为 listMessages 的方法，则代理会将访问权限匹配回 列表 方法配置中定义的角色。
- 您还可以使用完整的 MBean 方法名称配置访问权限。例如：

```
<access method="listMessages" roles="view,update,amq"/>
```

3. 启动或重启代理。

- 在 Linux: & *lt;broker_instance_dir>* /bin/artemis run
- 在 Windows 上： *<broker_instance_dir>* \bin\artemis-service.exe start

您还可以通过添加与 MBean 属性匹配的 key 属性来匹配域中的特定 MBeans。

5.3.2.2.1.1. 基于角色的访问示例

本节演示了以下应用基于角色的访问控制的示例：

- 将角色映射到域中的所有队列。
- 将角色映射到域 中的特定队列。
- 将角色映射到包含指定前缀 的所有队列名称。
- 将不同的角色映射到不同的队列集合。

以下示例演示了如何使用 key 属性将角色映射到指定域中的所有队列。

```
<match domain="org.apache.activemq.artemis" key="subcomponent=queues">
  <access method="list*" roles="view,update,amq"/>
  <access method="get*" roles="view,update,amq"/>
  <access method="is*" roles="view,update,amq"/>
  <access method="set*" roles="update,amq"/>
  <access method="*" roles="amq"/>
</match>
```

以下示例演示了如何使用 key 属性将角色映射到特定的名为 queue。在本例中， 命名队列为 exampleQueue。

```
<match domain="org.apache.activemq.artemis" key="queue=exampleQueue">
```

```
    <access method="list*" roles="view,update,amq"/>
    <access method="get*" roles="view,update,amq"/>
    <access method="is*" roles="view,update,amq"/>
    <access method="set*" roles="update,amq"/>
    <access method="*" roles="amq"/>
  </match>
```

以下示例演示了如何将角色映射到名称包含指定前缀的每个队列。在本例中，使用星号(*)通配符运算符匹配所有以前缀 ***example*** 开头的队列名称。

```
<match domain="org.apache.activemq.artemis" key="queue=example*">
  <access method="list*" roles="view,update,amq"/>
  <access method="get*" roles="view,update,amq"/>
  <access method="is*" roles="view,update,amq"/>
  <access method="set*" roles="update,amq"/>
  <access method="*" roles="amq"/>
</match>
```

您可能想要对同一属性的不同集合（例如，不同的队列集合）以不同的方式映射角色。在这种情况下，您可以在配置文件中包含多个 匹配 元素。但是，可以在同一域中有多个匹配项。

例如，假设两个 & lt;match& gt; 元素配置如下：

```
<match domain="org.apache.activemq.artemis" key="queue=example*">
```

和

```
<match domain="org.apache.activemq.artemis" key="queue=example.sub*">
```

基于此配置，org.apache.activemq.artemis 域中的名为 example.sub.queue 的队列与两个通配符键表达式匹配。因此，代理需要一个优先方案来决定哪组角色映射到队列；在第一个 match 项中指定的角色，或在第二个 match 项中指定的角色。

当同一域中有多个匹配项时，代理会在映射角色时使用以下优先级方案：

- 完全匹配优先于通配符匹配
- 较长的通配符匹配优先于较短的通配符匹配

在本例中，因为较长的通配符表达式与 example.sub.queue 的队列名称匹配，所以代理会应用第二个 < match > 元素中配置的 role-mapping。

注意

default-access 元素是每个方法调用的一个 catch-all 元素，它不使用 role-access 或 whitelist 配置进行处理。default-access 和 role-access 元素具有相同的 match 元素语义。

5.3.2.2.1.2. 配置 allowlist 元素

allowlist 是一组预先批准的域或 MBeans，不需要用户身份验证。您可以提供域列表或 MBeans 列表，或两者必须绕过身份验证。例如，您可以使用 允许列表 来指定运行 AMQ Broker 管理控制台所需的任何 MBeans。

以下示例流程演示了如何配置 allowlist 元素。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/management.xml 配置文件。

2. 搜索 allowlist 元素并编辑配置：

```
<allowlist>
  <entry domain="hawtio"/>
</allowlist>
```

在本例中，任何带有域 hawtio 的 MBean 都被允许在没有身份验证的情况下访问。您还可以对 MBean 属性使用 < entry domain="hawtio" key="typePROFILE" /> 形式的通配符条目。

3. 启动或重启代理。

- 在 Linux: & *lt;broker_instance_dir&gt*; /bin/artemis run
- 在 Windows 上： *<broker_instance_dir&*gt; \bin\artemis-service.exe start

5.3.2.2.2. 在 broker.xml 文件中配置基于角色的访问控制

您可以在 broker.xml 文件中为管理操作配置基于角色的访问控制，而不是 management.xml 文件。对 broker.xml 文件中的权限更新不需要代理重启才能生效。

在 broker.xml 文件中，您可以为管理操作授予 view 或 edit 权限。特定管理操作可供具有 查看或编辑 权限的角色使用，由预定义的正则表达式控制。任何匹配正则表达式的操作都可以由具有 view 权限的角色访问，所有其他操作都需要 编辑权限。

先决条件

- 您定义的用户和角色。更多信息请参阅 第 5.2.2.1 节 “配置基本用户和密码身份验证”。

流程

1. 从 management.xml 文件中删除 authorisation 元素配置，以防止代理使用此文件中的默认 RBAC 配置。
   a. 编辑 & lt;broker_instance_dir>/etc/management.xml 文件。
   b. 从文件中删除 authorisation 元素配置。
   c. 保存 & lt;broker_instance_dir>/etc/management.xml 文件。
2. 在代理 JVM 中添加环境变量，将代理配置为使用 broker.xml 文件中的 RBAC 配置。
   a. 打开 & lt;broker_instance_dir>/etc/artemis.profile 文件。
   b. 在 Java 系统参数的 JAVA_ARGS 列表 中添加以下参数：

```
-Djavax.management.builder.initial=org.apache.activemq.artemis.core.server.management.ArtemisRbacMBeanServerBuilder
```

c. 保存 artemis.profile 文件。

3. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 文件，以配置 RBAC 来管理操作。

4. 搜索 security-settings 元素，再为管理操作添加一个 security-setting 元素。

管理操作的匹配地址的格式是：

<_management-rbac-prefix_>.<_resource type_>.<_resource name_>.<_operation_>

management-rbac-prefix 参数的默认值为 mops。

在以下示例 RBAC 配置中，匹配地址中的数字符号(#)将 admin 角色 视图 和编辑 权限授予所有 MBeans。

```
<security-settings>
  ..
  <security-setting match="mops.#">
   <permission type="view" roles="admin"/>
   <permission type="edit" roles="admin"/>
  </security-setting>
  ..
</security-setting>
```

5. 保存 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

其他管理操作的基于角色的访问控制示例

以下示例将 管理器 角色 视图 和编辑权限授予 activemq.management 地址。操作位置中的星号 packagemanifests 授予对所有操作的访问权限。

```
<security-setting match="mops.address.activemq.management.*">
   <permission type="view" roles="manager"/>
</security-setting>
```

以下示例有一个空角色列表，它拒绝所有用户使用代理 MBean 执行指定操作的权限 forceFailover。

```
<security-setting match="mops.broker.forceFailover">
  <permission type="edit" roles=""/>
</security-setting>
```

5.3.2.3. 设置资源限值

有时，设置特定限制会超出与授权和身份验证相关的普通安全设置以外的特定限制。

5.3.2.3.1. 配置连接和队列限制

以下示例步骤演示了如何限制用户可以创建的连接和队列的数量。

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 添加 resource-limit-settings 元素。为 max-connections 和 max-queues 指定值。例如：

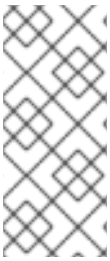
```
<resource-limit-settings>
  <resource-limit-setting match="myUser">
    <max-connections>5</max-connections>
    <max-queues>3</max-queues>
  </resource-limit-setting>
</resource-limit-settings>
```

max-connections

定义匹配用户可以在代理上创建的会话数量。默认值为 **-1**，这意味着没有限制。如果要限制会话数量，请考虑从 AMQ 核心协议 JMS 客户端到代理的连接会创建两个会话。

max-queues

定义匹配用户可以创建的队列数。默认值为 **-1**，这意味着没有限制。



注意

与您可以在代理配置的 address-setting 元素中指定的 匹配 字符串不同，您在 resource-limit-settings 中指定的匹配字符串 无法使用 通配符语法。相反，匹配字符串会定义将资源限值设置应用到的特定用户。

5.4. 使用 LDAP 进行身份验证和授权

LDAP 登录模块通过根据存储在中央 X.500 目录服务器中的用户数据检查传入的凭证来启用身份验证和授权。它通过 org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule 实施。

5.4.1. 配置 LDAP 以验证客户端

以下示例步骤演示了如何使用 LDAP 验证客户端。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 在 security-settings 元素中，添加一个 security-setting 元素来配置权限。例如：

```
<security-settings>
   <security-setting match="#">
      <permission type="createDurableQueue" roles="user"/>
      <permission type="deleteDurableQueue" roles="user"/>
      <permission type="createNonDurableQueue" roles="user"/>
      <permission type="deleteNonDurableQueue" roles="user"/>
      <permission type="send" roles="user"/>
      <permission type="consume" roles="user"/>
   </security-setting>
</security-settings>
```

上述配置 将所有 队列的特定权限 分配给用户 角色的成员。

3. 打开 & *lt;broker_instance_dir&gt*; /etc/login.config 文件。

4. 根据您使用的目录服务，配置 LDAP 登录模块。

   a. 如果您使用 Microsoft Active Directory 目录服务，请添加类似以下示例的配置：

```
activemq {
 org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
   debug=true
   initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
   connectionURL="LDAP://localhost:389"

connectionUsername="CN=Administrator,CN=Users,OU=System,DC=example,DC=
```

```
com"
    connectionPassword=redhat.123
    connectionProtocol=s
    connectionTimeout="5000"
    authentication=simple
    userBase="dc=example,dc=com"
    userSearchMatching="(CN={0})"
    userSearchSubtree=true
    readTimeout="5000"
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true
    ;
};
```

注意

如果您使用 Microsoft Active Directory，而您需要为 connectionUsername 属性指定的值包含一个空格（例如 OU=System Accounts），那么您必须将该值放在一对双引号("")中，并使用反斜杠(\)来转义对中的每个双引号。例如，connectionUsername="CN=Administrator,CN=Users,OU=\"System Accounts\",DC=example,DC=com"。

b. 如果您使用 ApacheDS 目录服务，请添加类似以下示例的配置：

```
activemq {
 org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
    debug=true
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
    connectionURL="ldap://localhost:10389"
    connectionUsername="uid=admin,ou=system"
    connectionPassword=secret
    connectionProtocol=s
    connectionTimeout=5000
    authentication=simple
    userBase="dc=example,dc=com"
    userSearchMatching="(uid={0})"
    userSearchSubtree=true
    userRoleName=
    readTimeout=5000
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true
    ;
};
```

debug

打开调试(true)或关闭(假)。默认值为 false。

initialContextFactory

必须始终设置为 com.sun.jndi.ldap.LdapCtxFactory

connectionURL

使用 LDAP URL 的目录服务器的位置__<ldap://Host:Port>。通过添加正斜杠 / 以及目录树中特定节点的 DN，可以选择性地验证此 URL。对于 Microsoft AD，Apache DS 的默认端口为 10389，默认为 389。

connectionUsername

打开与目录服务器的连接的用户的区分名称(DN)。例如：uid=admin,ou=system。目录服务器通常需要客户端提供用户名/密码凭证才能打开连接。

connectionPassword

与来自 connectionUsername 的 DN 匹配的密码。在目录服务器中，在 *Directory Information Tree* (DIT)中，密码通常作为 userPassword 属性存储在对应的目录条目中。

connectionProtocol

支持任何值，但实际上没有使用。必须明确设置这个选项，因为它没有默认值。

connectionTimeout

指定代理连接到目录服务器所需的最长时间（以毫秒为单位）。如果代理无法在此时间内连接到目录 sever，它会中止连接尝试。如果为此属性指定 0 或更小的值，则会使用底层 TCP 协议的超时值。如果没有指定值，代理会无限期等待建立连接，或者底层网络超时。

当为连接请求连接池时，此属性指定在达到最大池大小并且使用池中所有连接时代理等待连接的最长时间。如果您指定了零或更少的值，代理会无限期等待连接可用。否则，代理会在达到最大等待时间时中止连接尝试。

身份验证

指定绑定到 LDAP 服务器时使用的身份验证方法。这个参数可以设置为 simple（需要用户名和密码）或 none （允许匿名访问）。

userBase

选择 DIT 的特定子树来搜索用户条目。子树由 DN 指定，用于指定子树的基本节点。例如，通过将此选项设置为 ou=User,ou=ActiveMQ,ou=system，搜索用户条目仅

限于 ou=User,ou=ActiveMQ,ou=system 节点下的子树。

userSearchMatching

指定一个 LDAP 搜索过滤器，它应用到 userBase 所选的子树。详情请查看 第 5.4.1.1 节 “搜索匹配参数” 部分。

userSearchSubtree

相对于 userBase 指定的节点，指定用户条目的搜索深度。此选项是一个布尔值。指定 false 值表示搜索会尝试匹配 userBase 节点的一个子条目（映射到 javax.naming.directory.SearchControls.ONELEVEL_SCOPE）。值为 true 表示搜索会尝试与属于 userBase 节点 *的子树* 的任何条目匹配（映射到 javax.naming.directory.SearchControls.SUBTREE_SCOPE）。

userRoleName

用户条目的名称，其中包含用户的角色名称列表。角色名称由代理的授权插件解释为组名称。如果省略这个选项，则不会从用户条目中提取角色名称。

readTimeout

指定代理可以等待从目录服务器接收响应到 LDAP 请求的最长时间（以毫秒为单位）。如果代理目前没有收到来自目录服务器的响应，代理会中止请求。如果您指定了零或更少的值，或者没有指定值，代理会无限期等待目录服务器的响应到 LDAP 请求。

roleBase

如果角色数据直接存储在目录服务器中，可以使用角色选项 (roleBase,roleSearchMatching,roleSearchSubtree, 和 roleName)作为指定 userRoleName 选项的替代方案。这个选项选择 DIT 的特定子树来搜索角色/组条目。子树由 DN 指定，用于指定子树的基本节点。例如，通过将此选项设置为 ou=Group,ou=ActiveMQ,ou=system，搜索 role/group 条目仅限于 ou=Group,ou=ActiveMQ,ou=system 节点下的子树。

roleName

包含角色/组名称（如 C、O、OU 等）的角色条目的属性类型。如果省略这个选项，则角色搜索功能会有效地禁用。

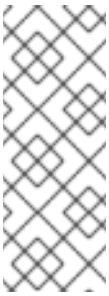
roleSearchMatching

指定一个 LDAP 搜索过滤器，它应用到 roleBase 所选的子树。详情请查看 第 5.4.1.1 节 “搜索匹配参数” 部分。

roleSearchSubtree

指定与 roleBase 指定的节点相关的角色条目的搜索深度。如果设置为 false （默认值）搜索会尝试匹配 roleBase 节点的一个子条目（映射到 javax.naming.directory.SearchControls.ONELEVEL_SCOPE）。如果为 true，它会

尝试匹配属于 roleBase 节点子树的任何条目（映射到 javax.naming.directory.SearchControls.SUBTREE_SCOPE）。



注意

Apache DS 使用 DN 路径的 OID 部分。Microsoft Active Directory 使用 CN 部分。例如，您可以在 Apache DS 中使用 DN 路径，如 oid=testuser,dc=example,dc=com，而您可能在 Microsoft Active Directory 中使用一个 DN 路径，如 cn=testuser,dc=example,dc=com。

5. 启动或重启代理（服务或进程）。

5.4.1.1. 搜索匹配参数

userSearchMatching

在传递到 LDAP 搜索操作前，此配置参数中提供的字符串值会受到字符串替换，如 java.text.MessageFormat 类所实施。

这意味着，特殊字符串 {0} 被用户名替换，从传入的客户端凭证中提取。替换后，字符串将解释为 LDAP 搜索过滤器（语法由 IETF 标准 RFC 2254 定义）。

例如，如果此选项被设置为 (uid={0})，并且收到的用户名为 jdoe，则搜索过滤器会在字符串替换后变为 (uid=jdoe)。

如果生成的搜索过滤器应用于用户 base, ou=User,ou=ActiveMQ,ou=system 选择的子树，它将与条目 uid=jdoe,ou=User,ou=ActiveMQ,ou=system.

roleSearchMatching

这的工作方式与 userSearchMatching 选项类似，但它支持两个替换字符串。

替换字符串 {0} 替换匹配用户条目的完整 DN （即，用户搜索的结果）。例如，对于用户，jdoe 可以是 uid=jdoe,ou=doe,ou=User,ou=ActiveMQ,ou=system。

替换字符串 {1} 会替换接收到的用户名。例如，jdoe.

如果此选项设为 (member=uid={1})，且接收到的用户名为 jdoe，在进行了字符串替换后，搜索过滤器变为 (member=uid=jdoe)（假设使用 ApacheDS 搜索过滤器语法）。

如果生成的搜索过滤器被应用由角色基础 (ou=Group,ou=ActiveMQ,ou=system) 所选择的子树，它将匹配具有 member 属性等于 uid=jdoe (member 属性的值是一个 DN)。

即使禁用了角色搜索，也必须始终设置此选项，因为它没有默认值。如果使用 OpenLDAP，则搜索过滤器的语法为 (member:=uid=jdoe)。

其他资源

- 有关搜索过滤器语法的简短介绍，请参阅 Oracle JNDI 指南。

5.4.2. 配置 LDAP 授权

LegacyLDAPSecuritySettingPlugin 安全设置插件读取之前在 AMQ 6 中由 LDAPAuthorizationMap 和 cachedLDAPAuthorizationMap 控制的安全信息，并在可能的情况下将这个信息转换为对应的 AMQ 7 安全设置。

AMQ 6 和 AMQ 7 中的代理的安全实现不完全匹配。因此，插件在两个版本之间执行一些转换，以实现近似的功能。

以下示例演示了如何配置插件。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。
2. 在 security-settings 元素中，添加 security-setting-plugin 元素。例如：

```
<security-settings>
   <security-setting-plugin class-
name="org.apache.activemq.artemis.core.server.impl.LegacyLDAPSecuritySettingPlugin">
      <setting name="initialContextFactory" value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <setting name="connectionURL"
value="ldap://localhost:1024"/>`ou=destinations,o=ActiveMQ,ou=system`
      <setting name="connectionUsername" value="uid=admin,ou=system"/>
      <setting name="connectionPassword" value="secret"/>
      <setting name="connectionProtocol" value="s"/>
      <setting name="authentication" value="simple"/>
   </security-setting-plugin>
</security-settings>
```

class-name

该实施是 **org.apache.activemq.artemis.core.server.impl.LegacyLDAPSecuritySettingPlugin**。

initialContextFactory

用于连接到 LDAP 的初始上下文工厂。它必须始终设置为 **com.sun.jndi.ldap.LdapCtxFactory** （即默认值）。

connectionURL

使用 LDAP URL <ldap://Host:Port> 指定 *目录服务器* 的位置。您可选择通过在目录树中添加特定节点的正斜杠 /，后跟特定节点的可分辨名称(DN)来确定此 URL。例如：ldap://ldapserver:10389/ou=system。默认值为 ldap://localhost:1024。

connectionUsername

打开与目录服务器连接的用户的 DN。例如： uid=admin,ou=system。目录服务器通常需要客户端提供用户名/密码凭证才能打开连接。

connectionPassword

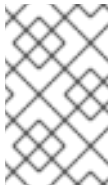
与来自 connectionUsername 的 DN 匹配的密码。在目录服务器中，在 *Directory Information Tree* (DIT)中，密码通常作为 userPassword 属性存储在对应的目录条目中。

connectionProtocol

当前未使用。未来，这个选项可能允许您为与目录服务器的连接选择安全套接字层 (SSL)。必须明确设置这个选项，因为它没有默认值。

身份验证

指定绑定到 LDAP 服务器时使用的身份验证方法。此参数的有效值为 simple （用户名和密码）或 none （匿名）。默认值为 simple。



注意

不支持 简单身份验证和安全层(SASL)身份验证。

前面配置示例中显示的其他设置有：

destinationBase

指定子对象为所有目的地提供权限的节点的 DN。在这种情况下，DN 是一个字面值（即，不会在属性值上执行字符串替换）。例如，此属性的典型值是 ou=destinations,o=ActiveMQ,ou=system。默认值为 ou=destinations,o=ActiveMQ,ou=system。

filter

指定 LDAP 搜索过滤器，用于在查找任何类型的目的地权限时使用。搜索过滤器会尝试与队列或主题节点的子子或子代之一匹配。默认值为 (cn=*)。

roleAttribute

指定节点的一个属性，它匹配其值是一个角色的 DN 的 filter。默认值为 uniqueMember。

adminPermissionValue

指定与 admin 权限匹配的值。默认值为 admin。

readPermissionValue

指定匹配 read 权限的值。默认值为 。

writePermissionValue

指定匹配 write 权限的值。默认值为 write。

enableListener

指定是否启用监听程序，自动接收 LDAP 服务器中所做的更新，并实时更新代理的授权配置。默认值为 true。

mapAdminToManage

指定是否将传统（即 AMQ 6）admin 的权限映射到 AMQ 7 manage 权限。请参阅下表中映射语义的详情。默认值为 false。

LDAP 中定义的队列或主题的名称充当安全设置的"匹配"，权限值从 AMQ 6 类型映射到 AMQ 7

类型，角色映射为 as-is。由于 LDAP 中定义的队列或主题的名称充当安全设置的匹配项，因此 security 设置可能无法应用到 JMS 目的地。这是因为 AMQ 7 始终根据需要为 JMS 目的地添加前缀 "jms.queue." 或 "jms.topic."。

AMQ 6 有三种权限类型 - 读取、写入 和管理。这些权限类型在 ActiveMQ 网站上进行了描述；安全性。

AMQ 7 有以下权限类型：

- createAddress
- deleteAddress
- createDurableQueue
- deleteDurableQueue
- createNonDurableQueue
- deleteNonDurableQueue
- send
- consume
- 管理
- browse

下表显示了安全设置插件如何将 AMQ 6 权限类型映射到 AMQ 7 权限类型：

AMQ 6 权限类型	AMQ 7 权限类型
读取	使用、浏览
write	send
admin	createAddress, deleteAddress, createDurableQueue, deleteDurableQueue, createNonDurableQueue, deleteNonDurableQueue, manage （如果 **mapAdminToManage** is set to **true**）

如下方所述，在一些情况下，插件在 AMQ 6 和 AMQ 7 权限类型间执行一些转换来实现等效性：

- **映射默认不包含 AMQ 7 管理 权限类型，因为 AMQ 6 中没有类似的权限类型。但是，如果 mapAdminToManage 设为 true，插件会将 AMQ 6 admin 权限映射到 AMQ 7 管理权限。**
- **AMQ 6 中的 admin 权限类型决定代理在目标不存在时自动创建目的地，用户是否向它发送消息。如果用户有向目的地发送消息的权限，AMQ 7 会自动允许自动创建目的地。因此，插件会将旧的 admin 权限映射到上面显示的 AMQ 7 权限。如果 mapAdminToManage 设为 true，则该插件也会将 AMQ 6 admin 权限映射到 AMQ 7 manage 权限。**

allowQueueAdminOnRead

是否要将旧的读取权限映射到 createDurableQueue、createNonDurableQueue 和 deleteDurableQueue 权限，以便 JMS 客户端能够在不需要 admin 权限的情况下创建持久和非持久订阅。这在 AMQ 6 中允许。默认值为 false。

下表显示了当 allowQueueAdminOnRead 为 true 时，安全设置插件如何将 AMQ 6 权限类型映射到 AMQ 7 权限类型：

AMQ 6 权限类型	AMQ 7 权限类型
读取	consume, browse, createDurableQueue, createNonDurableQueue, deleteDurableQueue
write	send

AMQ 6 权限类型	AMQ 7 权限类型
admin	createAddress, deleteAddress, deleteNonDurableQueue, manage （如果 **mapAdminToManage** 设为 **true**）

5.4.3. 在 login.config 文件中加密密码

因为机构经常安全地存储数据 LDAP，所以 login.config 文件可以包含代理与机构的 LDAP 服务器通信所需的配置。此配置文件通常包含用于登录到 LDAP 服务器的密码，因此需要加密此密码。

先决条件

- 确保您已修改了 login.config 文件来添加必要的属性，如 第 5.4.2 节 “配置 LDAP 授权” 所述。

流程

以下流程演示了如何屏蔽 < *broker_instance_dir*> /etc/login.config 文件中的 connectionPassword 参数的值。

1. 在命令提示符下，使用 mask 工具加密密码：

```
$ <broker_instance_dir>/bin/artemis mask <password>
result: 3a34fd21b82bf2a822fa49a8d8fa115d
```

2. 打开 & *lt;broker_instance_dir&gt*; /etc/login.config 文件。找到 connectionPassword 参数：

```
connectionPassword = <password>
```

3. 将纯文本密码替换为加密值：

```
connectionPassword = 3a34fd21b82bf2a822fa49a8d8fa115d
```

4. 使用标识符 "ENC （）" 嵌套加密值：

```
connectionPassword = "ENC(3a34fd21b82bf2a822fa49a8d8fa115d)"
```

login.config 文件现在包含屏蔽的密码。因为密码被嵌套为 "ENC （）" 标识符，所以 AMQ Broker 会在使用前对其进行解密。

其他资源

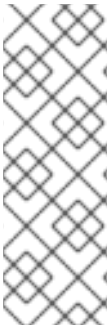
- 有关 AMQ Broker 中包含的配置文件的更多信息，请参阅 AMQ Broker 配置文件和位置。

5.4.4. 映射外部角色

您可以将来自外部身份验证供应商（如 LDAP）的角色映射到代理内部使用的角色。

要映射外部角色，请在 broker.xml 配置文件中的 security-settings 元素中创建 role-mapping 条目。例如：

```
<security-settings>
...
   <role-mapping from="cn=admins,ou=Group,ou=ActiveMQ,ou=system" to="my-admin-role"/>
   <role-mapping from="cn=users,ou=Group,ou=ActiveMQ,ou=system" to="my-user-role"/>
</security-settings>
```



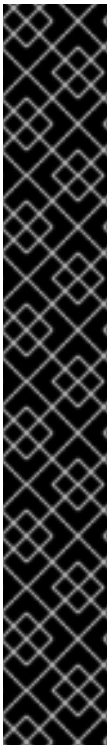
注意

- 角色映射是添加的。这意味着用户将保留原始角色以及新分配的角色。
- 角色映射仅影响授权队列访问的角色，不提供启用 Web 控制台访问的方法。

5.5. 使用 KERBEROS 进行身份验证和授权

使用 AMQP 协议发送和接收消息时，客户端可以使用 *Simple Authentication and Security Layer* (SASL)框架中的 GSSAPI 机制发送 Kerberos 安全凭证。Kerberos 凭据也可以通过将经过身份验证的用户映射到 LDAP 目录或基于文本的属性文件中配置的已分配角色来授权。

您可以使用带有 *传输层安全* (TLS)的 SASL 来保护消息传递应用程序。SASL 提供用户身份验证，TLS 提供数据完整性。



重要

- 在 AMQ Broker 能够验证和授权 Kerberos 凭证前，您必须部署和配置 Kerberos 基础架构。有关部署 Kerberos 的更多信息，请参阅您的操作系统文档。
  - 对于 RHEL 7，请参阅使用 Kerberos。
  - 对于 Windows，请参阅 Kerberos 身份验证概述。
- Oracle 或 IBM JDK 的用户应该安装 Java Cryptography Extension (JCE)。如需更多信息，请参阅 Oracle version of the JCE 或 IBM version of the JCE。

以下流程演示了如何为身份验证和授权配置 Kerberos。

5.5.1. 配置网络连接以使用 Kerberos

AMQ Broker 使用 *简单身份验证和安全层* (SASL)框架中的 GSSAPI 机制与 Kerberos 安全凭证集成。要在 AMQ Broker 中使用 Kerberos，每个接受者验证或授权使用 Kerberos 凭证的客户端都必须配置为使用 GSSAPI 机制。

以下流程演示了如何配置 acceptor 以使用 Kerberos。

先决条件

- 在 AMQ Broker 能够验证和授权 Kerberos 凭证前，您必须部署和配置 Kerberos 基础架构。

流程

1. 停止代理。
   a. 对于 Linux：

```
<broker_instance_dir>/bin/artemis stop
```

b. **在 Windows 上：**

```
<broker_instance_dir>\bin\artemis-service.exe stop
```

2. **打开 & *lt;broker_instance_dir>* /etc/broker.xml 配置文件。**

3. **将 name-value 对 saslMechanisms=GSSAPI 添加到 acceptor 的 URL 的查询字符串中。**

```
<acceptor name="amqp">
 tcp://0.0.0.0:5672?protocols=AMQP;saslMechanisms=GSSAPI
</acceptor>
```

前面的配置意味着 acceptor 在验证 Kerberos 凭证时使用 GSSAPI 机制。

4. **（可选） PLAIN 和 ANONYMOUS SASL 机制也被支持。要指定多个机制，请使用逗号分隔的列表。例如：**

```
<acceptor name="amqp">
 tcp://0.0.0.0:5672?protocols=AMQP;saslMechanisms=GSSAPI,PLAIN
</acceptor>
```

结果是一个接受者，它使用 GSSAPI 和 PLAIN SASL 机制。

5. **启动代理。**

a. **对于 Linux：**

```
<broker_instance_dir>/bin/artemis run
```

b. **在 Windows 上：**

```
<broker_instance_dir>\bin\artemis-service.exe start
```

其他资源

- 有关接收器的详情，请参考 第 2.1 节 “关于接收器”。

5.5.2. 使用 Kerberos 凭证验证客户端

AMQ Broker 支持 AMQP 连接的 Kerberos 身份验证，这些连接使用 *简单身份验证和安全层* (SASL) 框架中的 GSSAPI 机制。

代理使用 *Java 身份验证和授权服务* (JAAS)获取其 Kerberos 接受者凭证。Java 安装中包含的 JAAS 库与验证 Kerberos 凭据的登录模块 Krb5LoginModule 一起打包。有关其 Krb5LoginModule 的更多信息，请参阅 Java 供应商的文档。例如，Oracle 会提供其 Krb5LoginModule 登录模块的信息，作为其 Java 8 文档 的一部分。

先决条件

- 您必须启用 acceptor 的 GSSAPI 机制，然后才能使用 Kerberos 安全凭证验证 AMQP 连接。更多信息请参阅 第 5.5.1 节 “配置网络连接以使用 Kerberos”。

流程

1. 停止代理。

   a. 对于 Linux：

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. 在 Windows 上：

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. 打开 & *lt;broker_instance_dir>* /etc/login.config 配置文件。

3. 添加名为 amqp-sasl-gssapi 的配置范围。以下示例显示了在 JDK 的 Oracle 和 OpenJDK 版本中找到的 Krb5LoginModule 的配置。

   ```
   amqp-sasl-gssapi {
     com.sun.security.auth.module.Krb5LoginModule required
     isInitiator=false
   ```

```
    storeKey=true
    useKeyTab=true
    principal="amqp/my_broker_host@example.com"
    debug=true;
};
```

amqp-sasl-gssapi

默认情况下，代理上的 GSSAPI 机制实现使用名为 amqp-sasl-gssapi 的 JAAS 配置范围来获取其 Kerberos acceptor 凭证。

Krb5LoginModule

此版本的 Krb5LoginModule 由 JDK 的 Oracle 和 OpenJDK 版本提供。通过引用 Java 厂商的文档，验证 Krb5LoginModule 及其可用选项的完全限定类名称。

useKeyTab

Krb5LoginModule 配置为在验证主体时使用 Kerberos keytab。keytabs 使用 Kerberos 环境中的工具生成。有关生成 Kerberos keytab 的详情，请查看厂商中的文档。

主体

Principal 设置为 amqp/my_broker_host@example.com。这个值必须与 Kerberos 环境中创建的服务主体对应。有关创建服务主体的详情，请查看厂商中的文档。

4. 启动代理。

a. 对于 Linux：

```
<broker_instance_dir>/bin/artemis run
```

b. 在 Windows 上：

```
<broker_instance_dir>\bin\artemis-service.exe start
```

5.5.2.1. 使用其他配置范围

您可以通过在 AMQP 接受器的 URL 中添加参数 saslLoginConfigScope 来指定替代配置范围。在以下配置示例中，参数 saslLoginConfigScope 被赋予值 alternative-sasl-gssapi。结果是一个 acceptor，它使用名为 alternative-sasl-gssapi 的替代范围，在 < *broker_instance_dir*> /etc/login.config 配置文件中声明。

```
<acceptor name="amqp">
tcp://0.0.0.0:5672?
protocols=AMQP;saslMechanisms=GSSAPI,PLAIN;saslLoginConfigScope=alternative-sasl-gssapi`
</acceptor>
```

5.5.3. 使用 Kerberos 凭证授权客户端

AMQ Broker 包括实现 JAAS Krb5LoginModule 登录模块，以便在映射角色时供其他安全模块使用。模块将 Kerberos 验证的 Peer Principal 添加到 Subject 的主体设置为 AMQ Broker UserPrincipal。然后，可以将凭证传递给 PropertiesLoginModule 或 LDAPLoginModule 模块，该模块将 Kerberos 验证的 Peer Principal 映射到 AMQ Broker 角色。

注意

Kerberos Peer Principal 不存在，仅作为角色成员存在。

先决条件

- **您必须启用 acceptor 的 GSSAPI 机制，然后才能使用 Kerberos 安全凭证授权 AMQP 连接。**

流程

1. **停止代理。**

   a. 对于 **Linux：**

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. **在 Windows 上：**

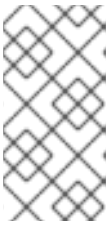
   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. **打开 & *lt;broker_instance_dir*> /etc/login.config 配置文件。**

3. 为 **AMQ Broker Krb5LoginModule 和 LDAPLoginModule 添加配置。通过引用 LDAP 供应商中的文档来验证配置选项。**

示例配置如下所示：

```
org.apache.activemq.artemis.spi.core.security.jaas.Krb5LoginModule required
    ;
org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule optional
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
    connectionURL="ldap://localhost:1024"
    authentication=GSSAPI
    saslLoginConfigScope=broker-sasl-gssapi
    connectionProtocol=s
    userBase="ou=users,dc=example,dc=com"
    userSearchMatching="(krb5PrincipalName={0})"
    userSearchSubtree=true
    authenticateUser=false
    roleBase="ou=system"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=false
    ;
```



注意

上例中显示的 **Krb5LoginModule** 版本与 **AMQ Broker** 一起分发，并将 **Kerberos** 身份转换为可供其他 **AMQ** 模块用于角色映射的代理身份。

4. 启动代理。

   a. 对于 **Linux**：

   ```
   <broker_instance_dir>/bin/artemis run
   ```

   b. 在 **Windows** 上：

   ```
   <broker_instance_dir>\bin\artemis-service.exe start
   ```

其他资源

- 有关在 **AMQ Broker** 中启用 **GSSAPI** 机制的更多信息，请参阅 第 5.5.1 节 “配置网络连接以使用 Kerberos”。
- 有关 **PropertiesLoginModule** 的更多信息，请参阅 第 5.2.2.1 节 “配置基本用户和密码身份

验证”。

- 有关 LDAPLoginModule 的更多信息，请参阅 第 5.4.1 节 “配置 LDAP 以验证客户端”。

5.6. 指定安全管理器

代理使用一个名为*安全管理器*的组件来处理身份验证和授权。

AMQ Broker 包括两个安全管理器：

- ActiveMQJAASSecurityManager 安全管理器.此安全管理器提供与 JAAS 和 Red Hat JBoss Enterprise Application Platform (JBoss EAP)安全性的集成。这是 AMQ Broker 使用的默认安全管理器。

- ActiveMQBasicSecurityManager 安全管理器。这个基本安全管理器不支持 JAAS。相反，它支持通过用户名和密码凭证进行身份验证和授权。此安全管理器支持使用管理 API 添加、删除和更新用户。所有用户和角色数据都存储在代理绑定日志中。这意味着，对 live 代理所做的任何更改都可用于备份代理。

作为包含的安全管理器的替代选择，系统管理员可能需要更多地控制代理安全实施。在这种情况下，也可以在代理配置中指定 *自定义* 安全管理器。自定义安全管理器是一个用户定义的类，它实现了 org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager5 接口。

以下子部分中的示例演示了如何将代理配置为使用：

- 基本安全管理器而不是默认的 JAAS 安全管理器

- 自定义安全管理器

5.6.1. 使用基本安全管理器

除了默认的 ActiveMQJAASSecurityManager 安全管理器外，AMQ Broker 还包括 ActiveMQBasicSecurityManager 安全管理器。

当您使用基本安全管理器时，所有用户和角色数据都存储在绑定日志中（或者绑定 *表*，如果您使用 JDBC 持久性）。因此，如果您配置了 live-backup 代理组，您在实时代理上复制的任何用户管理都会在故障切换时自动反映在备份代理中。这可避免单独管理 LDAP 服务器，这是实现此行为的替代方法。

在配置和使用基本安全管理器前，请注意以下几点：

- 基本安全管理器不像默认的 JAAS 安全管理器一样可插拔。
- 基本安全管理器不支持 JAAS。相反，它只支持通过用户名和密码凭证进行身份验证和授权。
- AMQ 管理控制台需要 JAAS。因此，如果您使用基本安全管理器并希望使用控制台，您还需要为用户和密码身份验证配置 login.config 配置文件。有关配置用户和密码身份验证的详情，请参考 第 5.2.2.1 节 “配置基本用户和密码身份验证”。
- 在 AMQ Broker 中，用户管理由代理管理 API 提供。此管理包括添加、列出、更新和删除用户和角色的功能。您可以使用 JMX、管理消息、HTTP （使用 Jolokia 或 AMQ 管理控制台）和 AMQ Broker 命令行界面来执行这些功能。由于代理直接存储此数据，因此代理必须正在运行，才能管理用户。无法手动修改绑定数据。
- 任何通过 HTTP 的管理访问（例如，使用 Jolokia 或 AMQ 管理控制台）都由控制台 JAAS 登录模块处理。通过 JConsole 或其他远程 JMX 工具的 MBean 访问是由基本安全管理器处理的。管理消息由基本安全管理器处理。

5.6.1.1. 配置基本安全管理器

以下流程演示了如何将代理配置为使用基本安全管理器。

流程

1. 打开 & *lt;broker-instance-dir*> /etc/boostrap.xml 配置文件。
2. 在 security-manager 元素中，对于 class-name 属性，指定完整的 ActiveMQBasicSecurityManager 类名称。

```
<broker xmlns="http://activemq.org/schema">
  ...
```

```
  <security-manager class-
name="org.apache.activemq.artemis.spi.core.security.ActiveMQBasicSecurityManager">
  </security-manager>
  ...
</broker>
```

3. 因为您无法手动修改包含用户和角色数据的绑定数据，并且因为代理必须运行来管理用户，因此建议在第一次引导时保护代理。要达到此目的，请定义一个 ***bootstrap 用户***，然后使用其凭证来添加其他用户。

   在 **security-manager** 元素中，添加 **bootstrapUser**、**bootstrapPassword** 和 **bootstrapRole** 属性，并指定值。例如：

```
<broker xmlns="http://activemq.org/schema">
  ...
  <security-manager class-
name="org.apache.activemq.artemis.spi.core.security.ActiveMQBasicSecurityManager">
      <property key="bootstrapUser" value="myUser"/>
      <property key="bootstrapPassword" value="myPass"/>
      <property key="bootstrapRole" value="myRole"/>
  </security-manager>
  ...
</broker>
```

bootstrapUser

bootstrap 用户的名称。

bootstrapPassword

boostrap 用户的 Passsword。您还可以指定加密的密码。

bootstrapRole

boostrap 用户的角色。

注意

如果您在配置中为 **bootstrap** 用户定义了前面的属性，则每次启动代理时都会设置这些凭证，无论您在代理运行时所做的任何更改。

4. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

5.

在 broker.xml 配置文件中，找到为 activemq.management# 地址匹配默认定义的 address-setting 元素。这些默认地址设置如下所示。

```
<address-setting match="activemq.management#">
   <dead-letter-address>DLQ</dead-letter-address>
   <expiry-address>ExpiryQueue</expiry-address>
   <redelivery-delay>0</redelivery-delay>
   <!--...-->
   <max-size-bytes>-1</max-size-bytes>
   <message-counter-history-day-limit>10</message-counter-history-day-limit>
   <address-full-policy>PAGE</address-full-policy>
   <auto-create-queues>true</auto-create-queues>
   <auto-create-addresses>true</auto-create-addresses>
   <auto-create-jms-queues>true</auto-create-jms-queues>
   <auto-create-jms-topics>true</auto-create-jms-topics>
</address-setting>
```

6. 在为 activemq.management# 地址匹配的地址设置中，对于您在此流程前面指定的 bootstrap 角色名称，添加以下所需的权限：

- createNonDurableQueue
- createAddress
- consume
- 管理
- send

例如：

```
<address-setting match="activemq.management#">
   ...
   <permission type="createNonDurableQueue" roles="myRole"/>
   <permission type="createAddress" roles="myRole"/>
   <permission type="consume" roles="myRole"/>
   <permission type="manage" roles="myRole"/>
   <permission type="send" roles="myRole"/>
</address-setting>
```

其他资源

- 有关 ActiveMQBasicSecurityManager 类的更多信息，请参阅 ActiveMQ Artemis Core API 文档中的 Class ActiveMQBasicSecurityManager。

- 要了解如何在配置文件中加密密码，请参阅 第 5.9 节 “在配置文件中加密密码”。

5.6.2. 指定自定义安全管理器

以下流程演示了如何在代理配置中指定自定义安全管理器。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/boostrap.xml 配置文件。

2. 在 security-manager 元素中，对于 class-name 属性，请指定作为 org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager5 接口的用户定义的实施的类。例如：

```
<broker xmlns="http://activemq.org/schema">
  ...
  <security-manager class-name="com.myclass.MySecurityManager">
    <property key="myKey1" value="myValue1"/>
    <property key="myKey2" value="myValue2"/>
  </security-manager>
  ...
</broker>
```

其他资源

- 有关 ActiveMQSecurityManager5 接口的更多信息，请参阅 ActiveMQ Artemis Core API 文档中的 Interface ActiveMQSecurityManager5。

5.6.3. 运行自定义安全管理器示例程序

AMQ Broker 有一个示例程序，演示了如何实现自定义安全管理器。在示例中，自定义安全管理器会记录身份验证和授权的详细信息，然后将详细信息传递给 ActiveMQJAASSecurityManager 的实例（即默认安全管理器）。

以下流程演示了如何运行自定义安全管理器示例程序。

先决条件

- 您的机器被设置为运行 AMQ Broker 示例程序。如需更多信息，请参阅 运行 AMQ Broker 示例。

下载了 自定义安全管理器示例。

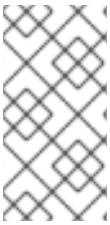
流程

1. 导航到包含自定义安全管理器示例的目录。以下示例假设您将示例下载到名为 amq-broker-examples 的目录。

```
$ cd amq-broker-examples/examples/features/standard/security-manager
```

2. 运行示例。

```
$ mvn verify
```



注意

如果您希望在运行示例程序时手动创建并启动代理实例，请将上一步中的命令替换为 mvn -PnoServer 验证。

其他资源

- 有关 ActiveMQJAASSecurityManager 类的更多信息，请参阅 ActiveMQ Artemis Core API 文档中的 Class ActiveMQJAASSecurityManager 部分。

5.7. 禁用安全性

默认启用 安全性。以下流程演示了如何禁用代理安全性。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 在 core 元素中，将 security-enabled 的值设置为 false。

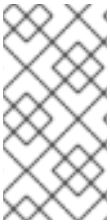
```
<security-enabled>false</security-enabled>
```

3. 如有必要，为 security-invalidation-interval 指定一个新值（以毫秒为单位）。此属性的值指定代理何时定期使安全登录无效。默认值为 10000。

5.8. 跟踪来自验证的用户的消息

要启用跟踪和记录消息的来源（例如，用于安全审核目的），您可以使用 _AMQ_VALIDATED_USER 消息键。

在 broker.xml 配置文件中，如果 populate-validated-user 选项被设置为 true，则代理将使用 _AMQ_VALIDATED_USER 键将验证的用户的名称添加到消息中。对于 JMS 和 STOMP 客户端，此消息密钥映射到 JMSXUserID 密钥。



注意

代理无法将验证的用户名添加到 AMQP JMS 客户端生成的消息中。在客户端发送 AMQP 消息后修改 AMQP 消息的属性是 AMQP 协议的违反。

对于基于其/一致性 SSL 证书验证的用户，代理填充的验证用户名是证书的可辨识名称(DN)映射的名称。

在 broker.xml 配置文件中，如果 security-enabled 为 false，并且 populate-validated-user 为 true，则代理会填充客户端提供的任何用户名（若有）。默认情况下，populate-validated-user 选项为 false。

您可以将代理配置为拒绝在发送消息时由客户端填充的用户名（即 JMSXUserID 密钥）的消息。您可能会发现此选项对 AMQP 客户端很有用，因为代理无法为这些客户端发送的消息填充验证的用户名本身。

要将代理配置为拒绝客户端设置没有 JMSXUserID 的信息，请在 broker.xml 配置文件中添加以下配置：

```
<reject-empty-validated-user>true</reject-empty-validated-user>
```

默认情况下，**reject-empty-validated-user** 设置为 **false**。

5.9. 在配置文件中加密密码

默认情况下，**AMQ Broker** 将所有密码保存在配置文件中，以纯文本形式保存。确保保护具有正确权限的所有配置文件，以防止未经授权的访问。您还可以加密或 *屏蔽* 纯文本密码，以防止不需要的查看器读取它们。

5.9.1. 关于加密密码

已加密或 *屏蔽* 的，密码是纯文本密码的加密版本。加密的版本由 **AMQ Broker** 提供的 **mask** 命令行工具生成。有关 **mask** 工具程序的更多信息，请参阅命令行帮助文档：

```
$ <broker_instance_dir>/bin/artemis help mask
```

要屏蔽密码，请将其纯文本值替换为加密的值。屏蔽的密码必须以标识符 **ENC ()** 包装，以便在需要实际值时解密。

在以下示例中，配置文件 **< *broker_instance_dir*>/etc/bootstrap.xml** 包含 **keyStorePassword** 和 **trustStorePassword** 参数的屏蔽密码。

```
<web bind="https://localhost:8443" path="web"
    keyStorePassword="ENC(-342e71445830a32f95220e791dd51e82)"
    trustStorePassword="ENC(32f94e9a68c45d89d962ee7dc68cb9d1)">
   <app url="activemq-branding" war="activemq-branding.war"/>
</web>
```

您可以在以下配置文件中使用屏蔽的密码。

- **broker.xml**
- **bootstrap.xml**

- management.xml
- artemis-users.properties
- login.config （用于 LDAPLoginModule）

配置文件位于 < *broker_instance_dir*> /etc。

注意

artemis-users.properties 仅支持经过哈希的屏蔽密码。在创建代理时创建用户时，artemis-users.properties 默认包含哈希密码。默认 PropertiesLoginModule 不会解码 artemis-users.properties 文件中的密码，而是哈希输入并比较密码验证的两个散列值。将散列密码改为屏蔽的密码不允许访问 AMQ Broker 管理控制台。

broker.xml、bootstrap.xml、management.xml 和 login.config 支持已屏蔽但未被哈希的密码。

5.9.2. 在配置文件中加密密码

以下示例演示了如何在 broker.xml 配置文件中屏蔽 cluster-password 的值。

流程

1. 在命令提示符下，使用 mask 工具加密密码：

```
$ <broker_instance_dir>/bin/artemis mask <password>
```

```
result: 3a34fd21b82bf2a822fa49a8d8fa115d
```

2. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件，包含您要屏蔽的纯文本密码：

```
<cluster-password>
  <password>
</cluster-password>
```

3. 将纯文本密码替换为加密值：

```
<cluster-password>
 3a34fd21b82bf2a822fa49a8d8fa115d
</cluster-password>
```

4. 使用标识符 ENC （） 嵌套加密值：

```
<cluster-password>
 ENC(3a34fd21b82bf2a822fa49a8d8fa115d)
</cluster-password>
```

配置文件现在包含加密的密码。因为密码被嵌套为 ENC （） 标识符，所以 AMQ Broker 会在使用前对其进行解密。

其他资源

- 有关 AMQ Broker 中包含的配置文件的详情，请参考 第 1.1 节 “AMQ Broker 配置文件和位置”。

5.9.3. 设置 codec 密钥以加密和解密密码

需要使用 codec 来加密和解密密码。如果没有配置自定义 codec，mask 工具使用默认 codec 来加密密码，AMQ Broker 会使用相同的默认 codec 来解密密码。codec 配置了一个默认密钥，它提供给底层加密算法来加密和解密密码。使用默认密钥存在风险，因为恶意者可能会使用该密钥来解密您的密码。

当您使用 mask 工具加密密码时，您可以指定您自己的密钥字符串以避免使用默认的 codec 密钥。然后，您必须在 ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY 环境变量中设置相同的密钥字符串，以便代理可以解密密码。在环境变量中设置密钥可以更加安全，因为它不会在配置文件中保留。另外，您可以在启动代理前立即设置密钥，并在代理启动后立即取消设置它。

流程

1. 使用 mask 实用程序加密配置文件中的每个密码。对于 key 参数，指定用于加密密码的字符串。使用相同的密钥字符串加密每个密码。

```
$ <broker_instance_dir>/bin/artemis mask --key <key> <password>
```

警告

确保保留了您在运行 mask 工具以加密密码时指定的密钥字符串记录。您必须在环境变量中配置相同的键值，以允许代理解密密码。

有关在配置文件中加密密码的详情请参考 第 5.9.2 节 “在配置文件中加密密码”。

2. 从命令提示符，将 ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY 环境变量设置为您在加密每个密码时指定的密钥字符串。

```
$ export ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY= <key>
```

3. 启动代理。

```
$ ./artemis run
```

4. 取消设置 ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY 环境变量。

```
$ unset ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY
```

注意

如果在启动代理后取消设置 ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY 环境变量，则必须在每次启动代理前将其再次设置为相同的密钥字符串。

5.10. 配置身份验证和授权缓存

默认情况下，AMQ Broker 将成功身份验证和授权响应的信息存储在单独的缓存中。您可以更改每个缓存中允许的默认条目数，以及缓存条目的持续时间。

1. 打开 & lt;broker-instance-dir>/etc/broker.xml 配置文件。

2. 要更改默认条目数 1000 个，在每个缓存中允许，设置 **authentication-cache-size** 和 **authorization-cache-size** 参数。例如：

```
<configuration>
 ...
 <core>
  ...
  <authentication-cache-size>2000</authentication-cache-size>
  <authorization-cache-size>1500</authorization-cache-size>
  ...
 </core>
 ...
</configuration>
```

注意

如果缓存达到限制集，则最早使用的条目将从缓存中移除。

3. 要更改默认持续时间 10000 毫秒，哪个条目被缓存，请设置 **security-invalidation-interval** 参数。例如：

```
<configuration>
 ...
 <core>
  ...
  <security-invalidation-interval>20000</security-invalidation-interval>
  ...
 </core>
 ...
</configuration>
```

注意

如果将 **security-invalidation-interval** 参数设置为 **0**，则禁用身份验证和授权缓存。

第 6 章 持久性消息数据

对于持久性（ *storing*）消息数据，AMQ Broker 有两个选项。

在日志中保留消息

这是默认选项。基于日志的持久性是一种高性能选项，可将消息写入文件系统上的日志。

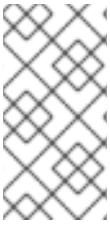
在数据库中保留消息

此选项使用 *Java 数据库连接* (JDBC)连接将消息保留至您选择的数据库。

另外，您还可以将代理配置为 不保留 任何消息数据。更多信息请参阅 第 6.3 节 “禁用持久性”。

代理使用不同的解决方案来持久保留消息日志外的大型消息。请参阅 第 8 章 *处理大型消息* 了解更多信息。

代理也可以配置为在低内存情况下将消息页面到磁盘。请参阅 第 7.1 节 “配置消息分页” 了解更多信息。



注意

有关 AMQ Broker 支持哪些数据库和网络文件系统的当前信息，请参阅红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

6.1. 在日志中保留消息数据

代理日志是磁盘上 仅附加文件 的集合。每个文件都预先创建为固定大小，最初使用 padding 填充。当消息传递操作在代理上执行时，记录会附加到日志的末尾。附加记录允许代理最小化磁盘头移动和随机访问操作，这通常是磁盘上的最慢的操作。当一个日志文件已满时，代理会创建一个新日志文件。

日志文件大小可以配置，从而尽量减少每个文件所使用的磁盘 cylinders 的数量。但是，现代磁盘拓扑拓扑比较复杂，代理无法控制文件映射到哪个 cylinder。因此，无法精确地控制日志文件大小。

代理使用的其他与持久性相关的功能有：

-

确定特定日志文件仍在使用的 *垃圾回收* 算法。如果日志文件不再使用，代理可以回收文件以供重复使用。

- 从日志中删除死空间并压缩数据的压缩算法。这会在磁盘上使用较少的文件生成日志。
- 支持本地事务。
- 在使用 JMS 客户端时支持扩展架构(XA)事务。

大多数日志都是以 Java 编写的。但是，与实际文件系统交互被抽象化，因此您可以使用不同的可插拔实现。AMQ Broker 包括以下实现：

NIO

NIO （新 I/O）使用标准 Java NIO 与文件系统进行接口。这提供了极佳的性能，并在具有 Java 6 或更高版本运行时的任何平台上运行。有关 Java NIO 的更多信息，请参阅 Java NIO。

AIO

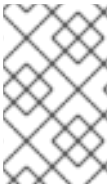
AIO (Aynshcronous I/O)使用精简原生打包程序与 Linux 异步 I/O 库(libaio)进行通信。使用 AIO 时，代理在将数据提供给磁盘后会重新调用，从而避免显式同步。默认情况下，代理会尝试使用 AIO 日志，如果 AIO 不可用，则回退到使用 NIO。

与 Java NIO 相比，AIO 通常提供更好的性能。要了解如何安装 libaio，请参阅 第 6.1.1 节 “安装 Linux 异步 I/O 库”。

以下子部分中的流程演示了如何为基于日志的持久性配置代理。

6.1.1. 安装 Linux 异步 I/O 库

红帽建议使用 AIO 日志（而不是 NIO）以获得更好的持久性性能。



注意

无法与其他操作系统或早期版本的 Linux 内核一起使用 AIO 日志。

要使用 AIO 日志，您必须安装 Linux Asynchronous I/O 库(libaio)。要安装 libaio，请使用 yum 命令，如下所示：

```
yum install libaio
```

6.1.2. 配置基于日志的持久性

以下流程描述了如何查看代理用于基于日志的持久性的默认配置。您可以根据需要使用此描述来调整配置。

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

   默认情况下，代理配置为使用基于日志的持久性，如下所示。

```
<configuration>
 <core>
  ...
  <persistence-enabled>true</persistence-enabled>
  <journal-type>ASYNCIO</journal-type>
  <bindings-directory>./data/bindings</bindings-directory>
  <journal-directory>./data/journal</journal-directory>
  <journal-datasync>true</journal-datasync>
  <journal-min-files>2</journal-min-files>
  <journal-pool-files>-1</journal-pool-files>
  <journal-device-block-size>4096</journal-device-block-size>
  <journal-file-size>10M</journal-file-size>
  <journal-buffer-timeout>12000</journal-buffer-timeout>
  <journal-max-io>4096</journal-max-io>
  ...
 </core>
</configuration>
```

persistence-enabled

如果此参数的值设为 true，则代理将使用基于文件的日志进行消息持久性。

journal-type

要使用的日志类型。如果设置为 ASYNCIO，代理会首先尝试使用 AIO。如果没有找到 AIO，代理将使用 NIO。

bindings-directory

绑定日志的文件系统位置。默认值相对于 < *broker_instance_dir*> 目录。

journal-directory

消息日志的文件系统位置。默认值相对于 *< broker_instance_dir>* 目录。

journal-datasync

如果此参数的值设为 **true**，代理将使用 **fdatasync** 函数来确认磁盘写入。

journal-min-files

代理启动时初始创建的日志文件数量。

journal-pool-files

回收未使用文件后要保留的文件数。默认值 **-1** 表示在清理过程中不会删除任何文件。

journal-device-block-size

存储设备上日志使用的最大数据块大小（以字节为单位）。默认值为 **4096** 字节。

journal-file-size

指定日志目录中每个日志文件的最大大小（以字节为单位）。当达到这个限制时，代理会启动一个新文件。这个参数还支持字节表示法（如 **K**、**M**、**G**）或二进制等效的(**Ki**、**Mi**、**Gi**)。如果您的配置中没有明确指定此参数，则默认值为 **10485760** 字节(**10MiB**)。

journal-buffer-timeout

指定代理清除日志缓冲区的频率（以纳秒为单位）。**AIO** 通常使用高于 **NIO** 的 **flush** 率，因此代理为 **NIO** 和 **AIO** 维护不同的值。如果未在配置中明确指定此参数，则 **NIO** 的默认值是 **3333333** 纳秒（每秒 **300** 次）。**AIO** 的默认值为 **50000** 纳秒（即每秒 **2000** 次）。

journal-max-io

任意一个时间可在 **IO** 队列中的最大写入请求数。如果队列已满，代理会阻止进一步写入，直到空间可用为止。

如果您使用 **NIO**，则该值应始终为 **1**。如果您使用 **AIO** 和此参数没有在配置中明确指定，则默认值为 **500**。

2. 根据前面的描述，根据您的存储设备需要调整持久性配置。

其他资源

- 要了解可用于配置基于日志的持久性的所有参数，请参阅 附录 E, *消息传递日志配置元素*。

6.1.3. 关于绑定日志

绑定日志用于存储与绑定相关的数据，如在代理上部署的队列集合及其属性。它还存储数据，如 ID 序列计数器。

绑定日志始终使用 NIO，因为它与消息日志相比通常是低吞吐量。此日志上的文件的前缀为 activemq-bindings。每个文件的扩展也具有 .bindings，默认大小为 1048576 字节。

要配置绑定日志，请在 < *broker_instance_dir&gt*; /etc/broker.xml 配置文件的 core 元素中包含以下参数。

bindings-directory

绑定日志的目录。默认值为 < *broker_instance_dir*>/data/bindings。

create-bindings-dir

如果此参数的值设为 true，则代理会在 bindings-directory 中指定的位置自动创建绑定目录（如果不存在）。默认值为 true。

6.1.4. 关于 JMS 日志

JMS 日志存储所有与 JMS 相关的数据，包括 JMS 队列、主题和连接工厂，以及这些资源的任何 JNDI 绑定。通过管理 API 创建的任何 JMS 资源都保留到此日志中，但通过配置文件配置的任何资源都不是。只有在使用 JMS 时，代理才会创建 JMS 日志。

JMS 日志中的文件的前缀为 activemq-jms。每个文件的扩展也具有 .jms 扩展名，默认大小为 1048576 字节。

JMS 日志与绑定日志共享其配置。

其他资源

- 有关绑定日志的更多信息，请参阅 第 6.1.3 节 “关于绑定日志”。

6.1.5. 配置日志保留

您可以将 AMQ Broker 配置为保留创建的每个日志文件的副本。配置日志保留后，您可以重播日志文件副本中的消息，以将消息发送到代理。

6.1.5.1. 配置日志保留

您可以将 AMQ Broker 配置为在特定时间段内保留日志文件副本，或直到达到存储限制或两者为止。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 在 core 元素中，添加 journal-retention-directory 属性。指定 period 或 storage-limit（或两者）来控制日志文件的保留。另外，指定日志文件副本的文件系统位置。以下示例将 AMQ Broker 配置为将日志文件副本保存在 7 天的 data/retention 目录中，或直到文件使用 10GB 存储为止。

```
<configuration>
  <core>
    ...
    <journal-retention-directory period="7" unit="DAYS" storage-limit="10G">data/retention</journal-retention-directory>
    ...
  </core>
</configuration>
```

周期

保留日志文件副本的时间周期。当时间段过期时，AMQ Broker 会删除任何比指定时间段旧的文件。

unit

应用到保留周期的测量单位。默认值为 DAYS。其他有效的值为 HOURS、MINUTES 和 SECONDS。

目录

日志文件副本的文件系统位置。指定的目录相对于 < *broker_instance_dir*> 目录。

storage-limit

所有日志文件副本可以使用的最大存储。如果达到存储限制，代理会删除最旧的日志文件，以便为新的日志文件复制提供空间。设置存储限制是确保日志文件保留不会导致代理耗尽磁盘空间并关闭

的有效方法。

6.1.5.2. 在日志文件副本中重播代理存在的地址

如果在 AMQ Broker 上已存在您要从日志文件副本重新执行的信息地址，请使用以下步骤重新显示消息。您可以将消息重播到代理上的原始地址或不同的地址。

流程

1. 登录到 AMQ 管理控制台。如需更多信息，请参阅 访问 AMQ 管理控制台。
2. 在主菜单中，单击 **Artemis**。
3. 在文件夹树中，单击 **address** 以显示 地址列表。
4. 点 **Addresses** 选项卡。
5. 在您要重播消息的地址的 **Action** 列中，单击 **operations**。
6. 选择 **replay** 操作。
   - 如果您希望 **replay** 操作搜索要在所有日志文件副本中重播的消息，请单击 **replay (String,String)** 操作。
   - 如果您希望 **replay** 操作搜索仅在在特定时间段内创建的日志文件副本中重播的消息，请选择 **replay (String,String, String,String)** 操作。在 **startScanDate** 和 **endScanDate** 字段中，指定时间段。
7. 指定 **replay** 选项。
   a. 在 **target** 字段中，指定要发送重播消息的代理上的地址。如果将此字段留空，则会将消息重新显示到代理上的原始地址。
   b.

（可选）在 filter 字段中，指定一个字符串来仅重播与过滤器字符串匹配的消息。例如，如果消息具有 storeID 属性，您可以使用 storeID="1000" 过滤器来重新执行存储 ID 值为 1000 的所有消息。如果您没有指定过滤器，则扫描的日志文件副本中的所有消息都会重新显示到 AMQ Broker。

8. 点 Execute。

其他资源

- 有关使用 AMQ 管理控制台的更多信息，请参阅使用 AMQ 管理控制台。

6.1.5.3. 在日志文件复制中重播从代理中删除的地址

如果要从日志文件副本中重新执行的信息地址已从 AMQ Broker 中删除，请使用以下步骤将信息重新提交到代理上的不同地址。

流程

1. 登录到 AMQ 管理控制台。如需更多信息，请参阅 访问 AMQ 管理控制台。
2. 在主菜单中，单击 Artemis。
3. 在文件夹树中，单击顶级服务器。
4. 单击 Operations 选项卡。
5. 选择 replay 操作。

- 如果您希望 replay 操作搜索要在所有日志文件副本中重播的消息，请单击 replay (String,String,String) 操作。
- 如果您希望 replay 操作搜索仅在在特定时间段内创建的日志文件副本中重播的消息，请选择 replay (String,String, String,String) 操作。在 startScanDate 和 endScanDate 字段中，指定时间段。

6. 指定 replay 选项。

   a. 在 address 字段中，指定要重播的消息地址。

   b. 在 target 字段中，指定要发送重播消息的代理上的地址。

   c. （可选）在 filter 字段中，指定一个字符串来仅重播与过滤器字符串匹配的消息。例如，如果消息具有 storeID 属性，您可以使用 storeID="1000" 过滤器来重新执行存储 ID 值为 1000 的所有消息。如果您没有指定过滤器，则扫描的日志文件副本中的所有消息都会重新显示到 AMQ Broker。

7. 点 Execute。

其他资源

- 有关使用 AMQ 管理控制台的更多信息，请参阅使用 AMQ 管理控制台。

6.1.6. 紧凑日志文件

AMQ Broker 包含一个压缩算法，它从日志中删除死空间并压缩数据，以便其占用较少的磁盘空间。

以下子部分演示了如何：

- 将代理配置为在满足特定条件时自动压缩日志文件
- 使用命令行界面手动运行压缩过程

6.1.6.1. 配置日志文件压缩

代理使用以下条件来确定何时开始压缩：

- 为日志创建的文件数量。

- 日志文件中实时数据的百分比。

达到这两个条件配置的值后，压缩过程会解析日志并删除所有死记录。因此，日志由较少的文件组成。

以下流程演示了如何为日志文件压缩配置代理。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 在 core 元素中，添加 journal-compact-min-files 和 journal-compact-percentage 参数并指定值。例如：

```
<configuration>
 <core>
  ...
  <journal-compact-min-files>15</journal-compact-min-files>
  <journal-compact-percentage>25</journal-compact-percentage>
  ...
 </core>
</configuration>
```

journal-compact-min-files

代理必须在压缩开始前创建的最小日志文件数量。默认值为 10。将值设为 0 可禁用压缩。在禁用压缩时应小心，因为日志的大小可能会无限期地增长。

journal-compact-percentage

日志文件中实时数据的百分比。当少于这个百分比时（也达到 journal-compact-min-files 的配置值）时，压缩开始。默认值为 30。

6.1.6.2. 使用命令行界面运行压缩

以下流程演示了如何使用命令行界面(CLI)来压缩日志文件。

流程

1. 作为 < *broker_instance_dir*> 目录的所有者，停止代理。以下示例显示了用户 amq-

broker。

```
su - amq-broker
cd <broker_instance_dir>/bin
$ ./artemis stop
```

2. （可选） 运行以下 CLI 命令以获取数据工具的完整参数列表。默认情况下，该工具使用 < *broker_instance_dir*> /etc/broker.xml 中的设置。

```
$ ./artemis help data compact.
```

3. 运行以下 CLI 命令以压缩数据。

```
$ ./artemis data compact.
```

4. 当工具成功压缩数据后，重启代理。

```
$ ./artemis run
```

其他资源

- AMQ Broker 包括多个用于管理日志文件的 CLI 命令。如需更多信息，请参阅附录中的命令行工具。

6.1.7. 禁用磁盘写入缓存

大多数磁盘都包含硬件写缓存。写入缓存可以提高磁盘的明显性能，因为稍后写入磁盘。默认情况下，很多系统都附带启用了磁盘写入缓存。这意味着，即使在从操作系统同步后，也无法保证数据实际提供给磁盘。因此，如果发生故障，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缓存，在出现故障时不一定丢失数据，但您应该测试它们。如果您的磁盘没有这样的功能，您应该确保禁用写缓存。请注意，禁用磁盘写入缓存可能会对性能造成负面影响。

以下流程演示了如何在 Windows 上的 Linux 上禁用磁盘写入缓存。

流程

1.

在 Linux 上，要管理磁盘写入缓存设置，请使用工具 hdparm （用于 IDE 磁盘）或 sdparm 或 sginfo （用于 SDSI/SATA 磁盘）。

2. 在 Windows 上，要管理磁盘写器缓存设置，请右键单击磁盘。选择 属性。

6.2. 在数据库中持久保留消息数据

当您在数据库中持久保留消息数据时，代理使用 *Java 数据库连接* (JDBC)连接将消息和绑定数据存储在数据库表中。表中的数据使用 AMQ Broker 日志编码进行编码。有关支持的数据库的详情，请参考红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

重要

管理员可以选择根据组织更广泛的 IT 基础架构要求将消息数据存储在数据库中。但是，使用数据库可能会对消息传递系统性能造成负面影响。具体来说，通过 JDBC 将消息传递数据写入数据库表可为代理创建显著的性能开销。

6.2.1. 配置 JDBC 持久性

以下流程演示了如何配置代理，以在数据库表中存储消息和绑定数据。

流程

1. 将适当的 JDBC 客户端库添加到代理运行时。要做到这一点，将相关的 .JAR 文件添加到 *<broker_instance_dir>*/lib 目录中。

   如果要将 .JAR 文件添加到不同的目录中，您必须将该目录添加到 Java 类路径中。如需更多信息，请参阅 第 1.5 节 “扩展 JAVA 类路径”

2. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

3. 在 core 元素中，添加一个包含 database-store 元素的 store 元素。

```
<configuration>
 <core>
  <store>
    <database-store>
    </database-store>
```

```
    </store>
  </core>
</configuration>
```

4. 在 database-store 元素中，为 JDBC 持久性添加配置参数并指定值。例如：

```
<configuration>
  <core>
    <store>
      <database-store>
        <jdbc-connection-url>jdbc:oracle:data/oracle/database-store;create=true</jdbc-
connection-url>
        <jdbc-user>ENC(5493dd76567ee5ec269d11823973462f)</jdbc-user>
        <jdbc-password>ENC(56a0db3b71043054269d11823973462f)</jdbc-password>
        <bindings-table-name>BIND_TABLE</bindings-table-name>
        <message-table-name>MSG_TABLE</message-table-name>
        <large-message-table-name>LGE_TABLE</large-message-table-name>
        <page-store-table-name>PAGE_TABLE</page-store-table-name>
        <node-manager-store-table-name>NODE_TABLE</node-manager-store-table-name>
        <jdbc-driver-class-name>oracle.jdbc.driver.OracleDriver</jdbc-driver-class-name>
        <jdbc-network-timeout>10000</jdbc-network-timeout>
        <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
        <jdbc-lock-expiration>20000</jdbc-lock-expiration>
        <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
        <jdbc-max-page-size-bytes>100K</jdbc-max-page-size-bytes>
      </database-store>
    </store>
  </core>
</configuration>
```

jdbc-connection-url

您的数据库服务器的完整 JDBC 连接 URL。连接 URL 应该包含所有配置参数和数据库名称。

jdbc-user

您的数据库服务器的加密用户名。有关加密在配置文件中使用的用户名和密码的详情请参考 第 5.9 节 “在配置文件中加密密码”。

jdbc-password

您的数据库服务器的加密密码。有关加密在配置文件中使用的用户名和密码的详情请参考 第 5.9 节 “在配置文件中加密密码”。

bindings-table-name

保存绑定数据的表的名称。指定表名称可让您在多个服务器间共享单个数据库，而无需干扰。

message-table-name

保存消息数据的表名称。指定此表名称可让您在多个服务器间共享单个数据库，而无需干扰。

large-message-table-name

保留大量消息和相关数据的表名称。另外，如果客户端在块中流出一个大消息，则块会存储在此表中。指定此表名称可让您在多个服务器间共享单个数据库，而无需干扰。

page-store-table-name

保存用于存储目录信息的表名称。指定此表名称可让您在多个服务器间共享单个数据库，而无需干扰。

node-manager-store-table-name

共享存储高可用性(HA)为实时和备份代理锁定的表名称，其他与 HA 相关的数据存储在代理服务器上。指定此表名称可让您在多个服务器间共享单个数据库，而无需干扰。使用共享存储 HA 的每个实时备份对都必须使用相同的表名称。您不能在多个（和不相关的）实时备份对间共享相同的表。

jdbc-driver-class-name

JDBC 数据库驱动程序的完全限定类名称。有关支持的数据库的详情，请参考红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

jdbc-network-timeout

JDBC 网络连接超时，以毫秒为单位。默认值为 20000 毫秒。将 JDBC 用于共享存储 HA 时，建议将超时设置为小于或等于 jdbc-lock-expiration 的值。

jdbc-lock-renew-period

当前 JDBC 锁的续订周期的长度，以毫秒为单位。当这个时间过后，代理可以续订锁定。建议设置一个值，它比 jdbc-lock-expiration 的值小几次。这为代理有足够的时间来扩展租期，并提供了代理时间在连接问题时尝试续订锁定。默认值为 2000 毫秒。

jdbc-lock-expiration

以毫秒为单位，当前 JDBC 锁定被视为拥有的时间（即获取或续订），即使 jdbc-lock-renew-period 的值已过。

代理定期尝试根据 jdbc-lock-renew-period 的值续订其拥有的锁定。如果代理 无法 续订锁定（例如，因为连接问题），代理会不断尝试续订锁定，直到 jdbc-lock-expiration 的值因为锁定上次成功获取或续订。

上述续订行为的例外是另一个代理获取锁定时。如果数据库管理系统(DBMS)和代理之间有时间不对齐，或者对于垃圾回收有长时间暂停，则会出现这种情况。在这种情况下，最初

拥有的锁定的代理会考虑丢失锁定，且不会尝试续订它。

过期时间后，如果当前拥有的代理没有续订 JDBC 锁定，则另一个代理可以建立 JDBC 锁定。

jdbc-lock-expiration 的默认值为 20000 毫秒。

jdbc-journal-sync-period

持续时间，以毫秒为单位，代理日志与 JDBC 同步。默认值为 5 毫秒。

jdbc-max-page-size-bytes

当 AMQ Broker 将消息保留到 JDBC 数据库时，每个页文件的最大大小（以字节为单位）。默认值为 102400，其值为 100KB。您指定的值也支持字节表示法，如 "K" "MB" 和 "GB"。

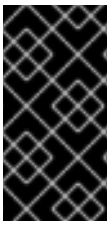
6.2.2. 配置 JDBC 连接池

如果您已经为 JDBC 持久性配置了代理，代理使用 JDBC 连接将消息和绑定数据存储在数据库表中。

如果 JDBC 连接失败，并且只要出现故障时没有活动连接活动（如数据库读取或写入），代理会继续运行，并尝试重新建立数据库连接。为达到此目的，AMQ Broker 使用 JDBC *连接池*。

通常，*连接池* 提供一组与指定数据库（可在多个应用程序之间共享）的开放连接。对于代理，如果代理和数据库之间的连接失败，代理会尝试使用与池不同的连接重新连接到数据库。池会在代理接收前测试新的连接。

以下示例演示了如何配置 JDBC 连接池。



重要

如果您没有显式配置 JDBC 连接池，代理将使用与默认配置的连接池。默认配置使用您现有 JDBC 配置中的值。如需更多信息，请参阅 默认连接池配置。

先决条件

- 本例基于示例构建，用于配置 JDBC 持久性。请查看 第 6.2.1 节 “配置 JDBC 持久性”

- 要启用连接池，AMQ Broker 使用 Apache Commons DBCP 软件包。在为代理配置 JDBC 连接池前，您应该熟悉此软件包提供的内容。如需更多信息，请参阅：

  - Apache Commons DBCP 概述

  - Apache Commons DBCP 配置参数

流程

1. 打开 & *lt;broker-instance-dir*> /etc/broker.xml 配置文件。

2. 在您之前为 JDBC 配置添加的 database-store 元素中，删除 jdbc-driver-class-name、jdbc-connection-url、jdbc-user、jdbc-password、参数。在此过程中，您要将它们替换为对应的 DBCP 配置参数。

注意

如果您没有显式删除上述参数，则您稍后添加的对应 DBCP 参数具有优先权。

3. 在 database-store 元素中，添加一个 data-source-properties 元素。例如：

```
<store>
   <database-store>
      <data-source-properties>
      </data-source-properties>
      <bindings-table-name>BINDINGS</bindings-table-name>
      <message-table-name>MESSAGES</message-table-name>
      <large-message-table-name>LARGE_MESSAGES</large-message-table-name>
      <page-store-table-name>PAGE_STORE</page-store-table-name>
      <node-manager-store-table-name>NODE_MANAGER_STORE</node-manager-store-
table-name>
      <jdbc-network-timeout>10000</jdbc-network-timeout>
      <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
      <jdbc-lock-expiration>20000</jdbc-lock-expiration>
      <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
   </database-store>
</store>
```

4. 在新的 data-source-properties 元素中，为连接池添加 DBCP 数据源属性。指定键值对。例如：

```
<store>
   <database-store>
      <data-source-properties>
         <data-source-property key="driverClassName" value="com.mysql.jdbc.Driver" />
         <data-source-property key="url" value="jdbc:mysql://localhost:3306/artemis" />
         <data-source-property key="username"
value="ENC(5493dd76567ee5ec269d1182397346f)"/>
         <data-source-property key="password"
value="ENC(56a0db3b71043054269d1182397346f)"/>
         <data-source-property key="poolPreparedStatements" value="true" />
         <data-source-property key="maxTotal" value="-1" />
      </data-source-properties>
      <bindings-table-name>BINDINGS</bindings-table-name>
      <message-table-name>MESSAGES</message-table-name>
      <large-message-table-name>LARGE_MESSAGES</large-message-table-name>
      <page-store-table-name>PAGE_STORE</page-store-table-name>
      <node-manager-store-table-name>NODE_MANAGER_STORE</node-manager-store-
table-name>
      <jdbc-network-timeout>10000</jdbc-network-timeout>
      <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
      <jdbc-lock-expiration>20000</jdbc-lock-expiration>
      <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
   </database-store>
</store>
```

driverClassName

JDBC 数据库驱动程序的完全限定类名称。

url

您的数据库服务器的完整 JDBC 连接 URL。

username

您的数据库服务器的加密用户名。您还可以将这个值指定为 unencrypted, plain text。有关加密在配置文件中使用的用户名和密码的详情请参考 第 5.9 节 “在配置文件中加密密码”。

password

您的数据库服务器的加密密码。您还可以将这个值指定为 unencrypted, plain text。有关加密在配置文件中使用的用户名和密码的详情请参考 第 5.9 节 “在配置文件中加密密码”。

poolPreparedStatements

当此参数的值设为 **true** 时，池可以有无限数量的缓存准备语句。这降低了初始化成本。

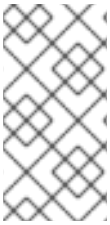
maxTotal

池中连接的最大数量。当此参数的值设为 **-1** 时，没有限制。

如果您没有显式配置 JDBC 连接池，代理将使用与默认配置的连接池。表中描述了默认配置。

表 6.1. 默认连接池配置

DBCP 配置参数	默认值
driverClassName	现有 **jdbc-driver-class-name** 参数的值
url	现有 **jdbc-connection-url** 参数的值
username	现有 **jdbc-user** 参数的值
password	现有 **jdbc-password** 参数的值
poolPreparedStatements	**true**
maxTotal	**-1**



注意

只有 没有客户端正在主动向代理发送消息时，重新连接才能正常工作。如果在重新连接过程中尝试写入数据库表，代理会失败并关闭。

其他资源

- 有关 AMQ Broker 支持的数据库的详情，请参考红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。
- 要了解 Apache Commons DBCP 软件包中所有可用的配置选项，请参阅 Apache Commons DBCP 配置参数。

6.3. 禁用持久性

在某些情况下，可能需要消息传递系统 不存储 任何数据。在这些情况下，您可以禁用代理上的持久

性。

以下步骤演示了如何禁用持久性。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

2. 在 core 元素中，将 persistence-enabled 参数的值设置为 false。

```
<configuration>
 <core>
  ...
  <persistence-enabled>false</persistence-enabled>
  ...
 </core>
</configuration>
```

没有消息数据、绑定数据、大消息数据、重复 ID 缓存或分页数据会被保留。

第 7 章 为地址配置内存用量

AMQ Broker 透明地支持包含数百万消息的大型队列，即使托管代理的机器使用有限内存运行。

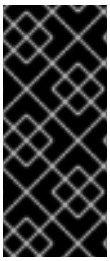
在这些情况下，可能无法随时将所有队列存储在内存中。要防止过量内存消耗，您可以配置代理上每个地址允许的最大内存用量。另外，您可以将代理配置为在内存用量达到配置的限制时执行以下操作之一：

- 页面消息
- 静默丢弃信息
- 丢弃消息并通知发送客户端
- 阻止发送消息的客户端

如果您在达到地址的最大内存用量时将代理配置为页面信息，您可以将特定地址的限制配置为：

- 限制用于页面传入的消息的磁盘空间
- 当客户端准备好使用消息时，限制代理从磁盘传输回内存的消息的内存。

您还可以设置磁盘用量阈值，以覆盖所有配置的分页限制。如果达到磁盘用量阈值，代理会停止分页并阻止所有传入的消息。



重要

使用事务时，代理可能会分配额外的内存以确保事务一致性。在这种情况下，代理报告的内存用量可能无法反映内存中使用的字节数。因此，如果您将代理配置为根据指定的最大内存用量将代理页面、丢弃或阻塞消息，则不应使用事务。

7.1. 配置消息分页

对于指定了最大内存用量限制的地址，您还可以指定达到该用量限制时代理所采取的操作。您可以配置的一个选项是 *分页*。

如果您配置分页选项，当达到最大地址大小时，代理将开始将那个地址的消息存储在磁盘上，并保存在称为 *页面文件 的文件中*。每个页面文件都有一个您可以配置的最大大小。您以这种方式配置的每个地址在文件系统中有一个专用文件夹来存储页面的消息。

在检查队列中的消息时，浏览器和消费者都可以浏览页面文件。但是，使用非常具体的过滤器的消费者可能无法使用存储在页面文件中的消息，直到队列中的现有消息被首先被消耗为止。例如，假设使用者过滤器包含字符串表达式，如 "color='red' "。如果满足此条件的消息遵循 100 万带有属性为 "color='blue'" 的属性，则消费者将无法消费此消息，直到首先消费了 "color='blue'"。

当客户端就绪使用时，代理传输（即，*depages*）消息从磁盘变为内存中。当该文件中的所有消息都已确认时，代理会从磁盘中删除页面文件。

以下流程演示了如何配置消息分页。

7.1.1. 指定分页目录

以下流程演示了如何指定分页目录的位置。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 core 元素中，添加 pages -directory 元素。指定文件系统中分页目录的位置。

```
<configuration ...>
  <core ...>
    ...
    <paging-directory>/path/to/paging-directory</paging-directory>
    ...
  </core>
</configuration>
```

对于您随后为分页配置的每个地址，代理会在您指定的分页目录中添加一个专用目录。

7.1.2. 为分页配置地址

以下流程演示了如何为分页配置地址。

先决条件

- 您应该熟悉如何配置地址和地址设置。更多信息请参阅 第 4 章 *配置地址和队列*。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 对于您为匹配地址 *或一组* 地址配置的 address-setting 元素，请添加配置元素来指定最大内存用量并定义分页行为。例如：

```
<address-settings>
  <address-setting match="my.paged.address">
    ...
    <max-size-bytes>104857600</max-size-bytes>
    <max-size-messages>20000</max-size-messages>
    <page-size-bytes>10485760</page-size-bytes>
    <address-full-policy>PAGE</address-full-policy>
    ...
  </address-setting>
</address-settings>
```

max-size-bytes

代理执行为 address-full-policy 属性指定的操作前允许地址的最大大小（以字节为单位）。默认值为 -1， 这意味着没有限制。您指定的值也支持字节表示法，如 "K"、"MB" 和 "GB"。

max-size-messages

代理执行为 address-full-policy 属性指定的操作前允许地址的最大消息数。默认值为 -1，这意味着没有消息限制。

page-size-bytes

分页系统上使用的每个页面文件的大小（以字节为单位）。默认值为 10485760 （即 10 MiB）。您指定的值也支持字节表示法，如 "K"、"MB" 和 "GB"。

address-full-policy

当达到地址的最大大小时，代理所采取的操作。默认值为 PAGE。有效值为：

页面

代理将任何进一步的信息页面到磁盘。

DROP

代理静默丢弃任何进一步的消息。

FAIL

代理将任何进一步的消息和异常丢弃给客户端消息制作者。

BLOCK

客户端消息制作者块尝试发送进一步消息。

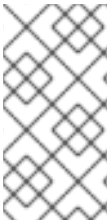
如果您为 max-size-bytes 和 max-size-message 属性设置了限制，代理会在达到任何限制时执行为 address-full-policy 属性指定的操作。在上例中，当内存中的地址总消息超过 20,000 或使用 104857600 字节可用内存时，代理会启动对 my.paged.address 地址的分页消息。

上例中 没有显示 的额外分页配置元素如下所述。

page-sync-timeout

定期页面同步之间的时间（以纳秒为单位）。如果您使用异步 IO 日志（即，在 broker.xml 配置文件中将 journal-type 设置为 ASYNCIO ），则默认值为 3333333。如果您使用标准 Java NIO 日志（即 journal-type 设置为 NIO），则默认值为 journal-buffer-timeout 参数的配置值。

在前面的示例中，当发送到地址 my.paged.address 的内存超过 104857600 字节的消息时，代理开始分页。



注意

如果在 address-setting 元素中指定 max-size-bytes，则该值将应用到 每个 匹配的地址。指定这个值 并不意味着 所有匹配地址 的总大小 限制为 max-size-bytes 的值。

7.1.3. 配置全局分页大小

有时，配置 *每个地址* 的内存限值不实际，例如当代理管理具有不同使用模式的多个地址时。在这些情况下，您可以指定一个全局内存限制。全局限制是代理可用于所有地址 的内存总量。当达到这个内存限值

时，代理会为与每个新传入的消息关联的地址执行为 address-full-policy 属性指定的操作。

以下流程演示了如何配置全局分页大小。

先决条件

- 您应该熟悉如何为分页配置地址。更多信息请参阅 第 7.1.2 节 “为分页配置地址”。

流程

1. 停止代理。

   a. 对于 Linux：

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. 在 Windows 上：

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

3. 在 core 元素中，添加 global-max-size 元素并指定一个值。例如：

   ```
   <configuration>
     <core>
       ...
       <global-max-size>1GB</global-max-size>
       <global-max-messages>900000</global-max-messages>
       ...
     </core>
   </configuration>
   ```

   global-max-size

   代理可用于所有地址的内存总量，以字节为单位。当达到这个限制时，代理会为与每个传入消息关联的地址执行为 address-full-policy 属性指定的操作。global-max-size 的默认值为托管代理的 Java 虚拟机(JVM)的最大内存的一半。

global-max-size 的值以字节为单位，但也支持字节表示法（例如："K"、"Mb"、"GB"）

在前面的示例中，代理配置为在处理消息时使用最多 1GB 可用内存。

global-max-messages

允许所有地址的消息总数。当达到这个限制时，代理会为与每个传入消息关联的地址执行为 address-full-policy 属性指定的操作。默认值为 -1，这意味着没有消息限制。

如果您为 global-max-size 和 global-max-messages 属性设置了限制，代理会在达到任何限制时执行为 address-full-policy 属性指定的操作。使用上例中的配置时，当内存中的消息数量超过 900,000 或使用 1 GB 可用内存时，代理会启动所有地址的分页消息。

注意

如果为单个地址设置的限制，通过使用 max-size-bytes 或 max-size-message 属性在为 global-max- size 或 global-max -messages 属性设置的限值之前访问，代理会执行为该地址的 address-full-policy 属性指定的操作。

4. 启动代理。

a. 对于 Linux：

```
<broker_instance_dir>/bin/artemis run
```

b. 在 Windows 上：

```
<broker_instance_dir>\bin\artemis-service.exe start
```

7.1.4. 在对特定地址进行分页期间限制磁盘用量

您可以在代理停止分页单个地址或地址集之前，限制代理可以使用的磁盘空间量。

流程

1. **停止代理。**

   a. **对于 Linux：**

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. **在 Windows 上：**

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. **打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。**

3. **在 core 元素中，添加属性，以根据磁盘使用情况或消息数或消息数指定分页限制，并指定在达到任一限制时要采取的操作。例如：**

```
<address-settings>
  <address-setting match="match="my.paged.address"">
   ...
   <page-limit-bytes>10G</page-limit-bytes>
   <page-limit-messages>1000000</page-limit-messages>
   <page-full-policy>FAIL</page-full-policy>
   ...
  </address-setting>
</address-settings>
```

page-limit-bytes

在代理执行为 page-full-policy 属性指定的操作前，允许进入地址的最大磁盘空间大小（以字节为单位）。您指定的值支持字节表示法，如 "K"、"MB" 和 "GB"。默认值为 -1，这意味着没有限制。

page-limit-messages

在代理执行为 page-full-policy 属性指定的操作前，可以为地址分页的最大传入消息数。默认值为 -1，这意味着没有消息限制。

page-full-policy

当达到了地址的 page-limit- bytes 或 page-limit -messages 属性中设置的限制时，代理需要执行的操作。有效值为：

DROP 代理静默丢弃任何进一步的消息。

FAIL。代理丢弃任何进一步的消息，并通知发送客户端

在上例中，代理页面会为 my.paged.address 地址分页信息，直到使用了 10 GB 的磁盘空间或直到所有 100万条消息都被被分页为止。

4. 启动**代理。**

a. 对于 **Linux：**

```
<broker_instance_dir>/bin/artemis run
```

b. **在 Windows 上：**

```
<broker_instance_dir>\bin\artemis-service.exe start
```

7.1.5. 控制将页面消息流到内存中

如果 AMQ Broker 配置为将信息页面到磁盘，代理会读取页面的消息，并在客户端就绪使用消息时将信息传送到内存中。要防止消息消耗过量内存，您可以为代理从磁盘传输到内存的消息限制每个地址使用的内存。

重要

如果客户端应用程序缺少太多消息待处理确认，代理不会在确认待处理消息前读取页面消息，这会导致代理上出现消息不足。

例如，如果将页面消息传输到内存（默认为 20 MB）的限值被访问，则代理会在读取任何更多消息前等待来自客户端的确认。如果同时，客户端在向代理发送确认前等待接收足够消息（由客户端使用的批处理大小决定），代理会耗尽消息。

为避免不足，可以将控制页面消息传输的代理限制增加到内存中，或减少传递消息的数量。您可以通过确保客户端很快提交消息确认，或使用超时和提交确认（不从代理接收更多消息）来减少发送消息的数量。

您可以在 AMQ 管理控制台中看到在队列的 Delivering Count 和 Delivering Bytes 指标中传递消息的编号和大小。

流程

1. 打开 & *lt;broker_instance_dir>* /etc/broker.xml 配置文件。

2. 对于您为匹配地址或一组地址配置的 address-settings 元素，请指定 paged 消息传输为内存中的限制。例如：

```
address-settings>
  <address-setting match="my.paged.address">
    ...
    <max-read-page-messages>104857600</max-read-page-messages>
    <max-read-page-bytes>20MB</max-read-page-bytes>
    ...
  </address-setting>
</address-settings>
```

max-read- page-messages 代理每个地址可以从磁盘读取的最大页面消息数。默认值为 -1，这意味着没有应用任何限制。

max-read-page-bytes 最大大小（以字节为单位），代理可以从磁盘读取到每个地址的内存。默认值为 20 MB。

当应用这些限制时，代理会计算内存中的消息，这些消息已准备好发送到当前提供的消费者

和消息。如果消费者确认消息缓慢，发送消息可能会导致达到内存或消息限制，并阻止代理将新消息读取到内存中。因此，代理可能会耗尽消息。

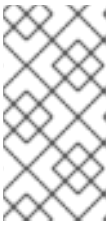
3. 如果消费者使用较慢来确认消息，并且当前提供的消息可以访问配置的 max-read-page-messages 或 max-read-page-bytes 限制，请为当前提供的消息指定单独的限制。例如：

```
address-settings>
  <address-setting match="my.paged.address">
    ...
    <prefetch-page-bytes>20MB</prefetch-page-bytes>
    <prefetch-page-messages>104857600</prefetch-page-messages>
    ...
  </address-setting>
</address-settings>
```

预抓取页字节 内存（以字节为单位），可用于将页面消息读取到每个队列的内存。默认值为 20 MB。

Prefetch-page-messages Number of paged 消息，代理可以从磁盘读取到每个队列的内存。默认值为 -1，这意味着没有应用任何限制。

如果您为 prefetch-page-bytes 或 prefetch-page-messages 参数指定限制当前交付的消息数量，请为 max-read-page- bytes 或 max-read-page -message 参数设置更高限制，以提供向内存中读取新消息的容量。



注意

如果在 prefetch-page-bytes 参数的值前达到 max-read-page-bytes 参数的值，代理将停止将进一步页面的消息读入内存中。

7.1.6. 设置磁盘用量阈值

您可以设置磁盘用量阈值，如果达到，则会导致代理停止分页并阻止所有传入的信息。

流程

1. 停止代理。

a. 对于 **Linux**：

```
<broker_instance_dir>/bin/artemis stop
```

b. 在 **Windows** 上：

```
<broker_instance_dir>\bin\artemis-service.exe stop
```

2. 打开 **&** ***lt;broker_instance_dir&*****gt; /etc/broker.xml** 配置文件。

3. 在 **core** 元素中，添加 **max-disk-usage** 配置元素并指定一个值。例如：

```
<configuration>
 <core>
  ...
  <max-disk-usage>80</max-disk-usage>
  ...
 </core>
</configuration>
```

max-disk-usage

代理可以使用的最大可用磁盘空间百分比。当达到这个限制时，代理会阻断传入的信息。默认值为 **90**。

在上例中，代理限制为使用百分之八十的可用磁盘空间。

4. 启动代理。

a. 对于 **Linux**：

```
<broker_instance_dir>/bin/artemis run
```

b. 在 **Windows** 上：

```
<broker_instance_dir>\bin\artemis-service.exe start
```

7.2. 配置消息丢弃

第 7.1.2 节 “为分页配置地址” 演示了如何为分页配置地址。作为该流程的一部分，您可以将 address-full-policy 的值设置为 PAGE。

当地址达到指定的最大值时，要 *丢弃* 信息（而不是分页），请将 address-full-policy 的值设置为以下之一：

DROP

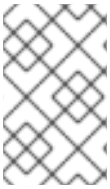
当达到给定地址的最大大小时，代理会静默丢弃任何进一步的信息。

FAIL

当达到给定地址的最大大小时，代理会丢弃任何进一步的消息和制作者问题。

7.3. 配置消息阻塞

以下流程演示了如何在给定地址达到您指定的最大大小限制时配置消息阻塞。



注意

您只能为 Core、OpenWire 和 AMQP 协议配置消息阻塞。

7.3.1. 阻塞内核和 OpenWire producer

以下流程演示了如何在给定地址达到您指定的最大大小限制时为 Core 和 OpenWire 消息生成者配置消息阻塞。

先决条件

- 您应该熟悉如何配置地址和地址设置。更多信息请参阅 第 4 章 *配置地址和队列*。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

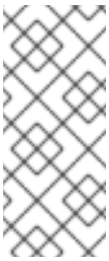
2.

对于您为匹配地址 *或一组* 地址配置的 address-setting 元素，请添加配置元素来定义消息阻塞行为。例如：

```
<address-settings>
  <address-setting match="my.blocking.address">
    ...
    <max-size-bytes>300000</max-size-bytes>
    <address-full-policy>BLOCK</address-full-policy>
    ...
  </address-setting>
</address-settings>
```

max-size-bytes

代理执行为 address-full-policy 指定的策略前允许地址的最大大小（以字节为单位）。您指定的值也支持字节表示法，如 "K"、"MB" 和 "GB"。



注意

如果在 address-setting 元素中指定 max-size-bytes，则该值将应用到每个 匹配的地址。指定这个值 并不意味着 所有匹配地址 的总大小 限制为 max-size-bytes 的值。

address-full-policy

代理在达到最大大小时执行的操作。

在上例中，当发送到地址 my.blocking.address 的内存超过 300000 字节的消息时，代理开始阻止来自 Core 或 OpenWire 消息制作者的进一步消息。

7.3.2. 阻塞 AMQP 生成者

Core 和 OpenWire 等协议使用一个窗口大小流控制系统。在这个系统中，信用代表字节，并分配到制作者。如果生成者希望发送消息，则生成者必须等待到其达到消息大小的足够信。

相反，AMQP 流控制信用不会代表字节数。相反，AMQP 学位允许生成者发送的消息数量，而不考虑消息大小。因此，在某些情况下，AMQP 生成者会显著超过地址的 max-size-bytes 值。

因此，要阻止 AMQP 生成者，您必须使用不同的配置元素 max-size-bytes-reject-threshold。对于匹配的地址或一组地址，此元素指定内存中所有 AMQP 消息的最大大小（以字节为单位）。当内存中所有消息的总大小达到指定的限制时，代理会阻止 AMQP 生成者发送进一步消息。

以下步骤演示了如何为 AMQP 消息生成者配置消息阻塞。

先决条件

- 您应该熟悉如何配置地址和地址设置。更多信息请参阅 第 4 章 *配置地址和队列*。

流程

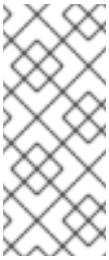
1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 对于您为匹配地址 *或一组* 地址配置的 address-setting 元素，请指定内存中所有 AMQP 消息的最大大小。例如：

```
<address-settings>
  <address-setting match="my.amqp.blocking.address">
    ...
    <max-size-bytes-reject-threshold>300000</max-size-bytes-reject-threshold>
    ...
  </address-setting>
</address-settings>
```

max-size-bytes-reject-threshold

代理阻止进一步的 AMQP 消息之前允许地址的最大大小（以字节为单位）。您指定的值也支持字节表示法，如 "K"、"MB" 和 "GB"。默认情况下，max-size-bytes-reject-threshold 设置为 -1， 这意味着没有最大大小。



注意

如果您在 address-setting 元素中指定 max-size-bytes-reject-threshold，则该值将应用到 每个 匹配的地址。指定这个值 并不意味着 所有匹配地址 的总大小 限制为 max-size-bytes-reject-threshold 的值。

在上例中，当发送到地址 my.amqp.blocking.address 的消息在内存中超过 300000 字节时，代理开始阻止来自 AMQP 生成者的进一步消息。

7.4. 了解多播地址上的内存用量

当消息路由到绑定了多播队列的地址时，内存中只有一个消息副本。每个队列仅具有对邮件的*引用*。因

此，只有在引用该消息 的所有 队列后才会释放相关的内存。

在这种情形中，如果您有缓慢的消费者，整个地址可能会遇到负面影响。

例如，请考虑这种情况：

- 一个地址有十个队列，它使用 multicast 路由类型。
- 由于消费者缓慢，其中一个队列不提供其消息。其他 9 个队列继续传递消息并为空。
- 消息继续到达地址。带有缓慢消费者的队列将继续积累对消息的引用，从而导致代理在内存中保留消息。
- 当达到最大地址大小时，代理会开始页信息。

在这种情况下，由于一个缓慢的消费者，所有 队列上的消费者都被强制使用来自页面系统的消息，这需要额外的 IO。

其他资源

- 要了解如何配置流控制，以规范代理和生产者和消费者之间的数据流，请参阅 AMQ 核心协议 JMS 文档中的 流控制。

第 8 章 处理大型消息

客户端可能会发送可能会超过代理内部缓冲区大小的大型消息，从而导致意外错误。要防止这种情况，您可以将代理配置为当消息大于指定最小值时将消息存储为文件。以这种方式处理大型消息意味着代理不会在内存中保存消息。相反，您可以在磁盘上指定目录，或者在代理存储大型消息文件的数据库表中。

当代理将消息存储为大消息时，队列会在大型消息目录或数据库表中保留对文件的引用。

大型消息处理可用于核心协议、AMQP、OpenWire 和 STOMP 协议。

对于核心协议和 OpenWire 协议，客户端在其连接配置中指定最小大消息大小。对于 AMQP 和 STOMP 协议，您可以在代理配置中为每个协议定义的接受者中指定最小大消息大小。

注意

建议您不要使用不同的协议 来生成和使用大消息。要做到这一点，代理可能需要执行一些消息的转换。例如，假设您想使用 AMQP 协议发送消息，并使用 OpenWire 接收消息。在这种情况下，代理必须首先读取大型消息的完整正文，并将其转换为使用 Core 协议。然后，代理必须执行另一个转换，这一次到 OpenWire 协议。比如这些信息转换，比如在代理中造成大量处理开销。

您为上述任何协议指定的最小大消息大小会受到系统资源（如可用的磁盘空间量以及消息大小）的影响。建议您使用几个值运行性能测试来确定适当的大小。

本节中的步骤演示了如何：

- 配置代理以存储大型消息
- 为 AMQP 和 STOMP 协议配置用于大型消息处理的接收器

本节还链接到有关配置 AMQ Core Protocol 和 AMQ OpenWire JMS 客户端以处理大量消息的其他资源。

8.1. 为大型消息处理配置代理

以下流程演示了如何在磁盘上指定目录，或者代理在其中存储大型消息文件的数据库表。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 指定希望代理存储大型消息文件的位置。

   a. 如果您在磁盘上存储大量消息，请在 core 元素中添加 large-messages-directory 参数并指定文件系统位置。例如：

```
<configuration>
  <core>
    ...
    <large-messages-directory>/path/to/my-large-messages-directory</large-messages-directory>
    ...
  </core>
</configuration>
```

注意

如果您没有为 large-messages-directory 明确指定值，代理将使用默认值 < *broker_instance_dir*> /data/largemessages

   b. 如果您在数据库表中存储大型消息，请将 large-message-table 参数添加到 database-store 元素中，并指定一个值。例如：

```
<store>
  <database-store>
    ...
    <large-message-table>MY_TABLE</large-message-table>
    ...
  </database-store>
</store>
```

注意

如果没有为 large-message-table 显式指定值，代理将使用默认值 LARGE_MESSAGE_TABLE。

其他资源

- 有关配置数据库存储的详情，请参考 第 6.2 节 “在数据库中持久保留消息数据”。

8.2. 为大型消息处理配置 AMQP 接受器

以下流程演示了如何配置 AMQP 接受器，以处理大于指定大小的 AMQP 消息作为大消息。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

   代理配置文件中的默认 AMQP 接受器如下所示：

```
<acceptors>
   ...
   <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>
   ...
</acceptors>
```

2. 在默认的 AMQP acceptor （或者您配置的另一个 AMQP acceptor ）中，添加 amqpMinLargeMessageSize 属性并指定一个值。例如：

```
<acceptors>
   ...
   <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCredits=1000;amqpLowCredits=300;amqpMinLargeMessageSize=204800</acceptor>
   ...
</acceptors>
```

   在上例中，代理配置为接受端口 5672 上的 AMQP 消息。根据 amqpMinLargeMessageSize 的值，如果接受者收到一个大于或等于 204800 字节的正文的 AMQP 消息（即 200 KB），代理将消息存储为大消息。如果您没有为此属性显式指定值，代理将使用默认值 102400 （即 100 KB）。

注意

- 如果将 amqpMinLargeMessageSize 设置为 -1，则对 AMQP 消息的大型消息处理被禁用。

- 如果代理收到一个持久性 AMQP 消息，它没有超过 amqpMinLargeMessageSize 的值，但*超出*消息传递日志缓冲区的大小（使用 journal-buffer-size 配置参数指定），代理会将消息转换为大型核心协议消息，然后再将其存储在日志中。

8.3. 配置 STOMP ACCEPTOR 用于处理大量信息

以下流程演示了如何配置 STOMP acceptor 来处理大于指定大小的 STOMP 消息。

流程

1. 打开 & *lt;broker_instance_dir&*gt; /etc/broker.xml 配置文件。

   代理配置文件中的默认 AMQP 接受器如下所示：

```
<acceptors>
   ...
   <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEp
oll=true</acceptor>
   ...
</acceptors>
```

2. 在默认的 STOMP acceptor （或其他您配置的 STOMP acceptor ）中，添加 stompMinLargeMessageSize 属性并指定值。例如：

```
<acceptors>
   ...
   <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEp
oll=true;stompMinLargeMessageSize=204800</acceptor>
   ...
</acceptors>
```

在上例中，代理被配置为接受端口 61613 上的 STOMP 信息。根据 stompMinLargeMessageSize 的值，如果 acceptor 收到一个大于或等于 204800 字节的 STOMP 消息（即 200 KB），代理会将消息存

储为大消息。如果您没有为此属性显式指定值，代理将使用默认值 102400 （即 100 KB）。

注意

为了向 STOMP 消费者发送大消息，代理会在将消息发送到客户端前自动将消息从大型消息转换为正常消息。如果压缩大型消息，代理会在将其发送到 STOMP 客户端前解压缩它。

8.4. 大消息和 JAVA 客户端

有两个选项可供 Java 开发人员使用大型消息编写客户端。

一个选项是使用 InputStream 和 OutputStream 的实例。例如，FileInputStream 可用于发送从物理磁盘上的大文件中获取的消息。然后，接收器可以使用 FileOutputStream 将消息流传输到其本地文件系统中的位置。

另一种选择是直接流传输一个 JMS BytesMessage 或 StreamMessage。例如：

```
BytesMessage rm = (BytesMessage)cons.receive(10000);
byte data[] = new byte[1024];
for (int i = 0; i < rm.getBodyLength(); i += 1024)
{
  int numberOfBytes = rm.readBytes(data);
  // Do whatever you want with the data
}
```

其他资源

- 要了解在 AMQ Core Protocol JMS 客户端中使用大量信息的信息，请参阅：
  - 大消息选项
  - 写入流大消息
  - 从流化大消息中读取

- 要了解在 AMQ OpenWire JMS 客户端中使用大量信息的信息，请参阅：
  - 大消息选项
  - 写入流大消息
  - 从流化大消息中读取
- 有关使用大型消息的示例，请查看 大型消息示例。要了解更多有关运行示例程序的信息，请参阅 运行 AMQ Broker 示例。

第 9 章 检测死连接

有时，客户端会意外停止，且没有清理其资源的机会。如果发生了这种情况，它可以使资源处于故障状态，并导致代理耗尽内存或其他系统资源。代理检测到客户端的连接没有在垃圾回收时正确关闭。然后，连接关闭，类似以下的消息将写入日志。日志捕获了客户端会话实例化的确切代码行。这可让您识别错误并进行更正。

```
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession]
I'm closing a JMS Conection you left open. Please make sure you close all connections 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) 1
```

1

实例化连接的客户端代码中的行。

9.1. CONNECTION TIME-TO-LIVE

因为客户端和服务器之间的网络连接可能会失败，然后返回在线，允许客户端重新连接，AMQ Broker 会等待清理不活跃的服务器端资源。这个等待周期被称为生存时间(TTL)。基于网络的连接的默认 TTL 为 60000 毫秒(1 分钟)。in-VM 连接中的默认 TTL 为 -1， 这意味着代理永远不会在代理端超时连接。

在 Broker 上配置 Time-To-Live

如果您不希望客户端指定自己的连接 TTL，您可以在代理端设置全局值。这可以通过在代理配置中指定 connection-ttl-override 元素来实现。

根据 connection-ttl-check-interval 元素决定，检查 TTL 违反连接的逻辑在代理上定期运行。

流程

- 通过添加 connection-ttl-override 配置元素并提供一个 time-to-live 的值来编辑 < *broker_instance_dir*> /etc/broker.xml，如下例所示。

```
<configuration>
 <core>
  ...
```

```
 <connection-ttl-override>30000</connection-ttl-override> (1)
 <connection-ttl-check-interval>1000</connection-ttl-check-interval> (2)
 ...
</core>
</configuration>
```

(1) 所有连接的全局 TTL 都设为 30000 毫秒。默认值为 -1， 它允许客户端设置自己的 TTL。

(2) 死连接检查之间的间隔设置为 1000 毫秒。默认情况下，检查会每 2000 毫秒进行一次。

9.2. 禁用异步连接执行

代理端接收的大部分数据包都在 remoting 线程上执行。这些数据包代表短暂运行的操作，并且始终在远程线程上执行，以提高性能。但是，有些数据包类型使用线程池执行，而不是使用远程线程，这会增加较少的网络延迟。

使用线程池的数据包类型在下面列出的 Java 类中实施。在软件包 org.apache.actiinvemq.artemis.core.protocol.core.impl.wireformat.

- RollbackMessage
- SessionCloseMessage
- SessionCommitMessage
- SessionXACommitMessage
- SessionXAPrepareMessage
- SessionXARollbackMessage

流程

- 要禁用异步连接执行，将 async-connection-execution-enabled 配置元素添加到 <*broker_instance_dir*> /etc/broker.xml，并将其设置为 false，如下例所示。默认值为 true。

```
<configuration>
 <core>
  ...
  <async-connection-execution-enabled>false</async-connection-execution-enabled>
  ...
 </core>
</configuration>
```

其他资源

- 要了解如何配置 AMQ Core Protocol JMS 客户端来检测死连接，请参阅 AMQ Core Protocol JMS 文档中的 Detecting dead 连接。

- 要了解如何在 AMQ 核心协议 JMS 客户端中配置连接时间，请参阅 AMQ 核心协议 JMS 文档中的配置生存时间。

第 10 章 检测重复消息

您可以将代理配置为自动检测和过滤重复的消息。这意味着您不必实施自己的重复检测逻辑。

如果没有重复检测，当意外连接失败时，客户端无法确定它发送到代理的消息是否收到。在这种情况下，客户端可能会假设代理没有收到消息，并重新发送它。这会生成重复的消息。

例如，假设客户端向代理发送消息。如果在代理接收和 *由代理处理消息前* 代理或连接失败，则消息永远不会到达其地址。由于失败，客户端不会从代理接收响应。如果在代理接收和处理消息 *后* 代理或连接失败，则消息会正确路由，但客户端仍然不会收到响应。

此外，使用事务来确定成功并不一定帮助。如果在处理事务提交时代理或连接失败，客户端仍然无法确定它是否已成功发送消息。

在这些情况下，为了更正假定的失败，客户端会重新发送最新的消息。结果可能是对您的系统有负面影响的重复消息。例如，如果您在一个订购系统中使用代理，则重复消息可能意味着处理购买顺序两次。

以下流程演示了如何配置重复消息检测以防止这些类型的情况。

10.1. 配置重复的 ID 缓存

要让代理检测重复消息，生成者必须在发送每个消息时为消息属性 _AMQ_DUPL_ID 提供唯一值。代理维护 _AMQ_DUPL_ID 属性收到的值的缓存。当代理在地址上收到新消息时，它会检查该地址的缓存，以确保之前没有为此属性处理具有相同值的消息。

每个地址都有自己的缓存。每个缓存都是循环并固定大小。这意味着新条目将最旧的条目替换为缓存空间需求。

以下流程演示了如何全局配置代理上每个地址使用的 ID 缓存。

流程

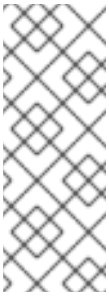
1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 core 元素中，添加 id-cache-size 和 persist-id-cache 属性并指定值。例如：

```
<configuration>
 <core>
  ...
  <id-cache-size>5000</id-cache-size>
  <persist-id-cache>false</persist-id-cache>
 </core>
</configuration>
```

id-cache-size

ID 缓存的最大大小，指定为缓存中单个条目的数量。默认值为 20,000 个条目。在本例中，缓存大小被设置为 5,000 个条目。



注意

当达到最大缓存大小时，代理可以开始处理重复消息。例如，假设您将缓存的大小设置为 3000。如果以前的消息在与 _AMQ_DUPL_ID 的值相同的新消息之前到达 3,000 条消息，则代理无法检测到重复的消息。这会导致代理处理这两个消息。

persist-id-cache

当此属性的值设为 true 时，代理会在接收时将 ID 保留为磁盘。默认值为 true。在上例中，您可以通过将值设为 false 来禁用持久性。

其他资源

- 要了解如何使用 AMQ Core Protocol JMS 客户端设置重复的 ID 消息属性，请参阅 AMQ Core Protocol JMS 客户端 文档中的使用重复消息检测。

10.2. 为集群连接配置重复检测

您可以配置集群连接，为每个移动在集群中的消息插入重复的 ID 标头。

先决条件

- 您应该已经配置了代理集群。如需更多信息，请参阅 第 14.2 节 “创建代理集群”。

流程

1. 打开 & *lt;broker_instance_dir*> /etc/broker.xml 配置文件。

2. 在 core 元素中，对于给定集群连接，添加 use-duplicate-detection 属性并指定值。例如：

```
<configuration>
 <core>
  ...
  <cluster-connections>
   <cluster-connection name="my-cluster">
    <use-duplicate-detection>true</use-duplicate-detection>
    ...
   </cluster-connection>
    ...
  </cluster-connections>
 </core>
</configuration>
```

use-duplicate-detection

当此属性的值设为 **true** 时，集群连接会为它处理的每个消息插入重复的 **ID** 标头。

第 11 章 截获消息

使用 AMQ Broker，您可以截获进入或退出代理的数据包，允许您审核数据包或过滤信息。拦截器可以更改它们拦截的数据包，使其功能强大，但也可能有危险。

您可以开发拦截器来满足您的业务需求。拦截器是特定于协议的，必须实施适当的接口。

拦截器必须实施 intercept （） 方法，它会返回一个布尔值。如果值为 true，则消息数据包将继续。如果为 false，则会中止进程，不调用其他拦截器，且消息数据包不会被进一步处理。

11.1. 创建拦截器

您可以创建自己的传入和传出拦截器。所有拦截器都是特定于协议的，对于分别进入或退出服务器的任何数据包都调用。这可让您创建拦截器来满足审计数据包等业务需求。拦截器可以更改它们拦截的数据包。这使得他们变得强大以及潜在的危险，因此请务必谨慎使用。

拦截器及其依赖项必须放在代理的 Java 类路径中。您可以使用 &lt ;*broker_instance_dir*> /lib 目录，因为它默认为 classpath 的一部分。

流程

以下示例演示了如何创建拦截器来检查传递给它的每个数据包的大小。请注意，示例为每个协议实施特定的接口。

- 实施适当的接口并覆盖其拦截器 （） 方法。
  - 如果您使用 AMQP 协议，请实施 org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor 接口。

```
package com.example;

import org.apache.activemq.artemis.protocol.amqp.broker.AMQPMessage;
import org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements AmqpInterceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
```

```
  public boolean intercept(final AMQPMessage message, RemotingConnection
connection)
  {
    int size = message.getEncodeSize();
    if (size <= ACCEPTABLE_SIZE) {
      System.out.println("This AMQPMessage has an acceptable size.");
      return true;
    }
    return false;
  }
}
```

- 如果使用核心协议，您的拦截器必须实施 org.apache.artemis.activemq.api.core.Interceptor 接口。

```
package com.example;

import org.apache.artemis.activemq.api.core.Interceptor;
import org.apache.activemq.artemis.core.protocol.core.Packet;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
  boolean intercept(Packet packet, RemotingConnection connection)
  throws ActiveMQException
  {
    int size = packet.getPacketSize();
    if (size <= ACCEPTABLE_SIZE) {
      System.out.println("This Packet has an acceptable size.");
      return true;
    }
    return false;
  }
}
```

- 如果您使用 MQTT 协议，请实施 org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor 接口。

```
package com.example;

import org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor;
import io.netty.handler.codec.mqtt.MqttMessage;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
  private final int ACCEPTABLE_SIZE = 1024;
```

```
  @Override
  boolean intercept(MqttMessage mqttMessage, RemotingConnection connection)
  throws ActiveMQException
  {
   byte[] msg = (mqttMessage.toString()).getBytes();
   int size = msg.length;
   if (size <= ACCEPTABLE_SIZE) {
    System.out.println("This MqttMessage has an acceptable size.");
    return true;
   }
   return false;
  }
}
```

- 如果您使用 STOMP 协议，请实施 org.apache.activemq.artemis.core.protocol.stomp.StompFrameInterceptor 接口。

```
package com.example;

import org.apache.activemq.artemis.core.protocol.stomp.StompFrameInterceptor;
import org.apache.activemq.artemis.core.protocol.stomp.StompFrame;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
  boolean intercept(StompFrame stompFrame, RemotingConnection connection)
  throws ActiveMQException
  {
   int size = stompFrame.getEncodedSize();
   if (size <= ACCEPTABLE_SIZE) {
    System.out.println("This StompFrame has an acceptable size.");
    return true;
   }
   return false;
  }
}
```

11.2. 将 BROKER 配置为使用 INTERCEPTORS

创建拦截器后，您必须将代理配置为使用它。

先决条件

您必须创建一个拦截器类，并将其（及其依赖项）添加到代理的 Java 类路径中，然后才能将其配置为代理使用。您可以使用 &lt ;*broker_instance_dir*> /lib 目录，因为它默认为 classpath 的一部分。

流程

- 通过将配置添加到 <broker _*instance_dir&gt*; /etc/broker.xml，将代理配置为使用拦截器

  - 如果您的拦截器用于传入的消息，请将其 class-name 添加到 remoting-incoming-interceptors 列表中。

```
<configuration>
  <core>
    ...
    <remoting-incoming-interceptors>
      <class-name>org.example.MyIncomingInterceptor</class-name>
    </remoting-incoming-interceptors>
    ...
  </core>
</configuration>
```

  - 如果您的拦截器用于传出消息，请将其 class-name 添加到 remoting-outgoing-interceptors 列表中。

```
<configuration>
  <core>
    ...
    <remoting-outgoing-interceptors>
      <class-name>org.example.MyOutgoingInterceptor</class-name>
    </remoting-outgoing-interceptors>
  </core>
</configuration>
```

其他资源

- 要了解如何在 AMQ 核心协议 JMS 客户端中配置 拦截器，请参阅 AMQ Core Protocol JMS 文档中的使用消息拦截器。

第 12 章 分离消息和分割消息流

在 AMQ Broker 中，您可以配置名为 *diverts* 的对象，可让您透明地将消息从一个地址分离到另一个地址，而无需更改任何客户端应用程序逻辑。您还可以配置一个 divert，将消息 的副本 转发到指定的转发地址，从而有效地分割消息流。

12.1. MESSAGE DIVERTS 如何工作

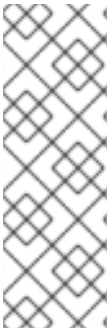
通过 diverts，您可以透明地将路由到一个地址的消息重定向到其他地址，而无需更改任何客户端应用程序逻辑。将代理服务器上的一组 diverts 视为消息的路由表类型。

The divert can is *exclusive*, 表示将消息定向到指定的转发地址，而无需转至其原始地址。

探测器也可以是 *非独占* 的，这意味着消息将继续进入其原始地址，而代理会将消息副本发送到指定的转发地址。因此，您可以使用非独占分离来分割消息流。例如，如果要单独监控发送到一个订购队列的每个顺序，您可以分割消息流。

可以为单个地址配置多个 diverts。当地址同时配置了 exclusive 和 non-exclusive diverts 时，代理会首先处理 exclusive diverts。如果特定消息已被一个 exclusive divert 处理，则代理不会处理该消息的任何非排除分离器。在这种情况下，消息永远不会进入原始地址。

当代理过滤消息时，代理会分配一个新的消息 ID，并将消息地址设置为新的转发地址。您可以通过 _AMQ_ORIG_ADDRESS （字符串类型）和 _AMQ_ORIG_MESSAGE_ID （长类型）消息属性来检索原始消息 ID 和地址值。如果您使用 Core API，请使用 Message.HDR_ORIGINAL_ADDRESS 和 Message.HDR_ORIG_MESSAGE_ID 属性。



注意

您只能将消息定向到同一代理服务器上的地址。如果您要将一个不同服务器上的地址分解到不同服务器上的地址，则常见的解决方案是首先将消息分解为本地 store-and-forward 队列。然后，设置从那个队列消耗的网桥，并将消息转发到不同代理上的地址。通过将分离与网桥相结合，您可以在地理分布的代理服务器之间创建分布式路由连接网络。这样，您可以创建一个全局消息传递网格。

12.2. 配置消息 DIVERTS

要在代理实例中配置一个 divert，在 broker.xml 配置文件的 core 元素中添加一个 divert 元素。

```
<core>
...
  <divert name= >
      <address> </address>
      <forwarding-address> </forwarding-address>
      <filter string= >
      <routing-type> </routing-type>
      <exclusive> </exclusive>
  </divert>
...
</core>
```

divert

分离的命名实例。您可以在 broker.xml 配置文件中添加多个 divert 元素，只要每个 divert 都有唯一的名称。

address

从中分离 消息的地址

forward-address

信息转发到的地址

filter

可选消息过滤器。如果您配置过滤器，则只有与过滤器字符串匹配的消息才会被分离。如果没有指定过滤器，则 divert 认为所有消息都被视为匹配项。

routing-type

划分消息的路由类型。您可以将 divert 配置为：

- 将 anycast 或 multicast 路由类型应用到消息
- *去除*（即删除）现有路由类型
- *传递*（即保留）现有路由类型

当消息设置了路由类型时，路由类型的控制很有用，但您想要将消息定向到使用不同路由类型的地址。例如，代理无法将 anycast 路由类型的消息路由到一个使用 multicast 的队列，除非您将 divert 的

routing-type 参数设置为 MULTICAST。divert 的 routing-type 参数的有效值为 ANYCAST、MULTICAST、PASS 和 STRIP。默认值为 STRIP。

exclusive

指定 divert 是 exclusive （将属性设为 true）还是非专用（将属性设为 false）。

以下小节展示了 exclusive 和 non-exclusive diverts 的配置示例。

12.2.1. 排除 divert 示例

下面是一个排除 divert 的示例配置。排除 divert 将原始配置的地址的所有匹配消息复制到新地址。匹配消息不会路由到原始地址。

```
<divert name="prices-divert">
  <address>priceUpdates</address>
  <forwarding-address>priceForwarding</forwarding-address>
  <filter string="office='New York'"/>
  <exclusive>true</exclusive>
</divert>
```

在前面的示例中，您定义了一个名为 price-divert 的 divert，它将发送到 priceUpdates 地址的任何消息转到另外一个本地地址 priceForwarding。您还指定了消息过滤器字符串。只有带有 office 属性以及值为 New York 的消息才会被 divert。所有其他消息都会路由到其原始地址。最后，您可以指定 divert 是 exclusive。

12.2.2. Non-exclusive divert 示例

下面是一个非排除的 divert 的示例配置。在非独占 divert 中，消息将继续进入其原始地址，而代理还会将消息副本发送到指定的转发地址。因此，非独占 divert 是一个分割消息流的方法。

```
<divert name="order-divert">
  <address>orders</address>
  <forwarding-address>spyTopic</forwarding-address>
  <exclusive>false</exclusive>
</divert>
```

在上例中，您定义了一个名为 order-divert 的 divert，它会获取发送到地址 orders 的每个消息的副本，并将其发送到名为 spyTopic 的本地地址。您还可以指定 divert 不是排他的。

其他资源

有关使用 exclusive 和 non-exclusive diverts 的详细示例，以及一个桥接将消息转发到另一个代理，请参阅 divert 示例 （外部）。

第 13 章 过滤消息

AMQ Broker 提供基于 SQL 92 表达式语法的子集的强大过滤器语言。过滤器语言使用与 JMS 选择器相同的语法，但预定义的标识符不同。下表列出了应用到 AMQ Broker 消息的标识符。

标识符	属性
AMQPriority	消息的优先级。消息优先级是带有从 **0** 到 **9** 的有效值的整数。**0** 是最低的优先级，**9** 是最高优先级。
AMQExpiration	消息的过期时间。该值是一个长整数。
AMQDurable	消息是否是持久化的。该值是一个字符串。有效值为 **DURABLE** 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 **encodeSize** 属性的值。**encodeSize** 的值是日志中消息占用的空间（以字节为单位）。因为代理使用双字节字符集来对消息进行编码，所以消息的实际大小是 **encodeSize** 的值的一半。

内核过滤器表达式中使用的任何其他标识符都假定为消息的属性。有关 JMS 消息选择器语法的文档，请参阅 Java EE API。

13.1. 将队列配置为使用过滤器

您可以将过滤器添加到您在 *< broker_instance_dir>* /etc/broker.xml 中配置的 队列。只有与过滤器表达式匹配的消息会进入队列。

流程

- 将 filter 元素添加到所需的 队列，并包含您要作为元素值应用的过滤器。在以下示例中，filter NEWS=' technical' 添加到队列 technologyQueue 中。

```
<configuration>
 <core>
  ...
  <addresses>
    <address name="myQueue">
      <anycast>
        <queue name="myQueue">
          <filter string="NEWS='technology'"/>
        </queue>
```

```
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

13.2. 过滤 JMS 消息属性

JMS 规格指出，在选择器中使用 String 属性时不得转换为数字类型。例如，如果消息将 age 属性设置为 String 值 21，则选择器 期限 > 18 不得与它匹配。这个限制限制 STOMP 客户端，因为它们只能使用 String 属性发送消息。

将过滤器配置为将字符串转换为一个数字

要将 String 属性转换为数字类型，请将前缀 convert_string_expressions: 添加到 过滤器 的值。

流程

- 编辑 *<broker_instance_dir>*/etc/broker.xml，为相关的 filter 应用前缀 convert_string_expressions:。以下示例将 filter 值从 age > 18 编辑为 convert_string_expressions:age > 18。

```
<configuration>
  <core>
    ...
    <addresses>
      <address name="myQueue">
        <anycast>
          <queue name="myQueue">
            <filter string="convert_string_expressions='age > 18'"/>
          </queue>
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

13.3. 根据注释上的属性过滤 AMQP 消息

在将过期或未提供 AMQP 消息移动到您配置的到期或死信队列之前，代理会将注解和属性应用到消息。客户端可以根据属性或注解创建过滤器，从到期或死信队列中选择要使用的特定消息。

注意

代理应用的属性是 *内部* 属性。这些属性不公开给客户端供常规使用，但 可以被 过滤器中的客户端指定。

以下是基于消息属性和注解的过滤器示例。根据属性进行过滤是推荐的方法，因为这种方法需要较少的代理处理。

根据消息属性过滤

```
ConnectionFactory factory = new JmsConnectionFactory("amqp://localhost:5672");
Connection connection = factory.createConnection();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
connection.start();
javax.jms.Queue queue = session.createQueue("my_DLQ");
MessageConsumer consumer = session.createConsumer(queue,
"_AMQ_ORIG_ADDRESS='original_address_name'");
Message message = consumer.receive();
```

根据消息注解过滤

```
ConnectionFactory factory = new JmsConnectionFactory("amqp://localhost:5672");
Connection connection = factory.createConnection();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
connection.start();
javax.jms.Queue queue = session.createQueue("my_DLQ");
MessageConsumer consumer = session.createConsumer(queue, "\"m.x-opt-ORIG-
ADDRESS\"='original_address_name'");
Message message = consumer.receive();
```

注意

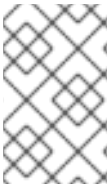
当基于注解消耗 AMQP 消息时，客户端必须将 m. 前缀附加到消息注解中，如上例中所示。

其他资源

- 有关代理应用到过期或未提供 AMQP 消息的注解和属性的更多信息，请参阅 第 4.14 节 “在过期或未提供 AMQP 消息上注解和属性”。

13.4. 过滤 XML 消息

AMQ Broker 提供过滤文本消息的方法，其中包含使用 XPath 的 XML 正文。XPath (XML 路径语言) 是从 XML 文档中选择节点的查询语言。



注意

仅支持基于文本的消息。不支持过滤大型消息。

要过滤基于文本的消息，您需要创建一个 XPATH '<xpath-expression> 形式的 Message Selector。

消息正文示例

```
<root>
   <a key='first' num='1'/>
   <b key='second' num='2'>b</b>
</root>
```

基于 XPath 查询过滤

```
PATH 'root/a'
```

警告

由于 XPath 适用于消息的正文并需要解析 XML，因此过滤可能会比正常过滤器要慢得多。

XPath 过滤器使用以下协议在生成者和消费者之间支持：

- OpenWire JMS
- Core （和核心 JMS）
- STOMP
- AMQP

配置 XML Parser

默认情况下，代理使用的 XML Parser 是 JDK 使用的平台默认的 DocumentBuilderFactory 实例。

用于 XPath 默认配置的 XML 解析器包括以下设置：

- http://xml.org/sax/features/external-general-entities: false
- http://xml.org/sax/features/external-parameter-entities: false
- http://apache.org/xml/features/disallow-doctype-decl: true

但是，为了处理任何特定于实施的问题，可以通过在 artemis.profile 配置文件中配置系统属性来自定义功能。

org.apache.activemq.documentBuilderFactory.feature:prefix

功能配置示例

```
-Dorg.apache.activemq.documentBuilderFactory.feature:http://xml.org/sax/features/external-general-entities=true
```

第 14 章 设置代理集群

集群由多个已分组在一起的代理实例组成。代理集群通过在多个代理间分布消息处理负载来提高性能。另外，代理集群可以通过高可用性来最小化停机时间。

您可以在许多不同的集群拓扑中连接代理。在集群中，每个活跃代理管理自己的信息，并处理自己的连接。

您还可以平衡跨集群的客户端连接并重新分发信息，以避免代理不足。

14.1. 了解代理集群

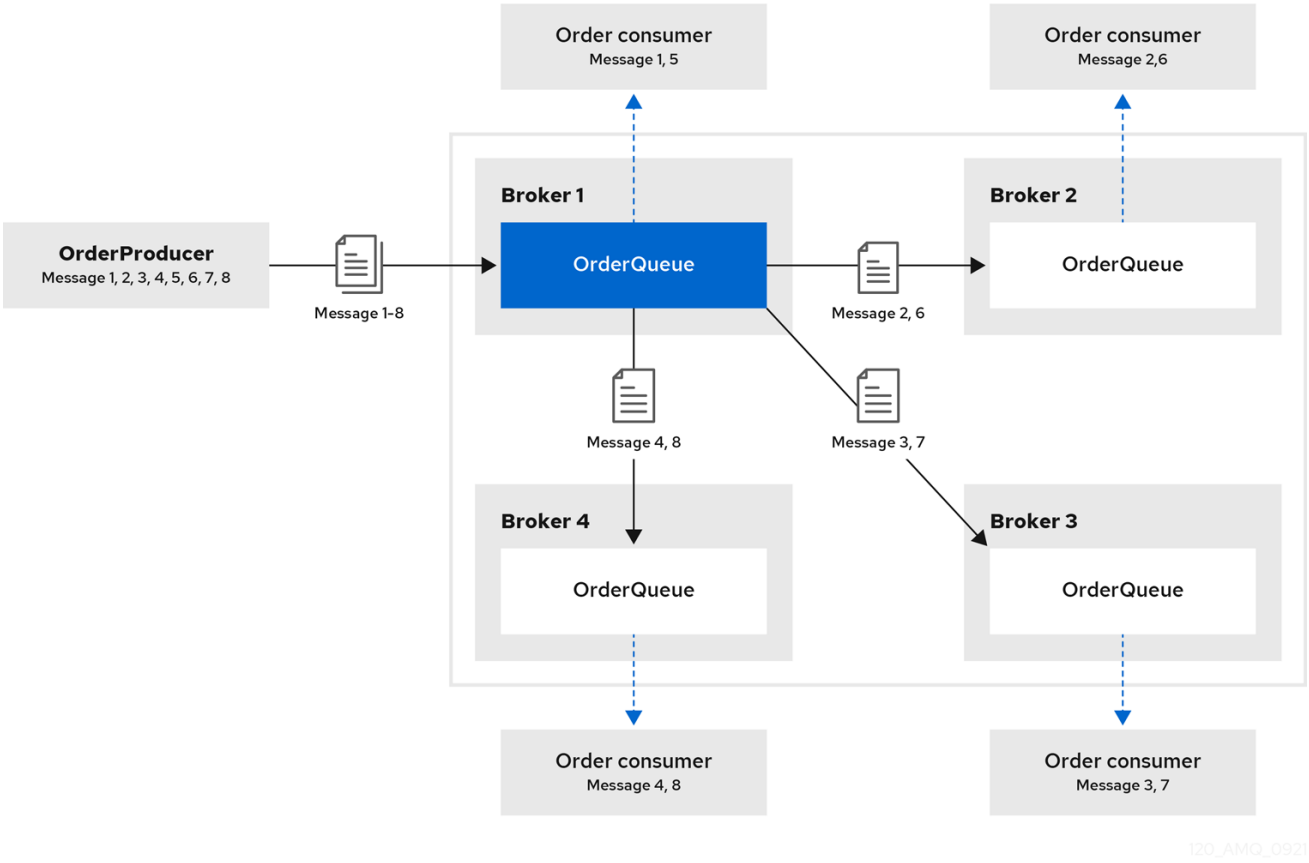
在创建代理集群前，您应该了解一些重要的集群概念。

14.1.1. 代理集群如何平衡消息负载

当代理连接到集群时，AMQ Broker 会自动平衡代理之间的消息负载。这样可确保集群可以维护高消息吞吐量。

考虑四个代理的对称集群。每个代理都配置有名为 OrderQueue 的队列。OrderProducer 客户端连接到 Broker1，并将消息发送到 OrderQueue。Broker1 以轮循方式将消息转发到其他代理。连接到每个代理的 OrderConsumer 客户端会消耗消息。

图 14.1. 消息负载均衡



如果没有消息负载均衡，发送到 Broker1 的消息将保留在 Broker1 上，只有 OrderConsumer1 可以使用它们。

AMQ Broker 默认自动负载均衡消息，将第一个消息组分发到第一个代理，第二个信息组发送到第二个代理。代理启动的顺序决定了哪个代理是第一个、第二个代理，以此类推。

您可以配置：

- 集群将消息负载平衡到具有匹配队列的代理。
- 集群将消息负载平衡到具有活跃消费者匹配队列的代理。
- 集群不进行负载平衡，而是从没有消费者的队列重新分发消息到具有消费者的队列。

- 从队列自动重新分发没有消费者的消息的地址。

其他资源

- 消息负载均衡策略配置有每个代理的以太网连接中的 message-load-balancing 属性。更多信息请参阅 附录 C, *集群连接配置元素*。

- 有关消息重新发布的详情，请参考 第 14.4.2 节 “配置消息重新发布”。

14.1.2. 代理集群如何提高可靠性

代理集群使高可用性和故障转移成为可能，这使其比独立代理更可靠。通过配置高可用性，您可以确保客户端应用程序可以继续发送和接收消息，即使代理遇到失败事件。

借助高可用性，集群中的代理被分组到 live-backup 组中。live-backup 组由一个实时代理组成，它会提供客户端请求，以及等待被动替换 live 代理（如果其失败）的一个或多个备份代理。如果发生故障，备份代理会替换其 live-backup 组中的 live 代理，客户端重新连接并继续其工作。

14.1.3. 集群限制

在集群环境中使用 AMQ 代理时应用以下限制。

临时队列

在故障转移期间，如果客户端有使用临时队列的消费者，则会自动重新创建这些队列。重新创建队列名称与原始队列名称不匹配，这会导致消息重新发布失败，并可以在现有的临时队列中保留消息。红帽建议您避免在集群中使用临时队列。例如，使用请求/回复模式的应用程序应该将固定队列用于 JMSReplyTo 地址。

14.1.4. 了解节点 ID

代理 *节点 ID* 是一个全局唯一标识符(GUID)，在首次创建和初始化代理实例时，以编程方式生成代理实例的日志。节点 ID 存储在 server.lock 文件中。节点 ID 用于唯一标识代理实例，无论代理是独立实例还是集群的一部分。实时备份代理对共享相同的节点 ID，因为它们共享相同的日志。

在代理集群中，代理实例（节点）相互连接，并创建网桥和内部"store-and-forward"队列。这些内部队列的名称基于其他代理实例的节点 ID。代理实例还监控与其自身匹配的节点 ID 的集群广播。如果代理标识重复 ID，则代理会在日志中生成警告信息。

当您使用复制高可用性(HA)策略时，启动 master 代理并将 check-for-live-server 设置为 true，搜索使用其节点 ID 的代理。如果 master 代理使用同一节点 ID 查找另一个代理，它不会启动，或者根据 HA 配置启动故障恢复。

节点 ID 是持久的，这意味着它在代理重启后保留。但是，如果您删除了代理实例（包括其日志），则节点 ID 也会被永久删除。

其他资源

- *有关配置复制策略的更多信息，请参阅配置复制高可用性。*

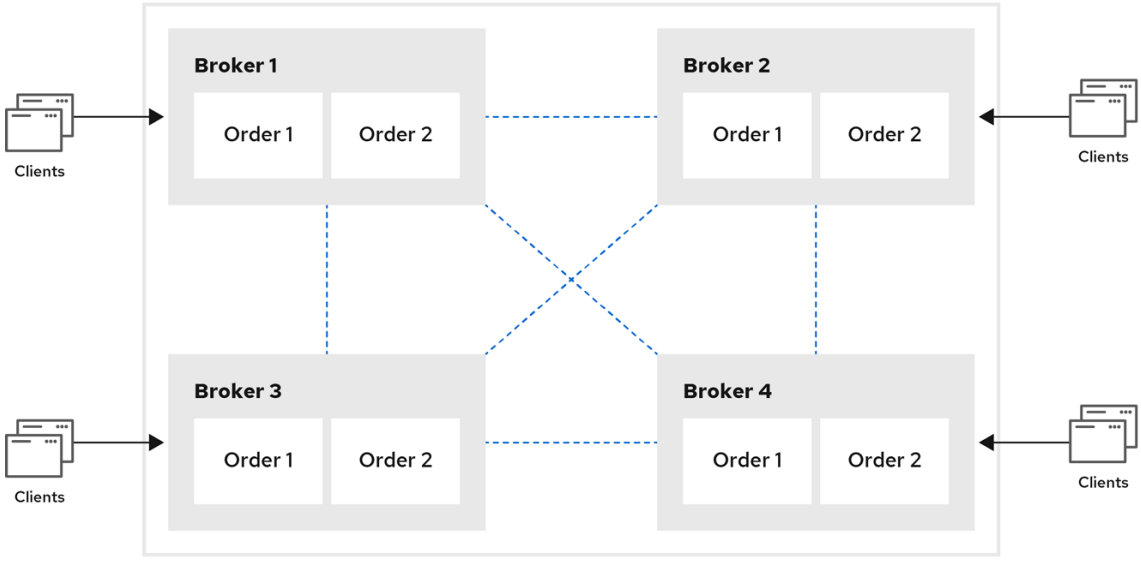
14.1.5. 常见代理集群拓扑

您可以连接代理来形成 对称 或 链 集群拓扑。您实现的拓扑取决于您的环境和消息传递要求。

对称集群

在对称集群中，每个代理都连接到所有其他代理。这意味着，每个代理都不止一个跃点，而每个代理都没有其他代理。

图 14.2. 对称集群



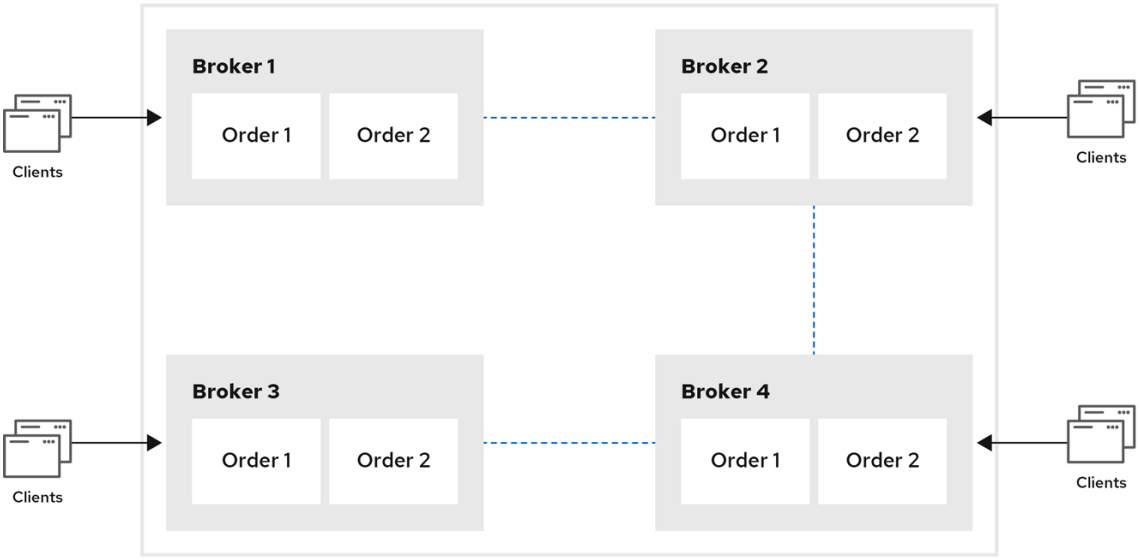
对称集群中的每个代理都知道集群中所有其他代理上存在的所有队列，以及侦听这些队列的消费者。因此，对称集群可以比链集群以最佳方式负载均衡和重新分发信息。

对称集群比链集群更容易设置，但很难在阻止代理直接连接的环境中使用网络限制。

链集群

在链集群中，集群中的每个代理都没有直接连接到集群中的每个代理。相反，代理会在链的末尾使用一个代理组成一个代理，所有其他代理只是连接到链中的前和下一个代理。

图 14.3. 链集群



120_AMQ_1120

链集群比对称集群更难以设置，但当代理位于单独的网络上时，可能很有用，且无法直接连接。通过使用链集群，中间代理可以间接连接两个代理，以便消息在它们之间流，即使这两个代理没有直接连接。

14.1.6. 代理发现方法

Discovery 是集群中的代理相互传播其连接详情的机制。AMQ Broker 支持动态发现 和静态发现。

动态发现

集群中的每个代理都通过 UDP 多播或 JGroups 将连接设置广播到其他成员。在这个方法中，每个代理都使用：

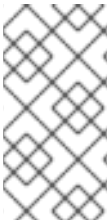
- 一个 广播组，将其集群连接的信息推送到集群的其他潜在成员。
- 用于接收和存储集群中其他代理的集群连接信息的 发现组。

静态发现

如果无法在网络中使用 UDP 或 JGroups，或者要手动指定集群的每个成员，您可以使用静态发现。在这个方法中，通过连接到第二个代理并发送其连接详情，代理"加入集群"。然后，第二个代理将这些详情传播到集群中的其他代理。

14.1.7. 集群大小注意事项

在创建代理集群前，请考虑消息传递吞吐量、拓扑和高可用性要求。这些因素会影响集群中要包含的代理数量。



注意

创建集群后，您可以通过添加和删除代理来调整大小。您可以在不丢失任何信息的情况下添加和删除代理。

消息传递吞吐量

集群应该包含足够的代理来提供您需要的消息传递吞吐量。集群中的更多代理，吞吐量越大。但是，大型集群可能比较复杂。

Topology

您可以创建对称集群或链集群。您选择的拓扑类型会影响您可能需要的代理数量。

更多信息请参阅 第 14.1.5 节 “常见代理集群拓扑”。

高可用性

如果您需要高可用性(HA)，请考虑在创建集群前选择 HA 策略。HA 策略会影响集群的大小，因为每个主代理都应该至少有一个从代理。

更多信息请参阅 第 14.3 节 “实施高可用性”。

14.2. 创建代理集群

您可以通过在应参与集群的每个代理上配置集群连接来创建代理集群。集群连接定义代理应如何连接到其他代理。

您可以创建一个使用静态发现或动态发现的代理集群(UDP 多播或 JGroups)。

先决条件

- 您应该已确定代理集群的大小。

  更多信息请参阅 第 14.1.7 节 “集群大小注意事项”。

14.2.1. 使用静态发现创建代理集群

您可以通过指定静态代理列表来创建代理集群。如果您无法对网络使用 UDP 多播或 JGroups，请使用此静态发现方法。

流程

1. 打开 <broker_instance_dir> /etc/broker.xml 配置文件。

2. 在 &lt ;core& gt; 元素中，添加以下连接器：

   - 定义其他代理如何连接这个连接器

   - 定义此代理如何连接到集群中的其他代理的一个或多个连接器

```
<configuration>
  <core>
    ...
    <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector> 1
      <connector name="broker2">tcp://localhost:61618</connector> 2
      <connector name="broker3">tcp://localhost:61619</connector>
    </connectors>
    ...
  </core>
</configuration>
```

1. 这个连接器定义了其他代理可以用来连接这个连接的连接信息。此信息将在发现过程中发送到集群中的其他代理。

2

broker2 和 broker3 连接器定义此代理如何连接到集群中的两个其他代理，其中一个将始终可用。如果集群中存在其他代理，则在进行初始连接时，这些连接器将发现它们。

有关连接器的更多信息，请参阅 第 2.3 节 “关于连接器”。

3. 添加集群连接并将其配置为使用静态发现。

默认情况下，集群连接将对对称拓扑中所有地址加载平衡消息。

```
<configuration>
  <core>
    ...
    <cluster-connections>
      <cluster-connection name="my-cluster">
        <connector-ref>netty-connector</connector-ref>
        <static-connectors>
          <connector-ref>broker2-connector</connector-ref>
          <connector-ref>broker3-connector</connector-ref>
        </static-connectors>
      </cluster-connection>
    </cluster-connections>
    ...
  </core>
</configuration>
```

cluster-connection

使用 name 属性指定集群连接的名称。

connector-ref

定义其他代理如何连接这个连接器的连接器。

static-connectors

此代理可以使用的一个或多个连接器来与集群中的另一个代理进行初始连接。进行此初始连接后，代理将发现集群中的其他代理。只有集群使用静态发现时，您只需要配置此属性。

4. 配置集群连接的任何其他属性。

这些额外集群连接属性具有适合大多数常见用例的默认值。因此，如果您不希望默认行为，您只需要配置这些属性。更多信息请参阅 附录 C, 集群连接配置元素。

5. 创建集群用户和密码。

AMQ Broker 附带默认集群凭证，但您应该更改它们，以防止未经授权的远程客户端使用这些默认凭证来连接到代理。

重要

集群密码在集群的每个代理中必须相同。

```
<configuration>
  <core>
    ...
    <cluster-user>cluster_user</cluster-user>
    <cluster-password>cluster_user_password</cluster-password>
    ...
  </core>
</configuration>
```

6. 在每个额外代理中重复此步骤。

您可以将集群配置复制到每个额外代理中。但是，不要复制任何其他 AMQ Broker 数据文件（如绑定、日志和大型消息目录）。这些文件在集群中的节点之间必须是唯一的，或者集群无法正确组成。

其他资源

- 有关使用静态发现的代理集群示例，请参阅 clustered-static-discovery 示例。

14.2.2. 使用基于 UDP 的动态发现创建代理集群

您可以创建一个代理集群，代理通过 UDP 多播相互动态发现。

流程

1. 打开 <broker_instance_dir> /etc/broker.xml 配置文件。

2. 在 &lt ;core> 元素中，添加连接器。

这个连接器定义了其他代理可以用来连接这个连接的连接信息。此信息将在发现过程中发送到集群中的其他代理。

```
<configuration>
  <core>
    ...
    <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector>
    </connectors>
    ...
  </core>
</configuration>
```

3. 添加 UDP 广播组。

broadcast 组可让代理将集群连接的信息推送到集群中的其他代理。这个广播组使用 UDP 来广播连接设置：

```
<configuration>
  <core>
    ...
    <broadcast-groups>
      <broadcast-group name="my-broadcast-group">
        <local-bind-address>172.16.9.3</local-bind-address>
        <local-bind-port>-1</local-bind-port>
        <group-address>231.7.7.7</group-address>
        <group-port>9876</group-port>
        <broadcast-period>2000</broadcast-period>
        <connector-ref>netty-connector</connector-ref>
      </broadcast-group>
    </broadcast-groups>
    ...
  </core>
</configuration>
```

除非另有说明，否则需要以下参数：

broadcast-group

使用 name 属性为广播组指定唯一名称。

local-bind-address

绑定到 UDP 套接字的地址。如果您的代理中有多个网络接口，您应该指定您要用于广

播的网络接口。如果没有指定此属性，则套接字将绑定到操作系统选择的 IP 地址。这是特定于 UDP 的属性。

local-bind-port

数据报套接字绑定到的端口。在大多数情况下，使用默认值 -1，它指定一个匿名端口。这个参数用于与 local-bind-address 的连接。这是特定于 UDP 的属性。

group-address

数据要广播的多播地址。它是范围 224.0.0.0 - 239.255.255.255 中的类 D IP 地址。地址 224.0.0.0 是保留的，不可用。这是特定于 UDP 的属性。

group-port

用于广播的 UDP 端口号。这是特定于 UDP 的属性。

broadcast-period （可选）

连续广播之间的间隔（毫秒）。默认值为 2000 毫秒。

connector-ref

以前配置应该广播的集群连接器。

4. 添加 UDP 发现组。

发现组定义此代理如何从其他代理接收连接器信息。代理维护一个连接器列表（每个代理有一个条目）。当它从代理接收广播时，它会更新其条目。如果它没有在较长时间内收到代理广播，它会删除该条目。

此发现组使用 UDP 来发现集群中的代理：

```
<configuration>
   <core>
      ...
      <discovery-groups>
         <discovery-group name="my-discovery-group">
            <local-bind-address>172.16.9.7</local-bind-address>
            <group-address>231.7.7.7</group-address>
            <group-port>9876</group-port>
            <refresh-timeout>10000</refresh-timeout>
         </discovery-group>
      <discovery-groups>
```

```
      ...
   </core>
</configuration>
```

除非另有说明，否则需要以下参数：

discovery-group

使用 **name** 属性为发现组指定唯一名称。

local-bind-address （可选）

如果运行代理的机器使用多个网络接口，您可以指定发现组应侦听的网络接口。这是特定于 **UDP** 的属性。

group-address

要侦听的组的多播地址。它应该与您要从中侦听的广播组中的 **group-address** 匹配。这是特定于 **UDP** 的属性。

group-port

多播组的 **UDP** 端口号。它应该与您要从中侦听的广播组中的 **group-port** 匹配。这是特定于 **UDP** 的属性。

refresh-timeout （可选）

从特定代理中删除该代理的连接器对条目前，发现组从特定代理接收最后一次广播后等待的时间（以毫秒为单位）。默认值为 **10000** 毫秒(**10** 秒)。

把它设置为高于广播组上的 **broadcast-period** 的值。否则，代理可能会定期从列表中消失，即使它们仍然被广播（因为时间略有差异）。

5. 创建集群连接并将其配置为使用动态发现。

默认情况下，集群连接将对对称拓扑中所有地址加载平衡消息。

```
<configuration>
   <core>
      ...
      <cluster-connections>
         <cluster-connection name="my-cluster">
            <connector-ref>netty-connector</connector-ref>
            <discovery-group-ref discovery-group-name="my-discovery-group"/>
         </cluster-connection>
```

```
        </cluster-connections>
        ...
    </core>
</configuration>
```

cluster-connection

使用 name 属性指定集群连接的名称。

connector-ref

定义其他代理如何连接这个连接器的连接器。

discovery-group-ref

此代理应使用的发现组来定位集群的其他成员。只有集群使用动态发现功能时，您只需要配置此属性。

6. 配置集群连接的任何其他属性。

这些额外集群连接属性具有适合大多数常见用例的默认值。因此，如果您不希望默认行为，您只需要配置这些属性。更多信息请参阅 附录 C, 集群连接配置元素。

7. 创建集群用户和密码。

AMQ Broker 附带默认集群凭证，但您应该更改它们，以防止未经授权的远程客户端使用这些默认凭证来连接到代理。

重要

集群密码在集群的每个代理中必须相同。

```
<configuration>
    <core>
        ...
        <cluster-user>cluster_user</cluster-user>
        <cluster-password>cluster_user_password</cluster-password>
        ...
    </core>
</configuration>
```

8. 在每个额外代理中重复此步骤。

您可以将集群配置复制到每个额外代理中。但是，不要复制任何其他 AMQ Broker 数据文件（如绑定、日志和大型消息目录）。这些文件在集群中的节点之间必须是唯一的，或者集群无法正确组成。

其他资源

- 有关使用 UDP 动态发现的代理集群配置示例，请查看 clustered-queue 示例。

14.2.3. 使用基于 JGroups 的动态发现创建代理集群

如果您已在您的环境中使用 JGroups，您可以使用它创建一个代理集群，代理器可以动态发现其他代理集群。

先决条件

- 必须安装并配置 JGroups。

  有关 JGroups 配置文件的示例，请参阅 clustered-jgroups 示例。

流程

1. 打开 <broker_instance_dir> /etc/broker.xml 配置文件。

2. 在 &lt ;core> 元素中，添加连接器。

   这个连接器定义了其他代理可以用来连接这个连接的连接信息。此信息将在发现过程中发送到集群中的其他代理。

```
<configuration>
  <core>
    ...
    <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector>
    </connectors>
    ...
  </core>
</configuration>
```

3.

在 &lt ;core> 元素内，添加 JGroups 广播组。

broadcast 组可让代理将集群连接的信息推送到集群中的其他代理。这个广播组使用 JGroups 来广播连接设置：

```
<configuration>
   <core>
      ...
      <broadcast-groups>
         <broadcast-group name="my-broadcast-group">
            <jgroups-file>test-jgroups-file_ping.xml</jgroups-file>
            <jgroups-channel>activemq_broadcast_channel</jgroups-channel>
            <broadcast-period>2000</broadcast-period>
            <connector-ref>netty-connector</connector-ref>
         </broadcast-group>
      </broadcast-groups>
      ...
   </core>
</configuration>
```

除非另有说明，否则需要以下参数：

broadcast-group

使用 name 属性为广播组指定唯一名称。

JGroups-file

用于初始化 JGroups 通道的 JGroups 配置文件的名称。该文件必须位于 Java 资源路径中，以便代理可以加载它。

jgroups-channel

要连接到广播的 JGroups 通道的名称。

broadcast-period （可选）

连续广播之间的间隔（毫秒）。默认值为 2000 毫秒。

connector-ref

以前配置应该广播的集群连接器。

4.

添加 JGroups 发现组。

发现组定义如何接收连接器信息。代理维护一个连接器列表（每个代理有一个条目）。当它从代理接收广播时，它会更新其条目。如果它没有在较长时间内收到代理广播，它会删除该条目。

此发现组使用 JGroups 来发现集群中的代理：

```
<configuration>
   <core>
      ...
      <discovery-groups>
         <discovery-group name="my-discovery-group">
            <jgroups-file>test-jgroups-file_ping.xml</jgroups-file>
            <jgroups-channel>activemq_broadcast_channel</jgroups-channel>
            <refresh-timeout>10000</refresh-timeout>
         </discovery-group>
      <discovery-groups>
      ...
   </core>
</configuration>
```

除非另有说明，否则需要以下参数：

discovery-group

使用 name 属性为发现组指定唯一名称。

JGroups-file

用于初始化 JGroups 通道的 JGroups 配置文件的名称。该文件必须位于 Java 资源路径中，以便代理可以加载它。

jgroups-channel

用于接收广播的 JGroups 通道的名称。

refresh-timeout （可选）

从特定代理中删除该代理的连接器对条目前，发现组从特定代理接收最后一次广播后等待的时间（以毫秒为单位）。默认值为 10000 毫秒(10 秒)。

把它设置为高于广播组上的 broadcast-period 的值。否则，代理可能会定期从列表中消失，即使它们仍然被广播（因为时间略有差异）。

5. 创建集群连接并将其配置为使用动态发现。

默认情况下，集群连接将对对称拓扑中所有地址加载平衡消息。

```
<configuration>
   <core>
      ...
      <cluster-connections>
         <cluster-connection name="my-cluster">
            <connector-ref>netty-connector</connector-ref>
            <discovery-group-ref discovery-group-name="my-discovery-group"/>
         </cluster-connection>
      </cluster-connections>
      ...
   </core>
</configuration>
```

cluster-connection

使用 name 属性指定集群连接的名称。

connector-ref

定义其他代理如何连接这个连接器的连接器。

discovery-group-ref

此代理应使用的发现组来定位集群的其他成员。只有集群使用动态发现功能时，您只需要配置此属性。

6. 配置集群连接的任何其他属性。

这些额外集群连接属性具有适合大多数常见用例的默认值。因此，如果您不希望默认行为，您只需要配置这些属性。更多信息请参阅 *附录 C, 集群连接配置元素*。

7. 创建集群用户和密码。

AMQ Broker 附带默认集群凭证，但您应该更改它们，以防止未经授权的远程客户端使用这些默认凭证来连接到代理。

重要

集群密码在集群的每个代理中必须相同。

```
<configuration>
  <core>
    ...
    <cluster-user>cluster_user</cluster-user>
    <cluster-password>cluster_user_password</cluster-password>
    ...
  </core>
</configuration>
```

8. 在每个额外代理中重复此步骤。

您可以将集群配置复制到每个额外代理中。但是，不要复制任何其他 AMQ Broker 数据文件（如绑定、日志和大型消息目录）。这些文件在集群中的节点之间必须是唯一的，或者集群无法正确组成。

其他资源

- 有关通过 JGroups 使用动态发现的代理集群示例，请参阅 clustered-jgroups 示例。

14.3. 实施高可用性

您可以通过实施高可用性(HA)来提高其可靠性，即使一个或多个代理离线，启用代理集群也会继续运行。

实现 HA 涉及几个步骤：

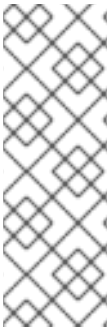
1. 为您的 HA 实现配置代理集群，如 第 14.2 节 “创建代理集群” 所述。

2. 您应该了解 live-backup 组是什么，然后选择最适合您的要求的 HA 策略。请参阅 了解 HA 在 AMQ Broker 中的工作方式。

3. 当您选择了合适的 HA 策略时，在集群中的每个代理上配置 HA 策略。请参阅：

- 配置共享存储高可用性
- 配置复制高可用性
- 使用实时配置有限的高可用性
- 使用 colocated 备份配置高可用性

4. 将您的客户端应用程序配置为使用故障切换。



注意

在稍后需要对为高可用性配置的代理集群进行故障排除时，建议您在集群中运行代理的每个 Java 虚拟机(JVM)实例启用 Garbage Collection (GC)日志记录。要了解如何在 JVM 上启用 GC 日志，请参阅您的 JVM 使用的 Java Development Kit (JDK)版本的官方文档。有关 AMQ Broker 支持的 JVM 版本的更多信息，请参阅 Red Hat AMQ 7 支持的配置。

14.3.1. 了解高可用性

在 AMQ Broker 中，您可以通过将集群中的代理分组到 live-backup 组中来实现高可用性(HA)。在 live-backup 组中，实时代理链接到备份代理，如果实时代理失败，这个代理可能会接管。AMQ Broker 还在 live-backup 组中为故障转移（称为 HA 策略）提供了几种不同的策略。

14.3.1.1. live-backup 组提供高可用性

在 AMQ Broker 中，您可以通过将集群中的代理链接为 实时备份组 来实现高可用性(HA)。实时备份组提供 故障切换，这意味着当一个代理失败时，另一个代理可以接管其消息处理。

live-backup 组由一个实时代理（有时也称为 主 代理）组成，链接到一个或多个备份代理（有时称为从代理）。live 代理服务客户端请求，而备份代理则以被动模式等待。如果 live 代理失败，备份代理替换了 live 代理，使客户端能够重新连接并继续其工作。

14.3.1.2. 高可用性策略

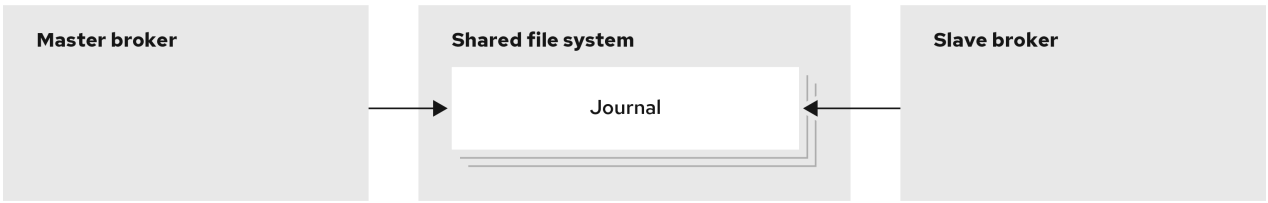
高可用性(HA)策略定义在 live-backup 组中进行故障转移的方式。AMQ Broker 提供几个不同的 HA

策略：

共享存储（推荐）

实时和备份代理将其消息传递数据存储在共享文件系统上的通用目录中；通常是存储区域网络(SAN)或网络文件系统(NFS)服务器。如果您配置了基于 JDBC 的持久性，您还可以将代理数据存储在指定的数据库中。使用共享存储时，如果实时代理失败，备份代理会从共享存储中加载消息数据，并接管失败的实时代理。

在大多数情况下，您应该使用共享存储而不是复制。因为共享存储不会通过网络复制数据，所以它通常比复制提供更好的性能。共享存储还避免了实时代理及其备份同时存在的网络隔离（也称为"脑裂"）问题。

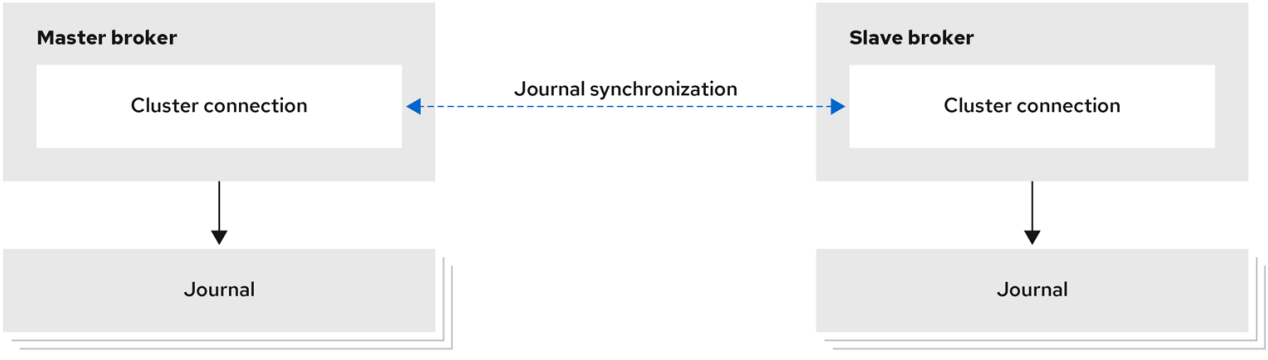


复制

实时和备份代理持续通过网络同步其消息传递数据。如果 live 代理失败，备份代理会加载同步的数据，并接管失败的 live 代理。

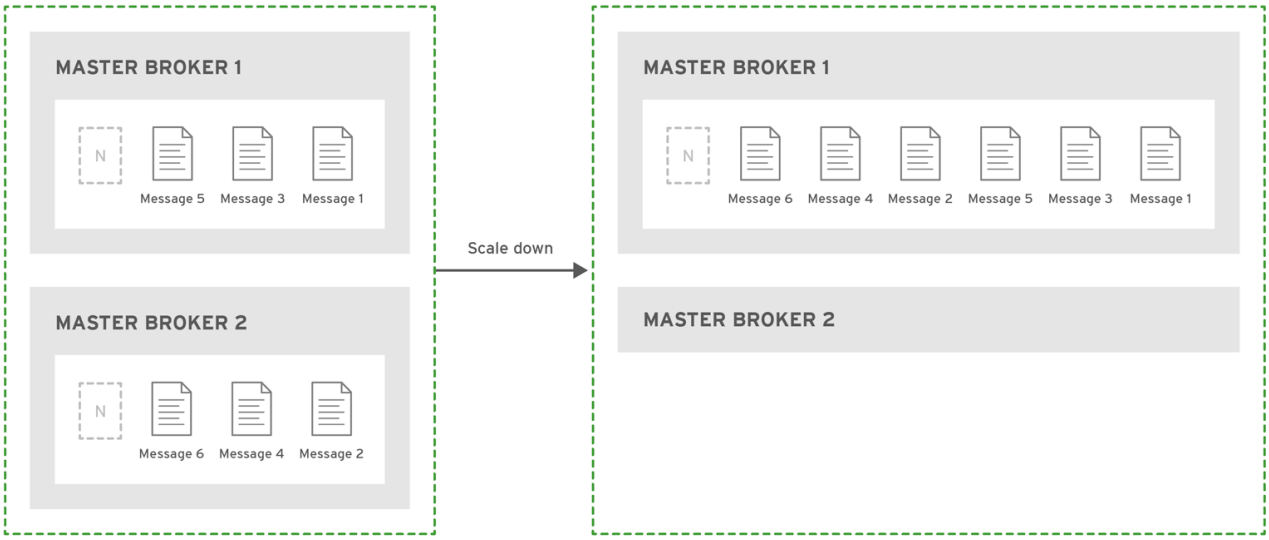
live 和 backup 代理之间的数据同步可确保当实时代理失败时不会丢失消息传递数据。当 live 和 backup 代理最初加入在一起时，实时代理将其所有现有数据复制到备份代理中。此初始阶段完成后，实时代理会将持久数据复制到备份代理中，因为 live 代理接收它。这意味着，如果 live 代理丢弃了网络，备份代理会具有所有当前代理收到的持久性数据。

由于复制通过网络同步数据，网络故障可能会导致实时代理及其备份同时存在网络隔离。



仅限实时（限 HA）

当实时代理被安全停止时，它会将消息和事务状态复制到另一个实时代理，然后关闭。然后，客户端可以重新连接到其他代理，以继续发送和接收信息。



其他资源

- 有关 live-backup 组中代理之间共享的持久性消息数据的更多信息，请参阅 第 6.1 节 “在日志中保留消息数据”。

14.3.1.3. 复制策略限制

当您使用复制来提供高可用性时，实时和备份代理可以同时存在风险，这称为"脑裂"。

如果实时代理及其备份丢失了连接，则发生脑裂。在这种情况下，实时代理及其备份可以同时变为活动状态。因为在这种情况下，代理之间没有消息复制，所以它们各自为客户端和处理消息提供服务，而无需其他了解它。在这种情况下，每个代理都有完全不同的日志。从这种情况中恢复可能非常困难，在某些情况下无法进行。

- 要 消除 脑裂的可能性，请使用 共享存储 HA 策略。
- 如果使用复制 HA 策略，请执行以下步骤来降低脑裂发生的风险。

如果您希望代理使用 ZooKeeper Coordination 服务来协调代理，请在至少三个节点上部署 ZooKeeper。如果代理丢失了与一个 ZooKeeper 节点的连接，至少三个节点可确保在实时备份代理对遇到复制中断时，大多数节点可以协调代理。

如果要使用嵌入式代理协调，它使用集群中的其他可用代理提供仲裁投票，这可以减少（但不会完全消除）遇到脑裂的可能性（最少三个 live-backup 对）。至少使用三个实时备份对可确保在实时备份代理对遇到复制中断时发生的任何仲裁投票中可实现大多数结果。

使用复制 HA 策略时还有一些额外的注意事项，如下所述：

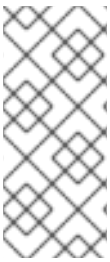
- 当 live 代理失败且备份转换为 live 时，不会进行进一步的复制，直到新的备份代理附加到 live 代理，或恢复到原始 live 代理。
- 如果 live-backup 组中的备份代理失败，则 live 代理将继续提供信息。但是，在添加另一个代理作为备份或重启原始备份代理前，消息不会被复制。在此期间，信息只会保留在 live 代理中。
- 如果代理使用嵌入的代理协调以及 live-backup 对中的代理，以避免消息丢失，您必须首先重启最新的活跃代理。如果最近活跃的代理是备份代理，您需要手动将这个代理重新配置为 master 代理，以便首先重启它。

14.3.2. 配置共享存储高可用性

您可以使用共享存储高可用性(HA)策略在代理集群中实现 HA。使用共享存储时，实时和备份代理都可以访问共享文件系统上的通用目录；通常是存储区域网络(SAN)或网络文件系统(NFS)服务器。如果您配置了基于 JDBC 的持久性，您还可以将代理数据存储在指定的数据库中。使用共享存储时，如果实时代理失败，备份代理会从共享存储中加载消息数据，并接管失败的实时代理。

一般情况下，SAN 可提供更好的性能（例如，速度）与 NFS 服务器，并且是推荐的选项（如果可用）。如果您需要使用 NFS 服务器，请参阅 Red Hat AMQ 7 支持的配置 以了解有关 AMQ Broker 支持的网络文件系统的更多信息。

在大多数情况下，您应该使用共享存储 HA 而不是复制。因为共享存储不会通过网络复制数据，所以它通常比复制提供更好的性能。共享存储还避免了实时代理及其备份同时存在的网络隔离（也称为"脑裂"）问题。



注意

使用共享存储时，备份代理的启动时间取决于消息日志的大小。当备份代理接管失败的 live 代理时，它会从共享存储中加载日志。如果日志包含大量数据，这个过程可能会非常耗时。

14.3.2.1. 配置 NFS 共享存储

使用共享存储高可用性时，您必须配置 live 和 backup 代理，以使用共享文件系统上的通用目录。通常，您使用 Storage Area Network (SAN)或网络文件系统(NFS)服务器。

在每个代理机器实例上从 NFS 服务器挂载导出的目录时，下面列出了一些推荐的配置选项。

sync

指定所有更改都会立即刷新到磁盘。

intr

如果服务器已关闭或无法访问，允许中断 NFS 请求。

noac

禁用属性缓存。需要在多个客户端间实现属性缓存一致性。

soft

指定如果 NFS 服务器不可用，则应报告错误，而不是等待服务器恢复在线。

lookupcache=none

禁用查找缓存。

timeo=n

NFS 客户端（即代理）在重试请求前等待来自 NFS 服务器的响应（以秒为单位）的时间。对于 TCP 上的 NFS，默认的 timeo 值为 600 (60 秒)。对于 UDP 上的 NFS，客户端使用自适应算法为常用请求类型（如读取和写入请求类型）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。一个默认的 timeo 等待时间为 600 deciseconds（60 秒），retrans 值为 5（重试 5 次），可能会导致 AMQ Broker 需要 5 分钟的等待时间来检测到一个 NFS 已断开连接。

其他资源

- *要了解如何从 NFS 服务器挂载导出的目录，请参阅 Red Hat Enterprise Linux 文档中的使用 mount 挂载 NFS 共享。*
- *有关 AMQ Broker 支持的网络文件系统的详情，请参考 Red Hat AMQ 7 支持的配置。*

14.3.2.2. 配置共享存储高可用性

此流程演示了如何为代理集群配置共享存储高可用性。

先决条件

- *共享存储系统必须可以被 live 和 backup 代理访问。*
  - *通常，您可以使用存储区域网络(SAN)或网络文件系统(NFS)服务器来提供共享存储。有关支持的网络文件系统的更多信息，请参阅 Red Hat AMQ 7 支持的配置。*
  - *如果您配置了基于 JDBC 的持久性，您可以使用指定的数据库提供共享存储。要了解如何配置 JDBC 持久性，请参阅 第 6.2 节 “在数据库中持久保留消息数据”。*

流程

1. *将集群中的代理分组到 live-backup 组中。*

   在大多数情况下，live-backup 组应该由两个代理组成：一个实时代理和一个备份代理。如果集群中有六个代理，则需要三个 live-backup 组。

2. *创建一个由一个 live 代理和一个备份代理组成的第一个 live-backup 组。*

   a. *打开 live 代理的 < broker_instance_dir> /etc/broker.xml 配置文件。*

   b. *如果使用：*

      i.

提供共享存储的网络文件系统，验证实时代理的分页、绑定、日志和大型消息目录是否指向备份代理也可以访问的共享位置。

```
<configuration>
   <core>
      ...
      <paging-directory>../sharedstore/data/paging</paging-directory>
      <bindings-directory>../sharedstore/data/bindings</bindings-directory>
      <journal-directory>../sharedstore/data/journal</journal-directory>
      <large-messages-directory>../sharedstore/data/large-messages</large-
messages-directory>
      ...
   </core>
</configuration>
```

ii.

一个提供共享存储的数据库，请确保 master 和 backup 代理都可以连接到同一数据库，并在 broker.xml 配置文件的 database-store 元素中指定相同的配置。示例配置如下所示：

```
<configuration>
  <core>
    <store>
      <database-store>
        <jdbc-connection-url>jdbc:oracle:data/oracle/database-
store;create=true</jdbc-connection-url>
        <jdbc-user>ENC(5493dd76567ee5ec269d11823973462f)</jdbc-user>
        <jdbc-password>ENC(56a0db3b71043054269d11823973462f)</jdbc-
password>
        <bindings-table-name>BIND_TABLE</bindings-table-name>
        <message-table-name>MSG_TABLE</message-table-name>
        <large-message-table-name>LGE_TABLE</large-message-table-name>
        <page-store-table-name>PAGE_TABLE</page-store-table-name>
        <node-manager-store-table-name>NODE_TABLE<node-manager-store-table-
name>
        <jdbc-driver-class-name>oracle.jdbc.driver.OracleDriver</jdbc-driver-class-
name>
        <jdbc-network-timeout>10000</jdbc-network-timeout>
        <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
        <jdbc-lock-expiration>15000</jdbc-lock-expiration>
        <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
      </database-store>
    </store>
  </core>
</configuration>
```

c.

配置 live 代理，以将共享存储用于其 HA 策略。

```
<configuration>
   <core>
      ...
```

```
      <ha-policy>
        <shared-store>
          <master>
            <failover-on-shutdown>true</failover-on-shutdown>
          </master>
        </shared-store>
      </ha-policy>
      ...
    </core>
  </configuration>
```

failover-on-shutdown

如果正常停止这个代理，此属性控制备份代理是否应该变为实时状态，并接管。

d. 打开备份代理的 < broker_instance_dir> /etc/broker.xml 配置文件。

e. 如果使用：

i. 提供共享存储的网络文件系统，验证备份代理的分页、绑定、日志和大型消息目录是否指向与实时代理相同的共享位置。

```
<configuration>
  <core>
    ...
    <paging-directory>../sharedstore/data/paging</paging-directory>
    <bindings-directory>../sharedstore/data/bindings</bindings-directory>
    <journal-directory>../sharedstore/data/journal</journal-directory>
    <large-messages-directory>../sharedstore/data/large-messages</large-
messages-directory>
    ...
  </core>
</configuration>
```

ii. 一个提供共享存储的数据库，请确保 master 和备份代理都可以连接到同一数据库，并在 broker.xml 配置文件的 database-store 元素中指定相同的配置。

f. 配置备份代理，以将共享存储用于其 HA 策略。

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <slave>
```

```
                <failover-on-shutdown>true</failover-on-shutdown>
                <allow-failback>true</allow-failback>
                <restart-backup>true</restart-backup>
            </slave>
        </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

failover-on-shutdown

如果这个代理已变为 live，然后被正常停止，则此属性控制备份代理（原始实时代理）是否应实时并接管。

allow-failback

如果发生故障转移，并且备份代理已接管用于 live 代理，则此属性控制备份代理在重启并重新连接到集群时是否应返回到原始 live 代理。

注意

故障恢复用于实时备份对（一个实时代理通过单一备份代理对）。如果 live 代理配置了多个备份，则不会进行故障恢复。相反，如果发生故障转移事件，备份代理将变为实时，下一个备份将变为备份。当原始的 live 代理重新上线时，它将无法启动故障恢复，因为现在处于活动状态的代理已具有备份。

restart-backup

此属性控制备份代理在备份代理失败后是否自动重启。此属性的默认值为 **true**。

3. 对集群中的每个剩余的 live-backup 组重复步骤 2。

14.3.3. 配置复制高可用性

您可以使用复制高可用性(HA)策略在代理集群中实现 HA。通过复制，持久数据会在实时和备份代理之间同步。如果实时代理遇到失败，消息数据会同步到备份代理中，并接管失败的实时代理。

如果没有共享文件系统，您应该使用复制作为共享存储的替代选择。但是，复制可能会导致实时代理及其备份同时成为实时代理。

注意

由于实时和备份代理必须通过网络同步其消息传递数据，复制会增加性能开销。此同步过程会阻止日志操作，但它不会阻止客户端。您可以配置日志操作在数据同步时阻止的最大时间。

如果 live-backup 代理对之间的复制连接中断，代理需要通过协调来确定实时代理是否处于活动状态，或者需要故障转移到备份代理。要提供此协调，您可以将代理配置为使用以下协调方法之一。

- Apache ZooKeeper 协调服务。
- 嵌入式代理协调，它使用集群中的其他代理提供仲裁投票。

14.3.3.1. 选择协调方法

红帽建议您使用 Apache ZooKeeper 协调服务来协调代理激活。在选择协调方法时，了解基础架构要求的不同以及两个协调方法之间的数据一致性管理非常有用。

基础架构要求

- 如果使用 ZooKeeper 协调服务，您可以使用单个 live-backup 代理对操作。但是，您必须将代理连接到至少 3 个 Apache ZooKeeper 节点，以确保代理在丢失与一个节点的连接时可以继续正常工作。要为代理提供协调服务，您可以共享其他应用程序使用的现有 ZooKeeper 节点。有关设置 Apache ZooKeeper 的更多信息，请参阅 Apache ZooKeeper 文档。
- 如果要使用嵌入式代理协调，它使用集群中的其他可用代理提供仲裁投票，则必须至少有三个 live-backup 代理对。至少使用三个实时备份对可确保当实时备份代理对遇到复制中断时发生的任何仲裁投票中大部分结果。

数据一致性

- 如果您使用 Apache ZooKeeper 协调服务，则 ZooKeeper 会跟踪每个代理上的数据版本，因此只有具有最新日志数据的代理可以激活为 live 代理，无论代理是否被配置为主代理或备份代理用于复制目的。版本跟踪消除了代理可以使用过时的日志激活的可能性，并启动服务客户端。
-

如果您使用嵌入的代理协调，则不存在任何机制来跟踪每个代理上的数据版本，以确保只有具有最新日志的代理才可以成为 live 代理。因此，对于具有过时的日志的代理成为实时和启动服务客户端，这导致日志中出现混乱。

14.3.3.2. 在复制中断后代理如何协调

本节介绍复制连接中断后这两个协调方法的工作方式。

使用 ZooKeeper 协调服务

如果您使用 ZooKeeper 协调服务来管理复制中断，则这两个代理都必须连接到多个 Apache ZooKeeper 节点。

- 如果任何时候，实时代理丢失与大多数 ZooKeeper 节点的连接，它会关闭以避免出现 "split brain" 的风险。
- 如果任何时候，备份代理丢失了与大多数 ZooKeeper 节点的连接，它会停止接收复制数据并等待它连接到大多数 ZooKeeper 节点，然后再再次作为备份代理。当连接恢复到大多数 ZooKeeper 节点时，备份代理使用 ZooKeeper 来确定它是否需要丢弃其数据并搜索要从中复制的实时代理，或使用其当前数据成为实时代理。

ZooKeeper 使用以下控制机制来管理故障转移过程：

- 在任何时间点上只能归单个实时代理所有的共享租期锁定。
- 跟踪代理数据最新版本的激活序列计数器。每个代理会跟踪其服务器锁定文件中存储的本地计数器中的日志数据版本，及其 NodeID。live 代理还会在 ZooKeeper 上的协调激活序列计数器中共享其版本。

如果 live 代理和备份代理之间的复制连接丢失，则 live 代理会同时增加其本地激活序列计数器值，并在 ZooKeeper 上增加协调激活序列计数器值（1）来公告它具有最新数据。现在，备份代理的数据被视为 stale，代理无法成为实时代理，直到恢复复制连接并同步最新的数据。

复制连接丢失后，备份代理会检查 ZooKeeper 锁定是否归 live 代理所有，并且 ZooKeeper 上的协调激活序列计数器是否与其本地计数器值匹配。

- *如果锁定由 live 代理所有，则备份代理检测到 ZooKeeper 上的激活序列计数器在复制连接丢失时由 live 代理更新。这表示 live 代理正在运行，因此备份代理不会尝试故障转移。*

- *如果锁定不归 live 代理所有，则 live 代理不会处于活动状态。如果备份代理中的激活序列计数器的值与 ZooKeeper 上的协调激活序列计数器值相同，这表示备份代理有最新的数据，则备份代理会失败。*

- *如果锁定不归 live 代理所有，但备份代理中的激活序列计数器的值小于 ZooKeeper 上的计数器值，则备份代理中的数据不是最新的，备份代理无法故障切换。*

使用嵌入的代理协调

如果 live-backup 代理对使用嵌入式代理协调来协调复制中断，则可以启动以下两个类型的仲裁投票。

vote 类型	描述	initiator	所需的配置	参与者	基于投票结果的操作
备份投票	如果备份代理丢失了与 live 代理的复制连接，备份代理会根据这个投票的结果决定是否启动。	备份代理	无。当备份代理丢失与其复制合作伙伴的连接时，会自动进行备份投票。 但是，您可以通过为这些参数指定自定义值来控制备份投票的属性： • **quorum-vote-wait** • **vote-retries** • **vote-retry-wait**	集群中的其他实时代理	如果备份代理从集群中的其他实时代理接收大多数（即 *仲裁*）投票，这表示其复制合作伙伴不再可用。

vote 类型	描述	initiator	所需的配置	参与者	基于投票结果的操作
实时投票	如果实时代理丢失与其复制合作伙伴的连接，实时代理会根据此投票决定是否继续运行。	实时代理	当实时代理丢失与复制合作伙伴的连接并将 **vote-on-replication-failure** 设置为 **true** 时，会出现实时投票。已激活的备份代理被视为实时代理，并可启动实时投票。	集群中的其他实时代理	如果**没有**从集群中的其他实时代理接收大多数投票，则 live 代理会关闭，这表示其集群连接仍处于活动状态。

重要

下面列出了一些重要事项，请注意代理集群的配置如何影响仲裁投票的行为。

- ***要使仲裁投票成功，集群的大小必须允许实现大多数结果。因此，您的集群应至少有三个 live-backup 代理对。***
- ***您添加到集群中的 live-backup 代理对越多，提高集群的整体容错能力。例如，假设您有三个 live-backup 对。如果您丢失了完整的 live-backup 对，剩余的两个 live-backup 对无法实现大部分情况，从而导致任何后续的仲裁投票。在这种情况下，集群中的任何进一步复制中断都可能会导致实时代理关闭，并阻止其备份代理启动。通过配置集群（例如 5 个代理对），集群可以至少遇到两个故障，同时仍然确保任何仲裁投票中的大多数结果。***
- ***如果您有意减少集群中的 live-backup 代理对数量，则之前为大多数投票创建的阈值不会自动减少。在这个时间中，丢失复制连接触发的任何仲裁投票都无法成功，使集群更容易受到脑裂的影响。要使集群重新计算仲裁投票的最大阈值，请首先关闭您要从集群中删除的实时备份对。然后，重启集群中剩余的 live-backup 对。当所有剩余的代理都已重启时，集群会重新计算仲裁票阈值。***

14.3.3.3. 使用 ZooKeeper 协调服务为代理集群配置复制

您必须在使用 Apache ZooKeeper 协调服务的 live-backup 对中为这两个代理指定相同的复制配置。然后，代理协调来确定哪个代理是主代理，以及备份代理。

先决条件

- ***至少 3 个 Apache ZooKeeper 节点，确保在代理丢失与一个节点的连接时可以继续运行。***

- 代理机器具有类似的硬件规格，即，您没有首选哪个机器运行 live 代理，并在任何时间点运行备份代理。

- ZooKeeper 必须有足够的资源来确保暂停时间明显低于 ZooKeeper 服务器 tick 时间。根据代理的预期负载，请考虑代理和 ZooKeeper 节点是否可以共享同一节点。如需更多信息，请参阅 https://zookeeper.apache.org/。

流程

1. 为 live-backup 对中的两个代理打开 <broker_instance_dir> /etc/broker.xml 配置文件。

2. 为对中的两个代理配置相同的复制配置。例如：

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
        <primary>
          <coordination-id>production-001</coordination-id>
          <manager>
            <properties>
              <property key="connect-string"
value="192.168.1.10:6666,192.168.2.10:6667,192.168.3.10:6668"/>
            </properties>
          </manager>
        </primary>
      </replication>
    </ha-policy>
    ...
  </core>
</configuration>
```

primary

将复制类型配置为 primary，以指示任一代理可以是主代理，具体取决于代理协调的结果。

coordination-id

为 live-backup 对中的两个代理指定一个通用字符串值。具有相同 Coordination-id 字符串的代理，协调激活。在协调过程中，两个代理都使用 Coordination-id 字符串作为节点 Id，并在 ZooKeeper 中尝试获取锁定。第一个获取锁定并有最新数据的代理作为 live 代理启动，其他代理变为备份。

属性

指定 property 元素，您可以在其中指定一组键值对，以提供 ZooKeeper 节点的连接详情：

键	值
connect-string	指定 ZooKeeper 节点的 IP 地址和端口号的逗号分隔列表。例如，**value="192.168.1.10:6666,192.168.2.10:6667,192.168.3.10:6668"**。
session-ms	代理在丢失与大多数 ZooKeeper 节点的连接后等待的持续时间。默认值为 **18000** ms。有效值介于 ZooKeeper 服务器 tick 时间的 2 倍和 20 倍之间。 **注意** ZooKeeper 暂停垃圾回收的时间必须小于 **session-ms** 属性的值的 0.33，以便 ZooKeeper heartbeat 能够可靠地正常工作。如果无法确保暂停时间小于这个限制，请增加每个代理的 **session-ms** 属性的值并接受较慢的故障转移。 **重要** 代理复制合作伙伴每 2 秒自动交换"ping"数据包，以确认合作伙伴代理可用。当备份代理没有收到来自 live 代理的响应时，备份会等待响应，直到代理的连接时间(ttl)过期为止。默认 connection-ttl 为 **60000** ms，这意味着备份代理尝试在 60 秒后失败。建议您将 connection-ttl 值设置为与 **session-ms** 属性值类似的值，以便更快地进行故障转移。要设置新的 connection-ttl，请配置 **connection-ttl-override** 属性。
命名空间（可选）	如果代理与其他应用程序共享 ZooKeeper 节点，您可以创建一个 ZooKeeper 命名空间来存储向代理提供协调服务的文件。您必须为 live-backup 对中的两个代理指定相同的命名空间。

3. *为代理配置任何其他 HA 属性。*

这些额外的 HA 属性具有适合大多数常见用例的默认值。因此，如果您不希望默认行为，您只需要配置这些属性。更多信息请参阅 附录 F, 其他复制高可用性配置元素。

4. *重复步骤 1 到 3，以配置集群中的每个额外的 live-backup 代理对。*

其他资源

- *有关使用 HA 的复制的代理集群示例，请参阅 HA 示例。*

- 有关节点 ID 的更多信息，请参阅了解节点 ID。

14.3.3.4. 使用嵌入式代理协调为复制高可用性配置代理集群

使用嵌入式代理协调进行复制至少需要三个实时备份对来降低（但不消除）"脑裂"的风险。

以下流程描述了如何为 6-broker 集群配置复制高可用性(HA)。在此拓扑中，6 个代理被分组到三个 live-backup 对中：三个实时代理都使用专用的备份代理来对。

先决条件

- 您必须有一个至少 6 个代理的代理集群。

  6 个代理被配置为三个实时备份对。有关在集群中添加代理的更多信息，请参阅 第 14 章 设置代理集群。

流程

1. 将集群中的代理分组到 live-backup 组中。

   在大多数情况下，live-backup 组应该由两个代理组成：一个实时代理和一个备份代理。如果集群中有六个代理，则需要三个 live-backup 组。

2. 创建一个由一个 live 代理和一个备份代理组成的第一个 live-backup 组。

   a. 打开 live 代理的 < broker_instance_dir> /etc/broker.xml 配置文件。

   b. 将 live 代理配置为使用其 HA 策略的复制。

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
        <master>
          <check-for-live-server>true</check-for-live-server>
          <group-name>my-group-1</group-name>
          <vote-on-replication-failure>true</vote-on-replication-failure>
```

```
            ...
          </master>
        </replication>
      </ha-policy>
      ...
    </core>
</configuration>
```

check-for-live-server

如果 live 代理失败，此属性控制客户端在重启时应该会失败。

如果将此属性设置为 true，则当 live 代理在之前的故障切换后重启时，它会搜索具有相同节点 ID 的集群中的另一个代理。如果 live 代理找到另一个具有相同节点 ID 的代理，这表示在 live 代理失败时成功启动备份代理。在这种情况下，live 代理将其数据与备份代理同步。然后，实时代理会请求备份代理关闭。如果为故障恢复配置了备份代理，如下所示，它会关闭。然后，实时代理恢复其活跃角色，客户端会重新连接它。

警告

如果您没有在 live 代理上将 check-for-live-server 设置为 true，则在之前的故障切换后重启 live 代理时可能会遇到重复的消息传递处理。特别是，如果您重启 live 代理，并将此属性设置为 false，则 live 代理不会将数据与其备份代理同步。在这种情况下，实时代理可能会处理备份代理已经处理的相同消息，从而导致重复。

group-name

此 live-backup 组的名称（可选）。要组成 live-backup 组，必须使用相同的组名称配置 live 和 backup 代理。如果没有指定 group-name，则备份代理可以使用任何 live 代理进行复制。

vote-on-replication-failure

此属性控制实时代理是否在中断复制连接时启动名为 live vote 的仲裁投票。

实时投票是一种实时代理，用于判断它还是其合作伙伴是中断复制连接的原因。根据投票的结果，实时代理会保持运行或关闭。

重要

要使仲裁投票成功，集群的大小必须允许实现大多数结果。因此，当您使用复制 HA 策略时，您的集群应至少有三个 live-backup 代理对。

您在集群中配置的更多代理对，提高集群的整体容错能力。例如，假设您有三个 live-backup 代理对。如果您丢失了与完整的 live-backup 对的连接，剩余的两个 live-backup 对无法再实现大部分的仲裁投票。这种情况意味着后续复制中断都可能会导致实时代理关闭，并阻止其备份代理启动。通过配置集群（例如 5 个代理对），集群可以至少遇到两个故障，同时仍然确保任何仲裁投票中的大多数结果。

c. 为 live 代理配置任何其他 HA 属性。

这些额外的 HA 属性具有适合大多数常见用例的默认值。因此，如果您不希望默认行为，您只需要配置这些属性。更多信息请参阅 附录 F, 其他复制高可用性配置元素。

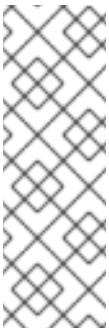
d. 打开备份代理的 < broker_instance_dir> /etc/broker.xml 配置文件。

e. 配置备份代理，以使用其 HA 策略的复制。

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
        <slave>
          <allow-failback>true</allow-failback>
          <group-name>my-group-1</group-name>
          <vote-on-replication-failure>true</vote-on-replication-failure>
          ...
        </slave>
      </replication>
    </ha-policy>
    ...
  </core>
</configuration>
```

allow-failback

如果发生故障转移，并且备份代理已接管用于 live 代理，则此属性控制备份代理在重启并重新连接到集群时是否应返回到原始 live 代理。



注意

故障恢复用于实时备份对（一个实时代理通过单一备份代理对）。如果 live 代理配置了多个备份，则不会进行故障恢复。相反，如果发生故障转移事件，备份代理将变为实时，下一个备份将变为备份。当原始的 live 代理重新上线时，它将无法启动故障恢复，因为现在处于活动状态的代理已具有备份。

group-name

此 live-backup 组的名称（可选）。要组成 live-backup 组，必须使用相同的组名称配置 live 和 backup 代理。如果没有指定 group-name，则备份代理可以使用任何 live 代理进行复制。

vote-on-replication-failure

此属性控制实时代理是否在中断复制连接时启动名为 live vote 的仲裁投票。已激活的备份代理被视为实时代理，并可启动实时投票。

实时投票是一种实时代理，用于判断它还是其合作伙伴是中断复制连接的原因。根据投票的结果，实时代理会保持运行或关闭。

f. （可选）配置备份代理启动的仲裁投票的属性。

```
<configuration>
   <core>
      ...
      <ha-policy>
         <replication>
            <slave>
            ...
               <vote-retries>12</vote-retries>
               <vote-retry-wait>5000</vote-retry-wait>
            ...
            </slave>
         </replication>
      </ha-policy>
      ...
   </core>
</configuration>
```

vote-retries

此属性控制备份代理重试仲裁投票的次数，以便接收允许备份代理启动的大部分结果。

vote-retry-wait

此属性控制备份代理在每次仲裁票重试之间等待的时间（毫秒）。

g. 为备份代理配置任何其他 HA 属性。

这些额外的 HA 属性具有适合大多数常见用例的默认值。因此，如果您不希望默认行为，您只需要配置这些属性。更多信息请参阅 *附录 F, 其他复制高可用性配置元素*。

3. 对集群中的每个额外的 live-backup 组重复步骤 2。

如果集群中有六个代理，请多次重复这个过程；每个剩余的 live-backup 组一次。

其他资源

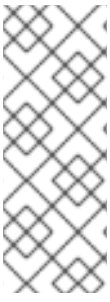
- 有关使用 HA 的复制的代理集群示例，请参阅 *HA 示例*。
- 有关节点 ID 的更多信息，*请参阅了解节点 ID*。

14.3.4. 使用实时配置有限的高可用性

仅限实时 HA 策略允许您在不丢失任何信息的情况下关闭集群中的代理。使用 live-only 时，当实时代理安全停止时，它会将消息和事务状态复制到另一个实时代理，然后关闭。然后，客户端可以重新连接到其他代理，以继续发送和接收信息。

仅限实时 HA 策略只有在代理安全停止时才处理问题单。它无法处理意外的代理失败。

虽然仅实时 HA 会阻止消息丢失，但可能无法保留消息顺序。如果配置了 live-only HA 的代理已停止，则其消息将附加到另一个代理队列的末尾。



注意

当代理准备缩减时，它会在代理断开连接前向客户端发送一条信息，通知代理准备好处理其消息。但是，只有在初始代理完成缩减后，客户端才应重新连接到新代理。这样可确保在客户端重新连接时，其他代理上可以使用任何状态，如队列或事务。客户端重新连接时应用正常重新连接设置，因此您应该设置这些高以处理缩减所需的时间。

这个步骤描述了如何在集群中配置每个代理来缩减。完成此步骤后，每当代理被安全停止后，它将其消息和事务状态复制到集群中的另一个代理中。

流程

1. *打开第一个代理的 < broker_instance_dir> /etc/broker.xml 配置文件。*

2. *将代理配置为使用实时 HA 策略。*

```
<configuration>
  <core>
    ...
    <ha-policy>
      <live-only>
      </live-only>
    </ha-policy>
    ...
  </core>
</configuration>
```

3. *配置缩减代理集群的方法。*

   指定此代理应缩减的代理或代理组。

缩减为...	这些以下操作

集群中的特定代理	指定要缩减的代理连接器。
	
	`<live-only>`
	`  <scale-down>`
	`  <connectors>`
	`  <connector-ref>broker1-connector</connector-ref>`
	`  </connectors>`
	`  </scale-down>`
	`</live-only>`

缩减为...	这些以下操作

集群中的任何代理	指定代理集群的发现组。
	
	`<live-only>`
	`  <scale-down>`
	`  <discovery-group-ref discovery-group-name="my-discovery-group"/>`
	`  </scale-down>`
	`</live-only>`

特定代理组中的代理	指定代理组。
	
	`<live-only>`
	`  <scale-down>`
	`  <group-name>my-group-name</group-name>`
	`  </scale-down>`
	`</live-only>`

4. *对集群中的每个剩余的代理重复此步骤。*

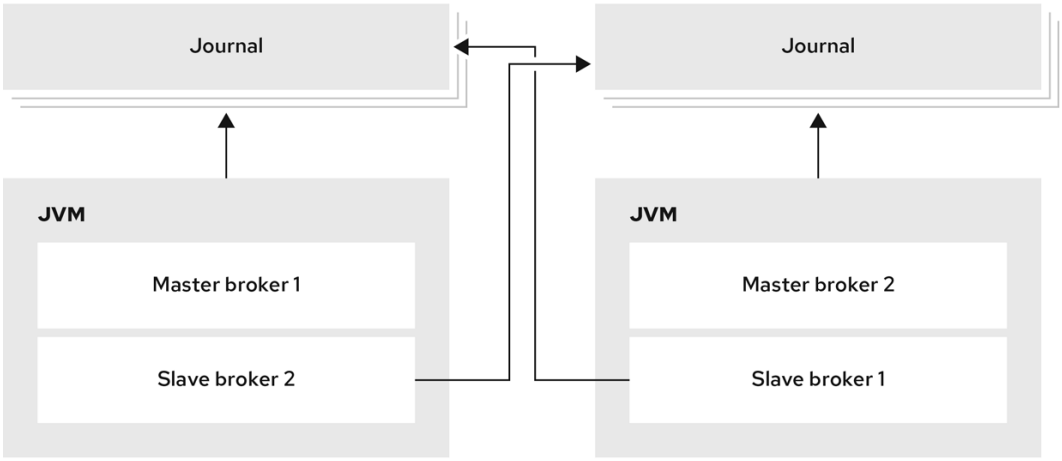
其他资源

- *有关使用 live-only 缩减集群的代理集群示例，请参阅 缩减 示例。*

14.3.5. 使用 colocated 备份配置高可用性

您可以在与另一个实时代理相同的 JVM 中并置备份代理，而不是配置 live-backup 组。在此配置中，每个实时代理都配置为请求另一个实时代理，以便在其 JVM 中创建和启动备份代理。

图 14.4. colocated live 和 backup 代理



120_AMQ_1120

您可以将共处用于共享存储或复制作为高可用性(HA)策略。新的备份代理从创建它的 live 代理继承其配置。备份的名称设置为 colocated_backup_n，其中 n 是实时代理创建的备份数量。

另外，备份代理会继承其连接器的配置，并从创建它的 live 代理中继承接受器。默认情况下，端口偏移 100 应用于每个端口。例如，如果 live 代理具有端口 61616 的接收器，则创建的第一个备份代理将使用端口 61716，第二个备份将使用 61816，以此类推。

日志、大消息和分页的目录会根据您选择的 HA 策略设置。如果选择共享存储，则请求代理会通知目标代理要使用的目录。如果选择了复制，则会从创建代理继承目录，并附加新的备份名称。

此流程将集群中的每个代理配置为使用共享存储 HA，并请求创建备份并与集群中的另一个代理在一起。

流程

1. 打开第一个代理的 < broker_instance_dir> /etc/broker.xml 配置文件。

2. 将代理配置为使用 HA 策略和 colocation。

   在本例中，代理配置了共享存储 HA 和 colocation。

```
<configuration>
  <core>
    ...
    <ha-policy>
```

```
            <shared-store>
               <colocated>
                  <request-backup>true</request-backup>
                  <max-backups>1</max-backups>
                  <backup-request-retries>-1</backup-request-retries>
                  <backup-request-retry-interval>5000</backup-request-retry-interval/>
                  <backup-port-offset>150</backup-port-offset>
                  <excludes>
                     <connector-ref>remote-connector</connector-ref>
                  </excludes>
                  <master>
                     <failover-on-shutdown>true</failover-on-shutdown>
                  </master>
                  <slave>
                     <failover-on-shutdown>true</failover-on-shutdown>
                     <allow-failback>true</allow-failback>
                     <restart-backup>true</restart-backup>
                  </slave>
               </colocated>
            </shared-store>
         </ha-policy>
         ...
      </core>
   </configuration>
```

request-backup

通过将此属性设置为 true，此代理将请求一个备份代理由集群中的另一个实时代理创建。

max-backups

此代理可以创建的备份代理数量。如果将此属性设置为 0，则此代理不接受来自集群中其他代理的备份请求。

backup-request-retries

此代理应尝试请求要创建的备份代理的次数。默认值为 -1， 即无限尝试。

backup-request-retry-interval

代理在重试请求创建备份代理前应等待的时间（毫秒）。默认值为 5000 或 5 秒。

backup-port-offset

用于新备份代理的接收器和连接器的端口偏移。如果此代理收到为集群中的另一个代理创建备份的请求，它将使用这个数量使用端口偏移创建备份代理。默认值为 100。

excludes （可选）

从备份端口偏移中排除连接器。如果您已经为应该从备份端口偏移中排除的外部代理配置了任何连接器，请为每个连接器添加一个 &lt ;connector-ref >。

master

此代理的共享存储或复制故障转移配置。

slave

此代理的备份的共享存储或复制故障转移配置。

3. *对集群中的每个剩余的代理重复此步骤。*

其他资源

- *有关使用 colocated 备份的代理集群示例，请参阅 HA 示例。*

14.3.6. 将客户端配置为故障转移

在代理集群中配置高可用性后，您要将客户端配置为故障转移。客户端故障转移可确保如果代理失败，则连接到它的客户端可以在最少的停机时间的情况下重新连接到集群中的另一个代理。

注意

如果出现临时网络问题，AMQ Broker 会自动重新连接到同一代理。这与故障转移类似，但客户端重新连接到同一代理。

您可以配置两种不同类型的客户端故障切换：

自动客户端故障切换

客户端在第一次连接时接收代理集群的信息。如果连接失败的代理，客户端会自动重新连接到代理的备份，并且备份代理会在故障转移前每个连接上重新创建任何会话和消费者。

应用程序级客户端故障转移

作为自动客户端故障切换的替代选择，您可以在失败处理程序中使用自己的自定义重新连接逻辑对客户端应用程序进行编码。

流程

- 使用 AMQ 核心协议 JMS 配置您的客户端应用程序，具有自动或应用级别故障转移。

  如需更多信息，请参阅使用 AMQ 核心协议 JMS 客户端。

14.4. 启用消息重新发布

如果您的代理集群配置了 message-load-balancing 设置为 ON_DEMAND 或 OFF_WITH_REDISTRIBUTION，您可以配置消息 redistribution 以防止消息在没有消费者使用的队列中"stuck"。

本节包含有关以下信息：

- 了解消息发布
- 配置消息重新发布

14.4.1. 了解消息重新发布

代理集群使用负载均衡来在集群中分发消息负载。在集群连接中配置负载均衡时，您可以使用以下 message-load-balancing 设置启用 redistribution ：

- ON_DEMAND - 启用负载平衡并允许重新发布
- OFF_WITH_REDISTRIBUTION - 禁用负载平衡，但允许重新发布

在这两种情况下，代理仅将消息转发到具有匹配消费者的其他代理。此行为可确保消息不会移到没有消费者使用消息的队列。但是，如果在消息转发到代理后接近队列的用户，这些消息将变为队列中的"stuck"，且不会被使用。此问题有时被称为 starvation。

消息重新分配通过自动将消息从队列重新分发到集群中没有消费者的代理，从而防止星号。

使用 OFF_WITH_REDISTRIBUTION 时，代理仅将消息转发到具有匹配消费者的其他代理（如果没有活跃的本地消费者），允许您在消费者不可用时优先选择代理。

消息重新发布支持使用过滤器（也称为 选择器），即，当消息与可用本地消费者的选择器不匹配时会被重新分发。

其他资源

- 有关集群负载均衡的更多信息，请参阅 第 14.1.1 节 “代理集群如何平衡消息负载”。

14.4.2. 配置消息重新发布

此流程演示了如何使用负载平衡配置消息重新分发。如果您希望消息在没有负载均衡的情况下重新发布，请将 &lt ;message-load-balancing& gt; 设置为 OFF_WITH_REDISTRIBUTION。

流程

1. 打开 & lt;broker_instance_dir> /etc/broker.xml 配置文件。

2. 在 & lt;cluster-connection& gt; 元素中，验证 <message-load-balancing > 是否已设置为 ON_DEMAND。

```
<configuration>
  <core>
    ...
    <cluster-connections>
      <cluster-connection name="my-cluster">
        ...
        <message-load-balancing>ON_DEMAND</message-load-balancing>
        ...
      </cluster-connection>
    </cluster-connections>
  </core>
</configuration>
```

3. 在 &lt ;address-settings > 元素中，为队列或一组队列设置 redistribution delay。

   在本例中，在最后一个消费者关闭后，负载均衡到 my.queue 的消息将被重新分发 5000 毫秒。

```
<configuration>
  <core>
    ...
    <address-settings>
```

```
            <address-setting match="my.queue">
               <redistribution-delay>5000</redistribution-delay>
            </address-setting>
         </address-settings>
         ...
      </core>
   </configuration>
```

address-setting

将 match 属性设置为您要重新分发消息的队列名称。您可以使用代理通配符语法来指定队列范围。更多信息请参阅 第 4.2 节 “将地址设置应用到一组地址”。

redistribution-delay

代理在将此队列的最终消费者关闭后应等待的时间（以毫秒为单位），然后再将消息重新分发到集群中的其他代理。如果将其设置为 0，则信息将立即重新分发。但是，您应该在 redistributing 前设置一个延迟 - 消费者通常关闭，但同一队列中需要快速创建另一个。

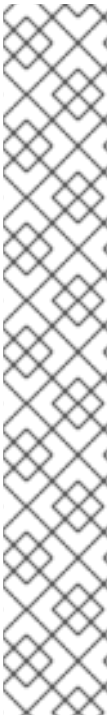
4. 对集群中的每个额外代理重复此步骤。

其他资源

- 有关重新分发消息的代理集群配置示例，请参阅 queue-message-redistribution 示例。

14.5. 配置集群消息分组

消息分组可让客户端发送特定类型的消息组，由同一消费者按顺序处理。通过向集群中的每个代理添加分组处理器，您可以确保客户端可以将分组的消息发送到集群中的任何代理，并且仍然使这些消息被同一消费者的正确顺序使用。



注意

分组和集群技术总结如下：

- *消息分组对消息消耗施加一个顺序。在组中，必须完全消耗和确认每个消息，然后才能继续下一消息。这个方法会导致串行消息处理，其中 concurrency 不是一个选项。*
- *集群旨在水平扩展代理，以提高消息吞吐量。通过添加额外的可同时处理消息的消费者来实现水平扩展。*

由于这些技术相互冲突，因此请避免将集群和分组一起使用。

有两种分组处理程序：local handlers 和 remote handlers。它们使代理集群能够将特定组中的所有消息路由到适当的队列，以便预期的消费者可以按照正确的顺序消耗它们。

先决条件

- *集群中每个代理均应至少有一个消费者。*

当消息固定到队列中的消费者时，具有相同组 ID 的所有消息都将路由到该队列。如果删除了消费者，队列将继续接收消息，即使没有消费者也是如此。

流程

1. *在集群中的一个代理上配置本地处理程序。*

   如果您使用高可用性，这应该是 master 代理。

   a. *打开代理的 < broker_instance_dir> /etc/broker.xml 配置文件。*

   b. *在 &lt ;core> 元素中，添加一个本地处理器：*

   本地处理程序充当远程处理程序的仲裁程序。它存储路由信息并将其与其他代理通信。

   -

```
<configuration>
  <core>
    ...
    <grouping-handler name="my-grouping-handler">
      <type>LOCAL</type>
      <timeout>10000</timeout>
    </grouping-handler>
    ...
  </core>
</configuration>
```

grouping-handler

使用 name 属性为分组处理程序指定唯一名称。

type

把它设置为 LOCAL。

timeout

决定在何处路由消息的时间（以毫秒为单位）。默认值为 5000 毫秒。如果在做出路由决策前达到超时，则会抛出异常，以确保严格的消息排序。

当代理收到带有组 ID 的消息时，它会提议一个到消费者附加到队列的路由。如果路由被集群中其他代理上的处理程序分组，则建立路由：集群中的所有代理会将具有此组 ID 的消息转发到该队列。如果代理的路由提议被拒绝，则它会提议一个备用路由，重复该过程，直到接受路由为止。

2. 如果您使用高可用性，请将本地处理器配置复制到主代理的 slave 代理。

   将本地处理器配置复制到 slave 代理可防止本地处理器的单点故障。

3. 在集群的每个剩余的代理上，配置远程处理器。

   a. 打开代理的 < broker_instance_dir> /etc/broker.xml 配置文件。

   b. 在 &lt ;core> 元素中，添加一个远程处理器：

```
<configuration>
  <core>
    ...
```

```
        <grouping-handler name="my-grouping-handler">
            <type>REMOTE</type>
            <timeout>5000</timeout>
        </grouping-handler>
        ...
    </core>
</configuration>
```

grouping-handler

使用 name 属性为分组处理程序指定唯一名称。

type

将它设置为 REMOTE。

timeout

决定在何处路由消息的时间（以毫秒为单位）。默认值为 5000 毫秒。将此值设置为本地处理程序的至少一半。

其他资源

- 有关为消息分组配置的代理集群示例，请参阅 JMS 集群分组示例。

14.6. 将客户端连接到代理集群

您可以使用 AMQ JMS 客户端连接到集群。通过使用 JMS，您可以配置消息传递客户端来动态或静态发现代理列表。您还可以配置客户端负载均衡，以分发从集群的连接中创建的客户端会话。

流程

- 使用 AMQ Core Protocol JMS 将客户端应用程序配置为连接到代理集群。

  如需更多信息，请参阅使用 AMQ 核心协议 JMS 客户端。

14.7. 对客户端连接进行分区

分区客户端连接涉及每次客户端发起连接时单个客户端到同一代理的路由连接。

分区客户端连接的两个用例是：

- 持久订阅的分区客户端，以确保订阅者始终连接到持久订阅者队列所在的代理。
- 尽量减少将数据移至源自的数据（也称为数据获取）来移动数据的需求。

持久化订阅

持久订阅表示为代理上的队列，在持久订阅者首次连接到代理时创建。此队列保留在代理中，并接收信息，直到客户端取消订阅为止。因此，您希望客户端重复连接到同一代理，以使用订阅者队列中的消息。

要为持久订阅队列对客户端进行分区，您可以在客户端连接中过滤客户端 ID。

Data gravity

如果您在不考虑数据获取的情况下扩展环境中的代理数量，一些性能优势会因为需要在代理之间移动消息而丢失。为了支持日期获取，您应该对客户端连接进行分区，以便客户端消费者连接到生成需要使用消息的代理。

要对客户端连接进行分区，您可以过滤客户端连接的以下任何属性：

- 分配给连接用户(ROLE_NAME)的角色
- 用户的用户名(USER_NAME)
- 客户端的主机名(SNI_HOST)
- 客户端的 IP 地址(SOURCE_IP)

14.7.1. 对客户端连接进行分区以支持持久订阅

要为持久订阅分区客户端，您可以使用一致的哈希算法或正则表达式过滤传入连接中的客户端 ID。

先决条件

客户端被配置为可以连接到集群中的所有代理，例如，通过使用负载均衡器，或连接 URL 中配置的所有代理实例。如果代理拒绝连接，因为客户端详情与该代理的分区配置不匹配，客户端必须能够连接到集群中的其他代理，以查找接受连接的代理。

14.7.1.1. 使用一致的哈希算法过滤客户端 ID

您可以配置集群中的每个代理，以使用一致的哈希算法来为每个客户端连接中的客户端 ID 进行哈希处理。在代理对客户端 ID 进行哈希后，它会在散列值上执行 modulo 操作，以返回整数值，用于标识客户端连接的目标代理。代理将返回的整数值与代理上配置的唯一值进行比较。如果存在匹配项，代理会接受连接。如果值不匹配，代理会拒绝连接。此过程会在集群中的每个代理上重复，直到找到匹配项并接受连接。

流程

1. 为第一个代理打开 < broker_instance_dir> /etc/broker.xml 配置文件。

2. 创建 connection-routers 元素，创建一个 connection-route 来使用一致的哈希算法过滤客户端 ID。例如：

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name=”consistent-hash-routing”>
        <key>CLIENT_ID</target-key>
        <local-target-filter>NULL|0</local-target-filter>
        <policy name="CONSISTENT_HASH_MODULO">
          <property key="modulo" value="<number_of_brokers_in_cluster>">
          </property>
        </policy>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

connection-route

对于 connection-route 名称，请为此连接路由配置指定一个识别字符串。您必须将此名称添加到您要应用一致的散列过滤器的每个代理接受者。

key

将过滤器应用到的密钥类型。要过滤客户端 ID，请在 key 字段中指定 CLIENT_ID。

local-target-filter

代理与 modulo 操作返回的整数值进行比较，以确定是否存在匹配项，代理是否可以接受连接。示例中的 NULL|0 值为没有客户端 ID(NULL)的连接提供了匹配的连接，其中 modulo 操作返回的数量为 0。

policy

接受 modulo 属性键，它对散列客户端 ID 执行 modulo 操作，以识别目标代理。modulo 属性键的值必须等于集群中的代理数量。

重要

策略名称必须是 CONSISTENT_HASH_MODULO。

3. *为第二个代理打开 < broker_instance_dir> /etc/broker.xml 配置文件。*

4. *创建 connection-routers 元素，并使用一致的哈希算法创建一个 连接路由 来过滤客户端 ID。*

在以下示例中，NULL|1 的 local-target-filter 值为没有客户端 ID (NULL)的连接提供匹配，以及 modulo 操作返回的值为 1 的连接。

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name=”consistent-hash-routing”>
        <key>CLIENT_ID</target-key>
        <local-target-filter>NULL|1</local-target-filter>
        <policy name="CONSISTENT_HASH_MODULO">
          <property key="modulo" value="<number_of_brokers_in_cluster>">
          </property>
        </policy>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

5. 重复这个过程，为集群中的每个额外代理创建一个一致的哈希过滤器。

14.7.1.2. 使用正则表达式过滤客户端 ID

您可以通过将代理配置为将正则表达式过滤器应用到客户端连接中的客户端 ID 的一部分来对客户端连接中的客户端连接进行分区。只有在正则表达式过滤器的结果与为代理配置的本地目标过滤器匹配时，代理才会接受连接。如果没有找到匹配项，代理会拒绝连接。此过程会在集群中的每个代理上重复，直到找到匹配项并接受连接。

先决条件

- 每个客户端 ID 中可以通过正则表达式过滤的通用字符串。

流程

1. 为第一个代理打开 < broker_instance_dir> /etc/broker.xml 配置文件。

2. 创建 connection-routers 元素，再创建一个 connection-route 来过滤客户端 ID 的一部分。例如：

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name="regex-routing">
        <key>CLIENT_ID</target-key>
        <key-filter>^.{3}</key-filter>
        <local-target-filter>NULL|CL1</local-target-filter>
      </connection-route>
  </connection-routers>
    ...
  </core>
</configuration>
```

connection-route

对于 connection-route 名称，请指定此路由配置的标识字符串。您必须将此名称添加到您要应用正则表达式过滤器的每个代理接受者。

key

将过滤器应用到的键。要过滤客户端 ID，请在 key 字段中指定 CLIENT_ID。

key-filter

将正则表达式应用到的客户端 ID 字符串的一部分，以提取键值。在上面的第一个代理示例中，代理提取一个键值，它是客户端 ID 的前 3 个字符。例如，如果客户端 ID 字符串为 CL100.consumer，则代理提取键值 CL1。在代理提取键值后，它会将其与 local-target-filter 的值进行比较。

如果传入的连接没有客户端 ID，或者代理无法使用为 key-filter 指定的正则表达式提取键值，则键值将设置为 NULL。

local-target-filter

代理与键值相比的值，以确定是否存在匹配项，代理是否可以接受连接。上面的第一个代理示例中所示 NULL|CL1 的值匹配，匹配客户端 ID (NULL)或客户端 ID 中有 3 个字符前缀 CL1 的连接。

3. 为第二个代理打开 < broker_instance_dir> /etc/broker.xml 配置文件。

4. 创建 connection-routers 元素，创建一个 连接路由，以根据客户端 ID 的一部分过滤连接。

在以下过滤器示例中，代理使用正则表达式来提取键值，这是客户端 ID 的前 3 个字符。代理将 NULL 和 CL2 的值与键值进行比较，以确定是否存在匹配项，代理是否可以接受连接。

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name="regex-routing">
        <key>CLIENT_ID</target-key>
        <key-filter>^.{3}</key-filter>
        <local-target-filter>NULL|CL2</local-target-filter>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

5. 重复此步骤，并为集群中的每个额外代理创建适当的连接路由过滤器。

14.7.2. 对客户端连接进行分区以支持数据获取

要支持日期的获取，您可以对客户端连接进行分区，以便客户端消费者连接到生成需要使用消息的代理。例如，如果您有一组由制作者和消费者应用程序使用的地址，您可以在特定代理上配置地址。然后，

您可以为使用这些地址的生成者和消费者对客户端连接进行分区，以便它们只能连接到该代理。

您可以根据分配给连接用户、用户的用户名或客户端的主机名或 IP 地址等属性对客户端连接进行分区。本节演示了如何通过过滤分配给客户端用户的用户角色来对客户端连接进行分区。如果需要客户端验证连接到代理，您可以将角色分配给客户端用户并过滤连接，以便只有符合角色条件的用户才能连接到代理。

先决条件

- 客户端被配置为可以连接到集群中的所有代理，例如，通过使用负载均衡器，或连接 URL 中配置的所有代理实例。如果代理拒绝连接，因为客户端与为那个代理配置的分区过滤器条件不匹配，客户端必须能够连接到集群中的其他代理，以查找接受连接的代理。

流程

1. 为第一个代理打开 < broker_instance_dir> /etc/artemis-roles.properties 文件。添加 broker1users 角色，并将用户添加到角色中。

2. 为第一个代理打开 < broker_instance_dir> /etc/broker.xml 配置文件。

3. 创建一个 connection-routers 元素，创建一个 connection-route，以根据分配给用户的角色过滤连接。例如：

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name="role-based-routing">
        <key>ROLE_NAME</target-key>
        <key-filter>broker1users</key-filter>
        <local-target-filter>broker1users</local-target-filter>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

connection-route

对于 connection-route 名称，请指定此路由配置的标识字符串。您必须将此名称添加到您要应用角色过滤器的每个代理接受者。

key

将过滤器应用到的键。要配置基于角色的过滤，在 key 字段中指定 ROLE_NAME。

key-filter

代理用来过滤用户角色并提取键值的字符串或正则表达式。如果代理找到匹配的角色，它会将 key 值设置为该角色。如果没有找到匹配的角色，代理会将 key 值设置为 NULL。在上例中，代理将 broker1users 的过滤器应用到客户端用户的角色。在代理提取键值后，它会将其与 local-target-filter 的值进行比较。

local-target-filter

代理与键值相比的值，以确定是否存在匹配项，代理是否可以接受连接。在示例中，代理将 broker1users 的值与键值进行比较。它有一个匹配项，这意味着用户具有 broker1users 角色，代理接受连接。

4. 重复此步骤并在过滤器中指定适当的角色，以在集群中的其他代理上对客户端进行分区。

14.7.3. 将连接路由添加到接收器

在代理上配置连接路由后，您必须将路由添加到一个或多个代理的 acceptors 来分区客户端连接。将连接路由添加到 acceptor 后，代理会将连接路由中配置的过滤器应用到接受者接收的连接。

流程

1. 为第一个代理打开 < broker_instance_dir> /etc/broker.xml 配置文件。

2. 对于您要启用分区的每个接受者，附加 路由器 密钥并指定 connection-route 名称。在以下示例中，将具有 consistent-hash-routing 的 connection-route name 添加到 artemis acceptor 中。

```
<configuration>
  <core>
    ...
    <acceptors>
    ...
    <!-- Acceptor for every supported protocol -->
    <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,S
TOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredit
s=300;router="consistent-hash-routing" </acceptor>
    </acceptors>
    ...
  </core>
</configuration>
```

3. 重复此步骤，为集群中的每个代理指定适当的连接路由过滤器。

第 15 章 使用 CEPH 配置多站点、容错消息传递系统

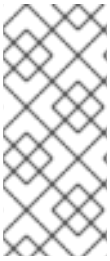
大规模企业消息传递系统通常具有位于地理上分布式数据中心的离散代理集群。如果数据中心停机，系统管理员可能需要保留现有的消息传递数据，并确保客户端应用程序可以继续生成和使用消息。您可以使用特定的代理拓扑和 Red Hat Ceph Storage （一个软件定义的存储平台）来确保消息传递系统在数据中心中断期间的连续性。这种类型的解决方案称为 多站点、容错架构。

注意

如果您只需要 AMQP 协议支持，请考虑 第 16 章 使用代理连接配置多站点、容错消息传递系统。

以下小节解释了如何使用 Red Hat Ceph Storage 保护您的消息传递系统免受数据中心中断的影响：

- Red Hat Ceph Storage 集群如何工作
- 安装和配置 Red Hat Ceph Storage 集群
- 在数据中心停机时，添加备份代理以从实时代理接管
- 使用 Ceph 客户端角色配置代理服务器
- 将每个代理配置为使用共享存储高可用性(HA)策略，指定 Ceph 文件系统中每个代理存储其消息传递数据的位置
- 在数据中心停机时将客户端应用程序配置为连接到新代理
- 中断后重启数据中心



注意

多站点容错不是数据中心内高可用性(HA)代理冗余的替代。基于 live-backup 组的代理冗余提供对单个集群中单一代理故障的自动保护。相反，多站点容错可防止大规模数据中心中断。

注意

要使用 Red Hat Ceph Storage 来确保消息传递系统的连续性，您必须将代理配置为使用共享存储高可用性(HA)策略。您无法将代理配置为使用复制 HA 策略。有关这些策略的更多信息，请参阅 实施高可用性。

15.1. RED HAT CEPH STORAGE 集群如何工作

Red Hat Ceph Storage 是一个集群对象存储系统。Red Hat Ceph Storage 使用对象和基于策略的复制的数据分片来确保数据完整性和系统可用性。

Red Hat Ceph Storage 使用名为 CRUSH （可扩展哈希下的受控复制）的算法来确定如何通过自动计算数据存储位置存储和检索数据。您可以配置 Ceph 项目，称为 CRUSH map，其详细说明了集群拓扑，并指定如何在存储集群之间复制数据。

CRUSH map 包含对象存储设备(OSD)列表、用于将设备聚合为故障域层次结构中的"buckets"列表，以及告知 CRUSH 如何在 Ceph 集群池中复制数据的规则。

通过反映安装的底层物理组织，CRUSH 映射可以建模，因此地址 - 关联设备故障的潜在源，如物理代理、共享电源源和共享网络。通过将此信息编码到 cluster map 中，CRUSH 可以在不同的故障域间分离对象副本（如数据中心），同时仍然在存储集群中维护伪随机数据分布。这有助于防止数据丢失并启用集群以降级状态运行。

Red Hat Ceph Storage 集群需要多个节点（物理或虚拟）才能运行。集群必须包括以下类型的节点：

监控节点

每个 monitor (MON)节点运行 monitor 守护进程(ceph-mon)，后者维护 cluster map 的主副本。集群映射包含集群拓扑。连接到 Ceph 集群的客户端从 monitor 检索 cluster map 的当前副本，这使得客户端能够从集群读取和写入数据。

重要

Red Hat Ceph Storage 集群可以使用一个 monitor 节点运行；但是，为了确保生产集群中的高可用性，红帽仅支持具有至少三个 monitor 节点的部署。至少三个 monitor 节点意味着，当一个 monitor 失败或不可用时，集群中剩余的 monitor 节点有一个仲裁来选举新的领导。

Manager 节点

每个管理器(MGR)节点运行 Ceph 管理器守护进程(ceph-mgr)，它负责跟踪运行时指标和 Ceph 集群的当前状态，包括存储利用率、当前的性能指标和系统负载。通常，管理器节点与 monitor 节点在一起（即在同一主机上）。

对象存储设备节点

每个对象存储设备(OSD)节点运行 Ceph OSD 守护进程(ceph-osd)，它与附加到节点的逻辑卷交互。Ceph 在 OSD 节点上存储数据。Ceph 可以在非常少的 OSD 节点（默认为三个）的情况下运行，但生产集群在大规模扩展时实现更好的性能，例如存储集群中有 50 个 OSD。在存储集群中有多个 OSD 可让系统管理员在 CRUSH map 中定义隔离的故障域。

元数据服务器节点

每个元数据服务器(MDS)节点运行 MDS 守护进程(ceph-mds)，后者管理与 Ceph 文件系统(CephFS)中存储的文件相关的元数据。MDS 守护进程也协调对共享集群的访问。

其他资源

有关 Red Hat Ceph Storage 的更多信息，请参阅什么是 Red Hat Ceph Storage？

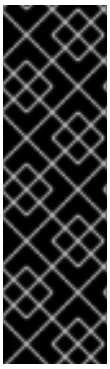
15.2. 安装 RED HAT CEPH STORAGE

AMQ Broker 多站点、容错架构使用 Red Hat Ceph Storage 3。通过跨数据中心复制数据，Red Hat Ceph Storage 集群实际上会在单独的数据中心中为代理创建可用的共享存储。您可以将代理配置为使用共享存储高可用性(HA)策略，并将消息传递数据存储在 Red Hat Ceph Storage 集群中。

用于生产环境的 Red Hat Ceph Storage 集群应该最少：

- 三个 monitor (MON)节点

- 三个管理器(MGR)节点
- 三个包含多个 OSD 守护进程的对象存储设备(OSD)节点
- 三个元数据服务器(MDS)节点



重要

您可以在相同或单独的物理或虚拟机上运行 OSD、MON、MGR 和 MDS 节点。但是，为了确保 Red Hat Ceph Storage 集群中的容错，最好在不同的数据中心中分发这些类型的节点。特别是，您必须确保在单个数据中心停机时，您的存储集群仍至少有两个可用的 MON 节点。因此，如果集群中有三个 MON 节点，则这些节点都必须在独立数据中心的单独主机上运行。不要在一个数据中心中运行两个 MON 节点，因为此数据中心的故障会使存储集群仅有一个剩余的 MON 节点。在这种情况下，存储集群将不再运行。

本节链接的步骤演示了如何安装包含 MON、MGR、OSD 和 MDS 节点的 Red Hat Ceph Storage 3 集群。

先决条件

- 有关准备 Red Hat Ceph Storage 安装的详情，请参考：
  - 先决条件
  - 安装 Red Hat Ceph Storage 的要求检查列表

流程

- 有关如何安装包含 MON、MGR、OSD 和 MDS 节点的 Red Hat Ceph 3 存储集群的步骤，请参阅：
  - 安装 Red Hat Ceph Storage 集群
  - 安装元数据服务器

15.3. 配置 RED HAT CEPH STORAGE 集群

本例步骤演示了如何为容错配置 Red Hat Ceph Storage 集群。您可以创建 CRUSH 存储桶，将对象存储设备(OSD)节点聚合到反映您的现实物理安装的数据中心。此外，您还可创建一个规则，告知 CRUSH 如何在存储池中复制数据。这些步骤更新 Ceph 安装创建的默认 CRUSH map。

先决条件

- 已安装 Red Hat Ceph Storage 集群。有关更多信息，请参阅安装 Red Hat Ceph Storage。
- 您应该了解 Red Hat Ceph Storage 如何使用放置组(PG)在池中组织大量数据对象，以及如何计算池中要使用的 PG 数量。如需更多信息，请参阅 放置组(PG)。
- 您应该了解如何在池中设置对象副本数量。如需更多信息，请参阅设置对象副本的数量。

流程

1. 创建 CRUSH 存储桶以组织 OSD 节点。bucket 是 OSD 列表，基于数据中心等物理位置。在 Ceph 中，这些物理位置称为 故障域。

```
ceph osd crush add-bucket dc1 datacenter
ceph osd crush add-bucket dc2 datacenter
```

2. 将 OSD 节点的主机机器移到您创建的数据中心 CRUSH 存储桶。将主机名 host1-host4 替换为您的主机机器的名称。

```
ceph osd crush move host1 datacenter=dc1
ceph osd crush move host2 datacenter=dc1
ceph osd crush move host3 datacenter=dc2
ceph osd crush move host4 datacenter=dc2
```

3. 确保您创建的 CRUSH 存储桶 是默认 CRUSH 树的一部分。

```
ceph osd crush move dc1 root=default
ceph osd crush move dc2 root=default
```

4. 创建一条规则来映射数据中心中的存储对象副本。这有助于防止数据丢失，并使集群在单个数据中心中断时保持运行。

创建规则的命令使用以下语法： ceph osd crush rule create-replicated <rule-name> <root> <failure-domain> <class>。下面是一个示例。

```
ceph osd crush rule create-replicated multi-dc default datacenter hdd
```

注意

在前面的命令中，如果您的存储集群使用固态驱动器(SSD)，请指定 ssd 而不是 hdd （硬磁盘驱动器）。

5. 配置 Ceph 数据和元数据池，以使用您创建的规则。最初，这可能会导致数据回填到由 CRUSH 算法决定的存储目的地。

```
ceph osd pool set cephfs_data crush_rule multi-dc
ceph osd pool set cephfs_metadata crush_rule multi-dc
```

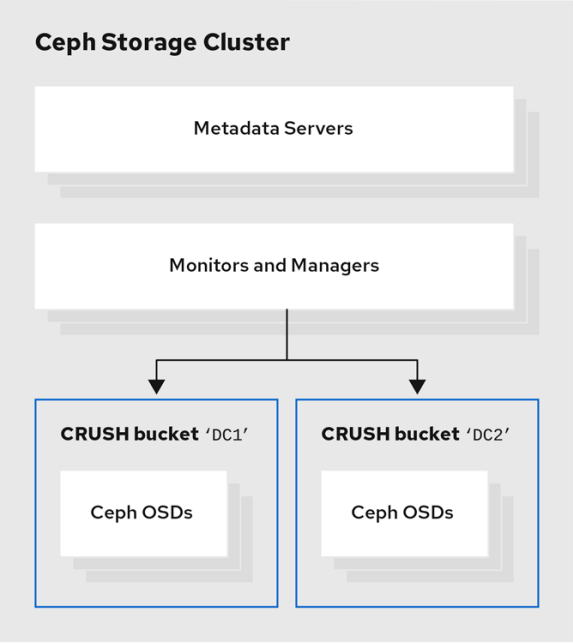
6. 为您的元数据和数据池指定放置组(PG)和放置组的数量。PGP 值应当等于 PG 值。

```
ceph osd pool set cephfs_metadata pg_num 128
ceph osd pool set cephfs_metadata pgp_num 128

ceph osd pool set cephfs_data pg_num 128
ceph osd pool set cephfs_data pgp_num 128
```

7. 指定数据和元数据池使用的副本数。

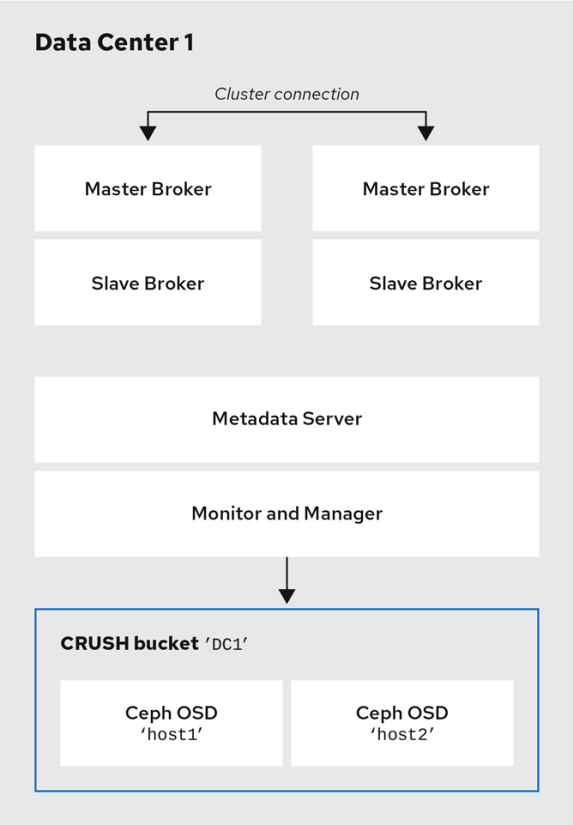
```
ceph osd pool set cephfs_data min_size 1
ceph osd pool set cephfs_metadata min_size 1

ceph osd pool set cephfs_data size 2
ceph osd pool set cephfs_metadata size 2
```

下图显示了前面的示例步骤创建的 Red Hat Ceph Storage 集群。存储集群的 OSD 整理到与数据中心对应的 CRUSH bucket 中。



下图显示了第一个数据中心（包括您的代理服务器）的可能布局。具体来说，数据中心主机：

- *两个 live-backup 代理对的服务器*
- *在前面的步骤中分配给第一个数据中心的 OSD 节点*
- *单个元数据服务器、监控和管理器节点。monitor 和 Manager 节点通常位于同一个机器上。*

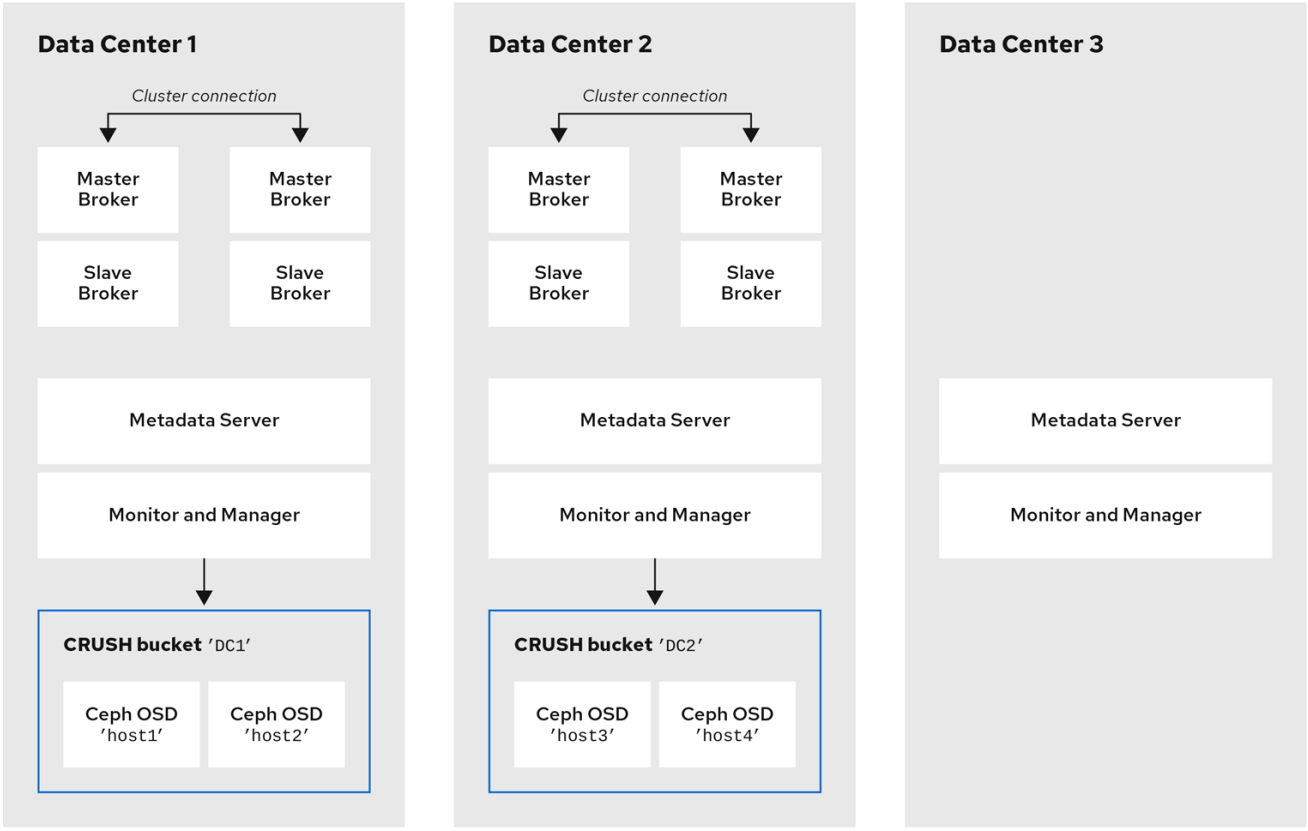


Ceph_41_0919

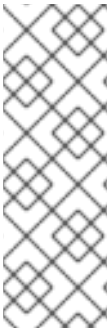
重要

您可以在相同或单独的物理或虚拟机上运行 OSD、MON、MGR 和 MDS 节点。但是，为了确保 Red Hat Ceph Storage 集群中的容错，最好在不同的数据中心中分发这些类型的节点。特别是，您必须确保在单个数据中心停机时，您的存储集群仍至少有两个可用的 MON 节点。因此，如果集群中有三个 MON 节点，则这些节点都必须在独立数据中心的单独主机上运行。

下图显示了完整的示例拓扑。为确保存储集群中的容错，MON、MGR 和 MDS 节点分布在三个单独的数据中心中。



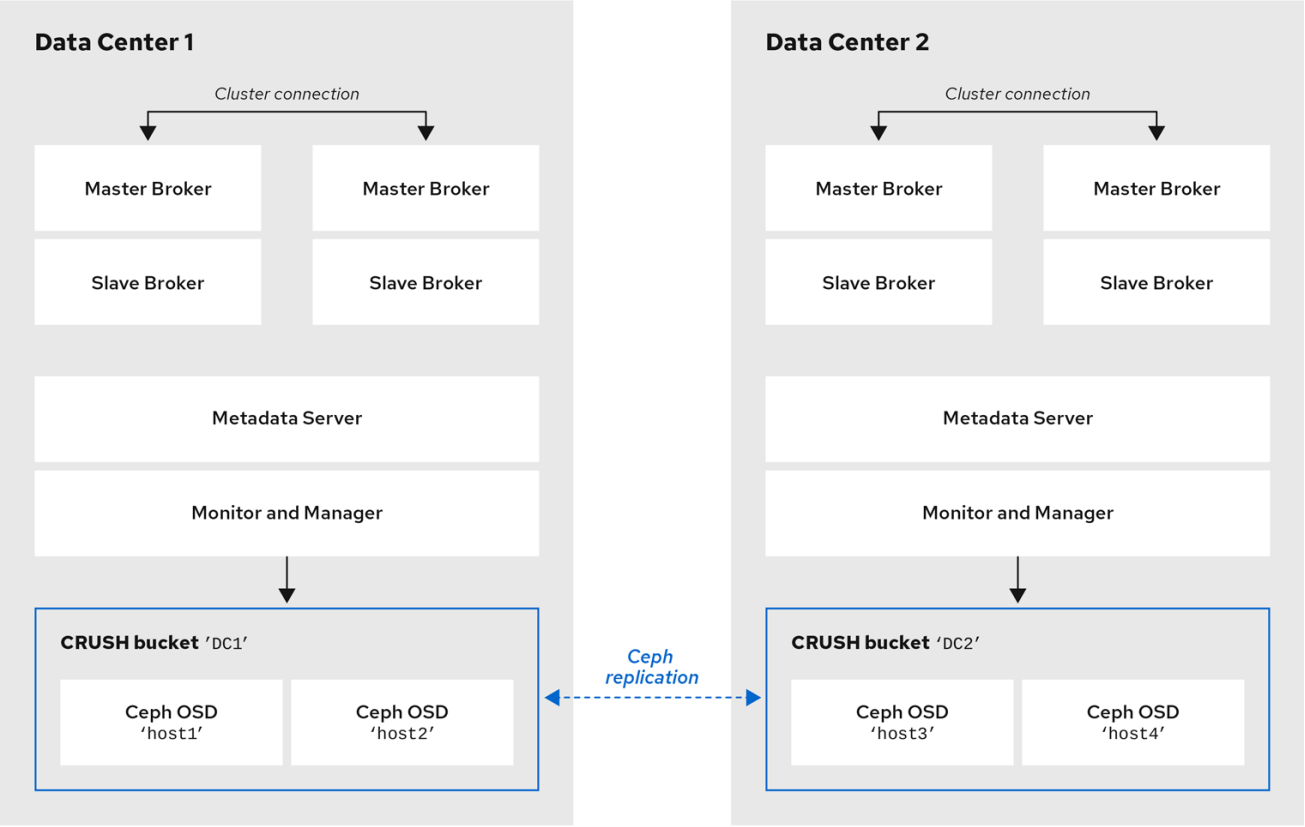
Ceph_41_0919



注意

在与代理服务器相同的数据中心中为某些 OSD 节点查找主机机器并不意味着您将消息传递数据存储在这些特定的 OSD 节点上。您可以配置代理，将消息传递数据存储在 Ceph 文件系统的指定目录中。然后，集群中的元数据服务器节点决定如何在数据中心的所有可用 OSD 之间分发存储的数据，并在数据中心间处理此数据的复制。以下章节演示了如何配置代理以在 Ceph 文件系统中存储消息传递数据。

下图展示了在具有代理服务器的两个数据中心之间复制数据。



其他资源

有关以下内容的更多信息：

- *为您的 Red Hat Ceph Storage 集群管理 CRUSH，请参阅 CRUSH 管理。*
- *您可以在存储池上设置的完整属性，请参阅池 值。*

15.4. 在代理服务器上挂载 CEPH 文件系统

在消息传递系统中配置代理以将消息传递数据存储在 Red Hat Ceph Storage 集群中，您首先需要挂载 Ceph 文件系统(CephFS)。

本节中的链接过程演示了如何在代理服务器上挂载 CephFS。

先决条件

- 您有：
  - 安装和配置 Red Hat Ceph Storage 集群。如需更多信息，请参阅安装 Red Hat Ceph Storage 和 配置 Red Hat Ceph Storage 集群。
  - 安装和配置三个或更多 Ceph 元数据服务器守护进程(ceph-mds)。如需更多信息，请参阅安装元数据服务器和配置元数据服务器守护进程。
  - 从 monitor 节点创建 Ceph 文件系统。有关更多信息，请参阅创建 Ceph 文件系统。
  - 创建具有您的代理服务器可用于授权访问的 Ceph 文件系统客户端用户。如需更多信息，请参阅创建 Ceph 文件系统客户端用户。

流程

有关在代理服务器上挂载 Ceph 文件系统的说明，请参阅将 Ceph 文件系统挂载为内核客户端。

15.5. 在多站点、容错消息传递系统中配置代理

要将代理配置为多站点、容错消息传递系统的一部分，您需要：

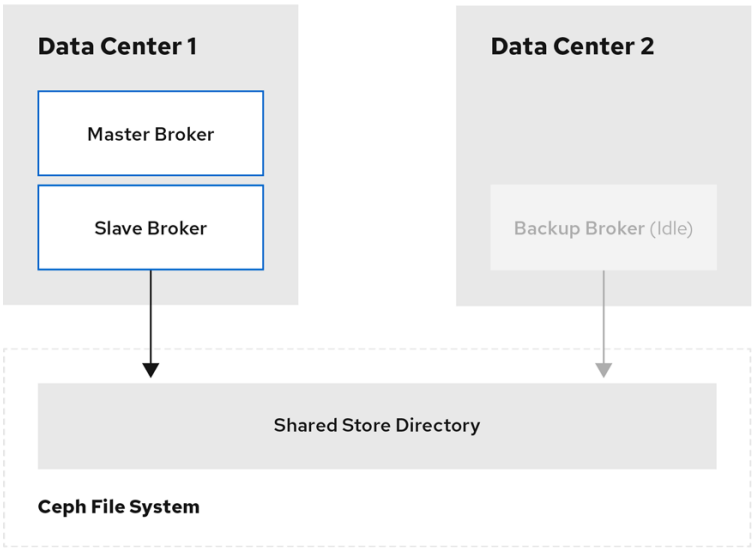
- 添加闲置备份代理，以便在数据中心失败时从 live 代理接管
- 使用 Ceph 客户端角色配置所有代理服务器
- 配置每个代理以使用共享存储高可用性(HA)策略，指定代理存储其消息传递数据的位置

15.5.1. 添加备份代理

在每个数据中心中，您需要添加空闲的备份代理，这些代理可以从 live master-slave 代理组中进行，这些代理会在数据中心中断时关闭。您应该在空闲的备份代理中复制 live master 代理的配置。您还需要配置备份代理以接受与现有代理相同的客户端连接。

在后续步骤中，您了解如何配置空闲的备份代理来加入现有的 master-slave 代理组。您必须在单独的数据中心中找到空闲的备份代理，以便 live master-slave 代理组。另外，建议您仅在数据中心失败时手动启动闲置备份代理。

下图显示了示例拓扑。



其他资源

- 要了解如何创建额外的代理实例，请参阅创建独立代理。
- 有关配置代理网络连接的详情，请参考 第 2 章 在网络连接中配置接收器和连接器。

15.5.2. 将代理配置为 Ceph 客户端

当您添加了容错系统所需的备份代理时，您必须使用 Ceph 客户端角色配置所有代理服务器。客户端角色可让代理在 Red Hat Ceph Storage 集群中存储数据。

要了解如何配置 Ceph 客户端，请参阅安装 Ceph 客户端角色。

15.5.3. 配置共享存储高可用性

Red Hat Ceph Storage 集群有效创建一个共享存储，该存储可供不同数据中心的代理使用。要确保在失败时消息仍可用于代理客户端，您可以在 live-backup 组中配置每个代理来使用：

- 共享存储高可用性(HA)策略
- Ceph 文件系统中的相同日志、分页和大型消息目录

以下流程演示了如何在 live-backup 组的 master、slave 和 idle 备份代理上配置共享存储 HA 策略。

流程

1. 编辑 live-backup 组中每个代理的 broker.xml 配置文件。将每个代理配置为使用 Ceph 文件系统中的相同分页、绑定、日志和大型消息目录。

```
# Master Broker - DC1
<paging-directory>mnt/cephfs/broker1/paging</paging-directory>
<bindings-directory>/mnt/cephfs/data/broker1/bindings</bindings-directory>
<journal-directory>/mnt/cephfs/data/broker1/journal</journal-directory>
<large-messages-directory>mnt/cephfs/data/broker1/large-messages</large-messages-directory>

# Slave Broker - DC1
<paging-directory>mnt/cephfs/broker1/paging</paging-directory>
<bindings-directory>/mnt/cephfs/data/broker1/bindings</bindings-directory>
<journal-directory>/mnt/cephfs/data/broker1/journal</journal-directory>
<large-messages-directory>mnt/cephfs/data/broker1/large-messages</large-messages-directory>

# Backup Broker (Idle) - DC2
<paging-directory>mnt/cephfs/broker1/paging</paging-directory>
<bindings-directory>/mnt/cephfs/data/broker1/bindings</bindings-directory>
<journal-directory>/mnt/cephfs/data/broker1/journal</journal-directory>
<large-messages-directory>mnt/cephfs/data/broker1/large-messages</large-messages-directory>
```

2. 将备份代理配置为 HA 策略中的 master，如下所示。此配置设置可确保当您手动启动备份代理时，备份代理会立即变为 master。因为代理是一个空闲的备份，所以您可以为活跃 master 代理指定 failover-on-shutdown 参数不会应用它。

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <master>
        </master>
      </shared-store>
    </ha-policy>
```

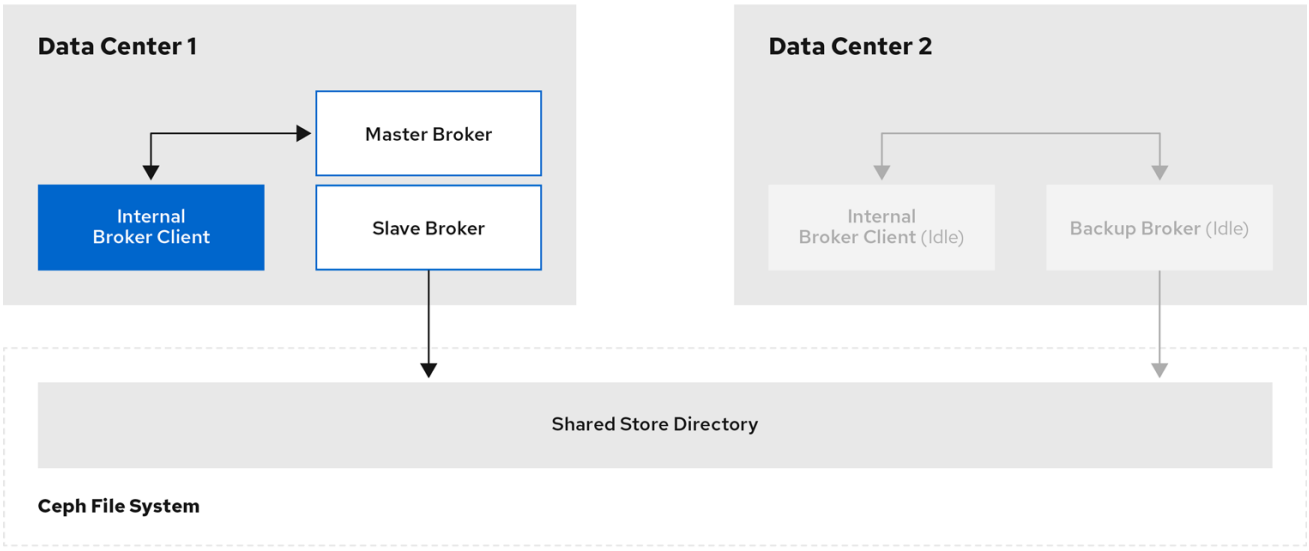
```
      ...
   </core>
</configuration>
```

其他资源

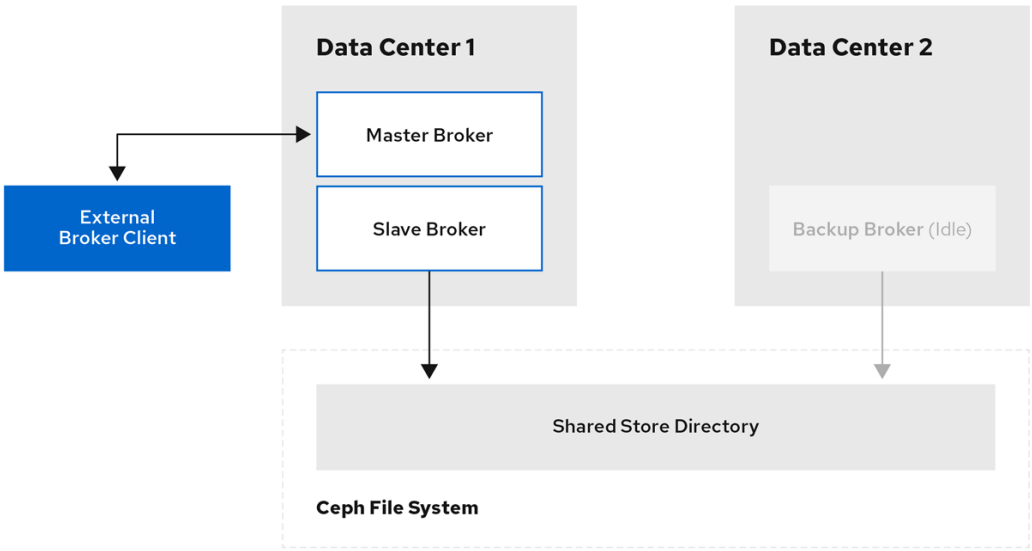
- 有关为 **live-backup** 代理组配置共享存储高可用性策略的更多信息，请参阅 **配置共享存储高可用性**。

15.6. 在多站点、容错消息传递系统中配置客户端

内部客户端应用程序是在位于与代理服务器相同的数据中心的机器上运行的应用程序。下图显示了此拓扑。



外部客户端应用程序是在代理数据中心外部的机器上运行。下图显示了此拓扑。



以下子章节描述了在数据中心中断时将内部和外部客户端应用程序配置为在另一个数据中心中连接到备份代理的示例。

15.6.1. 配置内部客户端

如果您遇到数据中心中断，内部客户端应用程序将与您的代理一起关闭。要缓解这种情况，您必须在单独的数据中心中具有另一个客户端应用程序实例。如果数据中心停机，您可以手动启动备份客户端以连接到您手动启动的备份代理。

要启用备份客户端连接到备份代理，您需要配置与主数据中心中客户端类似的客户端连接。

示例

AMQ Core Protocol JMS 客户端到 master-slave 代理组的基本连接配置如下所示。在本例中，host1 和 host2 是 master 和 slave 代理的主机服务器。

```
<ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("(tcp://host1:port,tcp://host2:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

要将备份客户端配置为在数据中心停机时连接到备份代理，请使用类似的连接配置，但只指定备份代理服务器的主机名。在本例中，备份代理服务器是 host3。

```
<ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("(tcp://host3:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

其他资源

- 有关配置代理网络连接的详情，请参考 第 2 章 在网络连接中配置接收器和连接器。

15.6.2. 配置外部客户端

要启用外部代理客户端在数据中心中断时继续生成或消耗消息传递数据，您必须将客户端配置为切换到另一个数据中心的代理。如果是多站点、容错系统，您可以将客户端配置为故障转移到您在停机时手动启动的备份代理。

例子

以下是将 AMQ Core Protocol JMS 和 AMQ JMS 客户端配置为在主 master-slave 组不可用时故障转移到备份代理的示例。在这些示例中，host1 和 host2 是主 master 和 slave 代理的主机服务器，而 host3 是您在数据中心停机时手动启动的备份代理的主机服务器。

- 要配置 AMQ Core Protocol JMS 客户端，请在客户端尝试连接的代理列表中包括备份代理。

```
<ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("(tcp://host1:port,tcp://host2:port,tcp://host3:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

- 要配置 AMQ JMS 客户端，请在您在客户端上配置的故障转移 URI 中包含备份代理。

```
failover:(amqp://host1:port,amqp://host2:port,amqp://host3:port)?
jms.clientID=myclient&failover.maxReconnectAttempts=20
```

其他资源

- 有关配置故障切换的更多信息：
  - AMQ 核心协议 JMS 客户端，请参阅 Reconnect 和 failover。
  - AMQ JMS 客户端，请参阅 Failover 选项。
  - 其他支持的客户端，请参阅 Red Hat AMQ 客户端 产品文档中的特定于客户端 的文档。

15.7. 在数据中心停机过程中验证存储集群健康状况

当您为容错配置了 Red Hat Ceph Storage 集群时，集群将继续在不丢失数据的情况下以降级状态运行，即使其中一个数据中心失败。

此流程演示了如何在处于降级状态时验证集群的状态。

流程

1. 要验证 Ceph 存储集群的状态，请使用 health 或 status 命令：

```
# ceph health
# ceph status
```

2. 要在命令行中观察集群的持续事件，请打开一个新的终端。然后，输入：

```
# ceph -w
```

运行上述任何命令时，您会看到指示存储集群仍在运行但处于降级状态的输出。特别是，您应该看到类似如下的警告：

```
health: HEALTH_WARN
        2 osds down
        Degraded data redundancy: 42/84 objects degraded (50.0%), 16 pgs unclean, 16 pgs
degraded
```

其他资源

- 有关监控 Red Hat Ceph Storage 集群健康状况的更多信息，请参阅监控。

15.8. 在数据中心中断期间保持消息传递连续

以下流程演示了如何在数据中心中断期间将代理和关联的消息传递数据提供给客户端。特别是，当数据中心失败时，您需要：

- 手动启动您创建的任何空闲备份代理，以便从失败数据中心的代理接管。

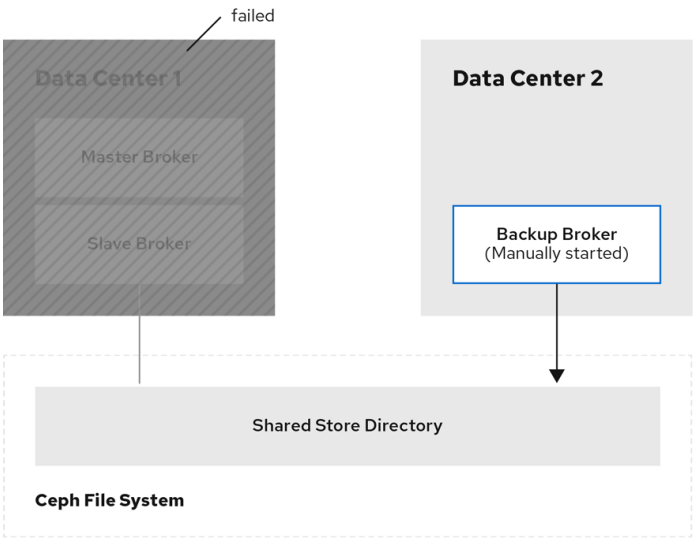
- 将内部或外部客户端连接到新的活跃代理。

先决条件

- 您必须：
  - 安装和配置 Red Hat Ceph Storage 集群。如需更多信息，请参阅安装 Red Hat Ceph Storage 和 配置 Red Hat Ceph Storage 集群。
  - 挂载 Ceph 文件系统.有关更多信息，请参阅在 代理服务器上挂载 Ceph 文件系统。
  - 添加了闲置备份代理，以便在数据中心出现故障时接管实时代理。如需更多信息，请参阅 添加备份代理。
  - 配置有 Ceph 客户端角色的代理服务器。有关更多信息，请参阅将代理配置为 Ceph 客户端。
  - 将每个代理配置为使用共享存储高可用性(HA)策略，指定 Ceph 文件系统中每个代理存储其消息传递数据的位置。如需更多信息，请参阅配置共享存储高可用性。
  - 将客户端配置为在数据中心中断时连接到备份代理。如需更多信息，请参阅 在多站点、容错消息传递系统中配置客户端。

流程

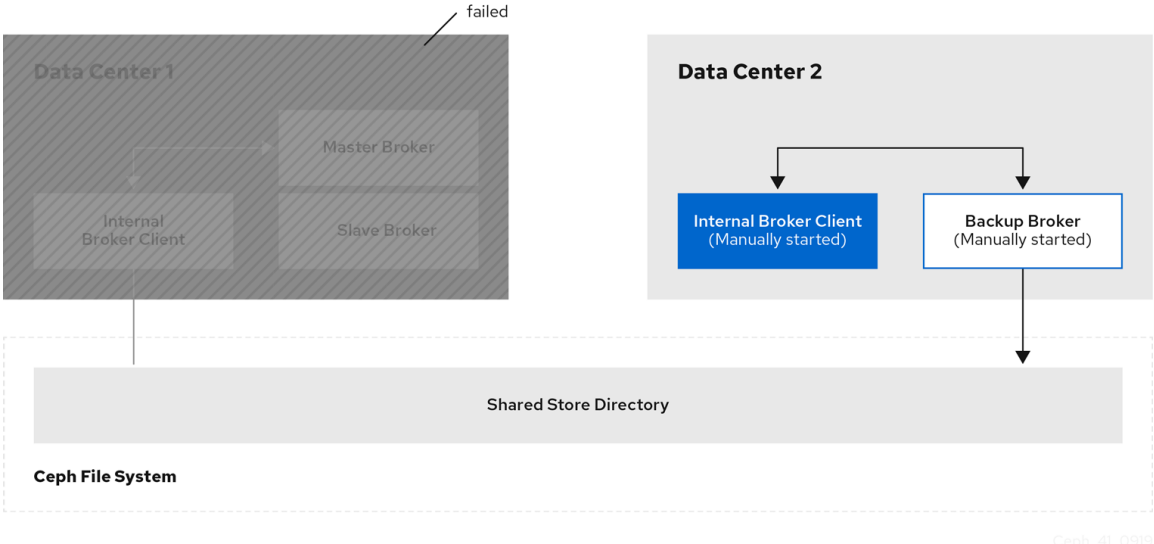
1. 对于失败的数据中心中的每个 master-slave 代理对，手动启动您添加的空闲备份代理。



2. *重新建立客户端连接。*

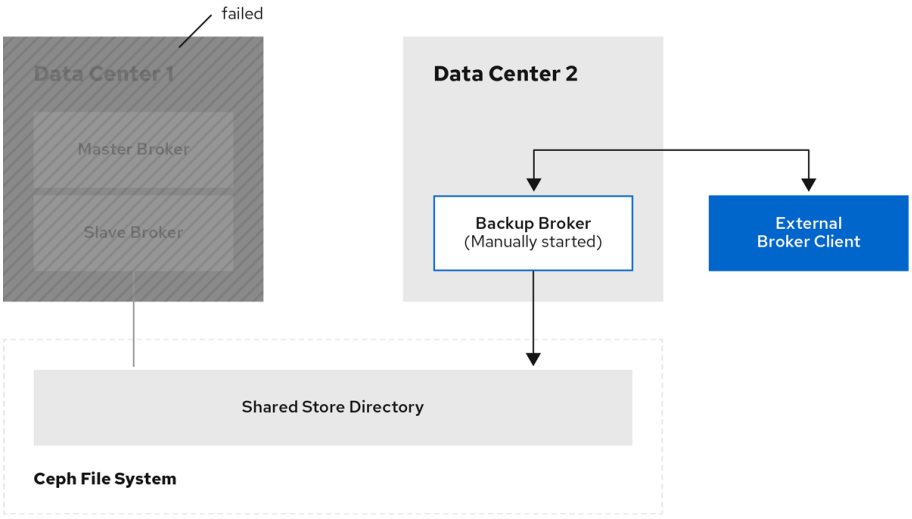
a. *如果您在失败的数据中心中使用内部客户端，请手动启动您创建的备份客户端。如 在多站点、容错消息传递系统 中配置客户端 中所述，您必须将客户端配置为连接到您手动启动的备份代理。*

下图显示了新拓扑。



b. *如果您有外部客户端，请手动将外部客户端连接到新的活跃代理，或者观察客户端自动根据其配置切换到新的活跃代理。如需更多信息，请参阅配置外部客户端。*

下图显示了新拓扑。



15.9. 重启之前失败的数据中心

当之前失败的数据中心恢复在线时，请按照以下步骤恢复消息传递系统的原始状态：

- 重启托管 Red Hat Ceph Storage 集群节点的服务器
- 重启消息传递系统中的代理
- 重新建立从客户端应用程序到恢复的代理的连接

以下子章节显示执行这些步骤。

15.9.1. 重启存储集群服务器

当您重启之前失败的数据中心中 monitor、元数据服务器、管理器和对象存储设备(OSD)节点时，您的 Red Hat Ceph Storage 集群自我修复来恢复完整的数据冗余。在此过程中，Red Hat Ceph Storage 会根据需要自动将数据回填到恢复的 OSD 节点。

要验证您的存储集群是否自动自我修复和恢复完整的数据冗余，请使用之前 *在数据中心中断期间验证存储集群健康状况* 的命令。重新执行这些命令时，您会看到之前 HEALTH_WARN 消息指示的百分比降级开始改进，直到它返回到 100%。

15.9.2. 重启代理服务器

以下流程演示了如何在存储集群不再处于降级状态时重启您的代理服务器。

流程

1. 停止所有连接到您在数据中心中断时手动启动的备份代理的客户端应用程序。

2. 停止手动启动的备份代理。

   a. 对于 Linux：

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. 在 Windows 上：

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

3. 在之前失败的数据中心中，重启原始 master 和 slave 代理。

   a. 对于 Linux：

   ```
   <broker_instance_dir>/bin/artemis run
   ```

   b. 在 Windows 上：

   ```
   <broker_instance_dir>\bin\artemis-service.exe start
   ```

原始 master 代理会在重启时自动将其角色恢复为 master。

15.9.3. 重新建立客户端连接

重启代理服务器后，将客户端应用程序重新连接到这些代理。以下小节介绍了如何重新连接内部和外部客户端应用程序。

15.9.3.1. 重新重新连接内部客户端

内部客户端是与恢复的代理相同的、之前失败的数据中心。要重新连接内部客户端，请重新启动它们。每个客户端应用程序都会重新连接到其连接配置中指定的恢复的 master 代理。

有关配置代理网络连接的详情，请参考 第 2 章 在网络连接中配置接收器和连接器。

15.9.3.2. 重新连接外部客户端

外部客户端是那些在之前出现故障的数据中心外运行的。根据您的客户端类型，以及 配置外部代理客户端 的信息，您可以将客户端配置为自动切换到备份代理，或者手动建立这个连接。当您恢复之前失败的数据中心时，您可以以类似的方式从客户端重新建立连接到恢复的 master 代理，如下所述。

- 如果您将外部客户端配置为自动切换到备份代理，则在关闭备份代理并重启原始 master 代理时，客户端会自动回退到原始 master 代理。

- 如果在发生数据中心中断时手动将外部客户端连接到备份代理，您必须手动将客户端重新连接到您重启的原始 master 代理。

第 16 章 使用代理连接配置多站点、容错消息传递系统

大规模企业消息传递系统通常具有位于地理上分布式数据中心的离散代理集群。如果数据中心停机，系统管理员可能需要保留现有的消息传递数据，并确保客户端应用程序可以继续生成和使用消息。您可以使用代理连接来确保在数据中心中断期间消息传递系统的连续性。这种类型的解决方案称为 多站点、容错架构。

注意

仅支持用于代理连接的代理间通信的 AMQP 协议。客户端可以使用任何受支持的协议。目前，消息将通过镜像过程转换为 AMQP。

以下小节解释了如何使用代理连接保护您的消息传递系统不受数据中心中断的影响：

- 第 16.1 节 “关于代理连接”
- 第 16.2 节 “配置代理镜像”

注意

多站点容错不是 数据中心内 高可用性(HA)代理冗余的替代。基于 live-backup 组的代理冗余提供对单个集群中单一代理故障的自动保护。相反，多站点容错可防止大规模数据中心中断。

16.1. 关于代理连接

使用代理连接，代理可以建立到另一个代理的连接，并将信息与代理和来自那个代理进行镜像。

AMQP 服务器连接

代理可以使用 AMQP 协议使用代理连接来启动到其他端点的连接。例如，这意味着代理可以连接到其他 AMQP 服务器，并在这些连接上创建元素。

AMQP 服务器连接支持以下类型的操作：

- mirror - 代理使用 AMQP 连接到另一个代理，并复制消息，并通过线路发送确认信息。
- senders - 在特定队列上接收的消息传输到另一个代理。
- Receivers - 代理从另一个代理拉取信息。
- peers - 代理在 AMQ Interconnect 端点上创建发送者和接收器。

本章论述了如何使用代理连接来创建容错系统。有关发送方、接收器和对等选项的信息，请参阅 第 17 章 桥接代理。

以下事件通过镜像发送：

- Message send - 发送到一个代理的消息将"复制"到目标代理。
- message confirmment - Acknowledgements 删除一个代理中的信息将发送到目标代理。
- 队列和地址创建。
- 队列和地址删除。

注意

如果消息在目标镜像上的消费者处于待处理状态，则确认不会成功，且消息可能由这两个代理发送。

镜像不会阻止任何操作，不会影响代理的性能。

代理只镜像配置镜像的时间点的消息。以前现有消息不会转发到其他代理。

16.2. 配置代理镜像

您可以使用代理连接来镜像一对代理之间的消息。任何时候只能有一个代理处于活跃状态。

先决条件

- 您有两个工作代理。

流程

1. 在 broker.xml 文件中为第一个代理创建一个 broker-connections 元素，例如：

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC1">
    <mirror/>
  </amqp-connection>
</broker-connections>
```

<hostname>

其他代理实例的主机名。

<port>

在其他主机上代理使用的端口。

第一个代理中的所有消息都会镜像到第二个代理，但创建镜像前存在的消息不会被镜像 (mirror)。

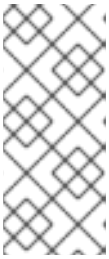
2. 如果您希望第一个代理同步镜像信息，以确保镜像代理为灾难恢复的 up-to-date，请在代理的 amqp-connection 元素中设置 sync=true 属性，如下例所示。

同步镜像要求代理发送到镜像代理的消息同时写入这两个代理的卷。在这两个代理中都完成写入操作后，源代理会确认写入请求已完成，并控制返回给客户端。

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror sync="true"/>
  </amqp-connection>
</broker-connections>
```

注意

如果无法在镜像代理上完成写入请求，例如，如果代理不可用，则客户端连接会被阻断，直到镜像可用来完成最新的写入请求。



注意

示例 DC1 中的代理连接名称用于创建名为 $ACTIVEMQ_ARTEMIS_MIRROR_mirror 的队列。确保对应的代理被配置为接受这些消息，即使队列在该代理上不可见。

3. 在 broker.xml 文件中为第二个代理创建一个 broker-connections 元素，例如：

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror/>
  </amqp-connection>
</broker-connections>
```

4. 如果您希望第二个代理同步镜像消息，请在代理的 amqp-connection 元素中设置 sync=true 属性。例如：

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror sync="true"/>
  </amqp-connection>
</broker-connections>
```

5. （可选）根据需要为镜像配置以下参数。

queue-removal

指定是否发送队列或地址删除事件。默认值为 true。

message-acknowledgments

指定是否发送消息确认。默认值为 true。

queue-creation

指定是否发送队列或地址创建事件。默认值为 true。

例如：

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror sync="true" queue-removal="false" message-acknowledgments ="false" queue-
creation="false"/>
  </amqp-connection>
</broker-connections>
```

6. （可选）自定义代理重试尝试确认目标镜像上的信息。

对于不在队列内存中的消息，目标镜像上可能会收到确认。要给代理有足够的时间在目标镜像上重试确认消息，您可以为您的环境自定义以下参数：

mirrorAckManagerQueueAttempts

代理在内存中查找消息的尝试次数。默认值为 **5**。如果代理在指定尝试次数后没有在内存中找到信息，代理会在页文件中搜索信息。

mirrorAckManagerPageAttempts

如果内存中找不到消息，代理会在页文件中查找消息的次数。默认值为 **2**。

mirrorAckManagerRetryDelay

代理尝试在内存中确认和页文件之间的间隔（以毫秒为单位）。

指定 **broker-connections** 元素之外的任何这些参数。例如：

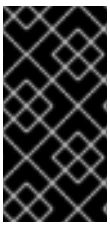
```
<mirrorAckManagerQueueAttempts>8</mirrorAckManagerQueueAttempts>

<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror/>
  </amqp-connection>
</broker-connections>
```

7. （可选）如果在目标镜像上对信息进行分页，如果您希望代理协调将消息写入页面文件的重复检测信息，请将 **mirrorPageTransaction** 设置为 **true**。

如果 **mirrorPageTransaction** 属性设置为 **false**，这是默认值，且代理之间发生通信失败，则会在个别情况下写入目标镜像。

将此参数设置为 true 可增加代理的内存用量。

8. 使用 第 15.6 节 “在多站点、容错消息传递系统中配置客户端” 中记录的说明配置客户端，请注意，没有共享存储。



重要

红帽不支持在镜像配置中使用来自这两个代理的客户端应用程序。要防止客户端在这两个代理上消耗消息，请在其中一个代理中禁用客户端 acceptors。

第 17 章 桥接代理

网桥提供了一种连接两个代理的方法，将消息从一个代理转发到另一个代理。

可用的网桥如下：

Core

core-bridge 示例演示了在一个代理上部署的核心网桥，该代理会消耗来自本地队列的消息并将其转发到第二个代理上的地址。

mirror

请查看 第 16 章 使用代理连接配置多站点、容错消息传递系统

发件人和接收器

请查看 第 17.1 节 “代理连接的发件人和接收器配置”

peer

请查看 第 17.2 节 “代理连接的对等配置”

注意

Core 网桥的 broker.xml 元素是 网桥。其他桥接技术使用 < broker-connection> 元素。

17.1. 代理连接的发件人和接收器配置

通过在 broker.xml 的 < broker-connections > 部分中创建发件人或接收器代理连接元素，可以将代理连接到另一个代理。

对于 发送者，代理会在队列上创建消息消费者，该队列将消息发送到另一个代理。

对于 接收器，代理会在接收来自另一个代理的消息的地址上创建一个消息制作者。

这两个元素都充当消息网桥。但是，进程消息不需要额外的开销。发件人和接收器的行为与代理中的任

何消费者或生成者一样。

特定的队列可以通过发送方或接收器配置。通配符表达式可用于将发件人和接收器与特定的地址 或一组 地址匹配。在配置发送者或接收方时，可以设置以下属性：

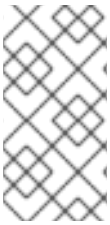
- address-match ：使用通配符表达式将发件人或接收器与特定地址 或一组 地址匹配。
- queue-name ：配置特定队列的发送者或接收器。
- 使用地址表达式：

```
<broker-connections>
 <amqp-connection uri="tcp://HOST:PORT" name="other-server">
  <sender address-match="queues.#"/>
  <!-- notice the local queues for remotequeues.# need to be created on this broker -->
  <receiver address-match="remotequeues.#"/>
 </amqp-connection>
</broker-connections>

<addresses>
 <address name="remotequeues.A">
  <anycast>
   <queue name="remoteQueueA"/>
  </anycast>
 </address>
 <address name="queues.B">
  <anycast>
   <queue name="localQueueB"/>
  </anycast>
 </address>
</addresses>
```

- 使用队列名称：

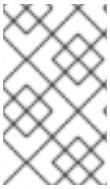
```
<broker-connections>
 <amqp-connection uri="tcp://HOST:PORT" name="other-server">
  <receiver queue-name="remoteQueueA"/>
  <sender queue-name="localQueueB"/>
 </amqp-connection>
</broker-connections>

<addresses>
  <address name="remotequeues.A">
   <anycast>
    <queue name="remoteQueueA"/>
```

```
    </anycast>
  </address>
  <address name="queues.B">
    <anycast>
      <queue name="localQueueB"/>
    </anycast>
  </address>
</addresses>
```



注意

Receivers 只能与已存在的本地队列匹配。因此，如果使用接收器，请确保预先在本地创建队列。否则，代理无法与远程队列和地址匹配。



注意

不要创建具有相同目的地的发件人和接收器，因为这会创建发送和接收的无限循环。

17.2. 代理连接的对等配置

代理可以被配置为连接到 AMQ Interconnect 实例的对等点，并指示代理将充当该路由器上配置的给定 AMQP 方法地址的存储和转发队列。在这种情况下，客户端使用某种方式地址连接到路由器来发送和接收消息，路由器将这些消息路由到代理上的队列或接收消息。

这个对等配置会为代理连接配置中的每个目的地创建一个发件人和接收器对。这些对包括使路由器能够与代理协作的配置。此功能避免路由器启动连接并创建自动链接的要求。

有关可能路由器配置的更多信息，请参阅使用 AMQ Interconnect 路由器。

使用对等配置时，与存在发件人和接收器相同的属性。例如，名称开头队列 队列 的配置。充当匹配路由器方法地址的存储：

```
<broker-connections>
  <amqp-connection uri="tcp://HOST:PORT" name="router">
    <peer address-match="queues.#"/>
  </amqp-connection>
</broker-connections>

<addresses>
  <address name="queues.A">
    <anycast>
      <queue name="queues.A"/>
```

```
    </anycast>
  </address>
  <address name="queues.B">
    <anycast>
      <queue name="queues.B"/>
    </anycast>
  </address>
</addresses>
```

路由器上必须有一个匹配的地址方法配置。这指示它将代理附加的特定路由器地址视为方法。例如，请查看以下基于前缀的路由器地址配置：

```
address {
    prefix: queue
    waypoint: yes
}
```

有关这个选项的更多信息，请参阅使用 AMQ Interconnect 路由器。

注意

不要使用 peer 选项直接连接到另一个代理。如果您使用这个选项连接到另一个代理，则所有消息都会立即可用，创建发送和接收的无限回显。

第 18 章 日志记录

AMQ Broker 使用 Apache Log4j 2 logging 工具提供消息日志记录。安装代理时，它在 <broker_instance_dir> /etc/log4j2.properties 文件中有一个默认的 Log4j 2 配置。使用默认配置时，日志记录器会同时写入控制台和文件。

下表中显示了 AMQ Broker 中的日志记录器。

logger	描述
org.apache.activemq.artemis.core.server	记录代理内核
org.apache.activemq.artemis.journal	日志日志调用
org.apache.activemq.artemis.utils	日志工具调用
org.apache.activemq.artemis.jms	日志 JMS 调用
org.apache.activemq.artemis.integration.bootstrap	日志 bootstrap 调用
org.apache.activemq.audit.base	记录访问所有 JMX 对象方法
org.apache.activemq.audit.message	记录消息操作，如生产、消耗和浏览消息
org.apache.activemq.audit.resource	记录身份验证事件、从 JMX 或 AMQ Broker 管理控制台创建和删除代理资源，并在管理控制台中浏览信息

18.1. 更改日志记录级别

您可以为每个日志程序配置日志级别，在 <logger name>.level 行的日志程序名称后，如以下 apache.activemq.artemis.core.server 日志程序：

```
logger.artemis_server.name=org.apache.activemq.artemis.core.server
logger.artemis_server.level=INFO
```

审计日志记录器的默认日志记录级别为 OFF，这意味着禁用了日志记录。AMQ Broker 中提供的其他日志记录器的默认日志记录级别为 INFO。有关 Log4j 2 中可用日志级别的详情，请查看 Log4j 2 文档。

18.2. 启用审计日志记录

有三个审计日志记录器可供您启用：基本的审计日志记录器、消息审计日志记录器和资源审计日志记录

器。

基本审计日志记录器(org.apache.activemq.audit.base)

记录到所有 JMX 对象方法的访问，如创建和删除地址和队列。日志 不 指示这些操作是成功还是失败。

消息审计日志记录器(org.apache.activemq.audit.message)

记录与消息相关的代理操作，如生产、消耗或浏览消息。

资源审计日志记录器(org.apache.activemq.audit.resource)

记录来自客户端、路由和 AMQ Broker 管理控制台的身份验证成功或失败。另外，记录从 JMX 或管理控制台创建、更新或删除队列，并在管理控制台中浏览消息。

您可以独立启用每个审计日志记录器。默认情况下，日志级别设置为 OFF，这意味着每个审计日志记录器都禁用日志记录。要启用其中一个审计日志记录器，请将日志级别从 OFF 更改为 INFO。例如：

```
logger.audit_base = INFO, audit_log_file
```

注意

INFO 是 logger.org.apache.activemq.audit.base,logger.org.apache.activemq.audit.message, 和 logger.org.apache.activemq.audit.resource audit loggers 的唯一可用日志记录级别。

重要

消息审计日志记录器在代理上的性能密集型路径上运行。启用日志记录器可能会对代理的性能造成负面影响，特别是代理在高消息传递负载下运行。红帽建议不要在需要高吞吐量的消息传递系统中启用审计日志记录。

18.3. 客户端或嵌入式服务器日志记录

如果要在客户端上启用日志记录，则需要在支持 SLF4J facade 的应用程序中包含日志实施。如果使用 Maven，请为 Log4j 2 添加以下依赖项：

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>artemis-jms-client</artifactId>
```

```
    <version>2.28.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.19.0</version>
  </dependency>
```

您可以在 classpath 中的 log4j2.properties 文件中提供 Log4j 2 配置。或者，您可以使用 log4j2.configurationFile 系统属性指定自定义配置文件。例如：

```
-Dlog4j2.configurationFile=file:///path/to/custom-log4j2-config.properties
```

以下是客户端的 log4j2.properties 文件示例：

```
# Log4J 2 configuration

# Monitor config file every X seconds for updates
monitorInterval = 5

rootLogger = INFO, console, log_file

logger.activemq.name=org.apache.activemq
logger.activemq.level=INFO

# Console appender
appender.console.type=Console
appender.console.name=console
appender.console.layout.type=PatternLayout
appender.console.layout.pattern=%d %-5level [%logger] %msg%n

# Log file appender
appender.log_file.type = RollingFile
appender.log_file.name = log_file
appender.log_file.fileName = log/application.log
appender.log_file.filePattern = log/application.log.%d{yyyy-MM-dd}
appender.log_file.layout.type = PatternLayout
appender.log_file.layout.pattern = %d %-5level [%logger] %msg%n
appender.log_file.policies.type = Policies
appender.log_file.policies.cron.type = CronTriggeringPolicy
appender.log_file.policies.cron.schedule = 0 0 0 * * ?
appender.log_file.policies.cron.evaluateOnStartup = true
```

18.4. AMQ BROKER 插件支持

AMQ 支持自定义插件。您可以使用插件来记录许多不同类型的事件的信息，否则只能通过调试日志获得。可以一起注册、关联和执行多个插件。该插件根据注册的顺序执行，即第一个插件注册始终会首先执行。

您可以创建自定义插件并使用 ActiveMQServerPlugin 接口实施它们。此接口可确保插件在 classpath 上，并在代理中注册。由于所有接口方法都是默认实现的，所以您必须仅添加需要实施的必要行为。

18.4.1. 在类路径中添加插件

通过将相关的 .jar 文件添加到 < broker_instance_dir> /lib 目录，将自定义创建的代理插件添加到代理运行时中。

如果您使用嵌入式系统，请将 .jar 文件放在嵌入式应用程序的常规类路径中。

18.4.2. 注册插件

您必须在 broker.xml 配置文件中添加 broker-plugins 元素来注册插件。您可以使用 属性 子元素指定插件配置值。这些属性在插件实例化后读取并传递到插件的 init (Map<String, String>)操作。

```
<broker-plugins>
  <broker-plugin class-name="some.plugin.UserPlugin">
    <property key="property1" value="val_1" />
    <property key="property2" value="val_2" />
  </broker-plugin>
 </broker-plugins>
```

18.4.3. 以编程方式注册插件

要以编程方式注册插件，请使用 registerBrokerPlugin （） 方法并传递插件的新实例。以下示例显示了注册 UserPlugin 插件：

```
Configuration config = new ConfigurationImpl();

config.registerBrokerPlugin(new UserPlugin());
```

18.4.4. 记录特定事件

默认情况下，AMQ 代理提供 LoggingActiveMQServerPlugin 插件来记录特定的代理事件。LoggingActiveMQServerplugin 插件默认为注释，且不会记录任何信息。

下表描述了每个插件属性。将配置属性值设为 true 以记录事件。

属性	描述
LOG_CONNECTION_EVENTS	创建或销毁连接时的信息。
LOG_SESSION_EVENTS	创建或关闭会话时记录信息。
LOG_CONSUMER_EVENTS	创建或关闭消费者时的日志信息。
LOG_DELIVERING_EVENTS	当消息被发送到消费者以及当消息被消费者确认时，日志信息。
LOG_SENDING_EVENTS	当消息被发送到地址以及在代理内路由消息时，日志信息。
LOG_INTERNAL_EVENTS	当一个队列创建或销毁时，会在部署网桥时以及发生关键故障时记录队列创建或销毁的信息。
LOG_ALL_EVENTS	记录以上所有事件的信息。

要配置 LoggingActiveMQServerPlugin 插件来记录连接事件，在 broker.xml 配置文件中取消注释 <broker-plugins> 部分。所有事件的值都在注释的默认示例中设置为 true。

```
<configuration ...>
...
<!-- Uncomment the following if you want to use the Standard LoggingActiveMQServerPlugin plugin
to log in events -->
         <broker-plugins>
        <broker-plugin class-
name="org.apache.activemq.artemis.core.server.plugin.impl.LoggingActiveMQServerPlugin">
           <property key="LOG_ALL_EVENTS" value="true"/>
           <property key="LOG_CONNECTION_EVENTS" value="true"/>
           <property key="LOG_SESSION_EVENTS" value="true"/>
           <property key="LOG_CONSUMER_EVENTS" value="true"/>
           <property key="LOG_DELIVERING_EVENTS" value="true"/>
           <property key="LOG_SENDING_EVENTS" value="true"/>
           <property key="LOG_INTERNAL_EVENTS" value="true"/>
        </broker-plugin>
      </broker-plugins>
...
</configuration>
```

当您更改 < broker-plugins> 部分中的配置参数时，您必须重启代理来重新载入配置更新。这些配置更改不会基于 configuration-file-refresh-period 设置重新载入。

当日志级别设置为 INFO 时，会在事件发生后记录条目。如果日志级别设置为 DEBUG，则会为事件之前和之后生成日志条目，例如 beforeCreateConsumer （） 和 afterCreateConsumer （）。如果日志

*级别设置为 **DEBUG**，则日志记录器会记录通知的更多信息（如果可用）。*

附录 A. ACCEPTOR 和 CONNECTOR 配置参数

下表详细介绍了一些用于配置 Netty 网络连接的可用参数。参数及其值附加到连接字符串的 URI 中。如需更多信息，请参阅 网络连接中配置接收器和连接器。每个表按名称列出参数，并记录这些参数是否与接收器或连接器一起使用。例如，您只能在 acceptors 中使用一些参数。

注意

所有 Netty 参数都在类 org.apache.activemq.artemis.core.remoting.impl.netty.TransportConstants 中定义。源代码可以在客户门户网站 上下载。http://access.redhat.com/downloads

表 A.1. Netty TCP 参数

参数	与... 一起使用	描述
batchDelay	两者都	在将数据包写入接受者或连接器之前，可将代理配置为批量写入最多 **batchDelay** 毫秒。这可提高非常小的消息的整体吞吐量。它确实会牺牲消息传输的平均延迟增加。默认值为 **0** ms。
connectionsAllowed	acceptors	限制接受者允许的连接数量。当达到这个限制时，会为日志发出 DEBUG 级别的消息，连接被拒绝。使用的客户端类型决定了连接被拒绝时会发生什么。默认值为 **-1**， 这意味着接受者允许的连接数量没有限制。
directDeliver	两者都	当消息到达服务器并传送到等待消费者时，默认情况下，发送会在与消息到达的同一线程上进行。这在具有相对较小的消息和少量消费者的环境中提供了很好的延迟，但以总体吞吐量和可扩展性为代价，特别是在多核机器上。如果您希望获得最低延迟并可能降低吞吐量，那么您可以将默认值用于 **directDeliver**，即 **true**。如果您要对延迟进行一些小的额外点击，但希望最高吞吐量设置为 **false**。
handshake-timeout	acceptors	防止未授权客户端打开大量连接并保持打开状态。由于每个连接都需要一个文件句柄，所以它会消耗资源，然后可供其他客户端使用。 此超时限制连接可以消耗资源的时间，而无需经过身份验证。验证连接后，您可以使用资源限制设置来限制资源消耗。 默认值为 **10** 秒。您可以将其设置为任何其他整数值。您可以通过将选项设置为 0 或负整数来关闭这个选项。 编辑超时值后，您必须重启代理。

参数	与... 一起使用	描述
localAddress	连接器	指定客户端在连接到远程地址时使用的本地地址。这通常用于应用服务器，或者在运行嵌入式时用于控制用于出站连接的地址。如果没有设置本地地址，则连接器将使用任何可用的本地地址。
localPort	连接器	指定客户端在连接到远程地址时使用的本地端口。这通常用于应用服务器，或者在运行嵌入式时用于控制用于出站连接的端口。如果使用默认值，即 0，则连接器将让系统获取临时端口。有效端口为 0 到 65535
nioRemotingThreads	两者都	当配置为使用 NIO 时，代理默认使用与处理传入的数据包所报告的内核数（或超线程）相同的线程数。**如果要覆盖这个值，您可以通过指定此参数来设置线程数量。此参数的默认值为 -1，这表示使用通过 从 Runtime.getRuntime().availableProcessors() * 3 获得的值。**
tcpNoDelay	两者都	如果为 **true**，则禁用 Nagle 的算法。这是 Java （客户端）套接字选项。默认值为 **true**。
tcpReceiveBufferSize	两者都	确定 TCP 接收缓冲区的大小（以字节为单位）。默认值为 **32768**。
tcpSendBufferSize	两者都	确定 TCP 发送缓冲区的大小（以字节为单位）。默认值为 **32768**。 应根据网络的带宽和延迟调整 TCP 缓冲区大小。 总的 TCP 发送/接收缓冲区大小应计算为： buffer_size = bandwidth * RTT. 其中带宽每秒字节数，网络往返用时(RTT)以秒为单位。可以使用 **ping** 实用程序轻松测量 RTT。 对于快速网络，您可能想要从默认值增加缓冲区大小。

表 A.2. Netty HTTP 参数

参数	与... 一起使用	描述
httpClientIdleTime	acceptors	在发送空 HTTP 请求前，客户端可以闲置多久，以保持连接处于活动状态。
httpClientIdleScanPeriod	acceptors	扫描空闲客户端的频率（以毫秒为单位）。

参数	与... 一起使用	描述
httpEnabled	acceptors	不再需要。有了单一端口支持，代理现在将自动检测是否使用 HTTP 并配置其自身。
httpRequiresSessionId	两者都	如果为 **true**，则客户端在第一次调用后将等待一个会话 ID。当 HTTP 连接器连接到 servlet 接受器时使用。不建议使用此配置。
httpResponseTime	acceptors	服务器在发送空 HTTP 响应前可以等待的时间，以保持连接处于活动状态。
httpServerScanPeriod	acceptors	扫描需要响应的客户端的频率，以毫秒为单位。

表 A.3. Netty TLS/SSL 参数

参数	与... 一起使用	描述
enabledCipherSuites	两者都	用于 SSL 通信的密码套件的逗号分隔列表。 指定客户端应用程序支持的最安全密码套件。如果您指定了代理和客户端通用的、以逗号分隔的密码套件列表，或者您没有指定任何密码套件、代理和客户端相互协商要使用的密码套件。如果您不知道要指定哪个密码套件，您可以首先与以 debug 模式运行的客户端建立 broker-client 连接，以验证代理和客户端通用的密码套件。然后，在代理上配置 **enabledCipherSuites**。 可用的密码套件取决于代理和客户端使用的 TLS 协议版本。如果在升级代理后默认 TLS 协议版本有变化，您可能需要选择早期的 TLS 协议版本，以确保代理和客户端可以使用通用密码套件。如需更多信息，请参阅 **enabledProtocols**。
enabledProtocols	两者都	用于 SSL 通信的 TLS 协议版本的逗号分隔列表。如果没有指定 TLS 协议版本，代理将使用 JVM 的默认版本。 如果代理使用 JVM 的默认 TLS 协议版本，且升级代理后该版本会改变，代理和客户端使用的 TLS 协议版本可能会不兼容。虽然建议您使用更新的 TLS 协议版本，但您可以在 **enabledProtocols** 中指定较早的版本，以便与不支持较新的 TLS 协议版本的客户端进行交互。
forceSSLParameters	连接器	控制是否在连接器上设置为参数设置的任何 SSL 设置，而不是 JVM 系统属性（包括 **javax.net.ssl** 和 AMQ Broker 系统属性）来为这个连接器配置 SSL 上下文。 有效值为 **true** 或者 **false**。默认值为 **false**。

参数	与... 一起使用	描述
keyStorePassword	两者都	在 acceptor 上使用时，这是服务器端密钥存储的密码。 在连接器中使用时，这是客户端密钥存储的密码。这只有在您使用双向 SSL （即 mutual 身份验证）时与连接器相关。虽然可以在服务器上配置这个值，但它被下载并供客户端使用。如果客户端需要使用与服务器上设置的不同密码，那么它可以使用 customary **javax.net.ssl.keyStorePassword** 系统属性或 ActiveMQ 特定的 **org.apache.activemq.ssl.keyStorePassword** 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使用标准的 Java 系统属性，则特定于 ActiveMQ 的系统属性非常有用。
keyStorePath	两者都	当在接受人上使用时，这是包含服务器证书的服务器上 SSL 密钥存储的路径（无论是由权威自签名还是由权威签名）。 当在连接器中使用时，这是包含客户端证书的客户端 SSL 密钥存储的路径。这只有在您使用双向 SSL （即 mutual 身份验证）时与连接器相关。虽然在服务器上配置了这个值，但它被下载并供客户端使用。如果客户端需要使用与服务器上设置的不同路径，那么它可以使用 customary **javax.net.ssl.keyStore** 系统属性或特定于 ActiveMQ 的 **org.apache.activemq.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使用标准的 Java 系统属性，则特定于 ActiveMQ 的系统属性非常有用。
keyStoreAlias	两者都	当在接受器上使用时，这是在客户端连接时，密钥存储中要存在的键的别名。在连接器中使用时，这是当客户端连接到时，密钥存储中要存在的键的别名。这只适用于使用双向 SSL （即 mutual 身份验证）的连接器。
needClientAuth	acceptors	告知客户端连接到此接收器，或者需要双向 SSL。有效值为 **true** 或者 **false**。默认值为 **false**。
sslEnabled	两者都	必须 **true** 才能启用 SSL。默认值为 **false**。
sslAutoReload	acceptors	如果设置为 **true**，代理会检查对 SSL 配置的更改，如密钥存储和信任存储路径，并在检测到更改时自动重新载入配置。默认值为 **false**。代理检查更改的时间间隔由 **configuration-file-refresh-period** 元素的值决定。如需更多信息，请参阅 Reloading 配置更新。

参数	与... 一起使用	描述
trustManagerFactoryPlugin	两者都	定义实现 **org.apache.activemq.artemis.api.core.TrustManagerFactoryPlugin** 的类名称。 这是一个简单的接口，它有一个返回 **javax.net.ssl.TrustManagerFactory** 的方法。当底层 **javax.net.ssl.ssl.SSLContext** 初始化时，**TrustManagerFactory** 会被使用。这允许精细自定义代理和客户端信任的人员。 **trustManagerFactoryPlugin** 的值优先于应用到信任管理器的所有其他 SSL 参数（即 **trustAll**,**trustStoreProvider**,**trustStorePath**,**trustStorePassword**, 和 **crlPath**）。 您需要将任何指定的插件放在代理的 Java 类路径上。您可以使用 &lt ***;broker_instance_dir>*** **/lib** 目录，因为它默认为 classpath 的一部分。
trustStorePassword	两者都	在接收器中使用时，这是服务器端信任存储的密码。只有在使用双向 SSL （即 mutual 身份验证）时，这才与接受者相关。 当在连接器中使用时，这是客户端信任存储的密码。虽然可以在服务器上配置这个值，但它被下载并供客户端使用。如果客户端需要使用与服务器上设置的不同密码，那么它可以使用 customary **javax.net.ssl.trustStorePassword** 系统属性或 ActiveMQ 特定的 **org.apache.activemq.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使用标准的 Java 系统属性，则特定于 ActiveMQ 的系统属性非常有用。
sniHost	两者都	当在接收器中使用时，**sniHost** 是一个正则表达式，用于匹配传入 SSL 连接中的 **server_name** 扩展（有关此扩展的更多信息，请参阅 https://tools.ietf.org/html/rfc6066）。如果名称不匹配，则拒绝与接收器的连接。如果发生此情况，则会记录 WARN 消息。 如果传入连接不包含 **server_name** 扩展，则接受连接。 在连接器中使用时，**sniHost** 值用于 SSL 连接上的 **server_name** 扩展。
sslProvider	两者都	用于在 **JDK** 和 **OPENSSL** 之间更改 SSL 提供程序。默认值为 **JDK**。 如果设置为 **OPENSSL**，您可以在 classpath 中添加 **netty-tcnative**，以使用原生安装的 OpenSSL。 如果要使用通过 OpenSSL 支持的特殊 ciphersuite-elliptic curve 组合，但没有通过 JDK 供应商支持，这个选项很有用。

参数	与... 一起使用	描述
trustStorePath	两者都	当用于接受者时，这是包含服务器信任的所有客户端的密钥的服务器端 SSL 密钥存储的路径。只有在使用双向 SSL （即 mutual 身份验证）时，这才与接受者相关。 当在连接器中使用时，这是客户端 SSL 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这个值，但它被下载并供客户端使用。如果客户端需要使用与服务器上设置的不同路径，那么它可以使用 customary **javax.net.ssl.trustStore** 系统属性或特定于 ActiveMQ 的 **org.apache.activemq.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使用标准的 Java 系统属性，则特定于 ActiveMQ 的系统属性非常有用。
useDefaultSslContext	连接器	允许连接器使用 "default" SSL 上下文（通过 **SSLContext.getDefault （）**），它可以通过客户端以编程方式设置（通过 **SSLContext.setDefault (SSLContext)**）。 如果此参数设为 **true**，则忽略 **sslEnabled** 以外的所有其他与 SSL 相关的参数。有效值为 **true** 或者 **false**。默认值为 **false**。
verifyHost	两者都	在连接器中使用时，服务器 SSL 证书的 CN 或 Subject Alternative Name 值会与连接到的主机名进行比较，以验证它们是否匹配。这对单向和双向 SSL 都很有用。 当在接收器中使用时，连接客户端的 SSL 证书的 CN 或 Subject Alternative Name 值会与其主机名进行比较，以验证它们是否匹配。这仅适用于双向 SSL。 有效值为 **true** 或者 **false**。连接器的默认值为 **true**，接受器的默认值是 **false**。
wantClientAuth	acceptors	告知客户端连接到此接收器，或者请求双向 SSL，但不需要。有效值为 **true** 或者 **false**。默认值为 **false**。 如果属性 **needClientAuth** 设为 **true**，则该属性具有优先权，并且 **wantClientAuth** 将被忽略。

附录 B. 地址设置配置元素

下表列出了 address-setting 的所有配置元素。请注意，一些元素被标记为 DEPRECATED。使用推荐的替换来避免潜在的问题。

表 B.1. 地址设置元素

名称	描述
address-full-policy	确定配置了 **max-size-bytes** 的地址变为满时会发生什么。可用的策略有： **PAGE** ：发送到完整地址的消息将传给磁盘。 **DROP** ：发送到完整地址的消息将静默丢弃。 **FAIL**: 发送到完整地址的消息将被丢弃，消息制作者将收到异常。 **BLOCK**: 当消息尝试并发送任何进一步消息时，消息制作者将阻止。  **注意** BLOCK 策略仅适用于 AMQP、OpenWire 和核心协议协议，因为它们具有流控制。
auto-create-addresses	客户端在客户端发送消息时是否自动创建地址，或尝试从映射到队列的队列中的消息来自动创建地址。默认值为 **true**。
auto-create-dead-letter-resources	指定代理是否自动创建死信地址和队列来接收未发送的消息。默认值为 **false**。 如果参数设为 **true**，代理会自动创建 < **address** > 元素来定义死信地址和关联的死信队列。自动创建的 < **address> 元素的名称与** 您为 < **dead-letter-address> 指定的 name 值匹配**。
auto-create-jms-queues	DEPRECATED ：改为使用 **auto-create-queues**。确定此代理是否应该自动创建与 JMS 生成者或消费者尝试使用此类队列时与地址设置匹配的 JMS 队列。默认值为 **false**。
auto-create-jms-topics	DEPRECATED ：改为使用 **auto-create-queues**。确定此代理是否应该自动创建与 JMS 生成者或消费者尝试使用此类队列时与地址设置匹配的 JMS 主题。默认值为 **false**。
auto-create-queues	当客户端向发送消息或尝试使用来自队列的消息时，是否自动创建队列。默认值为 **true**。
auto-delete-addresses	当代理不再有任何队列时，是否删除自动创建的地址。默认值为 **true**。
auto-delete-jms-queues	DEPRECATED ：改为使用 **auto-delete-queues**。决定，当没有消费者且没有信息时，AMQ Broker 是否应该自动删除自动创建的 JMS 队列。默认值为 **false**。

名称	描述
auto-delete-jms-topics	DEPRECATED ：改为使用 **auto-delete-queues**。决定在没有消费者且没有信息时，AMQ Broker 是否应该自动删除自动创建的 JMS 主题。默认值为 **false**。
auto-delete-queues	当队列没有消费者且没有消息时，是否删除自动创建的队列。默认值为 **true**。
config-delete-addresses	重新加载配置文件后，此设置指定如何处理从配置文件中删除的地址（及其队列）。您可以指定以下值： **OFF （默认）** 重新加载配置文件时，不会删除该地址。 **FORCE** 重新加载配置文件时，会删除地址及其队列。如果队列中有任何消息，它们也会被删除。
config-delete-queues	重新加载配置文件后，此设置指定如何处理从配置文件中删除的队列。您可以指定以下值： **OFF （默认）** 重新加载配置文件时，队列不会被删除。 **FORCE** 重新加载配置文件时，队列将被删除。如果队列中有任何消息，它们也会被删除。
dead-letter-address	代理发送死消息的地址。
dead-letter-queue-prefix	代理应用到自动创建的死信队列的名称的前缀。默认值为 **DLQ.**
dead-letter-queue-suffix	代理应用于自动创建的死信队列的后缀。默认值没有被定义（即，代理不会应用后缀）。
default-address-routing-type	自动创建的地址中使用的 route-type。默认值为 **MULTICAST**。
default-max-consumers	此队列中允许的最大消费者数量。默认值为 **200**。
default-purge-on-no-consumers	没有消费者后，是否清除队列的内容。默认值为 **false**。
default-queue-routing-type	自动创建队列上使用的路由类型。默认值为 **MULTICAST**。
enable-metrics	指定配置的指标插件，如 Prometheus 插件为匹配地址 *或一组* 地址收集指标。默认值为 **true**。
expiry-address	接收过期消息的地址。

名称	描述
expiry-delay	定义将使用默认过期时间的信息的过期时间（以毫秒为单位）。默认值为 **-1**，这代表没有过期时间。
last-value-queue	队列是否只使用最后的值。默认值为 **false**。
management-browse-page-size	管理资源可以浏览的消息数量。默认值为 **200**。
max-delivery-attempts	在发送到死信地址前尝试发送消息的次数。默认值为 **10**。
max-redelivery-delay	redelivery-delay 的最大值，以毫秒为单位。
max-size-bytes	此地址的最大内存大小，以字节为单位。当 **address-full-policy** 是 **PAGING**, **BLOCK**, 或 **FAIL** 时，这个值以字节标记（如 "K"、"Mb" 和 "GB"）指定。默认值为 **-1**，它表示无限字节。这个参数用于通过限制特定地址空间消耗的内存量来保护代理内存。此设置不代表客户端当前存储在代理地址空间中的字节数。它是代理内存使用率估算。这个值可能会因运行时条件和某些工作负载而异。建议您分配每个地址空间可负担的最大内存量。在典型的工作负载下，代理需要大约 150% 到 200% 的、内存中未处理消息的有效负载大小大约为 150%。
max-size-bytes-reject-threshold	当 **address-full-policy** 为 **BLOCK** 时使用。地址在代理开始拒绝消息前可以访问的最大大小（以字节为单位）。只适用于 AMQP 协议的 **max-size-bytes**。默认值为 **-1**，这意味着没有限制。
message-counter-history-day-limit	为这个地址保留消息计数器历史记录的天数。默认值为 **0**。
page-size-bytes	分页大小（以字节为单位）。还支持字节表示法，如 **K**、**Mb** 和 **GB**。默认值为 **10485760** 字节，几乎 10.5 MB。
redelivery-delay	在重新设计取消消息前等待的时间（毫秒）。默认值为 **0**。
redelivery-delay-multiplier	应用到 redelivery-delay 参数的倍数。默认值为 **1.0**。
redistribution-delay	定义在提交任何消息前，在队列中关闭最后一次消费者后等待的时间（毫秒）。默认值为 **-1**。
send-to-dla-on-no-route	当设置为 **true** 时，如果消息无法路由到任何队列，则会将消息发送到配置的死信地址。默认值为 **false**。
slow-consumer-check-period	检查缓慢消费者的频率（以秒为单位）。默认值为 **5**。
slow-consumer-policy	决定在识别缓慢消费者时会发生什么。有效选项为 **KILL** 或 **NOTIFY**。**KILL** 终止消费者的连接，这会影响到使用该连接的任何客户端线程。**NOTIFY** 向客户端发送 **CONSUMER_SLOW** 管理通知。默认值为 **NOTIFY**。

名称	描述
slow-consumer-threshold	在消费者被视为缓慢前允许的最小消息消耗率。以消息秒为单位的测量。默认值为 **-1，** 即未绑定。

附录 C. 集群连接配置元素

下表列出了 cluster-connection 的所有配置元素。

表 C.1. 集群连接配置元素

名称	描述
address	每个集群连接都只适用于与 **address** 字段中指定的值匹配的地址。如果没有指定地址，则所有地址都将进行负载均衡。 **address** 字段还支持以逗号分隔的地址列表。使用 exclude 语法 **!** 以防止匹配地址。以下是一些地址示例： **jms.eu** 匹配以 **jms.eu** 开头的所有地址。 **!jms.eu** 匹配除以 **jms.eu**开头的那些地址外的所有地址 **jms.eu.uk,jms.eu.de** 匹配以 **jms.eu.uk** 或 **jms.eu.de**开头的所有地址 **jms.eu,!jms.eu.uk** 匹配以 **jms.eu** 开头但不是以 **jms.eu.uk**开头的所有地址  **注意** 您不应该有多个带有重叠地址的集群连接（例如："europe"和"europe.news"），因为相同的消息可以在多个集群连接之间分布，可能会导致重复发送。
call-failover-timeout	在故障转移尝试期间调用时，请使用。默认值为 **-1，** 或没有超时。
call-timeout	当数据包通过集群连接发送时，且是一个阻塞调用，**call-timeout** 决定代理在抛出异常前等待（毫秒）回复的时长。默认值为 **30000**。
check-period	检查集群连接是否无法从另一个代理接收 ping 的时间间隔（以毫秒为单位）。默认值为 **30000**。
confirmation-window-size	用于向连接的代理发送确认的窗口的大小，以字节为单位。当代理收到 **confirmation-window-size** 字节时，它会通知其客户端。默认值为 **1048576**。值 **-1** 表示没有窗口。
connector-ref	标识将传送到集群中的其他代理的 **connector**，以便它们具有正确的集群拓扑。这个参数是必需的。
connection-ttl	决定集群连接在停止从集群中的特定代理接收消息时保持活动状态的时长。默认值为 **60000**。

名称	描述
discovery-group-ref	指向一个 **discovery-group**，用于与集群中的其他代理通信。此元素必须包含属性 **discovery-group-name**，它必须与之前配置的 **discovery-group** 的 **name** 属性匹配。
initial-connect-attempts	设置系统初始集群中尝试连接代理的次数。如果实现 max-retry，则此代理将被永久考虑，系统将不会将消息路由到这个代理。默认值为 **-1，** 即无限重试。
max-hops	将代理配置为将消息负载平衡到代理，这些代理只能与链中的其他代理间接连接。这可实现更复杂的拓扑，同时仍然提供消息负载平衡。默认值为 **1**，这意味着消息仅分发到直接连接到此代理的其他代理中。这个参数是可选的。
max-retry-interval	重试的最大延迟，以毫秒为单位。默认值为 **2000**。
message-load-balancing	决定如何在集群中的其他代理之间分发信息。包含 **message-load-balancing** 元素以启用负载平衡。默认值为 **ON_DEMAND**。您也可以提供一个值。有效值为： **OFF** 禁用负载平衡。 **STRICT** 启用负载平衡并将消息转发到具有匹配队列的所有代理，无论队列是否有活跃的消费者还是匹配的选择器。 **ON_DEMAND** 启用负载平衡并确保消息仅转发到具有匹配选择器的活动消费者的代理。 **OFF_WITH_REDISTRIBUTION** 禁用负载平衡，但确保消息仅在没有合适的本地消费者可用时仅转发到具有匹配选择器的活动消费者的代理。
min-large-message-size	如果消息大小（以字节为单位）大于 **min-large-message-size**，则在通过网络发送到其他群集成员时，它将被分成多个网段。默认值为 **102400**。
notification-attempts	设置在连接到集群时集群连接应该广播自身的次数。默认值为 **2**。
notification-interval	设置附加到集群时集群连接应广播其自身的频率，以毫秒为单位。默认值为 **1000**。
producer-window-size	生成者流控制集群连接的大小（以字节为单位）。默认情况下，它被禁用，但如果您在集群中使用真正大型消息，您可能需要设置值。值 **-1** 表示没有窗口。
reconnect-attempts	设置系统尝试重新连接到集群中的代理的次数。如果实现 max-retry，则此代理将被视为永久关闭，系统将停止将消息路由到这个代理。默认值为 **-1，** 即无限重试。

名称	描述
retry-interval	决定重试尝试之间的间隔（以毫秒为单位）。如果创建了集群连接，并且目标代理尚未启动或启动，则来自其他代理的集群连接将重试连接到目标，直到备份为止。这个参数是可选的。默认值为 500 毫秒。
retry-interval-multiplier	在每次重新连接尝试后增加 **retry-interval** 的倍数。默认值为 1。
use-duplicate-detection	集群连接使用网桥来链接代理，网桥可以配置为在转发的每个消息中添加重复的 ID 属性。如果网桥的目标代理崩溃，然后恢复，则消息可能会从源代理中重新发送。通过将 **use-duplicate-detection** 设置为 **true**，在目标代理时，所有重复的消息都会被过滤并忽略。默认值是 **true**。

附录 D. 命令行工具

AMQ Broker 包括一组命令行界面(CLI)工具，以便您管理消息传递日志。下表列出了每个工具的名称及其描述。

工具	描述
exp	使用特殊和独立的 XML 格式导出消息数据。
imp	使用 **exp** 提供的输出，将日志导入到正在运行的代理。
data	输出有关日志记录的报告，并压缩其数据。
encode	显示编码为 String 的日志的内部格式。
decode	从 encode 导入内部日志格式。

如需可用于每个工具的命令的完整列表，请使用 help 参数，后跟工具的名称。在以下示例中，CLI 输出列出了用户在输入命令 ./artemis help data 后对 data 工具使用的所有命令。

```
$ ./artemis help data

NAME
        artemis data - data tools group
        (print|imp|exp|encode|decode|compact) (example ./artemis data print)

SYNOPSIS
        artemis data
        artemis data compact [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data decode [--broker <brokerConfig>] [--suffix <suffix>]
                [--verbose] [--paging <paging>] [--prefix <prefix>] [--file-size <size>]
                [--directory <directory>] --input <input> [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data encode [--directory <directory>] [--broker <brokerConfig>]
                [--suffix <suffix>] [--verbose] [--paging <paging>] [--prefix <prefix>]
                [--file-size <size>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data exp [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data imp [--host <host>] [--verbose] [--port <port>]
                [--password <password>] [--transaction] --input <input> [--user <user>]
        artemis data print [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
```

```
COMMANDS
        With no arguments, Display help information

        print
            Print data records information (WARNING: don't use while a
            production server is running)

        ...
```

您可以在工具中使用帮助，以了解有关如何执行每个工具命令的更多信息。例如，CLI 在用户输入 *./artemis help data print* 后列出了 *data print* 命令的更多信息。

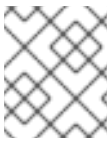
```
$ ./artemis help data print

NAME
        artemis data print - Print data records information (WARNING: don't use
        while a production server is running)

SYNOPSIS
        artemis data print [--bindings <binding>] [--journal <journal>]
                [--paging <paging>]

OPTIONS
        --bindings <binding>
            The folder used for bindings (default ../data/bindings)

        --journal <journal>
            The folder used for messages journal (default ../data/journal)

        --paging <paging>
            The folder used for paging (default ../data/paging)
```

附录 E. 消息传递日志配置元素

下表列出了与 AMQ Broker 消息传递日志相关的所有配置元素。

表 E.1. 地址设置元素

名称	描述
journal-directory	消息日志所在的目录。默认值为 < ***broker_instance_dir***>/data/journal。 为了获得最佳性能，日志应位于自己的物理卷上，以便最大程度降低磁盘头移动。如果日志位于与其他进程共享的卷中，可能会编写其他文件（例如，绑定日志、数据库或事务协调器），那么磁盘头也可能会在这些文件之间快速移动，从而大大降低性能。 使用 SAN 时，应为每个日志实例提供自己的 LUN （逻辑单元）。
create-journal-dir	如果设置为 **true**，则日志目录将在 **journal-directory** 中指定的位置自动创建（如果不存在）。默认值为 **true**。
journal-type	有效值为 **NIO** 或 **ASYNCIO**。 如果设置为 **NIO**，代理将使用 Java NIO 接口到 itsjournal。设置为 **ASYNCIO**，代理将使用 Linux 异步 IO 日志。如果您选择 **ASYNCIO** 但没有运行 Linux，或者没有安装 libaio，那么代理将检测到这一点，并自动回退到使用 **NIO**。
journal-sync-transactional	如果设置为 **true**，代理会将所有事务数据刷新到事务边界上的磁盘（即提交、准备和回滚）。默认值为 **true**。
journal-sync-non-transactional	如果设置为 **true**，代理会每次清除非事务消息数据（发送和确认）到磁盘。默认值为 **true**。
journal-file-size	每个日志文件的大小（以字节为单位）。默认值为 **10485760** 字节(10MiB)。
journal-min-files	代理在启动时预先创建的最小文件数。只有在没有现有消息数据时，才会预先创建文件。 根据您期望的队列包含处于 steady 状态的数据量，您应该调整这个数量的文件以匹配预期的数据总量。
journal-pool-files	系统将根据需要创建任意数量的文件；但是，当回收文件时，它将缩小到 **journal-pool-files**。 默认值为 **-1**， 表示它永远不会在日志创建后删除文件。但是，系统无法无限增长，因为您仍需要为可以无限期增长的目的地使用分页。

名称	描述
journal-max-io	控制任意一个时间可在 IO 队列中的最大写入请求数。如果队列已满，则写入将阻止，直到释放空间为止。 使用 NIO 时，这个值应始终为 **1**。使用 AIO 时，默认值为 **500**。最大 AIO 总数不能超过操作系统级别设置的值(**/proc/sys/fs/aio-max-nr**)，其通常为 65536。
journal-buffer-timeout	控制何时清空缓冲区的超时时间。AIO 通常可以有高于 NIO 的 flush 率，因此系统为 NIO 和 AIO 维护不同的默认值。 NIO 的默认值为 **3333333** 纳秒或每秒 300 次，AIO 的默认值是 **50000** 纳秒，每秒 2000 次。  **注意** 通过增加超时值，您可以以延迟费用增加系统吞吐量，因为选择默认值在吞吐量和延迟之间提供合理的平衡。
journal-buffer-size	AIO 中 timed 缓冲的大小。默认值为 **490KiB**。
journal-compact-min-files	代理压缩日志前所需的最少文件数。紧凑算法将不会启动，直到您至少有 **journal-compact-min-files**。默认值为 **10**。  **注意** 将值设为 **0** 将禁用紧凑，并可能危险，因为日志可能会无限期地增长。
journal-compact-percentage	开始压缩的阈值。如果日志数据小于 **journal-compact-percentage**，则日志数据将压缩为实时数据。另请注意，压缩将不会启动，直到您在日志中至少有 **journal-compact-min-files** 数据文件。默认值为 **30**。

附录 F. 其他复制高可用性配置元素

下表列出了 配置复制高可用性 部分未描述的额外 ha-policy 配置元素。这些元素具有默认设置，适用于大多数常见用例。

表 F.1. 使用复制高可用性时可用的其他配置元素

名称	使用于	描述
check-for-live-server	嵌入式代理协调	只适用于配置为 master 代理的代理。指定原始 master 代理是否在启动时使用自己的服务器 ID 检查另一个 live 代理。设置为 **true** 以切换到原始 master 代理，并避免两个代理同时存在的"脑裂"情况。此属性的默认值为 **false**。
cluster-name	嵌入式代理和 ZooKeeper 协调	用于复制的集群配置的名称。只有在配置多个集群连接时，才需要此设置。如果配置，则连接集群时将使用具有此名称的集群配置。如果未设置，则使用配置中定义的第一个集群连接。
initial-replication-sync-timeout	嵌入式代理和 ZooKeeper 协调	复制代理在完成副本的初始复制过程后等待的时间，以确认它已收到所有必要的数据。此属性的默认值为 **30,000** 毫秒。 注意：在这个间隔中，任何其他与日志相关的操作都会被阻止。
max-saved-replicated-journals-size	嵌入式代理和 ZooKeeper 协调	只适用于备份代理。指定备份代理保留多少个备份日志文件。达到这个值后，代理通过删除最旧的日志文件来为每个新的备份日志文件腾出空间。此属性的默认值为 **2**。

附录 G. 代理属性

以下是 AMQ Broker 属性列表，可直接应用到内部 java 配置 bean，而不使用 XML 配置。

criticalAnalyzerCheckPeriod

Type: long
Default: 0
XML name: critical-analyzer-check-period
Description: The period 默认为 half the critical-analyzer-timeout，并在运行时计算。

pageMaxConcurrentIO

Type: int
Default: 5
XML name: page-max-concurrent-io
Description: 在分页过程中允许的最大并发读取数。

messageCounterSamplePeriod

Type: long
Default: 10000
XML name: message-counter-sample-period
Description: sample period (in ms)用于消息计数器。

networkCheckNIC

Description:

globalMaxSize

Type: long
Default: -1
XML name: global-max-size
Description: Size (in bytes) before all addresses will enter into their Full Policy configured upon messages being produced.支持字节表示法，如 "K"、"Mb"、"MiB"、"GB"等。

journalFileSize

Type: int
Default: 10485760
XML name: journal-file-size
Description: 每个日志文件的大小（以字节为单位）。支持字节表示法，例如："K"、"Mb"、"MiB"、"GB"。

configurationFileRefreshPeriod

Type: long
Default: 5000
XML name: configuration-file-refresh-period
Description:

diskScanPeriod

Type: int
Default: 5000
XML name: disk-scan-period
Description:

journalRetentionDirectory

Description:

networkCheckPeriod

Type: long
Default: 10000
XML name: network-check-period
Description:

journalBufferSize_AIO

Type: int
Default: 501760
XML name: journal-buffer-size
Description:支持字节表示法，如 "K", "Mb", "MiB", "GB"。

networkCheckURLList

Description:

networkCheckTimeout

Type: int
Default: 1000
XML name: network-check-timeout
Description: timeout （毫秒）用于 ping。

pageSyncTimeout

Type: int
Default:
XML name: page-sync-timeout
Description: timeout, in nanoseconds, 用于同步页面。确切的默认值取决于日志是 ASYNCIO 或 NIO。

journalPoolFiles

Type: int
Default: -1
XML name: journal-pool-files
Description: The number of journal files to pre-create.

criticalAnalyzer

Type: boolean
Default: true
XML name: critical-analyzer
Description: Should analyze response time on critical 路径，并决定代理日志、关闭或停止。

readWholePage

Type: boolean
Default: false
XML name: read-whole-page
Description: 指定在页面缓存被驱除后是否读取整个页面。

maxDiskUsage

Type: int
Default: 90
XML name: max-disk-usage
Description: system blocks before before system blocks or failed client.

globalMaxMessages

Type: long
Default: -1
XML name: global-max-messages
Description: Number of messages before all addresses will enter into their address full policy configured.它与 global-max-size 结合使用，并在达到任一限制时执行配置的地址完全策略。

internalNamingPrefix

Description:

这些队列和地址默认前缀为 "$.activemq.internal"，以避免以用户名pacing 命名。这可以通过将此值设置为有效的 Artemis 地址来覆盖。

journalFileOpenTimeout

Type: int
Default: 5
XML name: journal-file-open-timeout
描述：打开新 Journal 文件的时间（以秒为单位）再超时和失败。

journalCompactPercentage

Type: int
Default: 30
XML name: journal-compact-percentage
Description: 用于考虑压缩日志的实时数据的百分比。

createBindingsDir

Type: boolean
Default: true

XML name: create-bindings-dir
Description: true 的值使服务器在启动时创建绑定目录。

suppressSessionNotifications

Type: boolean
Default: false
XML name: suppress-session-notifications
Description: 是否禁止 SESSION_CREATED 和 SESSION_CLOSED 通知。设置为 true 以减少通知开销。但是，对于 MQTT 客户端，需要在集群中强制使用唯一的客户端 ID。

journalBufferTimeout_AIO

Type: int
Default:
XML name: journal-buffer-timeout
Description: timeout,in nanoseconds, 用于清除日志上的内部缓冲区。确切的默认值取决于日志是 ASYNCIO 或 NIO。

journalType

type: JournalType
Default: ASYNCIO
XML name: journal-type
Description: 要使用的日志类型。

名称

Type:

如果设置，它将在拓扑通知中使用。

networkCheckPingCommand

Type: String
Default:
XML name: network-check-ping-command
Description: The ping command used to ping IPV4 addresses.

temporaryQueueNamespace

Type:

pagingDirectory

Description:

journalDirectory

Type:

journalBufferSize_NIO

Type: int
Default: 501760
XML name: journal-buffer-size
Description:支持字节表示法，例如："K"、"Mb"、"MiB"、"GB"。

journalDeviceBlockSize

type: Integer
Default:
XML name: journal-device-block-size
描述：设备使用的大小（以字节为单位）。这通常转换为 fstat/st_blksize，这是一种绕过作为 st_blksize 返回的值的方法。

nodeManagerLockDirectory

Description:

messageCounterMaxDayHistory

Type: int
Default: 10
XML name: message-counter-max-day-history
Description: 保存消息计数器历史记录的天数。

largeMessagesDirectory

Description:

networkCheckPing6Command

Type: String
Default:

XML name: network-check-ping6-command
Description: The ping command used to ping IPV6 addresses.

memoryWarningThreshold

Type: int
Default: 25
XML name: memory-warning-threshold
Description: Percentage of available memory is generated.

mqttSessionScanInterval

Type: long
Default: 5000
XML name: mqtt-session-scan-interval
Description: The frequency, in milliseconds, to scan for expired MQTT 会话。

journalMaxAtticFiles

Type: int
Default:
XML name: journal-max-attic-files
Description:

journalSyncTransactional

Type: boolean
Default: true
XML name: journal-sync-transactional
Description: 如果设为 true，请在返回对客户端的响应前等待事务数据同步到日志。

logJournalWriteRate

Type: boolean
Default: false
XML name: log-journal-write-rate
Description: 指定是否记录与日志 write-rate 相关的消息。

journalMaxIO_AIO

Type: int
Default:
XML name: journal-max-io
Description: 任意一个时间可在 AIO 队列中的最大写入请求数。AIO 为 500，对于 NIO，默认为 1。

messageExpiryScanPeriod

Type: long
Default: 30000
XML name: message-expiry-scan-period
Description:

criticalAnalyzerTimeout

Type: long
Default: 120000
XML name: critical-analyzer-timeout
Description: 用于分析关键路径上的超时的默认超时时间。

messageCounterEnabled

type: boolean
Default: false
XML name: message-counter-enabled
Description: true 值表示启用了消息计数器。

journalCompactMinFiles

Type: int
Default: 10
XML name: journal-compact-min-files
Description: 代理开始到紧凑文件前的最小数据文件数。

createJournalDir

Type: boolean
Default: true
XML name: create-journal-dir
Description: 一个 true 值表示创建了 journal 目录。

addressQueueScanPeriod

Type: long
Default: 30000
XML name: address-queue-scan-period
Description: 频率（以毫秒为单位），用于扫描需要删除的地址和队列。

memoryMeasureInterval

Type: long
Default: -1
XML name: memory-measure-interval
Description: The frequency, in milliseconds, to sample JVM memory.值 -1 禁用内存抽样。

journalSyncNonTransactional

Type: boolean
Default: true
XML name: journal-sync-non-transactional
Description: if true，请在返回对客户端的响应前等待非事务数据同步到日志。

connectionTtlCheckInterval

Type: long
Default: 2000
XML name: connection-ttl-check-interval
Description:

rejectEmptyValidatedUser

Type: boolean
Default: false
XML name: reject-empty-validated-user
Description: if true，服务器不允许任何没有验证的用户的消息。在 JMS 中，这是 JMSXUserID。

journalMaxIO_NIO

Type: int
Default:
XML name: journal-max-io
Description: 任意一个时间可在 AIO 队列中的最大写入请求数。AIO 为 500，对于 NIO，默认为 1。目前，代理属性只支持使用整数和测量（以字节为单位）。

transactionTimeoutScanPeriod

Type: long
Default: 1000
XML name: transaction-timeout-scan-period
Description: 频率（以毫秒为单位）来扫描超时事务。

systemPropertyPrefix

Type: String
Default:

XML name: system-property-prefix
Description: The prefix used to parse system properties for the configuration.

transactionTimeout

Type: long
Default: 300000
XML name: transaction-timeout
Description: 持续时间（以毫秒为单位），在创建时间后可以从资源管理器中删除事务。

journalLockAcquisitionTimeout

Type: long
Default: -1
XML name: journal-lock-acquisition-timeout
Description: The frequency, in milliseconds, to wait to acquire a file lock on the journal.

journalBufferTimeout_NIO

type: int
Default:
XML name: journal-buffer-timeout
Description: timeout （以纳秒为单位）用于清除日志上的内部缓冲区。确切的默认值取决于日志是 ASYNCIO 或 NIO。

journalMinFiles

Type: int
Default: 2
XML name: journal-min-files
Description: to pre-create 的日志文件数。

G.1. BRIDGECONFIGURATIONS

bridgeConfigurations.<name>.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
描述：应用到连续重试间隔的倍数。

bridgeConfigurations.<name>.maxRetryInterval

Type: long
Default: 2000
XML name: max-retry-interval
Description: limit to the retry-interval growth due due-interval-multiplier。

bridgeConfigurations.<name>.filterString

Description:

bridgeConfigurations.<name>.connectionTTL

Type: long
Default: 60000
XML name: connection-ttl
Description: 在没有从客户端接收数据时保留连接的时间。持续时间应大于 ping 周期。

bridgeConfigurations.<name>.confirmationWindowSize

Type: int
Default: 1048576

XML name: confirmation-window-size
Description: 网桥发送确认后的字节数。支持字节表示法，如 "K", "Mb", "MiB", "GB"。

bridgeConfigurations.<name>.staticConnectors

Type: List
Default:
XML name: static-connectors
Description:

bridgeConfigurations.<name>.reconnectAttemptsOnSameNode

Type: int
Default:
XML name: reconnect-attempts-on-same-node
Description:

bridgeConfigurations.<name>.concurrency

Type: int
Default: 1
XML name: concurrency
Description: concurrent worker 的数量。更多 worker 可以帮助提高高延迟网络上的吞吐量。默认值为 1。

bridgeConfigurations.<name>.transformerConfiguration

type: TransformerConfiguration
Default:
XML name: transformer-configuration
Description:

bridgeConfigurations.<name>.transformerConfiguration.className

Type:

bridgeConfigurations.<name>.transformerConfiguration.properties

Type: Map
Default:
XML name: property
Description: a KEY/VALUE 对在 transformer 上设置，例如 properties.MY_PROPERTY=MY_VALUE

bridgeConfigurations.<name>.password

Type:

bridgeConfigurations.<name>.queueName

Type:

bridgeConfigurations.<name>.forwardingAddress

Description:

如果省略，则使用原始地址。

bridgeConfigurations.<name>.routingType

Type: componentConfigurationRoutingType
Default: PASS
XML name: routing-type
Description: 如何设置网桥消息上的 route-type。

bridgeConfigurations.<name>.name

Type:

bridgeConfigurations.<name>.ha

Type: boolean
Default: false
XML name: ha
Description: 指定此网桥是否支持故障切换。

bridgeConfigurations.<name>.initialConnectAttempts

Type: int
Default: -1
XML name: initial-connect-attempts
Description: The maximum number of initial connection attempts.默认值 -1 表示没有限制。

bridgeConfigurations.<name>.retryInterval

Type: long
Default: 2000

XML name: retry-interval
Description: interval, in milliseconds, between successive retries。

bridgeConfigurations.<name>.producerWindowSize

Type: int
Default: 1048576
XML name: producer-window-size
Description: Producer flow control.支持字节表示法，例如："K"、"Mb"、"MiB"、"GB"。

bridgeConfigurations.<name>.clientFailureCheckPeriod

Type: long
Default: 30000
XML name: check-period
Description: interval, in milliseconds,, 一个网桥的客户端检查它是否无法从服务器接收 ping。指定 -1 来禁用此检查。

bridgeConfigurations.<name>.discoveryGroupName

Description:

bridgeConfigurations.<name>.user

Description:

如果未指定，则使用 cluster-user。

bridgeConfigurations.<name>.useDuplicateDetection

Type: boolean
Default: true
XML name: use-duplicate-detection
Description: 指定在转发消息中是否插入重复检测标头。

bridgeConfigurations.<name>.minLargeMessageSize

Type: int
Default: 102400
XML name: min-large-message-size
Description: The size, in bytes, which is a large message.大型消息在多个网段通过网络发送。支持字节表示法，如 "K", "Mb", "MiB", "GB"。

G.2. AMQPCONNECTIONS

AMQPConnections.<name>.reconnectAttempts

Type: int
Default: -1
XML name: reconnect-attempts
Description: The number of reconnection attempts after a failure.

AMQPConnections.<name>.password

Type:

如果没有指定，会尝试匿名连接。

AMQPConnections.<name>.retryInterval

Type: int
Default: 5000

XML name: retry-interval
Description: interval, in milliseconds, between successive retries。

AMQPConnections.<name>.connectionElements

Type: AMQPMirrorBrokerConnectionElement
Default:
XML name: amqp-connection
Description: AMQP Broker Connection support 4 type: 1.mirror - 代理使用 AMQP 连接到另一个代理，并复制消息，并通过线路发送确认。2.senders - 在特定队列上接收的消息传输到另一个端点。3.receivers - 代理从另一个端点拉取信息。4.peers - 代理在知道如何处理它们的另一个端点上创建发送者和接收器。目前由 Apache Qpid Dispatch 实施。目前，只支持镜像类型。

AMQPConnections.<name>.connectionElements.<name>.messageAcknowledgments

Type: boolean
Default:
XML name: message-acknowledgments
Description: If true, message confirmations are mirrored。

AMQPConnections.<name>.connectionElements.<name>.queueRemoval

Type: boolean
Default:
XML name: queue-removal
Description: 指定镜像队列是否删除地址和队列的事件。

AMQPConnections.<name>.connectionElements.<name>.addressFilter

Description:

AMQPConnections.<name>.connectionElements.<name>.queueCreation

Type: boolean
Default:
XML name: queue-creation
Description: 指定镜像队列是否为地址和队列创建事件。

AMQPConnections.<name>.autostart

Type: boolean
Default: true
XML name: auto-start
Description: 指定服务器启动时是否启动代理连接。

AMQPConnections.<name>.user

Type:

如果没有指定，会尝试匿名连接。

AMQPConnections.<name>.uri

Type:

G.3. DIVERTCONFIGURATION

divertConfiguration.transformerConfiguration

type: TransformerConfiguration
Default:

XML name: transformer-configuration
Description:

divertConfiguration.transformerConfiguration.className

Type:

divertConfiguration.transformerConfiguration.properties

Type: Map
Default:
XML name: property
Description: a KEY/VALUE 对在 transformer 上设置，例如 … properties.MY_PROPERTY=MY_VALUE。

divertConfiguration.filterString

Description:

divertConfiguration.routingName

Description:

divertConfiguration.address

Type: String
Default:
XML name: address
Description: The address this divert will divert from.

divertConfiguration.forwardingAddress

Description:

divertConfiguration.routingType

Type: componentConfigurationRoutingType (MULTICAST ANYCAST STRIP PASS)
Default:
XML name: routing-type
Description: How the routing-type on the diverted 消息。

divertConfiguration.exclusive

type: boolean
Default: false
XML name: exclusive
Description: 指定这是专用 divert。

G.4. ADDRESSSETTINGS

addressSettings.<address>.configDeleteDiverts

Type: DeletionPolicy (OFF FORCE)
Default:
XML name: config-delete-addresses
Description:

addressSettings.<address>.expiryQueuePrefix

Type: SimpleString
Default:
XML name: expiry-queue-prefix
Description:

addressSettings.<address>.defaultConsumerWindowSize

Type: int
Default:
XML name: default-consumer-window-size
Description:

addressSettings.<address>.maxReadPageBytes

Type: int
Default:
XML name: max-read-page-bytes
Description:

addressSettings.<address>.deadLetterQueuePrefix

Type: SimpleString
Default:
XML name: dead-letter-queue-prefix
Description:

addressSettings.<address>.defaultGroupRebalancePauseDispatch

Type: boolean
Default:

XML name: default-group-rebalance-pause-dispatch
Description:

addressSettings.<address>.autoCreateAddresses

type: boolean
Default:
XML name: auto-create-addresses
Description:

addressSettings.<address>.slowConsumerThreshold

type: long
Default:
XML name: slow-consumer-threshold
Description:

addressSettings.<address>.managementMessageAttributeSizeLimit

Type: int
Default:
XML name: management-message-attribute-size-limit
Description:

addressSettings.<address>.autoCreateExpiryResources

type: boolean
Default:
XML name: auto-create-expiry-resources
Description:

addressSettings.<address>.pageSizeBytes

Type: int
Default:
XML name: page-size-bytes
Description:

addressSettings.<address>.minExpiryDelay

Type: Long
Default:
XML name: min-expiry-delay
Description:

addressSettings.<address>.defaultConsumersBeforeDispatch

type: Integer
Default:
XML name: default-consumers-before-dispatch
Description:

addressSettings.<address>.expiryQueueSuffix

Type: SimpleString
Default:
XML name: expiry-queue-suffix
Description:

addressSettings.<address>.configDeleteQueues

Type: DeletionPolicy (OFF FORCE)
Default:
XML name: config-delete-queues
Description:

addressSettings.<address>.enableIngressTimestamp

type: boolean
Default:
XML name: enable-ingress-timestamp
Description:

addressSettings.<address>.autoDeleteCreatedQueues

type: boolean
Default:
XML name: auto-delete-created-queues
Description:

addressSettings.<address>.expiryAddress

Type: SimpleString
Default:
XML name: expiry-address
Description:

addressSettings.<address>.managementBrowsePageSize

Type: int
Default:
XML name: management-browse-page-size
Description:

addressSettings.<address>.autoDeleteQueues

type: boolean
Default:
XML name: auto-delete-queues
Description:

addressSettings.<address>.retroactiveMessageCount

Type: long
Default:
XML name: retroactive-message-count
Description:

addressSettings.<address>.maxExpiryDelay

Type: Long
Default:
XML name: max-expiry-delay
Description:

addressSettings.<address>.maxDeliveryAttempts

type: int
Default:
XML name: max-delivery-attempts
Description:

addressSettings.<address>.defaultGroupFirstKey

Type: SimpleString
Default:
XML name: default-group-first-key
Description:

addressSettings.<address>.slowConsumerCheckPeriod

type: long
Default:

XML name: slow-consumer-check-period
Description:

addressSettings.<address>.defaultPurgeOnNoConsumers

type: boolean
Default:
XML name: default-purge-on-no-consumers
Description:

addressSettings.<address>.defaultLastValueKey

Type: SimpleString
Default:
XML name: default-last-value-key
Description:

addressSettings.<address>.autoCreateQueues

type: boolean
Default:
XML name: auto-create-queues
Description:

addressSettings.<address>.defaultExclusiveQueue

Type: boolean
Default:
XML name: default-exclusive-queue
Description:

addressSettings.<address>.defaultMaxConsumers

type: Integer
Default:
XML name: default-max-consumers
Description:

addressSettings.<address>.defaultQueueRoutingType

Type: RoutingType (MULTICAST ANYCAST)
Default:
XML name: default-queue-routing-type
Description:

addressSettings.<address>.messageCounterHistoryDayLimit

Type: int
Default:
XML name: message-counter-history-day-limit
Description:

addressSettings.<address>.defaultGroupRebalance

Type: boolean
Default:
XML name: default-group-rebalance
Description:

addressSettings.<address>.defaultAddressRoutingType

Type: RoutingType (MULTICAST ANYCAST)
Default:
XML name: default-address-routing-type
Description:

addressSettings.<address>.maxSizeBytesRejectThreshold

Type: long
Default:
XML name: max-size-bytes-reject-threshold
Description:

addressSettings.<address>.pageCacheMaxSize

type: int
Default:
XML name: page-cache-max-size
Description:

addressSettings.<address>.autoCreateDeadLetterResources

type: boolean
Default:
XML name: auto-create-dead-letter-resources
Description:

addressSettings.<address>.maxRedeliveryDelay

type: long
Default:
XML name: max-redelivery-delay
Description:

addressSettings.<address>.configDeleteAddresses

type: DeletionPolicy
Default:
XML name: config-delete-addresses
Description:

addressSettings.<address>.deadLetterAddress

Type: SimpleString
Default:
XML name: dead-letter-address
Description:

addressSettings.<address>.autoDeleteQueuesMessageCount

Type: long
Default:
XML name: auto-delete-queues-message-count
Description:

addressSettings.<address>.autoDeleteAddresses

type: boolean
Default:
XML name: auto-delete-addresses
Description:

addressSettings.<address>.addressFullMessagePolicy

type: AddressFullMessagePolicy
Default:
XML name: address-full-message-policy
Description:

addressSettings.<address>.maxSizeBytes

Type: long
Default:

XML name: max-size-bytes
Description:

addressSettings.<address>.defaultDelayBeforeDispatch

Type: Long
Default:
XML name: default-delay-before-dispatch
Description:

addressSettings.<address>.redistributionDelay

Type: long
Default:
XML name: redistribution-delay
Description:

addressSettings.<address>.maxSizeMessages

Type: long
Default:
XML name: max-size-messages
Description:

addressSettings.<address>.redeliveryMultiplier

type: double
Default:
XML name: redelivery-multiplier
Description:

addressSettings.<address>.defaultRingSize

Type: long
Default:
XML name: default-ring-size
Description:

addressSettings.<address>.defaultLastValueQueue

type: boolean
Default:
XML name: default-last-value-queue
Description:

addressSettings.<address>.slowConsumerPolicy

Type: SlowConsumerPolicy (KILL NOTIFY)
Default:
XML name: slow-consumer-policy
Description:

addressSettings.<address>.redeliveryCollisionAvoidanceFactor

type: double
Default:
XML name: redelivery-collision-avoidance-factor
Description:

addressSettings.<address>.autoDeleteQueuesDelay

type: long
Default:
XML name: auto-delete-queues-delay
Description:

addressSettings.<address>.autoDeleteAddressesDelay

Type: long
Default:
XML name: auto-delete-addresses-delay
Description:

addressSettings.<address>.expiryDelay

Type: Long
Default:
XML name: expiry-delay
Description:

addressSettings.<address>.enableMetrics

Type: boolean
Default:
XML name: enable-metrics
Description:

addressSettings.<address>.sendToDLAOnNoRoute

Type: boolean
Default:
XML name: send-to-d-l-a-on-no-route
Description:

addressSettings.<address>.slowConsumerThresholdMeasurementUnit

Type: SlowConsumerThresholdMeasurementUnit (MESSAGES_PER_SECOND MESSAGES_PER_MINUTE MESSAGES_PER_HOUR MESSAGES_PER_DAY)
Default:

XML name: slow-consumer-threshold-measurement-unit
Description:

addressSettings.<address>.redeliveryDelay

type: long
Default:
XML name: redelivery-delay
Description:

addressSettings.<address>.deadLetterQueueSuffix

Type: SimpleString
Default:
XML name: dead-letter-queue-suffix
Description:

addressSettings.<address>.defaultNonDestructive

Type: boolean
Default:
XML name: default-non-destructive
Description:

G.5. FEDERATIONCONFIGURATIONS

federationConfigurations.<name>.transformerConfigurations

type: FederationTransformerConfiguration
Default:
XML name: transformer
Description: Optional transformer configuration.

federationConfigurations.<name>.transformerConfigurations.

<name>.transformerConfigurationConfiguration

type: TransformerConfiguration
Default:
XML name: transformer
Description: 允许添加自定义转换器来修改消息。

federationConfigurations.<name>.transformerConfigurations.
<name>.transformerConfiguration.<name>.className

Type:

federationConfigurations.<name>.transformerConfigurations.
<name>.transformerConfiguration.<name>.properties

Type: Map
Default:
XML name: property
Description: a KEY/VALUE 对在 transformer 上设置，例如 …
properties.MY_PROPERTY=MY_VALUE。

federationConfigurations.<name>.queuePolicies

type: FederationQueuePolicyConfiguration
Default:
XML name: queue-policy
Description:

federationConfigurations.<name>.queuePolicies.<name>.priorityAdjustment

Type: Integer

Default:
XML name: priority-adjustment
Description: when a consumer attach，其优先级用于创建上游消费者，但经过调整，本地消费者在远程消费者之前进行负载均衡。

federationConfigurations.<name>.queuePolicies.<name>.excludes

Type: Matcher
Default:
XML name: exclude
Description: A list of queue match to exclude。

federationConfigurations.<name>.queuePolicies.<name>.excludes.<name>.queueMatch

Type:

如果没有，则所有队列都匹配。

federationConfigurations.<name>.queuePolicies.<name>.transformerRef

Type:

federationConfigurations.<name>.queuePolicies.<name>.includes

Type: Matcher
Default:
XML name: queue-match
Description:

federationConfigurations.<name>.queuePolicies.<name>.excludes.<name>.queueMatch

Type:

如果没有，则所有队列都匹配。

federationConfigurations.<name>.queuePolicies.<name>.includeFederated

Type: boolean
Default:
XML name: include-federated
Description: 当值设为 false 时，配置不会重新联邦一个已联邦消费者，即联邦队列中的消费者。这可避免在对称或关闭的拓扑中，没有非联邦消费者和系统的消息流。

federationConfigurations.<name>.upstreamConfigurations

Type: FederationUpstreamConfiguration
Default:
XML name: upstream
Description:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration

type: FederationConnectionConfiguration
Default:
XML name: connection-configuration
Description: streams connection configuration。

federationConfigurations.<name>.upstreamConfigurations.
<name>.connectionConfiguration.priorityAdjustment

Type: int
Default:

XML name: priority-adjustment
Description:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
Description: multiplier to the retry-interval。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.shareConnection

Type: boolean
Default: false
XML name: share-connection
Description: 如果设为 true，如果为同一代理配置了一个下游和上游连接，则会共享同一连接。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.maxRetryInterval

Type: long
Default: 2000
XML name: max-retry-interval
Description: retry-interval 的最大值。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.connectionTTL

Type: long
Default:
XML name: connection-t-t-l
Description:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.circuitBreakerTimeout

type: long
Default: 30000
XML name: circuit-breaker-timeout
Description: 指定此连接是否支持故障切换。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.callTimeout

Type: long
Default: 30000
XML name: call-timeout
Description: 等待一个回复的时间。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.staticConnectors

Type: List
Default:
XML name: static-connectors
Description: 通过连接器配置的连接器引用列表。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.reconnectAttempts

Type: int
Default: -1
XML name: reconnect-attempts
Description: The number of reconnection attempts after a failure.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.password

Type:

如果没有指定，则使用联邦密码。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.callFailoverTimeout

Type: long
Default: -1
XML name: call-failover-timeout
Description: The duration to wait for a reply during a fail-over.值 -1 表示没有限制。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.hA

Type: boolean
Default:
XML name: h-a
Description:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.initialConnectAttempts

Type: int
Default: -1
XML name: initial-connect-attempts
Description: The number of initial attempts to connect.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.retryInterval

Type: long
Default: 500

XML name: retry-interval
Description: interval, in milliseconds, between successive retries。

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.clientFailureCheckPeriod

Type: long
Default:
XML name: client-failure-check-period
Description:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.username

Type:

federationConfigurations.<name>.upstreamConfigurations.<name>.policyRefs

Type: Collection
Default:
XML name: policy-refs
Description:

federationConfigurations.<name>.upstreamConfigurations.<name>.staticConnectors

Type: List
Default:
XML name: static-connectors
Description: 通过连接器配置的连接器引用列表。

federationConfigurations.<name>.downstreamConfigurations

type: FederationDownstreamConfiguration
Default:
XML name: downstream
Description:

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration

type: FederationConnectionConfiguration
Default:
XML name: connection-configuration
Description: streams connection configuration。

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration.priorityAdjustment

Type: int
Default:
XML name: priority-adjustment
Description:

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
Description: multiplier to the retry-interval。

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration.shareConnection

Type: boolean
Default: false

XML name: share-connection
Description: 如果设为 true，如果为同一代理配置了一个下游和上游连接，则会共享同一连接。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.maxRetryInterval

Type: long
Default: 2000
XML name: max-retry-interval
Description: retry-interval 的最大值。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.connectionTTL

Type: long
Default:
XML name: connection-t-t-l
Description:

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.circuitBreakerTimeout

type: long
Default: 30000
XML name: circuit-breaker-timeout
Description: 指定此连接是否支持故障切换。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.callTimeout

Type: long
Default: 30000
XML name: call-timeout
Description: 等待一个回复的时间。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.staticConnectors

Type: List
Default:
XML name: static-connectors
Description: 通过连接器配置的连接器引用列表。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.reconnectAttempts

Type: int
Default: -1
XML name: reconnect-attempts
Description: The number of reconnection attempts after a failure.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.password

Type:

如果没有指定，则使用联邦密码。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.callFailoverTimeout

Type: long
Default: -1
XML name: call-failover-timeout
Description: The duration to wait for a reply during a fail-over.值 -1 表示没有限制。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.hA

Type: boolean
Default:
XML name: h-a
Description:

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.initialConnectAttempts

Type: int
Default: -1
XML name: initial-connect-attempts
Description: The number of initial attempts to connect.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.retryInterval

Type: long
Default: 500
XML name: retry-interval
Description: 周期，以毫秒为单位（以毫秒为单位）。

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.clientFailureCheckPeriod

Type: long
Default:
XML name: client-failure-check-period
Description:

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.username

Type:

federationConfigurations.<name>.downstreamConfigurations.<name>.policyRefs

Type: Collection
Default:
XML name: policy-refs
Description:

federationConfigurations.<name>.downstreamConfigurations.<name>.staticConnectors

Type: List
Default:
XML name: static-connectors
Description: 通过连接器配置的连接器引用列表。

federationConfigurations.<name>.federationPolicys

Type: FederationPolicy
Default:
XML name: policy-set
Description:

federationConfigurations.<name>.addressPolicies

type: FederationAddressPolicyConfiguration
Default:
XML name: address-policy
Description:

federationConfigurations.<name>.addressPolicies.<name>.autoDeleteMessageCount

Type: Long
Default:
XML name: auto-delete-message-count
Description: 在该队列被自动删除前，仍可在动态创建的远程队列中的最大消息数。

federationConfigurations.<name>.addressPolicies.<name>.enableDivertBindings

type: boolean
Default:
XML name: enable-divert-bindings
Description: Setting to true enable divert bindings to be listened-to for demand。如果存在与地址策略包含的地址匹配的解析绑定，则任何与 divert 的转发地址匹配的队列绑定都会创建需求。默认值为 false。

federationConfigurations.<name>.addressPolicies.<name>.includes.{NAME}.addressMatch

Type: Matcher
Default:
XML name: include
Description:

federationConfigurations.<name>.addressPolicies.<name>.maxHops

Type: int
Default:
XML name: max-hops
Description: 消息可以进行联邦的跃点数。

federationConfigurations.<name>.addressPolicies.<name>.transformerRef

Type:

federationConfigurations.<name>.addressPolicies.<name>.autoDeleteDelay

Type: Long
Default:
XML name: auto-delete-delay
Description: 持续时间（以毫秒为单位），在下游代理断开连接后，自动删除上游队列。

federationConfigurations.<name>.addressPolicies.<name>.autoDelete

Type: Boolean
Default:
XML name: auto-delete
Description: 对于地址联邦，下游会在上游地址上动态创建持久队列。指定在下游断开连接后是否删除上游队列，以及满足延迟和消息计数参数。

federationConfigurations.<name>.addressPolicies.<name>.excludes.{NAME}.addressMatch

Type: Matcher
Default:
XML name: include
Description:

G.6. CLUSTERCONFIGURATIONS

clusterConfigurations.<name>.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
Description: multiplier to the retry-interval。

clusterConfigurations.<name>.maxRetryInterval

Type: long
Default: 2000
XML name: max-retry-interval
Description: retry-interval 的最大值。

clusterConfigurations.<name>.address

Type:

clusterConfigurations.<name>.maxHops

Type: int
Default: 1
XML name: max-hops
Description: 集群拓扑要传播到的最大跃点数。

clusterConfigurations.<name>.connectionTTL

Type: long
Default: 60000
XML name: connection-ttl
Description: 在没有从客户端接收数据时保留连接的时间。

clusterConfigurations.<name>.clusterNotificationInterval

Type: long
Default: 1000
XML name: notification-interval
Description: 集群连接通知其存在的时间间隔。

clusterConfigurations.<name>.confirmationWindowSize

Type: int
Default: 1048576
XML name: confirmation-window-size
描述：用于确认服务器数据的窗口的大小（以字节为单位）。支持字节表示法，如 "K", "Mb", "MiB", "GB"。

clusterConfigurations.<name>.callTimeout

Type: long
Default: 30000
XML name: call-timeout
Description: 等待一个回复的时间。

clusterConfigurations.<name>.staticConnectors

Type: List
Default:
XML name: static-connectors
Description: 连接器引用名称列表。

clusterConfigurations.<name>.clusterNotificationAttempts

Type: int
Default: 2
XML name: notification-attempts
Description: 此集群连接尝试通知其存在集群的次数。

clusterConfigurations.<name>.allowDirectConnectionsOnly

Type: boolean
Default: false

XML name: allow-direct-connections-only
Description: Restricts cluster connections to the listed connector-refs。

clusterConfigurations.<name>.reconnectAttempts

Type: int
Default: -1
XML name: reconnect-attempts
Description: The number of reconnection attempts after a failure.

clusterConfigurations.<name>.duplicateDetection

Type: boolean
Default: true
XML name: use-duplicate-detection
Description: 指定是否应在转发消息中插入重复检测标头。

clusterConfigurations.<name>.callFailoverTimeout

Type: long
Default: -1
XML name: call-failover-timeout
Description: The duration to wait for a reply during a fail-over.值 -1 表示没有限制。

clusterConfigurations.<name>.messageLoadBalancingType

Type: MessageLoadBalancingType (OFF STRICT ON_DEMAND OFF_WITH_REDISTRIBUTION)
Default:
XML name: message-load-balancing-type
Description:

clusterConfigurations.<name>.initialConnectAttempts

Type: int
Default: -1
XML name: initial-connect-attempts
Description: The number of initial attempts to connect.

clusterConfigurations.<name>.connectorName

Type:

clusterConfigurations.<name>.retryInterval

Type: long
Default: 500
XML name: retry-interval
Description: interval, in milliseconds, between successive retries。

clusterConfigurations.<name>.producerWindowSize

Type: int
Default: 1048576
XML name: producer-window-size
Description: Producer flow control.支持字节表示法，如 "K", "Mb", "MiB", "GB"。

clusterConfigurations.<name>.clientFailureCheckPeriod

Type: long
Default:
XML name: client-failure-check-period
Description:

clusterConfigurations.<name>.discoveryGroupName

Description:

clusterConfigurations.<name>.minLargeMessageSize

Type: int
Default:
XML name: min-large-message-size
Description: size, in bytes, which is a large message.大型消息在多个网段通过网络发送。支持字节表示法，如 "K", "Mb", "MiB", "GB"。

G.7. CONNECTIONROUTERS

connectionRouters.<name>.cacheConfiguration

Type: CacheConfiguration
Default:
XML name: cache
Description: Controls if the cache entries are persist and a cache removed its entries.

connectionRouters.<name>.cacheConfiguration.persisted

type: boolean
Default: false
XML name: persist
Description: true 值表示缓存条目是持久的。

connectionRouters.<name>.cacheConfiguration.timeout

Type: int

Default: -1
XML name: timeout
Description: The timeout, in milliseconds, before cache entries are removed.

connectionRouters.<name>.keyFilter

Description:

connectionRouters.<name>.keyType

Type: KeyType (CLIENT_ID SNI_HOST SOURCE_IP USER_NAME ROLE_NAME)
Default:
XML name: key-type
Description: 可选 target key。

connectionRouters.<name>.localTargetFilter

Description:

connectionRouters.<name>.poolConfiguration

Type: PoolConfiguration
Default:
XML name: pool
Description: pool configuration.

connectionRouters.<name>.poolConfiguration.quorumTimeout

Type: int
Default: 3000
XML name: quorum-timeout
Description: timeout （以毫秒为单位），用于获取最少就绪目标的数量。

connectionRouters.<name>.poolConfiguration.password

Type:

connectionRouters.<name>.poolConfiguration.localTargetEnabled

Type: boolean
Default: false
XML name: local-target-enabled
Description: true 的值表示启用了本地目标。

connectionRouters.<name>.poolConfiguration.checkPeriod

Type: int
Default: 5000
XML name: check-period
Description: The period （毫秒）用于检查目标是否已就绪。

connectionRouters.<name>.poolConfiguration.quorumSize

Type: int
Default: 1
XML name: quorum-size
Description: 最小就绪目标数。

connectionRouters.<name>.poolConfiguration.staticConnectors

Type: List
Default:
XML name: static-connectors
Description: 通过连接器配置的连接器引用列表。

connectionRouters.<name>.poolConfiguration.discoveryGroupName

Description:

connectionRouters.<name>.poolConfiguration.clusterConnection

Type:

connectionRouters.<name>.poolConfiguration.username

Type:

connectionRouters.<name>.policyConfiguration

Type: NamedPropertyConfiguration
Default:
XML name: policy-configuration
Description:

connectionRouters.<name>..properties.{PROPERTY}

Type: Properties
Default:
XML name: property
Description: 特定于每个指定属性的键值对集合。

更新于 2024-05-21