Red Hat AMQ 7.7

AMQ Broker の設定

AMQ Broker 7.7 向け

Last Updated: 2023-02-03

Red Hat AMQ 7.7 AMQ Broker の設定

AMQ Broker 7.7 向け

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本書では、AMQ Broker の設定方法について説明します。

目次

第1章 概要

AMQ Broker 設定ファイルは、ブローカーインスタンスの重要な設定を定義します。ブローカーの設定ファイルを編集することにより、ブローカーが環境で動作する方法を制御できます。

1.1. AMQ BROKER の設定ファイルおよび場所

ブローカーの設定ファイルはすべて **<broker-instance-dir>/etc** に保存されます。これらの設定ファイルで設定を編集すると、ブローカーを設定できます。

各ブローカーインスタンスは以下の設定ファイルを使用します。

broker.xml

主要設定ファイル。このファイルを使用して、ネットワーク接続、セキュリティー設定、メッセージアドレスなどのブローカーのほとんどの側面を設定します。

bootstrap.xml

AMQ Broker がブローカーインスタンスを起動するために使用するファイル。これは、**broker.xml** の場所を変更し、Web サーバーを設定し、セキュリティー設定を行います。

logging.properties

このファイルを使用して、ブローカーインスタンスのロギングプロパティーを設定します。

artemis.profile

このファイルを使用して、ブローカーインスタンスの実行中に使用される環境変数を設定します。

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

セキュリティー関連ファイル。これらのファイルを使用して、ブローカーインスタンスへのユーザーアクセスの認証を設定します。

1.2. デフォルトブローカー設定について

broker.xml 設定ファイルを編集して、ブローカーの機能の大半を設定します。このファイルにはデフォルト設定が含まれています。これはブローカーを起動し、操作するには十分です。ただし、デフォルト設定の一部を変更し、お使いの環境にブローカーを設定するために新しい設定を追加する必要があります。

デフォルトでは、**broker.xml** には、以下の機能のデフォルト設定が含まれます。

- メッセージの永続性
- アクセプター
- セキュリティー
- メッセージアドレス

デフォルトのメッセージ永続性設定

デフォルトでは、AMQ Broker の永続性は、ディスク上のファイルセットで設定される追加のみのファイルジャーナルを使用します。ジャーナルは、メッセージ、トランザクション、およびその他の情報を保存します。

```
<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>
```

デフォルトのアクセプター設定

ブローカーは、**acceptor** 設定要素を使用して受信クライアント接続をリッスンし、クライアントが接続を作成するためにポートおよびプロトコルを定義します。デフォルトでは、AMQ Broker にはサポートされる各メッセージングプロトコルのアクセプターの設定が含まれます。

```
<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</acceptor>

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

      </acceptors>

      ...

   </core>

</configuration>
```

デフォルトのセキュリティー設定
AMQ Broker には、アドレスに基づいてキューにセキュリティーを適用するための柔軟なロールベースのセキュリティーモデルが含まれています。デフォルト設定ではワイルドカードを使用して **amq** ロールをすべてのアドレスに適用します (数値記号 **#** で表されます)。

```
<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>
```

デフォルトのメッセージアドレス設定
AMQ Broker には、作成されたキューまたはトピックに適用されるデフォルトの設定セットを確立するデフォルトアドレスが含まれています。

さらに、デフォルト設定では、**DLQ** (Dead Letter Queue) の 2 つのキューが、既知の宛先で到達するメッセージを処理します。また、**Expiry Queue** は有効期限を過ぎたメッセージを保持しているため、元の宛先にルーティングしないでください。

```
<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. 設定の更新のリロード

デフォルトでは、ブローカーは 5000 ミリ秒ごとに設定ファイルの変更をチェックします。ブローカーが設定ファイルの最後の変更されたタイムスタンプの変更を検出する場合、ブローカーは設定変更の実行を決定します。この場合、ブローカーは設定ファイルを再読み込みして変更を有効にします。

ブローカーが **broker.xml** 設定ファイルを再読み込みすると、以下のモジュールを再読み込みします。

- アドレス設定およびキュー
  設定ファイルを再読み込みすると、アドレス設定が、設定ファイルから削除されたアドレスおよびキューを処理する方法を決定します。これは、**config-delete-addresses** および **config-delete-queues** プロパティーで設定できます。詳細は、付録B *アドレス設定設定要素* を参照してください。

- セキュリティー設定
  既存のアクセプターの SSL/TLS キーストアおよびトラストストアをリロードすると、既存のクライアントに影響を与えずに新しい証明書を確立できます。接続されているクライアントは、古い証明書または異なる証明書を持つクライアントであっても、メッセージの送受信を継続できます。

- Diverts
  設定の再読み込みにより、追加した **新しい** 迂回をデプロイします。ただし、設定から迂回を削除したり、**<divert>** 要素内のサブ要素に変更しても、ブローカーを再起動するまで反映されません。

以下の手順では、ブローカーが **broker.xml** 設定ファイルへの変更をチェックする間隔を変更する方法を説明します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. **<core>** 要素内に **<configuration-file-refresh-period> 要素を追加** し、更新期間 (ミリ秒単位) を設定します。
   以下の例では、設定の更新の期間を 60000 ミリ秒に設定します。

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

1.4. ブローカー設定ファイルのモジュラー

共通の設定設定を共有する複数のブローカーがある場合は、個別のファイルに共通設定を定義し、各ブローカーの **broker.xml** 設定ファイルにこれらのファイルを含めることができます。

ブローカー間で共有できる最も一般的な設定には、以下が含まれます。

- アドレス
- アドレス設定
- セキュリティー設定

手順

1. 共有する **broker.xml** セクションごとに個別の XML ファイルを作成します。
   各 XML ファイルには、**broker.xml** の単一のセクションのみを含めることができます (例: アドレス設定またはアドレス設定のどちらかですが、両方ではありません)。top-level 要素は、要素 namespace (**xmlns="urn:activemq:core"**) も定義する必要があります。

   以下の例は、**my-security-settings.xml** で定義されたセキュリティー設定を示しています。

   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. 共通の設定を使用する各ブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。
3. 開かれた **broker.xml** ファイルごとに、以下を行います。
   a. **broker.xml** の最初の **<configuration>** 要素で、以下の行が表示されることを確認します。

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

b. 共有設定設定が含まれる各 XML ファイルに対して XML 包含を追加します。
この例には **my-security-settings.xml** ファイルが含まれます。

broker.xml

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

c. 必要に応じて、**broker.xml** を検証し、XML がスキーマに対して有効であることを確認します。
任意の XML バリデータープログラムを使用できます。この例では、**xmllint** を使用して、**artemis-server.xsl** スキーマに対して **broker.xml** を検証します。

```
$ 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
```

関連情報

- XML 包含 (XIncludes) の詳細は、https://www.w3.org/TR/xinclude/ を参照してください。

1.4.1. モジュール設定ファイルのリロード

ブローカーが設定変更を定期的にチェックすると (**configuration-file-refresh-period** で指定される頻度)、**xi:include** を使用して **broker.xml** 設定ファイルに含まれる設定ファイルへの変更を自動的に**検出しません**。たとえば、**broker.xml** に **my-address-settings.xml** が含まれ、**my-address-settings.xml** に加えられると、ブローカーは **my-address-settings.xml** の変更を自動的に検出せず、設定を再読み込みしません。

broker.xml 設定ファイルのリロードと、それに含まれる変更された設定ファイルを **強制**するには、**broker.xml** 設定ファイルの last modified タイムスタンプが変更されたことを確認する必要があります。標準の Linux **touch** コマンドを使用して、他の変更を加えずに **broker.xml** の最終変更のタイムスタンプを更新できます。以下に例を示します。

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

1.5. 本書の表記慣例

本書では、**sudo** コマンドおよびファイルパスについて、以下の表記慣例を使用します。

sudo コマンド
本書では、root 権限を必要とするすべてのコマンドに対して **sudo** が使用されています。何らかの変更がシステム全体に影響を与える可能性があるため、**sudo** を使用する場合は、常に注意が必要です。

sudo の使用の詳細は、**sudo コマンド** を参照してください。

本書におけるファイルパスの使用

本書では、すべてのファイルパスは Linux、UNIX、および同様のオペレーティングシステムで有効です (例: /**home**/**...**)。Microsoft Windows を使用している場合は、同等の Microsoft Windows パスを使用する必要があります (例: **C:\Users\...**)。

第2章 ネットワーク接続: アクセプターとコネクター

AMQ Broker では、network と In-VM の 2 種類の接続が使用されます。ネットワーク接続は、同じサーバーまたは物理的にリモート上に関係なく、2 つの当事者が異なる仮想マシンにある場合に使用されます。仮想マシン内の接続は、クライアント (アプリケーションかサーバーかに関係なく) がブローカーと同じ仮想マシンにある場合に使用されます。

ネットワーク接続は Netty に依存します。Netty は、複数の方法でネットワーク接続を設定できるようにする、高パフォーマンスの低レベルネットワークライブラリーです。Java IO または NIO、TCP ソケット、SSL/TLS、または HTTP または HTTPS でのトンネリングの使用。Netty は、すべてのメッセージングプロトコルに単一のポートを使用することもできます。ブローカーは、使用されているプロトコルを自動的に検出し、さらに処理するために受信メッセージを適切なハンドラーに送ります。

ネットワーク接続の設定内の URI はそのタイプを決定します。たとえば、URI で **vm** を使用すると、In-VM 接続が作成されます。以下の例では、**acceptor** の URI が **vm** で始まることに注意してください。

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

URI で **tcp** を使用すると、ネットワーク接続が作成されます。

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

本章では、最初にネットワーク接続、アクセプター、コネクターに固有の 2 つの設定要素について説明します。次に、TCP、HTTP、および SSL/TLS ネットワーク接続の設定手順と、In-VM 接続について説明します。

2.1. アクセプター

AMQ Broker でネットワーク接続を説明する際に最も重要な概念の 1 つが **acceptor** です。アクセプターは、ブローカーへの接続方法を定義します。以下は、**BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイル内で見つかった **acceptor** の典型的な設定です。

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

各 **acceptor** は **acceptors** 要素内でグループ化されることに注意してください。サーバーごとに一覧表示できるアクセプターの数の上限はありません。

アクセプターの設定

acceptor に定義された URI のクエリー文字列にキーと値のペアを追加して **acceptor** を設定します。以下の例のように、セミコロン (';') を使用して複数のキーと値のペアを区切ります。**sslEnabled=true** で始まる URI の最後に複数のキーと値のペアを追加して、SSL/TLS のアクセプターを設定します。

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

connector 設定パラメーターの詳細は、Acceptor and Connector Configuration Parameters を参照してください。

2.2. コネクター

アクセプターはサーバーが接続を受け入れる方法を定義しますが、**connector** はクライアントによって使用され、サーバーへの接続方法を定義します。

以下は、**BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルで定義される通常の **connector** です。

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

コネクターは **connectors** 要素内で定義されることに注意してください。サーバーごとのコネクター数の上限はありません。

コネクターはクライアントによって使用されますが、アクセプターと同様にサーバーで設定されます。重要な理由には、以下の 2 つの理由があります。

- サーバー自体はクライアントとして機能するため、他のサーバーへの接続方法を把握しておく必要があります。たとえば、あるサーバーが別のサーバーにブリッジされている場合や、サーバーがクラスター内の一部を取得する場合などです。
- サーバーは通常、接続ファクトリーインスタンスを検索するために JMS クライアントによって使用されます。このような場合、JNDI はクライアント接続の作成に使用される接続ファクトリーの詳細を知っておく必要があります。情報は、JNDI ルックアップの実行時にクライアントに提供されます。詳細は、Configuring a Connection on the Client Side を参照してください。

コネクターの設定
アクセプターと同様に、コネクターには URI のクエリー文字列に割り当てられた設定があります。以下は、**tcpNoDelay** パラメーターが **false** に設定されている **connector** の例になります。これにより、この接続の Nagle アルゴリズムが無効になります。

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

connector 設定パラメーターの詳細は、Acceptor and Connector Configuration Parameters を参照してください。

2.3. TCP 接続の設定

AMQ Broker は Netty を使用して基本的な、暗号化されていない TCP ベースの接続を提供します。この接続は、ブロッキング Java IO または新しい非ブロッキング Java NIO を使用するように設定できます。多くの同時接続でスケーラビリティーを向上させるために、Java NIO が推奨されます。ただし、古い IO を使用すると、何万もの同時接続のサポートを受けても NIO より優れたレイテンシーが発生することがあります。

信頼できないネットワーク全体で接続を実行している場合は、TCP ネットワーク接続が暗号化されないことを認識する必要があります。暗号化が優先される場合には、SSL または HTTPS 設定を使用して、この接続で送信されたメッセージを暗号化することを検討してください。詳細は、トランスポート層セキュリティーの設定 を参照してください。

TCP 接続を使用すると、すべての接続がクライアント側から開始されます。つまり、サーバーはクライアントへの接続を開始しません。これは、ある方向から接続を強制するファイアウォールポリシーで適切に機能します。

TCP 接続では、コネクター URI のホストおよびポートは接続に使用されるアドレスを定義します。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 接続を追加または変更し、**tcp** をプロトコルとして使用する URI を追加します。IP またはホスト名、ポートの両方を含めるようにしてください。

以下の例では、**acceptor** は TCP 接続として設定されます。この **acceptor** で設定されたブローカーは、IP **10.10.10.1** およびポート **61617** への TCP 接続を行うクライアントを受け入れます。

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

同様の方法で TCP を使用するようにコネクターを設定します。

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

上記の **connector** は、指定された IP およびポート **10.10.10.2 :61617** に TCP 接続を確立する際に、クライアントまたはブローカー自体によって 参照されます。

TCP 接続で利用可能な設定パラメーターの詳細は、Acceptor and Connector Configuration Parameters を参照してください。ほとんどのパラメーターはアクセプターまたはコネクターで使用することができますが、アクセプターでのみ機能するものもあります。

2.4. HTTP 接続の設定

HTTP プロトコルを介した HTTP 接続トンネルは、ファイアウォールが HTTP トラフィックのみを許可する場合に便利です。単一ポートサポートでは、AMQ Broker は HTTP が使用されているかどうかを自動的に検出するため、HTTP のネットワーク接続の設定は TCP の接続の設定と同じです。HTTP の使用方法に関する詳細は、**INSTALL_DIR/examples/features/standard/** にある **http-transport** の例を参照してください。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 接続を追加または変更し、**tcp** をプロトコルとして使用する URI を追加します。IP またはホスト名、およびポートの両方を含めるようにしてください。

以下の例では、ブローカーは IP アドレス **10.10.10.1** でポート **80** に接続するクライアントからの HTTP 通信を受け入れます。さらに、ブローカーは HTTP プロトコルが使用されていることを自動検出し、それに応じてクライアントと通信します。

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

HTTP のコネクターの設定は TCP の場合と同じです。

```
<connectors>
```

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

上記の例の設定を使用すると、ブローカーは IP アドレス **10.10.10.2** でポート **80** へのアウトバウンド HTTP 接続を作成し ます。

HTTP 接続は TCP と同じ設定パラメーターを使用しますが、いくつかの設定パラメーターもあります。HTTP 関連およびその他の設定パラメーターの詳細は、Acceptor and Connector Configuration Parameters を参照してください。

2.5. SSL/TLS 接続の設定

SSL/TLS を使用するように接続を設定することもできます。詳細は、トランスポート層セキュリティーの設定 を参照してください。

2.6. 仮想マシン内の接続設定

たとえば、高可用性ソリューションの一部として、複数のブローカーが同じ仮想マシンに共存する場合に、In-VM 接続を使用できます。仮想マシン内のマシン接続は、ブローカーと同じ JVM で実行されているローカルクライアントでも使用できます。in-VM 接続では、URI の認証局部分は一意のサーバー ID を定義します。実際、URI の他の部分は必要ありません。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 接続を追加または変更し、**vm** をプロトコルとして使用する URI を追加します。

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

上記の **acceptor** の例では、ID が **0** のサーバーからの接続を受け入れるようにブローカーに指示します。他のサーバーはブローカーと同じ仮想マシンで実行されている必要があります。

コネクターを in-vm コネクションとして設定する場合は、同じ構文に従います。

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

上記の例の **connector** は、クライアントが同じ仮想マシンにある ID が **0** のサーバーへの in-VM 接続を確立する方法を定義します。クライアントはアプリケーションまたはブローカーになります。

2.7. クライアント側からの接続の設定

コネクターは、クライアントアプリケーションでも間接的に使用されます。サーバー側で **connector** を定義することなく、JMS 接続ファクトリーを直接クライアント側に設定できます。

```
Map<String, Object> connectionParams = new HashMap<String, Object>();
```

```
connectionParams.put(org.apache.activemq.artemis.core.remoting.impl.netty.TransportConstants.POR
T_PROP_NAME, 61617);

TransportConfiguration transportConfiguration =
    new TransportConfiguration(
    "org.apache.activemq.artemis.core.remoting.impl.netty.NettyConnectorFactory",
connectionParams);

ConnectionFactory connectionFactory =
ActiveMQJMSClient.createConnectionFactoryWithoutHA(JMSFactoryType.CF,
transportConfiguration);

Connection jmsConnection = connectionFactory.createConnection();
```

第3章 ネットワーク接続: プロトコル

AMQ Broker にはプラグ可能なプロトコルアーキテクチャーがあるため、ネットワーク接続に 1 つ以上のプロトコルを簡単に有効化できます。

ブローカーは以下のプロトコルをサポートします。

- AMQP
- MQTT
- OpenWire
- STOMP

注記

上記のプロトコルのほかに、ブローカーは Core と呼ばれる独自のネイティブプロトコルもサポートします。このプロトコルの以前のバージョンは HornetQ と呼ばれ、Red Hat JBoss Enterprise Application Platform によって使用されていました。

3.1. プロトコルを使用するためのネットワーク接続の設定

使用する前に、プロトコルをネットワーク接続に関連付ける必要があります。(ネットワーク接続の作成および設定方法の詳細は、ネットワーク接続: アクセプターおよびコネクター を参照してください。) **BROKER_INSTANCE_DIR/etc/broker.xml** ファイルにあるデフォルト設定には、すでに定義された接続が複数含まれています。便宜上、AMQ Broker にはサポートされる各プロトコルのアクセプターと、すべてのプロトコルをサポートするデフォルトのアクセプターが含まれます。

デフォルトのアクセプターの概要
以下は、**broker.xml** 設定ファイルにデフォルトで含まれるアクセプターです。

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

      <!-- All-protocols acceptor -->
      <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>
```

特定のネットワーク設定でプロトコルを有効にする唯一の要件は、**protocols** パラメーターを アクセプターの URI に追加することです。パラメーターの値は、プロトコル名のコンマ区切りリストである必要があります。protocol パラメーターが URI から省略される場合、すべてのプロトコルが有効になります。

たとえば、AMQP プロトコルを使用して 3232 番ポートでメッセージを受信するためにアクセプターを作成するには、以下の手順に従います。

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 以下の行を **<acceptors>** スタンザに追加します。

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

デフォルトのアクセプターの追加パラメーター

最小限のアクセプター設定では、接続 URI の一部としてプロトコルを指定します。ただし、**broker.xml** 設定ファイルのデフォルトのアクセプターには追加のパラメーターが設定されています。以下の表は、デフォルトのアクセプターに設定された追加パラメーターの詳細を示しています。

Acceptor(s)	パラメーター	説明
All-protocols acceptor AMQP STOMP	tcpSendBufferSize	TCP 送信バッファーのサイズ (バイト単位)。デフォルト値は **32768** です。
	tcpReceiveBufferSize	TCP 受信バッファーのサイズ (バイト単位)。デフォルト値は **32768** です。 TCP バッファーサイズは、ネットワークの帯域幅およびレイテンシーに従って調整する必要があります。 つまり、TCP の送信/受信バッファーサイズは以下のように計算する必要があります。 buffer_size = bandwidth * RTT 帯域幅とは秒単位で、ネットワークラウンドトリップタイム (RTT) は秒単位になります。RTT は、**ping** ユーティリティーを使用して簡単に測定できます。 高速ネットワークでは、デフォルトからバッファーサイズを増やす必要がある場合があります。

Acceptor(s)	パラメーター	説明
All-protocols acceptor AMQP STOMP HornetQ MQTT	useEpoll	サポートするシステム (Linux) を使用する場合は Netty epoll を使用します。Netty ネイティブトランスポートは NIO トランスポートよりも優れたパフォーマンスを提供します。このオプションのデフォルト値は **true** です。オプションを **false** に設定すると、OIO が使用されます。
All-protocols acceptor AMQP	amqpCredits	メッセージの合計サイズに関係なく、AMQP プロデューサーが送信できるメッセージの最大数。デフォルト値は **1000** です。 AMQP メッセージのフローを制御するためにクレジットが使用される方法の詳細は、「AMQP メッセージのブロック」 を参照してください。
All-protocols acceptor AMQP	amqpLowCredits	ブローカーによってプロデューサーのクレジットが返送される低いしきい値。デフォルト値は **300** です。プロデューサーがこのしきい値に達すると、ブローカーはプロデューサーに十分なクレジットを送信し、**amqpCredits** 値を復元します。 AMQP メッセージのフローを制御するためにクレジットが使用される方法の詳細は、「AMQP メッセージのブロック」 を参照してください。
HornetQ 互換性アクセプター	anycastPrefix	anycast および **multicast** の両方を使用するアドレスに接続するときに、クライアントが **anycast** および **anycast** ルーティングタイプを指定するために使用する接頭辞。デフォルト値は **jms.queue** です。 アドレスへの接続時にクライアントがルーティングタイプを指定できるように接頭辞を設定する方法は、「特定のルーティングタイプに接続するための接頭辞設定」 を参照してください。
	multicastPrefix	**anycast** および **multicast** の両方を使用するアドレスへの接続時に **multicast** ルーティングタイプを指定するためにクライアントが使用する接頭辞。デフォルト値は **jms.topic** です。 アドレスへの接続時にクライアントがルーティングタイプを指定できるように接頭辞を設定する方法は、「特定のルーティングタイプに接続するための接頭辞設定」 を参照してください。

関連情報

- Netty ネットワーク接続に設定できるその他のパラメーターに関する情報は、付録A *アクセプターおよびコネクター設定パラメーター* を参照してください。

3.2. ネットワーク接続での AMQP の使用

ブローカーは AMQP 1.0 仕様をサポートします。AMQP リンクは、ソースとターゲット間のメッセージ (クライアントとブローカー) の一方向プロトコルです。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 以下の例のように、AMQP の値を持つ **protocols** パラメーターを URI の一部として追加または設定することで、**acceptor** を追加または設定し、**AMQP** クライアントを受信します。

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

上記の例では、ブローカーはデフォルトの AMQP ポートであるポート 5672 で AMQP 1.0 クライアントを受け入れます。

AMQP リンクには、送信者と受信側の 2 つのエンドポイントがあります。送信者がメッセージを送信する場合、ブローカーはこれを内部形式に変換するため、ブローカーの宛先に転送できます。受信側はブローカーの宛先に接続し、メッセージを配信前に AMQP に戻します。

AMQP リンクが動的の場合、一時キューが作成され、リモートソースまたはリモートターゲットアドレスのいずれかが一時キューの名前に設定されます。リンクが動的ではない場合、リモートターゲットまたはソースのアドレスがキューに使用されます。リモートターゲットまたはソースが存在しない場合は、例外が発生します。

リンクターゲットは、基礎となるセッションをトランザクションとして処理するために使用される Coordinator でも、ロールバックまたはコミットします。

注記

AMQP ではセッションごとに複数のトランザクション **amqp:multi-txns-per-ssn** を使用できますが、現在のバージョンの AMQ Broker はセッションごとに単一のトランザクションのみをサポートします。

注記

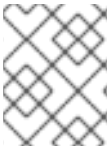
AMQP 内の分散トランザクション (XA) の詳細は、仕様の 1.0 バージョンでは提供されません。お使いの環境で分散トランザクションのサポートが必要な場合は、AMQ Core Protocol JMS を使用することが推奨されます。

プロトコルとその機能の詳細は、AMQP 1.0 仕様を参照してください。

3.2.1. AMQP リンクをトピックとして使用

JMS とは異なり、AMQP プロトコルにはトピックが含まれません。ただし、キューのコンシューマーだけでなく、AMQP コンシューマーまたは受信側をサブスクリプションとして扱うことは可能です。デフォルトでは、接頭辞 **jms.topic.** でアドレスにアタッチする受信リンクがサブスクリプションとして処理され、サブスクリプションキューが作成されます。以下の表でキャプチャーされているように、Terminus Durability の設定方法に応じて、サブスクリプションキューが永続的または揮発性になります。

この種のマルチキャスト専用キューのサブスクリプションを作成します。	Terminus Durability を 下記のようにセットします。
永続性	**UNSETTLED_STATE** または **CONFIGURATION**
Non-durable	**NONE**



注記

永続キューの名前は、コンテナー ID とリンク名 (例: **my-container-id:my-link-name**) で設定されます。

AMQ Broker は qpid-jms クライアントもサポートし、アドレスに使用される接頭辞に関係なくトピックの使用に対応します。

3.2.2. AMQP セキュリティーの設定

ブローカーは AMQP SASL 認証をサポートします。ブローカーで SASL ベースの認証を設定する方法は、Security を参照してください。

3.3. ネットワーク接続での MQTT の使用

ブローカーは MQTT v3.1.1(および古い v3.1 コードメッセージ形式) をサポートします。MQTT は軽量のクライアントからサーバーへ、パブリッシュ/サブスクライブメッセージングプロトコルです。MQTT は、メッセージングのオーバーヘッドおよびネットワークトラフィック、およびクライアントのコードフットプリントを削減します。このような理由から、MQTT は、センサーやアクチュエーターなどのデバイスを制限するのが適しており、この ng(IoT) のインターネット向けの de facto 標準通信プロトコルがすばやく考えられます。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. MQTT プロトコルが有効になっているアクセプターを追加します。以下に例を示します。

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

MQTT には、以下を含む便利な機能が多数含まれています。

QoS (Quality of Service)

各メッセージは、関連付けられた QoS(Quality of Service) を定義できます。ブローカーは、定義された最大 QoS(Quality of Service) レベルで、サブスクライバーに対してメッセージの配信を試みます。

保持されるメッセージ

特定のアドレスに対してメッセージを保持できます。クライアントの接続前に保持されるメッセージが送信された場合でも、他のメッセージの前に最後に保持されたメッセージを受信する新しいサブスクライバー。

ワイルドカードサブスクリプション

MQTT アドレスは、ファイルシステムの階層と同様に階層です。クライアントは、特定のトピックや、階層のブランチ全体にサブスクライブできます。

メッセージ

クライアントは、接続パケットの一部として will message を設定できます。クライアントが異常に切断されると、ブローカーは指定されたアドレスにメッセージを公開します。他のサブスクライバーはメッセージを受信し、対応できます。

MQTT プロトコルに関する情報の最適なソースは、仕様にあります。MQTT v3.1.1 仕様は、OASIS Web サイト からダウンロードできます。

3.4. ネットワーク接続での OPENWIRE の使用

ブローカーは OpenWire protocol プロトコルをサポートします。これにより、JMS クライアントはブローカーと直接対話できます。このプロトコルを使用して、古いバージョンの AMQ Broker と通信します。

現在、AMQ Broker は標準の JMS API のみを使用する OpenWire クライアントをサポートします。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 以下の例にあるように、**acceptor** を追加または変更して、**protocol** パラメーターの一部として **OPENWIRE** が含まれるようにします。

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

上記の例では、ブローカーは受信 OpenWire コマンドのポート 61616 でリッスンします。

詳細は、**INSTALL_DIR/examples/protocols/openwire** にある例を参照してください。

3.5. ネットワーク接続による STOMP の使用

STOMP は、STOMP クライアントが STOMP Broker と通信できるようにするテキスト指向のワイヤプロトコルです。ブローカーは STOMP 1.0、1.1、および 1.2 をサポートします。STOMP クライアントは複数の言語やプラットフォームで利用できます。これにより、相互運用性の選択肢があります。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 既存の **acceptor** を設定するか、または新しいアクセプターを作成し、以下のように **STOMP** の値を持つ **protocols** パラメーターを追加します。

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

上記の例では、ブローカーはポート **61613** の STOMP 接続を受け入れます。

STOMP を使用したブローカーの設定例については、 **INSTALL_DIR/examples/protocols** にある **stomp** の例を参照してください。

3.5.1. STOMP を使用する場合の制限について

STOMP を使用する場合、以下の制限が適用されます。

1. 現在、ブローカーは仮想ホストをサポートしません。つまり、**host** フレーム の **CONNECT** ヘッダーは無視されます。

2. メッセージ確認応答はトランザクションではありません。**ACK** フレームはトランザクションの一部にできず、**transaction** ヘッダーが設定されている場合は無視されます。

3.5.2. STOMP メッセージの ID の提供

JMS コンシューマーまたは QueueBrowser を介して STOMP メッセージを受信する場合、メッセージには **JMSMessageID** などの JMS プロパティーは含まれません。ただし、ブローカーパラメーターを使用して、各受信 STOMP メッセージにメッセージ ID を設定できます。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。

2. 以下の例のように、STOMP 接続に使用される **acceptor** の **stompEnableMessageId** パラメーターを **true** に設定します。

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

stompEnableMessageId パラメーターを使用することで、このアクセプターを使用して送信される各 stomp メッセージには追加のプロパティーが追加されます。property キーは **amq-message-id** で、以下の例のように、値は 「STOMP」 で始まる内部メッセージ ID の String 表現です。

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

設定で **stompEnableMessageId** が指定されていない場合、デフォルト値は **false** になります。

3.5.3. 接続の Time to Live (TTL) の設定

STOMP クライアントは、接続を閉じる前に **DISCONNECT** フレームを送信する必要があります。これにより、ブローカーはセッションやコンシューマーなどのサーバー側のリソースを閉じることができます。ただし、STOMP クライアントが DISCONNECT フレームを送信せずに終了する場合、または失敗すると、ブローカーにはクライアントが有効であるかどうかを即座に認識することができません。そのため、STOMP の接続は TTL(Time to Live) が 1 分となるように設定されています。複数の 1 分間の間アイドル状態であった場合、ブローカーは STOMP クライアントへの接続を停止します。

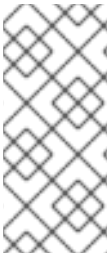
手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。

2. 以下の例のように、STOMP 接続に使用される **acceptor** の URI に **connectionTTL** パラメーターを追加します。

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

上記の例では、stomp **-acceptor** を使用するすべての stomp 接続では、その TTL を 20 秒に設定します。



注記

STOMP プロトコルのバージョン 1.0 には ハートビートフレーム が含まれていません。そのため、ユーザーは、データが connection-ttl 内に送信されることを確認するか、ブローカーがクライアントが停止されサーバー側のリソースをクリーンアップすることを想定します。バージョン 1.1 では、c he-beats を使用して stomp 接続のライフサイクルを維持することができます。

ブローカーのデフォルト Time to Live (TTL) の上書き
前述のように、STOMP 接続のデフォルトの TTL は 1 分です。この値は、**connection-ttl-override** 属性をブローカー設定に追加することで上書きできます。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. **connection-ttl-override** 属性を追加し、新しいデフォルトの値をミリ秒単位で指定します。以下のように、**<core>** スタンザに属します。

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

上記の例では、STOMP 接続のデフォルトの Time to Live(TTL) は 30 秒 (**30000** ミリ秒) に設定されます。

3.5.4. JMS からの STOMP メッセージの送受信

STOMP は主にテキスト指向のプロトコルです。JMS と相互運用を容易にするため、STOMP 実装は **content-length** ヘッダーの有無を確認し、STOMP メッセージを JMS にマップする方法を決定します。

STOMP のメッセージを…にマッピングしたい場合。	メッセージは下記の通りにしてください。
JMS TextMessage	**content-length** ヘッダーを**含めないでください**。
JMS BytesMessage	**content-length** ヘッダーを**含めます**。

JMS メッセージを STOMP にマップする場合も、同じロジックが適用されます。STOMP クライアントは、**content-length** ヘッダーが存在することを確認して、メッセージボディーのタイプ (文字列またはバイト) を判別できます。

メッセージヘッダーの詳細については、STOMP 仕様を参照してください。

3.5.5. STOMP 宛先の AMQ Broker アドレスおよびキューへのマッピング

メッセージを送信してサブスクライブする場合、STOMP クライアントには通常、**destination** ヘッダーが含まれます。宛先名は文字列の値で、ブローカーの宛先にマッピングされます。AMQ Broker では、これらの宛先は **アドレス** および **キュー** にマッピングされます。宛先フレームの詳細は、STOMP 仕様を参照してください。

たとえば、以下のメッセージ (ヘッダーおよびボディー) を送信する STOMP クライアントの例を考えます。

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

hello queue a
^@
```

この場合、ブローカーは **/my/stomp/queue** アドレスに関連付けられたキューへメッセージを転送します。

たとえば、STOMP クライアントが (**SEND** フレームを使用して) メッセージを送信する場合、指定された宛先がアドレスにマッピングされます。

これは、クライアントが **SUBSCRIBE** または **UNSUBSCRIBE** フレームを送信する場合 と同じように機能しますが、この場合は AMQ Broker は **destination** をキューにマッピングします。

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

^@
```

上記の例では、ブローカーは **destination** を キュー **/other/stomp/queue** にマップします。

STOMP 宛先と JMS 宛先のマッピング
JMS 宛先は、ブローカーアドレスおよびキューにマッピングされます。STOMP を使用してメッセージを JMS 宛先に送信する場合、STOMP 宛先は同じ規則に従う必要があります。

- JMS **Queue** を送受信するには、**jms.queue.** でキュー名を追加します。たとえば、JMS キューの **orders** にメッセージを送信するには、STOMP クライアントはフレームを送信する必要があります。

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

- **jms.topic.** でトピック名を追加し、JMS **Topic** を送信またはサブスクライブします。たとえば、**stocks** JMS Topic をサブスクライブするには、STOMP クライアントは以下のようなフレームを送信する必要があります。

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

第4章 アドレス、キュー、およびトピック

AMQ Broker には、強力で柔軟性があり、優れたパフォーマンスを提供する独自のアドレス指定モデルがあります。アドレスモデルは、addresses、queues、およびrouting types の 3 つの主要な概念で構成されています。

アドレス はメッセージングエンドポイントを表します。設定内では、通常のアドレスには一意の名前、0 個以上のキュー、およびルーティングタイプが指定されます。

キューがアドレスに関連付けられます。アドレスごとに複数のキューが存在する場合があります。受信メッセージがアドレスにマッチすると、設定されたルーティングタイプに応じて、メッセージは 1 つ以上のキューに送信されます。キューは、自動作成および削除ができるように設定できます。また、アドレス (およびその関連付けられたキュー) を **永続** として設定できます。キューのメッセージも永続永続キューにある限り、永続キューのメッセージも永続し、ブローカーのクラッシュや再起動を保ち続けることができます。一方、非永続キューのメッセージは、メッセージ自体が永続的であっても、ブローカーのクラッシュや再起動は維持されません。

ルーティングタイプ は、アドレスに関連付けられたキューへメッセージが送信される方法を決定します。AMQ Broker アドレスは、2 つの異なるルーティングタイプで設定できます。

表4.1ルーティングタイプ

メッセージをルーティング先とルーティングする場合	このルーティングタイプの使用...
ポイントツーポイント方式で、一致するアドレス内の単一キュー。	anycast
パブリッシュ/サブスクライブ方式で、一致するアドレス内のすべてのキュー。	multicast

注記

アドレスには、少なくとも 1 つのルーティングタイプが必要です。

アドレスごとに複数のルーティングタイプを定義できますが、これは通常、パターンに反する結果になるため、推奨されません。

ただし、アドレスが両方のルーティングタイプを使用し、クライアントがどちらにも優先していない場合、ブローカーは通常、デフォルトで **anycast** ルーティングタイプに設定されます。1 つの例外は、クライアントが MQTT プロトコルを使用する場合です。この場合、デフォルトのルーティングタイプは **multicast** です。

4.1. アドレスおよびキューの命名要件

アドレスおよびキューを設定する場合は、以下の要件に注意してください。

- クライアントが使用するワイヤプロトコルに関係なく、クライアントがキューに接続できるようにするには、アドレスおよびキュー名には以下のいずれの文字も含めないでください。
  & :: , ? >
- **#** および ***** 文字は、ワイルドカード表現用に予約されています。詳細は、「AMQ Broker ワイルドカード構文」 を参照してください。
- アドレスおよびキュー名にはスペースを含めないでください。

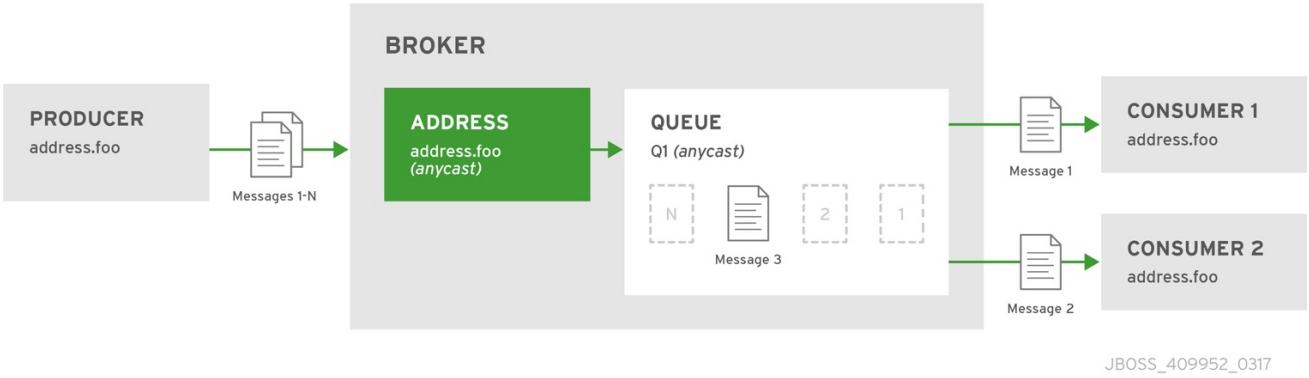
- アドレスまたはキュー名の単語を区切るには、設定された区切り文字 (デフォルトは . 文字) を使用します。詳細は、「AMQ Broker ワイルドカード構文」 を参照してください。

4.2. ポイントツーポイントメッセージングの設定

ポイントツーポイントメッセージングは、プロデューサーによって送信されたメッセージが 1 つのコンシューマーのみを持つ一般的なシナリオです。AMQP および JMS メッセージプロデューサーおよびコンシューマーは、たとえば、ポイントツーポイントメッセージングキューを使用できます。**address** に **anycast** ルーティングタイプを定義して、そのキューがポイントツーポイント方式でメッセージを受信できるようにします。

anycast を使用してアドレスでメッセージを受信すると、ブローカーはアドレスに関連付けられたキューを見つけ、そこにメッセージをルーティングします。コンシューマーがアドレスから使用の要求を出すと、ブローカーは関連するキューを見つけ、このキューを適切なコンシューマーに関連付けます。複数のコンシューマーが同じキューに接続されている場合、コンシューマーが均等に処理できる前提で、メッセージが各コンシューマー間で均等に分散されます。

図4.1 Point-to-Point



手順

1. ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** を開いて編集します。
2. **address** の選択された **queue** エレメントの周りに、**anycast** 設定エレメントをラップします。**アドレス** 名要素と **キュー名** 要素の値が同じであることを確認してください。

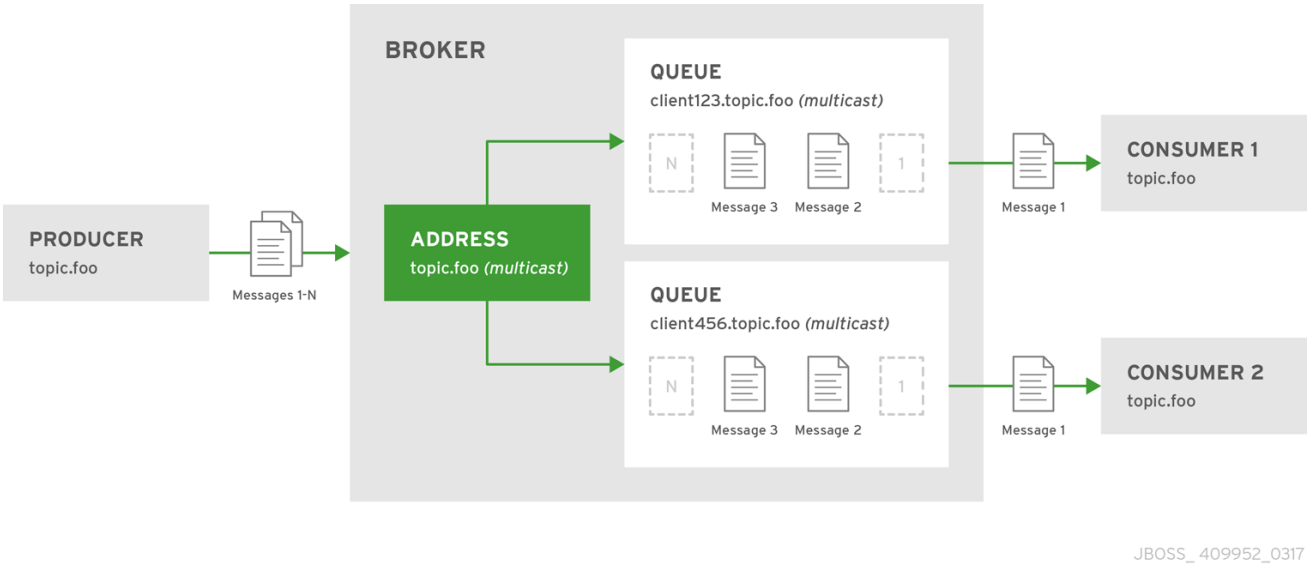
```
<configuration ...>
  <core ...>
    ...
    <address name="durable">
      <anycast>
        <queue name="durable"/>
      </anycast>
    </address>
  </core>
</configuration>
```

4.3. パブリッシュ/サブスクライブメッセージングの設定

パブリッシュ/サブスクライブのシナリオでは、メッセージはアドレスにサブスクライブしているすべてのコンシューマーに送信されます。JMS トピックおよび MQTT サブスクリプションは、パブリッシュ/サブスクライブメッセージングの 2 つの例です。メッセージが **multicast** ルーティングタイプの

アドレスで受信されると、AMQ ブローカはメッセージのコピーを各キューにルーティングします。コピーのオーバーヘッドを減らすために、各キューにはメッセージへの参照のみが送信され、完全なコピーは送信されません。

図4.2 パブリッシュ - サブスクライブ



手順

1. ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** を開いて編集します。
2. 空の **multicast** 設定要素を選択したアドレスに追加します。

```
<configuration ...>
  <core ...>
    ...
    <address name="topic.foo">
      <multicast/>
    </address>
  </core>
</configuration>
```

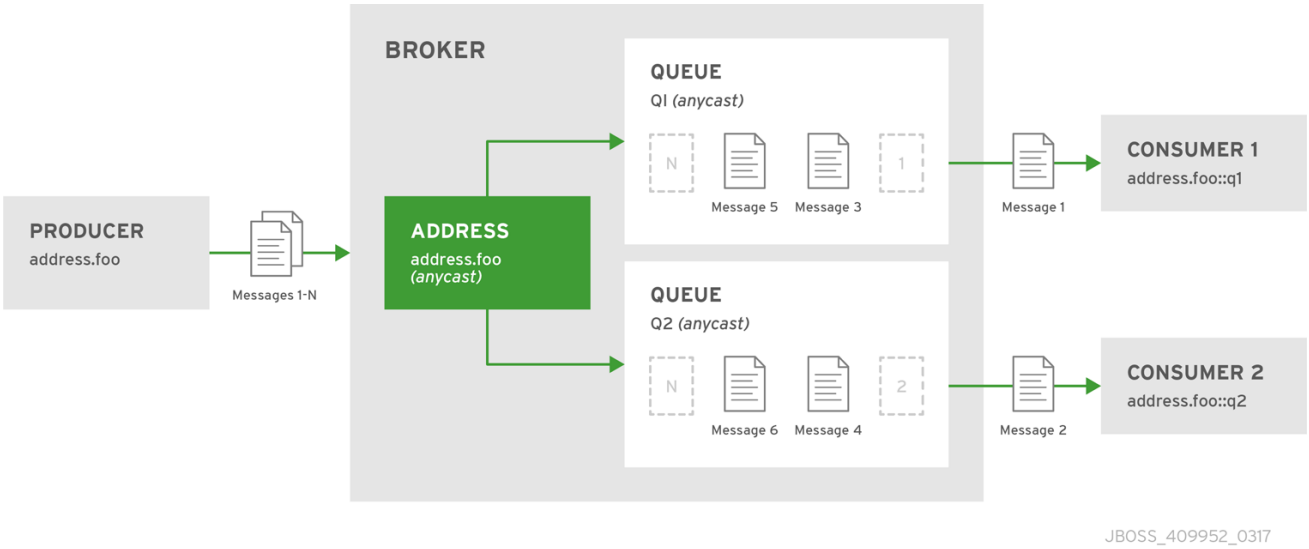
3. (オプション) 1 つ以上の **queue** 要素をアドレスに追加し、その中の **multicast** 要素をラップします。ブローカーはクライアントがリクエストする各サブスクリプションのキューを自動的に作成するため、この手順は必要ありません。

```
<configuration ...>
  <core ...>
    ...
    <address name="topic.foo">
      <multicast>
        <queue name="client123.topic.foo"/>
        <queue name="client456.topic.foo"/>
      </multicast>
    </address>
  </core>
</configuration>
```

4.4. 2 つのキューを使用したポイントツーポイントの設定

anycast ルーティングタイプを使用するアドレスで複数のキューを定義できます。**anycast** アドレスに送信されるメッセージは、関連するすべてのキューに均等に分散されます。後で説明する完全修飾キュー名を使用して、クライアントを特定のキューに接続させることができます。複数のコンシューマーが同じキューに接続している場合、AMQ Broker はそれらの間でメッセージを配信します。

図4.3 2 つのキューによるポイントツーポイント



注記

これは、AMQ Broker がクラスターにある複数のノード間でキューの負荷分散を処理する方法です。

手順

1. ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** を開いて編集します。
2. **address** の **queue** 要素で **anycast** 設定要素をラップします。

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

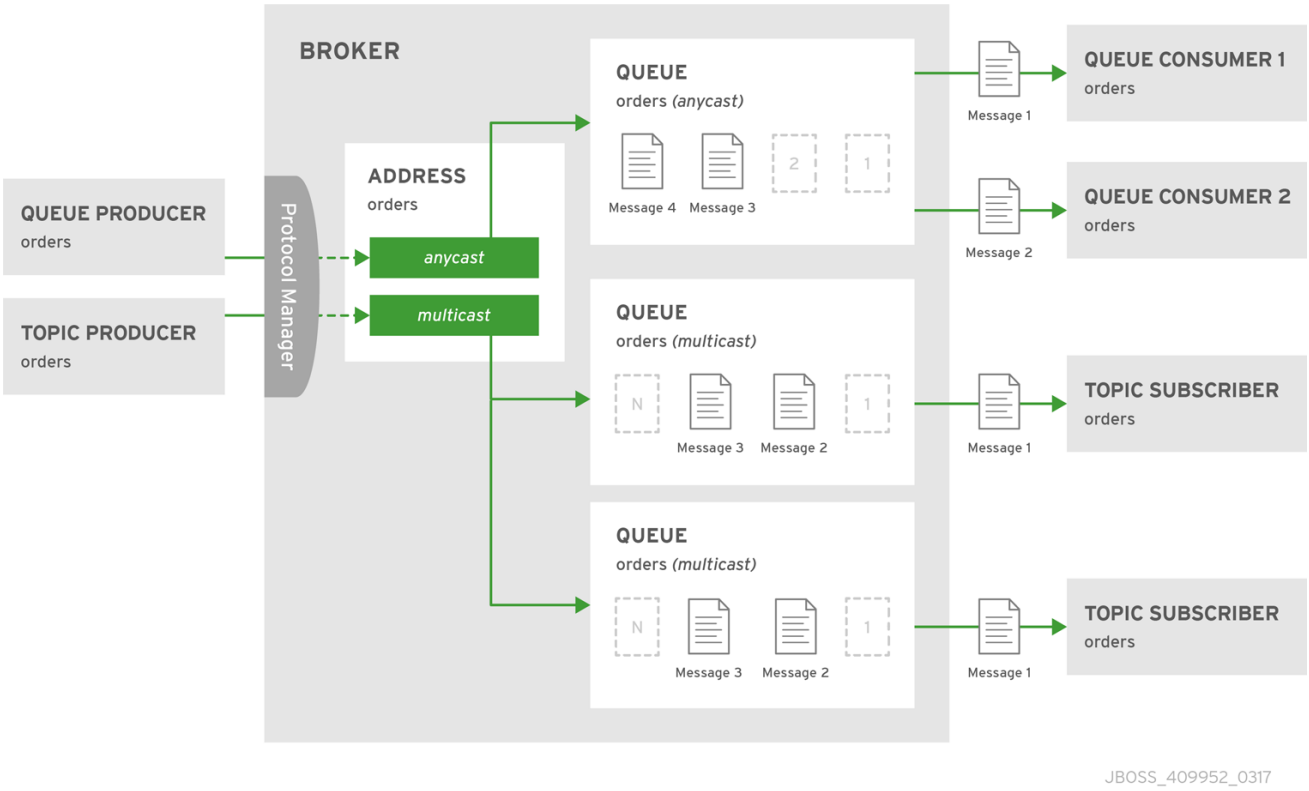
4.5. ポイントツーポイントとパブリッシュ - サブスクライブの併用

ポイントツーポイントとパブリッシュ/サブスクライブの両方のセマンティクスを有効にしてアドレスを定義することができます。通常は推奨されませんが、これは、**orders** という名前の JMS キューと、**orders** という名前の JMS トピックが必要な場合などに役立ちます。ルーティングの種類が異なるため、アドレスが異なっているように見えます。

JMS クライアントの例を使用すると、JMS キュープロデューサにーよって送信されるメッセージ

は、**anycast** ルーティングタイプを使用してルーティングされます。JMS トピックプロデューサーによって送信されるメッセージは、**multicast** ルーティング タイプを使用します。さらに、JMS トピックコンシューマーがアタッチすると、独自のサブスクリプションキューにアタッチされます。ただし、JMS キューコンシューマーは **anycast** キューに割り当てられます。

図4.4 ポイントツーポイントおよびパブリッシュ - サブスクライブ



注記

このシナリオの動作は、使用されるプロトコルによって異なります。JMS の場合、トピックとキュープロデューサーとコンシューマーの間に明確な区別があり、これによりロジックを簡単に転送できます。AMQP などの他のプロトコルはこの区別を行いません。AMQP 経由で送信されるメッセージは、**anycast** および **multicast** によってルーティングされ、コンシューマーデフォルトは **anycast** です。詳細は、プロトコルのセクションで各プロトコルの動作を確認してください。

以下の XML の抜粋は、**BROKER_INSTANCE_DIR/etc/broker.xml** で、**anycast** と **multicast** の両方のルーティングタイプを使用するアドレスの設定の例です。通常、サブスクリプションキューはオンデマンドで作成されるため、**multicast** ルーティングタイプ内に特定の **queue** 要素をリストする必要はありません。

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

4.6. サブスクリプションキューの設定

ほとんどの場合、プロトコルマネージャーは、クライアントが最初にアドレスへのサブスクライブを要求したときに、自動的にサブスクリプションキューを作成するため、サブスクリプションキューを事前に作成する必要はありません。詳細は 、プロトコルマネージャーとアドレス を参照してください。永続サブスクリプションの場合、生成されたキュー名はクライアント ID とアドレスを連結したものです。

永続サブスクリプションキューの設定

キューが永続サブスクリプションとして設定されると、ブローカーは非アクティブなサブスクライバーのメッセージを格納し、再接続時にサブスクライバーに配信します。そのため、クライアントはサブスクライブ後にキューへ配信される各メッセージを受信することが保証されます。

手順

1. ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** を開いて編集します。
2. **durable** の設定要素を、選択した **キュー** に追加し、**true** の値を割り当てます。

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

共有されていない永続的なサブスクリプションの設定

ブローカーは、一度に複数のコンシューマーがキューに接続できないように設定することができます。したがって、このように設定されたキューへのサブスクリプションは共有されません。

手順

1. ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** を開いて編集します。
2. 選択された各キューに **durable** 設定要素を追加します。

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

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

3. 選択した各 **queue** 要素に **max-consumers** 属性を追加し、値 **1** を割り当てます。

```
<configuration ...>
  <core ...>
    ...
    <address name="non.shared.durable.foo">
      <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>
```

非永続的なサブスクリプションキューの設定

非永続的なサブスクリプションは、通常、一時キューを作成および削除する関連するプロトコルマネージャーによって管理されます。

ただし、非永続サブスクリプションキューのように動作するキューを事前に作成する場合は、キューで **purge-on-no-consumers** 属性を使用できます。**purge-on-no-consumers** が **true** に設定されている場合、コンシューマーが接続されるまでキューはメッセージの受信を開始しません。さらに、最後のコンシューマーがキューから切断されると、キューはパージされます (つまり、メッセージが削除されます)。新しいコンシューマーがキューにアタッチされるまで、キューはそれ以上のメッセージを受信しません。

手順

1. ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** を開いて編集します。
2. 選択した **queue** 要素ごとに **purge-on-no-consumers** 属性を追加します。**purge-on-no-consumers** を **true** に設定します。

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

4.7. 完全修飾キュー名の指定

内部的には、ブローカーはアドレスのクライアント要求を特定のキューにマッピングします。ブロー

カーは、メッセージの送信先となるキューまたはメッセージを受信するキューをクライアントに代わって決定します。ただし、より高度なユースケースでは、クライアントが直接キューを指定する必要がある場合があります。このような場合、クライアントは、アドレス名とキュー名の両方を **::** で区切って指定して、完全修飾キュー名 (FQQN) を使用できます。

前提条件

- アドレスが、2 つ以上のキューで設定されている。以下の例では、アドレス **foo** に **q1** と **q2** の 2 つのキューがあります。

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

手順

- クライアントコードで、ブローカーから接続を要求する際に、アドレス名とキュー名の両方を使用します。以下の Java コードの例のように、2 つのコロン **::** を使用して名前を区切ることを忘れないでください。

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

4.8. シャード化されたキューの設定

部分的な順序付けのみが必要とされるキュー全体のメッセージの処理には、queue sharding を使用するのが一般的なパターンです。AMQ Broker では、単一の論理キューとして機能する **anycast** アドレスを作成してこれを実現できますが、多くの基礎となる物理キューでサポートされます。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** を開き、目的の名前の **アドレス** を追加します。以下の例では、**sharded** という名前の **アドレス** が設定に追加されます。

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

2. **anycast** ルーティングタイプを追加し、希望するシャードキューの数を入力します。以下の例では、キュー **q1**、**q2**、**q3** が **エニーキャスト** の宛先として追加されています。

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

上記の設定を使用すると、**sharded** に送信されるメッセージは **q1**、**q2**、および **q3** に均等に分散されます。クライアントは、完全修飾キュー名の使用 時に、特定の物理キューに直接接続して、その特定のキューにのみ送信されたメッセージを受け取ることができます。

特定のメッセージを特定のキューに結びつけるために、クライアントはメッセージごとにメッセージグループを指定できます。ブローカーは、グループ化されたメッセージを同じキューにルーティングし、1つのコンシューマーはそれをすべて処理します。詳細は、メッセージのグループ化 に関する章を参照してください。

4.9. LAST VALUE QUEUES の設定

last value queueとは、last value プロパティー名が同じメッセージでよりあたらいいものがキューに配置されると、キューからメッセージが破棄されるキューのタイプです。この動作により、last value キューは、同じプロパティーのメッセージは最後の値のみを保持します。

last value キューに対する単純なユースケースは、株式の株価しか監視するためのものです。特定の株式の最新値のみが関心があります。

注記

ブローカーは、last value プロパティーが設定されていない last value キューに送信されたメッセージを通常のメッセージとして配信します。last value プロパティーが設定された新規メッセージを着信すると、キューからこのようなメッセージは消去されません。

以下を使用して、last value キューを設定できます。

- **broker.xml** 設定ファイル
- JMS クライアント
- Core API
- アドレスワイルドカード

上記の各メソッドでは、Core API を除いて、last value キー (最後の値のプロパティーとも呼ばれます) のカスタム値を指定するか、この値を未設定のままにすることができるので、代わりにキーがデフォルト値に設定されます。last value キーのデフォルト値は **_AMQ_LVQ_NAME** です。

Core API の場合、定数の last value キー **Message.HDR_LAST_VALUE_NAME** を使用してlast value キューを作成し、last value メッセージを識別します。

4.9.1. broker.xml を使用した last value キューの設定

last value キーのカスタム値を指定するには、**broker.xml** 設定ファイルに次のような行を含めます。

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

また、デフォルトのラストバリューキー名 **_AMQ_LVQ_NAME** を使用するラストバリューキューを設定することもできます。これを行うには、last value キーの値を指定せずに、**broker.xml** 設定ファイルで last-value 設定パラメーターを true に設定します。この設定の例を以下に示します。

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

4.9.2. JMS クライアントを使用した last value キューの設定

JMS クライアントを使用してコンシューマーが使用する送り先を自動作成する場合、アドレス設定の一部として last value キーを指定できます。この場合、自動作成されたキューは last value キューです。この設定の例を以下に示します。

```
Queue queue = session.createQueue("my.destination.name?last-value-key=stock_ticker");
Topic topic = session.createTopic("my.destination.name?last-value-key=stock_ticker");
```

また、**_AMQ_LVQ_NAME** のデフォルトの last value キー名を使用する last value キューを設定することもできます。これを行うには、last-value キーの値を指定せずに、last-value 設定パラメーターを true に設定します。この設定の例を以下に示します。

```
Queue queue = session.createQueue("my.destination.name?last-value=true");
Topic topic = session.createTopic("my.destination.name?last-value=true");
```

4.9.3. Core API を使用した last value キューの設定

Core API を使用して last value キューを作成するには、キューの作成時に **lastvalue** パラメーターを **true** に設定します。これを行うには、**ClientSession** インターフェイスの **createQueue** メソッドを使用します。このメソッドを使用して last value キューを作成するための構文を以下に示します。

```
void createQueue(
SimpleString address,
RoutingType routingType,
SimpleString queueName,
SimpleString filter,
Boolean durable,
Boolean autoCreated,
int maxConsumers,
```

```
Boolean purgeOnNoConsumers,
Boolean exclusive,
Boolean lastValue
)
```

この場合、API は定数 **Message.HDR_LAST_VALUE_NAME** を使用して、キューに配信された last value のメッセージを識別します。

関連情報

- Core API の **createQueue** メソッドを使用して last value キューを作成する方法は、createQueue を参照してください。

4.9.4. アドレスワイルドカードを使用した last value キューの設定

broker.xml 設定ファイルでアドレスワイルドカードを使用して、一連のアドレスの last value キューを設定できます。この設定の例を以下に示します。

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

デフォルトでは、**default -last-value-key** の値は **null** です。

アドレスワイルドカードを使用する場合、デフォルトの last value キー名も使用できます。これを行うには、last value キーの値を指定せずに、**default-last-value-queue** パラメーターを **true** に設定します。

4.9.5. Last Value キューの動作例

この例は、last value キューの設定と動作を示しています。

broker.xml 設定ファイルで、以下のような設定が追加されているとします。

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

上記の設定では、last value キーが **stock_ticker** で、**price1** という last value キューが作成されます。

ここで、以下に示すように、クライアントが同じ last value プロパティー名を持つ 2 つのメッセージを **prices1** キューに送信するとします。

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

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

last value プロパティー名が同じ 2 つのメッセージが last value キュー **price1** に到着すると、最新のメッセージのみがキューに残り、最初のメッセージは消去されます。コマンドラインで以下の行を入力し、この動作を検証できます。

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

この例では、どちらのメッセージも last value プロパティーを使用しており、2 番目のメッセージが 1 番目のメッセージのあとにキューに入ったため、出力として 2 番目のメッセージが表示されます。

4.9.6. non-destructive コンシューマーの作成

コンシューマーがキューに接続すると、通常の動作として、そのコンシューマーに送信されたメッセージがコンシューマーによってのみ取得されます。コンシューマーがメッセージの受信を確認すると、ブローカーはキューからメッセージを削除します。

コンシューマーを設定する別の方法は、キュー **ブラウザー** として設定することです。この場合、キューは引き続きすべてのメッセージをコンシューマーに送信します。ただし、ブラウザーでは、他のコンシューマーがメッセージ受信しないようにすることはできません。さらに、ブラウザーがメッセージを消費すると、メッセージはキューに残ります。ブラウザーとして設定されたコンシューマーは、non-destructive コンシューマーのインスタンスです。

last value キューの場合、すべてのコンシューマーを non-destructive コンシューマーとして設定すると、すべての last value キーの最新の値が常にキューに保持されます。次のサブセクションの例は、last value キューを設定して、すべてのコンシューマーが non-destructive であることを確認する方法を示しています。

4.9.6.1. broker.xml を使用した non-destructive コンシューマーの設定

Broker.xml 設定ファイルで、last value キューを作成するには、 **non-destructive** パラメーターを **true** に設定します。この設定の例を以下に示します。

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

4.9.6.2. JMS クライアントを使用した non-destructive コンシューマーの作成

JMS クライアントを使用して、コンシューマーが使用する宛先を自動作成する場合、last value キューの動作と non-destructive コンシューマーをアドレス設定の一部として設定します。この設定の例を以下に示します。

```
Queue queue = session.createQueue("my.destination.name?last-value-key=stock_ticker&non-destructive=true");
Topic topic = session.createTopic("my.destination.name?last-value-key=stock_ticker&non-destructive=true");
```

4.9.6.3. アドレスワイルドカードを使用した non-destructive コンシューマーの設定

broker.xml 設定ファイルでアドレスワイルドカードを使用して、一連のアドレスの last value キューを設定できます。この設定の一部として、**default-non-destructive** を **true** に設定して、コンシューマーが non destructive であると指定できます。この設定の例を以下に示します。

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

デフォルトでは、**default-non-destructive** の値は **false** です。

4.10. キューに接続するコンシューマーの数の制限

max-consumers 属性を使用して、特定のキューに接続するコンシューマーの数を制限します。**max-consumers** フラグを **1** に設定して、排他的コンシューマーを作成します。デフォルト値は **-1** で、無制限のコンシューマーを設定します。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** を開き、必要な **queue** に **max-consumers** 属性を追加します。以下の例では、キュー **q3** に同時に接続できるコンシューマー数は **20** だけです。

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

2. (オプション) 以下の例のように、**max-consumers** を **1** に設定して排他的なコンシューマーを作成します。

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

3. (オプション) 以下の例のように、**max-consumers** を **-1** に設定することで、コンシューマーの数を無制限にします。

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

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

4.11. 専用キュー

専用キューは、すべてのメッセージを一度に 1 つのコンシューマーだけにルーティングする特別なキューです。この設定は、すべてのメッセージを同じコンシューマーによって順次処理する必要がある場合に便利です。キューに複数のコンシューマーがある場合は、1 つのコンシューマーのみがメッセージを受信します。このコンシューマーが切断されると、別のコンシューマーが選択されます。

4.11.1. 専用キューの設定

次のように、**broker.xml** 設定ファイルで専用キューを設定します。

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

アドレスパラメーターを使う JMS Client を使用する場合に、コンシューマーが使用する宛先の作成時に自動作成されます。

```
Queue queue = session.createQueue("my.destination.name?exclusive=true");
Topic topic = session.createTopic("my.destination.name?exclusive=true");
```

4.11.2. 専用キューのデフォルト設定

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

default-exclusive-queue のデフォルト値は **false** です。

4.12. リングキューの設定

通常、AMQ Broker のキューは first-in、first-out(FIFO) セマンティクスを使用します。これは、ブローカーがキューの末尾にメッセージを追加し、それらをヘッドから削除することを意味します。リングキューは、指定された数のメッセージを保持する特別なタイプのキューです。ブローカーは、新規メッセージが到達し、キューがすでに指定された数のメッセージを保持した場合にキューの先頭にメッセージを削除することで、固定キューのサイズを維持します。

たとえば、サイズ **3** で設定されたリングキューと、メッセージ **A**、**B**、**C**、および **D** を順番に送信するプロデューサーについて考えてみます。メッセージ **C** がキューに到達すると、キュー内のメッセージ数が設定済みのリングサイズに達したことになります。この時点で、メッセージ **A** はキューのヘッドにあり、メッセージ **C** はテイールにあります。メッセージ **D** がキューに到達すると、ブローカーはキューの末尾にメッセージを追加します。固定キューサイズを維持するために、ブローカーはキュー (メッセージ **A)** の先頭にメッセージを削除します。メッセージ **B** がキューの先頭にあるようになりました。

4.12.1. リングキューの設定

この手順では、リングキューを設定する方法を示します。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. 明示的なリングサイズが設定されていない一致するアドレスですべてのキューのデフォルトリングサイズを定義するには、address- **setting** 要素に **default-ring-size** の値を指定します。以下に例を示します。

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

   default-ring-size パラメーターは、自動作成されるキューのデフォルトサイズを定義する場合に特に便利です。**default-ring-size** のデフォルト値は **-1**(サイズ制限なし) です。

3. 特定のキューでリングサイズを定義するには、**ring-size** パラメーターを **queue** 要素に追加し、値を指定します。以下に例を示します。

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

注記

ブローカーの実行中に **ring-size** の値を更新できます。ブローカーは更新を動的に適用します。新しい **ring-size** の値が以前の値よりも小さい場合、ブローカーはキューのヘッドからメッセージをすぐに削除し、新しいサイズを強制しません。キューに送信された新しいメッセージは、古いメッセージの削除を強制的に実行しますが、キューはクライアントによる通常の消費を介して、自然に決まるまで、新しいサイズに到達しません。

4.12.2. リングキューの動作のトラブルシューティング

本セクションでは、リングキューの動作が設定とは異なる状況を説明します。

参照:

- 「再配信メッセージおよびロールバック」

- 「スケジュールされたメッセージ」
- 「ページ化されたメッセージ」

4.12.2.1. 再配信メッセージおよびロールバック

メッセージがコンシューマーに配信されている場合、メッセージは in-between 状態になります。ここでは、メッセージはキューには技術的には表示されなくなりますが、まだ確認されていません。メッセージは、コンシューマーによって確認されるまで、再配信の状態のままになります。再配信状態のままのメッセージは、リングキューから削除できません。

ブローカーは再配信メッセージを削除できないため、クライアントは許可するリングサイズの設定よりもリングキューにより多くのメッセージを送信できます。たとえば、以下のシナリオについて考えてみましょう。

1. プロデューサーは、**ring-size="3"** で設定したリングキューに 3 つのメッセージを送信します。
2. すべてのメッセージは即座にコンシューマーにディスパッチされます。
   この時点で、**messageCount= 3** および **deliveryCount= 3** です。
3. プロデューサーは別のメッセージをキューに送信します。その後、メッセージはコンシューマーにディスパッチされます。
   現在、**messageCount = 4** および **deliveryCount = 4** です。**4** のメッセージ数は、設定されたリングサイズ **3** を超えています。ただし、ブローカーはキューから再配信メッセージを削除できません。
4. 現在は、メッセージを承認せずにコンシューマーが閉じられているとします。
   この場合、4 つのインポイントで承認されていないメッセージはブローカーにキャンセルされ、消費元の逆順でキューの先頭に追加されます。このアクションにより、設定されたリングサイズにキューが追加されます。リングキューはヘッドでメッセージの末尾にあるメッセージを優先するため、キューは最後のメッセージがキューの先頭に追加されていたため、プロデューサーによって送信された最初のメッセージを破棄します。トランザクションまたはコアセッションのロールバックは同じように処理されます。

コアクライアントを直接使用する場合や、AMQ Core Protocol JMS クライアントを使用する場合は、**consumerWindowSize** パラメーターの値 (デフォルトでは 1024 * 1024 バイト) を減らすことで、配信中のメッセージ数を最小限に抑えることができます。

4.12.2.2. スケジュールされたメッセージ

スケジュールされたメッセージがキューに送信されると、メッセージは通常のメッセージのようにキューの末尾に即座に追加されません。代わりに、ブローカーはスケジュールされたメッセージを中間バッファーに保持し、メッセージの詳細に従ってキューのヘッドへ配信するようにメッセージをスケジュールします。ただし、スケジュールされたメッセージはキューのメッセージ数に反映されます。再配信メッセージの場合のように、この動作により、ブローカーがリングキューのサイズを強制していないことを確認できます。たとえば、以下のシナリオについて考えてみましょう。

1. 12:00 では、プロデューサーはメッセージ **A** を **ring-size="3"** で設定されたリングキューに送信します。メッセージは 12:05 に対してスケジュールされます。
   この時点で、**messageCount= 1** および **scheduledCount= 1** になります。
2. 12:01 では、プロデューサーはメッセージ **B** を同じリングキューに送信します。
   現在、**messageCount= 2** および **scheduledCount= 1**。
3. 12:02 では、プロデューサーはメッセージ **C** を同じリングキューに送信します。
   現在、**messageCount= 3** および **scheduledCount= 1**。

4. 12:03 時、プロデューサーはメッセージ **D** を同じリングキューに送信します。
現在、**messageCount**= **4** および **scheduledCount**= **1**。

キューのメッセージ数は **4** で、設定されたリングサイズから **3 よりも大きい** ようになりました。ただし、スケジュールされたメッセージはキューでは技術的ではありません (つまり、ブローカーにあり、キューに配置するようにスケジュールされています)。スケジュールされた配信時間 12:05 にすると、ブローカーはメッセージをキューのヘッドに配置します。ただし、リングキューが設定サイズをすでに到達しているため、スケジュールされたメッセージ **A** はすぐに削除されます。

4.12.2.3. ページ化されたメッセージ

スケジュールされたメッセージや配信のメッセージと同様に、ページングされたメッセージは、キューレベルではなく、実際にはアドレスレベルでページングされるため、ブローカーによって実施されるリングキューのサイズにはカウントされません。ページ化されたメッセージはキューでは技術的ではありませんが、キューの **messageCount** 値に反映されます。

リングキューを持つアドレスにはページングを使用しないことが推奨されます。代わりに、アドレス全体をメモリーに収めるようにします。または、 **address-full-policy** パラメーターを **DROP**、**BLOCK** または **FAIL** の値に設定します。

関連情報

- ブローカは、遡及アドレス指定を設定すると、リングキューの内部インスタンスを作成します。詳細は、「Retroactive アドレスの設定」 を参照してください。

4.13. 特定のルーティングタイプに接続するための接頭辞設定

通常、**anycast** と **multicast** の両方を使用しているアドレスでメッセージを受信した場合に、**anycast** のいずれかのキューがメッセージと、**multicast** のすべてのキューを受信します。ただし、クライアントは、アドレスに接続する際に特別な接頭辞を指定して、**anycast** と **multicast** キャストのどちらで接続するかを指定することができます。接頭辞は、**acceptor** の URL 内で **anycastPrefix** および **multicastPrefix** パラメーターを使用して指定されるカスタム値です。

Anycast 接頭辞の設定

- **BROKER_INSTANCE_DIR/etc/broker.xml** で、目的の **acceptor** の URL に **anycastPrefix** を追加します。以下の例では、**acceptor** は **anycastPrefix** に **anycast://** を使用するように設定されています。クライアントが anycast キューの 1 つだけにメッセージを送信する必要がある場合は、クライアントコードで **anycast://foo/** を指定できます。

```
<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>
```

Multicast 接頭辞の設定

- **BROKER_INSTANCE_DIR/etc/broker.xml** で、目的の **acceptor** の URL に **anycastPrefix** を追加します。以下の例では、**acceptor** は、multicastPrefix に **multicast ://** を使用するように設定されています。クライアントがアドレスの multicast キューのみにメッセージを送信する必要がある場合、クライアントコードは **multicast://foo/** を指定できます。

```
<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>
```

4.14. RETROACTIVE アドレスの設定

アドレスを **retroactive** として設定すると、アドレスに送信されたメッセージを保持することができます。これには、アドレスにキューがバインドされていない場合も含まれます。キューが後に作成され、アドレスにバインドされると、ブローカーはメッセージをこれらのキューにアクティブに配信します。アドレスが **retroactive** として設定されていない場合や、キューがバインドされていない場合、ブローカーはそのアドレスに送信されたメッセージを破棄します。

Retactive アドレスを設定する場合、ブローカーは **ring queue** と呼ばれるタイプのキューの内部インスタンスを作成します。リングキューは、指定された数のメッセージを保持する特別なタイプのキューです。キューが指定されたサイズに達すると、キューに到達する次のメッセージがキューから最も古いメッセージを強制的に強制的に実行します。Retroactive アドレスを設定する場合は、内部リングキューのサイズを間接的に指定します。デフォルトでは、内部キューは **multicast** ルーティングタイプを使用します。

Retactive アドレスによって使用される内部リングキューは、管理 API 経由で公開されます。メトリクスを検査し、キューが空など、その他の一般的な管理操作を実行できます。リングキューは、アドレスの全体的なメモリー使用量にも貢献し、メッセージのページングなどの動作に影響を与えます。

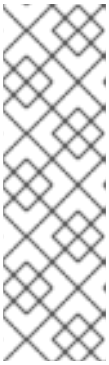
以下の手順では、アドレスを Retroactive として設定する方法を説明します。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルを開きます。
2. **address-setting** 要素で **retroactive-message-count** パラメーターの値を指定します。指定する値は、ブローカーによって保存されるメッセージの数を定義します。以下に例を示します。

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

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



注記

broker.xml 設定ファイルまたは管理 API のいずれかで、ブローカーの実行中に **retroactive-message-count** の値を更新できます。ただし、このパラメーターの値を**減らす** 場合は、Retroactive アドレスがリングキューで実装されるため、追加の手順が必要になります。**ring-size** パラメーターが減少するリングキューは、新しい **ring-size** 値を実現するためにキューからメッセージを自動的に削除しません。この動作は、意図しないメッセージ損失に対する安全なものです。この場合、管理 API を使用してリングキュー内のメッセージ数を手動で減らす必要があります。

関連情報

- リングキューの詳細は、「リングキューの設定」 を参照してください。

4.15. プロトコルマネージャーおよびアドレス

プロトコルマネージャーは、プロトコル固有の概念を AMQ Broker アドレスモデルの概念 (キューとルーティングの種類) にマップします。たとえば、クライアントがアドレス **/house/room1/lights** および **/house/room2/lights** を含む MQTT サブスクリプションパケットを送信すると、MQTT プロトコルマネージャーは、2 つのアドレスに **multicast** セマンティクスが必要であることを認識します。そのため、プロトコルマネージャーはまず、両方のアドレスで**multicast** が有効になっていることを確認します。そうでない場合は、動的に作成を試みます。成功すれば、プロトコル・マネージャーは、クライアントが要求した各サブスクリプションのために特別なサブスクリプションキューを作成します。

各プロトコルの動作は若干異なります。以下の表は、様々なタイプの **queue** へのサブスクライブフレームが要求されたときに、典型的に起こることを説明しています。

表4.2 プロトコルマネージャーのアクション

キューがこのタイプのものである場合	プロトコルマネージャーの通常のアクションは下記の通りです。
永続性のあるサブスクリプションキュー	適切なアドレスを検索し、**multicast** セマンティクスが有効化されていることを確認します。次に、クライアント ID で特別なサブスクリプションキューとアドレスをその名前として、**multicast** をルーティングタイプとして作成します。 特別な名前を使用すると、プロトコルマネージャーは、必要なクライアントのサブスクリプションキューを迅速に特定し、クライアントの接続を解除し、後で再接続できるようにします。 クライアントがキューのサブスクライブを解除すると、キューが削除されます。
一時サブスクリプションキュー	適切なアドレスを検索し、**multicast** セマンティクスが有効化されていることを確認します。そして、このアドレスの下に、ランダムな (UUID と呼ばれる) 名前のキューを、**multicast** のルーティングタイプで作成します。 クライアントがキューを切断すると、キューが削除されます。

キューがこのタイプのものである場合	プロトコルマネージャーの通常のアクションは下記の通りです。
Point-to-Point キュー	適切なアドレスを探し、**anycast** ルーティングタイプが有効になっていることを確認します。存在する場合は、アドレスと同じ名前のキューを見つけることを目的としています。存在しない場合は、利用可能な最初のキューを探します。 キューは存在しない場合は自動的に作成されます (これにより、自動作成が有効になっています)。キューコンシューマーはこのキューにバインドされます。 キューが自動作成されると、コンシューマーがなく、メッセージがなければ自動的に削除されます。

4.16. アドバイザリーメッセージの無効化

デフォルトでは、AMQ Broker は、OpenWire クライアントがブローカーに接続されているときにアドレスおよびキューに関するアドバイザリーメッセージを作成します。アドバイザリーメッセージは、ブローカーによって作成された内部管理アドレスに送信されます。これらのアドレスは、ユーザーがデプロイされたアドレスおよびキューと同じ表示内の AMQ コンソールに表示されます。有用な情報が提供されますが、ブローカーによって多数の宛先を管理する場合に、アドバイザリーメッセージにより不要な結果が生じる可能性があります。たとえば、メッセージはメモリー使用量やステンケーション接続リソースを増やす可能性があります。また、アドバイザリーメッセージを送信するために作成されたすべてのアドレスを表示しようとすると、AMQ コンソールが明確になる可能性があります。このような状況を回避するには、**supportAdvisory** パラメーターと **suppressInternalManagementObjects** パラメーターを使用して、ブローカー側でのアドバイスメッセージの動作を管理します。

- **supportAdvisory**: このオプションを **true** に設定してアドバイザリーメッセージの作成を有効にするか、**false** に設定して無効にします。デフォルト値は **true** です。
- **suppressInternalManagementObjects**: このオプションを **true** に設定して、アドバイスメッセージを JMX レジストリーや AMQ コンソールなどの管理サービスに公開するか、**false** に設定して公開しないようにします。デフォルト値は **true** です。

BROKER_INSTANCE_DIR/etc/broker.xml 設定ファイルを編集してこれらのパラメーターを使用し、URL を使用して OpenWire アクセプターのパラメーターを設定します。以下に例を示します。

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

4.17. アドレスの設定

AMQ Broker には、メッセージの配信方法と時期、試行回数、およびメッセージの有効期限を制御する設定可能なオプションがいくつかあります。これらの設定オプションはすべて、**<address-setting>** 設定要素内に存在します。ワイルドカード構文を使用して、AMQ Broker に単一の **<address-setting>** を複数の宛先に適用させることができます。

AMQ Broker ワイルドカード構文

AMQ Broker は、セキュリティー設定、アドレス設定、およびコンシューマーの作成時に、ワイルドカードを表すために特定の構文を使用します。

- ワイルドカード式には、「.」 文字で区切られた単語が含まれます。
- 特殊文字 「#」 と 「*」 にも特別な意味があり、単語の代わりになることがあります。

- 文字 「#」 は、0 個以上の単語のシーケンスとマッチします。式の最後に使用します。
- 文字 「*」 は 1 つの単語とマッチします。式内のどこかにこの変数を使用します。

マッチングは文字による文字は実行されませんが、各区切り文字境界では文字ごとに一致します。たとえば、名前に **my** を使用してキューを照合しようとする **address-setting** は、**myqueue** という名前のキューとは一致しません。

キューに複数の **address-setting** がマッチする場合、ブローカーは、ベースラインとして最も合致の少ない設定を使用して設定をオーバーレイします。リテラル式はワイルドカードよりも限定的であり、***** は **#** よりも限定的です。たとえば、**my.queue** と **my.*** の両方がキュー **my.queue** に一致します。この場合、ワイルドカード式はリテラルよりも具体的なため、ブローカーは最初に **my.*** にある設定を適用します。次に、ブローカーは **my.queue address-setting** の設定をオーバーレイし、**my.*** と共有される設定が上書きされます。以下の設定では、キュー **my.queue** の **max-delivery-attempts** が **3** に設定され、**last-value-queue** が **false** に設定されます。

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

以下の表の例は、ワイルドカードを使用してアドレスセットを照合する方法を示しています。

表4.3 ワイルドカードの例

例	説明
#	**broker.xml** で使用されるデフォルトの **address-setting**。すべてのアドレスに一致します。このキャッチオールを適用するか、必要に応じて各アドレスまたはアドレスグループに新しい **address-setting** を追加することができます。
news.europe.#	**news.europe**、**news.europe.sport**、**news.europe.politics.fr** は一致するが、**news.usa** や **europe** は一致しない。
news.*	**news.europe** と **news.usa** は一致するが、**news.europe.sport** は一致しない。
news.*.sport	**news.europe.sport** と **news.usa.sport** は一致するが、**news.europe.fr.sport** は一致しない。

ワイルドカード構文の設定

設定を **broker.xml** に追加することで、ワイルドカードアドレスに使用される構文をカスタマイズできます。

手順

- 以下の例のように、設定に **<wildcard-addresses>** セクションを追加して、**broker.xml** を編集します。

```
<configuration>
  <core>
    ...
    <wildcard-addresses> 1
      <enabled>true</enabled> 2
      <delimiter>,</delimiter> 3
      <any-words>@</any-words> 4
      <single-word>$</single-word> 5
    </wildcard-addresses>
    ...
  </core>
</configuration>
```

1. **コア** 設定要素の下に **ワイルドカードアドレス** を追加します。
2. ブローカーにカスタム設定を使用するように指示するには、**enabled** を **true** に設定します。
3. デフォルトの **.** の代わりに **delimiter** として使用するカスタム文字を指定します 。
4. **any-words** の値として提供された文字は、ゼロ以上の単語のシーケンスに一致し、デフォルトの **#** を置き換えます。式の最後にこの文字を使用します。
5. **single-word** の値として提供された文字は、単一の単語の照合に使用され、デフォルトの * を置き換えます。この文字は式内の任意の場所を使用します。

4.18. キューとアドレスの自動作成と削除

アドレスおよびキューを自動的に作成し、使用されなくなったら削除するように AMQ Broker を設定できます。これにより、クライアントが接続する前に各アドレスを事前に設定する必要がなくなります。

キューとアドレスの自動作成と削除は、**address-setting** ごとに設定されます。設定は、**address-setting** に一致する任意のアドレスまたはキューに適用されます。ワイルドカード構文を使用してアドレスとキューを **address-setting** に一致させる方法は、アドレスの設定 を参照してください。

重要

キューとアドレスの自動作成を無効にすると問題が発生する可能性があるため、お勧めしません。AMQ Broker は、**activemq.#** パターンを使用して、アドレスとキューを自動作成できる必要があります。例については アドレス設定 の例を参照してください。

以下の表は、**address -setting** を設定してキューとアドレス を自動的に作成し、削除する際に利用できる設定要素の一覧です。

address-setting を下記に設定したい場合	この設定を追加してください
クライアントが存在しないアドレスにマッピングされたキューからメッセージを消費するか、または消費しようとするとアドレスを作成します。	**auto-create-addresses**
クライアントがキューからメッセージを消費するか、または消費しようとするとキューを作成します。	**auto-create-queues**

address-setting を下記に設定したい場合	この設定を追加してください
キューがなくなったら、自動的に作成されたアドレスを削除します。	**auto-delete-addresses**
キューのコンシューマー 0 と 0 のメッセージがある場合に、自動作成されたキューを削除します。	**auto-delete-queues**
クライアントが指定しない場合は、特定のルーティングタイプを使用します。	**default-address-routing-type**

手順

BROKER_INSTANCE_DIR/etc/broker.xml ファイルを編集し、自動作成および削除の **address-setting** を設定します。以下の例では、前述の表に記載されているすべての設定要素を使用しています。

```
<configuration ...>
 <core ...>
  ...
  <address-settings>
   <address-setting match="activemq.#"> 1
    <auto-create-addresses>true</auto-create-addresses> 2
    <auto-delete-addresses>true</auto-delete-addresses> 3
    <auto-create-queues>true</auto-create-queues> 4
    <auto-delete-queues>true</auto-delete-queues> 5
    <default-address-routing-type>ANYCAST</default-address-routing-type> 6
   </address-setting>
  </address-settings>
  ...
 </core>
</configuration>
```

1. この **address-setting** に含まれる設定は、ワイルドカード **activemq.#** に一致するすべてのアドレスまたはキューに適用されます。ワイルドカード構文の使用の詳細は、AMQ Broker ワイルドカード構文 を参照してください。
2. ブローカは、クライアントが要求したときに存在しないアドレスを作成します。
3. 自動作成されたアドレスは、キューに関連付けられなくなった時点で削除されます。
4. ブローカは、クライアントが要求したときに存在しないキューを作成します。
5. 自動作成されたキューは、コンシューマーまたはメッセージがないと削除されます。
6. クライアントが接続時にルーティングタイプを指定しない場合、ブローカーはアドレスにメッセージを配信する際に **ANYCAST** を使用します。デフォルト値は **MULTICAST** です。ルーティングタイプの詳細は この章の冒頭 を参照してください。

4.19. 期限切れメッセージの期限切れアドレスへの移動

最後の値キュー以外のキューでは、非破壊的なコンシューマーしかない場合、ブローカーはキューから

メッセージを削除できず、キューのサイズが徐々に増加します。キューサイズで制約のない増加を防ぐには、メッセージの有効期限が切れるタイミングを設定し、ブローカーが期限切れのメッセージを移動するアドレスを指定します。

4.19.1. メッセージ有効期限の設定

以下の手順では、メッセージの有効期限を設定する方法を説明します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. **core** 要素で、**message-expiry-scan-period** を設定して、ブローカーが期限切れのメッセージをスキャンする頻度を指定します。

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

前述の設定に基づいて、ブローカーは 1000 ミリ秒ごと (1 秒ごと) に期限切れのメッセージのキューをスキャンします。

3. 一致するアドレスまたは一連のアドレスの **address-setting** 要素で、有効期限切れのアドレスを指定します。また、メッセージの有効期限を設定します。以下に例を示します。

```
<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

一致するアドレスや一連のアドレスの有効期限。前述の例では、ブローカーは **stocks** アドレスに対する期限切れメッセージを **ExpiryAddress** と呼ばれる期限切れアドレスに送信します。

expiry-delay

デフォルトの有効期限を使用するメッセージにブローカーが適用される有効期限 (ミリ秒単位)。デフォルトでは、メッセージの有効期限は **0** で、有効期限がないことを意味します。デフォルト以上の有効期限が設定されているメッセージに対しては、**expiry-delay** は効果がありません。
例えば、先ほどの例のように、アドレスの **expiry-delay** を **10** に設定したとします。デフォルトの有効期限が **0** のメッセージがこのアドレスのキューに到着した場合、ブローカーはメッセージの有効期限を **0** から **10** に変更します。しかし、有効期限を **20** に設定している

別のメッセージが到着した場合、その有効期限は変更されません。expiry-delay を **-1** に設定した場合、この機能は無効になります。デフォルトでは、**expiry-delay** は **-1** に設定されます。

4. また、**expiry-delay** に値を指定する代わりに、expiry delay の最小値と最大値を指定することもできます。以下に例を示します。

```
<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

ブローカーがメッセージに適用される最小有効期限 (ミリ秒単位)。

max-expiry-delay

ブローカーがメッセージに適用される最大有効期限 (ミリ秒単位)。
ブローカーは、以下のように **min-expiry-delay** と **max-expiry-delay** の値を適用します。

- デフォルトの有効期限が **0** のメッセージに対して、ブローカーは有効期限を指定された値の **max-expiry-delay** に設定します。**max-expiry-delay** の値を指定していない場合、ブローカーは指定された **min-expiry-delay** の値に満了時間を設定します。**min-expiry-delay** の値を指定しない場合、ブローカーはメッセージの有効期限を変更しません。
- **max-expiry-delay** の値を上回る有効期限のあるメッセージの場合、ブローカーは有効期限を **max-expiry-delay** の指定値に設定します。
- 有効期限が **min-expiry-delay** の値以下のメッセージに対して、ブローカーは有効期限を指定された **min-expiry-delay** の値に設定します。
- **min-expiry-delay** および **max-expiry-delay** の値の間に有効期限のあるメッセージの場合、ブローカーはメッセージの有効期限を変更しません。
- **expiry-delay** の値 (すなわちデフォルト値の **-1** 以外) を指定すると、**min-expiry-delay** および **max-expiry-delay** に指定する値が上書きされます。
- **min-expiry-delay** と **max-expiry-delay** の両方のデフォルト値は **-1** (つまり無効) です。

5. 設定ファイルの **address** 要素で、以前に **expiry-address** に指定したアドレスを設定します。このアドレスにキューを定義します。以下に例を示します。

```
<addresses>
  ...
```

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

前述の設定例では、期限切れキュー **ExpiryQueue** と期限切れアドレス **ExpiryAddress** を関連付けています。

4.19.2. 期限切れリソースの自動作成

一般的なユースケースとして、期限切れのメッセージを元のアドレスに従って分離することです。例えば、**stocks** というアドレスからの期限切れメッセージを **EXP.stocks** という期限切れキューにルーティングすることを選択したとします。同様に、**orders** というアドレスからの期限切れメッセージを **EXP.orders** という期限切れキューにルーティングする場合もあるでしょう。

このタイプのルーティングパターンにより、期限切れのメッセージを簡単に追跡、検査、および管理できるようになります。ただし、このようなパターンは、主に自動作成されたアドレスおよびキューを使用する環境に実装するのが困難です。このような環境では、管理者は、期限切れのメッセージを保持するためにアドレスおよびキューを手動で作成するために必要な追加の作業を必要としません。

解決策として、特定のアドレスまたは一連のアドレスの期限切れのメッセージを処理するように、リソース (すなわちアドレスとキュー) を自動的に作成するようにブローカーを設定できます。以下の手順はその一例です。

前提条件

- 指定したアドレスまたは一連のアドレスに対して、すでに有効期限付きのアドレスを設定しています。詳細は、「メッセージ有効期限の設定」 を参照してください。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. 以前、設定ファイルに追加した **<address-setting>** 要素を探して、一致するアドレスまたは一連のアドレスの有効期限を定義します。以下に例を示します。

```
<configuration ...>

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

3. **<address-setting>** 要素に、期限切れリソース (すなわちアドレスやキュー) を自動的に作成することや、これらのリソースにどのような名前を付けるかをブローカに指示する設定項目を追加します。以下に例を示します。

```
<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

期限切れのメッセージを受信するため、ブローカーが期限切れアドレスとキューを自動的に作成するかどうかを指定します。デフォルト値は **false** です。
パラメーターの値が **true** に設定されている場合、ブローカーは期限切れアドレスと関連付けられた期限切れキューを定義する **<address>** 要素を自動的に作成します。自動的に作成された **<address>** 要素の名前の値は、**<expiry-address>** に指定された名前の値と一致します。

自動作成された期限切れキューには、**multicast** ルーティングタイプがあります。デフォルトでは、ブローカは失効したメッセージが最初に送信されたアドレス (例えば、**stocks**) と一致するように失効キューに名前を付けています。

ブローカーは、**_AMQ_ORIG_ADDRESS** プロパティーを使用する期限切れキューのフィルターも定義します。このフィルターは、期限切れキューが対応する元のアドレスに送信されるメッセージのみを受け取るようになります。

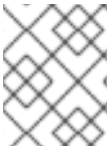
expiry-queue-prefix

ブローカーにより、自動作成された期限切れキューの名前に適用される接頭辞。デフォルト値は **EXP** です。
接頭辞の値を定義した場合、またはデフォルト値を維持した場合、期限切れキューの名前は、接頭辞と元のアドレスを連結したものになります (例: **EXP.stocks**)。

expiry-queue-suffix

ブローカーが自動作成された期限切れキューの名前に適用される接尾辞。デフォルト値は定義されていません (つまり、ブローカーは接尾辞を適用しません)。

キュー名自体を使用して (AMQ Broker Core Protocol JMS クライアントを使用する場合など)、または完全修飾キュー名 (別の JMS クライアントを使用する場合など) を使用して、期限切れキューに直接アクセスできます。



注記

期限切れアドレスとキューは自動的に作成され、自動作成されたアドレスやキューの削除に関連するアドレス設定もこれらの期限切れリソースに適用されます。

関連情報

- 自動作成されたアドレスやキューの自動削除を設定するためのアドレス設定については、「キューとアドレスの自動作成と削除」 を参照してください。

4.20. 配信されていないメッセージを DEAD LETTER アドレスへ移行

クライアントにメッセージの配信に失敗した場合は、ブローカーがメッセージの配信を継続しようとしない場合があります。無限配信を試行するのを防ぐために、**dead letter address** と 1 つ以上の **dead letter queues**を定義できます。指定の数の配信試行後、ブローカーは元のキューから未配信メッセージを削除し、そのメッセージを設定済みの dead letter アドレスに送信します。システム管理者は、デッド文字キューから未配信メッセージを後で消費してメッセージを検査できます。

指定のキューに dead letter アドレスを設定しない場合、ブローカーは指定された数の配信試行後にキューから未配信メッセージを完全に削除します。

dead letter キューから消費される配信されていないメッセージには、以下のプロパティーがあります。

_AMQ_ORIG_ADDRESS
: メッセージの元のアドレスを指定する string プロパティー

_AMQ_ORIG_QUEUE
: メッセージの元のキューを指定する string プロパティー

4.20.1. dead letter アドレスの設定

以下の手順は、dead letter アドレスと関連する dead letter キューを設定する方法を説明します。

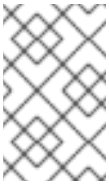
手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. キュー名と一致する **<address-setting>** 要素で、dead letter のアドレス名と配信試行回数の最大値を設定します。以下に例を示します。

```
<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 ...>
```

dead-letter-address

dead letter アドレスの名前。この例では、ブローカーは未配信メッセージをキュー **exampleQueue** から dead letter address(**DLA**) に移動します。



注記

<address-setting> 要素の **match** 属性には、ワイルドカード式を指定できます。ワイルドカード式を使用すると、**<address-setting>** 要素に設定された dead letter 設定を一致する**一連のアドレス**に関連付ける場合に便利です。

max-delivery-attempts

配信不能メッセージを設定済みの dead letter アドレスに移動するまでの、ブローカーによる配信試行の最大数。この例では、ブローカーは配信に失敗した 3 回失敗した後、未配信メッセージを dead letter address に移動します。デフォルト値は **10** です。ブローカーが再配信の試行を無限にするには、**-1** の値を指定します。

3. **addresses** セクションに、dead letter アドレス (**DLA**) の **address** 要素を追加します。dead letter キューを dead letter アドレスに関連付けるには、**queue** の名前を指定します。以下に例を示します。

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

上記の設定では、**DLQ** という名前の dead letter キューを dead letter アドレス (**DLA**) に関連付けます。

関連情報

- アドレス設定でワイルドカードを使用する方法の詳細は、AMQ Broker ワイルドカード構文 を参照してください。

4.20.2. dead letter キューの自動作成

一般的なユースケースは、元のアドレスに従って配信されていないメッセージを分離することです。たとえば、配信不能メッセージを **stocks** というアドレスから、**DLQ.stocks** という dead letter キューが関連付けられている **DLA.stocks** という dead letter キューにルーティングすることを選択できます。同様に、**orders** というアドレスから配信不能メッセージを **DLA.orders** という dead letter アドレスにルーティングすることもできます。

このタイプのルーティングパターンにより、未配信のメッセージを追跡、検査、および管理が容易になります。ただし、このようなパターンは、主に自動作成されたアドレスおよびキューを使用する環境に実装するのが困難です。このタイプの環境のシステム管理者は、アドレスおよびキューを手動で作成するのに必要な追加の作業を必要としない可能性があります。

解決策として、以下に示すように、未配信メッセージを処理するよう addressees および キューを自動的に作成するようにブローカーを設定できます。

前提条件

- キューまたはキューのセットに dead letter アドレスがすでに設定されている。詳細は、「dead letter アドレスの設定」 を参照してください。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. 以前に追加した **<address-setting>** 要素を見つけて、一致するキューまたは一連のキューの dead letter アドレスを定義します。以下に例を示します。

```
<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. **<address-setting>** 要素に、dead letter リソース (つまり、アドレスとキュー) を自動的に作成するようブローカーに指示する設定項目と、これらのリソースの命名方法を追加します。以下に例を示します。

```
<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

ブローカーが dead letter アドレスおよびキューを自動的に作成し、配信不能メッセージを受信するかどうかを指定します。デフォルト値は **false** です。
auto-create-dead-letter-resources が **true** に設定されている場合、ブローカーは dead letter アドレスと関連する dead letter キューを定義する **<address>** 要素を自動的に作成します。自動作成された **<address>** 要素の名前は、**<dead-letter-address>** に指定する name の値と一致します。

ブローカーが自動作成された **<address>** 要素で定義する dead letter キューには、**multicast** ルーティングタイプがあります。デフォルトでは、ブローカーが dead letter キューに名前を付け、配信不能メッセージの元のアドレス (**stocks** など) を照合します。

ブローカーは、**_AMQ_ORIG_ADDRESS** プロパティーを使用する dead letter キューのフィルターも定義します。このフィルターは、dead letter キューが対応する元のアドレスに送信されるメッセージのみを受け取るようになります。

dead-letter-queue-prefix

ブローカーにより、自動作成された dead letter キューの名前に適用される接頭辞。デフォルト値は **DLQ** です。
接頭辞値を定義するか、デフォルト値をそのまま使用すると、配信不能キューの名前は接頭辞と元のアドレスを連結したものになります (例: **DLQ.stocks**)。

dead-letter-queue-suffix

ブローカーにより、自動作成された dead letter キューに適用される接尾辞。デフォルト値は定義されていません (つまり、ブローカーは接尾辞を適用しません)。

4.21. 期限切れまたは配信不能の AMQP メッセージに対するアノテーションおよびプロパティー

ブローカーは、期限切れの AMQP メッセージまたは未配信の AMQP メッセージを、設定した期限切れまたは dead letter キューに移動する前に、アノテーションおよびプロパティーをメッセージに適用します。クライアントは、これらのプロパティーまたはアノテーションに基づいてフィルターを作成し、有効期限または dad letter キューから消費する特定のメッセージを選択できます。

注記

ブローカーが適用されるプロパティーは **内部** プロパティーです。これらのプロパティーは、通常の使用のためにクライアントに公開されませんが、フィルターのクライアントで指定**できます**。

以下の表は、ブローカーが期限切れの AMQP メッセージまたは配信されないアノテーションおよび内部プロパティーを表しています。

アノテーション名	内部プロパティー名	説明
x-opt-ORIG-MESSAGE-ID	_AMQ_ORIG_MESSAGE_ID	メッセージが期限切れまたは dead letter キューに移動される前の、元のメッセージ ID。
x-opt-ACTUAL-EXPIRY	_AMQ_ACTUAL_EXPIRY	最後のエポックの開始からの経過時間 (ミリ秒単位) に指定されたメッセージの有効期限。
x-opt-ORIG-QUEUE	_AMQ_ORIG_QUEUE	期限切れまたは未配信メッセージの元のキュー名。
x-opt-ORIG-ADDRESS	_AMQ_ORIG_ADDRESS	期限切れまたは未配信メッセージの元のアドレス名。

関連情報

- AMQP クライアントがアノテーションに基づいて AMQP メッセージをフィルターするように設定する例は、「アノテーションを基にした AMQP メッセージのフィルター」 を参照してください。

第5章 ユーザーとロール

このブローカーは、該当のアドレスに基づいてキューにセキュリティーを適用するための柔軟なロールベースのセキュリティーモデルをサポートします。キューは、1対1(ポイントツーポイントスタイルのメッセージングの場合) または多対1(パブリッシュ/サブスクライブスタイルのメッセージングの場合) のアドレスにバインドされる点を理解することが重要です。メッセージがアドレスに送信されると、サーバーはそのアドレスにバインドされたキューのセットを検索し、メッセージをそのキューのセットにルーティングします。

デフォルトの設定 (**PropertiesLoginModule** を使用) では、ユーザーと割り当てられたロールは 3 つの設定ファイルで定義されます。

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

これらの各ファイルについては、次のセクションで詳しく説明します。

コマンドラインインターフェイスを使用すると、インタラクティブなプロセスを介してこれらのファイルにユーザーとロールを追加できます。

注記

artemis-users.properties ファイルには、セキュリティーを確保するために、ハッシュ化されたパスワードを含めることができます。

5.1. ゲストアクセスの有効化

ログイン認証情報のないユーザー、または認証情報の検証に失敗したユーザーには、**guest** アカウントを使用してブローカーへの制限付きアクセスを許可できます。

コマンドラインの切り替えを使用して **--allow-anonymous** (**--require-login** の逆)、ゲストアクセスを有効にし、ブローカーインスタンスを作成できます。

ゲストアカウントは、**login.config** ファイルで設定されます。

手順

1. **login.config** ファイルで、ゲストアカウントの名前とロールを次のように定義します。

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule required
     org.apache.activemq.jaas.guest.user="guest" 1
     org.apache.activemq.jaas.guest.role="guest"; 2
};
```

1 匿名ユーザーに割り当てるユーザー名を定義します。

2 匿名ユーザーに割り当てるロールを定義します。

ゲストログインモジュールを使用すると、認証情報を持たないユーザー (設定方法によっては、認証情報が無効なユーザーも) がブローカーにアクセスできます。**org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule** により実装されます。

ゲストログインモジュールは、プロパティーログインモジュールなどの別のログインモジュールと組み合わせて使用するのが一般的です。そのユースケースの詳細は、「複数のログインモジュールの使用」セクションを参照してください。

5.2. ユーザーの追加

基本的なユーザー名とパスワードの検証が必要な場合は、**Properties** ログインモジュールを使用して定義します。このログインモジュールは、一連のローカルプロパティーファイルに対してユーザーの認証情報をチェックします。

ユーザーとそれに対応するパスワードは、**BROKER_INSTANCE_DIR/etc/artemis-users.properties** ファイルにリストされています。利用可能なロールとそれらのロールが割り当てられているユーザーは、**BROKER_INSTANCE_DIR/etc/artemis-roles.properties** ファイルで定義されています。

これらのファイルは両方とも、**BROKER_INSTANCE_DIR/etc/login.config** ファイルで参照されます。

JAAS の詳細は、Java ベンダーのドキュメントを参照してください。たとえば、Oracle には、**login.config** の設定に関するチュートリアルがあります。

例5.1 login.config

```
activemq { (1)
   org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule required (2) (3)
      org.apache.activemq.jaas.properties.user="artemis-users.properties"; (4)
      org.apache.activemq.jaas.properties.role="artemis-roles.properties";
};
```

(1) 設定のエイリアス。このセクションで使用されるエイリアスは **activemq** です。お使いの環境で別のものに置き換えてください。

(2) 実装クラス (**org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule**)。

(3) **LoginModule の成功が required**、**requisite**、**sufficient**、または **optional** のいずれであるかを示すフラグ。

(4) ログインモジュールの実装固有の設定オプションのリスト。

以下は、前の例にリストされている各成功状態の説明です。

Required

LoginModule は成功する必要があり、認証は成功または失敗に関係なく、LoginModule リストを下に進みます。

Requisite

LoginModule が正常に実行される必要があります。失敗すると、すぐに制御がアプリケーションに返され、認証は LoginModule リストの下に進みません。

Sufficient

LoginModule は正常に実行される必要はありません。成功した場合には、制御がアプリケーションに返り、認証はこれ以上続行しません。失敗した場合、認証の試行は **LoginModule** リストを下に進みます。

Optional

LoginModule は正常に実行される必要はありません。認証は、成功または失敗に関係なく、**LoginModules** リストの下方に進みます。

これらのフラグと認証プロセスの詳細は、Oracle のドキュメント を参照してください。

例5.2 artemis-users.properties

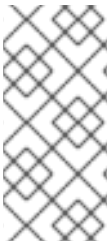
```
user1=secret (1)
user2=swordfish (2)
user3=myPassword (3)
```

1. User1 のパスワードは secret です。
2. User2 のパスワードは swordfish です。
3. User3 のパスワードは myPassword です。

例5.3 artemis-roles.properties

```
admin=user1,user2 (1)
developer=user3 (2)
```

1. user1 と user2 は admin ロールに属しています。
2. User3 は developer ロールに属しています。



注記

必要に応じて、以下に示すように、セキュリティードメインエイリアス (この例では activemq) を **bootstrap.xml** ファイルに追加します。

```
<jaas-security domain="activemq"/>
```

5.3. パーミッションの設定

権限は、**broker.xml** の **<security-setting>** 要素を介してアドレスに基づいてキューに対して定義されます。**<security-setting>** の複数のインスタンスを **<security-settings>** で定義できます。アドレスの完全一致を使用することができます。または、ワイルドカード文字 **#** と ***** を使ったワイルドカード一致を使用することも可能です。

アドレスに一致する一連のキューに、別の権限を指定できます。これらの権限は次のとおりです。

表5.1 パーミッション

ユーザーによるアクセス許可	以下のパラメーター...
アドレスの作成	**createAddress**
アドレスの削除	**deleteAddress**
一致するアドレスの永続キューの作成	**createDurableQueue**
一致するアドレスの永続キューの削除	**deleteDurableQueue**
一致するアドレスの非永続キューの作成	**createNonDurableQueue**
一致するアドレスの非永続キューの削除	**deleteNonDurableQueue**
一致するアドレスへのメッセージの送信	**send**
一致するアドレスにバインドされたキューからのメッセージの消費	**consume**
管理アドレスに管理メッセージを送信して管理操作を呼び出します。	**manage**
一致するアドレスにバインドされるキューの参照	**browse**

パーミッションごとに、そのパーミッションが付与されているロールのリストが指定されています。ユーザーにこれらのロールのいずれかが割り当てられている場合、そのアドレスのセットに対するそのアクセス許可が付与されます。

5.3.1. 単一アドレス向けメッセージ実稼働の設定

単一のアドレスの送信許可を定義するには、以下に示す例のような設定が使用されます。

```
<security-settings>
   <security-setting match="queue1"> 1
      <permission type="send" roles="producer"/> 2
   </security-setting>
</security-settings>
```

1 このキューに送信されたメッセージは、指定された権限を取得します。

2 指定されたキュー内のメッセージに適用される権限。

上記の例では、**producer** ロールのメンバーは、**queue1** に対する **send** パーミッションを持っています。

5.3.2. 単一アドレスのメッセージ消費の設定

単一のアドレスの消費パーミッションを定義するには、以下に示す例のような設定が使用されます。

■

```
<security-settings>
   <security-setting match="queue1"> 1
      <permission type="consume" roles="consumer"/> 2
   </security-setting>
</security-settings>
```

1 このキューに送信されたメッセージは、指定された権限を取得します。

2 指定されたキュー内のメッセージに適用される権限。

上記の例では、**consumer** ロールのメンバーは **queue1** に対する **consume** 権限が割り当てられます。

5.3.3. すべてのアドレスでの完全なアクセスの設定

アドレスとキューへの完全なアクセスを許可するには、以下に示す例のような設定を使用します。

```
<security-settings>
   <security-setting match="#"> 1
      <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>
```

1 すべてのキューに適用するワイルドカード設定。

上記の設定では、すべての権限がすべてのキューの **guest** ロールのメンバーに付与されます。これは、すべてのユーザーに **guest** ロールを割り当てるように匿名認証を設定する開発のシナリオで役に立ちます。

より複雑な使用例については アドレスの複数の権限の設定 を参照してください。

5.3.4. ユーザーでのキューの設定

キューには、自動作成時に接続クライアントのユーザー名が割り当てられます。これは、キューでメタデータとして公開されます。JMX およびコンソールで公開されます。キューには、自動作成時に接続クライアントのユーザー名が割り当てられます。これは、キューでメタデータとして公開されます。JMX およびコンソールで公開されます

broker.xml で事前定義されたキューでユーザーを設定できます。

キューのユーザーを定義するには、以下に示す例のような設定を使用します。

```
<address name="ExempleQueue">
   <anycast>
      <queue name="ExampleQueue" user="admin" />
```

```
    </anycast>
  </address>
```

注記

キューでユーザーを設定しても、そのキューのセキュリティーセマンティクスは変更されず、そのキューのメタデータにのみ使用されます。

5.4. ロールベースのアクセス制御の設定

ロールベースのアクセス制御 (RBAC) を使用して、MBean の属性とメソッドへのアクセスを制限します。RBAC を使用すると、管理者は、ロールに基づいて、Web コンソール、管理インターフェイス、コアメッセージなどのすべてのユーザーに適切なレベルのアクセスを許可できます。RBAC は、**BROKER_INSTANCE_DIR/etc/management.xml** 設定ファイルの **authorization** 要素を使用して設定されます。authorization 要素内で、**Whitelist**、**default-access**、および **role-access** サブ要素を設定できます。

前提条件

RBAC を設定するには、最初 にロールを設定してユーザーを追加する 必要があります。

5.4.1. 認証をバイパスするためのホワイトリスト要素の設定

whitelist は、ユーザー認証を必要としない事前承認済みのドメインまたは MBean のセットです。認証を省略する必要のあるドメインの一覧または MBean の一覧、またはその両方を指定できます。たとえば、AMQ 管理コンソールの実行に必要な MBean の **whitelist** 要素を使用できます。

手順

1. **BROKER_INSTANCE_DIR/etc/management.xml** 設定ファイルを開きます。
2. **whitelist** 要素を検索し、設定を編集します。

```
<whitelist>
  <entry domain="hawtio"/> 1
</whitelist>
```

1 このドメインの MBean は認証をバイパスします。

この例では、ドメインが **hawtio** の MBean が、認証なしでアクセスできます。MBean プロパティーに一致させるために **<entry domain="hawtio" key="type=*"/>** 形式のワイルドカードエントリーを使用することもできます。

3. 次のコマンドを入力して、ブローカーを起動または再起動します。
   - Linux の場合: **BROKER_INSTANCE_DIR/bin/artemis run**
   - Windows の場合: **BROKER_INSTANCE_DIR\bin\artemis-service.exe start**

5.4.2. ロールに基づく認証の設定

role-access 要素は、特定の MBean とその属性およびメソッドにロールをマップする方法を定義します。

手順

1. **BROKER_INSTANCE_DIR/etc/management.xml** 設定ファイルを開きます。
2. **role-access** 要素を検索し、設定を編集します。

```
<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>
```

   - この場合、ドメイン名が **org.apache.activemq.apache** の MBean 属性を照合検索します。
   - 一致する MBean 属性に対する **view**、**update**、または **amq** ロールへのアクセスは、ロールを追加する **list***、**get***、**set***、**is***、および * アクセスメソッドによって制御されます。**method = "*"**(ワイルドカード) 構文は、設定にリストされていない他の前メソッドをすべて指定する方法として使用されます。設定の各アクセスメソッドが MBean メソッド呼び出しに変換されます。
   - 呼び出された MBean メソッドは、設定に記載されているメソッドと照合されます。たとえば、ドメインが **org.apache.activemq.artemis** の MBean で **listMessages** というメソッドを呼び出すと、ブローカーは **list** メソッドの設定で定義されたロールと、アクセスの照合を行います。
   - MBean の完全なメソッド名を使用してアクセスを設定することもできます。以下に例を示します。

```
<access method="listMessages" roles="view,update,amq"/>
```

3. 次のコマンドを入力して、ブローカーを起動または再起動します。
   - Linux の場合: **BROKER_INSTANCE_DIR/bin/artemis run**
   - Windows の場合: **BROKER_INSTANCE_DIR\bin\artemis-service.exe start**

MBean プロパティーに一致する **key** 属性を追加して、ドメイン内の特定の MBean を照合することもできます。以下にいくつかの例を示します。

ロールのドメインのすべてのキューへのマッピング

この例は、**key** 属性を使用して、指定したドメインのすべてのキューにロールをマッピングする方法を示しています。

```
<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>
```

ドメインの特定のキューへのロールのマッピング

この例は、**key** 属性を使用して、特定の名前付きキューにロールをマッピングする方法を示しています。この例では、名前付きキューは **exampleQueue** です。

```
<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>
```

指定の接頭辞を含むすべてのキュー名へのロールのマッピング

以下の例は、指定した接頭辞が含まれるすべてのキューにロールをマップする方法を示しています。この例では、ワイルドカード演算子 (*) を使用して、接頭辞 **example** で始まるすべてのキュー名を照合します。

```
<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>
```

さまざまなロールのさまざまなキューセットへのマッピング

同じ属性を組み合わせた各種セット (例: さまざまなキューのセット) に対して、異なるロールをマッピングする場合などです。これを実現するために、設定ファイルに複数の **match** 要素を含めることができます。ただし、この場合、同じドメインに複数の一致が存在する可能性があります。

たとえば、以下のように設定された 2 つ **<match>** 要素について考えてみましょう。

```
<match domain="org.apache.activemq.artemis" key="queue=example*">
```

および

```
<match domain="org.apache.activemq.artemis" key="queue=example.sub*">
```

この設定を基して、**org.apache.activemq.artemis** ドメインの **example.sub.queue** という名前のキューは、両方のワイルドカードキーの式にマッチします。したがって、ブローカーは、最初の **match** 要素で指定されたロール、または 2 番目の **match** 要素で指定されたロールなど、キューにマップするロールのセットを決定するための優先順位付けスキームを必要とします。

同じドメインに複数の一致がある場合、ブローカーはロールのマッピング時に以下の優先順位スキームを使用します。

- 完全一致がワイルドカードの一致よりも優先される
- ワイルドカードのより長い一致が、より短いワイルドカード一致よりも優先されます。

この例では、長いワイルドカード式が **example.sub.queue** のキュー名と一致するため、ブローカーは 2 番目の **<match>** 要素に設定された role-mapping を適用します。

注記

default-access 要素は、**role-access** または **whitelist** 設定で処理されないすべてのメソッド呼び出しをすべて受け入れる要素です。**default-access** 要素および **role-access** 要素には、**match** 要素と同じセマンティクスがあります。

第6章 セキュリティー

この章では、管理者が使用できるさまざまなセキュリティーオプションと、その設定方法について説明します。管理者は、この章で提供される情報を使用して、AMQ Broker セキュリティーサブシステムの機能をニーズに合わせて調整できます。

6.1. AMQ 管理コンソールへのアクセス

AMQ Broker 7.1.0 以降では、デフォルトでローカルホストからのみ AMQ 管理コンソールにアクセスできます。リモートアクセスを有効にするには、**BROKER_INSTANCE_DIR/etc/jolokia-access.xml** の設定を変更する必要があります。詳細は、AMQ 管理コンソールと AMQ ブローカ接続の保護 を参照してください。

6.2. ネットワーク接続のセキュリティー保護

トランスポート層セキュリティー (TLS) には、次の 2 つの基本的な使用例があります。

- サーバー側 (または **一方向**)。サーバーのみが証明書を提示します。これは最も一般的な使用例です。
- クライアント側 (または **双方向**);サーバーとクライアントの両方が証明書を提示します。これは相互認証と呼ばれることもあります。

6.2.1. サーバー側証明書の設定

一方向 TLS は、**broker.xml** 内の関連する **acceptor** の URL で設定されます。以下は、TLS を使用しない非常に基本的な **acceptor** 設定です。

```
<acceptor name="artemis">tcp://0.0.0.0:61616</acceptor>
```

一方向 TLS を使用するように設定された同じ **acceptor** を次に示します。

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

この **acceptor** は、**sslEnabled**、**keyStorePath**、および **keyStorePassword** という 3 つの追加パラメーターを使用します。これらは、少なくとも、一方向 TLS を有効にするために必要です。

6.2.2. クライアント側証明書の設定

双方向 TLS は、一方向 TLS と同じ **sslEnabled**、**keyStorePath**、および **keyStorePassword** プロパティーを使用しますが、**needClientAuth** を追加して、独自の証明書を提示する必要があることをクライアントに指示します。以下に例を示します。

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

この設定は、クライアントの証明書が信頼できるプロバイダーによって署名されていることを前提としています。クライアントの証明書が信頼できるプロバイダーによって署名されていない場合 (自己署名など)、サーバーはクライアントの証明書をトラストストアにインポートし、**trustStorePath** と **trustStorePassword** を使用してアクセプターを設定する必要があります。以下に例を示します。

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!;needClientAuth=true;tru
stStorePath=../etc/client.truststore;trustStorePassword=5678!</acceptor>
```

注記

クライアント側の考慮事項

AMQ Broker は複数のプロトコルをサポートし、各プロトコルとプラットフォームには TLS パラメーターを指定する方法が異なります。ただし、Core protocol (ブリッジ) を使用するクライアントの場合、TLS パラメーターはブローカーのアクセプターと同様にコネクター URL で設定されます。

DNS 設定の詳細

以下は、注意が必要な設定の詳細です。

オプション	備考
sslEnabled	TLS を有効にするには **true** にする必要があります。デフォルトは **false** です。
keyStorePath	**acceptorで使用される場合は**、サーバーの証明書を保持するサーバーの SSL キーストアへのパスです (自己署名または認証局によって署名されているかどうか)。 **connectorで使用される場合は**、これはクライアント証明書を保持するクライアント側の TLS キーストアへのパスです。双方向 TLS (つまり相互認証) を使用している場合は、これは **connector** にのみ該当します。この値はサーバー上で設定されますが、ダウンロードしてクライアントで使用します。クライアントでサーバーの設定と異なるパスを使用する必要がある場合は、カスタムの**javax.net.ssl.keyStore** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.keyStore** システムプロパティーのいずれかを使用してサーバー側の設定をオーバーライドできます。AMQ 固有のシステムプロパティーは、クライアント上の別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に役立ちます。

オプション	備考
keyStorePassword	**acceptorで使用されると**、サーバー側のキーストアのパスワードになります。 **connector で使用されると**、クライアント側のキーストアのパスワードになります。双方向 TLS (つまり相互認証) を使用している場合は、これは **connector** にのみ該当します。この値はサーバー上で設定可能ですが、ダウンロードしてクライアントで使用します。クライアントでサーバーの設定と異なるパスワードを使用する必要がある場合は、カスタムの **javax.net.ssl.keyStorePassword** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.keyStorePassword** システムプロパティーを使用してサーバー側の設定をオーバーライドできます。AMQ 固有のシステムプロパティーは、クライアント上の別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に役立ちます。
trustStorePath	**acceptorで使用される場合**、これはサーバーが信頼するすべてのクライアントのキーを保持するサーバー側の TLS キーストアへのパスとなります。これは、双方向 TLS (相互認証) を使用している場合は、**acceptor** のみに該当します。 **connectorで使用される場合は**、クライアント側の TLS キーストアへのパスとなります。これは、クライアントが信頼するすべてのサーバーの公開鍵を保持します。この値はサーバー上で設定可能ですが、ダウンロードしてクライアントで使用します。クライアントでサーバーの設定と異なるパスを使用する必要がある場合は、カスタムの **javax.net.ssl.trustStore** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.trustStore** システムプロパティーのいずれかを使用してサーバー側の設定をオーバーライドできます。AMQ 固有のシステムプロパティーは、クライアント上の別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に役立ちます。

オプション	備考
trustStorePassword	**acceptor で使用されると**、サーバー側のトラストストアのパスワードになります。双方向 TLS (つまり相互認証) を使用している場合、**acceptor** のみに該当します。 **connector で使用されると**、クライアント側のトラストストアのパスワードになります。この値はサーバー上で設定可能ですが、ダウンロードしてクライアントで使用します。クライアントがサーバーで設定されているものとは異なるパスワードを使用する必要がある場合は、通常の **javax.net.ssl.trustStorePassword** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.trustStorePassword** システムプロパティーを使用して、サーバー側の設定を上書きできます。AMQ 固有のシステムプロパティーは、クライアント上の別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に役立ちます。
enabledCipherSuites	**acceptor** または **connector** で使用されるかに関係なく、TLS 通信に使用される暗号スイートのコンマ区切りリストになります。デフォルト値は **null** です (JVM のデフォルト値を使用)。
enabledProtocols	**acceptor** または **connector** で使用されるかに関係なく、TLS 通信に使用されるプロトコルのコンマ区切りリストになります。デフォルト値は **null** です (JVM のデフォルト値を使用)。
needClientAuth	このプロパティーは **acceptor** 専用です。このアクセプターに接続するクライアントに対して、双方向 TLS が必要であることを示します。有効な値は **true** または **false** です。デフォルトは **false** です。

6.2.3. 証明書ベースの認証の追加

JAAS 証明書認証ログインモジュールでは、TLS を使用する必要があり、クライアントは独自の証明書で設定する必要があります。このシナリオでは、認証は、JAAS 証明書認証プラグインによって直接ではなく、TLS ハンドシェーク中に実際に実行されます。

プラグインのロールは次のとおりです。

- 受け入れ可能なユーザーのセットをさらに制限する (関連するプロパティーファイルに明示的にリストされているユーザー識別名 (DN) のみが認証を受ける資格があるため)。
- グループのリストを受信したユーザー ID に関連付けて、承認との統合を容易にする。
- 着信証明書の存在を要求する (デフォルトでは、TLS 層はクライアント証明書の存在をオプションとして扱うように設定されています)。

JAAS 証明書ログインモジュールは、フラットテキストファイルのペアに証明書 DN のコレクションを保存します。ファイルは、ユーザー名とグループ ID のリストを各識別名に関連付けます。

証明書ログインモジュールは、**org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule** クラスで実装されます。

前提条件

- **login.config** ファイルで設定された証明書ログイン。
- 有効な **artemis-users.properties** ファイル。
- 有効な **artemis-roles.properties** ファイル。
- ユーザー証明書からのサブジェクト DN

手順

1. ユーザー証明書からサブジェクト DN を取得します。
   a. キーストアファイルから一時ファイルに証明書をエクスポートします。必要な値を次のコマンドに置き換えます。

```
keytool -export -file __FILENAME__ -alias broker-localhost -keystore broker.ks -storepass __PASSWORD__
```

   b. エクスポートされた証明書の内容を出力します。

```
keytool -printcert -file __FILENAME__
```

   出力は以下のようになります。

```
Owner: CN=localhost, OU=broker, O=Unknown, L=Unknown, ST=Unknown, C=Unknown (1)
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
```

   (1) サブジェクト DN。Subject DN の入力の使用形式はプラットフォームによって異なります。上記の文字列は、以下のように表現することもできます。

```
Owner: `CN=localhost,\ OU=broker,\ O=Unknown,\ L=Unknown,\ ST=Unknown,\ C=Unknown`
```

2. 証明書ベースの認証の設定
   a. **login.config** ファイルを開き、ユーザーおよびロールのプロパティーファイルを参照します。
   b. 前のステップで宣言されたファイルを開き、必要な情報を提供します。

ユーザーとそれに対応する DN は、**artemis-users.properties** ファイルにリストされている必要があります。利用可能なロールとそれらのロールを保持するユーザーは、**artemis-roles.properties** ファイルで定義されています。

これらのファイルの構文の例を以下に示します。

c. 以下に示すように、セキュリティードメインエイリアス (このインスタンスでは **activemq**) が **bootstrap.xml** で参照されていることを確認します。

```
<jaas-security domain="activemq"/>
```

設定例

次の例は、**login.config** ファイルで証明書ログインモジュールを設定する方法を示しています。

例6.1 login.config

```
activemq {
   org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule 1
      debug=true 2
      org.apache.activemq.jaas.textfiledn.user="artemis-users.properties" 3
      org.apache.activemq.jaas.textfiledn.role="artemis-roles.properties"; 4
};
```

1. JAAS レルムを設定します。この例では、単一の **org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule** を使用しています。
2. デバッグのオン (**true**) またはオフ (**false**) を切り替えます。デフォルトは **false** です。
3. ユーザーデータの保存に使用するファイルを定義します (**login.config** ファイルを含むディレクトリーへの相対パス)。
4. ロールデータの保存に使用するファイルを定義します (**login.config** ファイルを含むディレクトリーへの相対パス)。

artemis-users.properties ファイルは、**user=StringifiedSubjectDN** (文字列エンコーディングは RFC 2253 で指定されています) を持つプロパティーのリストで設定されています。

例6.2 artemis-users.properties

```
system=CN=system,O=Progress,C=US 1
user=CN=humble user,O=Progress,C=US
guest=CN=anon,O=Progress,C=DE
```

1. **system** という名前のユーザーは、**CN=system,O=Progress,C=US** サブジェクト DN にマップされます。

artemis-roles.properties ファイルは、**role=user** のパターンに従います。**user** は、単一のユーザーまたはユーザーのコンマ区切りリストのいずれかに置き換えます。

例6.3 artemis-roles.properties

```
admins=system
users=system,user (1)
guests=guest
```

(1) 複数のユーザーをコンマ区切りのエントリーとして含めることができます。

6.2.4. AMQP クライアントの証明書ベースの認証設定

SASL EXTERNAL メカニズム設定パラメーターを使用して、ブローカーへの接続時に証明書ベースの認証用に AQMP クライアントを設定します。

ブローカーは、証明書を認証するのと同じ方法で、AMQP クライアントの TSL/SSL 証明書を認証します。

- ブローカーはクライアントの TLS/SSL 証明書を読み取り、証明書のサブジェクトからアイデンティティーを取得します。
- 証明書サブジェクトは、証明書ログインモジュールによってブローカー ID にマッピングされます。次に、ブローカーは、マップされたユーザーをロールに基づいて承認します。

前提条件
証明書ベースの認証を使用するように AMQP クライアントを設定する前に、次のタスクを完了する必要があります。

- サーバー側の SSL/TLS 証明書の設定
- クライアント側の SSL/TLS 証明書の設定
- 証明書ベースの認証を使用するブローカーの設定

手順
AMQP クライアントが証明書ベースの認証を使用できるようにするには、クライアントがブローカーへの接続に使用する URI に設定パラメーターを追加する必要があります。

1. 編集する URI が含まれるリソースを開きます。

   ```
   amqps://localhost:5500
   ```

2. **sslEnabled=true** パラメーターを追加して、接続の TSL/SSL を有効にします。

   ```
   amqps://localhost:5500?sslEnabled=true
   ```

3. クライアントの TrustStore および KeyStore に関連するパラメーターを追加して、ブローカーとの TSL/SSL 証明書の交換を有効にします。

   ```
   amqps://localhost:5500?
   sslEnabled=true&trustStorePath=TRUST_STORE_PATH&trustStorePassword=TRUST_STORE_PASSWORD&keyStorePath=KEY_STORE_PATH&keyStorePassword=KEY_STORE_PASSWORD
   ```

4. パラメーター **saslMechanisms=EXTERNAL** を追加し、TSL/SSL 証明書で見つかったアイデンティティーを使用してブローカーがクライアントを認証するよう要求します。

```
amqps://localhost:5500?
sslEnabled=true&trustStorePath=TRUST_STORE_PATH&trustStorePassword=TRUST_STORE_PASSWORD&keyStorePath=KEY_STORE_PATH&keyStorePassword=KEY_STORE_PASSWORD&saslMechanisms=EXTERNAL
```

関連情報

- AMQ Broker での証明書ベースの認証に関する詳細は、「証明書ベースの認証の追加」 を参照してください。
- AMQP クライアントの設定に関する詳細は、クライアント固有の製品ドキュメントについて Red Hat カスタマーポータル を参照してください。

6.2.5. 複数のログインモジュールの使用

ログインモジュールを組み合わせて、より複雑なユースケースに対応することができます。ログインモジュールを組み合わせる最も一般的な理由は、匿名ユーザーと認証情報を送信するユーザーの両方の認証をサポートすることです。

前提条件

さまざまな認証の組み合わせの前提条件は、実装されている方法によって異なります。最も一般的な複数ログインシナリオの前提条件は次のとおりです。

- 有効な **artemis-users.properties** ファイル
- 有効な **artemis-roles.properties** ファイル
- 匿名アクセス用に設定された **login.config** ファイル

手順

1. **login.config** ファイルを編集して、目的の認証モジュールのエントリーを追加します。
2. 環境のニーズに合わせて、各モジュールエントリーにパラメーターを設定します。
3. 以下に示すように、セキュリティードメインエイリアス (このインスタンスでは **activemq**) が **bootstrap.xml** で参照されていることを確認します。

例6.4 bootstrap.xml

```
<jaas-security domain="activemq"/>
```

設定例

次の例は、複数のログイン設定のカスケードの性質を示しています。

例6.5 login.config

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient (1)
     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 (2)
        debug=true
        org.apache.activemq.jaas.guest.user="guest"
        org.apache.activemq.jaas.guest.role="restricted";
};
```

(1) ユーザーが認証情報を提示すると、パスワード認証モジュールがアクティブになります

(2) ユーザーが認証情報を提示しない場合や、指定した認証情報が正しくない場合は、ゲスト認証がアクティブになります。

次の例は、認証情報を持たないユーザーのみがゲストとしてログインするユースケースの JAAS ログインエントリーを設定する方法を示しています。ログインモジュールの順序が逆になり、プロパティーログインモジュールに付加されたフラグが **requisite** に変更されることに注意してください。

例6.6 login.config

```
activemq {
    org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule sufficient (1)
        debug=true
        credentialsInvalidate=true (2)
        org.apache.activemq.jaas.guest.user="guest"
        org.apache.activemq.jaas.guest.role="guests";

    org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule requisite (3)
        debug=true
        org.apache.activemq.jaas.properties.user="artemis-users.properties"
        org.apache.activemq.jaas.properties.role="artemis-roles.properties";
};
```

(1) ログイン認証情報が提示されない場合には、ゲスト認証モジュールが有効になります。

(2) ゲストログインモジュールの設定で、**credentialsInvalidate** オプションを **true** に設定する必要があります。

(3) 認証情報が提示され、認証情報が有効である必要がある場合、パスワードログインモジュールが有効になります。

6.2.6. アドレスグループとサブグループの複数のセキュリティーの設定

以下は、**broker.xml** ファイルのセキュリティーブロックの例です。この例に基づくさまざまな設定オプションについて、このセクションで説明します。

```
<security-setting match="globalqueues.europe.#"> (1)
  <permission type="createDurableQueue" roles="admin"/> (2)
  <permission type="deleteDurableQueue" roles="admin"/> (3)
  <permission type="createNonDurableQueue" roles="admin, guest, europe-users"/> (4)
```

```
    <permission type="deleteNonDurableQueue" roles="admin, guest, europe-users"/> 5
    <permission type="send" roles="admin, europe-users"/> 6
    <permission type="consume" roles="admin, europe-users"/> 7
</security-setting>
```

1 「#」 文字は単語の任意のシーケンスを意味します。単語は、「.」 文字で区切ります。ワイルドカード構文の完全な説明については、AMQ Broker ワイルドカード構文 を参照してください。上記のセキュリティーブロックは、文字列 **globalqueues.europe** で始まるすべてのアドレスに適用されます。

2 3 **admin** ロールが割り当てられたユーザーのみが、文字列 **globalqueues.europe** で始まるアドレスにバインドされた永続キューを作成したり、削除したりできます。

4 5 **admin** ロール、**guest**、または **europe-users** ロールが割り当てられたユーザーは、文字列 **globalqueues.europe** で始まるアドレスにバインドされた一時キューを作成したり、削除したりできます。

6 7 **admin** ロールまたは **europe-users** ロールを持つすべてのユーザーは、これらのアドレスにメッセージを送信したり、文字列 **globalqueues.europe** で始まるアドレスにバインドされたキューからメッセージを消費したりできます。

ユーザー間のマッピングと、ユーザーに割り当てられたロールのマッピングは、**セキュリティーマネージャー** と呼ばれるコンポーネントによって処理されます。セキュリティーマネージャーは、ブローカーに保存されているプロパティーファイルからユーザーの認証情報を読み取ります。デフォルトでは、AMQ Broker は **org.apache.activemq.artemis.spi.core.security.ActiveMQJAASSecurityManager** セキュリティーマネージャーを使用します。このデフォルトのセキュリティーマネージャーは、JAAS および Red Hat JBoss Enterprise Application Platform(JBoss EAP) のセキュリティーとの統合を提供します。

要件に応じて、各 XML ファイルに複数の **security-setting** 要素を含めることも、何も含めないこともできます。**broker.xml** ファイルに、一連のアドレスに適用できる複数の security-setting 要素が含まれている場合には、最も一致率の高いものが優先されます。

この例について見てみましょう。ここに別の **security-setting** ブロックがあります:

```
<security-setting match="globalqueues.europe.orders.#">
    <permission type="send" roles="europe-users"/>
    <permission type="consume" roles="europe-users"/>
</security-setting>
```

この **security-setting** ブロックでは、**globalqueues.europe.orders.#** の一致は、前の一致 'globalqueues.europe.\#' よりも一致する部分が多くなっています。したがって、globalqueues.europe.orders.\# に一致するアドレスは、後者の security-setting ブロックから ***のみ*** セキュリティー設定を取得します。

設定は以前のブロックから継承されないことに注意してください。すべての設定は、より一致部分の多いブロックから取得されるため、**globalqueues.europe.orders.plastics** のアドレスの場合に、唯一存在するパーミッションは **europe-users** ロールの **send** と **consume** です。**createDurableQueue**、**deleteDurableQueue**、**createNonDurableQueue**、**deleteNonDurableQueue** のパーミッションは、他の security-setting ブロックからは継承されません。

パーミッションは継承されないため、指定しないだけで、より限定的な security-setting ブロックのパーミッションを効果的に拒否することができます。それ以外の方法では、アドレスのサブグループのパーミッションを拒否することはできません。

6.2.7. リソース制限の設定

承認や認証に関連する通常のセキュリティー設定以外に、特定のユーザーが実行できる特定の制限を設定すると便利です。たとえば、ユーザーが作成できる接続の数や、ユーザーが作成できるキューの数を制限できます。

6.2.7.1. 接続およびキュー制限の設定

リソース制限の設定に使用される XML の例を次に示します。

```
<resource-limit-settings>
  <resource-limit-setting match="myUser">
    <max-connections>5</max-connections>
    <max-queues>3</max-queues>
  </resource-limit-setting>
</resource-limit-settings>
```

address-setting からの **match** とは異なり、この **match** ではワイルドカード構文は使用されません。これは、ユーザーに対する制限の単純な 1:1 マッピングです。

- **max.connections**一致したユーザーがブローカーに対して実行できる接続の数を定義します。デフォルトは **-1** で、制限がないことを意味します。
- **max-queues**一致したユーザーが作成できるキューの数を定義します。デフォルトは **-1** で、制限がないことを意味します。

6.3. LDAP との統合

6.3.1. 認証での LDAP の使用

LDAP ログインモジュールは、中央の X.500 ディレクトリーサーバーに保存されているユーザーデータに対して受信認証情報を確認して、認証および承認を有効にします。これは **org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule** で実装されます。

手順

1. **BROKER_INSTANCE_DIR/etc/broker.xml** ファイルを開き、次の行を追加します。

```
<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>
```

2. **<broker-instance-dir>/etc/login.config** ファイルを開きます。
3. 適切なパラメーターを使用して、適切なエイリアスブロックを見つけて編集します (以下に含まれる例を参照してください)。

4. ブローカーを起動または再起動します (サービスまたはプロセス)。

注記

Apache DS は DN パスの **OID** 部分を使用します。ただし、Microsoft AD はそうではなく、代わりに **CN** 部分を使用します。
例えば; DN パス **oid=testuser,dc=example,dc=com** は Apache DS で、**cn=testuser,dc=example,dc=com** は Microsoft AD で使用されます。

例6.7 Apache DS login.config 設定の例

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
    debug=true (1)
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory (2)
    connectionURL="ldap://localhost:10389" (3)
    connectionUsername="uid=admin,ou=system" (4)
    connectionPassword=secret (5)
    connectionProtocol=s (6)
    connectionTimeout=5000 (7)
    authentication=simple (8)
    userBase="dc=example,dc=com" (9)
    userSearchMatching="(uid={0})" (10)
    userSearchSubtree=true (11)
    userRoleName= (12)
    readTimeout=5000 (13)
    roleBase="dc=example,dc=com" (14)
    roleName=cn (15)
    roleSearchMatching="(member={0})" (16)
    roleSearchSubtree=true (17)
    ;
};
```

例6.8 Microsoft Active Directory login.config 設定の例

```
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,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
        ;
};
```

注釈

1. デバッグのオン (**true**) またはオフ (**false**) を切り替えます。デフォルトは **false** です。
2. **initialContextFactory** パラメーターは常に **com.sun.jndi.ldap.LdapCtxFactory** に設定する必要があります
3. LDAP URL ldap://Host:Port を使用してディレクトリーサーバーの場所を指定します。オプションでこの URL を修飾するには、スラッシュ / とその後にディレクトリーツリーの特定ノードの DN を追加します。Apache DS のデフォルトポートは **10389** で、Microsoft AD のデフォルト値は **389** です。
4. ディレクトリーサーバーへの接続を開くユーザーの DN。たとえば、**uid=admin,ou=system** です。ディレクトリーサーバーでは、通常、クライアントが接続を開くためにユーザー名/パスワードの認証情報を提示する必要があります。
5. **connectionUsername** の DN に一致するパスワード。DIT のディレクトリーサーバーでは通常、パスワードは対応するディレクトリーエントリーの **userPassword** 属性として保存されます。
6. すべての値はサポートされますが、実質的に使用されません。このオプションはデフォルト値がないために明示的に設定する必要があります。
7. ブローカーがディレクトリーサーバーに接続することができる最大時間をミリ秒単位で指定します。ブローカーがこの時間内にディレクトリーに接続できない場合、接続の試行を中止します。このプロパティーにゼロまたはそれよりも小さい値を指定すると、代わりに基盤の TCP プロトコルのタイムアウト値が使用されます。値を指定しない場合、ブローカーは接続を無期限に待機するか、または基盤のネットワークがタイムアウトします。

   接続に対して接続プールが要求された場合、このプロパティーは、最大プールサイズにすでに達し、プール内のすべての接続が使用されている場合に、ブローカーが接続を待つ最大時間を指定します。ゼロまたはそれよりも小さい値を指定すると、ブローカーは接続を無期限に利用可能になるまで待機します。それ以外の場合は、ブローカーは最大待機時間に達すると接続の試行を中止します。
8. LDAP サーバーにバインドする際に使用する認証方法を指定します。このパラメーターは **simple** (ユーザー名とパスワードが必要) または **none** (匿名アクセスを許可) のいずれかに設定できます。
9. ユーザーエントリーを検索する DIT の特定のサブツリーを選択します。サブツリーは、サブツリーのベースノードを指定する DN で指定されます。たとえば、このオプションを **ou=User,ou=ActiveMQ,ou=system** に設定すると、ユーザーエントリーの検索は **ou=User,ou=ActiveMQ,ou=system** ノードの下にあるサブツリーに限定されます。
10. **userBase** で選択したサブツリーに適用される LDAP 検索フィルターを指定します。詳細は、以下の 検索一致 セクションを参照してください。
11. **userBase** で指定されたノードを基準にして、どの程度の深さまでユーザーエントリーの検索を掘り下げるかを指定します。このオプションはブール値です。**false** に設定すると、**userBase** ノードにある子エントリーの 1 つとの一致を試みることを示し (**javax.naming.directory.SearchControls.ONELEVEL_SCOPE** にマップ)、**true** は、**userBase**

ノードのサブツリーに属する任意のエントリーとの一致を試みることを示します (**javax.naming.directory.SearchControls.SUBTREE_SCOPE** にマップ)。

12 ユーザーのロール名のリストを含むユーザーエントリーの多値属性の名前を指定します (ロール名は、ブローカの承認プラグインによってグループ名として解釈されます)。このオプションを省略すると、ユーザーエントリーからロール名は抽出されません。

13 ブローカーがディレクトリーサーバーから LDAP 要求への応答を受信するまで待機する最大時間をミリ秒単位で指定します。この時間内にブローカーがディレクトリーサーバーから応答を受信しない場合、ブローカーは要求を中止します。0 または less の値を指定すると、値を指定しないと、ディレクトリーサーバーから LDAP 要求への応答が無期限に待機します。

14 ロールデータをディレクトリーサーバーに直接保存する場合は、**userRoleName** オプションを指定する代わりに (または指定してその上に)、ロールオプション (**roleBase**、**roleSearchMatching**、**roleSearchSubtree**、および **roleName**) の組み合わせを使用できます。このオプションは、ロール/グループエントリーを検索する DIT の特定のサブツリーを選択します。サブツリーは、サブツリーのベースノードを指定する DN で指定されます。たとえば、このオプションを **ou=Group,ou=ActiveMQ,ou=system** に設定すると、role/group エントリーの検索が **ou=Group,ou=ActiveMQ,ou=system** ノードの下にあるサブツリーに限定されます。

15 ロール/グループの名前が含まれるロールエントリーの属性タイプ (C、O、OU など) を指定します。このオプションを省略すると、ロール検索機能は事実上無効になっています。

16 **roleBase** で選択したサブツリーに適用される LDAP 検索フィルターを指定します。詳細は、以下の 検索一致 セクションを参照してください。

17 **roleBase** で指定されたノードを基準にして、どの程度の深さまでロールエントリーの検索を掘り下げるかを指定します。**false** (デフォルト) に設定すると、**roleBase** ノードの子エントリー (**javax.naming.directory.SearchControls.ONELEVEL_SCOPE** にマップ) のいずれかに一致するものの検索を試行します。**true** の場合、roleBase ノードのサブツリーに属するエントリーに一致するものの検索を試行します (**javax.naming.directory.SearchControls.SUBTREE_SCOPE** にマップ)。

検索一致

userSearchMatching

LDAP 検索操作に渡す前に、この設定パラメーターで指定される文字列の値は **java.text.MessageFormat** クラスで実装される文字列置換の内容により異なります。つまり、特別な文字列 **{0}** は、受信クライアントの認証情報から抽出されたユーザー名に置き換えられます。置換後、この文字列は LDAP 検索フィルターとして解釈されます (構文は IETF 標準 RFC 2254 で定義されています)。

たとえば、このオプションが **(uid={0})** に設定され、受信したユーザー名が **jdoe** の場合には、文字列置換後の検索フィルターは **(uid=jdoe)** になります。

ユーザーベースが選択したサブツリー (**ou=User,ou=ActiveMQ,ou=system**) に、結果の検索フィルターが適用されると、エントリー **uid=jdoe,ou=User,ou=ActiveMQ,ou=system** にマッチします。

検索フィルター構文の簡単な説明は、Oracle の JNDI チュートリアル を参照してください。

roleSearchMatching

これは、2 つの置換文字列をサポートする点を除き、**userSearchMatching** オプションと同じように機能します。

置換文字列 **{0}** は、一致したユーザーエントリーの DN をすべて置き換えます (つまり、ユーザー検索の結果)。たとえば、ユーザー **jdoe** の場合には、置換された文字列は **uid=jdoe,ou=User,ou=ActiveMQ,ou=system** になります。

置換文字列 **{1}** は、受信したユーザー名を置き換えます。たとえば、**jdoe** です。

このオプションを **(member=uid={1})** に設定し、受信したユーザー名が **jdoe** の場合には、文字列の置換後に検索フィルターは **(member=uid=jdoe)** になります (ApacheDS 検索フィルター構文を想定)。

ロールベース **ou=Group,ou=ActiveMQ,ou=system** で選択したサブツリーに、結果の検索フィルターが適用されると、**uid=jdoe** と同等の **member** 属性を持つすべてのロールエントリーがマッチします (**member** 属性の値は DN です)。

このオプションはデフォルト値がないため、ロールの検索が無効であっても常に設定する必要があります。OpenLDAP を使用すると、検索フィルターの構文は **(member:=uid=jdoe)** になります。

6.3.2. LDAP 認証の設定

LegacyLDAPSecuritySettingPlugin セキュリティー設定プラグインは、**LDAPAuthorizationMap** および **cachedLDAPAuthorizationMap** で過去に AMQ 6 で処理されたセキュリティー情報を読み取り、できるだけ、この情報を対応する AMQ 7 セキュリティー設定に変換します。

AMQ 6 および AMQ 7 のブローカーのセキュリティー実装は、完全に一致しません。そのため、プラグインは、2 つのバージョン間の翻訳を実行し、ほぼ同等の機能を実現します。

プラグイン設定の例を以下に示します。

```
<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>
```

class-name

この実装は **org.apache.activemq.artemis.core.server.impl.LegacyLDAPSecuritySettingPlugin** です。

initialContextFactory

LDAP への接続に使用される初期コンテキストファクトリー。これは、常に **com.sun.jndi.ldap.LdapCtxFactory** (デフォルト値) に設定する必要があります。

connectionURL

LDAP URL **ldap://Host:Port** を使用してディレクトリーサーバーの場所を指定します。オプションでこの URL を修飾するには、スラッシュ / とその後にディレクトリーツリーの特定ノードの識別名 (DN) を追加します。例: **ldap://ldapserver:10389/ou=system**デフォルト値は **ldap://localhost:1024** です。

connectionUsername

ディレクトリーサーバーへの接続を開くユーザーの DN。たとえば、**uid=admin,ou=system** です。ディレクトリーサーバーでは、通常、クライアントが接続を開くためにユーザー名/パスワードの認証情報を提示する必要があります。

connectionPassword

connectionUsername の DN に一致するパスワード。DIT のディレクトリーサーバーでは通常、パスワードは対応するディレクトリーエントリーの **userPassword** 属性として保存されます。

connectionProtocol

現在は使用されていません。今後、このオプションを使用すると、ディレクトリーサーバーへの接続に Secure Socket Layer(SSL) を選択できます。このオプションはデフォルト値がないために明示的に設定する必要があります。

authentication

LDAP サーバーにバインドする際に使用する認証方法を指定します。このパラメーターの有効な値は **simple** (ユーザー名とパスワード) または **none** (匿名) です。デフォルト値は **simple** です。

注記

Simple Authentication and Security Layer(SASL) 認証はサポートされ **ません**。

前述の設定例に記載されていない他の設定は次のとおりです。

destinationBase

子がすべての宛先にパーミッションを提供するノードの DN を指定します。この場合、DN はリテラル値です (つまり、プロパティー値の文字列は置き換えられません)。たとえば、このプロパティーの通常の値は **ou=destinations,o=ActiveMQ,ou=system** で、デフォルト値は **ou=destinations,o=ActiveMQ,ou=system** です。

filter

あらゆる種類の宛先のパーミッションを検索するときに使用する LDAP 検索フィルターを指定します。検索フィルターは、キューまたはトピックノードの子孫の 1 つとの照合を試行します。デフォルト値は **(cn=*)** です。

roleAttribute

ロールの DN の値が **filter** に一致するノードの属性を指定します。デフォルト値は **uniqueMember** です。

adminPermissionValue

admin パーミッションに一致する値を指定します。デフォルト値は **admin** です。

readPermissionValue

read パーミッションに一致する値を指定します。デフォルト値は **read** です。

writePermissionValue

write パーミッションに一致する値を指定します。デフォルト値は **write** です。

enableListener

LDAP サーバーでのを自動的に受信し、ブローカーの認証設定をリアルタイムで更新するリスナーを有効にするかどうかを指定します。デフォルト値は **true** です。

mapAdminToManage

レガシー (AMQ 6) の **admin** パーミッションを AMQ 7 の **manage** パーミッションにマップするかどうかを指定します。以下の表のマッピングセマンティクスの詳細を参照してください。デフォルト値は **false** です。

LDAP で定義されたキューまたはトピックの名前は、セキュリティー設定の一致として機能し、パー

ミッション値は AMQ 6 タイプから AMQ 7 タイプにマッピングされ、ロールはそのままマッピングされます。LDAP で定義されたキューまたはトピックの名前はセキュリティー設定の一致として機能するため、セキュリティー設定は想定どおりに JMS 宛先に適用されない可能性があります。これは、必要に応じて、AMQ 7 は常に JMS 宛先を jms.queue.または jms.topic.で接頭辞を付けるためです。

AMQ 6 には、**read**、**write**、および **admin** の 3 つのパーミッションタイプがあります。これらのアクセス許可の種類については、ActiveMQ Web サイト (http://activemq.apache.org/security.html) で説明されています。

AMQ 7 には以下のパーミッションタイプがあります。

- **createAddress**
- **deleteAddress**
- **createDurableQueue**
- **deleteDurableQueue**
- **createNonDurableQueue**
- **deleteNonDurableQueue**
- **send**
- **consume**
- **manage**
- **browse**

以下の表は、セキュリティー設定プラグインが AMQ 6 パーミッションタイプを AMQ 7 パーミッションタイプにマッピングする方法を示しています。

AMQ 6 パーミッションタイプ	AMQ 7 のパーミッションタイプ
read	consume, browse
write	send
admin	createAddress、deleteAddress、createDurableQueue、deleteDurableQueue、createNonDurableQueue、deleteNonDurableQueue、manage(**mapAdminToManage** が **true** に設定されている場合)

下記のとおり、プラグインが AMQ 6 と AMQ 7 のパーミッションタイプ間の変換を実行するケースがあります。これにより、同等の機能を実現できます。

- AMQ 6 には同様のパーミッションタイプがないため、このマッピングにはデフォルトで、AMQ 7 の **manage** パーミッションタイプが含まれません。ただし、**mapAdminToManage** が **true** に設定されている場合、プラグインは AMQ 6 **admin** パーミッションを AMQ 7 **manage** にマップします。

- AMQ 6 の **admin** パーミッションタイプは、宛先が存在しない場合にブローカーが宛先を自動作成し、ユーザーがその宛先にメッセージを送信するかどうかを決定します。AMQ 7 では、ユーザーが宛先にメッセージを送信するパーミッションがある場合に、自動的に宛先の自動作成を許可します。したがって、プラグインは、デフォルトで、レガシー **admin** パーミッションを上記の AMQ 7 パーミッションにマップします。また、このプラグインは、**mapAdminToManage** が **true** に設定されている場合に、AMQ 6 **admin** パーミッションを AMQ 7 の **manage** パーミッションにマップします。

6.3.3. login.config ファイルでのパスワードの暗号化

組織は LDAP でデータを安全に保存することが多いので、**login.config** ファイルには、ブローカーが組織の LDAP サーバーと通信するために必要な設定を含めることができます。この設定ファイルには、通常、LDAP サーバーにログインするパスワードが含まれるため、このパスワードをマスクする必要があります。

前提条件

- 「LDAP 認証の設定」 の説明に従って、必要なプロパティーを追加するように **login.config** ファイルを修正している。

手順

次の手順では、**BROKER_INSTANCE_DIR/etc/login.config** ファイルにある **connectionPassword** の値をマスクする方法について説明します。

1. コマンドプロンプトで、**mask** ユーティリティーを使用してパスワードを暗号化します。

```
$ BROKER_INSTANCE_DIR/bin/artemis mask PASSWORD
```

   暗号化されたパスワードが画面に表示されます。

```
result: 3a34fd21b82bf2a822fa49a8d8fa115d
```

2. **BROKER_INSTANCE_DIR/etc/login.config** ファイルを開き、**connectionPassword** を見つけます。

```
connectionPassword = PASSWORD
```

3. プレーンテキストのパスワードを、手順 1 で作成した暗号化された値に置き換えます。

```
connectionPassword = 3a34fd21b82bf2a822fa49a8d8fa115d
```

4. 暗号化された値は、識別子 **"ENC()"** でラップします。

```
connectionPassword = "ENC(3a34fd21b82bf2a822fa49a8d8fa115d)"
```

login.config ファイルにマスクされたパスワードが追加されました。パスワードは **"ENC()"** 識別子でラップされるため、AMQ Broker では使用前にこれを復号化します。

関連情報

AMQ Broker に含まれる設定ファイルの詳細は、AMQ Broker 設定ファイルおよび場所 を参照してください。

6.4. KERBEROS との統合

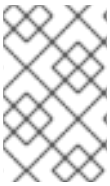
AMQP プロトコルでメッセージを送受信する場合、クライアントは Simple Authentication and Security Layer (SASL) フレームワークの GSSAPI メカニズムを使用して、AMQ Broker が認証する Kerberos セキュリティー認証情報を送信できます。Kerberos 認証情報は、認証されたユーザーを LDAP ディレクトリーまたはテキストベースのプロパティーファイルで設定された割り当てられたロールにマッピングすることで、承認にも使用できます。

Transport Layer Sockets (TLS) と SASL を併用して、メッセージングアプリケーションのセキュリティーを確保できます。SASL はユーザー認証を行い、TLS はデータの整合性を確保します。

AMQ Broker が Kerberos 認証情報を認証および承認する前に、Kerberos インフラストラクチャーをデプロイし、設定する必要があります。Kerberos のデプロイメントに関する詳細は、オペレーティングシステムのドキュメントを参照してください。たとえば、オペレーティングシステムが RHEL 7 の場合は、システムレベル認証ガイドの Kerberos の使用 の章を参照してください。Windows 用の Kerberos 認証の概要 も利用できます。

注記

AMQ Broker が Kerberos 認証情報を認証および承認する前に、Kerberos インフラストラクチャーをデプロイし、設定する必要があります。



注記

Oracle または IBM JDK のユーザーは、Java Cryptography Extension(JCE) をインストールする必要があります。詳細は、Oracle version of the JCE または IBM version of the JCE のドキュメントを参照してください。

6.4.1. Kerberos を使用するためのネットワーク接続の有効化

AMQ Broker は、Simple Authentication and Security Layer (SASL) フレームワークの GSSAPI メカニズムを使用して、Kerberos セキュリティー認証情報と統合します。AMQ Broker で Kerberos を使用するには、Kerberos 認証情報を使用するクライアントを認証または承認する各 **acceptor** を GSSAPI メカニズムを使用するように設定する必要があります。

前提条件

AMQ Broker が Kerberos 認証情報を認証および承認する前に、Kerberos インフラストラクチャーをデプロイし、設定する必要があります。

手順

1. ブローカーを停止します。
   a. ブローカーが Linux で実行されている場合:

      ```
      BROKER_INSTANCE_DIR/bin/artemis stop
      ```

   b. ブローカーがサービスとして Windows で実行されている場合:

      ```
      BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
      ```

2. **BROKER_INSTANCE_DIR/etc** にある **broker.xml** 設定ファイルを開きます。
3. 次の例に示すように、名前と値のペア **saslMechanisms=GSSAPI** を、**acceptor** の URL のクエリー文字列に追加します。

```
<acceptor name="amqp">
  tcp://0.0.0.0:5672?protocols=AMQP;saslMechanisms=GSSAPI
</acceptor>
```

その結果、Kerberos クレデンシャルの認証時に GSSAPI メカニズムを使用するアクセプターが作成されます。

4. (オプション) **PLAIN** および **ANONYMOUS** SASL メカニズムもサポートされます。**GSSAPI** に加えてこれらの他のメカニズムを使用する場合は、それらを **saslMechanisms** のリストに追加します。各値は必ずコンマで区切ります。次の例では、名前と値のペア **saslMechanisms=GSSAPI** を変更して、値 **PLAIN** を追加します。

```
<acceptor name="amqp">
  tcp://0.0.0.0:5672?protocols=AMQP;saslMechanisms=GSSAPI,PLAIN
</acceptor>
```

結果は、**GSSAPI** および **PLAIN** SASL メカニズムの両方を使用するアクセプターです。

5. ブローカーを起動します。

   a. ブローカーが Linux で実行されている場合:

   ```
   BROKER_INSTANCE_DIR/bin/artemis run
   ```

   b. ブローカーがサービスとして Windows で実行されている場合:

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe start
   ```

関連情報
アクセプターの詳細は、アクセプター を参照してください。

6.4.2. Kerberos 認証情報を使用したクライアントの認証

AMQ Broker は、Simple Authentication and Security Layer (SASL) フレームワークからの GSSAPI メカニズムを使用する AMQP 接続の Kerberos 認証をサポートします。

ブローカーは、Java Authentication and Authorization Service (JAAS) を使用して Kerberos アクセプターの認証情報を取得します。Java インストールに含まれる JAAS ライブラリーは、Kerberos 認証情報を認証するログインモジュール **Krb5LoginModule** でパッケージ化されています。**Krb5LoginModule** の詳細は、Java ベンダーのドキュメントを参照してください。たとえば、Oracle は、Java 8 ドキュメント の一部として、**Krb5LoginModule** ログインモジュールに関する情報を提供します。

前提条件
Kerberos セキュリティー認証情報を使用して AMQP 接続を認証する前に、アクセプターの GSSAPI メカニズムを有効にする必要があります。

手順

1. ブローカーを停止します。

   a. ブローカーが Linux で実行されている場合:

   ```
   BROKER_INSTANCE_DIR/bin/artemis stop
   ```

b. ブローカーがサービスとして Windows で実行されている場合:

```
BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
```

2. **BROKER_INSTANCE_DIR/etc** の下にある **login.config** 設定ファイルを開きます。

3. **amqp-sasl-gssapi** という名前の設定スコープを **login.config** に追加します。以下の例は、Oracle および OpenJDK バージョンの JDK にある **Krb5LoginModule** の設定を示しています。

注記

Java ベンダーのドキュメントを参照して、**Krb5LoginModule** の完全修飾クラス名とその利用可能なオプションを確認します。

```
amqp-sasl-gssapi { (1)
   com.sun.security.auth.module.Krb5LoginModule required (2)
   isInitiator=false
   storeKey=true
   useKeyTab=true (3)
   principal="amqp/my_broker_host@example.com" (4)
   debug=true;
};
```

(1) デフォルトでは、ブローカーの GSSAPI メカニズム実装は **amqp-sasl-gssapi** という名前の JAAS 設定スコープを使用して Kerberos アクセプター認証情報を取得します。

(2) このバージョンの **Krb5LoginModule** は、Oracle および OpenJDK バージョンの JDK で提供されます。Java ベンダーのドキュメントを参照して、**Krb5LoginModule** の完全修飾クラス名とその利用可能なオプションを確認します。

(3) **Krb5LoginModule** は、プリンシパルの認証時に Kerberos キータブを使用するように設定されます。キータブは、Kerberos 環境からツールを使用して生成されます。Kerberos キータブの生成に関する詳細は、ベンダーのドキュメントを参照してください。

(4) プリンシパルは **amqp/my_broker_host@example.com** に設定されます。この値は、Kerberos 環境で作成されたサービスプリンシパルに対応する必要があります。サービスプリンシパルの作成に関する詳細は、ベンダーのドキュメントを参照してください。

4. ブローカーを起動します。

a. ブローカーが Linux で実行されている場合:

```
BROKER_INSTANCE_DIR/bin/artemis run
```

b. ブローカーがサービスとして Windows で実行されている場合:

```
BROKER_INSTANCE_DIR\bin\artemis-service.exe start
```

関連情報

AMQ Broker で GSSAPI メカニズムを有効にする方法の詳細は、ネットワーク接続と Kerberos を参照してください。

6.4.2.1. 代わりの設定スコープの使用

AMQP アクセプターの URL にパラメーター **saslLoginConfigScope** を追加して、 別の設定スコープを指定できます。以下の設定例では、**saslLoginConfigScope** パラメーターに **alternative-sasl-gssapi** の値が指定されています。結果は、**BROKER_INSTANCE_DIR/etc/login.config** で宣言される **alternative-sasl-gssapi** という名前の代わりのスコープを使用するアクセプターです。

```
<acceptor name="amqp">
tcp://0.0.0.0:5672?
protocols=AMQP;saslMechanisms=GSSAPI,PLAIN;saslLoginConfigScope=alternative-sasl-gssapi`
</acceptor>
```

6.4.3. Kerberos 認証情報を使用したクライアントの承認

AMQ Broker は、ロールをマッピングするときに他のセキュリティーモジュールで使用できるように、JAAS **Krb5LoginModule** の実装とともにパッケージ化されています。モジュールは、Kerberos 認証されていないピアプリンシパルを AMQ Broker UserPrincipal として設定された Subject のプリンシパルに追加します。その後、認証情報は **PropertiesLoginModule** または **LDAPLoginModule** モジュールに渡して、Kerberos 認証のピアプリンシパルを AMQ Broker ロールにマップできます。

注記

Kerberos ピアプリンシパルはロールメンバーとしてのみ存在し、ブローカーユーザーとしては存在しません。

前提条件

Kerberos セキュリティー認証情報を使用して AMQP 接続を承認する前に、アクセプターの GSSAPI メカニズムを有効にする必要があります。

手順

1. ブローカーを停止します。
   a. ブローカーが Linux で実行されている場合:

      BROKER_INSTANCE_DIR/bin/artemis stop

   b. ブローカーがサービスとして Windows で実行されている場合:

      BROKER_INSTANCE_DIR\bin\artemis-service.exe stop

2. **BROKER_INSTANCE_DIR/etc** の下にある **login.config** 設定ファイルを開きます。
3. AMQ Broker **Krb5LoginModule** および **LDAPLoginModule** の設定を追加します。

注記

LDAP プロバイダーのドキュメントを参照して、設定オプションを確認します。

```
org.apache.activemq.artemis.spi.core.security.jaas.Krb5LoginModule required
  ; (1)
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
;
```

1 このバージョンの **Krb5LoginModule** は AMQ Broker と共に配布され、Kerberos ID を他の AMQ モジュールでロールマッピングに使用できるブローカー ID に変換します。

4. ブローカーを起動します。

   a. ブローカーが Linux で実行されている場合:

   ```
   BROKER_INSTANCE_DIR/bin/artemis run
   ```

   b. ブローカーがサービスとして Windows で実行されている場合:

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe start
   ```

関連情報
AMQ Broker で GSSAPI メカニズムを有効にする方法の詳細は、ネットワーク接続と Kerberos を参照してください。

PropertiesLoginModule の詳細は、ユーザーとロール を参照してください。

LDAPLoginModule の詳細は、LDAP との統合を 参照してください。

6.5. 設定ファイルのパスワードの暗号化

デフォルトでは、AMQ Broker はすべてのパスワードをプレーンテキストとして設定ファイルに保存します。承認されていないアクセスを防ぐために、適切なパーミッションですべての設定ファイルのセキュリティーを保護するようにしてください。また、プレーンテキストのパスワードを暗号化するか、マスクして、不要なビューアーが読み込まれないようにすることもできます。

マスクされたパスワードは、プレーンテキストのパスワードを暗号化したものです。暗号化バージョンは、AMQ Broker によって提供される **mask** コマンドラインユーティリティーによって生成されます。**mask** ユーティリティーの詳細は、コマンドラインのヘルプドキュメントを参照してください。

```
$ BROKER_INSTANCE_DIR/bin/artemis help mask
```

パスワードをマスクするには、プレーンテキストの値を暗号化された値に置き換えます。マスクされたパスワードは、実際の値が必要になったときに復号化されるように、識別子 **ENC()** でラップする必要があります。

以下の例では、**BROKER_INSTANCE_DIR/etc/bootstrap.xml** 設定ファイルには、**keyStorePassword** および **trustStorePassword** 属性のマスクされたパスワードが含まれます。

bootstrap.xml を使用した例

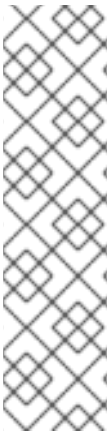
```
<web bind="https://localhost:8443" path="web"
    keyStorePassword="ENC(-342e71445830a32f95220e791dd51e82)"
    trustStorePassword="ENC(32f94e9a68c45d89d962ee7dc68cb9d1)">
   <app url="activemq-branding" war="activemq-branding.war"/>
</web>
```

以下の設定ファイルでは、マスクされたパスワードのみを使用できます。

サポートされている設定ファイル

- broker.xml
- bootstrap.xml
- management.xml
- artemis-users.properties
- login.config(**LDAPLoginModule** で使用)

設定ファイルは **BROKER_INSTANCE_DIR/etc** にあります。



注記

artemis-users.properties は、ハッシュ化されたパスワードをマスクする場合のみサポートします。ブローカーの作成時にユーザーが作成されると、**artemis-users.properties** にはデフォルトでハッシュ化されたパスワードが含まれます。デフォルトの **PropertiesLoginModule** は **artemis-users.properties** ファイルのパスワードを復号化しませんが、代わりに入力をハッシュ化してパスワード検証の 2 つのハッシュ値を比較します。ハッシュされたパスワードをマスクされたパスワードに変更すると、AMQ コンソールにアクセスできなくなります。

broker.xml、**bootstrap.xml**、**management.xml**、および **login.config** はマスクされているが、ハッシュ化されていないパスワードをサポートします。

手順

例として、次の手順では、ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** にある **cluster-password** 設定要素の値をマスクする方法について説明します。

1. コマンドプロンプトで、**mask** ユーティリティーを使用してパスワードを暗号化します。

   ```
   $ BROKER_INSTANCE_DIR/bin/artemis mask PASSWORD
   ```

   暗号化されたパスワードが画面に表示されます。

   ```
   result: 3a34fd21b82bf2a822fa49a8d8fa115d
   ```

2. マスクするプレーンテキストパスワードを含む設定ファイルを開きます。

```
<cluster-password>
  PASSWORD
</cluster-password>
```

3. プレーンテキストのパスワードを、手順 1 で作成した暗号化された値に置き換えます。

```
<cluster-password>
  3a34fd21b82bf2a822fa49a8d8fa115d
</cluster-password>
```

4. 暗号化された値を識別子 **ENC()** でラップします。

```
<cluster-password>
  ENC(3a34fd21b82bf2a822fa49a8d8fa115d)
</cluster-password>
```

設定ファイルには、マスクされたパスワードが含まれるようになりました。パスワードは **ENC()** 識別子でラップされるため、AMQ Broker では使用前にこれを復号化します。

関連情報

AMQ Broker に含まれる設定ファイルの詳細は、AMQ Broker 設定ファイルおよび場所 を参照してください。

6.6. 検証済みユーザーからのメッセージの追跡

メッセージの発信元の追跡およびロギングを有効にするには (セキュリティー監査の目的など)、**_AMQ_VALIDATED_USER** メッセージキーを使用できます。

broker.xml 設定ファイルで **populate-validated-user** オプションが **true** に設定されている場合には、ブローカーは **_AMQ_VALIDATED_USER** キーを使用して検証済みユーザーの名前をメッセージに追加します。JMS および STOMP クライアントの場合には、このメッセージキーは **JMSXUserID** キーにマッピングされます。

注記

ブローカーは、AMQP JMS クライアントによって生成されたメッセージに、検証されたユーザー名を追加できません。クライアントによって送信された AMQP メッセージのプロパティーを変更すると、AMQP プロトコルの違反があります。

自分の SSL 証明書を基に認証されるユーザーの場合、ブローカーによって設定される検証されたユーザー名は、証明書の識別名 (DN) マップの名前です。

broker.xml 設定ファイルで **security-enabled** が **false** で、**populate-validated-user** が **true** の場合に、ブローカーはクライアントが指定するユーザー名 (ある場合) を指定します。**populate-validated-user** オプションのデフォルトは **false** です。

メッセージの送信時に、クライアントによりユーザー名 (つまり、**JMSXUser ID** キー) がまだ入力されていないメッセージを拒否するようにブローカーを設定できます。ブローカーはこれらのクライアントによって送信されたメッセージについて検証されたユーザー名自体を設定できないため、このオプションが AMQP クライアントに役立ちます。

クライアントによって **JMSXUserID** が設定されていないメッセージを拒否するようにブローカーを設定するには、以下の設定を **broker.xml** 設定ファイルに追加します。

```
<reject-empty-validated-user>true</reject-empty-validated-user>
```

デフォルトでは、**reject-empty-validated-user** は **false** に設定されます。

6.7. セキュリティーの無効化

セキュリティーはデフォルトで**有効**になっています。ブローカーのセキュリティーは、**broker.xml** 設定ファイルの **<core>** 要素に **<security-enabled>** パラメーターを設定して有効または無効にすることができます。

手順

1. **broker.xml** ファイルを開きます。
2. **<security-enabled>** パラメーターを見つけます。
3. 必要に応じてエントリーを編集します。
   - セキュリティーを無効にするには、パラメーターを **false** に設定します。

     ```
     <security-enabled>false</security-enabled>
     ```
4. 必要に応じて、**security-invalidation-interval** エントリー (セキュリティーで保護されたログインを定期的に無効にする) を別の値 (**ミリ秒単位**) に変更します。デフォルト値は **10000** です。

6.8. カスタムセキュリティーマネージャーの使用

ブローカーは、**セキュリティーマネージャー** と呼ばれるコンポーネントを使用して認証および承認を処理します。デフォルトでは、AMQ Broker は **org.apache.activemq.artemis.spi.core.security.ActiveMQJAASSecurityManager** セキュリティーマネージャーを使用します。このデフォルトのセキュリティーマネージャーは、JAAS および Red Hat JBoss Enterprise Application Platform(JBoss EAP) のセキュリティーとの統合を提供します。

ただし、システム管理者は、ブローカーセキュリティーの実装をより詳細に制御する必要がある場合があります。このような場合には、ブローカー設定でカスタムセキュリティーマネージャーを指定できます。カスタムセキュリティーマネージャーは、**org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager5** インターフェイスを実装するユーザー定義のクラスです。

6.8.1. カスタムセキュリティーマネージャーの指定

以下の手順では、ブローカー設定でカスタムセキュリティーマネージャーを指定する方法を説明します。

手順

1. **<broker-instance-dir>/etc/boostrap.xml** 設定ファイルを編集します。
2. **security-manager** 要素の **class-name** 属性に、**org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager3** インターフェイスのユーザー定義の実装であるクラスを指定します。以下に例を示します。

   ```
   <broker xmlns="http://activemq.org/schema">
   ```

```
    <security-manager class-name="com.foo.MySecurityManager">
      <property key="myKey1" value="myValue1"/>
      <property key="myKey2" value="myValue2"/>
    </security-manager>

    ...
</broker>
```

関連情報

- **org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager3** インターフェイスの詳細は、ActiveMQ Artemis Core JMS API ドキュメントの Interface ActiveMQSecurityManager3 を参照してください。

6.8.2. カスタムセキュリティーマネージャーのサンプルプログラムの実行

AMQ Broker には、カスタムセキュリティーマネージャーの実装方法を実証するサンプルプログラムが含まれています。この例では、カスタムセキュリティーマネージャーは認証および承認の詳細をログに記録してから、その詳細を **org.apache.activemq.artemis.spi.core.security.ActiveMQJAASSecurityManager** (デフォルトのセキュリティーマネージャー) のインスタンスに渡します。

以下の手順は、カスタムセキュリティーマネージャーのサンプルプログラムを実行する方法を示しています。

前提条件

- AMQ Broker サンプルプログラムを実行するようにマシンを設定する必要があります。詳細は、AMQ Broker の例の実行 を参照してください。

手順

1. カスタムセキュリティーマネージャーの例が含まれるディレクトリーに移動します。

```
$ cd <install-dir>/examples/features/standard/security-manager
```

2. サンプルを実行します。

```
$ mvn verify
```

注記

サンプルプログラムの実行時にブローカーインスタンスを手動で作成して起動する場合は、前の手順のコマンドを **mvn -PnoServer verify** に置き換えてください。

第7章 メッセージの永続化

本章では、AMQ Broker で永続性がどのように機能するか、および設定方法を説明します。

ブローカーには 2 つの永続性オプションが同梱されています。

1. ジャーナルベース
   デフォルトです。ファイルシステムのジャーナルにメッセージを書き込む非常に高性能なオプション。

2. JDBC ベース
   ブローカーの JDBC Store を使用して、メッセージを選択したデータベースに永続化します。

または、ゼロ永続化 のブローカーを設定することもできます。

ブローカーは、メッセージジャーナル以外で大きなメッセージを永続化するために別のソリューションを使用します。詳細は、Large Messages の使用 を参照してください。ブローカーは、メモリー不足の状況でメッセージをディスクにページングするように設定することもできます。詳細は、ページングメッセージを参照してください。

注記

サポートされるデータベースおよびネットワークファイルシステムに関する現在の情報は、Red Hat カスタマーポータルのRed Hat AMQ 7 Supported Configurations を参照してください。

7.1. ジャーナルベースの永続性

ブローカージャーナルは、ディスク上の **append-only** のファイルセットです。各ファイルは固定サイズに事前作成され、最初にパディングが入力されます。メッセージング操作がブローカーで実行されると、レコードがジャーナルの最後に追加されます。レコードを追加すると、ブローカーはディスクヘッドの移動およびランダムアクセス操作を最小限に抑えることができます。これは通常、ディスク上で最も遅い操作です。1 つのジャーナルファイルがいっぱいになると、ブローカーは新しいジャーナルファイルを使用します。

ジャーナルファイルサイズは設定可能です。各ファイルによって使用されるディスクリプラーの数を最小限に抑えることができます。最新のディスクトポロジーは複雑で、ブローカーはファイルのマッピング先の cylinder を制御することはできません。したがって、ジャーナルファイルのサイジングは、正確な情報ではありません。

その他の永続性関連の機能には、以下が含まれます。

- 特定のジャーナルファイルがまだ使用中かどうかを判断する洗練されたファイルのガベージコレクションアルゴリズム。そうでない場合は、ファイルを回収して再利用できます。

- ジャーナルからデッドスペースを削除してデータを圧縮する圧縮アルゴリズム。これにより、ジャーナルがディスク上のファイル数より小さくなります。

- ローカルトランザクションのサポート。

- AMQ JMS クライアントを使用する場合の XA トランザクションのサポート。

ジャーナルの大半は Java で書かれています。ただし、実際のファイルシステムの操作は抽象化されるため、プラグ可能な実装を各種使用できます。AMQ Broker には、2 つの実装が同梱されています。

- Java NIO

ファイルシステムとのインターフェイスに標準の Java NIO を使用します。これにより、非常に優れたパフォーマンスを実現し、Java 6 以降のランタイムを備えたプラットフォーム上で稼働します。

- Linux 非同期 IO
  シンネイティブラッパーを使用して Linux 非同期 IO ライブラリー (AIO) と通信します。AIO では、ブローカーはデータがディスクの作成後に呼び出されるので、同期を明示的に回避できます。デフォルトでは、ブローカーは AIO ジャーナルの使用を試み、AIO が利用できない場合は NIO を使用するようにフォールバックします。

  通常、AIO を使用すると、Java NIO を使用するよりもパフォーマンスが向上されます。libaio のインストール方法は、AIO ジャーナルの使用 を参照してください。

注記

サポートされるネットワークファイルシステムに関する現在の情報は、Red Hat カスタマーポータルの Red Hat AMQ 7 Supported Configurations を参照してください。

7.1.1. AIO の使用

Java NIO ジャーナルは非常にパフォーマンスが高くなりますが、Linux カーネル 2.6 以降を使用してブローカーを実行している場合、Red Hat は永続性パフォーマンスを強化するために AIO ジャーナルを使用することを推奨します。他のオペレーティングシステムや、それ以前のバージョンの Linux カーネルで AIO ジャーナルを使用することはできません。

AIO ジャーナルを使用するには、**libaio** がインストールされていない場合はインストールする必要があります。

手順

- 以下の例のように、**yum** コマンドを使用して **libaio** をインストールします。

```
yum install libaio
```

7.2. ジャーナルベースの永続性の設定

永続性の設定は、**BROKER_INSTANCE_DIR/etc/broker.xml** ファイルで維持されます。ブローカーのデフォルト設定はジャーナルベースの永続化を使用し、以下の要素が含まれます。

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

persistence-enabled

メッセージ永続性にファイルベースのジャーナルを使用するかどうかを指定します。

journal-type
: 使用するジャーナルのタイプ。**ASYNCIO** に設定されている場合には、ブローカーはまず AIO の使用を試行します。ASYNCIO が見つからなかった場合、ブローカーは NIO にフォールバックします。

bindings-directory
: バインディングジャーナルのファイルシステムの場所。デフォルト設定は **BROKER_INSTANCE_DIR** に対して相対的です。

journal-directory
: メッセージジャーナルのファイルシステムの場所。デフォルト設定は **BROKER_INSTANCE_DIR** に対して相対的です。

journal-datasync
: **fdatasync** を使用してディスクへの書き込みを確認するかどうかを指定します。

journal-min-files
: ブローカーの起動時に作成するジャーナルファイルの数。

journal-pool-files
: 未使用のファイルを回収した後に保持するファイルの数。デフォルト値の -1 は、クリーンアップ中にファイルが削除されないことを意味します。

7.2.1. メッセージジャーナル

メッセージジャーナルは、メッセージ自体や重複 ID キャッシュなど、メッセージ関連のデータをすべて保存します。このジャーナルのファイルには **activemq-data** という接頭辞が付けられます。各ファイルには **amq** 拡張子があり、デフォルトサイズは **10485760** バイトです。メッセージジャーナルの場所は、**journal-directory** 設定要素を使用して設定されます。デフォルト値は **BROKER_INSTANCE_DIR/data/journal** です。デフォルト設定には、メッセージングジャーナルに関連する他の要素が含まれます。

- **journal-min-files**
  ブローカーの起動時に事前作成するジャーナルファイルの数。デフォルトは **2** です。

- **journal-pool-files**
  未使用のファイルを回収した後に保持するファイルの数。デフォルト値の **-1** は、ブローカーによって作成されるとファイルが削除されることを意味します。ただし、システムは無限に拡張できないため、この方法でバインドされていない宛先のページングを使用する必要があります。詳細は、メッセージのページングの章を参照してください。

メッセージングジャーナルには、他にもいくつかの設定要素を使用できます。全一覧の付録を参照してください。

7.2.2. バインディングジャーナル

バインディングジャーナルは、サーバーにデプロイされたキューのセットや属性などのバインディング関連のデータを保存するために使用されます。また、ID シーケンスカウンターなどのデータも格納します。

バインディングジャーナルは常に NIO を使用します。これは通常、メッセージジャーナルと比べるとスループットが低くなるためです。このジャーナルのファイルには **activemq-bindings** という接頭辞が付けられます。各ファイルには **bindings** 拡張子があり、デフォルトサイズは **1048576** バイトです。

BROKER_INSTANCE_DIR/etc/broker.xml の設定要素を使用して、バインディングジャーナルを設定します。

- **bindings-directory**
  これは、バインディングジャーナルが配置されるディレクトリーです。デフォルト値は **BROKER_INSTANCE_DIR/data/bindings** です。

- **create-bindings-dir**
  これを **true** に設定すると、bindings ディレクトリーが存在しない場合は、**bindings-directory** で指定された場所にバインディングディレクトリーが自動的に作成されます。デフォルト値は **true** です。

7.2.3. JMS ジャーナル

JMS ジャーナルは、JMS キュー、トピック、接続ファクトリー、ならびにこれらのリソースの JNDI バインディングを含む JMS 関連のデータをすべて格納します。管理 API で作成された JMS リソースはこのジャーナルに永続化されますが、設定ファイルを介して設定されたリソースは永続化されません。JMS ジャーナルは、JMS が使用されている場合にのみ作成されます。

このジャーナルのファイルには **activemq-jms** という接頭辞が付けられます。各ファイルには **jms** 拡張子があり、デフォルトサイズは **1048576** バイトです。

JMS ジャーナルはバインディングジャーナルと設定を共有します。

7.2.4. ジャーナルファイルの圧縮

AMQ Broker には、ジャーナルからデッド領域を削除し、データの圧縮解除によりディスク領域が少なくなるようにする圧縮アルゴリズムが含まれています。コンパクションを開始するタイミングを決定するには、2 つの条件が使用されます。両方の条件が満たされると、コンパクションプロセスはジャーナルを解析し、期限切れのレコードをすべて削除します。そのため、ジャーナルにはファイルが少なくなります。条件は次のとおりです。

- ジャーナル用に作成されたファイルの数。
- ジャーナルファイルのライブデータの割合。

BROKER_INSTANCE_DIR/etc/broker.xml で両方の条件を設定します。

手順

- コンパクションプロセスの条件を設定するには、以下の例のように以下の 2 つの要素を追加します。

```
<configuration>
 <core>
  ...
  <journal-compact-min-files>15</journal-compact-min-files> 1
  <journal-compact-percentage>25</journal-compact-percentage> 2
  ...
 </core>
</configuration>
```

1 コンパクションの開始前に作成されたファイルの最小数。少なくとも **journal-compact-min-files** が設定されない限り、圧縮アルゴリズムは開始されません。デフォルト値は **10** です。これを **0** に設定すると、ジャーナルが無限に増加する可能性があるため、コンパク

ションが無効になります。

2 ジャーナルファイルのライブデータの割合。ライブデータがこの割合より少ないとみなされた場合は、圧縮が開始されます。ジャーナルに少なくとも **journal-compact-min-files** データファイルがあるまで、圧縮は開始されないことに注意してください。デフォルト値は **30** です。

CLI を使用したジャーナルの圧縮
また、コマンドラインインターフェイス (CLI) を使用してジャーナルを圧縮することもできます。

手順

1. **BROKER_INSTANCE_DIR** の所有者としてブローカーを停止します。以下の例では、AMQ Broker のインストール時にユーザー **amq-broker** が作成されています。

```
su - amq-broker
cd __BROKER_INSTANCE_DIR__/bin
$ ./artemis stop
```

2. (オプション) 以下の CLI コマンドを実行して、データツールのパラメーターの全一覧を取得します。デフォルトでは、このツールは **BROKER_INSTANCE_DIR/etc/broker.xml** にある設定を使用する点に注意してください。

```
$ ./artemis help data compact.
```

3. 以下の CLI コマンドを実行して、データを圧縮します。

```
$ ./artemis data compact.
```

4. ツールがデータを正常に圧縮したら、ブローカーを再起動します。

```
$ ./artemis run
```

関連情報

AMQ Broker には、ジャーナルファイル管理用の CLI コマンドが多数含まれています。詳細は、付録の コマンドラインツール を参照してください。

7.2.5. ディスク書き込みキャッシュの無効化

ほとんどのディスクには、ハードウェア書き込みキャッシュが含まれます。書き込みキャッシュは、後でディスクに遅延書き込みされるため、ディスクの見かけのパフォーマンスを向上させることができます。多くのシステムでは、ディスク書き込みキャッシュがデフォルトで有効になっています。つまり、オペレーティングシステムから同期した後であっても、データが実際にディスクに書き込まれる保証はありません。したがって障害が発生した場合は、重大なデータが失われることがあります。

一部の高価なディスクには、非揮発性、またはバッテリー駆動の書き込みキャッシュがあります。これらを使用した場合は、障害発生時に必ずしもデータが失われるわけではありませんが、テストが必要になります。ディスクにこのような機能がない場合は、書き込みキャッシュを必ず無効にする必要があります。ディスク書き込みキャッシュを無効にすると、パフォーマンスに悪影響を及ぼす可能性があることに注意してください。

手順

- Linux の場合は、IDE ディスク用の **hdparm** ツール、あるいは SDSI/SATA ディスク用の **sdparm** または **sginfo** で、ディスクの書き込みキャッシュ設定を管理します。
- Windows では、ディスクを右クリックし、**Properties** をクリックしてキャッシュ設定を管理します。

7.3. JDBC 永続性の設定

JDBC 永続ストアは JDBC 接続を使用してメッセージとバインディングデータをデータベーステーブルに保存します。テーブルのデータは AMQ Broker ジャーナルエンコーディングを使用してエンコードされます。サポートされるデータベースの詳細は、Red Hat カスタマーポータルの Red Hat AMQ 7 Supported Configurations を参照してください。

注記

管理者は、組織の IT インフラストラクチャーの要件に基づいて、メッセージデータをデータベースに保管できます。ただし、データベースを使用すると、メッセージングシステムのパフォーマンスに悪影響を及ぼす可能性があります。具体的には、JDBC 経由でメッセージングデータをデータベーステーブルに書き込むと、ブローカーのパフォーマンスに大きなオーバーヘッドが発生します。

手順

1. 適切な JDBC クライアントライブラリーをブローカーランタイムに追加します。これを行うには、関連する jar を **BROKER_INSTANCE_DIR/lib** ディレクトリーに追加します。
2. 以下の例のように、**BROKER_INSTANCE_DIR/etc/broker.xml** 設定ファイルの **core** 要素の下に **store** 要素を作成します。

```
<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>BINDINGS_TABLE</bindings-table-name>
    <message-table-name>MESSAGE_TABLE</message-table-name>
    <large-message-table-name>LARGE_MESSAGES_TABLE</large-message-table-
name>
    <page-store-table-name>PAGE_STORE_TABLE</page-store-table-name>
    <node-manager-store-table-name>NODE_MANAGER_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>
   </database-store>
  </store>
 </core>
</configuration>
```

jdbc-connection-url

データベースサーバーの完全な JDBC 接続 URL。接続 URL には、すべての設定パラメーターとデータベース名が含まれている必要があります。

jdbc-user

データベースサーバー用に暗号化されたユーザー名。設定ファイルで使用するユーザー名とパスワードの暗号化の詳細は、設定ファイルのパスワードの暗号化 を参照してください。

jdbc-password

データベースサーバー用に暗号化されたパスワード。設定ファイルで使用するユーザー名とパスワードの暗号化の詳細は、設定ファイルのパスワードの暗号化 を参照してください。

bindings-table-name

データのバインディングが保存されるテーブルの名前。このテーブル名を指定すると、干渉なしに複数のサーバー間で単一のデータベースを共有できます。

message-table-name

メッセージデータが保存されるテーブルの名前。このテーブル名を指定すると、干渉なしに複数のサーバー間で単一のデータベースを共有できます。

large-message-table-name

サイズの大きいメッセージと関連データが永続化されるテーブルの名前。さらに、クライアントがサイズの大きいメッセージをチャンクでストリーミングする場合に、チャンクはこのテーブルに保存されます。このテーブル名を指定すると、干渉なしに複数のサーバー間で単一のデータベースを共有できます。

page-store-table-name

ページストアディレクトリー情報が格納されるテーブルの名前。このテーブル名を指定すると、干渉なしに複数のサーバー間で単一のデータベースを共有できます。

node-manager-store-table-name

共有ストア高可用性 (HA) がライブおよびバックアップブローカー向けにロックされ、他の HA 関連のデータがブローカーサーバーに保存されているテーブルの名前。このテーブル名を指定すると、干渉なしに複数のサーバー間で単一のデータベースを共有できます。共有ストア HA を使用する各ライブ/バックアップペアは、同じテーブル名を使用する必要があります。複数の (および関連性のない) ライブ/バックアップペア間で同じテーブルを共有できません。

jdbc-driver-class-name

JDBC データベースドライバーの完全修飾クラス名。サポートされるデータベースの詳細は、Red Hat カスタマーポータルの Red Hat AMQ 7 Supported Configurations を参照してください。

jdbc-network-timeout

JDBC ネットワーク接続のタイムアウト (ミリ秒単位)。デフォルト値は 20000 ミリ秒 (つまり、20 秒) です。共有ストア HA に JDBC を使用する場合は、**jdbc-lock-expiration** 未満の値にタイムアウトを設定することが推奨されます。

jdbc-lock-renew-period

現在の JDBC ロックの更新期間の長さ (ミリ秒単位)。この時間が経過すると、ブローカーはロックを更新できます。デフォルト値は 2000 ミリ秒 (つまり、2 秒) です。

jdbc-lock-expiration

jdbc-lock-renew-period が経過する場合でも、現在の JDBC ロックがアクティブなとみなされる時間 (ミリ秒単位)。このプロパティーを **jdbc-lock-renew-period** よりも大きな値に設定すると、ロックを所有するブローカーが更新中に予期しない遅延が発生してもすぐにロックが失われないようにします。有効期限の経過後、現在 JDBC ロックを所有しているブローカーによって JDBC ロックが更新されない場合、別のブローカーが JDBC ロックを確立できます。デフォルト値は 20000 ミリ秒 (つまり、20 秒) です。

jdbc-journal-sync-period

ブローカージャーナルが JDBC と同期する期間 (ミリ秒単位)。デフォルト値は 5 ミリ秒です。

7.4. ゼロ永続化の設定

メッセージングシステムに、ゼロ永続化が必要になる場合があります。ゼロ永続化を実行するためのブローカーを設定することは簡単です。**BROKER_INSTANCE_DIR/etc/broker.xml** の **persistence-enabled** パラメーターを **false** に設定します。

このパラメーターを false に設定すると、**ゼロ** 永続化が発生することに注意してください。つまり、バインディングデータ、メッセージデータ、大きなメッセージデータ、重複した ID キャッシュまたはページングデータが永続化されません。

第8章 ページングメッセージ

AMQ Broker は、サーバーが制限されたメモリーで実行されている間、数百万のメッセージが含まれる大規模なキューを透過的にサポートします。

そのため、すべてのキューを一度にメモリーに保存することはできません。そのため、AMQ Broker は必要に応じて**ページ**メッセージを透過的にメモリーに出し入れするため、メモリーフットプリントの大きいキューが可能になります。

ページングはアドレスごとに個別に行われます。AMQ Broker は、アドレスのすべてのメッセージサイズが設定済みの最大サイズを超えると、メッセージをディスクにページングします。アドレスの詳細は、アドレス、キュー、およびトピック を参照してください。

デフォルトでは、AMQ Broker はメッセージをページングしません。ページングを明示的に設定して有効にする必要があります。

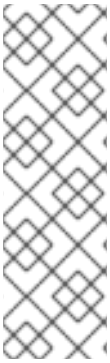
AMQ Broker での **paging** の使用方法を示す作業例は、**INSTALL_DIR/examples/standard/** にあるページングの例を参照してください。

8.1. ページファイルについて

メッセージはファイルシステムのアドレスごとに保存されます。各アドレスには、メッセージが複数のファイル (ページファイル) に保存される個別のフォルダーがあります。各ファイルには、最大に設定されたサイズ (**page-size-bytes**) までのメッセージが含まれます。システムは必要に応じてファイルを移動し、すべてのメッセージがそのポイントに確認応答されるとすぐにページファイルが削除されます。

ブラウザーは page-cursor システムを通じて読み込まれます。

セレクターを持つコンシューマーもページファイルを検索し、基準に見合わないメッセージを無視します。



注記

キューがあり、非常に制限のあるセレクターでキューをフィルターすると、キューからメッセージを消費するまで、ページングからさらにデータを読み取ることができない状況になる可能性があります。

例: あるコンシューマーでは、セレクターを 'color="red"' として作成しますが、青色のメッセージが 100 万件しか赤くなると、青色を消費するまで赤い量を消費できません。これは、キュー全体がメッセージを検索する際にブラウジングするのに対し、キューを提供しながらページを表示するためです。

8.2. ページングディレクトリーの場所の設定

ページングディレクトリーの場所を設定するには、以下の例のように **paging-directory** 設定要素をブローカーのメイン設定ファイル **BROKER_INSTANCE_DIR/etc/broker.xml** に追加します。

```
<configuration ...>
  ...
  <core ...>
    <paging-directory>/somewhere/paging-directory</paging-directory>
    ...
  </core>
</configuration>
```

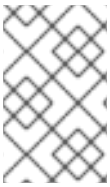
AMQ Broker は、設定された場所でページされる各アドレスに 1 つのディレクトリーを作成します。

8.3. ページング用のアドレスの設定

ページングの設定は、以下の例のように特定の **address-settings** に要素を追加して、アドレスレベルで行われます。

```
<address-settings>
  <address-setting match="jms.paged.queue">
    <max-size-bytes>104857600</max-size-bytes>
    <page-size-bytes>10485760</page-size-bytes>
    <address-full-policy>PAGE</address-full-policy>
  </address-setting>
</address-settings>
```

上記の例では、アドレス **jms.paged.queue** に送信されたメッセージがメモリーの **104857600** バイトを超えると、ブローカーはページングを開始します。



注記

ページングはアドレスごとに個別に行われます。アドレスに **max-size-bytes** を指定した場合、一致する各アドレスは指定した最大サイズを超えません。一致するアドレスすべての合計サイズが **max-size-bytes** に制限されるわけではありません。

これは、アドレス設定で利用可能なパラメーターの一覧です。

表8.1 ページング設定要素

要素名	説明	デフォルト
max-size-bytes	ブローカーがページモードに入る前にアドレスで許可されるメモリーの最大サイズ。	-1 (無効) このパラメーターが無効になると、ブローカーは **global-max-size** をページングのメモリー使用制限として使用します。詳細は、「グローバルページングサイズの設定」 を参照してください。
page-size-bytes	ページングシステムで使用される各ページファイルのサイズ。	10MiB (10 * 1024 * 1024 バイト)

要素名	説明	デフォルト
address-full-policy	有効な値は **PAGE**、**DROP**、**BLOCK**、および **FAIL** です。値が **PAGE** の場合、追加のメッセージはディスクにページングされます。値が **DROP** の場合、追加のメッセージは通知なしで破棄されます。値が **FAIL** の場合、メッセージは破棄され、クライアントメッセージプロデューサーが例外を受け取ります。値が **BLOCK** の場合、追加のメッセージを送信しようとすると、クライアントメッセージプロデューサーがブロックします。	PAGE
page-max-cache-size	ページングナビゲーション中に IO を最適化するために、システムはこの数のページファイルをメモリー内に保持します。	5

8.4. グローバルページングサイズの設定

アドレスごとにメモリー制限を設定することは、ブローカーが異なる使用パターンを持つアドレスを多数管理する場合など、実用的ではないことがあります。このような状況では、**global-max-size** パラメーターを使用して、ブローカーが受信メッセージに関連付けられたアドレスに設定されたページモードに入る前に、ブローカーが使用可能なメモリー量にグローバル制限を設定します。

global-max-size のデフォルト値は、Java 仮想マシン (JVM) で利用可能な最大メモリーの半分です。**broker.xml** 設定ファイルで設定すると、このパラメーターに独自の値を指定できます。**global-max-size** の値はバイト単位ですが、便宜上バイト表記 ("K"、Mb、"GB" など) を使用できます。

以下の手順では、**broker.xml** 設定ファイルで **global-max-size** パラメーターを設定する方法を説明します。

global-max-size パラメーターの設定

手順

1. ブローカーを停止します。
   a. ブローカーが Linux で実行している場合は、以下のコマンドを実行します。

      ```
      BROKER_INSTANCE_DIR/bin/artemis stop
      ```

   b. ブローカーがサービスとして Windows で実行している場合は、以下のコマンドを実行します。

      ```
      BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
      ```

2. **BROKER_INSTANCE_DIR/etc** にある **broker.xml** 設定ファイルを開きます。
3. **global-max-size** パラメーターを **broker.xml** に追加し、ブローカーが使用できるメモリーの量をバイト単位で制限します。以下の例のように、**global-max-size** の値にバイト表記 (**K**、**Mb**、**GB**) を使用することもできます。

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

```
    ...
    <global-max-size>1GB</global-max-size>
    ...
  </core>
</configuration>
```

上記の例では、ブローカーはメッセージの処理時に利用可能な最大 1 ギガバイト (**1GB**) を使用するように設定されています。設定した上限を超えると、ブローカーは受信メッセージに関連付けられたアドレスに設定されたページモードに入ります。

4. ブローカーを起動します。

   a. ブローカーが Linux で実行している場合は、以下のコマンドを実行します。

   ```
   BROKER_INSTANCE_DIR/bin/artemis run
   ```

   b. ブローカーがサービスとして Windows で実行している場合は、以下のコマンドを実行します。

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe start
   ```

関連情報

アドレスのページングモードの設定に関する詳細は、「ページング用のアドレスの設定」 を参照してください。

8.5. ページング時のディスク使用量の制限

ブローカーが受信メッセージをブロックする前に、ブローカーが使用する物理ディスクの量を制限することができます。**max-disk-usage** を **broker.xml** 設定ファイルに追加し、ページングメッセージ時にブローカーが使用できるディスク領域の割合の値を指定します。**max-disk-usage** のデフォルト値は **90** です。これは、制限がディスク領域の **90** パーセントに設定されることを意味します。

max-disk-usage の設定

手順

1. ブローカーを停止します。

   a. ブローカーが Linux で実行している場合は、以下のコマンドを実行します。

   ```
   BROKER_INSTANCE_DIR/bin/artemis stop
   ```

   b. ブローカーがサービスとして Windows で実行している場合は、以下のコマンドを実行します。

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
   ```

2. **BROKER_INSTANCE_DIR/etc** にある **broker.xml** 設定ファイルを開きます。

3. **max-disk-usage** 設定要素を追加し、ページングメッセージ時に使用するディスク容量に制限を設定します。

```
<configuration>
```

```
  <core>
    ...
    <max-disk-usage>50</max-disk-usage>
    ...
  </core>
</configuration>
```

上記の例では、ブローカーはメッセージのページング時に **50** パーセントのディスク領域の使用に制限されます。メッセージはブロックされ、ディスクの **50** パーセント後にページングされなくなりました。

4. ブローカーを起動します。

   a. ブローカーが Linux で実行している場合は、以下のコマンドを実行します。

   ```
   BROKER_INSTANCE_DIR/bin/artemis run
   ```

   b. ブローカーがサービスとして Windows で実行している場合は、以下のコマンドを実行します。

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe start
   ```

8.6. メッセージのドロップ方法

最大サイズに達したときにメッセージをページングするのではなく、アドレスが満杯になったときにメッセージをドロップするように設定できます。

これを行うには、アドレス設定で **address-full-policy** を **DROP** に設定します。

8.6.1. メッセージの破棄とプロデューサーへの例外の出力

最大サイズに達したときにメッセージをページングするのではなく、メッセージをドロップするようにアドレスを設定し、アドレスが満杯になるとクライアント側で例外を出力することもできます。

これを行うには、アドレス設定で **address-full-policy** を **FAIL** に設定します。

8.7. プロデューサーをブロックする方法

最大サイズに達したときにメッセージをページングせずに、アドレスが満杯になったときにプロデューサーがさらにメッセージを送信しないように設定することもできます。そのため、サーバーでメモリーが使い切られるのを防ぐことができます。

注記

ブロックは、使用されているプロトコルがサポートする場合にのみ機能します。たとえば、AMQP プロデューサーはブローカーによって送信されるときにブロックパケットを理解しますが、STOMP プロデューサーは処理されません。

サーバー上でメモリーが解放されると、プロデューサーは自動的にブロック解除され、送信を継続できます。

これを行うには、アドレス設定で **address-full-policy** を **BLOCK** に設定します。

デフォルトの設定では、10 MiB のデータがアドレスに含まれていると、すべてのアドレスがプロデューサーをブロックするように設定されています。

8.8. マルチキャストキューを持つアドレスでの注意

マルチキャストキューがバインドされているアドレス (Topic の JMS サブスクリプションなど) にメッセージがルーティングされた場合は、メッセージのコピーはメモリー内に 1 つだけ存在します。各キューは参照のみを処理します。このため、メモリーは、メッセージを参照するすべてのキューがメッセージを配信した後のみ解放されます。

1 つのレイジー (遅延) サブスクリプションがある場合、ページングシステム上の追加のストレージを介して送信されたメッセージがすべてのキューにあるため、アドレス全体が IO パフォーマンスの影響を受けます。

以下に例を示します。

- アドレスにはキューが 10 個ある
- キューの 1 つがメッセージを配信しません (低速なコンシューマーが原因です)。
- メッセージはアドレスに継続的に到達し、ページングが開始されます。
- メッセージが送信された場合でも、他の 9 キューは空です。

この例では、他のすべての 9 キューはページシステムからメッセージを消費します。これにより、これが望ましくない状態である場合にパフォーマンスの問題が発生する可能性があります。

第9章 大きなメッセージの処理

クライアントは、ブローカーの内部バッファーのサイズを超える大きなメッセージを送信する可能性があり、予期せぬエラーが発生する可能性があります。この状態を回避するには、メッセージが指定の最小値よりも大きい場合にメッセージをファイルとして保存するようにブローカーを設定できます。このように大きなメッセージを処理すると、ブローカーはメモリー内にメッセージを保持しません。代わりに、ディスクまたはブローカーが大きなメッセージファイルを保存するデータベーステーブルのディレクトリーを指定します。

ブローカーがメッセージを大きなメッセージとして保存すると、キューは大きなメッセージディレクトリーまたはデータベーステーブルのファイルへの参照を保持します。

大規模なメッセージ処理は、Core、AMQP、OpenWire、および STOMP プロトコルで利用できます。

Core および OpenWire プロトコルの場合、クライアントは接続設定でメッセージの最小サイズを指定します。AMQP および STOMP プロトコルの場合は、ブローカー設定のプロトコルごとに定義されたアクセプターに大きなメッセージの最小サイズを指定します。

注記

大きなメッセージを生成および消費するのに、異なるプロトコルを使用 **しない** ことが推奨されます。これには、ブローカーはメッセージの複数の変換を実行する必要がある場合があります。たとえば、AMQP プロトコルを使用してメッセージを送信し、OpenWire を使用して受信したいとします。この場合、ブローカーは最初に大きなメッセージのボディー全体を読み取り、Core プロトコルを使用するように変換する必要があります。次に、ブローカーは別の変換を実行し、今回は OpenWire プロトコルへ変換する必要があります。このようなメッセージ変換により、ブローカーに大きな処理のオーバーヘッドが発生します。

前述のプロトコルに指定した最小大きなメッセージサイズは、利用可能なディスク領域やメッセージのサイズなどのシステムリソースの影響を受けます。適切なサイズを決定するために、いくつかの値を使用してパフォーマンステストを実行することが推奨されます。

本セクションの手順では以下の方法を説明します。

- 大きなメッセージを格納するようにブローカーを設定します。
- 大きなメッセージ処理のための AMQP および STOMP プロトコルのアクセプター設定

このセクションでは、大きなメッセージと連携する AMQ Core Protocol および AMQ OpenWire JMS クライアントの設定に関する追加リソースへのリンクも提供します。

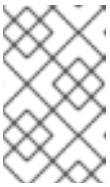
9.1. 大きなメッセージ処理のためのブローカーの設定

以下の手順では、ブローカーが大きなメッセージファイルを保存するディスクまたはデータベーステーブルにディレクトリーを指定する方法を説明します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. ブローカーが大きなメッセージファイルを保存する場所を指定します。
    a. ディスクに大きなメッセージを保存する場合は、**core** 要素に **large-messages-directory** パラメーターを追加し、ファイルシステムの場所を指定します。以下に例を示します。

```
<configuration>
  <core>
    ...
    <large-messages-directory>/path/to/my-large-messages-directory</large-messages-
directory>
    ...
  </core>
</configuration>
```



注記

large-messages-directory の値を明示的に指定しない場合、ブローカーは **<broker-instance-dir> /data/largemessages** のデフォルト値を使用します。

b. 大きなメッセージをデータベーステーブルに保存する場合は、**database-store** 要素に **large-message-table** パラメーターを追加して値を指定します。以下に例を示します。

```
<store>
  <database-store>
    ...
    <large-message-table>MY_TABLE</large-message-table>
    ...
  </database-store>
</store>
```

注記

large-message-table の値を明示的に指定しない場合、ブローカーは **LARGE_MESSAGE_TABLE** のデフォルト値を使用します。

関連情報

- データベースストアの設定に関する詳細は、「JDBC 永続性の設定」 を参照してください。

9.2. 大規模なメッセージ処理のための AMQP アクセプターの設定

以下の手順は、指定したサイズよりも大きい AMQP メッセージを大規模メッセージとして処理するように AMQP アクセプターを設定する方法を説明します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
ブローカー設定ファイルのデフォルトの AMQP アクセプターは以下のようになります。

```
<acceptors>
    ...
    <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;a
mqpCredits=1000;amqpLowCredits=300</acceptor>
    ...
</acceptors>
```

2. デフォルトの AMQP アクセプター (または設定した別の AMQP アクセプター) で、**amqpMinLargeMessageSize** プロパティーを追加し、値を指定します。以下に例を示します。

```
<acceptors>
   ...
   <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;a
mqpCredits=1000;amqpLowCredits=300;amqpMinLargeMessageSize=204800</acceptor>
   ...
</acceptors>
```

上記の例では、ブローカーはポート 5672 で AMQP メッセージを受け入れるように設定されます。**amqpMinLargeMessageSize** の値に基づいて、アクセプターが 204800 バイトよりも大きい AMQP メッセージ (200 キロバイト以上) を受信する場合、ブローカーはメッセージを大きなメッセージとして格納します。このプロパティーの値を明示的に指定しない場合、 ブローカーは 102400 (100 キロバイト) のデフォルト値を使用します。

注記

- **amqpMinLargeMessageSize** を -1 に設定すると、AMQP メッセージに対するサイズの大きいメッセージ処理が無効になります。
- ブローカーが **amqpMinLargeMessageSize** の値を超えない永続的な AMQP メッセージを受信して、これがメッセージングジャーナルバッファーのサイズ (**journal-buffer-size** 設定パラメーターを使用して指定) を **超える** 場合、ブローカーはメッセージをジャーナルに保存する前に大きな Core メッセージに変換します。

9.3. サイズの大きいメッセージ処理向けの STOMP アクセプターの設定

以下の手順は、指定したサイズよりも大きい STOMP メッセージをサイズの大きいメッセージとして処理するように STOMP アクセプターを設定する方法を説明します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
ブローカー設定ファイルのデフォルトの AMQP アクセプターは以下のようになります。

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

2. デフォルトの STOMP アクセプター (または設定した別の STOMP アクセプター) で、**stompMinLargeMessageSize** プロパティーを追加し、値を指定します。以下に例を示します。

```
<acceptors>
   ...
   <acceptor name="stomp">tcp://0.0.0.0:61613?
```

```
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true;
stompMinLargeMessageSize=204800</acceptor>
   ...
</acceptors>
```

前の例では、ブローカーはポート 61613 で STOMP メッセージを受け入れるように設定されています。**stompMinLargeMessageSize** の値に基づいて、アクセプターが 204800 バイトよりも大きい STOMP メッセージ (200 キロバイト以上) を受信する場合、ブローカーはメッセージを大きなメッセージとして格納します。このプロパティーの値を明示的に指定しない場合、 ブローカーは 102400 (100 キロバイト) のデフォルト値を使用します。

注記

大きなメッセージを STOMP コンシューマーに配信するために、ブローカーは自動的にメッセージを大きなメッセージから通常のメッセージに変換してからクライアントに送信します。大きなメッセージが圧縮されている場合、ブローカーはこれを STOMP クライアントに送信する前に圧縮します。

9.4. 大きなメッセージと JAVA クライアント

大きなメッセージを使用するクライアントを作成している Java 開発者には、2 つのオプションがあります。

1 つのオプションとして、**InputStream** および **OutputStream** のインスタンスを使用することができます。たとえば、**FileInputStream** を使用して、物理ディスク上の大きなファイルから作成されたメッセージを送信します。その後、受信者が **FileOutputStream** を使用して、メッセージをローカルファイルシステムの場所にストリーミングできます。

別のオプションとして、JMS **BytesMessage** または **StreamMessage** を直接ストリーミングする方法もあります。以下に例を示します。

```
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
}
```

関連情報

- AMQ Core Protocol JMS クライアントでの大きなメッセージの使用方法は、以下を参照してください。
  - 大型メッセージのオプション
  - ストリーミングされた大きなメッセージへの書き込み
  - ストリームされた大きなメッセージからの読み取り
- AMQ OpenWire JMS クライアントでの大きなメッセージの使用方法は、以下を参照してください。
  - 大型メッセージのオプション

- ストリーミングされた大きなメッセージへの書き込み
- ストリームされた大きなメッセージからの読み取り

- 大きなメッセージの例については、AMQ Broker インストールの **<install_dir>/examples/features/standard/** ディレクトリーの **large-message** の例を参照してください。サンプルプログラムの実行に関する詳細は、AMQ Broker のサンプルプログラムの実行 を参照してください。

第10章 デッド接続の検出

クライアントが予期せずに停止し、それらのリソースをクリーンアップする機会がないことがあります。これが発生すると、リソースが faulty 状態のままになり、ブローカーがメモリー不足または他のシステムリソースで実行されている可能性があります。ブローカーは、ガベージコレクション時にクライアントの接続が適切にシャットダウンされなかったことを検出します。その後、コネクションは閉じられ、以下のようなメッセージがログに書き込まれます。ログは、クライアントセッションがインスタンス化されたコードの正確な行を取得します。これにより、エラーを特定し、これを修正できます。

```
[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 接続がインスタンス化されたクライアントコードの行。

クライアント側からのデッド接続の検出

ブローカーからデータを受信する限り、クライアントは接続がアライブ状態であると判断します。**client-failure-check-period** プロパティーの値を指定して、接続の失敗を確認するようにクライアントを設定します。ネットワーク接続のデフォルトのチェック期間は **30000** ミリ秒 (30 秒) で、仮想マシン間の接続のデフォルト値は **-1** です。これは、データが受信されなかった場合、クライアントがその側から接続を失敗させないことを意味します。

通常、チェック期間はブローカーの接続の Time-to-live に使用される値よりもはるかに低い値に設定し、一時的な障害が発生した場合にクライアントが再接続できるようにします。

以下の例は、Core JMS クライアントを使用してチェック期間を **10000** ミリ秒 (10 秒) に設定する方法を示しています。

手順

- デッド接続を検出するためのチェック期間を設定します。
  - Core JMS クライアントで JNDI を使用している場合は、以下のように JNDI コンテキスト環境 (**jndi.properties**) 内のチェック期間を設定します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?
clientFailureCheckPeriod=10000
```

  - JNDI を使用していない場合は、値を **ActiveMQConnectionFactory.setClientFailureCheckPeriod()** に渡してチェック期間を直接設定します。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setClientFailureCheckPeriod(10000);
```

10.1. 接続 TIME-TO-LIVE

クライアントとサーバー間のネットワーク接続に失敗してオンラインに戻る可能性があるため、AMQ Broker は非アクティブなサーバー側のリソースをクリーンアップします。この待機時間は Time-to-live (TTL) と呼ばれます。ネットワークベースの接続のデフォルトの TTL は **60000** ミリ秒 (1 分) です。in-VM 接続のデフォルトの TTL は **-1** です。つまり、ブローカーはブローカー側で接続をタイムアウトしません。

ブローカーでの Time-To-Live の設定

クライアントが独自の接続 TTL を指定しないようにするには、ブローカー側でグローバル値を設定します。これは、ブローカー設定で **connection-ttl-override** 要素を指定して実行できます。

connection-ttl-check-interval 要素によって決定されるように、TTL 違反の接続をチェックするロジックはブローカーで定期的に実行されます。

手順

- 以下の例のように、**BROKER_INSTANCE_DIR**/etc/broker.xml を編集して **connection-ttl-override** 設定要素を追加し、time-to-live の値を指定します。

```
<configuration>
 <core>
  ...
  <connection-ttl-override>30000</connection-ttl-override> (1)
  <connection-ttl-check-interval>1000</connection-ttl-check-interval> (2)
  ...
 </core>
</configuration>
```

(1) すべての接続のグローバル TTL は **30000** ミリ秒 (30 秒) に設定されます。デフォルト値は **-1** で、クライアントが独自の TTL を設定できるようにします。

(2) デッド接続をチェックする間隔は **1000** ミリ秒 (1 秒) に設定されます。デフォルトでは、チェックは **2000** ミリ秒 (2 秒) ごとに行われます。

クライアント上での Time-To-Live の設定

デフォルトでは、クライアントは独自の接続に TTL を設定できます。以下の例は、Core JMS クライアントを使用して Time-To-Live を設定する方法を示しています。

手順

- クライアント接続に Time-To-Live を設定します。
  - JNDI を使用して接続ファクトリーをインスタンス化する場合は、**connectionTTL** パラメーターを使用して xml 設定に指定できます。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?connectionTTL=30000
```

  - JNDI を使用していない場合は、接続 TTL は **ActiveMQConnectionFactory** インスタンスの **ConnectionTTL** 属性によって定義されます。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setConnectionTTL(30000);
```

10.2. 非同期接続実行の無効化

サーバー側で受信されたパケットの多くは、**リモート** スレッドで実行されます。これらのパケットは短時間実行される操作を表し、パフォーマンス上の理由から常に **リモーティング** スレッドで実行されます。ただし、一部のパケットタイプは **リモーティング** スレッドではなくスレッドプールを使用して実行され、ネットワークのレイテンシーが少し追加されます。

スレッドプールを使用するパケットタイプは、以下に示す Java クラス内に実装されます。クラスはすべて **org.apache.actiinvemq.artemis.core.protocol.core.impl.wireformat** パッケージにあります。

- RollbackMessage
- SessionCloseMessage
- SessionCommitMessage
- SessionXACommitMessage
- SessionXAPrepareMessage
- SessionXARollbackMessage

手順

- 非同期接続実行を無効にするには、以下の例のように **async-connection-execution-enabled** 設定要素を **BROKER_INSTANCE_DIR**/etc/broker.xml に追加し、**false** に設定します。デフォルト値は **true** です。

```
<configuration>
 <core>
  ...
  <async-connection-execution-enabled>false</async-connection-execution-enabled>
  ...
 </core>
</configuration>
```

10.3. クライアント側からの接続を閉じる

クライアントアプリケーションは、デッド接続が発生しないように、終了する前に制御された方法でリソースを閉じる必要があります。Java では、**finally** ブロック内で接続を閉じることが推奨されます。

```
Connection jmsConnection = null;
try {
   ConnectionFactory jmsConnectionFactory =
ActiveMQJMSClient.createConnectionFactoryWithoutHA(...);
   jmsConnection = jmsConnectionFactory.createConnection();
   ...use the connection...
}
finally {
   if (jmsConnection != null) {
```

```
        jmsConnection.close();
    }
}
```

第11章 フロー制御

フロー制御は、プロデューサーとコンシューマー間のデータのフローを制限することで、プロデューサーとコンシューマーの超過を防ぎます。AMQ Broker を使用すると、コンシューマーとプロデューサーの両方のフロー制御を設定できます。

11.1. コンシューマーフロー制御

コンシューマーフロー制御は、クライアントがブローカーからメッセージを消費する際に、ブローカーとクライアント間のデータフローを制御します。AMQ Broker クライアントは、デフォルトでメッセージをバッファーしてからコンシューマーに配信します。バッファーがない場合、クライアントはまず、消費する前にブローカーから各メッセージを要求する必要があります。このタイプのラウンドトリップ通信はコストがかかります。メモリー不足の問題によりコンシューマーがメッセージをすばやく処理できず、バッファーが受信メッセージでオーバーフローを開始するため、クライアント側のデータのフローを制限することが重要になります。

11.1.1. コンシューマーウィンドウサイズの設定

クライアント側のバッファーに保持されるメッセージの最大サイズは、その **ウィンドウサイズ** によって決定されます。AMQ Broker クライアントのウィンドウのデフォルトサイズは 1 MiB または 1024 * 1024 バイトです。ほとんどのユースケースでは、デフォルトでは問題ありません。その他のケースでは、ウィンドウサイズの最適な値を見つけるには、システムのベンチマークが必要になる場合があります。AMQ Broker では、デフォルトを変更する必要がある場合はバッファーウインドウサイズを設定できます。

ウィンドウサイズの設定
以下の例は、Core JMS クライアントを使用する場合にコンシューマーウインドウサイズパラメーターを設定する方法を示しています。それぞれの例では、コンシューマーウィンドウサイズを **300000** バイトに設定します。

手順

- コンシューマーウィンドウサイズを設定します。
  - Core JMS Client が JNDI を使用して接続ファクトリーをインスタンス化する場合は、connection string URL の一部として **consumerWindowSize** パラメーターを含めます。JNDI コンテキスト環境内に URL を保存します。以下の例では、**jndi.properties** ファイルを使用して URL を保存します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?
consumerWindowSize=300000
```

  - Core JMS クライアントが JNDI を使用して接続ファクトリーをインスタンス化しない場合は、値を **ActiveMQConnectionFactory.setConsumerWindowSize()** に渡します。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setConsumerWindowSize(300000);
```

11.1.2. 高速コンシューマーの処理

高速コンシューマーは、メッセージをコンシュームすると同時に処理できます。メッセージングシステムのコンシューマーが高速であると確信できる場合は、ウィンドウサイズを **-1** に設定することを検討

してください。この設定により、クライアント側でバインドされていないメッセージバッファーリングが可能になります。ただし、この設定は注意して使用してください。コンシューマーが受信と同時にメッセージを処理できない場合、クライアント側のメモリーをオーバーフローさせることができます。

高速コンシューマーのウィンドウサイズの設定

手順

以下の例は、メッセージの高速コンシューマーである Core JMS クライアントを使用する場合に、ウィンドウサイズを **-**1 に設定する方法を示しています。

- コンシューマーウィンドウサイズを **-**1 に設定します。
  - Core JMS Client が JNDI を使用して接続ファクトリーをインスタンス化する場合は、connection string URL の一部として **consumerWindowSize** パラメーターを含めます。JNDI コンテキスト環境内に URL を保存します。以下の例では、**jndi.properties** ファイルを使用して URL を保存します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?consumerWindowSize=-1
```

  - Core JMS クライアントが JNDI を使用して接続ファクトリーをインスタンス化しない場合は、値を **ActiveMQConnectionFactory.setConsumerWindowSize()** に渡します。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setConsumerWindowSize(-1);
```

11.1.3. 低速なコンシューマーの処理

低速なコンシューマーは、各メッセージを処理するのにかなり時間がかかります。このような場合は、クライアント側でメッセージをバッファーしないことが推奨されます。メッセージはブローカー側で残り、他のコンシューマーによって消費されます。バッファーをオフにする利点の 1 つは、キュー上の複数のコンシューマー間で確定的な分散を提供することです。クライアント側のバッファーを無効にして低速なコンシューマーを処理するには、ウィンドウサイズを **0** に設定します。

低速なコンシューマーのウィンドウサイズの設定

手順

以下の例は、メッセージの低速なコンシューマーである Core JMS クライアントを使用する場合に、ウィンドウサイズを **0** に設定する方法を示しています。

- コンシューマーウィンドウサイズを **0** に設定します。
  - Core JMS Client が JNDI を使用して接続ファクトリーをインスタンス化する場合は、connection string URL の一部として **consumerWindowSize** パラメーターを含めます。JNDI コンテキスト環境内に URL を保存します。以下の例では、**jndi.properties** ファイルを使用して URL を保存します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?consumerWindowSize=0
```

- Core JMS クライアントが JNDI を使用して接続ファクトリーをインスタンス化しない場合は、値を **ActiveMQConnectionFactory.setConsumerWindowSize()** に渡します。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setConsumerWindowSize(0);
```

関連情報

低速なコンシューマーを処理する場合にコンシューマーをバッファーしないようにブローカーを設定する方法を示す例は、**INSTALL_DIR/examples/standard** の **no-consumer-buffering** の例を参照してください。

11.1.4. メッセージ消費率の設定

コンシューマーがメッセージを消費できるレートを調整できます。スロットリングとしても知られており、消費率は、コンシューマーが設定を許可するよりも高速にメッセージを消費しないようにします。

注記

レート制限のあるフロー制御は、ウィンドウベースのフロー制御と併用できます。レート制限のあるフロー制御は、クライアントが 1 秒以内に消費できるメッセージ数のみに影響し、バッファー内のメッセージ数には影響しません。レート制限が遅く、ウィンドウベースの制限が高いと、クライアントの内部バッファーがメッセージですぐに一杯になります。

この機能を有効にするには、レートは正の整数である必要があります。1 秒あたりのメッセージ単位で指定される必要なメッセージ消費率の最大値です。レートを **-1** に設定すると、レート制限のあるフロー制御が無効になります。デフォルト値は **-1** です。

メッセージ消費率の設定

手順

以下の例では、メッセージの消費速度を毎秒 **10** メッセージに制限する Core JMS クライアントを使用しています。

- コンシューマーレートを設定します。
  - Core JMS Client が JNDI を使用して接続ファクトリーをインスタンス化する場合は、connection string URL の一部として **consumerMaxRate** パラメーターを含めます。JNDI コンテキスト環境内に URL を保存します。以下の例では、**jndi.properties** ファイルを使用して URL を保存します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

java.naming.provider.url=tcp://localhost:61616?consumerMaxRate=10
```

  - Core JMS クライアントが JNDI を使用して接続ファクトリーをインスタンス化しない場合は、値を **ActiveMQConnectionFactory.setConsumerMaxRate()** に渡します。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setConsumerMaxRate(10);
```

関連情報

コンシューマーレートの制限方法の作業例は、**INSTALL_DIR/examples/standard** の **consumer-rate-limit** の例を参照してください。

11.2. プロデューサーフロー制御

コンシューマーウィンドウベースのフロー制御と同様に、AMQ Broker はプロデューサーからブローカーに送信されるデータ量をブローカーに制限し、ブローカーが大量のデータで過負荷にならないようにすることができます。プロデューサーの場合、ウィンドウサイズは 1 度にインフライトにできるバイト数を決定します。

11.2.1. プロデューサーウィンドウサイズの設定

ウィンドウサイズは、クレジットに基づいてブローカーとプロデューサーの間でネゴシエートされます。これは、ウィンドウの各バイトに 1 つのクレジットです。メッセージが送信されてクレジットが使用されると、プロデューサーは追加のメッセージを送信する前に、ブローカーからクレジットを要求し、付与する必要があります。プロデューサーとブローカー間のクレジットの交換により、プロデューサーとブローカー間のデータフローが分離されます。

ウィンドウサイズの設定
以下の例は、Core JMS クライアントを使用する場合にプロデューサーウインドウサイズを **1024** バイトに設定する方法を示しています。

手順

- プロデューサーウィンドウサイズを設定します。
  - Core JMS Client が JNDI を使用して接続ファクトリーをインスタンス化する場合は、接続文字列 URL の一部として **producerWindowSize** パラメーターを含めます。JNDI コンテキスト環境内に URL を保存します。以下の例では、**jndi.properties** ファイルを使用して URL を保存します。

    ```
    java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

    java.naming.provider.url=tcp://localhost:61616?producerWindowSize=1024
    ```

  - Core JMS クライアントが JNDI を使用して接続ファクトリーをインスタンス化しない場合は、値を **ActiveMQConnectionFactory.setProducerWindowSize()** に渡します。

    ```
    ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
    cf.setProducerWindowSize(1024);
    ```

11.2.2. メッセージのブロック

複数のプロデューサーを同じアドレスに関連付けることができるので、実際に利用できるものよりも、ブローカーがすべてのプロデューサーにクレジットを割り当てることができます。ただし、ブローカーが利用可能な量よりも多くのクレジットを送信できないアドレスに最大サイズを設定できます。

デフォルトの設定では、各アドレスにグローバル最大サイズ 100Mb が使用されます。アドレスが満杯になると、ブローカーはキューにルーティングする代わりに、ページングジャーナルにさらにメッセージを書き込みます。ページングの代わりに、古いメッセージが消費されるまで、クライアント側でより多くのメッセージの送信をブロックできます。このようにプロデューサーフロー制御をブロックすると、プロデューサーが一度に処理できる数を超えるメッセージを送信するため、ブローカーがメモリー不足になるのを防ぎます。

この設定では、プロデューサーフロー制御のブロックは **address-setting** ごとに管理されます。この設定は、アドレスに登録されているすべてのキューに適用されます。つまり、そのアドレスにバインドされるすべてのキューの合計メモリーは **max-size-bytes** に指定された値によって上限になります。

注記

ブロックはプロトコルに依存します。AMQ Broker では、AMQP、OpenWire、および Core Protocol がプロデューサーフロー制御を備えています。ただし、AMQP プロトコルはフロー制御を別の方法で処理します。詳細は、AMQP を使用したフロー制御のブロックを参照してください。

アドレスの最大サイズの設定
メッセージが設定された最大バイト数よりも大きい場合にブローカーを設定するには、**BROKER_INSTANCE_DIR/etc/broker.xml** に新しい **addres-setting** 設定要素を追加します。

手順

- 以下の設定例の場合、**address-setting** は最大サイズ **300000** バイトに達すると、メッセージの送信から **BLOCK** プロデューサーに設定されます。

```
<configuration>
 <core>
  ...
  <address-settings>
    <address-setting match="my.blocking.queue"> 1
      <max-size-bytes>300000</max-size-bytes> 2
      <address-full-policy>BLOCK</address-full-policy> 3
    </address-setting>
  </address-settings>
 </core>
</configuration>
```

1 上記の設定は、**my.blocking.queue** アドレス によって参照されるキューに適用されます。

2 最大サイズを **300000** バイトに設定します。メッセージが **max-size-bytes** を超えると、ブローカーはプロデューサーがアドレスに送信をブロックします。この要素は、K、Mb、GB などのバイト表記に対応していることに注意してください。

3 **address-full-policy** を **BLOCK** に設定し、プロデューサーフロー制御のブロックを有効にします。

11.2.3. AMQP メッセージのブロック

本章で前述したように、Core Protocol はプロデューサーのウィンドウサイズのフロー制御システムを使用します。このシステムでは、クレジットはバイトを表し、プロデューサーに割り当てられます。プロデューサーがメッセージを送信する場合は、送信する前にメッセージのサイズに対応するのに十分なクレジットがあるまで待機する必要があります。

しかし、AMQP フロー制御のクレジットはバイトを表すのではなく、メッセージサイズに関係なく、プロデューサーが送信できるメッセージの数を表します。そのため、AMQP クライアントがアドレスの **max-size-bytes** を大幅に超過するシナリオが考えられます。

この状況を管理するには、要素 **max-size-bytes-reject-threshold** を **address-setting** に追加し、アド

レスサイズの上限をバイト単位で指定します。この上限に達すると、ブローカーは AMQP メッセージを拒否します。デフォルトでは、**max-size-bytes-reject-threshold** は **-1** に設定されているか、制限なしに設定されます。

AMQP メッセージをブロックするブローカーの設定
AMQP メッセージが設定された最大バイト数よりも大きい場合にブローカーを設定するには、**BROKER_INSTANCE_DIR/etc/broker.xml** に新しい **addres-setting** 設定要素を追加します。

手順

- 以下の設定例は、最大サイズ **300000** バイトを **my.amqp.blocking.queue** アドレスにルーティングされる AMQP メッセージに適用します。

```
<configuration>
 <core>
  ...
  <address-settings>
    ...
    <address-setting match="my.amqp.blocking.queue"> 1
      <max-size-bytes-reject-threshold>300000</max-size-bytes-reject-threshold> 2
    </address-setting>
  </address-settings>
 </core>
</configuration>
```

1 上記の設定は、**my.amqp.blocking.queue** アドレスによって参照されるキューに適用されます。

2 ブローカーは、**300000** バイトの **max-size-bytes-reject-threshold** を超える場合は、このアドレスに一致するキューに送信される AMQP メッセージを拒否するよう設定されます。この要素は、**K**、**Mb**、**GB** などのバイト表記 に対応して**いません**。

関連資料

- AMQP プロデューサーにクレジットを設定する方法は、*3章ネットワーク接続: プロトコル* を参照してください。

11.2.4. メッセージの送信レートの設定

AMQ Broker は、プロデューサーがメッセージを生成できるレートを制限することもできます。プロデューサーレートは、1秒あたりのメッセージ単位で指定します。デフォルトの **-1** に設定すると、レート制限のあるフロー制御が無効になります。

メッセージの送信レートの設定
以下の例は、プロデューサーが Core JMS クライアントを使用している場合にメッセージを送信するレートを設定する方法を示しています。各例では、メッセージ送信の最大レートを1秒あたり **10** に設定します。

手順

- プロデューサーがメッセージを送信できるレートを設定します。
  - Core JMS Client が JNDI を使用して接続ファクトリーをインスタンス化する場合は、接続文字列 URL の一部として **producerMaxRate** パラメーターを含めます。JNDI コンテキスト環境内に URL を保存します。以下の例では、**jndi.properties** ファイルを使用して URL

を保存します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

java.naming.provider.url=tcp://localhost:61616?producerMaxRate=10
```

- Core JMS クライアントが JNDI を使用して接続ファクトリーをインスタンス化しない場合は、値を **ActiveMQConnectionFactory.setProducerMaxRate()** に渡します。

```
ConnectionFactory cf =  ActiveMQJMSClient.createConnectionFactory(...)
cf.setProducerMaxRate(10);
```

関連情報

メッセージの送信速度を制限する方法の例は、**INSTALL_DIR/examples/standard** の **producer-rate-limit** の例を参照してください。

第12章 メッセージのグループ化

メッセージグループは、以下の特性を持つメッセージセットです。

- メッセージグループのメッセージは同じグループ ID を共有します。つまり、グループ識別子プロパティーは同じです。JMS メッセージの場合、プロパティーは **JMSXGroupID** です。
- メッセージグループのメッセージは、キューに多くのコンシューマーがある場合でも、常に同じコンシューマーによって消費されます。元のコンシューマーが閉じられている場合、別のコンシューマーがメッセージグループを受信するように選択されます。

メッセージグループは、プロパティーの特定値のすべてのメッセージを同じコンシューマーで順次に処理したい場合に便利です。たとえば、特定の株式購入の注文を同じコンシューマーで順次処理したい場合があります。これを行うには、コンシューマーのプールを作成し、株式名をメッセージプロパティーの値として設定します。これにより、特定の株式のすべてのメッセージが常に同じコンシューマーによって処理されます。

注記

グループ化されたメッセージは、キューの基礎となる FIFO セマンティクスが原因で、グループ化されていないメッセージの同時処理に影響を及ぼす可能性があります。たとえば、キューの先頭に 100 のグループ化されたメッセージのチャンクがあり、その後に 1,000 個の非グループ化のメッセージが続くと、グループ化されていないメッセージが消費される前に、グループ化されたすべてのメッセージが適切なクライアントに送信されます。このシナリオにおける機能への影響は、グループ化されたすべてのメッセージが処理される間、同時メッセージ処理の一時的な停止です。メッセージグループのサイズを決定する際には、パフォーマンスのボトルネックの可能性を念頭に置いてください。グループ化されたメッセージをグループ以外のメッセージから分離するかどうかを検討してください。

12.1. クライアント側のメッセージのグループ化

以下の例は、Core JMS クライアントを使用してメッセージのグループ化を使用する方法を示しています。

手順

- グループ ID を設定します。
  - JNDI を使用して JMS クライアントの JMS 接続ファクトリーを確立する場合は、**groupID** パラメーターを追加して値を指定します。この接続ファクトリーを使用して送信されたすべてのメッセージでは、**JMSXGroupID** プロパティーが指定の値に設定されます。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?groupID=MyGroup
```

  - JNDI を使用していない場合は、**setStringProperty()** メソッドを使用して **JMSXGroupID** プロパティーを設定 します。

```
Message message = new TextMessage();
message.setStringProperty("JMSXGroupID", "MyGroup");
producer.send(message);
```

関連情報

メッセージグループの設定および使用方法の作業例は、**INSTALL_DIR/examples/features/standard** の **mesagge-group** および **message-group2** を参照してください。

12.2. 自動メッセージのグループ化

グループ ID を独自に指定する代わりに、ID を自動的に生成することができます。この方法でグループ化されたメッセージは、1 つのコンシューマーによって順次処理されます。

手順

以下の例は、Core JMS クライアントを使用してメッセージのグループ化を有効にする方法を示しています。

- グループ ID の自動生成を有効にします。
  - JNDI コンテキスト環境を使用して JMS 接続ファクトリーをインスタンス化する場合は、**autogroup=true** name-value ペアを接続 URL のクエリー文字列に追加します。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory

connectionFactory.myConnectionFactory=tcp://localhost:61616?autoGroup=true
```

  - JNDI を使用していない場合は、**ActiveMQConnectonFactory** で **autogroup** を **true** に設定します。

```
ActiveMQConnectionFactory cf =
ActiveMQJMSClient.createConnectionFactoryWithoutHA(...);
cf.setAutoGroup(true);
```

第13章 複製メッセージの検出

AMQ Broker には、受信する重複メッセージをフィルターする自動複製メッセージ検出が含まれるため、独自の複製検出ロジックをコーディングする必要はありません。

重複検出がないと、クライアントはターゲットブローカーまたは接続に失敗するたびに、送信したメッセージが正常に終了したかどうかを判断できません。たとえば、ブローカーによってメッセージが受信および処理される **前** にブローカーまたは接続に失敗すると、メッセージはそのアドレスに到達することはなく、クライアントは失敗のためにブローカーから応答を受信しません。一方、ブローカーまたは接続がブローカーによって受信および処理された **後** に失敗した場合、メッセージは正しくルーティングされますが、クライアントは応答を受信しません。

さらに、トランザクションを使用して成功を判断することは、このような場合に役立ちません。たとえば、トランザクションコミットの処理中にブローカーまたは接続が失敗する場合、クライアントはメッセージを正常に送信したかどうかを判断できません。

クライアントが最後のメッセージを再送信して仮定の失敗を修正する場合、結果としてアドレスに送信された重複メッセージが表示され、システムに悪影響を及ぼす可能性があります。重複メッセージを送信すると、たとえば、発注書が 2 回満たされる可能性があることを意味します。幸い、{AMQ Broker} は、このような問題の発生を防ぐ方法として、自動複製メッセージ検出を提供します。

13.1. 重複 ID メッセージプロパティーの使用

複製メッセージ検出を有効にするには、メッセージプロパティー **_AMQ_DUPL_ID** に一意の値を指定します。ブローカーがメッセージを受信すると、**_AMQ_DUPL_ID** の値があるかどうかを確認します。存在する場合、ブローカーはメモリーキャッシュを確認し、その値を持つメッセージをすでに受信したかどうかを確認します。同じ値を持つメッセージが見つかると、受信メッセージは無視されます。

手順

以下の例は、コア JMS クライアントを使用して重複検出プロパティーを設定する方法を示しています。便宜上、クライアントは重複 ID プロパティーの名前**_AMQ_DUPL_ID** の一貫した **org.apache.activemq.artemis.api.core.Message.HDR_DUPLICATE_DETECTION_ID** の値を使用することに注意してください。

- **_AMQ_DUPL_ID** の値を一意の **String** に設定します。

```
Message jmsMessage = session.createMessage();
String myUniqueID = "This is my unique id";
message.setStringProperty(HDR_DUPLICATE_DETECTION_ID.toString(), myUniqueID);
```

13.2. 重複 ID キャッシュの設定

ブローカーは、**_AMQ_DUPL_ID** プロパティーの受信値のキャッシュを維持します。各アドレスには個別のキャッシュがあります。キャッシュは円形で固定されます。新しいエントリーは、キャッシュ領域の要求に合わせて最も古いエントリーを置き換えます。

注記

キャッシュのサイズを適切に設定してください。以前のメッセージが同じ **_AMQ_DUPL_ID** を持つ新しいメッセージを受け取る前に **id-cache-size** メッセージよりも多くのメッセージが到達した場合、ブローカーは重複を検出できません。これにより、両方のメッセージがブローカーによって処理されます。

手順

以下の設定例は、**BROKER_INSTANCE_DIR/etc/broker.xml** に要素を追加して ID キャッシュを設定する方法を示しています。

```
<configuration>
 <core>
  ...
  <id-cache-size>5000</id-cache-size> 1
  <persist-id-cache>false</persist-id-cache> 2
 </core>
</configuration>
```

1 キャッシュの最大サイズは、パラメーター **id-cache-size** で設定します。デフォルト値は **20000** エントリーです。上記の例では、キャッシュサイズは **5000** エントリーに設定されます。

2 **persist-id-cache** を **true** に設定して、受信時に各 ID をディスクに永続化します。デフォルト値は **true** です。上記の例では、値を **false** に設定して persistence は無効になっています。

13.3. 重複検出とトランザクション

重複検出を使用してブローカー間でメッセージを移動すると、XA トランザクションを使用してメッセージを消費するのと同じ 1 度と 1 回のみ配信を保証できますが、オーバーヘッドが少なく、XA を使用するよりも設定がはるかに簡単になります。

トランザクションでメッセージを送信する場合は、トランザクション内のすべてのメッセージに **_AMQ_DUPL_ID** を設定する必要はなく、そのうちの 1 つにのみ設定する必要はありません。ブローカーがトランザクションのメッセージに対して重複メッセージを検出すると、トランザクション全体を無視します。

13.4. 重複検出およびクラスター接続

クラスター接続を設定して、クラスター全体に移動する各メッセージの重複 ID を挿入できます。

手順

- 要素 **use-duplicate-detection** を、**BROKER_INSTANCE_DIR/etc/broker.xml** にある必要なクラスター接続の設定に追加します。このパラメーターのデフォルト値は **true** です。以下の例では、要素はクラスター接続 **my-cluster** の設定に追加されます。

```
<configuration>
 <core>
  ...
  <cluster-connection>
   <cluster-connection name="my-cluster"> 2
    <use-duplicate-detection>true</use-duplicate-detection>
    ...
   </cluster-connection>
   ...
  </cluster-connections>
 </core>
</configuration>
```

関連情報

ブローカークラスターの詳細は、「ブローカークラスターの作成」 を参照してください。

第14章 メッセージの傍受

AMQ Broker では、ブローカーに出入りするパケットを傍受して、パケットの監査やメッセージのフィルターを実行できます。インターセプターはインターセプトするパケットを変更できます。これにより、強力になりますが、潜在的に危険性があります。

ビジネス要件を満たすためのインターセプターを開発できます。インターセプターはプロトコル固有であるため、適切なインターフェイスを実装する必要があります。

インターセプターは、ブール値を返す **intercept()** メソッドを実装する必要があります。値が **true** の場合、メッセージパケットは続行されます。**false** の場合、プロセスは中止され、他のインターセプターは呼び出されず、メッセージパケットはこれ以上処理されません。

14.1. インターセプターの作成

独自の受信インターセプターおよび発信インターセプターを作成できます。すべてのインターセプターはプロトコル固有で、サーバーに出入りするパケットに対して呼び出されます。これにより、監査パケットなどのビジネス要件を満たすインターセプターを作成できます。インターセプターは、傍受するパケットを変更できます。これにより、危険が伴います。そのため、注意して使用してください。

インターセプターとその依存関係は、ブローカーの Java クラスに配置する必要があります。**BROKER_INSTANCE_DIR/lib**ディレクトリーは、デフォルトでクラスパスに含まれているため、使用することができます。

手順

以下の例は、渡された各パケットのサイズをチェックするインターセプターを作成する方法を示しています。この例では、プロトコルごとに特定のインターフェイスを実装します。

- 適切なインターフェイスを実装し、その **intercept()** メソッドをオーバーライドします。
  - AMQP プロトコルを使用している場合は、**org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor** インターフェイスを実装してください。

```
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;
  }
}
```

- Core プロトコルを使用している場合は、インターセプターは、**org.apache.artemis.activemq.api.core.Interceptor** インターフェイスを実装する必要があります。

```
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;
  }
}
```

- MQTT プロトコルを使用している場合は、**org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor** インターフェイスを実装してください。

```
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;
  }
}
```

- STOMP プロトコルを使用している場合は、**org.apache.activemq.artemis.core.protocol.stomp.StompFrameInterceptor** インターフェイスを実装してください。

```
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;
  }
}
```

14.2. インターセプターを使用するためのブローカーの設定

インターセプターを作成したら、それを使用するようにブローカーを設定する必要があります。

前提条件

ブローカーで使用するために、インターセプタークラスを作成し、ブローカーの Java クラスパス (およびその依存関係) に追加する必要があります。**BROKER_INSTANCE_DIR/lib**ディレクトリーは、デフォルトでクラスパスに含まれているため、使用することができます。

手順

- ブローカーがインターセプターを使用するように設定するには、**BROKER_INSTANCE_DIR/etc/broker.xml**に設定を追加します。
  - インターセプターが着信メッセージを対象としている場合は、その **class-name** を **remoting-incoming-interceptors** のリストに追加します。

```
<configuration>
  <core>
    ...
    <remoting-incoming-interceptors>
      <class-name>org.example.MyIncomingInterceptor</class-name>
    </remoting-incoming-interceptors>
```

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

- インターセプターが発信メッセージを対象としている場合は、その **class-name** を **remoting-outgoing-interceptors** のリストに追加します。

```
<configuration>
  <core>
    ...
    <remoting-outgoing-interceptors>
      <class-name>org.example.MyOutgoingInterceptor</class-name>
    </remoting-outgoing-interceptors>
  </core>
</configuration>
```

14.3. クライアント側でのインターセプター

クライアントはインターセプターを使用して、クライアントからサーバーに送信したパケットを、またはサーバーがクライアントへインターセプトできます。ブローカーサイドインターセプターの場合は、**false** を返す場合、他のインターセプターは呼び出されず、クライアントはさらにパケットを処理しません。このプロセスは、**blocking** 方式で発信パケットが送信される場合を除き、クライアントに透過的に行われます。このような場合、ブロック送信は信頼性を提供するため、**ActiveMQException** が呼び出し元に出力されます。出力された**ActiveMQException** には、false を返したインターセプターの名前が含まれています。

サーバーと同様に、クライアントインターセプタークラスとその依存関係を適切にインスタンス化および呼び出すには、クライアントの Java クラスパスに追加する必要があります。

第15章 メッセージの迂回およびメッセージフローの分割

AMQ Broker では、**迂回** と呼ばれるオブジェクトを設定できます。これにより、クライアントアプリケーションロジックを変更せずにメッセージをあるアドレスから別のアドレスに透過的に迂回できます。迂回を設定してメッセージの **コピー** を指定された転送アドレスに転送し、メッセージフローを効果的に分割することもできます。

15.1. メッセージの迂回の仕組み

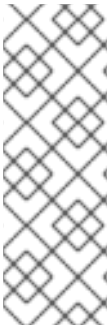
迂回により、クライアントアプリケーションロジックを変更せずに、あるアドレスにルーティングされたメッセージを他のアドレスに透過的に迂回できます。ブローカーサーバーの迂回のセットをメッセージのルーティングテーブルの種類と見なすことができます。

迂回は **排他的** にすることができます。つまり、メッセージは元のアドレスに入りずに指定されたフォワードアドレスに迂回されます。

迂回は **排他的** ではない場合もあります。つまり、ブローカーはメッセージのコピーを指定された転送アドレスに送信する一方で、メッセージを元のアドレスに引き続き送信します。そのため、メッセージフローを分割するのに排他的でない迂回を使用できます。たとえば、注文キューに送信された全順序を個別に監視する場合は、メッセージフローを分割できます。

アドレスに排他的な迂回と非排他的な迂回の両方が設定されている場合、ブローカーは特別な迂回を最初に処理します。特定のメッセージがすでに特別な迂回によって迂回されている場合、ブローカーはそのメッセージの特別でない迂回を処理しません。この場合、メッセージは元のアドレスには決して送信されません。

ブローカーがメッセージを迂回すると、ブローカーは新しいメッセージ ID を割り当て、メッセージアドレスを新しい転送アドレスに設定します。元のメッセージ ID およびアドレスの値は、**_AMQ_ORIG_ADDRESS** (文字列タイプ) および **_AMQ_ORIG_MESSAGE_ID** (long タイプ) メッセージプロパティーから取得できます。Core API を使用している場合は、**Message.HDR_ORIGINAL_ADDRESS** および **Message.HDR_ORIG_MESSAGE_ID** プロパティーを使用します。



注記

同じブローカーサーバーのアドレスにのみメッセージを迂回できます。別のサーバーのアドレスに迂回したい場合は、最初にメッセージをローカルストアアンドフォワードキューに迂回する一般的な解決策があります。次に、そのキューから消費し、メッセージを別のブローカーのアドレスに転送するブリッジを設定します。迂回とブリッジを組み合わせると、地理的に分散したブローカーサーバー間のルーティング接続の分散ネットワークを作成できます。これにより、グローバルメッセージングメッシュを作成できます。

15.2. メッセージ迂回の設定

ブローカーインスタンスで迂回を設定するには、**broker.xml** 設定ファイルの **core** 要素に **divert** 要素を追加します。

```
<core>
...
  <divert name= >
      <address> </address>
      <forwarding-address> </forwarding-address>
      <filter string= >
```

```
        <routing-type> </routing-type>
        <exclusive> </exclusive>
    </divert>
...
</core>
```

divert

迂回の名前付きインスタンス。各迂回に一意の名前がある限り、複数の **divert** 要素を **broker.xml** 設定ファイルに追加できます。

address

メッセージを迂回するアドレス

forwarding-address

メッセージを転送する アドレス

filter

オプションのメッセージフィルター。フィルターを設定すると、フィルター文字列に一致するメッセージのみが迂回されます。フィルターを指定しない場合、すべてのメッセージは迂回と一致すると見なされます。

routing-type

迂回されたメッセージのルーティングタイプ。迂回は以下に設定できます。

- **anycast** または **マルチキャスト** ルーティングのタイプのメッセージへの適用
- 既存のルーティングタイプを **削除** します。
- 既存のルーティングタイプを **通過** します (つまり保持されます)。

ルーティングタイプの制御は、メッセージにルーティングタイプがすでに設定されているものの、別のルーティングタイプを使用するアドレスにメッセージを迂回する必要がある場合に役立ちます。たとえば、ブローカーは、迂回の **routing-type** パラメーターを **MULTICAST** に設定しない限り、**anycast** ルーティングタイプのメッセージを **マルチキャスト** を使用するキューにルーティングできません。迂回の **routing-type** パラメーターの有効な値は **ANYCAST**、**MULTICAST**、**PASS**、および **STRIP** です。デフォルト値は **STRIP** です。

exclusive

迂回が排他的であるか (プロパティーを **true** に設定) または非排他的であるかを指定します (プロパティーを **false** に設定)。

以下のサブセクションは、排他的な迂回および排他的でない迂回の設定例を示しています。

15.2.1. 排他的な迂回の例

以下は、排他的な迂回の設定例です。排他的な迂回により、一致するメッセージはすべて最初に設定されたアドレスから新しいアドレスに迂回されます。一致するメッセージは元のアドレスにルーティングされません。

```
<divert name="prices-divert">
   <address>priceUpdates</address>
   <forwarding-address>priceForwarding</forwarding-address>
   <filter string="office='New York'"/>
   <exclusive>true</exclusive>
</divert>
```

上記の例では、**prices-divert** と呼ばれる迂回を定義します。これは、アドレス **priceUpdates** に送信されたメッセージを別のローカルアドレスである **priceForwarding** に迂回します。メッセージフィルター文字列も指定します。メッセージプロパティー **office** と値 **New York** を持つメッセージのみが迂回されます。その他のメッセージはすべて元のアドレスにルーティングされます。最後に、迂回が排他的であることを指定します。

15.2.2. 排他的でない迂回の例

以下は、排他的でない迂回の設定例です。排他的でない迂回では、メッセージは元のアドレスに送信されますが、ブローカーはメッセージのコピーを指定された転送アドレスにも送信します。したがって、排他的でない迂回は、メッセージフローを分割する方法です。

```
<divert name="order-divert">
  <address>orders</address>
  <forwarding-address>spyTopic</forwarding-address>
  <exclusive>false</exclusive>
</divert>
```

上記の例では、**order-divert** という迂回を定義し、アドレスの **順序** に送信されたすべてのメッセージのコピーを取り、**spyTopic** と呼ばれるローカルアドレスに送信します。また、迂回が排他的ではないように指定します。

関連情報

排他的な迂回と非排他的な迂回の両方を使用する詳細な例は、Divert Example (外部) を参照してください。

第16章 メッセージのフィルターリング

AMQ Broker は、SQL 92 式構文のサブセットに基づいて強力なフィルター言語を提供します。フィルター言語は JMS セレクターに使用されるものと同じ構文を使用しますが、事前定義された識別子は異なります。以下の表は、AMQ Broker メッセージに適用される識別子を示しています。

識別子	属性
AMQPriority	メッセージの優先度。メッセージの優先度は、**0** から **9** 間の有効な値の整数を指定します。**0** の優先度が最も低く、**9** が最も高くなります。
AMQExpiration	メッセージの有効期限。値は長い整数です。
AMQDurable	メッセージが永続化されているかどうか。値は文字列です。有効な値は **Durable** または **NonDurable** です。
AMQTimestamp	メッセージが作成された時点のタイムスタンプです。値は長い整数です。
AMQSize	メッセージの **encodeSize** プロパティーの値。**encodeSize** の値は、メッセージがジャーナルで起動するスペース (バイト単位) です。ブローカーは二重バイト文字セットを使用してメッセージをエンコードするので、メッセージの実際のサイズは **encodeSize** の半分になります。

コアフィルター式で使用される他の識別子はメッセージのプロパティーであると考えられます。JMS メッセージのセレクター構文に関するドキュメントは、Java EE API を参照してください。

16.1. フィルターを使用するようにキューを設定する

BROKER_INSTANCE_DIR/etc/broker.xml で設定するキューにフィルターを追加できます。フィルター式に一致するメッセージのみがキューに格納されます。

手順

- **filter** 要素を希望の **キュー** に追加し、要素の値として適用するフィルターを追加します。以下の例では、フィルター **NEWS='technology'** がキュー **technologyQueue** に追加されます。

```
<configuration>
 <core>
  ...
  <addresses>
    <address name="myQueue">
      <anycast>
        <queue name="myQueue">
          <filter string="NEWS='technology'"/>
        </queue>
      </anycast>
    </address>
  </addresses>
 </core>
</configuration>
```

16.2. JMS メッセージプロパティーのフィルターリング

JMS 仕様は、セレクターで使用されると String プロパティーを数値型に変換してはならないことを示しています。たとえば、メッセージの **age** プロパティーが String 値 **21** に設定されていると、セレクターの **age > 18** は一致できません。この制限により、STOMP クライアントは String プロパティーでメッセージを送信できるため制限されます。

文字列を数値に変換するフィルターの設定
String プロパティーを数値型に変換するには、接頭辞 **convert_string_expressions:** を **filter** の値に追加します。

手順

- 接頭辞 **convert_string_expressions:** を必要な **filter** に適用して、**BROKER_INSTANCE_DIR /etc/broker.xml** を編集します。以下の例では、**age > 18** から **convert_string_expressions:age > 18** に **フィルター** 値を編集します。

```
<configuration>
 <core>
  ...
  <addresses>
    <address name="myQueue">
      <anycast>
        <queue name="myQueue">
          <filter string="convert_string_expressions='age > 18'"/>
        </queue>
      </anycast>
    </address>
  </addresses>
 </core>
</configuration>
```

16.3. アノテーションを基にした AMQP メッセージのフィルター

ブローカーは、期限切れの AMQP メッセージまたは未配信の AMQP メッセージを、設定した期限切れまたは dead letter キューに移動する前に、アノテーションおよびプロパティーをメッセージに適用します。クライアントはこれらのプロパティーまたはアノテーションに基づいてフィルターを作成し、期限切れまたは dead letter キューから消費する特定のメッセージを選択できます。

注記

ブローカーが適用されるプロパティーは **内部** プロパティーです。これらのプロパティーは、通常の使用のためにクライアントに公開されませんが、フィルターのクライアントで指定**できます**。

以下は、メッセージプロパティーとアノテーションに基づくフィルターの例です。このアプローチではブローカーによる処理が少なくなるため、プロパティーに基づくフィルターリングは可能な場合に推奨される方法です。

メッセージプロパティーを基にしたフィルター

```
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();
```

メッセージのアノテーションに基づくフィルター

```
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();
```

注記

アノテーションに基づいて AMQP メッセージを消費する場合、クライアントには前述の例に示されるように **m.** 接頭辞をメッセージアノテーションに追加する必要があります。

関連情報

- ブローカーが期限切れの AMQP メッセージまたは未配信の AMQP メッセージに適用するアノテーションおよびプロパティーに関する詳細は、「期限切れまたは配信不能の AMQP メッセージに対するアノテーションおよびプロパティー」 を参照してください。

第17章 ブローカークラスターの設定

クラスターは、グループ化された複数のブローカーインスタンスで設定されます。ブローカークラスターは、メッセージ処理の負荷を複数のブローカーに分散してパフォーマンスを向上します。さらに、ブローカークラスターは、高可用性によりダウンタイムを最小限に抑えることができます。

多くの異なるクラスタートポロジーでブローカーを接続できます。クラスター内では、アクティブな各ブローカーは独自のメッセージを管理し、独自の接続を処理します。

また、クラスター全体でクライアント接続を分散し、メッセージの再分配を行うことで、ブローカーの不足を避けることもできます。

17.1. ブローカークラスターについて

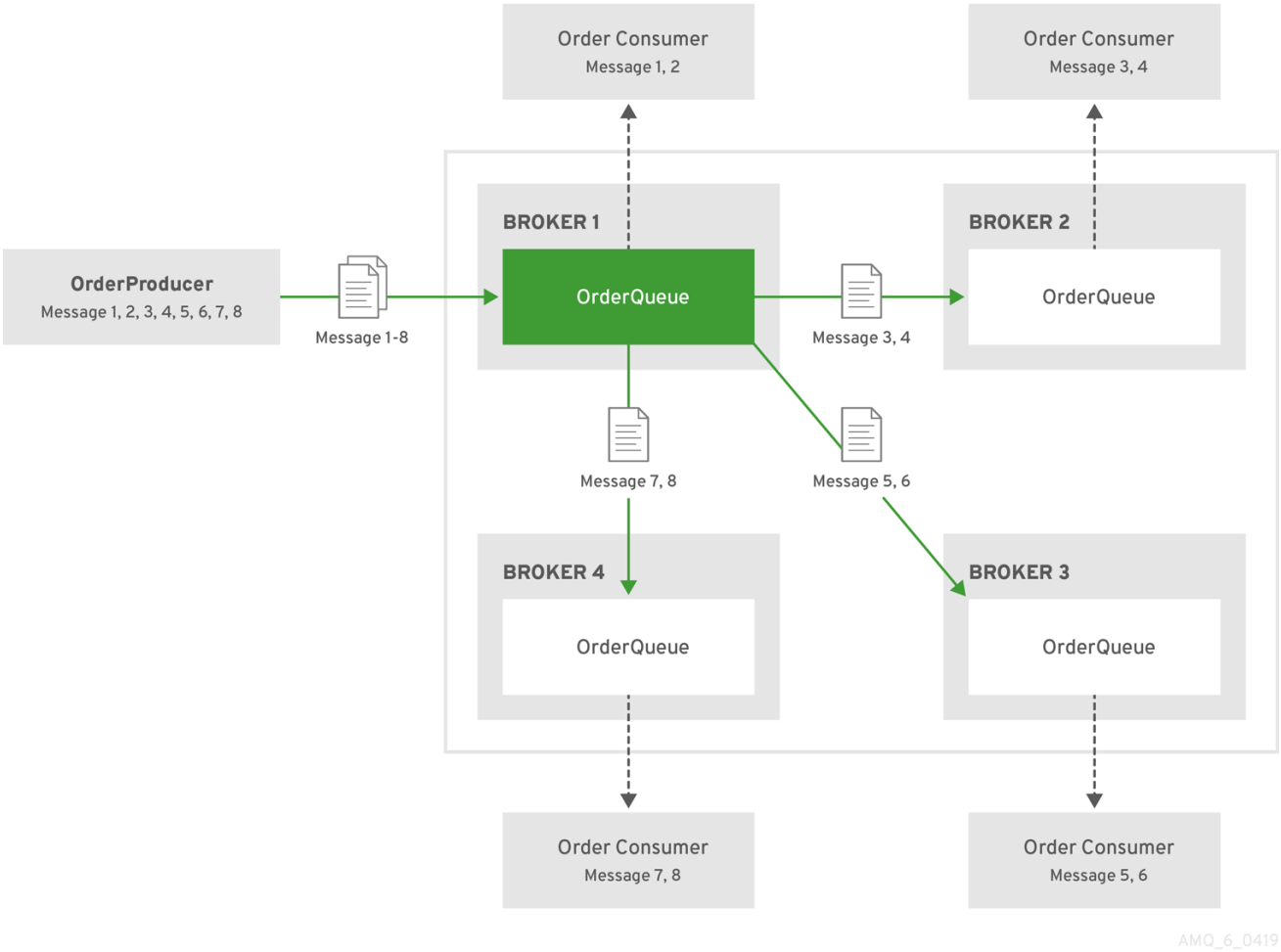
ブローカークラスターを作成する前に、いくつかの重要なクラスターリングの概念を理解する必要があります。

17.1.1. ブローカークラスターがメッセージ負荷のバランスを取る方法

ブローカーがクラスターを形成するとき、AMQ Broker はブローカー間でメッセージの負荷を自動的に分散します。これにより、クラスターが高メッセージスループットを維持できるようになります。

4 つのブローカーの対称クラスターについて考えてみましょう。各ブローカーは **OrderQueue** という名前のキューで設定されます。**OrderProducer** クライアントは **Broker1** に接続し、メッセージを **OrderQueue** に送信します。**Broker1** は、ラウンドロビン方式でメッセージを他のブローカーに転送します。各ブローカーに接続されている **OrderConsumer** クライアントはメッセージを消費します。正確な順序は、ブローカーが起動した順序によって異なります。

図17.1 メッセージの負荷分散



メッセージの負荷分散がない場合、**Broker1** に送信されたメッセージは **Broker1** に残り、**OrderConsumer1** のみがそれらを消費できます。

AMQ Broker はデフォルトでメッセージを自動的に負荷分散しますが、一致するコンシューマーを持つブローカーへのメッセージのみを負荷分散するようにクラスターを設定できます。また、コンシューマーのないキューからコンシューマーを持つキューからメッセージを自動的に再分配するようにメッセージ再分配を設定することもできます。

関連情報

- メッセージの負荷分散ポリシーは、各ブローカーのクラスター接続の **message-load-balancing** プロパティーで設定されます。詳細は、*付録C クラスター接続設定要素* を参照してください。
- メッセージ再分配に関する詳細は、「メッセージ再分配の設定」 を参照してください。

17.1.2. ブローカークラスターが信頼性を向上させる方法

ブローカークラスターは高可用性とフェイルオーバーを可能にします。これにより、スタンドアロンブローカーよりも信頼性が高くなります。高可用性を設定すると、ブローカーが障害イベントに遭遇しても、クライアントアプリケーションがメッセージを送受信できます。

高可用性により、クラスターのブローカーはライブバックアップグループにグループ化されます。ライブバックアップグループは、クライアントのリクエストに対応するライブブローカーと、パッシブに待機してライブブローカーを置き換える1つ以上のバックアップブローカーで設定されます。障害が発生

した場合、バックアップブローカーはライブバックアップグループのライブブローカーを置き換え、クライアントが再接続して作業を続行します。

17.1.3. ノード ID について

ブローカー **ノード ID** は、ブローカーインスタンスのジャーナルの初回作成および初期化時にプログラムで生成されるグローバル一意識別子 (GUID) です。ノード ID は **server.lock** ファイルに保存されます。ノード ID は、ブローカーがスタンドアロンインスタンスか、またはクラスターの一部であるかに関わらず、ブローカーインスタンスを一意に識別するために使用されます。ライブバックアップブローカーペアは、同じジャーナルを共有するため、同じノード ID を共有します。

ブローカークラスターでは、ブローカーインスタンス (ノード) は相互に接続し、ブリッジおよび内部のストアアンドフォワードキューを作成します。これらの内部キューの名前は、他のブローカーインスタンスのノード ID に基づいています。ブローカーインスタンスは、独自のノード ID のクラスターブロードキャストも監視します。ブローカーは、重複 ID を特定する場合にログに警告メッセージを生成します。

レプリケーション高可用性 (HA) ポリシーを使用している場合、起動するマスターブローカーと、**check-for-live-server** が **true** に設定されたマスターブローカーは、ノード ID を使用するブローカーを検索します。マスターブローカーが同じノード ID を使用して別のブローカーを見つける場合、それは起動しないか、または HA 設定に基づいてフェイルバックを開始します。

ノード ID は **永続性** があるため、ブローカーの再起動後も維持されます。ただし、(ジャーナルを含む) ブローカーインスタンスを削除すると、ノード ID も永続的に削除されます。

関連情報

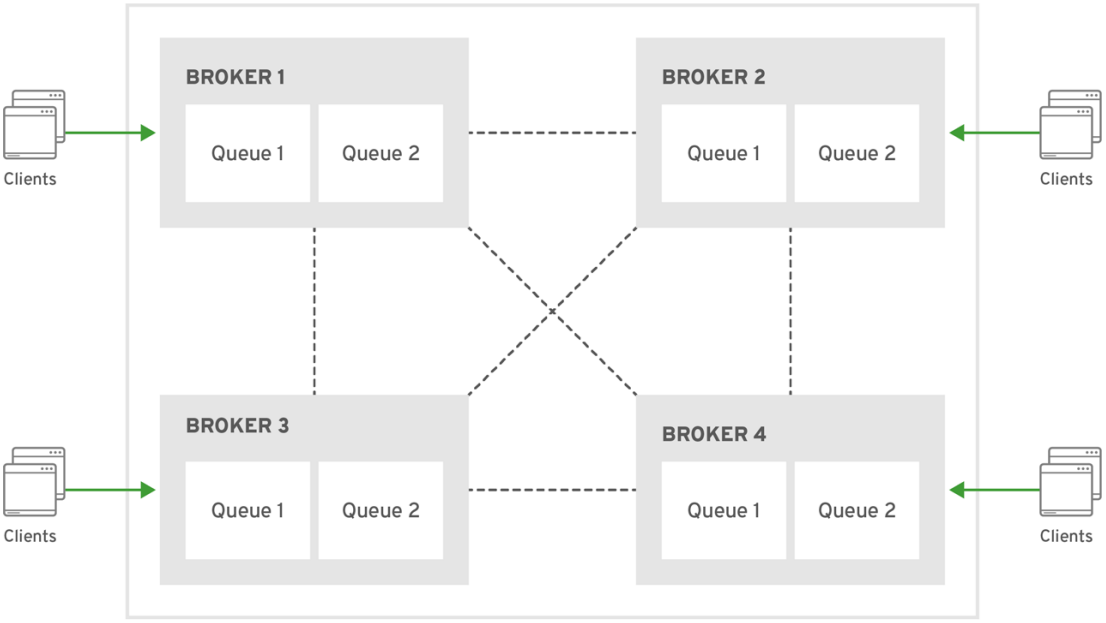
- レプリケーション HA ポリシーの設定に関する詳細は、Configuring replication high availability を参照してください。

17.1.4. 一般的なブローカークラストポロジー

ブローカーに接続し、**対称** または **チェーン** クラストポロジーのいずれかを形成できます。実装するトポロジーは、環境およびメッセージングの要件によって異なります。

対称クラスター
対称クラスターでは、すべてのブローカーが他のすべてのブローカーに接続されます。つまり、すべてのブローカーは他のすべてのブローカーから複数のホップを削除できません。

図17.2 対称クラスター



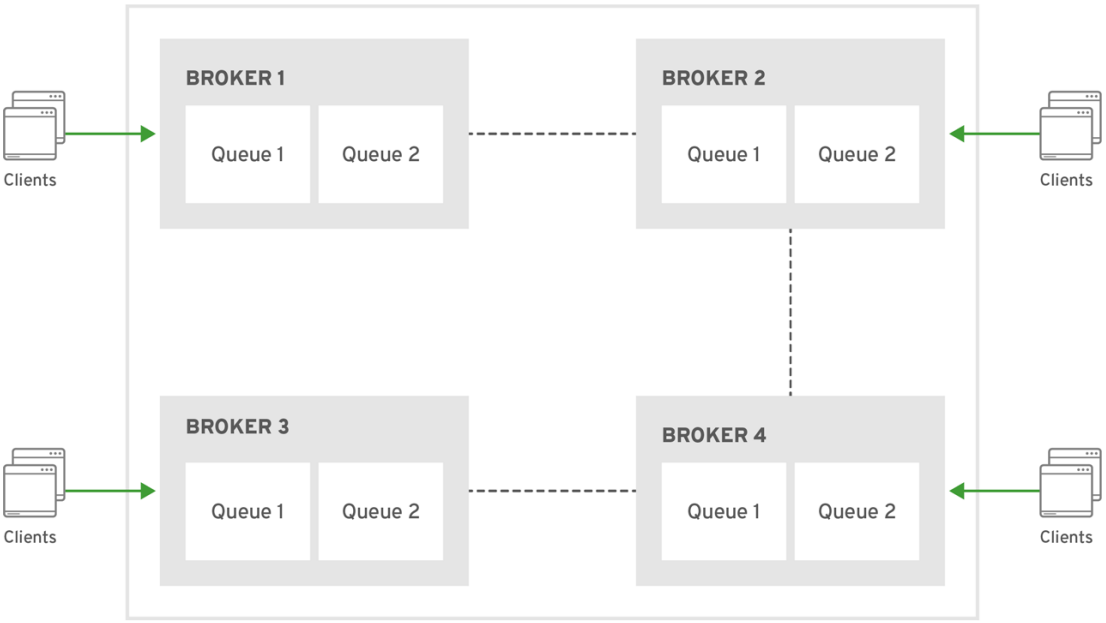
対称クラスターの各ブローカーは、クラスター内の他のすべてのブローカーに存在するすべてのキューと、これらのキューでリッスンしているコンシューマーを認識します。そのため、対称クラスターは、メッセージをチェーンクラスターよりも最適に負荷分散および再分散できます。

対称クラスターはチェーンクラスターよりも設定が簡単ですが、ネットワーク制限が原因でブローカーが直接接続されないようにする環境では使用が困難になる可能性があります。

チェーンクラスター

チェーンクラスターでは、クラスターの各ブローカーはクラスター内のすべてのブローカーに直接接続されません。代わりに、ブローカーはチェーンの最後ごとにブローカーと、チェーンの前のブローカーおよび次のブローカーに接続するすべてのブローカーでチェーンを形成します。

図17.3 チェーンクラスター



チェーンクラスターは対称クラスターよりも設定が難しくなりますが、ブローカーが別個のネットワー

ク上にあり、直接接続できない場合に役立ちます。チェーンクラスターを使用すると、中間ブローカーは 2 つのブローカーを間接的に接続し、2 つのブローカーが直接接続されていない場合でも、メッセージ間のフローが可能になります。

17.1.5. ブローカー検出メソッド

検出は、クラスターのブローカーが接続の詳細を相互に伝播するメカニズムです。AMQ Broker は、**動的検出** と **静的検出** の両方をサポートします。

動的検出
クラスターの各ブローカーは、UDP マルチキャストまたは JGroups のいずれかを使用して接続設定を他のメンバーにブロードキャストします。この方法では、各ブローカーは以下を使用します。

- クラスター接続に関する情報をクラスターの他の潜在的なメンバーにプッシュする **ブロードキャストグループ**。
- クラスターの他のブローカーに関するクラスター接続情報を受信し、保存する **ディスカバリーグループ**。

静的検出
ネットワークで UDP または JGroups を使用できない場合や、クラスターの各メンバーを手動で指定する場合は、静的検出を使用できます。この方法では、2 番目のブローカーに接続し、その接続の詳細を送信することで、ブローカーを結合します。次に、2 番目のブローカーはそれらの詳細をクラスターの他のブローカーに伝播します。

17.1.6. クラスターのサイジングに関する考慮事項

ブローカークラスターを作成する前に、メッセージングのスループット、トポロジー、および高可用性の要件を考慮してください。これらの要因は、クラスターに追加するブローカーの数に影響します。

注記

クラスターの作成後に、ブローカーを追加および削除してサイズを調整できます。メッセージを失うことなく、ブローカーを追加および削除できます。

メッセージングスループット
クラスターには、必要なメッセージングスループットを提供するのに十分なブローカーが含まれる必要があります。クラスターの他のブローカーにより、スループットが高くなります。ただし、大規模なクラスターは管理が複雑になる可能性があります。

トポロジー
対称クラスターまたはチェーンクラスターのいずれかを作成できます。選択したトポロジーのタイプは、必要なブローカーの数に影響します。

詳細は、「一般的なブローカークラストポロジー」 を参照してください。

高可用性
高可用性 (HA) が必要な場合は、クラスターを作成する前に HA ポリシーを選択することを検討してください。各マスターブローカーは少なくとも 1 つのスレーブブローカーを持つ必要があるため、HA ポリシーはクラスターのサイズに影響します。

詳細は、「高可用性の実装」 を参照してください。

17.2. ブローカークラスターの作成

クラスターに参加する各ブローカーにクラスター接続を設定して、ブローカークラスターを作成します。クラスター接続は、ブローカーが他のブローカーに接続する方法を定義します。

静的検出または動的検出を使用するブローカークラスターを作成できます (UDP マルチキャストまたは JGroups のいずれか)。

前提条件

- ブローカークラスターのサイズを決定する必要があります。
  詳細は、「クラスターのサイジングに関する考慮事項」 を参照してください。

17.2.1. 静的検出を使用したブローカークラスターの作成

ブローカーの静的リストを指定して、ブローカークラスターを作成できます。ネットワーク上で UDP マルチキャストまたは JGroups を使用できない場合は、この静的検出メソッドを使用します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. **<core>** 要素内に以下のコネクターを追加します。
   - 他のブローカーがこのブローカーに接続する方法を定義するコネクター。
   - このブローカーがクラスター内の他のブローカーに接続する方法を定義する 1 つ以上のコネクター。

```
<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 このコネクターは、他のブローカーがこの接続に使用できる接続情報を定義します。この情報は、検出中にクラスター内の他のブローカーに送信されます。

2 **broker2** および **broker3** コネクターは、このブローカーがクラスター内の 2 つの他のブローカーに接続する方法を定義します。クラスターに他のブローカーがある場合、それらは最初の接続が確立されたときにこれらのコネクターのいずれかによって検出されます。

   コネクターの詳細は、「コネクター」 を参照してください。

3. クラスター接続を追加し、静的検出を使用するように設定します。
   デフォルトでは、クラスター接続は対称トポロジーのすべてのアドレスに対してメッセージの負荷分散を行います。

```
<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

name 属性を使用してクラスター接続の名前を指定します。

connector-ref

他のブローカーがこのブローカーに接続する方法を定義するコネクター。

static-connectors

このブローカーがクラスター内の別のブローカーへの最初の接続を確立するために使用できる 1 つ以上のコネクター。この最初の接続を行った後、ブローカーはクラスター内の他のブローカーを検出します。クラスターが静的検出を使用する場合のみ、このプロパティーを設定する必要があります。

4. クラスター接続の追加プロパティーを設定します。
これらの追加のクラスター接続プロパティーには、最も一般的なユースケースに適したデフォルト値があります。そのため、デフォルト動作が必要ない場合のみこのプロパティーを設定する必要があります。詳細は、付録C *クラスター接続設定要素* を参照してください。

5. クラスターユーザーおよびパスワードを作成します。
AMQ Broker にはデフォルトのクラスター認証情報が同梱されていますが、承認されていないリモートクライアントがこれらのデフォルト認証情報を使用してブローカーに接続するのを防ぐため、これらの変更を行う必要があります。

重要

クラスターのパスワードは、クラスター内のすべてのブローカーで同じである必要があります。

```
<configuration>
   <core>
      ...
      <cluster-user>cluster_user</cluster-user>
      <cluster-password>cluster_user_password</cluster-password>
      ...
   </core>
</configuration>
```

6. 追加のブローカーごとにこの手順を繰り返します。
クラスター設定を各追加ブローカーにコピーできます。ただし、他の AMQ Broker データファイルはコピーしないでください (バインディング、ジャーナル、大きなメッセージディレクトリーなど)。これらのファイルは、クラスター内のノード間で一意である必要があり、クラスターが適切に形成されません。

関連情報

- 静的検出を使用するブローカークラスターの例は、**clustered-static-discovery** AMQ Broker example program を参照してください。

17.2.2. UDP ベースの動的検出を使用したブローカークラスターの作成

ブローカーが UDP マルチキャストを使用して動的に相互を検出するブローカークラスターを作成できます。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。
2. **<core>** 要素内にコネクターを追加します。
   このコネクターは、他のブローカーがこの接続に使用できる接続情報を定義します。この情報は、検出中にクラスター内の他のブローカーに送信されます。

```
<configuration>
   <core>
      ...
      <connectors>
         <connector name="netty-connector">tcp://localhost:61617</connector>
      </connectors>
      ...
   </core>
</configuration>
```

3. UDP ブロードキャストグループを追加します。
   ブロードキャストグループにより、ブローカーはクラスター接続に関する情報をクラスター内の他のブローカーにプッシュできます。このブロードキャストグループは UDP を使用して接続設定をブロードキャストします。

```
<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>
```

   特に指示がない限り、以下のパラメーターが必要です。

   broadcast-group

   name 属性を使用してブロードキャストグループの一意の名前を指定します。

local-bind-address

UDP ソケットがバインドされるアドレス。ブローカーに複数のネットワークインターフェイスがある場合は、ブロードキャストに使用するインターフェイスを指定する必要があります。このプロパティーが指定されていない場合、ソケットはオペレーティングシステムによって選択される IP アドレスにバインドされます。これは UDP 固有の属性です。

local-bind-port

データグラムソケットがバインドされるポート。ほとんどの場合、匿名ポートを指定するデフォルト値の **-1** を使用します。このパラメーターは、**local-bind-address** との接続で使用されます。これは UDP 固有の属性です。

group-address

データがブロードキャストされるマルチキャストアドレス。これは、**224.0.0.0** - **239.255.255.255** の範囲のクラス D IP アドレスです。アドレス **224.0.0.0** は予約されており、使用することはできません。これは UDP 固有の属性です。

group-port

ブロードキャストに使用される UDP ポート番号。これは UDP 固有の属性です。

broadcast-period (オプション)

連続するブロードキャストの間隔 (ミリ秒単位)。デフォルト値は 2000 ミリ秒 (2 秒) です。

connector-ref

ブロードキャストされる必要がある以前に設定されたクラスターコネクター。

4. UDP 検出グループを追加します。
検出グループは、このブローカーが他のブローカーからコネクター情報を受け取る方法を定義します。ブローカーはコネクターのリストを維持します (各ブローカーに 1 エントリー)。ブローカーからブロードキャストを受信すると、そのエントリーが更新されます。期間のブローカーからブロードキャストを受信しない場合は、エントリーを削除します。

この検出グループは UDP を使用してクラスター内のブローカーを検出します。

```
<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>
```

特に指示がない限り、以下のパラメーターが必要です。

discovery-group

name 属性を使用して検出グループの一意の名前を指定します。

local-bind-address (オプション)

ブローカーが稼働しているマシンが複数のネットワークインターフェイスを使用する場合は、ディスカバリーグループがリッスンするネットワークインターフェイスを指定できます。これは UDP 固有の属性です。

group-address

リッスンするグループのマルチキャストアドレス。リッスンするブロードキャストグループの **group-address** と一致する必要があります。これは UDP 固有の属性です。

group-port

マルチキャストグループの UDP ポート番号。リッスンするブロードキャストグループの **group-port** と一致する必要があります。これは UDP 固有の属性です。

refresh-timeout (オプション)

特定のブローカーから最後のブロードキャストを受信した後、そのブローカーのコネクターペアエントリーをリストから削除するまで、ディスカバリーグループが待機する時間 (ミリ秒単位)。デフォルトは 10000 ミリ秒 (10 秒) です。
この値は、ブロードキャストグループの **broadcast-period** よりも大きな値に設定します。そうしないと、(タイミングで若干の違いを及ぼすため) ブロードキャスト中であってもブローカーが一覧から定期的に消える可能性があります。

5. クラスター接続を作成し、動的検出を使用するように設定します。
デフォルトでは、クラスター接続は対称トポロジーのすべてのアドレスに対してメッセージの負荷分散を行います。

```
<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

name 属性を使用してクラスター接続の名前を指定します。

connector-ref

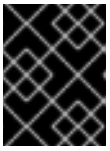
他のブローカーがこのブローカーに接続する方法を定義するコネクター。

discovery-group-ref

このブローカーがクラスターの他のメンバーを見つけるために使用する検出グループ。クラスターが動的検出を使用する場合のみ、このプロパティーを設定する必要があります。

6. クラスター接続の追加プロパティーを設定します。
これらの追加のクラスター接続プロパティーには、最も一般的なユースケースに適したデフォルト値があります。そのため、デフォルト動作が必要ない場合のみこのプロパティーを設定する必要があります。詳細は、付録C *クラスター接続設定要素* を参照してください。

7. クラスターユーザーおよびパスワードを作成します。
AMQ Broker にはデフォルトのクラスター認証情報が同梱されていますが、承認されていないリモートクライアントがこれらのデフォルト認証情報を使用してブローカーに接続するのを防ぐため、これらの変更を行う必要があります。



重要

クラスターのパスワードは、クラスター内のすべてのブローカーで同じである必要があります。

```
<configuration>
   <core>
      ...
      <cluster-user>cluster_user</cluster-user>
      <cluster-password>cluster_user_password</cluster-password>
      ...
   </core>
</configuration>
```

8. 追加のブローカーごとにこの手順を繰り返します。
クラスター設定を各追加ブローカーにコピーできます。ただし、他の AMQ Broker データファイルはコピーしないでください (バインディング、ジャーナル、大きなメッセージディレクトリーなど)。これらのファイルは、クラスター内のノード間で一意である必要があり、クラスターが適切に形成されません。

関連情報

- UDP で動的検出を使用するブローカークラスター設定の例は、 **clustered-queue** AMQ Broker example program を参照してください。

17.2.3. JGroups ベースの動的検出を使用したブローカークラスターの作成

すでに JGroups をお使いの環境で使用している場合は、これを使用して、ブローカーが相互に動的に検出するブローカークラスターを作成できます。

前提条件

- JGroups がインストールされ、設定されている必要があります。
JGroups 設定ファイルの例は、 **clustered-jgroups** AMQ Broker example program を参照してください。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。

2. **<core>** 要素内にコネクターを追加します。
このコネクターは、他のブローカーがこの接続に使用できる接続情報を定義します。この情報は、検出中にクラスター内の他のブローカーに送信されます。

```
<configuration>
   <core>
      ...
      <connectors>
         <connector name="netty-connector">tcp://localhost:61617</connector>
      </connectors>
      ...
   </core>
</configuration>
```

3. **<core>** 要素内に JGroups ブロードキャストグループを追加します。
ブロードキャストグループにより、ブローカーはクラスター接続に関する情報をクラスター内の他のブローカーにプッシュできます。このブロードキャストグループは JGroups を使用して接続設定をブロードキャストします。

```
<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>
```

特に指示がない限り、以下のパラメーターが必要です。

broadcast-group

name 属性を使用してブロードキャストグループの一意の名前を指定します。

jgroups-file

JGroups チャネルを初期化する JGroups 設定ファイルの名前。ブローカーが読み込めるように、このファイルは Java リソースパスにある必要があります。

jgroups-channel

ブロードキャストするために接続する JGroups チャネルの名前。

broadcast-period (オプション)

連続するブロードキャストの間隔 (ミリ秒単位)。デフォルト値は 2000 ミリ秒 (2 秒) です。

connector-ref

ブロードキャストされる必要がある以前に設定されたクラスターコネクター。

4. JGroups 検出グループを追加します。
検出グループは、コネクター情報を受け取る方法を定義します。ブローカーはコネクターのリストを維持します (各ブローカーに 1 エントリー)。ブローカーからブロードキャストを受信すると、そのエントリーが更新されます。期間のブローカーからブロードキャストを受信しない場合は、エントリーを削除します。

この検出グループは JGroups を使用してクラスター内のブローカーを検出します。

```
<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>
```

特に指示がない限り、以下のパラメーターが必要です。

discovery-group

name 属性を使用して検出グループの一意の名前を指定します。

jgroups-file

JGroups チャネルを初期化する JGroups 設定ファイルの名前。ブローカーが読み込めるように、このファイルは Java リソースパスにある必要があります。

jgroups-channel

ブロードキャストを受信するために接続する JGroups チャネルの名前。

refresh-timeout (オプション)

特定のブローカーから最後のブロードキャストを受信した後、そのブローカーのコネクターペアエントリーをリストから削除するまで、ディスカバリーグループが待機する時間 (ミリ秒単位)。デフォルトは 10000 ミリ秒 (10 秒) です。
この値は、ブロードキャストグループの **broadcast-period** よりも大きな値に設定します。そうしないと、(タイミングで若干の違いを及ぼすため) ブロードキャスト中であってもブローカーが一覧から定期的に消える可能性があります。

5. クラスター接続を作成し、動的検出を使用するように設定します。
デフォルトでは、クラスター接続は対称トポロジーのすべてのアドレスに対してメッセージの負荷分散を行います。

```
<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

name 属性を使用してクラスター接続の名前を指定します。

connector-ref

他のブローカーがこのブローカーに接続する方法を定義するコネクター。

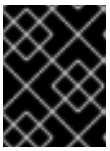
discovery-group-ref

このブローカーがクラスターの他のメンバーを見つけるために使用する検出グループ。クラスターが動的検出を使用する場合のみ、このプロパティーを設定する必要があります。

6. クラスター接続の追加プロパティーを設定します。

これらの追加のクラスター接続プロパティーには、最も一般的なユースケースに適したデフォルト値があります。そのため、デフォルト動作が必要ない場合のみこのプロパティーを設定する必要があります。詳細は、付録C *クラスター接続設定要素* を参照してください。

7. クラスターユーザーおよびパスワードを作成します。
AMQ Broker にはデフォルトのクラスター認証情報が同梱されていますが、承認されていないリモートクライアントがこれらのデフォルト認証情報を使用してブローカーに接続するのを防ぐため、これらの変更を行う必要があります。



重要

クラスターのパスワードは、クラスター内のすべてのブローカーで同じである必要があります。

```
<configuration>
  <core>
    ...
    <cluster-user>cluster_user</cluster-user>
    <cluster-password>cluster_user_password</cluster-password>
    ...
  </core>
</configuration>
```

8. 追加のブローカーごとにこの手順を繰り返します。
クラスター設定を各追加ブローカーにコピーできます。ただし、他の AMQ Broker データファイルはコピーしないでください (バインディング、ジャーナル、大きなメッセージディレクトリーなど)。これらのファイルは、クラスター内のノード間で一意である必要があり、クラスターが適切に形成されません。

関連情報

- JGroups で動的検出を使用するブローカークラスターの例は、**clustered-jgroups** AMQ Broker example program を参照してください。

17.3. 高可用性の実装

ブローカークラスターの作成後に、高可用性 (HA) を実装してその信頼性を強化できます。HA では、1 つ以上のブローカーがオフラインになっても、ブローカークラスターは機能し続けます。

HA の実装には、複数のステップが関係します。

1. ライブバックアップグループの概要を理解し、要件に最も適した HA ポリシーを選択する必要があります。Understanding how HA works in AMQ Broker を参照してください。

2. 適切な HA ポリシーを選択したら、クラスター内の各ブローカーに HA ポリシーを設定します。以下を参照してください。

   - Configuring shared store high availability
   - Configuring replication high availability
   - Configuring limited high availability with live-only
   - Configuring high availability with colocated backups

3. フェイルオーバーを使用するようにクライアントアプリケーションを設定します。

注記

高可用性用に設定されたブローカークラスターをトラブルシューティングする必要がある場合は、クラスターでブローカーを実行している各 Java Virtual Machine (JVM) インスタンスにガベージコレクション (GC) ロギングを有効にすることが推奨されます。JVM で GC ログを有効にする方法については、JVM で使用される Java Development Kit (JDK) バージョンの公式ドキュメントを参照してください。AMQ Broker がサポートする JVM バージョンの詳細は、Red Hat AMQ 7 Supported Configurations を参照してください。

17.3.1. High Availability Deployment and Usage

AMQ Broker では、クラスターでブローカーを **ライブバックアップグループ** にグループ化して、高可用性 (HA) を実装します。ライブバックアップグループでは、ライブブローカーはバックアップブローカーにリンクされ、失敗した場合はライブブローカーを引き継ぐことができます。AMQ Broker は、ライブバックアップグループ内のフェイルオーバー (**HA ポリシー** と呼ばれる) にいくつかの異なるストラテジーも提供します。

17.3.1.1. ライブバックアップグループがどのように高可用性を提供するか

AMQ Broker では、クラスターにブローカーをリンクして高可用性 (HA) を実装し、**ライブバックアップグループ** を形成します。ライブバックアップグループは **フェイルオーバー** を提供します。つまり、1 つのブローカーが失敗すると、別のブローカーがメッセージ処理を引き継ぐことができます。

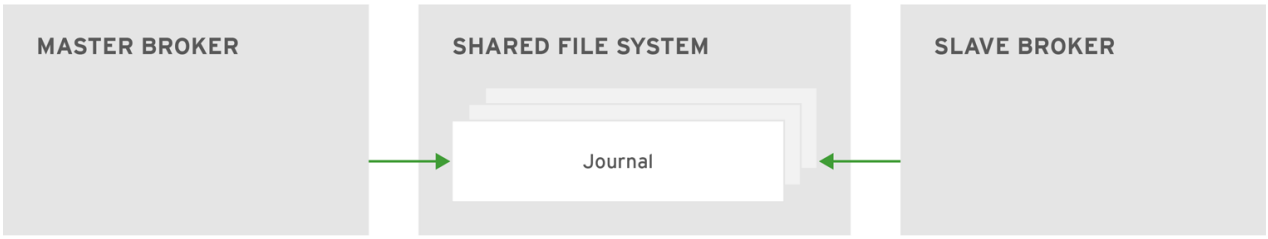
ライブバックアップグループは、1 つ以上のバックアップブローカー (**スレーブ** ブローカーとも呼ばれる) にリンクされた 1 つのライブブローカー (**マスター** ブローカーとも呼ばれる) で設定されます。ライブブローカーはクライアント要求に対応し、バックアップブローカーはパッシブモードで待機します。ライブブローカーが失敗すると、バックアップブローカーはライブブローカーに置き換わり、クライアントが再接続し、作業を続行します。

17.3.1.2. 高可用性ポリシー

高可用性 (HA) ポリシーは、ライブバックアップグループでのフェイルオーバーの発生方法を定義します。AMQ Broker では、複数の HA ポリシーが提供されます。

共有ストア (推奨)

ライブおよびバックアップブローカーは、メッセージングデータを共有ファイルシステム上の共通ディレクトリー (通常は Storage Area Network (SAN) または Network File System (NFS) サーバー) に保存します。また、JDBC ベースの永続を設定している場合は、ブローカーデータを指定されたデータベースに保存することもできます。共有ストアでは、ライブブローカーが失敗すると、バックアップブローカーは共有ストアからメッセージデータを読み込み、失敗したライブブローカーに対して引き継ぎします。
ほとんどの場合、レプリケーションの代わりに共有ストアを使用する必要があります。共有ストアはネットワーク経由でデータを複製しないため、通常はレプリケーションよりもパフォーマンスが向上します。共有ストアは、ライブブローカーとそのバックアップが同時に行われるという問題である、ネットワーク分離 (スプリットブレインとも呼ばれる) を回避します。



JBOSS_409952_0317

レプリケーション

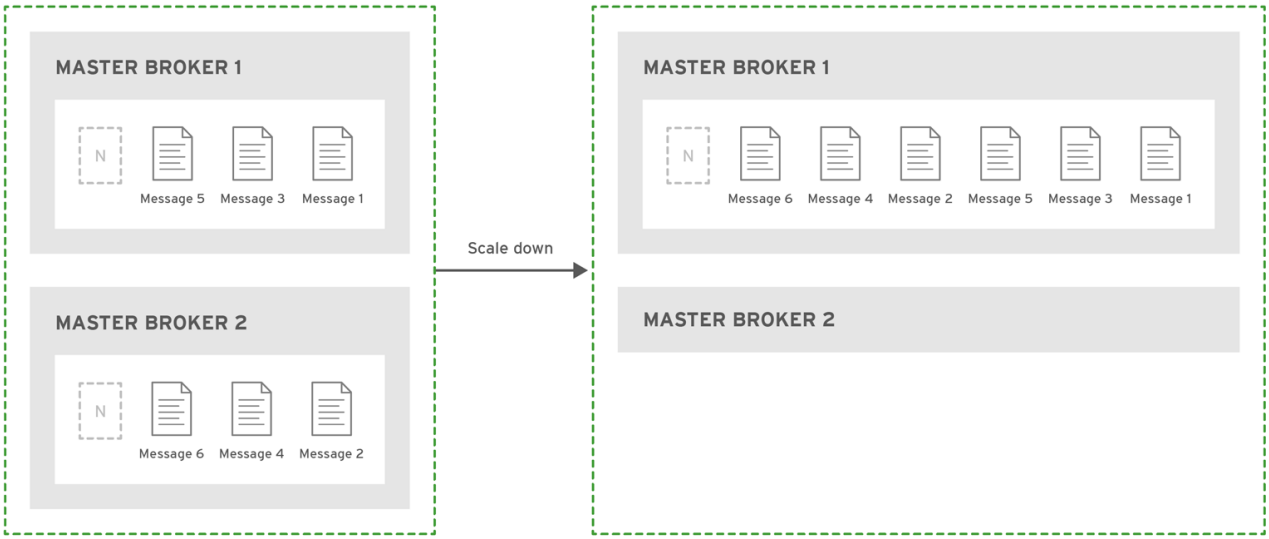
ライブおよびバックアップブローカーは、そのメッセージングデータをネットワーク経由で継続的に同期します。ライブブローカーが失敗すると、バックアップブローカーは同期データを読み込み、失敗したライブブローカーに対して引き継ぎます。
ライブブローカーとバックアップブローカー間のデータ同期により、ライブブローカーが失敗してもメッセージングデータが失われないようにします。ライブブローカーおよびバックアップブローカーが最初に結合すると、ライブブローカーはネットワーク経由ですべての既存データをバックアップブローカーに複製します。この最初のフェーズが完了すると、ライブブローカーは永続データを、ライブブローカーが受信時にバックアップブローカーに複製します。つまり、ライブブローカーがネットワークから切断されると、バックアップブローカーには、ライブブローカーが受信したすべての永続データが含まれます。

レプリケーションはネットワーク経由でデータを同期するため、ネットワーク障害により、ライブブローカーとそのバックアップが同時に存続するネットワーク分離が発生する可能性があります。

JBOSS_409952_0317

ライブのみ (限定 HA)

ライブブローカーが正常に停止されると、メッセージとトランザクションの状態を別のライブブローカーにコピーし、シャットダウンします。その後、クライアントは他のブローカーに再接続し、メッセージの送受信を続行します。



関連情報

- ライブバックアップグループのブローカー間で共有される永続メッセージデータの詳細は、「ジャーナルベースの永続性」 を参照してください。

17.3.1.3. レプリケーションポリシーの制限

ネットワーク分離 (スプリットブレインと呼ばれることもあります) は、レプリケーションの高可用性 (HA) ポリシーの制限です。その発生方法、およびその回避方法を理解している必要があります。

ライブブローカーとそのバックアップが接続を失うと、ネットワーク分離が発生する可能性があります。この場合、ライブブローカーとそのバックアップの両方を同時にアクティブにすることができます。具体的には、バックアップブローカーがクラスター内のライブブローカーの半分以上に接続できる場合は、アクティブになります。この状況では、ブローカー間でメッセージレプリケーションがないため、各ブローカーはクライアントとプロセスメッセージに、他のメッセージを認識せずに提供します。この場合、各ブローカーは完全に異なるジャーナルを持ちます。この状況からの復元は非常に困難であり、場合によっては不可能です。

ネットワーク分離を回避するには、以下の点を考慮してください。

- ネットワーク分離の可能性を **なくす** には、**共有ストア** HA ポリシーを使用します。
- レプリケーション HA ポリシーを使用する場合は、**少なくとも 3 つのライブ / バックアップのペア** を使用することで、ネットワーク分離が発生する可能性を軽減 (ただし除外しない) できます。
  少なくとも 3 つのライブ / バックアップのペアを使用することで、ライブ / バックアップブローカーのペアでレプリケーションが中断された場合に発生する **クォーラム票** で過半数の結果を得ることができます。

レプリケーション HA ポリシーを使用する場合の追加に関する考慮事項は以下のとおりです。

- ライブブローカーが失敗し、バックアップがライブに移行すると、新しいバックアップブローカーがライブに割り当てられるまで、または元のライブブローカーへのフェイルバックが発生するまでレプリケーションは行われません。
- ライブバックアップグループでバックアップブローカーが失敗すると、ライブブローカーはメッセージを提供します。ただし、メッセージは別のブローカーがバックアップとして追加されるか、または元のバックアップブローカーが再起動されるまで複製されません。この間、

メッセージはライブブローカーにのみ永続化されます。

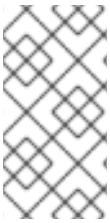
- ライブバックアップペアの両方のブローカーが以前にシャットダウンしたが、再起動できるようになったとします。この場合、メッセージが失われないようにするには、最初にアクティブなブローカーを再起動する必要があります。最近アクティブなブローカーがバックアップブローカーであった場合は、このブローカーをマスターブローカーとして手動で再設定し、最初に再起動できるようにする必要があります。

17.3.2. Configuring shared store high availability

共有ストア高可用性 (HA) ポリシーを使用して HA をブローカークラスターに実装できます。共有ストアでは、ライブおよびバックアップブローカーの両方が、共有ファイルシステム上の共通ディレクトリー (通常は Storage Area Network (SAN) または Network File System (NFS) サーバー) にアクセスします。また、JDBC ベースの永続を設定している場合は、ブローカーデータを指定されたデータベースに保存することもできます。共有ストアでは、ライブブローカーが失敗すると、バックアップブローカーは共有ストアからメッセージデータを読み込み、失敗したライブブローカーに対して引き継ぎします。

通常、SAN は NFS サーバーよりもパフォーマンスが向上する (速度など)、推奨されるオプションになります (利用可能な場合)。NFS サーバーを使用する必要がある場合は、AMQ Broker がサポートするネットワークファイルシステムに関する詳細は、Red Hat AMQ 7 Supported Configurations を参照してください。

ほとんどの場合、レプリケーションの代わりに共有ストア HA を使用する必要があります。共有ストアはネットワーク経由でデータを複製しないため、通常はレプリケーションよりもパフォーマンスが向上します。共有ストアは、ライブブローカーとそのバックアップが同時に行われるという問題である、ネットワーク分離 (スプリットブレインとも呼ばれる) を回避します。



注記

共有ストアを使用する場合、バックアップブローカーの起動時間はメッセージジャーナルのサイズによって異なります。バックアップブローカーが失敗したライブブローカーに対して引き継がると、共有ストアからジャーナルが読み込まれます。ジャーナルに多くのデータが含まれる場合、このプロセスには時間がかかることがあります。

17.3.2.1. NFS 共有ストアの設定

共有ストアの高可用性を使用する場合、ライブおよびバックアップブローカーの両方を、共有ファイルシステムの共通ディレクトリーを使用するように設定する必要があります。通常、Storage Area Network (SAN) またはネットワークファイルシステム (NFS) サーバーを使用します。

以下は、各ブローカーインスタンスの NFS サーバーからエクスポートされたディレクトリーをマウントする際に推奨される設定オプションです。

sync

すべての変更が即時にディスクにフラッシュされることを指定します。

intr

サーバーがダウンした場合やサーバーにアクセスできない場合に、NFS 要求の割り込みを許可します。

noac

属性キャッシュを無効にします。この動作は、複数のクライアント間で属性キャッシュの一貫性を実現するために必要です。

軽度

NFS サーバーが利用できない場合、サーバーがオンラインに戻るのを待つのではなく、エラーを報告する必要があることを指定します。

lookupcache=none

ルックアップキャッシングを無効にします。

timeo=n

NFS クライアント (ブローカー) が NFS サーバーからの応答を待機する時間 (デシ秒 (10 秒)) は、要求を再試行するまで NFS サーバーからの応答を待つ時間です。TCP 経由の NFS の場合、デフォルトの **timeo** 値は **600** (60 秒) です。UDP 経由の NFS では、クライアントは適応アルゴリズムを使用して、読み取り要求や書き込み要求など、頻繁に使用される要求のタイプに適切なタイムアウト値を予測します。

retrans=n

さらなるリカバリーアクションを試行する前に、NFS クライアントが要求を再試行する回数。**retrans** オプションが指定されていない場合、NFS クライアントは各リクエストを 3 回試行します。

重要

timeo オプションおよび **retrans** オプションを設定する場合、適切な値を使用することが重要です。デフォルト **timeo** の 600 デシ秒 (60 秒) と **retrans** 値の 5 を組み合わせると、ActiveMQ Artemis が NFS 接続の切断を検出するのを 5 分間待機する可能性があります。

関連情報

- NFS サーバーからエクスポートしたディレクトリーをマウントする方法は、Red Hat Enterprise Linux ドキュメントの Mounting an NFS share with mount を参照してください。
- AMQ Broker でサポートされるネットワークファイルシステムに関する詳細は、Red Hat AMQ 7 Supported Configurations を参照してください。

17.3.2.2. Configuring shared store high availability

この手順では、ブローカークラスターの共有ストアの高可用性を設定する方法を説明します。

前提条件

- 共有ストレージシステムは、ライブブローカーおよびバックアップブローカーからアクセスできる必要があります。
  - 通常、Storage Area Network (SAN) またはネットワークファイルシステム (NFS) サーバーを使用して共有ストアを提供します。サポートされるネットワークファイルシステムの詳細は、Red Hat AMQ 7 Supported Configurations を参照してください。
  - JDBC ベースの永続性を設定している場合は、指定したデータベースを使用して共有ストアを提供できます。JDBC 永続性 の設定方法は、JDBC 永続性の設定を参照してください。

手順

1. クラスターのブローカーをライブバックアップグループにグループ化します。
ほとんどの場合、ライブバックアップグループは、ライブブローカーとバックアップブローカーという 2 つのブローカーで設定される必要があります。クラスターに 6 つのブローカーがある場合は、3 つのライブバックアップグループが必要になります。

2. 1つのライブブローカーと1つのバックアップブローカーで設定される最初のライブバックアップグループを作成します。

   a. ライブのブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

   b. 以下を使用している場合:

      i. 共有ストアを提供するネットワークファイルシステム。ライブブローカーのページング、バインディング、ジャーナル、および大きなメッセージディレクトリーが、バックアップブローカーがアクセスできる共有場所をポイントすることを確認します。

```
<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. 共有ストアを提供するデータベース。マスターとバックアップブローカーの両方が同じデータベースに接続でき、同じ設定を **broker.xml** 設定ファイルの **database-store** 要素に指定できるようにします。以下は設定例です。

```
<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>BINDINGS_TABLE</bindings-table-name>
        <message-table-name>MESSAGE_TABLE</message-table-name>
        <large-message-table-name>LARGE_MESSAGES_TABLE</large-message-
table-name>
        <page-store-table-name>PAGE_STORE_TABLE</page-store-table-name>
        <node-manager-store-table-name>NODE_MANAGER_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. HA ポリシーの共有ストアを使用するようにライブブローカーを設定します。

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <master>
          <failover-on-shutdown>true</failover-on-shutdown>
        </master>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

failover-on-shutdown

このブローカーが正常に停止された場合、このプロパティーはバックアップブローカーが稼働して引き継ぐかどうかを制御します。

d. バックアップブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

e. 以下を使用している場合:

i. 共有ストアを提供するネットワークファイルシステム。バックアップブローカーのページング、バインディング、ジャーナル、および大きなメッセージディレクトリーがライブブローカーと同じ共有場所をポイントすることを確認します。

```
<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. 共有ストアを提供するデータベース。マスターとバックアップブローカーの両方が同じデータベースに接続でき、**broker.xml** 設定ファイルの **database-store** 要素に同じ設定が指定されるようにします。

f. HA ポリシーの共有ストアを使用するようにバックアップブローカーを設定します。

```
<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>
```

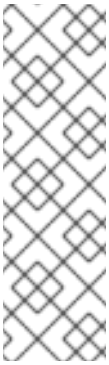
```
            </shared-store>
        </ha-policy>
        ...
    </core>
</configuration>
```

failover-on-shutdown

このブローカーが稼働状態になり、通常は停止された場合、このプロパティーはバックアップブローカー (元のライブブローカー) が有効になり、引き継ぐかどうかを制御します。

allow-failback

フェイルオーバーが発生し、バックアップブローカがライブブローカを引き継いだ場合、このプロパティーは、バックアップブローカが再起動してクラスターに再接続したときに、元のライブブローカにフェイルバックするかどうかを制御します。



注記

フェイルバックは、ライブバックアップのペア (単一のバックアップブローカーとペアの 1 つのライブブローカー) を対象としています。ライブブローカーが複数のバックアップで設定されている場合、フェイルバックは発生しません。代わりに、フェイルオーバーイベントが発生すると、バックアップブローカーはライブになり、次のバックアップがバックアップになります。元のライブブローカーがオンラインに戻ると、現在のブローカーにバックアップがすでにあるため、フェイルバックを開始できなくなります。

restart-backup

このプロパティーは、ライブブローカーにフェイルバックした後にバックアップブローカーを自動的に再起動するかどうかを制御します。このプロパティーのデフォルト値は **true** です。

3. クラスター内の残りのライブバックアップグループごとに、ステップ 2 を繰り返します。

17.3.3. Configuring replication high availability

レプリケーション高可用性 (HA) ポリシーを使用して HA をブローカークラスターに実装できます。レプリケーションでは、永続データはライブブローカーとバックアップブローカー間で同期されます。ライブブローカーが障害に遭遇すると、メッセージデータはバックアップブローカーに同期され、失敗したライブブローカーに対して引き継ぎされます。

共有ファイルシステムがない場合は、共有ストアの代わりにレプリケーションを使用する必要があります。ただし、レプリケーションにより、ライブブローカーとそのバックアップが同時に存続するネットワークの分離が発生する可能性があります。

レプリケーションでは、ネットワークの分離のリスクを低くするために (ただし、除外しない)、**少なくとも 3 つのライブバックアップペア** が必要です。3 つ以上のライブバックアップブローカーのペアを使用すると、クラスターは **クォーラムの投票** を使用して 2 つのライブブローカーを持つことができます。

以下のセクションでは、クォーラムの投票の仕組みと、少なくとも 3 つのライブバックアップのペアを使用してブローカークラスターにレプリケーション HA を設定する方法を説明します。

注記

ライブおよびバックアップブローカーは、ネットワーク経由でメッセージングデータを同期する必要があるため、レプリケーションによりパフォーマンスのオーバーヘッドが追加されます。この同期プロセスでは、ジャーナル操作がブロックされますが、クライアントはブロックされません。データの同期のためにジャーナル操作がブロックできる最大時間を設定できます。

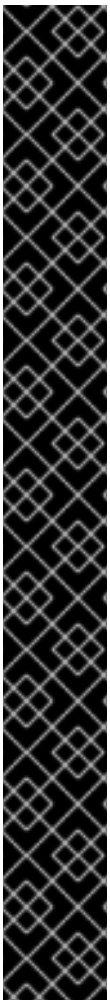
17.3.3.1. クォーラムの投票

ライブブローカーとそのバックアップでレプリケーション接続が中断された場合、**クォーラム投票** と呼ばれるプロセスを設定して、ネットワーク分離 (スプリットブレイン) の問題を軽減することができます。ネットワーク分離中、ライブブローカーとそのバックアップを同時にアクティブにすることができます。

以下の表は、AMQ Broker が使用する 2 種類のクォーラム投票を示しています。

投票タイプ	説明	イニシエーター	必要な設定	参加者	投票の結果に基づくアクション
バックアップ投票	バックアップブローカーがライブブローカーへのレプリケーション接続を失うと、バックアップブローカーはこの投票の結果に基づいて開始するかどうかを決定します。	バックアップブローカー	なし。バックアップブローカーがレプリケーションパートナーへの接続を失った際に、バックアップ評価が自動的に行われます。 ただし、これらのパラメーターにカスタムの値を指定すると、バックアップ評価のプロパティーを制御できます。 ● **quorum-vote-wait** ● **vote-retries** ● **vote-retry-wait**	クラスターの他のライブブローカー	バックアップブローカーは、クラスター内の他のライブブローカーから過半数 (**クォーラム**) 票を受信すると開始します。これは、レプリケーションパートナーが利用可能でなくなったことを示しています。

投票タイプ	説明	イニシエーター	必要な設定	参加者	投票の結果に基づくアクション
ライブ評価	ライブブローカーがレプリケーションパートナーへの接続を失った場合、ライブブローカーはこの投票に基づいて実行を継続するかどうかを決定します。	ライブブローカー	ライブ評価は、ライブブローカーがレプリケーションパートナーへの接続を失い、**vote-on-replication-failure** が **true** に設定されている場合に発生します。アクティブになったバックアップブローカーはライブブローカーと見なされ、ライブ評価を開始できます。	クラスターの他のライブブローカー	ライブブローカーは、クラスターの他のライブブローカーから過半数を受け取ら **ない** 場合はシャットダウンします。これは、クラスター接続がまだアクティブであることを示します。



重要

以下は、ブローカークラスターの設定がクォーラム投票の動作にどのように影響するかについて留意すべき重要な事項になります。

- クォーラム評価を成功させるには、クラスターのサイズで、過半数の結果が得られる必要があります。そのため、レプリケーション HA ポリシーを使用する場合、クラスターには**少なくとも 3 つの**ライブバックアップブローカーのペアが必要です。
- クラスターに追加するその他のライブバックアップブローカーのペア。クラスターのフォールトトレランス全体を増やすことができます。たとえば、ライブ / バックアップのペアが 3 つあるとします。完全なライブバックアップペアがなくなると、残りの 2 つのライブ / バックアップペアが、後続のクォーラムの投票で最大数に達することができません。この状況では、クラスターでさらにレプリケーションが中断されると、ライブブローカーがシャットダウンし、バックアップブローカーが起動しない可能性があります。例えば 5 組のブローカペアでクラスターを設定することで、クラスターでは少なくとも 2 つの障害が発生することができますが、それでもクォーラム投票で過半数の結果を得ることができます。
- クラスター内のライブバックアップブローカーのペアの数を意図的に **減らす** と、過半数に以前に設定されたしきい値は自動的に減らされません。この間、失われたレプリケーション接続によってトリガーされるクォーラム評価は成功せず、クラスターがネットワーク分離に対してより脆弱になります。クラスターがクォーラム票の過半数を再計算するには、まずクラスターから削除するライブバックアップのペアをシャットダウンします。次に、クラスター内の残りのライブバックアップペアを再起動します。残りのブローカーがすべて再起動すると、クラスターはクォーラム評価しきい値を再計算します。

17.3.3.2. レプリケーションの高可用性のためのブローカークラスターの設定

以下の手順では、6 つのブローカークラスターにレプリケーションの高可用性 (HA) を設定する方法を説明します。このトポロジーでは、6 つのブローカーはライブバックアップのペアにグループ化されます。3 つのライブブローカーは、専用のバックアップブローカーとペアになります。

レプリケーションでは、ネットワークの分離のリスクを低くするために (ただし、除外しない)、少なくとも 3 つのライブバックアップペアが必要です。

前提条件

- 6 つ以上のブローカーを持つブローカークラスターが必要です。
  6 つのブローカーは 3 つのライブバックアップペアに設定されます。ブローカーのクラスターへの追加に関する詳細は、17章 *ブローカークラスターの設定* を参照してください。

手順

1. クラスターのブローカーをライブバックアップグループにグループ化します。
   ほとんどの場合、ライブバックアップグループは、ライブブローカーとバックアップブローカーという 2 つのブローカーで設定される必要があります。クラスターに 6 つのブローカーがある場合は、3 つのライブバックアップグループが必要です。

2. 1 つのライブブローカーと 1 つのバックアップブローカーで設定される最初のライブバックアップグループを作成します。

   a. ライブのブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

   b. HA ポリシーのレプリケーションを使用するようにライブブローカーを設定します。

```
<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

ライブブローカーが失敗すると、このプロパティーは、再起動時にクライアントがフェイルバックするかどうかを制御します。
このプロパティーを **true** に設定すると、以前のフェイルオーバー後にライブブローカーが再起動すると、同じノード ID を持つクラスター内の別のブローカーを検索します。ライブブローカーが同じノード ID を持つ別のブローカーを見つけると、ライブブローカーの失敗時にバックアップブローカーが正常に起動することを示しています。この場合、ライブブローカーはデータをバックアップブローカーと同期します。その後、ライブブローカーはバックアップブローカーにシャットダウンするよう要求します。以下に示すように、バックアップブローカーがフェイルバック用に設定されている場合、シャットダウンします。その後、ライブブローカーはアクティブなロールを再開します。そして、クライアントはこれに再接続します。

警告

ライブブローカーで **check-for-live-server** を **true** に設定しない場合、以前のフェイルオーバー後にライブブローカーを再起動すると、重複したメッセージング処理が発生する可能性があります。具体的には、このプロパティーを **false** に設定してライブブローカーを再起動すると、ライブブローカーはバックアップブローカーとデータを同期しません。この場合、ライブブローカーはバックアップブローカーがすでに処理された同じメッセージを処理し、重複が発生する可能性があります。

group-name

この live-backup グループの名前。ライブバックアップグループを形成するには、ライブおよびバックアップブローカーを同じグループ名で設定する必要があります。

vote-on-replication-failure

このプロパティーは、レプリケーション接続が中断された場合に、ライブブローカーが**ライブ評価** と呼ばれるクォーラム評価を開始するかどうかを制御します。
ライブ評価は、ライブブローカーで、それまたはパートナーが中断されたレプリケーション接続の原因であるかどうかを判断する方法です。評価の結果に基づいて、ライブブローカーは稼働またはシャットダウンします。

重要

クォーラム評価を成功させるには、クラスターのサイズで、過半数の結果が得られる必要があります。そのため、レプリケーション HA ポリシーを使用する場合、クラスターには**少なくとも 3 つの**ライブバックアップブローカーのペアが必要です。

クラスターに設定するブローカーのペアが多いほど、クラスターの全体的なフォールトトレランスが向上します。たとえば、3 つのライブバックアップブローカーのペアがあるとします。完全なライブバックアップペアへの接続を失った場合、残りの 2 つのライブ / バックアップペアが、クォーラムの投票で過半数に達しなくなります。この状況では、後続のレプリケーションが中断されると、ライブブローカーがシャットダウンし、バックアップブローカーが起動しない可能性があります。例えば 5 組のブローカペアでクラスターを設定することで、クラスターでは少なくとも 2 つの障害が発生することができますが、それでもクォーラム投票で過半数の結果を得ることができます。

c. ライブブローカーの追加の HA プロパティーを設定します。
これらの追加の HA プロパティーには、最も一般的なユースケースに適したデフォルト値があります。そのため、デフォルト動作が必要ない場合のみこのプロパティーを設定する必要があります。詳細は、付録F *レプリケーション高可用性設定要素* を参照してください。

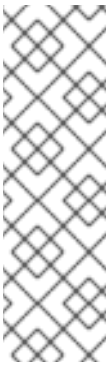
d. バックアップブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

e. HA ポリシーのレプリケーションを使用するようにバックアップ (スレーブ) ブローカーを設定します。

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
        <slave>
          <allow-failback>true</allow-failback>
          <restart-backup>true</restart-backup>
          <group-name>my-group-1</group-name>
          <vote-on-replication-failure>true</vote-on-replication-failure>
          ...
        </slave>
      </replication>
    </ha-policy>
    ...
  </core>
</configuration>
```

allow-failback

フェイルオーバーが発生し、バックアップブローカがライブブローカを引き継いだ場合、このプロパティーは、バックアップブローカが再起動してクラスターに再接続したときに、元のライブブローカにフェイルバックするかどうかを制御します。



注記

フェイルバックは、ライブバックアップのペア (単一のバックアップブローカーとペアの 1 つのライブブローカー) を対象としています。ライブブローカーが複数のバックアップで設定されている場合、フェイルバックは発生しません。代わりに、フェイルオーバーイベントが発生すると、バックアップブローカーはライブになり、次のバックアップがバックアップになります。元のライブブローカーがオンラインに戻ると、現在のブローカーにバックアップがすでにあるため、フェイルバックを開始できなくなります。

restart-backup

このプロパティーは、ライブブローカーにフェイルバックした後にバックアップブローカーを自動的に再起動するかどうかを制御します。このプロパティーのデフォルト値は **true** です。

group-name

このバックアップが接続するライブブローカーのグループ名。バックアップブローカーは、同じグループ名を共有するライブブローカーにのみ接続します。

vote-on-replication-failure

このプロパティーは、レプリケーション接続が中断された場合に、ライブブローカーが **ライブ評価** と呼ばれるクォーラム評価を開始するかどうかを制御します。アクティブになったバックアップブローカーはライブブローカーと見なされ、ライブ評価を開始できます。
ライブ評価は、ライブブローカーで、それまたはパートナーが中断されたレプリケーション接続の原因であるかどうかを判断する方法です。評価の結果に基づいて、ライブブローカーは稼働またはシャットダウンします。

f. (任意設定) バックアップブローカーが開始するクォーラム票のプロパティーを設定します。

```
<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

このプロパティーは、バックアップブローカーの起動を可能にする大部分の結果を受信するために、バックアップブローカーがクォーラム評価を再試行する回数を制御します。

vote-retry-wait

このプロパティーは、バックアップブローカーがクォーラム評価の再試行ごとに待機する期間 (ミリ秒単位) を制御します。

g. バックアップブローカーの追加の HA プロパティーを設定します。
これらの追加の HA プロパティーには、最も一般的なユースケースに適したデフォルト値があります。そのため、デフォルト動作が必要ない場合のみこのプロパティーを設定する必要があります。詳細は、付録F *レプリケーション高可用性設定要素* を参照してください。

3. クラスター内の追加のライブバックアップグループごとに、手順 2 を繰り返します。
クラスターに 6 つのブローカーがある場合は、この手順を 2 回ずつ繰り返し、残りのライブバックアップグループごとに 1 回ずつ繰り返します。

関連情報

- HA のレプリケーションを使用するブローカークラスターの例については、HA サンプルプログラム を参照してください。
- ノード ID の詳細は、Understanding node IDs を参照してください。

17.3.4. Configuring limited high availability with live-only

ライブのみの HA ポリシーを使用すると、メッセージを失わずにクラスターのブローカーをシャットダウンすることができます。ライブのみでは、ライブブローカーが正常に停止されると、メッセージとトランザクションの状態を別のライブブローカーにコピーし、シャットダウンします。その後、クライアントは他のブローカーに再接続し、メッセージの送受信を続行します。

ライブのみの HA ポリシーは、ブローカーが正常に停止された場合にのみケースを処理します。予期しないブローカーの失敗を処理しません。

ライブのみの HA はメッセージの損失を防ぎますが、メッセージの順序は保持されない可能性があります。ライブのみの HA で設定されたブローカーが停止すると、そのメッセージは別のブローカーのキューの最後に追加されます。

注記

ブローカーがスケールダウンする準備時に、接続が切断される前に、新しいブローカーがメッセージを処理する準備ができていることを通知する前に、メッセージをクライアントに送信します。ただし、クライアントは、初期ブローカーがスケールダウンされた後にのみ新しいブローカーに再接続する必要があります。これにより、キューやトランザクションなどの状態が、クライアントが再接続する際に他のブローカーで利用可能になります。通常の再接続設定はクライアントが再接続する際に適用されるため、スケールダウンに必要な時間を処理するのに十分な時間を設定する必要があります。

この手順では、クラスター内の各ブローカーをスケールダウンするように設定する方法を説明します。この手順を完了すると、ブローカーが正常に停止されるたびに、メッセージとトランザクションの状態がクラスター内の別のブローカーにコピーされます。

手順

1. 最初のブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。
2. ライブのみの HA ポリシーを使用するようにブローカーを設定します。

```
<configuration>
   <core>
      ...
      <ha-policy>
         <live-only>
         </live-only>
      </ha-policy>
      ...
   </core>
</configuration>
```

3. ブローカークラスターをスケールダウンする方法を設定します。
このブローカーがスケールダウンするブローカーのブローカーまたはグループを指定します。

スケールダウン	以下を行います

クラスター内の特定のブローカー	スケールダウンするブローカーのコネクターを指定します。
	`<live-only>`
	`  <scale-down>`
	`  <connectors>`
	`  <connector-ref>broker1-connector</connector-ref>`
	`  </connectors>`
	`  </scale-down>`
	`</live-only>`

スケールダウン	以下を行います

クラスター内のすべてのブローカー	ブローカークラスターの検出グループを指定します。
	
	`<live-only>`
	`  <scale-down>`
	`  <discovery-group-ref discovery-group-name="my-discovery-group"/>`
	`  </scale-down>`
	`</live-only>`

特定のブローカーグループのブローカー	ブローカーグループを指定します。
	
	`<live-only>`
	`  <scale-down>`
	`  <group-name>my-group-name</group-name>`
	`  </scale-down>`
	`</live-only>`

4. クラスター内の残りのブローカーごとに、この手順を繰り返します。

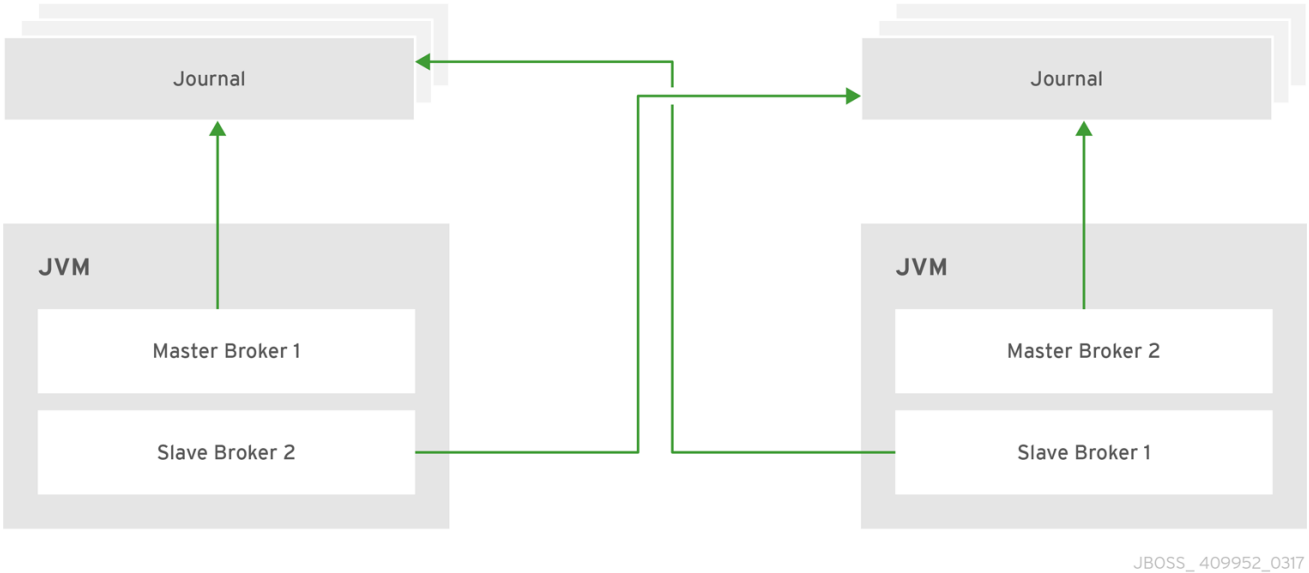
関連情報

- クラスターをスケールダウンするためにライブのみを使用するブローカークラスターの例については、scale-down example programs を参照してください。

17.3.5. Configuring high availability with colocated backups

ライブバックアップグループを設定するのではなく、バックアップブローカーを別のライブブローカーと同じ JVM にコロケートできます。この設定では、各ライブブローカーは別のライブブローカーを要求し、その JVM でバックアップブローカーを作成し、開始します。

図17.4 コロケートされたライブブローカーおよびバックアップブローカー



コロケーションは、共有ストアまたはレプリケーションのいずれかを高可用性 (HA) ポリシーとして使用できます。新しいバックアップブローカーは、これを作成するライブブローカーからその設定を継承します。バックアップの名前は、**colocated_backup_n** に設定されます。**n** は、ライブブローカーが作成したバックアップの数です。

さらに、バックアップブローカーは、コネクターと、これを作成するライブブローカーからアクセプターの設定を継承します。デフォルトでは、ポートオフセット 100 がそれぞれに適用されます。たとえば、ライブブローカーにポート 61616 のアクセプターがある場合、作成される最初のバックアップブローカーはポート 61716 を使用し、2 番目のバックアップは 61816 を使用します。

ジャーナル、大きなメッセージ、ページングのディレクトリーは、選択した HA ポリシーに従って設定されます。共有ストアを選択する場合、要求ブローカーはターゲットブローカーに対し、使用するディレクトリーについて通知します。レプリケーションが選択されている場合、ディレクトリーは作成ブローカーから継承され、新しいバックアップの名前が追加されます。

この手順では、共有のストア HA を使用するようにクラスター内の各ブローカーを設定し、バックアップを作成してクラスター内の別のブローカーと共存させるよう要求します。

手順

1. 最初のブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

2. HA ポリシーとコロケーションを使用するようにブローカーを設定します。
   この例では、ブローカーは共有ストア HA およびコロケーションで設定されます。

```
<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

このプロパティーを **true** に設定すると、このブローカーはクラスター内の別のライブブローカーによって作成されるバックアップブローカーを要求します。

max-backups

このブローカーが作成できるバックアップブローカーの数。このプロパティーを **0** に設定すると、このブローカーはクラスターの他のブローカーからのバックアップ要求を受け入れません。

backup-request-retries

このブローカーがバックアップブローカーの作成を要求する回数。デフォルトは、無制限を意味する **-1** です。

backup-request-retry-interval

バックアップブローカーを作成する要求を再試行する前にブローカーが待機する時間 (ミリ秒単位)。デフォルトは **5000** (5 秒) です。

backup-port-offset

新しいバックアップブローカーにアクセプターおよびコネクターに使用するポートオフセット。このブローカーがクラスター内の別のブローカーのバックアップを作成する要求を受信すると、この量によってポートオフセットでバックアップブローカーが作成されます。デフォルトは **100** です。

excludes (オプション)

バックアップポートのオフセットからコネクターを除外します。バックアップポートのオフセットから除外する必要のある外部ブローカーのコネクターを設定している場合は、コネクターごとに **<connector-ref>** を追加します。

master

このブローカーの共有ストアまたはレプリケーションフェイルオーバーの設定。

slave

このブローカーのバックアップの共有ストアまたはレプリケーションのフェイルオーバー設定。

3. クラスター内の残りのブローカーごとに、この手順を繰り返します。

関連情報

- コロケートバックアップを使用するブローカークラスターの例については、HA example programs を参照してください。

17.3.6. フェイルオーバーするクライアントの設定

ブローカークラスターで高可用性を設定したら、クライアントがフェイルオーバーするように設定します。クライアントのフェイルオーバーにより、ブローカーが失敗すると、接続したクライアントは最小限のダウンタイムでクラスター内の別のブローカーに再接続できます。

注記

一時的なネットワークの問題が発生すると、AMQ Broker は同じブローカーへの接続を自動的に再割り当てします。これは、クライアントが同じブローカーに再接続する以外は failover と似ています。

2 種類のクライアントフェイルオーバーを設定できます。

自動クライアントフェイルオーバー

クライアントは、初回接続時にブローカークラスターに関する情報を受け取ります。接続先のブ

ローカーが失敗すると、クライアントはブローカーのバックアップに自動的に再接続し、バックアップブローカーはフェイルオーバーの前に各接続に存在するセッションおよびコンシューマーを再作成します。

アプリケーションレベルのクライアントフェイルオーバー

自動クライアントのフェイルオーバーの代わりに、障害ハンドラーで独自のカスタム再接続ロジックを使用してクライアントアプリケーションをコーディングすることもできます。

手順

- AMQ Core Protocol JMS を使用して、自動またはアプリケーションレベルのフェイルオーバーでクライアントアプリケーションを設定します。
  詳細は、AMQ Core Protocol JMS Client の使用 を参照してください。

17.4. メッセージ再分配の有効化

ブローカークラスターがオンデマンドメッセージの負荷分散を使用する場合、**メッセージ再分配** を設定して、メッセージを消費するコンシューマーを持たないキューでメッセージがストックにならないようにすることができます。

本セクションでは、以下の情報を提供します。

- メッセージディストリビューションについて
- メッセージ再分配の設定

17.4.1. メッセージ再分配について

ブローカークラスターは負荷分散を使用して、メッセージの負荷をクラスター全体に分散します。クラスター接続で負荷分散を設定する場合、**message-load-balancing** を **ON_DEMAND** に設定すると、ブローカーは一致するコンシューマーを持つ他のブローカーにのみメッセージを転送します。この動作により、メッセージがメッセージを消費するコンシューマーを持たないキューに移動されないようにします。ただし、メッセージがブローカーに転送された後にキューにアタッチされたコンシューマーがブローカーに転送されると、これらのメッセージはキュー内でストックになり、消費されません。この問題は、**不足**と呼ばれることもあります。

メッセージ再分配は、一致するコンシューマーを持つクラスター内のコンシューマーのないキューからメッセージを自動的に再分配することで、不足を防ぎます。

17.4.1.1. メッセージフィルターを使用したメッセージ再分配の制限

メッセージ再分配は、コンシューマーによるフィルター (**セレクター**とも呼ばれる) の使用をサポート**しません**。フィルターのあるコンシューマーの一般的なユースケースは、相関 ID を使用したリクエストリプライパターンです。例として以下のシナリオを見てみましょう。

1. **brokerA** と **brokerB** の 2 つのブローカーで設定されるクラスターがある。各ブローカーは **redistribution-delay** が **0** に設定され、**message-load-balancing** が **ON_DEMAND** に設定された状態で設定されます。
2. **brokerA** および **brokerB** には、それぞれ **myQueue** という名前のキューがあります。
3. リクエストに基づいて、プロデューサーは、**brokerA** のキュー **myQueue** にルーティングされるメッセージを送信します。メッセージには、**myCorrelID** という名前の相関 ID プロパティーがあり、値は **10** です。

4. コンシューマーは、**myCorrelID=5** のフィルターで **brokerA** のキュー **myQueue** に接続します。このフィルターはメッセージの相関 ID 値と一致しません。

5. 別のコンシューマーは、**myCorrelID=10** のフィルターで **brokerB** のキュー **myQueue** に接続します。このフィルターはメッセージの相関 ID 値と一致します。
この場合、**brokerB** のコンシューマーのフィルターはメッセージに一致しますが、キュー **myQueue** のコンシューマーが **brokerA** に存在するため、メッセージは **brokerA** から **brokerB** に再分配**されません**。

上記のシナリオでは、リクエストがプロデューサーに送信される **前** にコンシューマーを作成して、目的のクライアントがメッセージを受信するようにできます。このメッセージは、メッセージの相関 ID と一致するフィルターを持つコンシューマーに即座にルーティングされます。再分配は必要ありません。

関連情報

- クラスターの負荷分散に関する詳細は、「ブローカークラスターがメッセージ負荷のバランスを取る方法」 を参照してください。

17.4.2. メッセージ再分配の設定

この手順では、メッセージ再分配を設定する方法を説明します。

手順

1. **<broker_instance_dir>/etc/broker.xml** 設定ファイルを開きます。

2. **<cluster-connection>** 要素で、**<message-load-balancing>** が **<ON_DEMAND>** に設定されていることを確認します。

```
<configuration>
   <core>
      ...
      <cluster-connections>
         <cluster-connection name="my-cluster">
            ...
            <message-load-balancing>ON_DEMAND</message-load-balancing>
            ...
         </cluster-connection>
      </cluster-connections>
   </core>
</configuration>
```

3. **<address-settings>** 要素内で、キューまたはキューのセットの再分配遅延を設定します。
この例では、**my.queue** へのメッセージの負荷分散は、最後のコンシューマーが閉じられてから 5000 ミリ秒 (5 秒) に再分散されます。

```
<configuration>
   <core>
      ...
      <address-settings>
         <address-setting match="my.queue">
            <redistribution-delay>5000</redistribution-delay>
         </address-setting>
      </address-settings>
```

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

address-setting

match 属性を、メッセージを再分配するキューの名前に設定します。ブローカーのワイルドカード構文を使用してキューの範囲を指定できます。詳細は、「AMQ Broker ワイルドカード構文」 を参照してください。

redistribution-delay

このキューの最後のコンシューマーが閉じられてからクラスターの他のブローカーにメッセージを再分配するまでブローカーが待機する時間 (ミリ秒単位)。これを **0** に設定すると、メッセージは即座に再分配されます。ただし、通常は、再分配する前に遅延を設定する必要があります。コンシューマーを閉じるのは一般的ですが、同じキューに別のキューを迅速に作成する必要があります。

4. クラスター内の追加のブローカーごとにこの手順を繰り返します。

関連情報

- メッセージを再分散するブローカークラスター設定の例は、**queue-message-redistribution** AMQ Broker のサンプルプログラム を参照してください。

17.5. クラスター化されたメッセージのグループ化の設定

メッセージのグループ化により、クライアントは特定のタイプのメッセージのグループを、同じコンシューマーによって順次処理できます。クラスターの各ブローカーにグルーピングハンドラーを追加すると、クライアントはグループ化されたメッセージをクラスター内のブローカーに送信し、それらのメッセージを同じコンシューマーによって正しい順序で消費できるようにします。

グルーピングハンドラーには、**ローカルハンドラー**と**リモートハンドラー**の 2 つのタイプがあります。ブローカークラスターは、特定のグループのすべてのメッセージを適切なキューにルーティングし、目的のコンシューマーが正しい順序でそれらを使用できるようにします。

前提条件

- クラスターの各ブローカーに少なくとも 1 つのコンシューマーが必要です。
メッセージがキュー上のコンシューマーに固定されると、同じグループ ID を持つすべてのメッセージがそのキューにルーティングされます。コンシューマーが削除されると、キューはコンシューマーがない場合でもメッセージを受信し続けます。

手順

1. クラスター内の 1 つのブローカーにローカルハンドラーを設定します。
高可用性を使用している場合、これはマスターブローカーである必要があります。

   a. ブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

   b. **<core>** 要素内にローカルハンドラーを追加します。
   ローカルハンドラーはリモートハンドラーの Arbiter として機能します。ルート情報を保存し、これを他のブローカーと通信します。

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

```
        <grouping-handler name="my-grouping-handler">
           <type>LOCAL</type>
           <timeout>10000</timeout>
        </grouping-handler>
        ...
     </core>
  </configuration>
```

grouping-handler

name 属性を使用してグルーピングハンドラーに一意の名前を指定します。

type

これを **LOCAL** に設定します。

timeout

メッセージをルーティングする場所について決定する待機時間 (ミリ秒単位)。デフォルトは 5000 ミリ秒 (5 秒) です。ルーティングを決定する前にタイムアウトに達すると、例外が発生します。これにより、厳格なメッセージの順序が確保されます。
ブローカーがグループ ID を持つメッセージを受信すると、コンシューマーが割り当てられるキューへのルートを提案します。ルートがクラスターの他のブローカーのグルーピングハンドラーによって許可される場合、ルートが確立されます。クラスター内のすべてのブローカーは、このグループ ID を持つメッセージをそのキューに転送します。
ブローカーのルートプロポーザルが拒否されると、別のルートを提案し、ルートが受け入れられるまでプロセスを繰り返します。

2. 高可用性を使用している場合、ローカルハンドラー設定をマスターブローカーのスレーブブローカーにコピーします。
ローカルハンドラー設定をスレーブブローカーにコピーしても、ローカルハンドラーに対する単一障害点を防ぐことができます。

3. クラスターの残りの各ブローカーで、リモートハンドラーを設定します。

   a. ブローカーの **<broker-instance-dir>/etc/broker.xml** 設定ファイルを開きます。

   b. **<core>** 要素内にリモートハンドラーを追加します。

```
<configuration>
  <core>
     ...
     <grouping-handler name="my-grouping-handler">
        <type>REMOTE</type>
        <timeout>5000</timeout>
     </grouping-handler>
     ...
  </core>
</configuration>
```

grouping-handler

name 属性を使用してグルーピングハンドラーに一意の名前を指定します。

type

これは **REMOTE** に設定します。

timeout

メッセージをルーティングする場所について決定する待機時間 (ミリ秒単位)。デフォルトは 5000 ミリ秒 (5 秒) です。この値をローカルハンドラーの半分以上に設定します。

関連情報

- メッセージのグループ化に設定されたブローカークラスターの例は、 **clustered-grouping** AMQ Broker example program を参照してください。

17.6. クライアントのブローカークラスターへの接続

AMQ JMS クライアントを使用してクラスターに接続できます。JMS を使用すると、メッセージングクライアントを設定して、ブローカーのリストを動的または静的に検出できます。クライアント側の負荷分散を設定して、クラスター全体で接続から作成されたクライアントセッションを分散することもできます。

手順

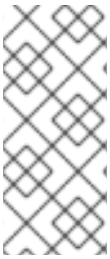
- AMQ Core Protocol JMS を使用して、ブローカークラスターに接続するクライアントアプリケーションを設定します。
  詳細は、AMQ Core Protocol JMS Client の使用 を参照してください。

第18章 マルチサイトの耐障害性のあるメッセージングシステムの設定

大規模なエンタープライズメッセージングシステムには、通常、地理的に分散したデータセンターに個別のブローカークラスターがあります。データセンターが停止した場合に、システム管理者は既存のメッセージングデータを保持し、クライアントアプリケーションがメッセージを生成および消費できるようにする必要がある場合があります。特定のブローカートポロジーと Red Hat Ceph Storage のソフトウェア定義ストレージプラットフォームを使用して、データセンターの停止時にメッセージングシステムの継続性を確保できます。このタイプのソリューションは、**マルチサイトのフォールトトレランスアーキテクチャー**と呼ばれます。

以下のセクションでは、メッセージングシステムをデータセンター停止から保護する方法を説明します。これらのセクションでは、以下の情報を提供します。

- Red Hat Ceph Storage クラスターの仕組み
- Red Hat Ceph Storage クラスターのインストールおよび設定
- データセンターが停止した場合に、ライブブローカーからバックアップブローカーを引き継ぐためのバックアップブローカーの追加
- Ceph クライアントロールを使用したブローカーサーバーの設定
- 共有ストア高可用性 (HA) ポリシーを使用するように各ブローカーを設定し、各ブローカーでメッセージングデータを保存する場所を指定する
- データセンターが停止した場合に新しいブローカーに接続するためのクライアントアプリケーションの設定
- 停止後のデータセンターの再起動



注記

マルチサイトのフォールトトレランスは、データセンター**内**の高可用性 (HA) ブローカーの冗長性の代わりではありません。ライブバックアップグループに基づくブローカーの冗長性は、単一クラスター内の単一ブローカー障害に対する自動保護を提供します。これとは対照的に、マルチサイトのフォールトトレランスは、大規模なデータセンターの停止から保護します。

注記

Red Hat Ceph Storage を使用してメッセージングシステムの継続を確保するには、共有ストアの高可用性 (HA) ポリシーを使用するようにブローカーを設定する必要があります。レプリケーション HA ポリシーを使用するようにブローカーを設定することはできません。これらのポリシーについての詳細は、Implementing High Availability を参照してください。

18.1. RED HAT CEPH STORAGE クラスターの仕組み

Red Hat Ceph Storage は、クラスター化されたオブジェクトストレージシステムです。Red Hat Ceph Storage は、オブジェクトおよびポリシーベースのレプリケーションのデータシャーディングを使用して、データの整合性とシステムの可用性を保証します。

Red Hat Ceph Storage は CRUSH (Controlled Replication Under Scalable Hashing) と呼ばれるアルゴリズムを使用して、データストレージの場所を自動的に計算してデータを保存および取得する方法を決定

します。**CRUSH マップ** と呼ばれる Ceph アイテムを設定します。これは、クラスターのトポロジーの詳細と、ストレージクラスター全体でデータの複製方法を指定します。

CRUSH マップには、オブジェクトストレージデバイス (OSD) の一覧、デバイスを障害ドメイン階層に集約するためのバケットの一覧、および CRUSH に Ceph クラスターのプールでデータを複製する方法を指示するルールが含まれます。

インストールの基礎となる物理組織を反映することで、CRUSH マップはモデル化し、物理近接性、共有電源ソース、共有ネットワークなどの関連するデバイス障害の潜在的なソースとなります。この情報をクラスターマップにエンコードすることで、CRUSH は異なる障害ドメイン (データセンターなど) 間でオブジェクトレプリカを分離しつつ、ストレージクラスター全体でデータの擬似分散を維持できます。これは、データ損失を回避し、クラスターが動作が低下した状態で動作できるようにします。

Red Hat Ceph Storage クラスターでは、動作に多数のノード (物理または仮想) が必要です。クラスターには以下のタイプのノードが含まれている必要があります。

ノードの監視

各モニターノード (MON) は、クラスターマップのマスターコピーを維持する monitor デーモン (**ceph-mon**) を実行します。クラスターマップにはクラスタートポロジーが含まれます。Ceph クラスターに接続するクライアントは、モニターからクラスターマップの現在のコピーを取得します。これにより、クライアントがクラスターへのデータの読み取りおよび書き込みが可能になります。

重要

Red Hat Ceph Storage クラスターは 1 つの Monitor ノードで実行できますが、実稼働クラスターで高可用性を確保するために、Red Hat は 3 つ以上の Monitor ノードを持つデプロイメントのみをサポートします。少なくとも 3 つのモニターノードは、1 つのモニターに障害が発生したり、1 つのモニターが利用できなくなると、クラスター内の残りの Monitor ノードが新しいリーダーを選択するためにクォーラムが存在することを意味します。

Manager ノード

各 Manager (MGR) ノードは Ceph Manager デーモン (**ceph-mgr**) を実行します。これは、ストレージ使用率、現在のパフォーマンスメトリック、システム負荷など、ランタイムメトリクスと Ceph クラスターの現在の状態を追跡します。通常、Manager ノードは (つまり、同じホストマシンにある) モニター ノードの同じ場所に配置されます。

オブジェクトストレージデバイスノード

各 Object Storage Device (OSD) ノードは Ceph OSD デーモン (**ceph-osd**) を実行し、ノードに割り当てられている論理ディスクと相互作用します。Ceph は OSD ノードにデータを保存します。Ceph は、非常に少数の OSD ノード (デフォルトは 3) で実行できますが、実稼働クラスターでは、ストレージクラスターで中程度のスケール (たとえば OSD が 50 個) のパフォーマンスが実現されます。ストレージクラスターに複数の OSD を持つと、システム管理者は CRUSH マップ内に分離された障害ドメインを定義できます。

メタデータサーバーノード

各 Metadata Server (MDS) ノードは、Ceph ファイルシステム (CephFS) に保存されているファイルに関連する MDS デーモン (**ceph-mds**) を実行します。MDS デーモンは、共有クラスターへのアクセスも調整します。

関連情報

Red Hat Ceph Storage の詳細は、What is Red Hat Ceph Storage? を参照してください。

18.2. RED HAT CEPH STORAGE のインストール

AMQ Broker のマルチサイトのフォールトトレランスアーキテクチャーは、Red Hat Ceph Storage 3 を使用します。データセンター間でデータを複製することで、Red Hat Ceph Storage クラスターは、別のデータセンターのブローカーで利用可能な共有ストアを効果的に作成します。共有ストア高可用性 (HA) ポリシーを使用し、メッセージングデータを Red Hat Ceph Storage クラスターに保存するようにブローカーを設定します。

実稼働環境で使用する Red Hat Ceph Storage クラスターには、少なくとも以下の項目が必要です。

- 3 つのモニター (MON) ノード
- 3 台のマネージャー (MGR) ノード
- 複数の OSD デーモンが含まれる 3 つのオブジェクトストレージデバイス (OSD) ノード
- 3 台のメタデータサーバー (MDS) ノード

重要

OSD、MON、MGR、および MDS ノードは、同じまたは別個の物理または仮想マシンで実行できます。ただし、Red Hat Ceph Storage クラスター内でフォールトトレランスを確保するには、これらのタイプのノードを異なるデータセンターに分散することが推奨されます。特に、1 つのデータセンターが停止した場合に、ストレージクラスターに少なくとも 2 つの MON ノードが含まれるようにする必要があります。そのため、クラスターに 3 つの MON ノードがある場合、それらの各ノードは別々のデータセンターにある別のホストマシンで実行する必要があります。このデータセンターに障害が発生しても残りの MON ノードが 1 つしかないままであるため、単一のデータセンターでは 2 つの MON ノードを実行しないでください。この場合、ストレージクラスターは機能しなくなります。

本セクションのリンク先の手順では、MON、MGR、OSD、および MDS ノードが含まれる Red Hat Ceph Storage 3 クラスターをインストールする方法について説明します。

前提条件

- Red Hat Ceph Storage インストールの準備に関する情報は、以下を参照してください。
  - 前提条件
  - Red Hat Ceph Storage のインストール要件チェックリスト

手順

- MON、MGR、OSD、および MDS ノードを含む Red Hat Ceph 3 ストレージクラスターのインストール方法を示す手順については、以下を参照してください。
  - Red Hat Ceph Storage クラスターのインストール
  - メタデータサーバーのインストール

18.3. RED HAT CEPH STORAGE CLUSTER の設定

以下の手順では、フォールトトレランスを確保するために Red Hat Ceph Storage クラスターを設定する方法を説明します。CRUSH バケットを作成し、Object Storage Device (OSD) ノードを実際の物理

インストールを反映したデータセンターに集約します。さらに、CRUSH に対してストレージプールでデータを複製する方法を指示するルールを作成します。以下の手順は、Ceph インストールで作成されたデフォルトの CRUSH マップを更新します。

前提条件

- Red Hat Ceph Storage クラスターがすでにインストールされている。詳細は、Installing Red Hat Ceph Storage を参照してください。
- Red Hat Ceph Storage が配置グループ (PG) を使用してプール内に多数のデータオブジェクトを整理する方法、およびプールで使用する PG の数を計算する方法を理解する必要があります。詳細は、Placement Groups (PGs) を参照してください。
- プール内のオブジェクトレプリカ数の設定方法を理解している。詳細は、Set the Number of Object Replicas を参照してください。

手順

1. CRUSH バケットを作成し、OSD ノードを整理します。バケットは、データセンターなどの物理的な場所に基づく OSD の一覧です。Ceph では、これらの物理的な場所は **障害ドメイン** と呼ばれます。

```
ceph osd crush add-bucket dc1 datacenter
ceph osd crush add-bucket dc2 datacenter
```

2. OSD ノードのホストマシンを、作成したデータセンター CRUSH バケットに移動します。ホスト名 **host1** - **host4** は、ホスト マシン名に置き換えます。

```
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. 作成した CRUSH バケットが **デフォルト** の CRUSH ツリーに含まれていることを確認します。

```
ceph osd crush move dc1 root=default
ceph osd crush move dc2 root=default
```

4. データセンター全体でストレージオブジェクトレプリカをマッピングするルールを作成します。これにより、データ損失を回避でき、1 つのデータセンターが停止した場合にクラスターが稼働を継続できるようになります。
ルールを作成するコマンドは、**ceph osd crush rule create-replicated <rule-name> <root> <failure-domain> <class>** 構文を使用します。以下に例を示します。

```
ceph osd crush rule create-replicated multi-dc default datacenter hdd
```

注記

上記のコマンドでは、ストレージクラスターがソリッドステートドライブ (SSD) を使用する場合は、**hdd** (ハードディスクドライブ) の代わりに **ssd** を指定します。

5. 作成したルールを使用するように、Ceph データおよびメタデータプールを設定します。最初は、これにより、CRUSH アルゴリズムによって決定されるストレージの宛先にデータがバックフィルされる可能性があります。

```
ceph osd pool set cephfs_data crush_rule multi-dc
ceph osd pool set cephfs_metadata crush_rule multi-dc
```

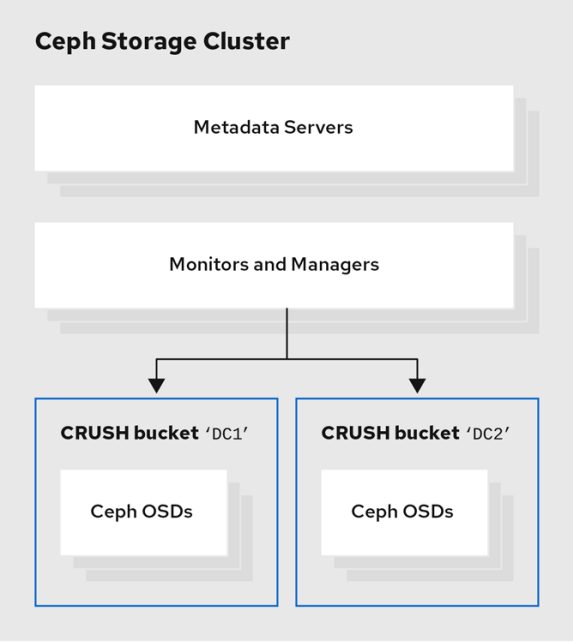
6. メタデータおよびデータプールの配置グループ (PG) および配置グループ (PGP) の数を指定します。PGP の値は PG の値と同じである必要があります。

```
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. データおよびメタデータプールによって使用されるレプリカの数を指定します。

```
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
```

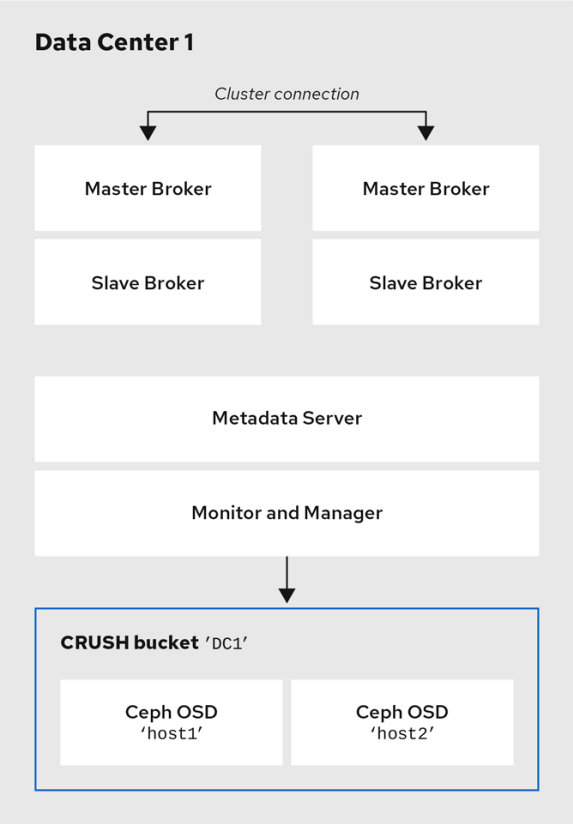
以下の図は、前述の例で作成した Red Hat Ceph Storage クラスターを示しています。ストレージクラスターには、データセンターに対応する CRUSH バケットに OSD が編成されています。



以下の図は、ブローカーサーバーを含む最初のデータセンターのレイアウトを示しています。特に、データセンターホスト。

- 2 つのライブバックアップブローカーペアのサーバー
- 前の手順で最初のデータセンターに割り当てられた OSD ノード

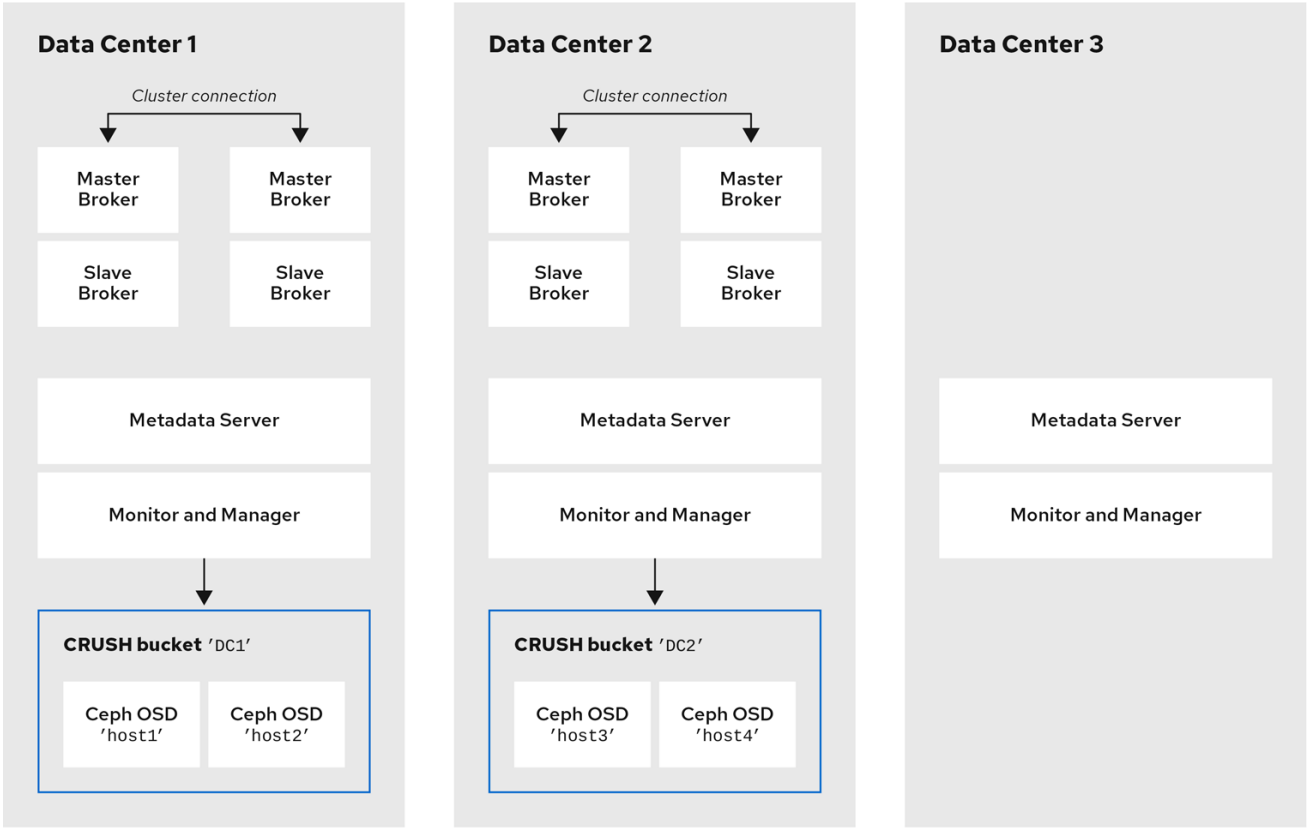
- 単一の Metadata Server、Monitor、および Manager ノード。Monitor ノードおよび Manager ノードは通常、同じマシンに共存します。



重要

OSD、MON、MGR、および MDS ノードは、同じまたは別個の物理または仮想マシンで実行できます。ただし、Red Hat Ceph Storage クラスター内でフォールトトレランスを確保するには、これらのタイプのノードを異なるデータセンターに分散することが推奨されます。特に、1 つのデータセンターが停止した場合に、ストレージクラスターに少なくとも 2 つの MON ノードが含まれるようにする必要があります。そのため、クラスターに 3 つの MON ノードがある場合、それらの各ノードは別々のデータセンターにある別のホストマシンで実行する必要があります。

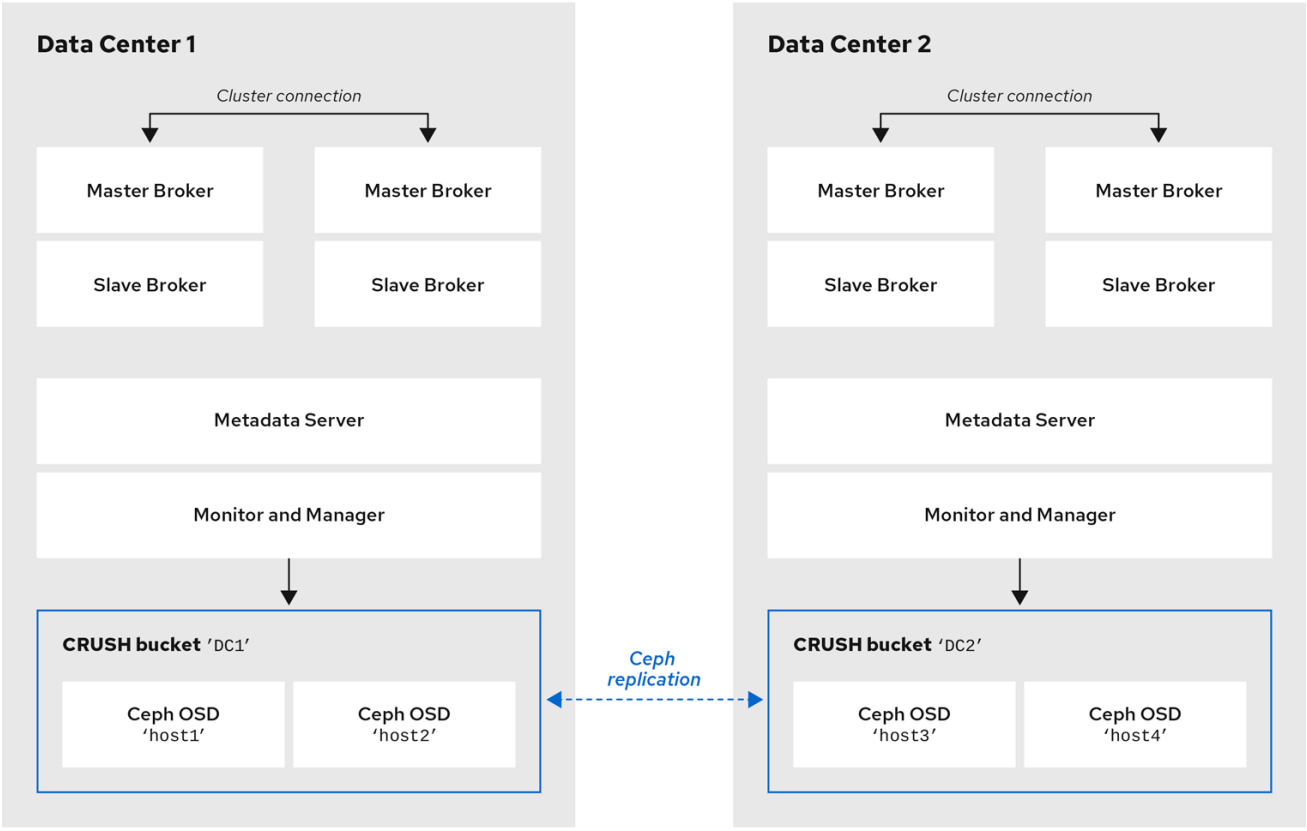
以下の図は、トポロジーの完全な例を示しています。ストレージクラスターでのフォールトトレランスを確保するために、MON、MGR、および MDS ノードは 3 つの異なるデータセンターに分散されます。



注記

ブローカーサーバーと同じデータセンターに特定の OSD ノードのホストマシンを見つけると、メッセージングデータを特定の OSD ノードに保存するわけではありません。メッセージングデータを Ceph File System の指定されたディレクトリーに保存するようにブローカーを設定します。クラスター内の Metadata Server ノードは、保存したデータをデータセンターで利用可能なすべての OSD に配信し、データセンター全体でこのデータの複製を処理する方法を決定します。以下のセクションでは、ブローカーを設定して Ceph File System にメッセージングデータを保存する方法を紹介します。

以下の図は、ブローカーサーバーを持つ 2 つのデータセンター間のデータレプリケーションを示しています。



関連情報

詳細情報:

- Red Hat Ceph Storage クラスターの CRUSH の管理については、CRUSH Administration を参照してください。
- ストレージプールに設定できる属性の完全なセットについては、Pool Values を参照してください。

18.4. ブローカーサーバーへの CEPH FILE SYSTEM のマウント

メッセージングデータを Red Hat Ceph Storage クラスターに保管するようにメッセージングシステムにブローカーを設定する前に、まず Ceph File System (CephFS) をマウントする必要があります。

本セクションのリンク先の手順では、CephFS をブローカーサーバーにマウントする方法を説明します。

前提条件

- 以下を行いました。
  - Red Hat Ceph Storage クラスターをインストールおよび設定していること。詳細は、Installing Red Hat Ceph Storage および Configuring a Red Hat Ceph Storage cluster を参照してください。
  - 3 つ以上の Ceph Metadata Server デーモンをインストールし、設定します (**ceph-mds**)。詳細は、Installing Metadata Servers および Configuring Metadata Server Daemons を参照してください。

- Monitor ノードから Ceph File System を作成します。詳細は、Creating the Ceph File System を参照してください。
- ブローカーサーバーが承認されたアクセスに使用できるキーで Ceph File System クライアントユーザーを作成しました。詳細は、Creating Ceph File System Client Users を参照してください。

手順

ブローカーサーバーに Ceph File System をマウントする方法は、Mounting the Ceph File System as a kernel client を参照してください。

18.5. マルチサイトの耐障害性のあるメッセージングシステムでのブローカーの設定

マルチサイトの耐障害性のあるメッセージングシステムの一部としてブローカーを設定するには、以下を行う必要があります。

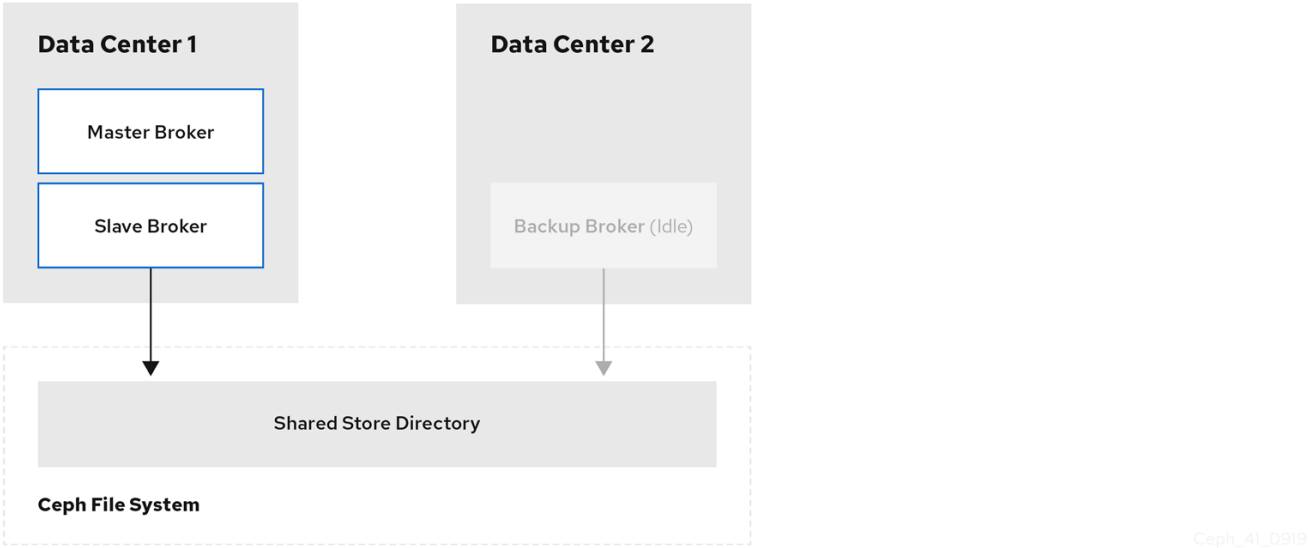
- データセンターに障害が発生した場合のライブブローカーから取得するアイドルバックアップブローカーを追加する
- Ceph クライアントロールですべてのブローカーサーバーの設定
- 共有ストア高可用性 (HA) ポリシーを使用するように各ブローカーを設定します。これは、ブローカーがメッセージングデータを保存する Ceph File System の場所を指定する

18.5.1. バックアップブローカーの追加

各データセンター内で、データセンターが停止した場合にシャットダウンするライブの master-slave ブローカーグループから引き継ぐアイドルバックアップブローカーを追加する必要があります。アイドルバックアップブローカーでライブマスターブローカーの設定を複製する必要があります。また、既存のブローカーと同じ方法でクライアント接続を受け入れるようにバックアップブローカーを設定する必要もあります。

後の手順で、既存の master-slave ブローカーグループに参加するようにアイドルバックアップブローカーを設定する方法を説明します。ライブの master-slave ブローカーグループのデータセンターとは別のデータセンターでアイドル状態のバックアップブローカーを見つける必要があります。また、データセンターに障害が発生した場合のみ、アイドルバックアップブローカーを手動で起動することが推奨されます。

以下の図はトポロジーの例を示しています。



関連情報

- 追加のブローカーインスタンスの作成方法は、Creating a standalone broker を参照してください。
- ブローカーネットワーク接続の設定は、ネットワーク接続: アクセプターおよびコネクター を参照してください。

18.5.2. Ceph クライアントとしてのブローカーの設定

耐障害性の高いシステムに必要なバックアップブローカーを追加したら、すべてのブローカーサーバーを Ceph クライアントロールで設定する必要があります。クライアントロールにより、ブローカーは Red Hat Ceph Storage クラスターにデータを保存できます。

Ceph クライアントを設定する方法は、Installing the Ceph Client Role を参照してください。

18.5.3. Configuring shared store high availability

Red Hat Ceph Storage クラスターは、異なるデータセンターのブローカーで利用可能な共有ストアを効果的に作成します。失敗時にメッセージがブローカークライアントが利用可能なままになるように、ライブバックアップグループの各ブローカーが以下を使用するように設定します。

- 共有ストア高可用性 (HA) ポリシー
- Ceph ファイルシステム内の同じジャーナル、ページング、および大きなメッセージディレクトリー。

以下の手順では、ライブバックアップグループのマスター、スレーブ、およびアイドルバックアップブローカーに共有ストア HA ポリシーを設定する方法を説明します。

手順

1. ライブバックアップグループの各ブローカーの **broker.xml** 設定ファイルを編集します。Ceph File System で、同じページング、バインディング、ジャーナル、および大きなメッセージディレクトリーを使用するように各ブローカーを設定します。

```
# 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. 以下に示すように、バックアップブローカーを HA ポリシー内のマスターとして設定します。この設定により、手動で起動すると、バックアップブローカーがマスターにすぐにになります。ブローカーはアイドル状態のバックアップであるため、アクティブなマスターブローカーに指定できる **failover-on-shutdown** パラメーターは適用されません。

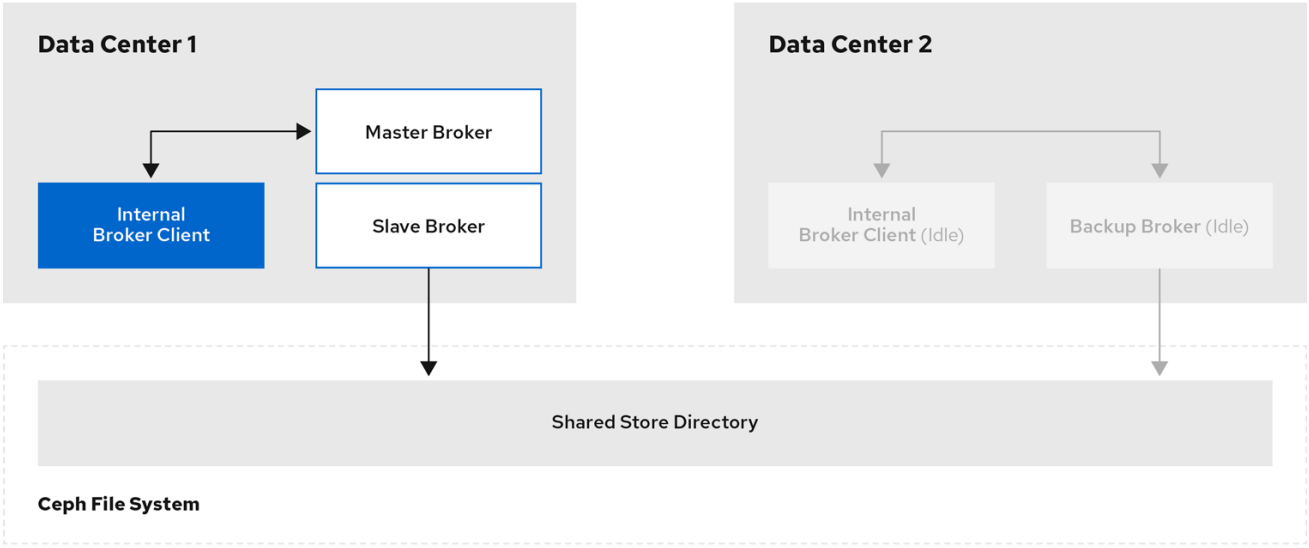
```
<configuration>
   <core>
      ...
      <ha-policy>
         <shared-store>
            <master>
            </master>
         </shared-store>
      </ha-policy>
      ...
   </core>
</configuration>
```

関連情報

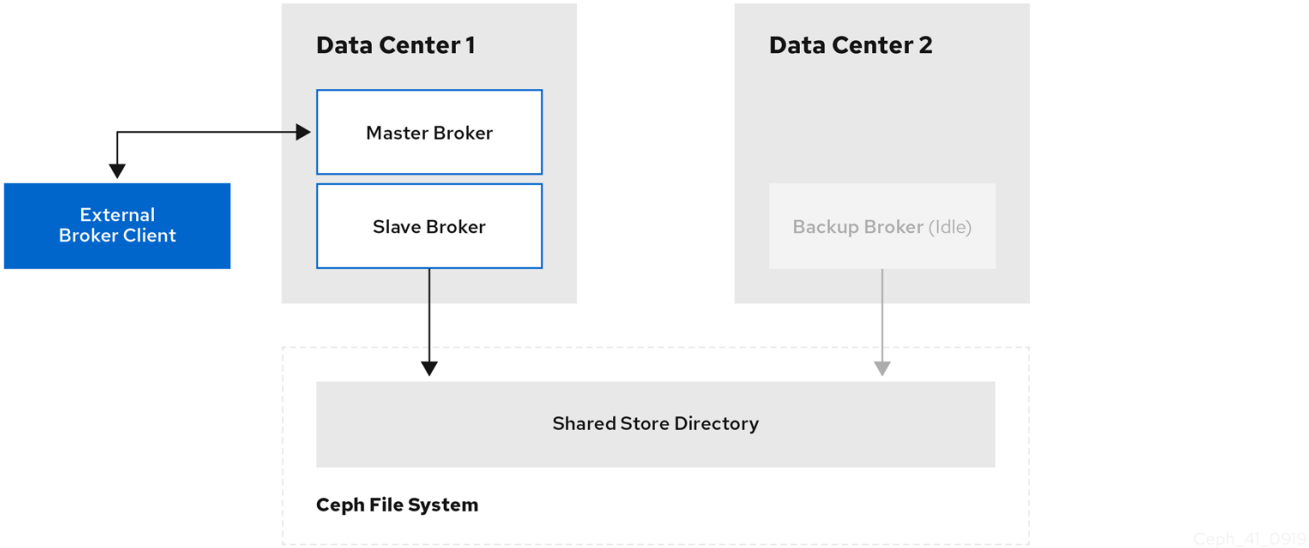
- ライブバックアップブローカーグループに共有ストアの高可用性ポリシーの設定に関する詳細は、Configuring shared store high availability を参照してください。

18.6. マルチサイトの耐障害性のあるメッセージングシステムでのクライアントの設定

内部クライアントアプリケーションは、ブローカーサーバーと同じデータセンターにあるマシンで実行されているアプリケーションです。以下の図はこのトポロジーを示しています。



外部クライアントアプリケーションは、ブローカーデータセンター外のマシンで実行されているものです。以下の図はこのトポロジーを示しています。



以下のサブセクションでは、データセンターが停止した場合に別のデータセンターのバックアップブローカーに接続するように内部および外部クライアントアプリケーションを設定する例を示します。

18.6.1. 内部クライアントの設定

データセンターが停止した場合に、内部クライアントアプリケーションがブローカーと共にシャットダウンします。この状況を軽減するには、別のデータセンターで利用可能なクライアントアプリケーションの別のインスタンスが必要です。データセンターが停止した場合に、バックアップクライアントを手動で起動し、手動で開始したバックアップブローカーに接続します。

バックアップクライアントがバックアップブローカーに接続できるようにするには、プライマリーデータセンターのクライアントと同じクライアント接続を設定する必要があります。

例

以下は、マスター / スレーブブローカーグループへの AMQ Core Protocol JMS クライアントの基本的な接続設定です。この例では、**host1** および **host2** はマスターおよびスレーブブローカーのホストサーバーです。

```
<ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("(tcp://host1:port,tcp://host2:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

データセンターが停止した場合にバックアップブローカーに接続するようにバックアップクライアントを設定するには、同様の接続設定を使用しますが、バックアップブローカーサーバーのホスト名のみを指定します。この例では、バックアップブローカーサーバーは **host3** です。

```
<ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("(tcp://host3:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

関連情報

ブローカーおよびクライアントネットワーク接続の設定に関する詳細は、以下を参照してください。

- ネットワーク接続: アクセプターとコネクター。
- クライアント側からの接続の設定。

18.6.2. 外部クライアントの設定

外部ブローカークライアントがデータセンターが停止した場合にメッセージングデータの生成または消費を継続できるようにするには、別のデータセンターのブローカーにフェイルオーバーするようにクライアントを設定する必要があります。マルチサイトのフォールトトレランスシステムの場合、障害が発生したときに手動で起動するバックアップブローカーにクライアントがフェイルオーバーするように設定します。

例

以下は、プライマリーマスターとスレーブグループが利用できない場合に、AMQ Core Protocol JMS および AMQP JMS クライアントがバックアップブローカーにフェイルオーバーするように設定する例です。この例では、**host1** および **host2** はプライマリーマスターおよびスレーブブローカーのホストサーバーですが、**host3** は、データセンターが停止した場合に手動で起動するバックアップブローカーのホストサーバーです。

- AMQ Core Protocol JMS クライアントを設定するには、クライアントが接続しようとするブローカーの順序付けされたリストにバックアップブローカーを含めます。

```
<ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("(tcp://host1:port,tcp://host2:port,tcp://host3:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

- AMQP JMS クライアントを設定するには、クライアントに設定するフェイルオーバー URI にバックアップブローカーを含めます。

```
failover:(amqp://host1:port,amqp://host2:port,amqp://host3:port)?
jms.clientID=foo&failover.maxReconnectAttempts=20
```

関連情報

フェイルオーバーの設定に関する詳細は、以下を行います。

- AMQ Core Protocol JMS クライアントについては、再接続およびフェイルオーバー を参照してください。
- AMQP JMS クライアントについては、Failover optionsを参照してください。
- サポートされるその他のクライアントについては、Red Hat AMQ 7.7 の製品ドキュメント の AMQ Clients セクションのクライアント固有のドキュメントを参照してください。

18.7. データセンターの停止時のストレージクラスターの正常性の確認

フォールトトレランスのために Red Hat Ceph Storage クラスターを設定すると、データセンターのいずれかに障害が発生しても、クラスターはデータを損失せずに動作が低下します。

この手順では、動作が低下した状態でクラスターのステータスを検証する方法を説明します。

手順

1. Ceph Storage クラスターのステータスを確認するには、**health** または **status** コマンドを使用します。

```
# ceph health
# ceph status
```

2. コマンドラインでクラスターの続行中のイベントを監視するには、新しいターミナルを開きます。次に、以下を入力します。

```
# ceph -w
```

上記のコマンドを実行すると、ストレージクラスターが実行中であるものの、動作が低下したことを示す出力が表示されます。具体的には、以下のような警告が表示されます。

```
health: HEALTH_WARN
        2 osds down
        Degraded data redundancy: 42/84 objects degraded (50.0%), 16 pgs unclean, 16 pgs degraded
```

関連情報

- Red Hat Ceph Storage クラスターの正常性のモニターリングについての詳細は、Monitoring を参照してください。

18.8. データセンターの停止時のメッセージングの持続性の維持

以下の手順は、データセンターの停止時にブローカーおよび関連するメッセージングデータをクライアントが利用できる状態にする方法を説明します。特に、データセンターに障害が発生した場合に、以下を行う必要があります。

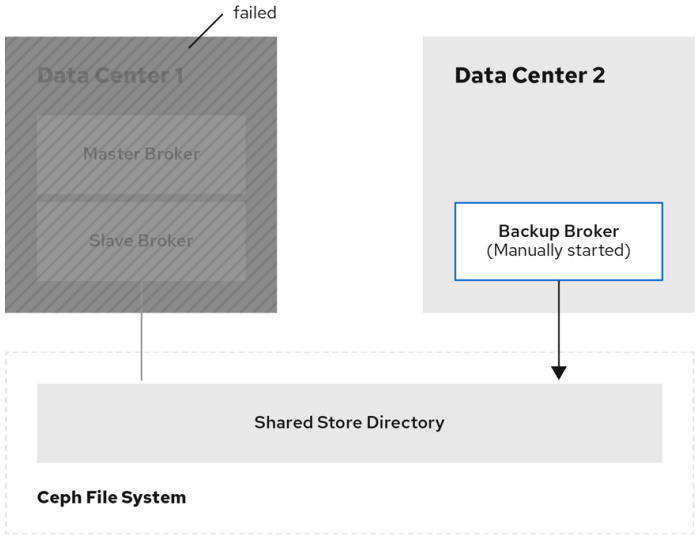
- 失敗したデータセンターのブローカーを引き継ぐために作成したアイドルバックアップブローカーを手動で開始します。
- 内部クライアントまたは外部クライアントを新しいアクティブなブローカーに接続します。

前提条件

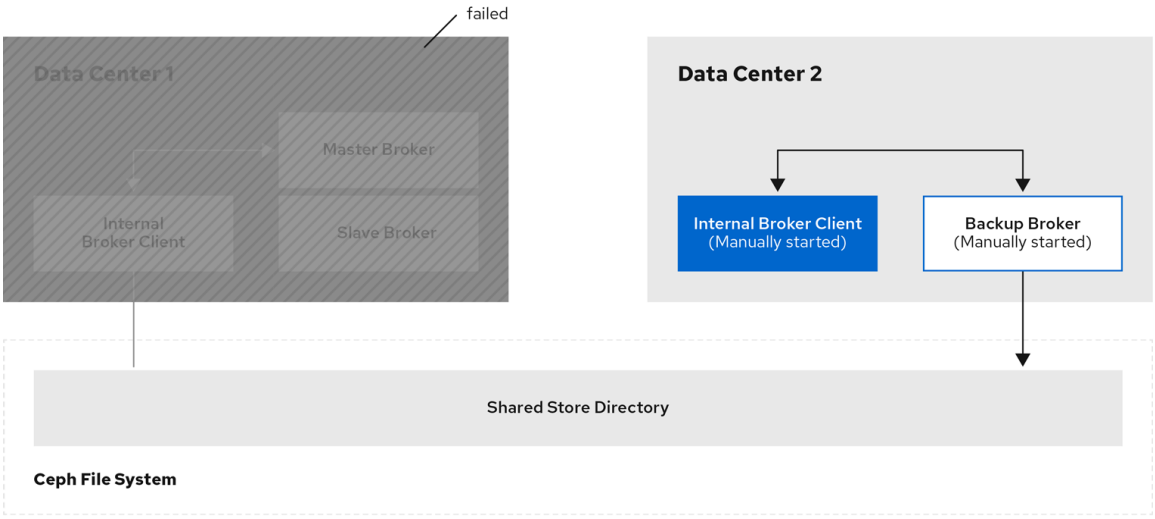
- 以下が必要になります。
  - Red Hat Ceph Storage クラスターをインストールおよび設定していること。詳細は、Installing Red Hat Ceph Storage および Configuring a Red Hat Ceph Storage cluster を参照してください。
  - Ceph File System をマウントしました。詳細は、Mounting the Ceph File System on your broker servers を参照してください。
  - データセンターに障害が発生した場合のライブブローカーから取得するためにアイドルバックアップブローカーを追加。詳細は、Adding backup brokers を参照してください。
  - Ceph クライアントロールを使用したブローカーサーバーを設定。詳細は、Configuring brokers as Ceph clients を参照してください。
  - 共有ストア高可用性 (HA) ポリシーを使用するように各ブローカーを設定し、各ブローカーでメッセージングデータを保存する場所を指定。詳細は、Configuring shared store high availability を参照してください。
  - データセンターが停止した場合にバックアップブローカーに接続するようにクライアントを設定する。詳細は、Configuring clients in a multi-site, fault-tolerant messaging system を参照してください。

手順

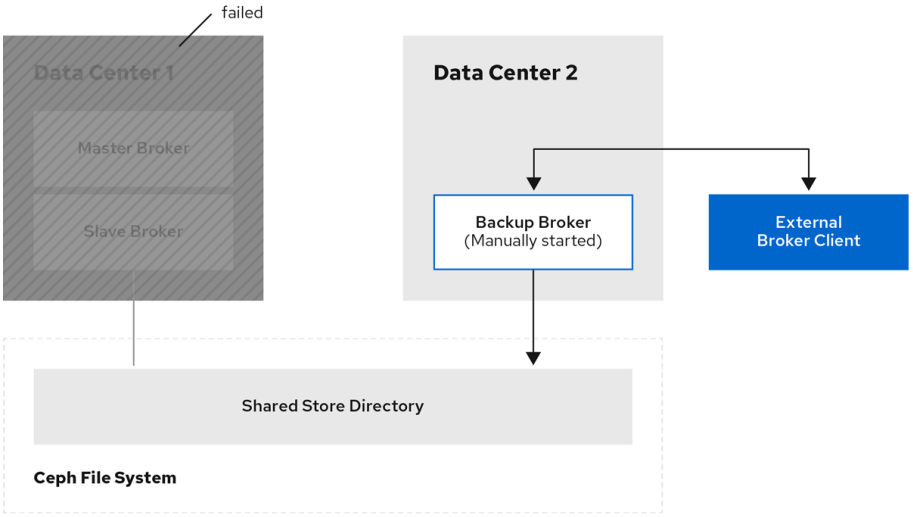
1. 失敗したデータセンターの各 master-slave ブローカーペアに対して、追加したアイドルバックアップブローカーを手動で起動します。



2. クライアント接続を再確立します。
   a. 障害の発生したデータセンターで内部クライアントを使用している場合は、作成したバックアップクライアントを手動で起動します。Configuring clients in a multi-site, fault-tolerant messaging system で説明されているように、手動で開始したバックアップブローカーに接続するようにクライアントを設定する必要があります。
   以下の図は、新しいトポロジーを示しています。



b. 外部クライアントがある場合、外部クライアントを手動で新しいアクティブなブローカーに接続するか、設定に基づいて、クライアントが新しいアクティブなブローカーに自動的にフェイルオーバーすることを確認します。詳細は、Configuring external clients を参照してください。
以下の図は、新しいトポロジーを示しています。



18.9. 以前に失敗したデータセンターの再起動

以前失敗したデータセンターがオンラインに戻る場合は、以下の手順に従ってメッセージングシステムの元の状態を復元します。

- Red Hat Ceph Storage クラスターのノードをホストするサーバーを再起動します。
- メッセージングシステムでブローカーを再起動します。
- クライアントアプリケーションから復元されたブローカーへの接続を再確立します。

以下のサブセクションでは、これらのステップを実行する方法を示します。

18.9.1. ストレージクラスターサーバーの再起動

以前障害の発生したデータセンターで Monitor、metadata Server、Manager、および Object Storage Device (OSD) ノードを再起動する際に、Red Hat Ceph Storage クラスターの自己修復を行い、データの冗長性を完全に復元します。このプロセス中に、Red Hat Ceph Storage は必要に応じて復元された OSD ノードにデータを自動的にバックフィルします。

ストレージクラスターが自動的に自己修復され、データ冗長性を完全に復元していることを確認するには、上記の Verifying storage cluster health during a data center outage に記載のコマンドを使用します。これらのコマンドを再実行すると、以前の **HEALTH_WARN** メッセージによるパフォーマンスの低下により、100% に戻るまで改善が開始することが分かります。

18.9.2. ブローカーサーバーの再起動

以下の手順では、ストレージクラスターが degraded 状態で機能しなくなったときにブローカーサーバーを再起動する方法を説明します。

手順

1. データセンターが停止したときに手動で開始したバックアップブローカーに接続されているクライアントアプリケーションを停止します。
2. 手動で起動したバックアップブローカーを停止します。
   a. Linux の場合:

```
BROKER_INSTANCE_DIR/bin/artemis stop
```

   b. Windows の場合:

```
BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
```

3. 以前はデータセンターに障害が発生し、元のマスターおよびスレーブブローカーを再起動します。
   a. Linux の場合:

```
BROKER_INSTANCE_DIR/bin/artemis run
```

   b. Windows の場合:

```
BROKER_INSTANCE_DIR\bin\artemis-service.exe start
```

元のマスターブローカーは、再起動時にそのロールを master として自動的に再開します。

18.9.3. クライアント接続の再設定

ブローカーサーバーを再起動したら、クライアントアプリケーションをこれらのブローカーに再接続します。以下のサブセクションでは、内部と外部の両方のクライアントアプリケーションを再接続する方法を説明します。

18.9.3.1. 内部クライアントの再接続

内部クライアントは、復元されたブローカーと同じデータセンターで実行されているクライアントです。内部クライアントを再接続するには、そのクライアントを再起動します。各クライアントアプリケーションは、接続設定に指定された復元されたマスターブローカーに再接続します。

ブローカーおよびクライアントネットワーク接続の設定に関する詳細は、以下を参照してください。

- ネットワーク接続: アクセプターとコネクター
- クライアント側からの接続の設定

18.9.3.2. 外部クライアントの再接続

外部クライアントは、以前に失敗したデータセンター外で実行されているものです。クライアントの種類と、外部ブローカークライアントの設定 に関する情報に基づいて、クライアントをバックアップブローカーに自動的にフェイルオーバーするように設定するか、この接続を手動で確立します。以前に失敗したデータセンターを復元する際に、以下で説明されているように、クライアントから復元されたマスターブローカーへの接続を再確立します。

- 外部クライアントをバックアップブローカーに自動的にフェイルオーバーするように設定した場合、バックアップブローカーをシャットダウンして元のマスターブローカーを再起動すると、クライアントは元のマスターブローカーに自動的に失敗します。
- データセンターが停止したときに外部クライアントをバックアップブローカーに手動で接続した場合、再起動する元のマスターブローカーに手動でクライアントを再接続する必要があります。

第19章 ロギング

AMQ Broker は JBoss Logging フレームワークを使用してロギングを実行し、**BROKER_INSTANCE_DIR/etc/logging.properties** 設定ファイルで設定できます。この設定ファイルはキーと値のペアの一覧です。

以下に示すように、**logging.properties** 設定ファイルの **ロガー** キーに追加して、ブローカー設定でロガーを指定します。

```
loggers=org.eclipse.jetty,org.jboss.logging,org.apache.activemq.artemis.core.server,org.apache.activem
q.artemis.utils,org.apache.activemq.artemis.journal,org.apache.activemq.artemis.jms.server,org.apache
activemq.artemis.integration.bootstrap,org.apache.activemq.audit.base,org.apache.activemq.audit.mess
age,org.apache.activemq.audit.resource
```

AMQ Broker で利用可能なロガーを以下の表に示します。

ロガー	説明
org.jboss.logging	ルートロガー。他のブローカーロガーによって処理されない呼び出しをログに記録します。
org.apache.activemq.artemis.core.server	ブローカーコアをログに記録します。
org.apache.activemq.artemis.utils	ユーティリティー呼び出しのログ
org.apache.activemq.artemis.journal	Journal 呼び出しのログ
org.apache.activemq.artemis.jms	JMS 呼び出しをログに記録します。
org.apache.activemq.artemis.integration.bootstrap	ブートストラップ呼び出しのログ
org.apache.activemq.audit.base	すべての JMX オブジェクトメソッドへのアクセスをログに記録します。
org.apache.activemq.audit.message	実稼働、消費消費、メッセージのブラウジングなどのメッセージ操作をロギングします。
org.apache.activemq.audit.resource	JMX または AMQ Broker 管理コンソールからの認証イベント、作成または削除、および管理コンソールでのメッセージの参照をログに記録します。

また、以下のように **logger.handlers** キーに設定されたデフォルトのロギングハンドラーが 2 つあります。

```
logger.handlers=FILE,CONSOLE
```

logger.handlers=FILE

ロガーはログエントリーをファイルに出力します。

logger.handlers=CONSOLE

ロガーはログエントリーを AMQ Broker 管理コンソールに出力します。

19.1. ログレベルを変更する

すべてのロガーのデフォルトのロギングレベルは **INFO** で、以下のようにルートロガーで設定されます。

```
logger.level=INFO
```

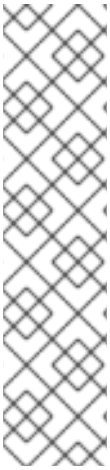
以下に示すように、他のすべてのロガーのログレベルを個別に設定できます。

```
logger.org.apache.activemq.artemis.core.server.level=INFO
logger.org.apache.activemq.artemis.journal.level=INFO
logger.org.apache.activemq.artemis.utils.level=INFO
logger.org.apache.activemq.artemis.jms.level=INFO
logger.org.apache.activemq.artemis.integration.bootstrap.level=INFO
logger.org.apache.activemq.audit.base.level=INFO
logger.org.apache.activemq.audit.message.level=INFO
logger.org.apache.activemq.audit.resource.level=INFO
```

利用可能なロギングレベルは、以下の表で説明されています。ロギングレベルは、最低詳細から順に昇順に一覧表示されます。

レベル	説明
FATAL	重要なサービス障害を示すイベントには、FATAL ロギングレベルを使用します。サービスが FATAL エラーを発行すると、あらゆる種類の要求を実行できません。
ERROR	リクエストが中断されたことを示すイベントまたはリクエストを処理する機能を示す ERROR ロギングレベルを使用します。このエラーの発生時に要求の処理を続行するには、サービスの **ある程度の** 容量が必要です。
WARN	重要でないサービスエラーを示す可能性のあるイベントの WARN ロギングレベルを使用します。再開可能なエラー、またはリクエスト内のマイナーな違反がこの説明を満たすことが期待されます。WARN と ERROR の違いは、アプリケーション開発者にとっての 1 つです。これを区別するための簡単な条件として、エラーがテクニカルサポートをシークする必要が生じるかどうかが挙げられます。エラーがテクニカルサポートを必要とする場合は、ロギングレベルを ERROR に設定します。それ以外の場合は、レベルを WARN に設定します。

レベル	説明
INFO	サービスのライフサイクルイベントおよびその他の重要な関連情報には INFO ロギングレベルを使用します。特定のサービスカテゴリーの INFO レベルメッセージは、サービスの状態を明確に示す必要があります。
DEBUG	ライフサイクルイベントに関する追加情報を伝えるログメッセージには DEBUG ロギングレベルを使用します。開発者向けの情報や技術サポートに必要な詳細情報については、このログレベルを使用します。DEBUG ロギングレベルを有効にすると、サーバー要求の数と比例して JBoss サーバーのログが増加 **しないはず** です。所定のサービスカテゴリーの DEBUG および INFO レベルのメッセージは、サービスがどの状態にあるか、ポート、インターフェイス、ログファイルなど、どのブローカーリソースを使用しているかを明確に示する必要があります。
TRACE	リクエストアクティビティーに直接関連するログメッセージに TRACE ロギングレベルを使用します。このようなメッセージは、ロガーカテゴリーの優先度のしきい値でメッセージがレンダリングされることを示す場合を除き、ロガーに送信しないでください。**Logger.isTraceEnabled()** メソッドを使用して、カテゴリーの優先度のしきい値が有効かどうかを判断します。TRACE レベルのロギングは、必要に応じてブローカーの動作のディーププローティングを有効にします。TRACE ロギングレベルが有効な場合、JBoss ログのメッセージ数は少なくとも **a** * **N** まで増えます。ここで、**N** はブローカーによって受信されるリクエストの数で、**a** は何らかの定数になります。トレースされるリクエスト処理層によっては、サーバーログが **N** のべき乗に増加する可能性があります。



注記

- **INFO** は、**logger.org.apache.activemq.audit.base**、**logger.org.apache.activemq.audit.message**、および **logger.org.apache.activemq.audit.resource** 監査ロガーの唯一の利用可能なロギングレベルです。
- ルートロガーに指定されたロギングレベルは、他のロガーにより詳細なログレベルが指定されている場合でも、**すべて** のロガーに対して最も詳細なログレベルを決定します。たとえば、**org.apache.activemq.artemis.utils** に **DEBUG** の指定されたロギングがあり、ルートロガーの **org.jboss.logging** のロギングレベルが **WARN** であるとします。この場合、両方のロガーはログレベル **WARN** を使用します。

19.2. 監査ロギングの有効化

有効化には 3 つの監査ロガーを使用できます。これは、ベース監査ロガー、メッセージ監査ロガー、およびリソース監査ロガーです。

ベース監査ロガー (org.apache.activemq.audit.base)

アドレスやキューの作成や削除など、すべての JMX オブジェクトメソッドへの **アクセス** をログに記録します。ログには、これらの操作が成功したか失敗したかは表示 **されません**。

メッセージ監査ロガー (org.apache.activemq.audit.message)

実稼働、消費、メッセージの参照などのメッセージ関連のブローカー操作をログに記録します。

リソース監査ロガー (org.apache.activemq.audit.resource)

クライアント、ルート、および AMQ Broker 管理コンソールから認証の成功または失敗をログに記録します。また、JMX または管理コンソールからキューの作成、更新、または削除をログに記録し、管理コンソールでメッセージを閲覧します。

各監査ロガーは、互いに独立して有効にできます。デフォルトでは、各監査ロガーは無効になっています (つまり、ロギングレベルは **ERROR** に設定され、監査ロガーの有効なログレベルではありません)。監査ロガーの 1 つを有効にするには、ロギングレベルを **INFO** に設定します。以下に例を示します。

```
logger.org.apache.activemq.audit.base.level=INFO
```

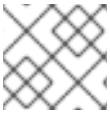
重要

メッセージ監査ロガーは、ブローカーのパフォーマンス集約型パスで実行されます。ロガーを有効にすると、特にブローカーがメッセージング負荷の高い状態で実行されている場合は、ブローカーのパフォーマンスに悪影響を与える可能性があります。高スループットが必要なメッセージングシステムでは、メッセージ監査ロガーの使用は推奨されません。

19.3. コンソールロギングの設定

以下のキーを使用してコンソールロギングを設定できます。

```
handler.CONSOLE=org.jboss.logmanager.handlers.ConsoleHandler
handler.CONSOLE.properties=autoFlush
handler.CONSOLE.level=DEBUG
handler.CONSOLE.autoFlush=true
handler.CONSOLE.formatter=PATTERN
```



注記

handler.CONSOLE は **logger.handlers** キーで指定された名前を参照します。

コンソールのロギング設定オプションは、以下の表で説明されています。

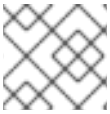
プロパティー	説明
name	ハンドラー名
encoding	ハンドラーによって使用される文字エンコーディング

プロパティー	説明
level	メッセージレベルを記録したログレベル。この値よりも小さいメッセージレベルは破棄されます。
formatter	フォーマッターを定義します。「ロギング形式の設定」 を参照してください。
autoflush	書き込みのたびにログを自動的にフラッシュするかどうかを設定します。
target	コンソールハンドラーのターゲット。値は SYSTEM_OUT または SYSTEM_ERR のいずれかになります。

19.4. ファイルロギングの設定

以下のキーを使用して、ファイルロギングを設定できます。

```
handler.FILE=org.jboss.logmanager.handlers.PeriodicRotatingFileHandler
handler.FILE.level=DEBUG
handler.FILE.properties=suffix,append,autoFlush,fileName
handler.FILE.suffix=.yyyy-MM-dd
handler.FILE.append=true
handler.FILE.autoFlush=true
handler.FILE.fileName=${artemis.instance}/log/artemis.log
handler.FILE.formatter=PATTERN
```



注記

handler.FILE は **logger.handlers** キーで指定された名前を参照します。

ファイルのロギング設定オプションについては、以下の表で説明されています。

プロパティー	説明
name	ハンドラー名
encoding	ハンドラーによって使用される文字エンコーディング
level	メッセージレベルを記録したログレベル。この値よりも小さいメッセージレベルは破棄されます。
formatter	フォーマッターを定義します。「ロギング形式の設定」 を参照してください。

プロパティー	説明
autoflush	書き込みのたびにログを自動的にフラッシュするかどうかを設定します。
append	ターゲットファイルに追加するかどうかを指定します。
file	パスと任意の相対パスで設定されるファイルの説明。

19.5. ロギング形式の設定

フォーマッターはログメッセージの表示方法を記述します。デフォルト設定は次のとおりです。

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

上記の設定では、**%s** がメッセージで、**%E** が存在する場合は例外となります。

形式は Log4J 形式と同じです。詳細は、こちら を参照してください。

19.6. クライアントまたは埋め込みサーバーのロギング

クライアントでロギングを有効にする場合は、クライアントのクラスパスに JBoss ロギング JAR を含める必要があります。Maven を使用している場合は、以下の依存関係を追加します。

```
<dependency>
  <groupId>org.jboss.logmanager</groupId>
  <artifactId>jboss-logmanager</artifactId>
  <version>1.5.3.Final</version>
</dependency>
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>artemis-core-client</artifactId>
  <version>1.0.0.Final</version>
</dependency>
```

Java プログラムの開始時に設定する必要があるプロパティーは 2 つあります。1 つ目は、Log Manager が JBoss Log Manager を使用するように設定することです。これは、'-Djava.util.logging.manager' プロパティーを設定して行います。以下に例を示します。

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

2 つ目は、使用する **logging.properties** ファイルの場所を設定します。これは、**-Dlogging.configuration** プロパティーに有効な URL を設定して行います。以下に例を示します。

```
-Dlogging.configuration=file:///home/user/projects/myProject/logging.properties
```

以下は、クライアントの典型的な **logging.properties** ファイルです。

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

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

# Root logger handlers
logger.handlers=FILE,CONSOLE

# Console handler configuration
handler.CONSOLE=org.jboss.logmanager.handlers.ConsoleHandler
handler.CONSOLE.properties=autoFlush
handler.CONSOLE.level=FINE
handler.CONSOLE.autoFlush=true
handler.CONSOLE.formatter=PATTERN

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

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

19.7. AMQ BROKER プラグインのサポート

AMQ はカスタムプラグインをサポートします。このプラグインを使用して、デバッグログでのみ使用できる各種のさまざまな種類のイベントに関する情報をログに記録できます。複数のプラグインは、登録、関連付け、および実行が可能です。プラグインは登録順序に基づいて実行されます。つまり、最初に登録されたプラグインは常に最初に実行されます。

カスタムプラグインを作成して、**ActiveMQServerPlugin** インターフェイスを使用して実装できます。このインターフェイスにより、プラグインがクラスパスに置かれ、ブローカーに登録されます。すべてのインターフェイスメソッドはデフォルトで実装されるため、実装する必要がある動作のみを追加する必要があります。

19.7.1. プラグインのクラスパスへの追加

関連する **.jar** ファイルを **BROKER_INSTANCE_DIR/lib** ディレクトリーに追加して、カスタム作成したブローカープラグインをブローカーランタイムに追加します。

埋め込みシステムを使用している場合は、**.jar** ファイルを埋め込みアプリケーションの通常のクラスパス下に置きます。

19.7.2. プラグインの登録

broker.xml 設定ファイルに **broker-plugins** 要素を追加してプラグインを登録する必要があります。**property** 子要素を使用して、プラグイン設定値を指定できます。これらのプロパティーは、プラグインがインスタンス化された後にプラグインの init (Map<String, String>) 操作を読み取り、渡されます。

```
<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>
```

19.7.3. プログラムによるプラグインの登録

プラグインをプログラムで登録するには、**registerBrokerPlugin()** メソッドを使用してプラグインの新規インスタンスを渡します。以下の例は、**UserPlugin** プラグインの登録を示しています。

```
Configuration config = new ConfigurationImpl();

config.registerBrokerPlugin(new UserPlugin());
```

19.7.4. 特定のイベントのロギング

デフォルトでは、AMQ Broker は **LoggingActiveMQServerPlugin** プラグインを提供し、特定のブローカーイベントをログに記録します。**LoggingActiveMQServerplugin** プラグインはデフォルトではコメントアウトされ、情報はログに記録されません。

以下の表は、各プラグインプロパティーについて説明しています。設定プロパティーの値を **true** に設定してイベントをログに記録します。

プロパティー	説明
LOG_CONNECTION_EVENTS	接続が作成または破棄されたときに情報を記録します。
LOG_SESSION_EVENTS	セッションの作成時または終了時に情報をログに記録します。
LOG_CONSUMER_EVENTS	コンシューマーが作成または閉じられたときの情報をログに記録します。
LOG_DELIVERING_EVENTS	メッセージがコンシューマーに配信され、メッセージがコンシューマーによって確認されると、情報を記録します。
LOG_SENDING_EVENTS	メッセージがアドレスに送信され、メッセージがブローカー内にルーティングされたときの情報をログに記録します。

LOG_INTERNAL_EVENTS	キューが作成または破棄されたとき、メッセージが期限切れになったとき、ブリッジのデプロイ時、および重大な障害が発生するときに、情報をログに記録します。
LOG_ALL_EVENTS	上記のすべてのイベントの情報をログに記録します。

接続イベントをログに記録するように **LoggingActiveMQServerPlugin** プラグインを設定するには、**broker.xml** 設定ファイルの **<broker-plugins>** セクションのコメントを解除します。すべてのイベントの値は、コメントされたデフォルト例では **true** に設定されます。

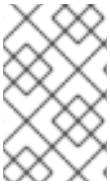
```
<configuration ...>
...
<!-- Uncomment the following if you want to use the Standard LoggingActiveMQServerPlugin plugin
to log in events -->
          <broker-plugins>
         <broker-plugin class-
name="org.apache.activemq.artemis.core.server.plugin.impl.LoggingActiveMQServerPlugin">
            <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>
```

<broker-plugins> セクション内で設定パラメーターを変更した場合は、設定の更新を再ロードするためにブローカーを再起動する必要があります。これらの設定変更は、**configuration-file-refresh-period** 設定に基づいてリロード **されません**。

ログレベルを **INFO** に設定すると、イベントの発生後にエントリーがログに記録されます。ログレベルが **DEBUG** に設定されている場合、イベントの前後でログエントリーが生成されます (例 : **beforeCreateConsumer()** および **afterCreateConsumer())**。ログレベルが **DEBUG** に設定されている場合、ロガーは利用可能な場合に通知の詳細情報をログに記録します。

付録A アクセプターおよびコネクター設定パラメーター

以下の表は、Netty ネットワーク接続の設定に使用される利用可能なパラメーターの一部を示しています。パラメーターとその値は、接続文字列の URI に追加されます。詳細は、ネットワーク接続: アクセプターとコネクター を参照してください。各テーブルは、名前別にパラメーターを一覧表示し、アクセプターまたはコネクターと使用できるか、またはその両方とともに使用するかをメモします。たとえば、アクセプターでしか使用するなど、一部のパラメーターを使用できます。



注記

すべての Netty パラメーターは **org.apache.activemq.artemis.core.remoting.impl.netty.TransportConstants** クラスで定義されます。カスタマーポータル では、ソースコードをダウンロードできます。

表A.1 Netty TCP パラメーター

パラメーター	用途	説明
batchDelay	両方	パケットをアクセプターまたはコネクターに書き込む前に、ブローカーは **batchDelay** ミリ秒の書き込みをバッチ処理するよう設定できます。これにより、非常に小さなメッセージで全体的なスループットが増える可能性があります。これは、メッセージ転送の平均レイテンシーが増加するため、経費します。デフォルト値は **0** ミリ秒です。
connectionsAllowed	アクセプター	アクセプターが許可する接続の数を制限します。この制限に達すると、DEBUG レベルのメッセージがログに出力され、接続は拒否されました。使用中のクライアントのタイプによって、接続が拒否されたときに何が起こるかが決まります。
directDeliver	両方	メッセージがサーバーに到達し、待機しているコンシューマーに配信されると、デフォルトでは、メッセージが到達した同じスレッドで配信が実行されます。これにより、メッセージが比較的小さく、コンシューマーの数が少ない環境では適切な待ち時間が発生しますが、特にマルチコアマシンではスループットとスケーラビリティー全体のコストが発生します。レイテンシーを最小限に抑え、スループットが低い場合は、**directDeliver** のデフォルト値 (**true**) を使用できます。レイテンシーに若干ヒットするが、スループットが最も高い場合は **directDeliver** を **false** に設定します。

パラメーター	用途	説明
handshake-timeout	アクセプター	承認されていないクライアントが多数の接続を開かないようにし、開いた状態にします。各接続にはファイルハンドルが必要なため、他のクライアントでは利用できないリソースを消費します。 このタイムアウトにより、接続が認証されずにリソースを消費できる時間を制限します。接続が認証されると、リソース制限設定を使用してリソース消費を制限できます。 デフォルト値は **10** 秒に設定されます。これは、他の整数値に設定できます。このオプションをオフにするには、0 または負の整数に設定します。 タイムアウト値を編集したら、ブローカーを再起動する必要があります。
localAddress	コネクター	リモートアドレスへの接続時にクライアントが使用するローカルアドレスを指定します。通常、これはアプリケーションサーバーで使用され、Embeddent 接続に使用されるアドレスを制御するために Embedded を実行する場合に使用されます。local-address が設定されていない場合、コネクターは利用可能なローカルアドレスを使用します。
localPort	コネクター	リモートアドレスへの接続時にクライアントが使用するローカルポートを指定します。通常、これはアプリケーションサーバーで使用され、Embeddent 接続に使用されるポートを制御するために Embedded を実行する場合に使用されます。デフォルトに (0) が使用される場合、コネクターによりシステムで一時ポートを取得することが許可されます。有効なポートは 0 から 65535 です。
nioRemotingThreads	両方	NIO を使用するように設定されている場合には、 ブローカーはデフォルトで受信パケットを処理するために **Runtime.getRuntime().availableProcessors()** によって報告されるコア (または hyper-threads) の数の 3 倍のスレッドを使用します。この値をオーバーライドする場合は、このパラメーターを指定してスレッド数を設定できます。このパラメーターのデフォルト値は **-1** です。これは、**Runtime.getRuntime().availableProcessors()** * 3 から派生する値を使用することを意味します。
tcpNoDelay	両方	**true** の場合、Nagle アルゴリズム が有効になります。これは、Java(クライアント) ソケットオプション です。デフォルト値は **true** です。
tcpReceiveBufferSize	両方	TCP 受信バッファーのサイズ (バイト単位) を決定します。デフォルト値は **32768** です。

パラメーター	用途	説明
tcpSendBufferSize	両方	TCP 送信バッファーのサイズ (バイト単位) を決定します。デフォルト値は **32768** です。 TCP バッファーサイズは、ネットワークの帯域幅およびレイテンシーに従って調整する必要があります。 つまり、TCP の送信/受信バッファーサイズは以下のように計算する必要があります。 buffer_size = bandwidth * RTT 帯域幅とは秒単位で、ネットワークラウンドトリップタイム (RTT) は秒単位になります。RTT は、**ping** ユーティリティーを使用して簡単に測定できます。 高速ネットワークでは、デフォルトからバッファーサイズを増やす必要がある場合があります。

表A.2 Netty HTTP パラメーター

パラメーター	用途	説明
httpClientIdleTime	アクセプター	接続を維持するために空の HTTP 要求を送信する前にクライアントがアイドル状態でいられる期間。
httpClientIdleScanPeriod	アクセプター	アイドル状態のクライアントに対してスキャンを行う頻度 (ミリ秒単位)。
httpEnabled	アクセプター	不要になりました。単一ポートがサポートされる場合、ブローカーは HTTP が使用されているかどうかを自動的に検出し、それ自体を設定するようになりました。
httpRequiresSessionId	両方	**true** の場合、クライアントは最初の呼び出し後にセッション ID を受け取るまで待機します。HTTP コネクターがサーブレットアクセプターに接続する場合に使用されます。この設定は推奨されません。
httpResponseTime	アクセプター	接続を維持するために空の HTTP 応答を送信する前にサーバーが待機する時間。
httpServerScanPeriod	アクセプター	応答が必要なクライアントに対してスキャンを行う頻度 (ミリ秒単位)。

表A.3 Netty TLS/SSL パラメーター

パラメーター	用途	説明

パラメーター	用途	説明
enabledCipherSuites	両方	SSL 通信に使用される暗号スイートのコンマ区切りリスト。デフォルト値は空で、JVM のデフォルトが使用されます。
enabledProtocols	両方	SSL 通信に使用されるプロトコルのコンマ区切りリスト。デフォルト値は空で、JVM のデフォルトが使用されます。
forceSSLParameters	コネクター	このコネクターの SSL コンテキストを設定するために、JVM システムプロパティー (**javax.net.ssl** および AMQ Broker システムプロパティーの両方を含む) ではなく、コネクターのパラメーターとして設定される SSL 設定を使用するかどうかを制御します。 有効な値は **true** または **false** です。デフォルト値は **false** です。
keyStorePassword	両方	アクセプターで使用されると、サーバー側のキーストアのパスワードになります。 コネクターで使用されると、クライアント側のキーストアのパスワードになります。双方向 SSL(つまり相互認証) を使用している場合は、これはコネクターにのみ関係します。この値はサーバー上で設定可能ですが、ダウンロードしてクライアントで使用します。クライアントでサーバーの設定と異なるパスワードを使用する必要がある場合は、カスタムの **javax.net.ssl.keyStorePassword** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.keyStorePassword** システムプロパティーを使用してサーバー側の設定をオーバーライドできます。ActiveMQ 固有のシステムプロパティーは、クライアントの別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に便利です。
keyStorePath	両方	アクセプターで使用される場合は、サーバーの証明書を保持するサーバーの SSL キーストアへのパスになります (自己署名または認証局によって署名されているかどうか)。 コネクターで使用される場合、これはクライアント証明書を保持するクライアント側の SSL キーストアへのパスとなります。双方向 SSL(つまり相互認証) を使用している場合は、これはコネクターにのみ関係します。この値はサーバー上で設定されますが、ダウンロードしてクライアントで使用します。クライアントでサーバーの設定と異なるパスを使用する必要がある場合は、カスタムの **javax.net.ssl.keyStore** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.keyStore** システムプロパティーのいずれかを使用してサーバー側の設定をオーバーライドできます。ActiveMQ 固有のシステムプロパティーは、クライアントの別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に便利です。

パラメーター	用途	説明
needClientAuth	アクセプター	このプロパティーは、このアクセプターに接続するクライアントに双方向 SSL が必要であることを伝えます。有効な値は **true** または **false** です。デフォルト値は **false** です。
sslEnabled	両方	SSL を有効にするには **true** にする必要があります。デフォルト値は **false** です。
trustManagerFactoryPlugin	両方	**org.apache.activemq.artemis.api.core.Trust Manager Factory Plugin**を実装するクラスの名前を定義します。 これは、**javax.net.ssl.TrustManagerFactory** を返す単一のメソッドを持つ簡単なインターフェイスです。**TrustManagerFactory** は、基礎となる **javax.net.ssl.SSLContext** が初期化されるときに使用されます。これにより、ブローカーおよびクライアントの信頼について詳細にわたるカスタマイズが可能になります。 **trustManagerFactoryPlugin** の値は、トラストマネージャーに適用される他のすべての SSL パラメーターよりも優先されます (例 : **trustAll**、**truststoreProvider**、**truststorePath**、**truststorePassword**、および **crlPath**)。 ブローカーの Java クラスパスに、指定したプラグインを配置する必要があります。**BROKER_INSTANCE_DIR/lib** ディレクトリーは、デフォルトでクラスパスに含まれているため、使用することができます。
trustStorePassword	両方	アクセプターで使用されると、サーバー側のトラストストアのパスワードになります。双方向 SSL(つまり相互認証) を使用している場合、アクセプターにのみ関係します。 コネクターで使用されると、クライアント側のトラストストアのパスワードになります。この値はサーバー上で設定可能ですが、ダウンロードしてクライアントで使用します。クライアントがサーバーで設定されているものとは異なるパスワードを使用する必要がある場合は、通常の**javax.net.ssl.trust Store Password**システムプロパティーまたは ActiveMQ 固有の**org.apache.activemq.ssl.trust Store Password**システムプロパティーを使用して、サーバー側の設定を上書きできます。ActiveMQ 固有のシステムプロパティーは、クライアントの別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に便利です。

パラメーター	用途	説明
sniHost	両方	アクセプターで使用される場合、**sniHost** は受信 SSL 接続の **server_name** 拡張と一致するために使用される正規表現です (この拡張機能に関する詳細は、https://tools.ietf.org/html/rfc6066 を参照してください)。名前が一致しない場合、アクセプターへの接続は拒否されます。この場合、WARN メッセージが記録されます。 受信接続に **server_name** 拡張子が含まれていない場合、接続は受け入れられます。 コネクターで使用されると、SSL 接続の **server_name** 拡張子に **sniHost** 値が使用されます。
sslProvider	両方	**JDK** と **OPENSSL** の間で SSL プロバイダーを変更するために使用されます。デフォルトは **JDK** です。 **OPENSSL** に設定した場合は、クラスパスに **netty-tcnative** を追加して、ネイティブにインストールした OpenSSL を使用できます。 このオプションは、OpenSSL でサポートされ、JDK プロバイダーでサポートされていない、特別な ciphersuite-elliptic 曲線の組み合わせを使用する場合に便利です。
trustStorePath	両方	アクセプターで使用される場合、これはサーバーが信頼するすべてのクライアントのキーを保持するサーバー側の SSL キーストアへのパスとなります。双方向 SSL(つまり相互認証) を使用している場合、アクセプターにのみ関係します。 コネクターで使用される場合は、クライアント側の SSL/TLS キーストアへのパスとなります。これは、クライアントが信頼するすべてのサーバーの公開鍵を保持します。この値はサーバー上で設定可能ですが、ダウンロードしてクライアントで使用します。クライアントでサーバーの設定と異なるパスを使用する必要がある場合は、カスタムの **javax.net.ssl.trustStore** システムプロパティーまたは ActiveMQ 固有の **org.apache.activemq.ssl.trustStore** システムプロパティーのいずれかを使用してサーバー側の設定をオーバーライドできます。ActiveMQ 固有のシステムプロパティーは、クライアントの別のコンポーネントがすでに標準の Java システムプロパティーを使用している場合に便利です。
useDefaultSslContext	コネクター	コネクターが (**SSLContext.getDefault()** を介して) 「default」SSL コンテキストを使用できるようにします。これは、(**SSLContext.setDefault(SSLContext)** を介して) クライアントによってプログラム的に設定できます。 このパラメーターが **true** に設定されている場合、**sslEnabled** 以外の SSL 関連のパラメーターはすべて無視されます。有効な値は **true** または **false** です。デフォルト値は **false** です。

パラメーター	用途	説明
verifyHost	両方	アクセプターで使用されると、接続しているクライアントの SSL 証明書の CN がホスト名と比較され、それらが一致することを確認します。これは双方向 SSL でのみ便利です。 コネクターで使用されると、サーバーの SSL 証明書の CN がホスト名と比較され、それらが一致することを確認します。これは一方向および双方向 SSL の両方に役立ちます。 有効な値は **true** または **false** です。デフォルト値は **false** です。
wantClientAuth	アクセプター	このアクセプターに接続するクライアントに対して、双方向 SSL が要求されているが、必須ではないことを伝えます。有効な値は **true** または **false** です。デフォルト値は **false** です。 **needClientAuth** を **true** に設定すると、そのプロパティーが優先され **wantClientAuth** は無視されます。

付録B アドレス設定設定要素

以下の表は、**address-setting** のすべての設定要素の一覧です。一部の要素は DEPRECATED とマークされていることに注意してください。潜在的な問題を回避するには、推奨される代替手段を使用してください。

表B.1 アドレス設定要素

名前	説明
address-full-policy	**max-size-bytes** で設定したアドレスが満杯になるとどうなるかを決定します。利用可能なポリシーは以下のとおりです。 **PAGE** : 完全なアドレスに送信されたメッセージは、ディスクにページングされます。 **DROP** : 完全なアドレスに送信されたメッセージは警告なしで破棄されます。 **FAIL** : 完全なアドレスに送信されたメッセージはドロップされ、メッセージプロデューサーは例外を受け取ります。 **BLOCK**: メッセージプロデューサーは、それ以上メッセージを送信しようとするとブロックします。  **注記** BLOCK ポリシーは、AMQP、OpenWire、および Core Protocol でのみ機能します。
auto-create-addresses	マッピング先のキューからキューの存在しないアドレスにメッセージを送信するか、消費しようとした場合にアドレスを自動作成するかどうか。デフォルト値は **true** です。
auto-create-dead-letter-resources	ブローカーが dead letter アドレスおよびキューを自動的に作成し、配信不能メッセージを受信するかどうかを指定します。デフォルト値は **false** です。 パラメーターが **true** に設定されている場合、ブローカーは dead letter アドレスと関連する dead letter キューを定義する **<address>** 要素を自動的に作成します。自動作成された **<address>** 要素の名前は、**<dead-letter-address>** に指定する name の値と一致します。
auto-create-jms-queues	非推奨: 代わりに **auto-create-queues** を使用してください。JMS プロデューサーまたはコンシューマーがこのようなキューを使用しようとするときに、このブローカーが、アドレス設定に対応する JMS キューを自動的に作成するかどうかを決定します。デフォルト値は **false** です。
auto-create-jms-topics	非推奨: 代わりに **auto-create-queues** を使用してください。JMS プロデューサーまたはコンシューマーがこのようなキューを使用しようとするときに、このブローカーが、アドレス設定に対応する JMS トピックを自動的に作成するかどうかを決定します。デフォルト値は **false** です。

名前	説明
auto-create-queues	クライアントがキューに対してメッセージを送信するとき、またはキューからメッセージを消費しようとするときにキューを自動的に作成するかどうか。デフォルト値は **true** です。
auto-delete-addresses	ブローカーにキューがなくなったときに自動作成されたアドレスを削除するかどうか。デフォルト値は **true** です。
auto-delete-jms-queues	非推奨: 代わりに **auto-delete-queues** を使用してください。自動作成された JMS キューにコンシューマーもメッセージもない場合は、AMQ Broker が自動的に削除するかどうかを決定します。デフォルト値は **false** です。
auto-delete-jms-topics	非推奨: 代わりに **auto-delete-queues** を使用してください。自動作成された JMS トピックにコンシューマーもメッセージもない場合は、AMQ Broker が自動的に削除するかどうかを決定します。デフォルト値は **false** です。
auto-delete-queues	キューにコンシューマーがなく、メッセージがない場合に自動作成されたキューを削除するかどうか。デフォルト値は **true** です。
config-delete-addresses	設定ファイルを再読み込みすると、この設定で、設定ファイルから削除されたアドレス (とそのキュー) を処理する方法を指定します。以下の値を指定できます。 **OFF (デフォルト)** 設定ファイルを再読み込みすると、アドレスは削除されません。 **FORCE** 設定ファイルが再ロードされると、アドレスとそのキューが削除されます。キューにメッセージがある場合は、それらも削除されます。
config-delete-queues	設定ファイルを再読み込みすると、この設定は、設定ファイルから削除されたキューを処理する方法を指定します。以下の値を指定できます。 **OFF (デフォルト)** 設定ファイルを再読み込みすると、キューは削除されません。 **FORCE** キューは、設定ファイルが再ロードされると削除されます。キューにメッセージがある場合は、それらも削除されます。
dead-letter-address	ブローカーがデッドメッセージを送信するアドレス。
dead-letter-queue-prefix	ブローカーにより、自動作成された dead letter キューの名前に適用される接頭辞。デフォルト値は **DLQ** です。
dead-letter-queue-suffix	ブローカーにより、自動作成された dead letter キューに適用される接尾辞。デフォルト値は定義されていません (つまり、ブローカーは接尾辞を適用しません)。

名前	説明
default-address-routing-type	自動作成されたアドレスで使用されるルーティングタイプ。デフォルト値は **MULTICAST** です。
default-max-consumers	このキューで一度に許可されるコンシューマーの最大数。デフォルト値は **200** です。
default-purge-on-no-consumers	コンシューマーがない場合にキューの内容をパージするかどうか。デフォルト値は **false** です。
default-queue-routing-type	自動作成されたキューで使用されるルーティングタイプ。デフォルト値は **MULTICAST** です。
enable-metrics	Prometheus プラグインなどの設定されたメトリクスプラグインが一致するアドレスまたはアドレスのセットのメトリクスを収集するかどうかを指定します。デフォルト値は **true** です。
expiry-address	期限切れのメッセージを受信するアドレス。
expiry-delay	デフォルトの有効期限を使用してメッセージに使用される有効期限 (ミリ秒単位) を定義します。デフォルト値は **-1** で、有効期限がないことを意味します。
last-value-queue	キューが最後の値のみを使用するかどうかを指定します。デフォルト値は **false** です。
management-browse-page-size	管理リソースが参照できるメッセージの数を指定します。デフォルト値は **200** です。
max-delivery-attempts	dead letter アドレスに送信する前にメッセージの配信を試行する回数。デフォルトは **10** です。
max-redelivery-delay	再配信遅延の最大値 (ミリ秒単位)。
max-size-bytes	このアドレスの最大メモリーサイズ (バイト単位)。**address-full-policy** が **PAGING**、**BLOCK**、または **FAIL** の場合、この値は "K"、"Mb"、および "GB" などのバイト表記で指定されます。デフォルト値は **-1** で、無限のバイトを示します。このパラメーターは、特定のアドレス領域によって消費されるメモリー量を制限してブローカーメモリーを保護するために使用されます。この設定は、現在ブローカーアドレス空間に保存されているクライアントによって送信されるバイトの合計量を表しません。これは、ブローカーのメモリー使用率の推定値です。この値は、ランタイムの状態と特定のワークロードによって異なります。アドレス空間ごとに使用できる最大メモリー容量を割り当てることが推奨されます。通常のワークロードでは、ブローカーはメモリーの未処理のメッセージのペイロードサイズの約 150% から 200% が必要になります。

名前	説明
max-size-bytes-reject-threshold	**address-full-policy** が **BLOCK** の場合に使用されます。ブローカーがメッセージを拒否する前にアドレスが到達できる最大サイズ (バイト単位)。AMQP プロトコルの場合のみ **max-size-bytes** と組み合わせて動作します。デフォルト値は **-1** で、制限なしを意味します。
message-counter-history-day-limit	このアドレスのメッセージカウンター履歴を保持する日数。デフォルト値は **0** です。
page-max-cache-size	ページングナビゲーション中に I/O を最適化するためにメモリー内に保持するページファイルの数。デフォルト値は **5** です。
page-size-bytes	ページングサイズ (バイト単位)。**K**、**Mb**、**GB** などのバイト表記にも対応します。デフォルト値は **10485760** バイト (約 10.5 MB) です。
redelivery-delay	キャンセルされたメッセージを再配信するまでの待機時間 (ミリ秒単位)。デフォルト値は **0** です。
redelivery-delay-multiplier	redelivery-delay パラメーターに適用する乗数。デフォルト値は **1.0** です。
redistribution-delay	キューの最後のコンシューマーが閉じられてから残りのメッセージを再分配するまでブローカーが待機する時間 (ミリ秒単位) を定義します。デフォルト値は **-1** です。
send-to-dla-on-no-route	**true** に設定すると、キューにルーティングされないメッセージは、設定済みの dead letter アドレスに送信されます。デフォルト値は **false** です。
slow-consumer-check-period	低速なコンシューマーについてチェックする頻度 (秒単位)。デフォルト値は **5** です。
slow-consumer-policy	低速なコンシューマーが特定されたときにどうするのかを決定します。有効なオプションは **KILL** または **NOTIFY** です。**KILL** はコンシューマーの接続を強制終了します。これは、同じ接続を使用するすべてのクライアントスレッドに影響を与えます。**NOTIFY** は **CONSUMER_SLOW** 管理通知をクライアントに送信します。デフォルト値は **NOTIFY** です。
slow-consumer-threshold	最小限許可されるメッセージ消費率。この値を下回るとコンシューマーは遅いと見なされます。1 秒あたりのメッセージで測定されます。デフォルト値は **-1** で、バインドされません。

付録C クラスター接続設定要素

以下の表は、**cluster-connection** のすべての設定要素の一覧です。

表C.1 クラスター接続設定要素

名前	説明
address	各クラスター接続は、**address** フィールドで指定された値と一致するアドレスにのみ適用されます。アドレスが指定されていない場合、すべてのアドレスは負荷分散されます。 **address** フィールドは、アドレスのコンマ区切りリストもサポートします。アドレスが一致しないようにするには、除外構文 (**!**) を使用します。以下は、アドレスの例です。 **jms.eu** **jms.eu** で始まるすべてのアドレスと一致させます。 **!jms.eu** **jms.eu** で始まるアドレス以外のすべてのアドレスに一致させます。 **jms.eu.uk,jms.eu.de** **jms.eu.uk** または **jms.eu.de** で始まるすべてのアドレスと一致させます。 **jms.eu,!jms.eu.uk** **jms.eu** で始まるすべてのアドレスと一致させ、**jms.eu.uk** で始まるアドレスには一致させません。  **注記** 重複アドレス (例 : "europe" および "europe.news") を持つ複数のクラスター接続を持つべきではありません。複数のクラスター接続間で同じメッセージが分散され、配信が重複してしまう可能性があるためです。
call-failover-timeout	フェイルオーバーの試行中に呼び出しが行われる場合に使用します。デフォルトは **-1** で、タイムアウトなしです。
call-timeout	パケットがクラスター接続上で送信され、ブロッキング呼び出しである場合、**call-timeout** は例外を出力する前にブローカーが応答を待つ時間 (ミリ秒単位) を決定します。デフォルトは **30000** です。
check-period	クラスター接続が別のブローカーから ping を受信しなかったかどうかを確認する間隔 (ミリ秒単位)。デフォルトは **30000** です。
confirmation-window-size	接続先のブローカーから確認の送信に使用されるウインドウのサイズ (バイト単位)。ブローカーが **confirmation-window-size** バイトを受信すると、クライアントに通知します。デフォルトは **1048576** です。**-1** の値は、ウィンドウがないことを意味します。
connector-ref	適切なクラスタートポロジーを持つようにクラスター内の他のブローカーに送信される **コネクター** を特定します。このパラメーターは必須です。

名前	説明
connection-ttl	クラスター内の特定のブローカーからメッセージを受信しなくなった場合に、クラスター接続が有効である期間を決定します。既定値は **60000** です。
discovery-group-ref	クラスター内の他のブローカーとの通信に使用される **discovery-group** を参照します。この要素には **discovery-group-name** 属性を含める必要があります。これは、以前に設定した **discovery-group** の **name** 属性と一致する必要があります。
initial-connect-attempts	クラスターでシステムがブローカーへの接続を試行する回数を設定します。max-retry が達成されると、このブローカーは永続的にダウンしているとみなされ、システムはメッセージをこのブローカーにルーティングしません。デフォルトは **-1** で、再試行が無限を意味します。
max-hops	チェーンの中間として他のブローカーにのみ接続される可能性のあるブローカーへのメッセージの負荷分散を行うようにブローカーを設定します。これにより、メッセージ負荷分散を提供しながらより複雑なトポロジーが可能になります。デフォルト値は **1** です。つまり、メッセージはこのブローカーに直接接続された他のブローカーにのみ分散されます。このパラメーターは任意です。
max-retry-interval	再試行の最大遅延 (ミリ秒単位)。既定値は **2000** です。
message-load-balancing	クラスターの他のブローカー間でメッセージを分散するかどうか、およびその方法を決定します。負荷分散を有効にするために **message-load-balancing** 要素を含めます。デフォルト値は **ON_DEMAND** です。値を指定することもできます。有効な値は以下のとおりです。 **OFF** 負荷分散を無効にします。 **STRICT** キューにアクティブなコンシューマーまたは一致するセレクターがあるかどうかに関係なく、一致するキューを持つすべてのブローカーにメッセージを転送します。 **ON_DEMAND** メッセージがアクティブなコンシューマーまたは一致するセレクターを持つブローカーにのみ転送されるようにします。
min-large-message-size	メッセージのサイズ (バイト単位) が **min-large-message-size** を超える場合には、ネットワーク上で他のクラスターメンバーへの送信時に複数のセグメントに分割されます。デフォルトは **102400** です。
notification-attempts	クラスターへの接続時にクラスター接続がそれ自体をブロードキャストする回数を設定します。デフォルトは **2** です。
notification-interval	クラスターへの接続時にクラスター接続がそれ自体をブロードキャストする頻度 (ミリ秒単位) を設定します。既定値は **1000** です。

名前	説明
producer-window-size	クラスター接続に対するプロデューサーフロー制御のサイズ (バイト単位)。デフォルトでは無効になっていますが、クラスターで実際に大きなメッセージを使用している場合は、値を設定します。**-1** の値は、ウィンドウがないことを意味します。
reconnect-attempts	システムがクラスターのブローカーへの再接続を試行する回数を設定します。max-retry が達成されると、このブローカーは永続的にダウンしていると見なされ、システムはこのブローカーへのメッセージのルーティングを停止します。デフォルトは **-1** で、再試行が無限を意味します。
retry-interval	再試行の間隔 (ミリ秒単位) を指定します。クラスター接続が作成され、ターゲットブローカーが起動または起動されていない場合、他のブローカーからのクラスター接続は、バックアップされるまでターゲットへの接続を再試行します。このパラメーターは任意です。デフォルト値は **500** ミリ秒です。
retry-interval-multiplier	それぞれの再接続試行後に **retry-interval** を増やすために使用される乗数。デフォルトでは 1 回です。
use-duplicate-detection	クラスター接続はブリッジを使用してブローカーをリンクし、ブリッジを設定して転送される各メッセージに複製 ID プロパティーを追加できます。ブリッジのターゲットブローカーがクラッシュし、復旧すると、メッセージがソースブローカーから再送信される可能性があります。**use-duplicate-detection** を **true** に設定すると、重複メッセージがフィルターされ、ターゲットブローカーで受信時に無視されます。デフォルトは **true** です。

付録D コマンドラインツール

AMQ Broker にはコマンドラインインターフェイス (CLI) ツールのセットが含まれるため、メッセージングジャーナルを管理できます。以下の表は、各ツールの名前と説明を一覧表示しています。

ツール	説明
exp	特別な XML 形式および独立した XML 形式を使用して、メッセージデータをエクスポートします。
imp	**exp** によって提供された出力を使用して、ジャーナルを稼働中のブローカーにインポートします。
data	ジャーナルレコードとデータの圧縮に関するレポートを出力します。
encode	String にエンコードされるジャーナルの内部形式を示しています。
decode	エンコードから内部ジャーナル形式をインポートします。

各ツールで利用可能なコマンドの全一覧については、**help** パラメーターの後にツール名を使用してください。たとえば、以下の例で CLI 出力には、ユーザーが **./artemis help data** コマンドを入力すると、 **data** ツールで利用可能なコマンドがすべて表示されます。

```
$ ./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)

        ...
```

各ツールのコマンドを実行する方法の詳細については、ツールでヘルプを使用できます。たとえば、CLI は、ユーザーが **./artemis help data print** の入力後に **data print** コマンドに関する詳細情報を一覧表示します。

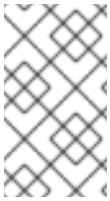
```
$ ./artemis help data print

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)
```

付録E メッセージングジャーナル設定要素

以下の表は、AMQ Broker メッセージングジャーナルに関連する設定要素の一覧です。

表E.1 アドレス設定要素

名前	説明
journal-directory	メッセージジャーナルが置かれているディレクトリー。デフォルト値は **BROKER_INSTANCE_DIR/data/journal** です。 最適なパフォーマンスを得るには、ディスクヘッドの移動を最小限に抑えるために、ジャーナルを独自の物理ボリュームに置く必要があります。ジャーナルが、バインディングジャーナル、データベース、トランザクションコーディネーターなど、他のファイルを書き込む可能性のある他のプロセスと共有するボリュームにあるとします。その場合、書き込み時にディスクヘッドがこれらのファイル間で素早く移動するため、パフォーマンスが大幅に低下する可能性があります。 SAN を使用する場合、各ジャーナルインスタンスに独自の LUN (論理ユニット) を指定する必要があります。
create-journal-dir	**true** に設定すると、ジャーナルディレクトリーがまだ存在しない場合は、**journal-directory** で指定された場所にジャーナルディレクトリーが自動的に作成されます。デフォルト値は **true** です。
journal-type	有効な値は **NIO** または **ASYNCIO** です。 **NIO** に設定すると、ブローカーは Java NIO インターフェイスをそのジャーナルに使用します。**ASYNCIO** に設定され、ブローカーは Linux 非同期 IO ジャーナルを使用します。**ASYNCIO** を選択していても Linux を実行していない場合や、libaio がインストールされていない場合、ブローカーはこれを検出し、**NIO** の使用に自動的にフォールバックします。
journal-sync-transactional	**true** に設定すると、ブローカーはすべてのトランザクションデータをトランザクション境界のディスクにフラッシュします (つまり、コミット、準備、およびロールバック)。デフォルト値は **true** です。
journal-sync-non-transactional	**true** に設定すると、ブローカーは非トランザクションメッセージデータ (送信および確認応答) を毎回ディスクにフラッシュします。デフォルト値は **true** です。
journal-file-size	各ジャーナルファイルのサイズ (バイト単位)。デフォルト値は **10485760** バイト (10MiB) です。
journal-min-files	起動時にブローカーが事前作成するファイルの最小数。既存のメッセージデータがない場合にのみ、ファイルが事前に作成されます。 定常状態でキューに格納する予定のデータ量に応じて、データ全体の量と一致するようにこのファイルの数を調整する必要があります。

名前	説明
journal-pool-files	システムは必要な数だけファイルを作成します。ただし、ファイルを回収すると、**journal-pool-files** に縮小されます。 デフォルト値は **-1** です。つまり、作成後はジャーナルのファイルは削除されません。ただし、システムは無限に拡張することができませんが、永久に拡張できる宛先のページングを使用する必要があるため、システムを拡張する必要があります。
journal-max-io	常時 IO キューに格納できる書き込み要求の最大数を制御します。キューが満杯になると、領域が解放されるまで書き込みがブロックされます。 NIO を使用する場合、この値は常に **1** である必要があります。AIO を使用する場合、デフォルト値は **500** です。最大 AIO の合計は、OS レベルに設定された値 (**/proc/sys/fs/aio-max-nr**) より大きくすることはできません。通常、65536 になります。
journal-buffer-timeout	バッファーがフラッシュされるタイミングのタイムアウトを制御します。AIO は通常 NIO よりもフラッシュ率が高いため、システムは NIO と AIO の両方で異なるデフォルト値を維持します。 NIO のデフォルト値は **3333333** ナノ秒 (300 回 / 秒) で、AIO のデフォルト値は **50000** ナノ秒 (2000 回 / 秒) です。  **注記** タイムアウト値を長くすると、レイテンシーを代償にシステムのスループットを増やすことができる可能性があり、デフォルト値はスループットと待ち時間のバランスをうまくとるように選択されています。
journal-buffer-size	AIO でのタイムアウトバッファーのサイズ。デフォルト値は **490KiB** です。
journal-compact-min-files	ブローカーがジャーナルを圧縮する前に必要なファイルの最小数。少なくとも **journal-compact-min-files** が設定されない限り、圧縮アルゴリズムは開始されません。デフォルト値は **10** です。  **注記** 値を **0** に設定すると、ジャーナルが無限に増大するため、圧縮は無効になり危険になる可能性があります。
journal-compact-percentage	圧縮を開始するしきい値。**journal-compact-percentage** 未満がライブデータであると判断されると、ジャーナルデータが圧縮されます。また、少なくとも **journal-compact-min-files** のデータファイルがジャーナルに存在しなければ、圧縮は起動されません。デフォルト値は **30** です。

付録F レプリケーション高可用性設定要素

以下の表は、レプリケーション HA ポリシーを使用する場合の有効な **ha-policy** 設定要素の一覧です。

表F.1 レプリケーションの高可用性を使用する場合に利用可能な設定要素

名前	説明
check-for-live-server	マスターブローカーとして設定されたブローカーにのみ適用されます。元のマスターブローカーが起動時に独自のサーバー ID を使用して別のライブブローカーのクラスターをチェックするかどうかを指定します。**true** に設定すると、元のマスターブローカーに戻り、2 つのブローカーが同時に存続するスプリットブレインの状況を回避します。このプロパティーのデフォルト値は **false** です。
cluster-name	レプリケーションに使用するクラスター設定の名前。この設定は、複数のクラスター接続を設定する場合にのみ必要です。設定された場合、クラスターへの接続時にこの名前のクラスター設定が使用されます。未設定の場合は、設定に定義された最初のクラスター接続が使用されます。
group-name	これが設定されている場合、バックアップブローカーは **group-name** の一致する値を持つライブブローカーとのみペアになります。
initial-replication-sync-timeout	レプリカの最初のレプリケーションプロセスが完了すると、レプリカが必要なデータをすべて受け取ったことを確認するため、レプリケートブローカーが待機する時間。このプロパティーのデフォルト値は **30,000** ミリ秒、つまり 30 秒です。  **注記** この間隔の間、その他のジャーナル関連の操作はブロックされます。
max-saved-replicated-journals-size	バックアップブローカーにのみ適用されます。バックアップブローカーが保持するバックアップジャーナルファイルの数を指定します。この値に達すると、ブローカーは最も古いジャーナルファイルを削除して、新しいバックアップジャーナルファイルごとに領域を作成します。このプロパティーのデフォルト値は **2** です。
allow-failback	バックアップブローカーにのみ適用されます。ライブブローカーなどの別のブローカーが原因で、バックアップブローカーが元のロールを再開するかどうかを決定します。このプロパティーのデフォルト値は **true** です。
restart-backup	バックアップブローカーにのみ適用されます。別のブローカーにフェイルバックした後にバックアップブローカーが自動的に再起動するかどうかを決定します。このプロパティーのデフォルト値は **true** です。

改訂日時：2023-01-28 12:29:06 +1000