Red Hat JBoss Enterprise Application Platform 7.4-Beta

メッセージングの設定

Red Hat JBoss Enterprise Application Platform のメッセージングアプリケーションの開発とデプロイを行う開発者および管理者のための手順と情報。

Last Updated: 2021-05-17

Red Hat JBoss Enterprise Application Platform 7.4-Beta メッセージングの設定

Red Hat JBoss Enterprise Application Platform のメッセージングアプリケーションの開発とデプロイを行う開発者および管理者のための手順と情報。

法律上の通知

概要

本書は、Red Hat JBoss Enterprise Application Platform でメッセージングアプリケーションの開発とデプロイを行う開発者と管理者に情報を提供します。

目次

パート I. メッセージングと JBOSS EAP 7

JBoss EAP 6 のメッセージングブローカーは、HornetQ と呼ばれる JBoss コミュニティープロジェクトでした。HornetQ コードベースは Apache ActiveMQ プロジェクトに提供され、HornetQ コミュニティーは、提供後のコードベースを強化し、次世代のメッセージングブローカーを作成するために、そのプロジェクトに参加しました。その結果が Apache ActiveMQ Artemis です。この JBoss EAP 7 のメッセージングブローカーは、JBoss EAP 6 とのメッセージング統合および後方互換性を提供します。ActiveMQ Artemis は、JBoss EAP 6 の HornetQ ブローカーとのプロトコル互換性を維持する一方で、スマートな新機能もいくつか備えています。本ガイドでは、JBoss EAP 7.4 で利用可能な ActiveMQ Artemis ブローカーの多くの機能について考察し、役に立つサンプルを提供します。

第1章 メッセージングの概念

1.1. メッセージングシステム

メッセージングシステムにより、異種システムを疎結合して、信頼性も高めることができます。Remote Procedure Call (RPC) パターンに基づいたシステムとは異なり、メッセージングシステムは、リクエストと応答の間に密接な関係がないパターンを渡す非同期メッセージを主に使用します。ほとんどのメッセージングシステムは、必要に応じてリクエスト応答モードもサポートできる柔軟性がありますが、これはメッセージングシステムの主要な機能ではありません。

メッセージングシステムは、そのコンシューマーからメッセージの送信者を切り離します。実際、メッセージの送信者とコンシューマーは完全に独立していて、相互に何も知らないため、柔軟な疎結合のシステムを作成できます。大企業は多くの場合、メッセージングシステムを使用して、異種システムを疎結合するメッセージバスを実装します。メッセージバスは、Enterprise Service Bus (ESB) の中核を形成することができます。メッセージバスを使用して共通点のないシステムを切り離すことで、システムを拡大し、より簡単に適応させることができます。また、不安定な相互依存性がないため、より柔軟に、新しいシステムを追加したり、古いシステムを廃止したできます。

メッセージングシステムには、信頼できるメッセージングを保証する配信保証、複数のメッセージの送信または消費を単一の作業単位として集約するトランザクション、メッセージがサーバーの障害や再起動に耐えられるようにする耐久性などの概念を組み込むこともできます。

1.2. メッセージングスタイル

ほとんどのメッセージングシステムがサポートするメッセージングスタイルには、**ポイントツーポイント**パターンと**パブリッシュ/サブスクライブ**パターンの 2 種類があります。

- ポイントツーポイントパターン
  ポイントツーポイントパターンでは、キューでリッスンする単一のコンシューマーにメッセージを送信します。キューに入ると、通常、メッセージは永続化され、配信が保証されます。キューを通り抜けたメッセージは、メッセージングシステムによってコンシューマーに配信されます。処理されると、コンシューマーはメッセージの配信を認識します。同じメッセージの同じキューでリッスンしているコンシューマーは複数ある可能性がありますが、それぞれのメッセージを受け取るコンシューマーは 1 つのみです。

- パブリッシュ/サブスクライブパターン
  パブリッシュ/サブスクライブパターンにより、送信者は単一の宛先を使用して複数のコンシューマーにメッセージを送信できます。この宛先は通常、**トピック**として知られています。各トピックには、コンシューマー (サブスクライバー) を複数持つことができます。ポイントツーポイントメッセージングと異なり、すべてのサブスクライバーはトピックに発行されるすべてのメッセージを受け取ります。

  もう 1 つの興味深い違いは、サブスクライバーが永続できることです。永続的なサブスクリプションは、接続時にサーバーに一意の識別子を渡します。これにより、サーバーは、サブスクライバーが最後に接続した以降にトピックに発行されるすべてのメッセージを識別して送信できます。通常、このようなメッセージは、再起動後にもサーバーによって保持されます。

1.3. JAVA MESSAGING SERVICE (JMS)

Java Messaging Service (JMS) 2.0 は JSR 343 で定義されています。Jakarta EE と同等の仕様は Jakarta Messaging です。JMS は、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メッセージングスタイルを提供する Java API です。JMS にはトランザクションの使用も組み込まれて

います。JMS は標準の回線形式を定義しないため、JMS プロバイダーのベンダーはすべて標準の API を使用しながら、クライアントとサーバー間の通信にさまざまな内部ネットワークプロトコルを使用できます。

1.4. JMS 宛先

JMS 宛先は、JMS 接続ファクトリーとともに、JMS 管理オブジェクトです。宛先は、JMS クライアントによってメッセージの生成と消費の両方に使用されます。JMS クライアントは宛先を使用して、メッセージの生成時にターゲットを、メッセージの消費時にメッセージのソースを指定できます。パブリッシュ/サブスクライブパターンを使用する場合、宛先はトピックと呼ばれます。ポイントツーポイントパターンを使用する場合、宛先はキューと呼ばれます。

アプリケーションは、サーバー側で設定され、通常 JNDI を介してアクセスされる多くの異なる JMS 宛先を使用できます。

第2章 統合 ACTIVEMQ ARTEMIS メッセージングブローカー

2.1. ACTIVEMQ ARTEMIS

Apache ActiveMQ Artemis は非同期メッセージングシステムのオープンソースプロジェクトです。高性能で、埋め込み可能であり、クラスター化されていて、複数のプロトコルをサポートします。JBoss EAP 7 は Apache ActiveMQ Artemis を JMS ブローカーとして使用し、**messaging-activemq** サブシステムを使用して設定されます。これは HornetQ ブローカーを完全に置き換えますが、JBoss EAP 6 とのプロトコル互換性を維持します。

コアの ActiveMQ Artemis は JMS に依存せず、**コア API** と呼ばれる非 JMS API を提供します。ActiveMQ Artemis は、ファサードレイヤーを使用してコア API 上で JMS セマンティクスを実装する JMS クライアント API も提供します。基本的に、JMS のやり取りは、JMS クライアント API を使用してクライアント側のコア API 操作に変換されます。そこから、コアクライアント API および Apache ActiveMQ Artemis の回線形式を使用してすべての操作を送信します。サーバー自体はコア API のみを使用します。コア API とその概念の詳細は、ActiveMQ Artemis のドキュメントを参照してください。

2.2. APACHE ACTIVEMQ ARTEMIS コア API と JMS 宛先

JMS 宛先が Apache ActiveMQ Artemis アドレスにマップされる方法を簡単に説明します。

Apache ActiveMQ Artemis コアは JMS に依存しません。JMS トピックの概念はありません。JMS トピックは、0 個以上のキューがバインドされたアドレス (トピック名) としてコアに実装されます。このアドレスにバインドされる各キューが、トピックサブスクリプションを表します。同様に、JMS キューは、JMS キューを表す単一のキューがバインドされるアドレス (JMS キュー名) として実装されます。

通常、すべての JMS キューは、コアキュー名の先頭に文字列 **jms.queue** が付加されたコアキューにマップさ れます。たとえば、**orders.europe** の名前を持つ JMS キューが、**jms.queue.orders.europe** の名前でコアキューにマップされます。コアキューがバインドされるアドレスもコアキュー名で指定されます。

JMS トピックの場合、サブスクリプションを表すキューがバインドされるアドレスは、文字列 **jms.topic.** を JMS トピックの名前の先頭に付加して指定されます。たとえば、**news.europe** という名前の JMS トピックは、コアアドレス **jms.topic.news.europe** にマップされます。

つまり、**orders.europe** という名前の JMS キューへの JMS メッセージを送信する場合は、サーバーを介してアドレス **jms.queue.orders.europe** にバインドされるコアキューにルーティングされます。**news.europe** という名前の JMS トピックへの JMS メッセージを送信する場合は、サーバーを介してアドレス **jms.topic.news.europe** にバインドされるコアキューにルーティングされます。

名前が **orders.europe** の JMS キューの設定を行うには、対応するコアキュー **jms.queue.orders.europe** を設定する必要があります。

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

パート II. 単一ノードのメッセージングシステムの設定

パート II ではまず、**helloworld-mdb** クイックスタートを使用して JBoss EAP 7 メッセージングを開始するガイドを説明します。次に、セキュリティーや永続性などのトピックを含め、いずれのインストールにも利用可能な設定オプションについて説明します。クラスタリング、高可用性、別のサーバーへの接続などのトピックを始め、JBoss EAP 7 の複数のインストールに関連する設定については、パート III の「複数ノードのメッセージングシステムの設定」を参照してください。

第3章 はじめに

3.1. HELLOWORLD-MDB クイックスタートの使用

helloworld-mdb クイックスタートでは単純なメッセージ駆動 Bean を使用して、基本的な Jakarta EE メッセージング機能を示します。JBoss EAPメッセージングサーバーに用意されている機能を知るには、クイックスタートを起動して基本設定を確認しながら実行する方法が非常に優れています。

helloworld-mdb クイックスタートのビルドとデプロイ

helloworld-mdb クイックスタートをビルドし、デプロイする方法については、クイックスタートで提供される **README.md** ファイルの手順を参照してください。**messaging-activemq** サブシステムを含む **full** 設定を指定する JBoss EAP サーバーを開始する必要があります。異なる設定ファイルを使用して JBoss EAP を開始する方法は、**README.md** ファイルまたは JBoss EAP **『設定ガイド』** を参照してください。

3.2. メッセージングサブシステム設定の概要

messaging-activemq サブシステムのデフォルト設定は、JBoss EAP サーバーを **full** または **full-ha** 設定で開始するときに含まれます。**full-ha** オプションには、クラスタリングや高可用性などの機能の高度な設定が含まれます。

必須ではありませんが、**helloworld-mdb** クイックスタートを作業例として使用して、この設定の概要と一緒に実行することが推奨されます。

messaging-activemq サブシステムで使用できるすべての設定については、**EAP_HOME/docs/schema/** ディレクトリーにあるスキーマ定義を参照するか、以下のように、管理 CLI からサブシステムで **read-resource-description** 操作を実行します。

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

サーバー設定ファイルの以下のエクステンションは、JBoss EAP に対して、そのランタイムの一部として **messaging-activemq** サブシステムを含めるように指示するものです。

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

messaging-activemq サブシステムの設定は **<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">** 要素内に含まれます。

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

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

接続ファクトリー
メッセージングクライアントは JMS **ConnectionFactory** オブジェクトを使用してサーバーに接続します。デフォルトの JBoss EAP 設定は複数の接続ファクトリーを定義します。in-vm、http、および pooled 接続の **<connection-factory>** があることに留意します。

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

詳細は、「接続ファクトリーの設定」のセクションを参照してください。

コネクターとアクセプター
各 JMS 接続ファクトリーはコネクターを使用して、クライアントプロデューサーまたはコンシューマーからメッセージングサーバーへの JMS 対応通信を有効にします。コネクターオブジェクトは、メッセージングサーバーに接続するために使用されるトランスポートとパラメーターを定義します。それに対応するのが、メッセージングサーバーが受け入れる接続タイプを識別するアクセプターオブジェクトです。

デフォルトの JBoss EAP 設定は複数のコネクターとアクセプターを定義します。

例: デフォルトのコネクター

```
<http-connector name="http-connector" socket-binding="http" endpoint="http-acceptor"/>
```

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

例: デフォルトのアクセプター

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

詳細については、「アクセプターとコネクター」のセクションを参照してください。

ソケットバインディンググループ
デフォルトのコネクターの **socket-binding** 属性は、**http** という名前のソケットバインディングを参照します。JBoss EAP は標準の Web ポートで受信リクエストを多重化できるため、http コネクターが使用されます。

この**socket-binding** は、設定ファイルの他の場所の **<socket-binding-group>** セクションの一部として見つけることができます。http および https ソケットバインディングの設定が、**<socket-binding-groups>** 要素内でどのように表示されているかに注目してください。

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

ソケットバインディングに関する詳細は、JBoss EAP **『設定ガイド』** の「**ソケット バインディングの設定** 」を参照してください。

メッセージングセキュリティー
messaging-activemq サブシステムには、JBoss EAP の初回インストール時に単一の **security-setting** 要素が含まれます。

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

これは、ワイルドカード **#** で示され、ロール **guest** が指定されたすべてのユーザーが、サーバー上の任意のアドレスにアクセスできることを宣言します。ワイルドカード構文の詳細は、「アドレス設定の設定」を参照してください。

接続先およびリモート接続のセキュリティー保護の詳細は、「メッセージングセキュリティーの設定」を参照してください。

メッセージング宛先
full および **full-ha** 設定には、JBoss EAP が期限切れのメッセージまたは適切な宛先にルーティングできないメッセージを保持するのに使用できる 2 つの有用なキューが含まれます。

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

以下のメソッドのいずれかを使用して、JBoss EAP に独自のメッセージング宛先を追加できます。

- 管理 CLI の使用
  以下の管理 CLI コマンドを使用して、キューを追加します。

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

  以下の管理 CLI コマンドを使用して、トピックを追加します。

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

- 管理コンソールの使用
  メッセージング宛先は管理コンソールから設定できます。**設定 → サブシステム → Messaging (ActiveMQ) → サーバー**の順に移動し、サーバーを選択して**宛先**を選択し、**表示**をクリックします。**JMS キュー**タブを選択し てキューを設定し、**JMS トピック**を選択してトピック を設定します。

- Java EE デプロイメント記述子またはアノテーションを使用した宛先の定義。
  Jakarta EE 8 では、デプロイメント記述子にキューとトピックの設定を含めることができます。以下は、JMS キューを定義する Java EE 記述子ファイルのスニペットです。

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

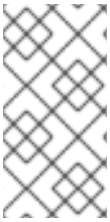
  たとえば、**helloworld-mdb** クイックスタートのメッセージ駆動 Bean には、アプリケーションの実行に必要なキューとトピックを定義するアノテーションが含まれます。この方法で作成された宛先は、ランタイムキューのリストに表示されます。管理 CLI を使用してランタイムキューのリストを表示します。クイックスタートをデプロイすると、作成されたランタイムキューは以下のように表示されます。

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

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

詳細は、「メッセージング宛先の設定」を参照してください。

第4章 メッセージング宛先の設定



注記

メッセージング宛先を設定するには、JBoss EAP でメッセージングを有効にする必要があります。この機能は、**standalone-full.xml** または **standalone-full-ha.xml** 設定ファイルで実行する際にデフォルトで有効になります。**domain.xml** 設定ファイルでは、メッセージングも有効になっています。

4.1. キューの追加

JMS キューを追加するには、管理 CLI から次の **jms-queue** コマンドを使用します。

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

entries 属性は、複数の JNDI 名が単一スペースで区切られたリストであることに注意してください。JNDI 名のリストが角括弧 **[]** を使用して囲んであることにも留意します。**queue-address** はルーティング設定を提供し、**entries** はクライアントがキューを検索するために使用できる JNDI 名のリストを提供します。

キューの属性の読み取り

管理 CLI で **jms-queue** コマンドを使用してキュー の設定を読み取ることができます。

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

または、管理 CLI を使用して **messaging-activemq** サブシステムにアクセスすることで、キューの設定を読み取ることもできます。

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

jms-queue の属性

管理 CLI は、以下のコマンドを実行すると、**jms-queue** 設定要素のすべての属性を表示します。

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

以下の表は、**jms-queue** のすべての属性を示します。

属性	説明
consumer-count	このキューからメッセージを消費するコンシューマーの数。ランタイム時に利用可能です。

属性	説明
dead-letter-address	デッドメッセージの送信先アドレス。詳細は、「デッドレターアドレスの設定」を参照してください。
deliverying-count	このキューが現在コンシューマーに配信しているメッセージの数。ランタイム時に利用可能です。
durable	キューが永続化されているかどうか。永続的なサブスクリプションの詳細については、「メッセージングスタイル」を参照してください。
entries	キューがバインドされる JNDI 名のリスト。必須。
expiry-address	期限切れのメッセージを受信するアドレス。詳細は、「メッセージ有効期限の 設定」を参照してください。
legacy-entries	キューがバインドされる JNDI 名。
message-count	このキューに現在あるメッセージの数。ランタイム時に利用可能です。
message-added	作成以降、このキューに追加されたメッセージの数。ランタイム時に利用可能です。
paused	キューが一時停止されているかどうか。ランタイム時に利用可能です。
queue-address	ルーティングメッセージに使用されるアドレスを定義するキューアドレス。アドレス設定 の詳細は、「アドレス設定の設定」を参照してください。必須。
scheduled-count	このキューにスケジュールされているメッセージの数。ランタイム時に利用可能です。
selector	キューセレクター。セレクターの詳細は、「フィルター式とメッセージセレクター」 を参照してください。
temporary	キューが一時的であるかどうか。詳細は、「一時キューとランタイム キュー」を参照してください。

4.2. トピックの追加

トピックの追加や読み取りは、キューの追加によく似ています。

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

トピック属性の読み取り

トピック属性の読み取りの構文も、キューに使用される構文に似ています。

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

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

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

jms-topic の属性

管理 CLI は、以下のコマンドを実行すると、**jms-topic** 設定要素のすべての属性を表示します。

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

以下の表は、**jms-topic** の属性を示します。

属性	説明
delivering-count	このキューが現在コンシューマーに配信しているメッセージの数。ランタイム時に利用可能です。
durable-message-count	このトピックのすべての永続的サブスクライバーに対するメッセージの数。ランタイム時に利用可能です。
durable-subscription-count	このトピックの永続的サブスクライバーの数。ランタイム時に利用可能です。
entries	トピックがバインドされる JNDI 名。必須。
legacy-entries	トピックがバインドされるレガシー JNDI 名。
message-count	このキューに現在あるメッセージの数。ランタイム時に利用可能です。
message-added	作成以降、このキューに追加されたメッセージの数。ランタイム時に利用可能です。
non-durable-message-count	このトピックのすべての非永続的サブスクライバーに対するメッセージの数。ランタイム時に利用可能です。
non-durable-subscription-count	このトピックの非永続的サブスクライバーの数。ランタイム時に利用可能です。
subscription-count	このトピックの (永続的および非永続的) サブスクライバーの数。ランタイム時に利用可能です。

属性	説明
temporary	トピックが一時的であるかどうか。
topic-address	このトピックが参照するアドレス。必須。

4.3. JNDI エントリーとクライアント

リモートクライアントが検索できるようにするため、キューまたはトピックは **java:jboss/exported** 名前空間にバインドする必要があります。クライアントは、ルックアップを行うときに **java:jboss/exported/** の後にテキストを使用する必要があります。たとえば、**testQueue** という名前のキューは、その **entries** にリスト **jms/queue/test java:jboss/exported/jms/queue/test** が含まれます。**testQueue** にメッセージを送信するリモートクライアントは、文字列 **jms/queue/test** を使用してキューを検索します。一方、ローカルクライアントは **java:jboss/exported/jms/queue/test**、**java:jms/queue/test**、またはさらに単純な **jms/queue/test** を使用して検索できます。

管理 CLI のヘルプ

--help --commands フラグを使用すると、**jms-queue** コマンドと **jms-topic** コマンドに関する詳細情報を確認できます。

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

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

第5章 ロギングの設定

org.apache. activemq の JBoss EAP **logging** サブシステムにログカテゴリーを追加し、希望のログレベルを設定して、**messaging-activemq** サブシステムのロギングを設定できます。カテゴリーのログハンドラーを設定して、ログメッセージの記録方法を設定することもできます。

XA トランザクションに関するログの詳細を見るには、**com.arjuna** カテゴリーのログレベルを、**TRACE** や **DEBUG** などのより詳細な設定に変更します。

カテゴリーや他のオプションの設定など、ロギングの詳細は、JBoss EAP『設定 **ガイド**』 の ロギング のセクションを参照してください。

表5.1 ロギングカテゴリー

ログの対象	使用するカテゴリー
XA トランザクション	com.arjuna
すべてのメッセージングアクティビティー	org.apache.activemq
メッセージングジャーナル呼び出しのみ	org.apache.activemq.artemis.journal
JMS 呼び出しのみ	org.apache.activemq.artemis.jms
メッセージングユーティリティ呼び出しのみ	org.apache.activemq.artemis.utils
メッセージングコアサーバーのみ	org.apache.activemq.artemis.core.server

ロギング用のクライアントの設定

以下の手順を実施してメッセージングクライアントを設定します。

1. JBoss JMS クライアントおよびログマネージャーに依存関係をダウンロードします。Maven を使用している場合は、以下の依存関係を **pom.xml** ファイルに追加します。

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

詳細は、JBoss EAP **『開発ガイド』** の「 JBoss EAP で Maven を使用 」を参照してください。

2. ロガーのプロパティーファイルを作成します。**logging.properties** という名前を設定して、既知の場所に保存します。以下は、プロパティーファイルの例です。クライアント側の ロギング オプションの設定に関する詳細は、JBoss EAP **『開発ガイド』** のロギングのセクションを参照してください。

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

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

# Root logger handlers
logger.handlers=FILE

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

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

3. 想定パラメーターを使用してクライアントを起動します。**java** コマンドを使用してクライアントコードを起動するには、以下のパラメーターを追加します。

   a. JBoss クライアントおよびロガー JAR をクラスパスに追加します。

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

   b. JBoss ロギングマネージャーを有効にします。

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

   c. ロギングプロパティーファイルの場所を設定します。

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

   クライアントを起動する full コマンドは以下の例のようになります。

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

第6章 アドレス設定

messaging-activemq サブシステムには、メッセージを配信する方法とタイミング、試行すべき回数、メッセージの有効期限を制御するいくつかの設定可能なオプションがあります。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、単一の **<address-setting>** を複数の宛先に適用するよう JBoss EAP を設定できます。

6.1. ワイルドカード構文

ワイルドカードを使用すると、多数のシステムがアスタリスク記号 * を使用して単一のクエリーで複数のファイルまたは文字列を照合するのと同様に、単一のステートメントで類似したアドレスを照合できます。以下の表は、**<address-setting>** の定義に使用できる特殊文字のリストです。

表6.1 JMS ワイルドカード構文

文字	説明
. (シングルピリオド)	ワイルドカード式で単語と単語の間を示す。
# (ポンドまたはハッシュ記号)	0 個以上の連続する単語に相当する。
* (アスタリスク)	1つの単語に相当する。

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6.2 JMS ワイルドカードの例

例	説明
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。

6.2. デフォルトのアドレス設定

初期状態では、JBoss EAP には **messaging-activemq** サブシステムの設定の一部として単一の **address-setting** 要素が含まれています。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <address-setting
      name="#"
      dead-letter-address="jms.queue.DLQ"
      expiry-address="jms.queue.ExpiryQueue"
```

```
        max-size-bytes="10485760"
        page-size-bytes="2097152"
        message-counter-history-day-limit="10" />
      ...
  </server>
</subsystem>
```

注記

name 属性に単一の **#** を使用すると、**#** はすべてのアドレスに一致するため、このデフォルトの **address-setting** がすべての宛先に使用される設定となります。このキャッチオール設定をこのまますべてのアドレスに適用するか、独自の設定セットを必要とする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。

管理 CLI を使用したアドレス設定の設定

アドレス設定は、管理 CLI または管理コンソールを使用して設定できますが、管理 CLI の方が編集用に多くの設定属性が公開されています。属性の詳細リストは、本ガイドの付録の「アドレス設定の属性」を参照してください。

新しい address-setting の追加

必要に応じて **add** 操作を使用して新規アドレス設定を作成します。管理 CLI セッションのルートからこのコマンドを実行できます。以下の例では、名前を指定した新しいパターンが作成されます。**address-setting** の設定属性を含めることができます。以下では、**news.europe.#** に一致する新しい **address-setting** が作成され、その **dead-letter-address** 属性が、あらかじめ作成されたキュー **DLQ.news** に設定されます。スタンドアロンサーバーと、**full** プロファイルを使用した管理対象サーバードメインの両方の例をそれぞれ示します。

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

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

address-setting 属性の編集

write-attribute 操作を使用して新しい値を属性に書き込みます。タブ補完を使用すると、入力時のコマンド文字列の補完に役立つほか、利用可能な属性を明らかにできます。以下の例は、**max-bearery-attempts** の値を **10** に更新します。

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

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

address-setting 属性の読み取り

値が変更されたことを確認するには、**include-runtime=true** パラメーターを指定して **read-resource** 操作を実行し、サーバーモデルでアクティブな現在の値をすべて開示します。

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

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

管理コンソールを使用したアドレス設定の設定

以下の手順に従うと、管理コンソールを使用してアドレス設定の作成およびレビューを実行できます。

1. 管理コンソールにログインします。
2. 画面上部の**設定**タブを選択します。管理対象ドメインを実行している場合は、更新するプロファイルを選択します。
3. **Messaging (ActiveMQ) → サーバー**を選択します。
4. メッセージングサーバーを選択します。デフォルト設定では、**default** という 1 台のサーバーのみが表示されます。
5. **宛先**を選択 し、**表示**をクリックします。
6. **アドレス設定**タブを選択して、アドレス設定を設定します。

news.europe.# などの新しいパターンを追加する場合、**Pattern** フィールドは **address-setting** 要素の **name** 属性を参照します。管理 CLI を使用して属性の読み取り/書き込みを行う際にこの値を入力します。

管理コンソールを使用する場合は、**dead-letter-address**、**expiry-address**、**redelivery-delay**、および **max-delivery-attempts** 属性のみを編集できます。その他の属性は、管理 CLI を使用して設定する必要があります。

メッセージングサーバーのグローバルリソースの使用設定

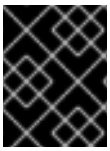
address-setting 要素の 3 つの属性により、メッセージングサーバーのグローバルリソースの使用を制御できます。

属性	説明
global-max-memory-size	メモリーのアドレスが満杯になり、**address-full-policy** の適用が開始される前に、Artemis がそのアドレスのメッセージ保存に使用できるメモリーの最大量を制御します。デフォルト値は **-1** で、制限なしを意味します。
global-max-disk-usage	Artemis がファイルシステムにデータを保存するために使用できる最大領域を制御します。制限に達すると、新しいメッセージはブロックされます。この属性は、ディスク上で利用可能な領域をパーセンテージで示します。最小は **0%** で、最大は **100%** です。デフォルト値は **100%** です。
disk-scan-period	Artemis がファイルシステムで利用可能な領域をチェックする頻度を制御します。デフォルトの値は **5000 milliseconds** です。

6.3. LAST-VALUE キュー

last-value キューは、明確に定義された last-value プロパティーと同じ値のメッセージが新たにキューに入れられると、メッセージをすべて破棄する特別なキューです。つまり、last-value キューは最後の

値のみを保持します。last-value キューの代表的な応用例には、特定の株式の最終株価のみに関心がある場合の株価があります。



重要

キューのページングが有効になっていると、last-value キューは期待通りに動作しません。last-value キューを使用する前に、ページングを無効にしてください。

last-value キューの設定

last-value キューは、**address-setting** 設定要素内に定義されます。

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

管理 CLI を使用して、特定の **address-setting** の **last-value-queue** の値を読み取ります。

```
/subsystem=messaging-activemq/server=default/address-setting=news.europe.#:read-
attribute(name=last-value-queue)
{
    "outcome" => "success",
    "result" => false
}
```

last-value-queue に許可される値は **true** または **false**です。管理 CLI を使用して、次のようにいずれかの値を設定します。

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

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

last-value プロパティーの使用

最終値の特定に使用されるプロパティー名は **_AMQ_LVQ_NAME** (またはコア API の定数 **Message.HDR_LAST_VALUE_NAME**) です。以下の Java コードで last-value プロパティーの使用方法を示します。

- まず、パブリッシャーが最終値キューにメッセージを送信します。

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

- 次に、同じ last-value を使用して別のメッセージをキューに送信します。

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

- 次に、コンシューマーが last-value を持つメッセージを受け取ります。

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

上記の例では、両方のメッセージが **_AMQ_LVQ_NAME** を **"MY_MESSAGE"** に設定しており、最初のメッセージの後に 2 番目のメッセージがキューで受信されたため、クライアントの出力は **"My 2nd message with the last-value property set"** になります。

第7章 セキュリティーの設定

7.1. リモート接続のセキュリティー保護

7.1.1. レガシーセキュリティーサブシステムの使用

JBoss EAP でレガシー **security** サブシステムを使用して、**messaging-activemq** サブシステムをセキュアにできます。レガシー **security** サブシステムはレガシーセキュリティーレルムおよびドメインを使用します。セキュリティーレルムおよびセキュリティー ドメインに関する詳細は、JBoss EAP『セキュリティーアーキテクチャー』ガイドを参照してください。**messaging-activemq** サブシステムは、**ApplicationRealm** という名前のセキュリティーレルムと **other** という名前のセキュリティードメインを使用するよう事前設定されています。

注記

レガシー **security** サブシステムのアプローチは、JBoss EAP 7.0 のデフォルト設定です。

ApplicationRealm は、設定ファイルの最上部付近で定義されます。

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

この名前が示すように、**ApplicationRealm** は **messaging-activemq**、**undertow**、**ejb3** サブシステムなどの JBoss EAP のすべてのアプリケーション中心のサブシステムに対するデフォルトのセキュリティーレルムです。**ApplicationRealm** は、ローカルファイルシステムを使用して、ユーザー名とハッシュ化されたパスワードを保存します。便宜上、JBoss EAP には **ApplicationRealm** にユーザーを追加するために使用できるスクリプトが含まれています。詳細は、JBoss EAP『**How To Configure Server Security**』の「Default User Configuration」を参照してください。

other セキュリティードメインは、**messaging-activemq** などのアプリケーション関連のサブシステムに対するデフォルトのセキュリティードメインです。これは設定で明示的に宣言されるものではありませんが、以下の管理 CLI コマンドを使用すると、**messaging-activemq** サブシステムで使用されるセキュリティードメインを確認することができます。

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

```
{
    "outcome" => "success",
    "result" => "other"
}
```

また、使用するセキュリティードメインを更新することもできます。

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

JBoss EAP『How To Configure Server Security』ガイドには、新しいセキュリティーレルムおよびドメインの作成方法についての詳細が記載されています。ここでは、**other** ドメインが設定内にどのように表示されるかに注目すべきです。

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

「other」ドメインでは、認証手段として 2 つの login-module を使用しています。最初のモジュール **Remoting** がリモート EJB 呼び出しを認証し、**RealmDirect** モジュールが、指定されたレルムに定義されている保存情報を使用してユーザーを認証します。この例では、レルムが宣言されていないため、デフォルトのレルム **ApplicationRealm** が使用されます。各モジュールの **password-stacking** オプションは **useFirstPass** に設定されており、認証されたユーザーのプリンシパル名とパスワードを格納するよう login-module に伝えます。ログインモジュールおよびそれらのオプションに関する詳細は、JBoss EAP『Login Module Reference』を参照してください。

ロールベースのアクセスはアドレスレベルで設定されます。アドレスの「ロールベースのセキュリティー」を参照してください。

7.1.2. Elytron サブシステムの使用

elytron サブシステムを使用して **messaging-activemq** サブシステムをセキュアにすることもできます。**elytron** サブシステムの使用および Elytron セキュリティードメインの作成に関する詳細は、『**How to Configure Identity Management**』ガイドの「Elytron Subsystem」を参照してください。

Elytron セキュリティードメインを使用するには、以下を指定します。

1. レガシーセキュリティードメインの定義を解除します。

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

2. Elytron セキュリティードメインを設定します。

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

reload
```

7.1.2.1. 管理コンソールを使用した Elytron セキュリティードメインの設定

管理コンソールを使用して Elytron セキュリティードメインを設定するには、以下の操作を行います。

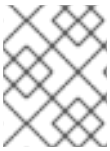
1. 管理コンソールにアクセスします。詳細は、JBoss EAP『設定ガイド』の「 管理コンソール 」を参照してください。
2. **Configuration → Subsystems → Messaging (ActiveMQ) → Server → default** と選択し、**View** をクリックします。
3. **Security** タブに移動し、**Edit** をクリックします。
4. **Elytron Domain**の値を追加または編集します。
5. **保存**をクリックして変更を保存します。
6. 変更を反映するためにサーバーをリロードします。

注記

security-domain または **elytron-domain** のいずれかを定義できますが、両方を同時に定義することはできません。いずれも定義されていない場合、JBoss EAP は **other** のデフォルト値 **security-domain** を使用します。これは **other** のレガシーセキュリティードメインに対応します。

7.1.3. トランスポートのセキュリティー保護

JBoss EAP メッセージングにバンドルされたデフォルトの **http-connector** は、デフォルトではセキュリティー保護されていません。JBoss EAP『 **How to Configure Server Security**』の「configure one-way and two-way SSL/TLS for applications」の手順に従って、メッセージトランスポートのセキュリティーを保護し、SSL/TLS の Web トラフィックを有効にできます。



注記

メッセージトランスポートのセキュリティーを保護する上記の方法は、**http-acceptor** のセキュリティー保護にも使用できます。

上記のようにトランスポートを設定する場合は、以下の手順を追加する必要があります。

- デフォルトでは、HTTP アクセプターはすべて、デフォルトの **http-listener** を使用し、その HTTP ポートでリッスンするように設定されています。HTTP アクセプターは **https-listener** を使用して HTTPS ポートでリッスンするように設定する必要があります。
- すべての HTTP コネクターの **socket-binding** 要素は、**http** ではなく **https** を使用するように更新する必要があります。
- SSL/TLS 経由で通信する **http-connector** の **ssl-enabled** パラメーターは **true** に設定する必要があります。
- HTTP コネクターが別のサーバーへの接続に使用される場合は、**trust-store** や **key-store** など

の関連するパラメーターを設定する必要があります。**http-connector** のセキュリティーを保護するには、**remote-connector** と同様のパラメーター設定を行う必要があります。設定の詳細は「リモートコネクターのセキュリティー保護」で説明されています。

メッセージングトランスポートのアクセプターとコネクターの設定の詳細は、「メッセージングトランスポートの設定」を参照してください。

7.1.4. リモートコネクターのセキュリティー保護

TCP 通信にデフォルトの **http-connector** を使用せず、代わりに独自の **remote-connector** と **remote-acceptor** を作成した場合、以下の表のプロパティーを使用して SSL/TLS を設定できます。以下のプロパティーは、アクセプターまたはコネクターの子 **<param>** 要素の一部として設定に表示されます。

通常、サーバーは SSL/TLS 秘密鍵を所有し、その公開鍵をクライアントと共有します。この場合、サーバーは **remote-acceptor** で **key-store-path** パラメーターおよび **key-store-password** パラメーターを定義します。各クライアントは異なる場所にトラストストアを持ち、異なるパスワードで暗号化できます。そのため、**remote-connector** で **trust-store-path** プロパティーと **trust-store-password** プロパティーを指定することは推奨されません。代わりに、システムプロパティー **javax.net.ssl.trustStore** と **javax.net.ssl.trustStorePassword** を使用してクライアント側のこれらのパラメーターを設定します。**remote-connector** で設定する必要があるパラメーターは **ssl-enabled=true** だけです。ただし、サーバーが **remote-connector** を使用して別のサーバーに接続する場合は、**remote-connector** の **trust-store-path** パラメーターと **trust-store-password** パラメーターを設定するのが適切です。

上記のユースケースでは、以下の管理 CLI コマンドを使用して **remote-acceptor** を作成します。

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

上記のユースケースから **remote-connector** を作成するには、以下の管理 CLI コマンドを使用します。

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

管理 CLI を使用して、次のように既存の **remote-acceptor** または **remote-connector** にパラメーターを追加することもできます。

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

remote-acceptor と **remote-connector** では両方とも、通信に使用されるポートを宣言する際に **socket-binding** を参照することに注意してください。ソケットバインディングおよびアクセプターとコネクターとの関係についての詳細は、「メッセージングサブシステム設定の概要」を参照してください。

表7.1 NettyConnectorFactory の SSL/TLS 関連の設定プロパティー

プロパティー	説明
enabled-cipher-suites	アクセプターまたはコネクターの設定に使用できます。これは、SSL/TLS 通信に使用される暗号化スイートのコンマで区切ったリストです。デフォルト値は null です (JVM のデフォルト値を使用)。

プロパティー	説明
enabled-protocols	アクセプターまたはコネクターの設定に使用できます。これは、SSL/TLS 通信に使用されるプロトコルのコンマ区切りのリストです。デフォルト値は null です (JVM のデフォルト値を使用)。
key-store-password	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/TLS を使用している場合、これはコネクターにのみ関係してきます。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、標準の **javax.net.ssl.keyStorePassword** 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。クライアント上の別のコンポーネントで標準のシステムプロパティーをすでに使用している場合は、**org.apache.activemq.ssl.keyStorePassword** プロパティーを使用します。
key-store-path	アクセプターで使用される場合、これはサーバーの証明書を保持するサーバーの SSL/TLS キーストアへのパスとなります。自己署名証明書または認証局が署名した証明書のいずれかに使用します。 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/TLS キーストアへのパスとなります。双方向 SSL/TLS を使用している場合、これはコネクターにのみ関係してきます。 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、標準の **javax.net.ssl.keyStore** 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。クライアント上の別のコンポーネントで標準のプロパティーをすでに使用している場合は、**org.apache.activemq.ssl.keyStore** システムプロパティーを使用します。
key-store-provider	キーを保存するファイルの形式 (PKCS11 や PKCS12 など) を定義します。許可される値は JDK に固有のものです。
needs-client-auth	このプロパティーはアクセプター専用です。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/TLS 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。デフォルトは **false** です。
ssl-enabled	SSL/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。

プロパティー	説明
trust-store-password	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/TLS を使用している場合、これはアクセプターにのみ関係してきます。 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、標準の **javax.net.ssl.trustStorePassword** 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。クライアント上の別のコンポーネントで標準のプロパティーをすでに使用している場合は、**org.apache.activemq.ssl.trustStorePassword** システムプロパティーを使用します。
trust-store-path	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/TLS キーストアへのパスとなります。双方向 SSL/TLS を使用している場合、これはアクセプターにのみ関係してきます。 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、標準の **javax.net.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。クライアント上の別のコンポーネントがすでに標準のシステムプロパティーを使用している場合は、**org.apache.activemq.ssl.trustStore** システムプロパティーを使用します。
trust-store-provider	キーを保存するファイルの形式 (PKCS11 や PKCS12 など) を定義します。許可される値は JDK に固有のものです。

7.2. 宛先のセキュリティー保護

メッセージングサーバーへのリモート接続のセキュリティーを保護する他に、特定の宛先のセキュリティーを設定することもできます。この設定をする場合は、**security-setting** 設定要素を使用してセキュリティー制約を追加します。JBoss EAP メッセージングには、以下の管理 CLI コマンドの出力のような **security-setting** がデフォルトで設定されています。

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

```
            "create-non-durable-queue" => true,
            "delete-durable-queue" => false,
            "delete-non-durable-queue" => true,
            "manage" => false,
            "send" => true
        }}}}
    }
}
```

security-setting オプションでは、**name** フィールドにワイルドカードを使用して、セキュリティー制約を適用する宛先を指定できます。**#** 1つでアドレスが任意であることを表します。セキュリティー制約でのワイルドカードの使用に関する詳細は、「アドレスのロールベースのセキュリティー」を参照してください。

7.2.1. アドレスのロールベースのセキュリティー

JBoss EAP メッセージングでは、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、柔軟なロールベースのセキュリティーモデルを採用しています。

コア JBoss EAP メッセージングサーバーは、主にアドレスにバインドされたキューのセットで構成されます。メッセージをアドレスに送信すると、サーバーはまずそのアドレスにバインドされたキューのセットをルックアップし、そのメッセージをバインドされたキューに転送します。

JBoss EAP メッセージングには、アドレスに基づいてキューに適用できる一連のパーミッションがあります。アドレスの文字列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。ワイルドカード構文の使用方法の詳細は、「アドレス設定」を参照してください。

security-setting ごとに複数のロールを作成できます。またロールに適用できるパーミッション設定は 7 つあります。利用可能なパーミッションの全リストは次のとおりです。

- **create-durable-queue** は、一致するアドレスの永続キューの作成をロールに許可します。
- **delete-durable-queue** は、一致するアドレスの永続キューの削除をロールに許可します。
- **create-non-durable-queue** は、一致するアドレスの非永続キューの作成をロールに許可します。
- **delete-non-durable-queue** は、一致するアドレスの非永続キューの削除をロールに許可します。
- **Send** は、一致するアドレスへのメッセージの送信をロールに許可します。
- **consume** は、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をロールに許可します。
- **manage** は、管理アドレスに管理メッセージを送信して管理操作を起動することをロールに許可します。

ロールベースのセキュリティーの設定
security-setting にロールベースのセキュリティーを使用するには、最初に security-setting を作成する必要があります。たとえば、次のように **news.europe.#** の **security-setting** を作成します。これは **news.europe.fr** や **news.europe.tech.uk** などの **news.europe.** から始まる宛先に適用されます。

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

次に、作成した **security-setting** にロールを追加し、そのロールのパーミッションを宣言します。以下の例では、**dev** ロールが作成され、キューからの消費とキューへの送信のパーミッションが付与されます。また非永続キューの作成と削除も同様です。デフォルトは **false** であるため、オンにするパーミッションのみを JBoss EAP に指示する必要があります。

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

パーミッションの使用方法を詳しく説明すると、以下の例では **admin** ロールを作成し、**manage** パーミッションをオンにして管理メッセージを送信できるようにします。永続的なキューの作成と削除を行うためのパーミッションもオンにします。

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

security-setting の設定を確認するには、管理 CLI を使用します。**recursive=true** オプションを使用して、パーミッションの全体表示を指定することを忘れないでください。

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

```
        }
    }}
}
```

上記の例では、文字列 **news.europe.** から始まるアドレスのパーミッションはすべて、管理 CLI で表示されます。要約すると、**admin** ロールを持つユーザーのみが永続キューの作成または削除を行うことができます。また、**dev** ロールを持つユーザーのみが非永続キューの作成または削除を行うことができます。さらに、**dev** ロールを持つユーザーは、メッセージを送信または消費できますが、**admin** ユーザーはできません。ただし、**manage** パーミッションが **true** に設定されているため、管理メッセージは送信できます。

アドレスのセットに複数の一致が当てはまる場合は、より限定的な一致が優先されます。たとえば、**news.europe.tech.uk.#** アドレスは **news.europe.tech.#** よりも限定的です。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。

ユーザーと、ユーザーが持つロールのマッピングは、セキュリティーマネージャーが管理します。JBoss EAP には、ディスク上のファイルからユーザーの認証情報を読み取るユーザーマネージャーが含まれています。ユーザーマネージャーは JAAS または JBoss EAP セキュリティーにプラグインすることもできます。

セキュリティーマネージャーの設定に関する詳細は、JBoss EAP『**セキュリティーアーキテクチャー**』ガイドを参照してください。

7.2.1.1. レガシーセキュリティーサブシステムを使用した非認証クライアントのゲストロールの許可

認証されていないクライアントを JBoss EAP に自動許可させる必要がある場合、**guest** ロールでは以下の 2 つの変更を行います。

1. 新しい **module-option** を **other** セキュリティードメインに追加します。新しいオプションの **unauthenticatedIdentity** は、認証されていないクライアントに **guest** アクセスを許可するよう JBoss EAP に指示します。これは管理 CLI を使用して実施することを推奨します。

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

コマンドを発行した後にサーバーのリロードが必要なことに注意してください。以下の管理 CLI コマンドを使用すると新しいオプションを確認できます。

```
/subsystem=security/security-domain=other/authentication=classic/login-
module=RealmDirect:read-resource()
{
    "outcome" => "success",
    "result" => {
        "code" => "RealmDirect",
```

```
        "flag" => "required",
        "module" => undefined,
        "module-options" => {
            "password-stacking" => "useFirstPass",
            "unauthenticatedIdentity" => "guest"
        }
    }
}
```

またサーバー設定ファイルは、コマンドの実行後に以下のようになります。

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

2. # 文字を削除し、**application-roles.properties** ファイルの以下の行をコメント解除します。ファイルは **EAP_HOME/standalone/configuration/** または **EAP_HOME/domain/configuration/** のいずれかにあります。これは、スタンドアロンサーバー、ドメインコントローラーのいずれを使用しているかに応じて異なります。

```
#guest=guest
```

これで、リモートクライアントは認証せずにサーバーにアクセスできるようになりました。**guest** ロールに関連するパーミッションが付与されます。

7.3. JMS OBJECTMESSAGE デシリアライズの制御

ObjectMessage には潜在的に危険性があるオブジェクトを含めることができます。そのため、ActiveMQ Artemis では信頼できるパッケージやクラスと信頼できないパッケージやクラスを管理するシンプルなクラスのフィルタリングメカニズムを提供します。信頼できるパッケージのクラスを持つオブジェクトをホワイトリストに追加することで、問題なくデシリアライズできることを示すことができます。信頼できないパッケージのクラスを持つオブジェクトはブラックリストに追加することで、デシリアライズされないようにすることができます。

ActiveMQ Artemis は以下のようにデシリアライズするオブジェクトをフィルターにかけます。

- ホワイトリストおよびブラックリストの両方が空の場合 (デフォルト)、シリアライズ可能なオブジェクトはすべてデシリアライズできます。
- オブジェクトのクラスまたはパッケージがブラックリスト内のエントリーのいずれかに一致する場合は、デシリアライズすることはできません。

- オブジェクトのクラスまたはパッケージがホワイトリスト内のエントリーと一致する場合は、デシリアライズすることができます。
- オブジェクトのクラスまたはパッケージが、ブラックリストとホワイトリストの両方のエントリーと一致する場合は、ブラックリスト内のクラスが優先されます。つまり、デシリアライズはできません。
- オブジェクトのクラスまたはパッケージがブラックリストとホワイトリストの両方に一致しない場合は、ホワイトリストが空の場合 (ホワイトリストが指定されていない場合) を除き、オブジェクトデシリアライズは拒否されます。

オブジェクトのフルネームがリスト内のエントリーのいずれかと完全に一致する場合、そのパッケージがリストにあるエントリーのいずれかと一致する場合、またはこれがリスト内にあるエントリーのいずれかのサブパッケージである場合、オブジェクトは一致すると見なされます。

deserialization-white-list 属性と **deserialization- black-list** 属性を使用して、**connection-factory** と **pooled-connection-factory** でデシリアライズできるオブジェクトを指定できます。**deserialization-white-list** 属性は、デシリアライズできるクラスまたはパッケージのリストを定義するために使用されます。**deserialization-black-list** 属性は、デシリアライズできないクラスまたはパッケージのリストを定義するために使用されます。

以下のコマンドにより、デフォルトサーバーの **RemoteConnectionFactory** 接続ファクトリーのブラックリストおよび **activemq-ra** プールされた接続ファクトリーのホワイトリストが作成されます。

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

これらのコマンドにより、**messaging-activemq** サブシステムで以下の設定が生成されます。

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

接続ファクトリーとプールされた接続ファクトリーの詳細は、本ガイドの「接続ファクトリーの設定」を参照してください。

また、アクティベーションプロパティーを設定して、MDB でデシリアライズできるオブジェクトを指定することもできます。**deserializationWhiteList** プロパティーは、デシリアライズできるクラスまたはパッケージのリストを定義するために使用されます。**deserializationBlackList** プロパティーは、デシリアライズできないクラスまたはパッケージのリストを定義するために使用されます。アクティベーションプロパティーの詳細は、JBoss EAP 『**Developing EJB Applications**』の「Configuring MDBs Using a Deployment Descriptor」を参照してください。

7.4. 認可インバリデーション管理

security-invalidation-interval サブシステムのサーバーの **messaging-activemq** 属性は、アクションを再認証する必要がある前に承認がキャッシュされる期間を決定します。

システムがアドレスでアクションの実行をユーザーに承認すると、承認はキャッシュされます。次に同じユーザーが同じアドレスで同じアクションを実行すると、システムはキャッシュされた承認をアクションに使用します。

たとえば、ユーザー **admin** はメッセージをアドレス **news** に送信しようとします。システムはアクションを承認し、承認をキャッシュします。次に **管理者** がメッセージを送信しようとすると、**システム** はキャッシュされた承認を使用します。

キャッシュされた承認が無効化間隔で指定された時間内に再度使用されない場合、承認はキャッシュから消去されます。システムは、要求されたアドレスで要求されたアクションを実行するためにユーザーを再認証する必要があります。

インストール後に、JBoss EAP では、デフォルト値が 10000 ミリ秒 (10 秒) となります。

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

security-invalidation-interval 属性は設定可能です。たとえば、以下のコマンドは間隔を 60000 ミリ秒 (60 秒または 1 分) に更新します。

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

設定変更を有効にするには、サーバーをリロードする必要があります。

属性を読み取ると、新しい結果が表示されます。

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

第8章 メッセージングトランスポートの設定

ここでは、JBoss EAP メッセージングトランスポート (特にコネクターとアクセプター) を理解するために必要な概念について説明します。アクセプターは、サーバーが接続を許可する方法を定義するためにサーバーで使用されます。一方、コネクターはクライアントのサーバーへの接続方法を定義するためにクライアントで使用されます。それぞれの概念を順番に説明し、実用的な例として、クライアントが JNDI または Core API を使用した、JBoss EAP メッセージングサーバーへの接続方法を示します。

8.1. アクセプターおよびコネクターのタイプ

JBoss EAP の設定では、3 つの主要なタイプのアクセプターとコネクターが定義されています。

in-vm: In-vm は Intra Virtual Machine の省略形です。クライアントおよびサーバーの両方が同じ JVM で実行されている場合は、このコネクタータイプを使用します (例: Message Driven bean (MDB) が JBoss EAP の同じインスタンスで実行されている場合)。

http: クライアントとサーバーが異なる JVM で実行されている場合に使用します。**undertow** サブシステムのデフォルトのポートである **8080** を使用すると、HTTP による多重メッセージ通信が可能になります。Red Hat は、クライアントおよびサーバーが異なる JVM で実行されている場合、特にクラウド環境におけるポート管理などを考慮して、**http** コネクターを使用することを推奨します。

remote: リモートトランスポートは、クライアントとサーバーが異なる JVM で実行されている場合に、ネイティブ TCP 通信に使用される Netty ベースのコンポーネントです。このコンポーネントは **http** を使用できない場合の代替手段となります。

クライアントは、いずれかのサーバーのアクセプターと互換性のあるコネクターを使用する必要があります。たとえば、**in-vm-connector** のみが **in-vm-acceptor** に接続でき、**http-connector** のみが **http-acceptor** に接続できます。

管理 CLI の **read-children-attributes** 操作を使用すると、指定のアクセプターまたはコネクタータイプの属性を一覧表示できます。たとえば、デフォルトのメッセージングサーバーのすべての **http-connectors** の属性を表示するには、以下の入力を行います。

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

同様のコマンドを使用して、すべての **http-acceptors** の属性を読み込みます。

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

他のアクセプターおよびコネクタータイプは同じ構文に従います。**child-type** で **remote-connector** や **in-vm-acceptor** などのアクセプタータイプまたはコネクタータイプを指定するだけです。

8.2. アクセプター

アクセプターは、JBoss EAP 統合メッセージングサーバーによって許可される接続のタイプを定義します。サーバーごとに任意の数のアクセプターを定義できます。以下の設定例はデフォルトの **full-ha** 設定プロファイルを修正したもので、各アクセプタータイプの例を示しています。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <http-acceptor name="http-acceptor" http-listener="default"/>
```

```
    <remote-acceptor name="legacy-messaging-acceptor" socket-binding="legacy-messaging"/>
    <in-vm-acceptor name="in-vm" server-id="0"/>
    ...
  </server>
</subsystem>
```

上記の設定では、**http-acceptor** は JBoss EAP のデフォルトの http ポート 8080 をリッスンする Undertow のデフォルト **http-listener** を使用しています。**http-listener** は **undertow** サブシステムで定義されます。

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

また、上記の **remote-acceptor** が **legacy-messaging** という名前の **socket-binding** を使用する仕組みに注意してください。これは、サーバーのデフォルトの **socket-binding-group** の一部として、設定の後半で定義されています。

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

この例では、**legacy-messaging socket-binding** は JBoss EAP をポート **5445** にバインドしており、上記の **remote-acceptor** がレガシークライアントで使用するために **messaging-activemq** サブシステムの代わりにポートを要求しています。

最後に、**in-vm-acceptor** は **server-id** 属性に一意の値を使用しており、このサーバーインスタンスを同じ JVM で実行されている他のサーバーと区別することができます。

8.3. コネクター

コネクターは、統合 JBoss EAP メッセージングサーバーに接続する方法を定義しており、クライアントで接続するために使用されます。

コネクターは実際にはクライアントで使用されるのに、なぜサーバーで定義するのかとお考えになるかもしれません。これには、以下のような理由があります。

- 場合によっては、サーバーが別のサーバーに接続するときにクライアントとして動作する場合があります。たとえば、あるサーバーが別のサーバーへのブリッジとして機能するか、またはクラスターに参加する必要がある可能性があります。この場合、サーバーは他のサーバーに接続する方法を認識する必要があり、その接続方法はコネクターで定義されます。
- サーバーが JNDI でクライアントからルックアップされる **ConnectionFactory** を使用するコネクターを提供できるため、サーバーへの接続の作成が簡単になります。

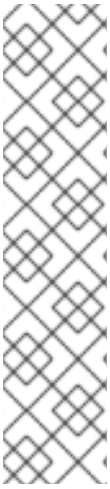
サーバーごとに任意の数のコネクターを定義できます。以下の設定例は **full-ha** 設定プロファイルをベースとしたもので、各タイプのコネクターが含まれています。

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

full-ha プロファイルの **http-acceptor** と同様に、**http-connector** は **undertow** サブシステムで定義されたデフォルトの **http-listener** を使用します。**endpoint** 属性は、接続する **http-acceptor** を宣言します。この場合、コネクターはデフォルトの **http-acceptor** に接続します。

JBoss EAP 7.1 では、**http-connector** に新しい **server-name** 属性が導入されました。この新しい属性はオプションですが、複数の ActiveMQ Artemis インスタンスを実行しているリモートサーバーの適切な **http-acceptor** に接続できることが要求されます。この属性が定義されていない場合、値は実行時に解決され、コネクターが定義されている親 ActiveMQ Artemis サーバーの名前になります。

また、**remote-connector** は **remote-acceptor** のカウンターパートとして同じ **socket-binding** を参照します。最後に、**in-vm-connector** と **in-vm-acceptor** は同じサーバーインスタンス内で実行しているため、**server-id** に同じ値を使用します。



注記

パブリックインターフェースのバインドアドレスが **0.0.0.0** に設定されている場合は、JBoss EAP サーバーの起動時に以下の警告がログに表示されます。

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

これは、リモートコネクターが **0.0.0.0** アドレスを使用してサーバーに接続できず、**messaging-activemq** サブシステムがサーバーのホスト名で置き換えようとするためです。管理者は、ソケットバインディングに異なるインターフェースアドレスを使用するようリモートコネクターを設定する必要があります。

8.4. アクセプターとコネクターの設定

コネクターとアクセプターには複数の設定オプションがあります。こうしたオプションは、設定の子要素 **<param>** として表示されます。**<param>** の各要素には、コネクターまたはアクセプターをインスタンス化するデフォルトの Netty ベースのファクトリークラスによって認識され、使用される **name** と **value** の属性ペアが含まれています。

管理 CLI では、各リモートコネクターまたはアクセプターの要素に、パラメーターの名前と値のペアの内部マップが含まれます。たとえば、**myRemote** という名前の **remote-connector** に新しい **param** を追加するには、以下のコマンドを使用します。

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

同様の構文を使用してパラメーター値を取得します。

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

また、アクセプターまたはコネクターを作成する際に、以下の例のようにパラメーターを含めることもできます。

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

表8.1 トランスポート設定プロパティー

プロパティー	説明
batch-delay	パケットをトランスポートに書き込む前に、メッセージングサーバーは **batch-delay** (ミリ秒単位) を上限として書き込みを一括処理するよう設定できます。これにより、メッセージ転送の平均待ち時間が長くなり、非常に小さいメッセージのスループットの合計が増加します。デフォルトは 0 です。
direct-deliver	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になりますが、スループットと待ち時間は低減されます。スループットを最大限にする場合は、このプロパティーを **false** に設定します。デフォルトは **true** です。
http-upgrade-enabled	**http-connector** で使用され、HTTP アップグレードを使用しているため HTTP のメッセージングトラフィックを多重化することを指定します。このプロパティーは、**http-connector** が作成され、管理者を必要としない場合、JBoss EAP によって自動的に **true** に設定されます。
http-upgrade-endpoint	**http-connector** が接続するサーバー側の **http-acceptor** を指定します。コネクターは HTTP で多重化されます。HTTP アップグレードの後に関連する **http-acceptor** を見つけるために、コネクターはこの情報を必要とします。このプロパティーは、**http-connector** が作成され、管理者を必要としない場合、JBoss EAP によって自動的に設定されます。
local-address	http コネクターまたはリモートコネクターについて、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するために使用されます。ローカルアドレスが指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。

プロパティー	説明
local-port	http コネクターまたはリモートコネクターについて、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するために使用されます。local-port のデフォルトに (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。有効なポートの値は **0** から **65535** です。
nio-remoting-threads	NIO を使用するように設定されている場合、メッセージング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には、スレッド数にカスタム値を設定できます。デフォルトは **-1** です。
tcp-no-delay	**true** の場合、Nagle アルゴリズムが有効になります。このアルゴリズムによりネットワーク上で送信されるパケット数が減少し、TCP/IP ネットワークの効率を向上させることができます。デフォルトは **true** です。
tcp-send-buffer-size	このパラメーターにより、バイト単位で TCP 送信バッファーのサイズが決まります。デフォルトは **32768** です。
tcp-receive-buffer-size	このパラメーターは、バイト単位で TCP 受信バッファーのサイズを指定します。デフォルトは **32768** です。
use-nio-global-worker-pool	このパラメーターにより、各コネクションに独自のプールを持つのではなく、すべての JMS 接続で Java スレッドの 1 つのプールを共有します。これは、オペレーティングシステムのプロセスの最大数を使い切らないようにするためのものです。デフォルトは **true** です。

8.5. サーバーへの接続

クライアントをサーバーに接続する場合は、適切なコネクターが必要です。これには 2 つの方法があります。サーバーに設定され、JNDI ルックアップから取得できる ConnectionFactory を使用することができます。または、ActiveMQ Artemis コア API を使用して、クライアント側で **ConnectionFactory** 全体を設定することもできます。

8.5.1. JMS 接続ファクトリー

クライアントでは JNDI を使用して、サーバーへの接続を提供する ConnectionFactory オブジェクトをルックアップできます。接続ファクトリーでは、それぞれ以下の 3 つのタイプのコネクターを公開することができます。

remote-connector を参照する **connection-factory** は、リモートクライアントがサーバーからメッセージを送受信するために使用できます (connection-factory に適切なエクスポートエントリーがあることを前提とします)。**remote-connector** は、**connection-factory** の使用して接続先をクライアントに指示する **socket-binding** と関連付けられます。

in-vm-connector を参照する **connection-factory** は、ローカルクライアントよるローカルサーバーとのメッセージの送受信にに適しています。**in-vm-connector** は、複数のメッセージングサーバーを 1 つの JVM で実行できるため、**connection-factory** を使用して接続先をクライアントに指示する **server-id** と関連付けられます。

http-connector を参照する **connection-factory** は、HTTP ポートに接続してサーバーからメッセージを送受信してからメッセージングプロトコルにアップグレードするにあたり、リモートクライアントで使用するのに適しています。**http-connector** は、デフォルトで **http** という名前の HTTP ソケットを表す **socket-binding** に関連付けられています。

JMS 2.0 以降、デフォルトの JMS 接続ファクトリーは、JNDI 名 **java:comp/DefaultJMSConnectionFactory** で Java EE アプリケーションを利用できます。**messaging-activemq** サブシステムでは、このデフォルトの接続ファクトリーを提供するために使用される **pooled-connection-factory** を定義します。

以下は、JBoss EAP の **full** 設定プロファイルに含まれるデフォルトのコネクターおよび接続ファクトリーです。

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

ファクトリーの **entries** 属性では、ファクトリーが公開される JNDI 名を指定します。リモートクライアントに利用できるのは、**java:jboss/exported** 名前空間にバインドされた JNDI 名のみです。**connection-factory** が **java:jboss/exported** 名前空間にバインドされたエントリーを持っている場合、リモートクライアントは **java:jboss/exported** の後のテキストを使用して **connection-factory** をルックアップします。たとえば、**RemoteConnectionFactory** はデフォルトで **java:jboss/exported/jms/RemoteConnectionFactory** にバインドされています。これは、リモートクライアントが **jms/RemoteConnectionFactory** を使用してこの connection-factory をルックアップすることを意味します。**pooled-connection-factory** はリモートクライアントに適していないため、**pooled-connection-factory** では **java:jboss/exported** 名前空間内にエントリーがバインドされないようにする必要があります。

8.5.2. JNDI を使用したサーバーへの接続

クライアントがサーバーと同じ JVM 内に存在する場合は、**InVmConnectionFactory** が提供する **in-vm** コネクターを使用できます。以下は、**standalone-full.xml** などでみられる **InvmConnectionFactory** の典型的な設定方法です。

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

entries 属性の値に注意してください。**InVmConnectionFactory** を使用するクライアントは、以下の例のように、ルックアップ時に先頭の **java:/** を省略します。

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

リモートクライアントでは **RemoteConnectionFactory** を使用しますが、これは通常以下のような設定になります。

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

リモートクライアントでは **entries** の値の先頭の **java:jboss/exported/** を無視し、以下のコードスニペットの例に従います。

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

PROVIDER_URL プロパティーの値と、クライアントの JBoss EAP http-remoting プロトコルの使用方法に注意してください。クライアントの **org.wildfly.naming.client.WildFlyInitialContextFactory** の使用方法にも注意してください。これはクライアントがこのクラスを持ち、クラスパスのどこかにクライアント JAR を包含するということを意味します。Maven プロジェクトでは、以下の依存関係を含めることで実現できます。

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

8.5.3. コア API を使用したサーバーへの接続

Core API を使用すると、JNDI ルックアップを使用せずにクライアント接続を行うことができます。Core API を使用するクライアントでは、JNDI ベースのクライアントと同様、クラスパスでクライアント JAR が必要になります。

ServerLocator

クライアントは **ServerLocator** インスタンスを使用して **ClientSessionFactory** インスタンスを作成します。名前が示すように、**ServerLocator** インスタンスはサーバーを検索し、そのサーバーへの接続を作成するために使用されます。

JMS では、JMS Connection Factory と同様に **ServerLocator** を考えてみましょう。

ServerLocator インスタンスは、**ActiveMQClient** ファクトリークラスを使用して作成されます。

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

-

ClientSessionFactory
クライアントは **ClientSessionFactory** を使用して **ClientSession** インスタンスを作成します。これは基本的にサーバーへの接続です。JMS 関係ではこれを JMS 接続と考えます。

ClientSessionFactory インスタンスは **ServerLocator** クラスを使用して作成されます。

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

ClientSession
クライアントは **ClientSession** を使用してメッセージの消費と生成を行い、トランザクションでメッセージをグループ化します。**ClientSession** インスタンスは、トランザクションセマンティクスと非トランザクションセマンティクスの両方をサポートし、JTA トランザクションの一部として、メッセージング操作を実行できるように XAResource インターフェースも提供できます。

ClientSession インスタンスは **ClientConsumers** と **ClientProducers** をグループ化します。

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

今説明したことを簡単な例で以下に示します。

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

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

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

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

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

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

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

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

session.close();
```

8.6. ロードバランサーを介したメッセージング

JBoss EAP をロードバランサーとして使用する場合、クライアントは静的 Undertow HTTP ロードバランサーまたは mod_cluster ロードバランサーの内側でメッセージングサーバーを呼び出します。

静的ロードバランサーを使用してメッセージングサーバーを呼び出すメッセージングクライアントをサポートする設定は、以下の要件を満たす必要があります。

- JBoss EAP をロードバランサーとして使用する場合は、HTTP または HTTPS を使用してロードバランサーを設定する必要があります。AJP はメッセージングロードバランサーではサポートされません。
  - Undertow を静的ロードバランサーとして設定する方法の詳細は、JBoss EAP **『設定ガイド』の「Undertow を静的ロードバランサーとして設定」**を参照してください。
- ロードバランサーの背後のメッセージングサーバーで JNDI ルックアップが発生する場合は、バックエンドメッセージングワーカーを設定する必要があります。
- クライアントがロードバランサーに接続する場合、ロードバランサーへの最初の接続を再利用して、それらが同じサーバーと通信できるようにする必要があります。クライアントがロードバランサーに接続する場合は、クラスタートポロジーを使用してロードバランサーに接続することはできません。クラスタートポロジーを使用すると、メッセージが別のサーバーに送信される可能性があります。そのため、トランザクション処理が中断される可能性があります。

mod_cluster を使用して Undertow をロードバランサーとして設定する方法の詳細は、JBoss EAP **『設定ガイド』の「mod_cluster を使用して Undertow をロードバランサーとして設定」を参照してください。**

ロードバランサーを介して通信するメッセージングクライアントの設定

ロードバランサーに接続するクライアントでは、クラスタートポロジーを使用してロードバランサーに接続するのではなく、初期接続を再利用するように設定する必要があります。

最初の接続を再利用すると、クライアントが同じサーバーに接続します。クラスタートポロジーを使用すると、メッセージが別のサーバーに転送され、トランザクション処理が中断される可能性があります。

ロードバランサーへの接続に使用される接続ファクトリーまたはプールされた接続ファクトリーは、**use-topology-for-load-balancing** 属性を false に設定する必要があります。以下の例では、CLI でこの設定を定義する方法を示しています。

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

バックエンドワーカーの設定

バックエンドメッセージングワーカーは、ロードバランサーの内側で JNDI ルックアップを行う予定がある場合のみ設定する必要があります。

1. ロードバランシングサーバーを参照する新しいアウトバウンドソケットバインディングを作成します。

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

2. ロードバランシングサーバーのソケットバインディングを参照する HTTP コネクターを作成します。

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

3. HTTP コネクターをクライアントで使用される接続ファクトリーに追加します。

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

クライアントを設定して、初期接続を再利用します。

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

第9章 接続ファクトリーの設定

デフォルトでは、JBoss EAP の **messaging-activemq** サブシステムは、**InVmConnectionFactory** および **RemoteConnectionFactory** 接続ファクトリーと **activemq-ra** プールされた接続ファクトリーを提供します。

基本的な接続ファクトリー

InVmConnectionFactory は **in-vm-connector** を参照し、クライアントとサーバーの両方が同じ JVM で実行されている場合にメッセージを送受信するために使用できます。**RemoteConnectionFactory** は **http-connector** を参照します。また、クライアントとサーバーが異なる JVM で実行されている場合に HTTP でメッセージを送受信するために使用できます。

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

各種のコネクターのタイプの詳細は、「アクセプターとコネクター」のセクションを参照してください。

接続ファクトリーの追加

以下の管理 CLI コマンドを使用すると、新しい接続ファクトリーを追加できます。接続ファクトリーを追加する場合、**connectors** と JNDI **entries** を指定する必要があります。

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

接続ファクトリーの設定

管理 CLI を使用すると、接続ファクトリーの設定を更新できます。

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

接続ファクトリーで使用できる属性の詳細は、「接続ファクトリーの属性」を参照してください。

接続ファクトリーの削除

管理 CLI を使用すると、接続ファクトリーを削除できます。

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

プールされた接続ファクトリー

JBoss EAP の **messaging-activemq** サブシステムでは、統合 ActiveMQ Artemis リソースアダプターのインバウンドコネクターとアウトバウンドコネクターの設定を可能にするプールされた接続ファクトリーを提供します。リモート ActiveMQ Artemis サーバーに接続する **pooled-connection-factory** の設定の詳細は、「リモート接続での統合リソースアダプターの使用」を参照してください。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
```

```
  <server name="default">
    ...
    <pooled-connection-factory name="activemq-ra" transaction="xa" entries="java:/JmsXA
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm"/>
  </server>
</subsystem>
```

プールされた接続ファクトリーにはいくつかの固有の特性があります。

- これはローカルクライアントでのみ利用できますが、リモートサーバーを指定するように設定することも可能です。リモート ActiveMQ Artemis サーバーへの接続の詳細は、「リモート接続での統合 Artemis リソースアダプターの使用」を参照してください。
- JNDI でのルックアップまたはインジェクトの場合のみメッセージ送信に使用する必要があります。
- セキュリティー認証情報を使用するように設定できます。これはセキュアなリモートサーバーを参照する場合に便利です。
- そこから取得されたリソースは継続中の JTA トランザクションに自動的に登録されます。

プールされた接続ファクトリーの追加

以下の管理 CLI コマンドを使用すると、新しいプールされた接続ファクトリーを追加できます。接続ファクトリーを追加する場合、**connectors** と JNDI **entries** を指定する必要があります。

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

プールされた接続ファクトリーの設定

管理 CLI を使用すると、プールされた接続ファクトリーの設定を更新できます。

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

プールされた接続ファクトリーで使用できる属性の詳細は、「プールされた接続ファクトリーの属性」を参照してください。

管理 CLI を使用して、このプールされた接続ファクトリーのエンリストメントトレースの記録を無効にするには、**enlistment-trace** 属性を **false** に設定します。

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

警告

エンリストメントトレースを無効にすると、トランザクションエンリストメント中のエラーの追跡がより困難になります。

プールされた接続ファクトリーで使用される管理対象接続プールの実装を設定することもできます。詳細は、JBoss EAP **『設定ガイド』の「管理接続プール の設定」**を参照してください。

プールされた接続ファクトリーの削除

管理 CLI を使用すると、プールされた接続ファクトリーを削除できます。

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

第10章 永続性の設定

10.1. JBOSS EAP 7 のメッセージングにおける永続性

JBoss EAP には、バインディングデータとメッセージを格納するための次の 2 つの永続性オプションが備わっています。

- デフォルトのファイルベースのジャーナルを使用できます。これはメッセージングユースケースに対して高度に最適化されており、優れたパフォーマンスを実現します。このオプションはデフォルトで利用でき、追加設定しない限り適用されます。
- JDBC データストアにデータを保存できます。これにより JDBC を使用して、選択したデータベースに接続します。このオプションでは、サーバーの設定ファイルに **datasources** サブシステムと **messaging-activemq** サブシステムを設定する必要があります。

10.2. デフォルトのファイルジャーナルを使用したメッセージングジャーナルの永続化

JBoss EAP メッセージングには、メッセージング向けに最適化された高パフォーマンスのファイルベースのジャーナルが備わっています。

JBoss EAP メッセージングジャーナルのファイルには設定可能なファイルサイズがあります。また、このジャーナルは、追加のみ可能です。このように単一の書き込み操作を有効にすることでパフォーマンスが向上します。これは、ディスク上の 1 組のファイルで構成されます。このファイルは、最初に固定サイズで事前作成され、パディングが書き込まれます。メッセージの追加、削除、更新などのサーバーオペレーションが実行されると、ジャーナルファイルが満杯になるまで、オペレーションの記録が追加されます。満杯になると、次のジャーナルファイルが使用されます。

すべてのデータが削除された場合、高度なガベージコレクションアルゴリズムにより、ジャーナルファイルを回収して再利用できるかどうかが決定されます。圧縮アルゴリズムにより、ジャーナルファイルから不要な領域が削除され、データが圧縮されます。

またジャーナルは、ローカルトランザクションと XA トランザクションの両方に完全に対応しています。

10.2.1. メッセージングジャーナルファイルシステムの実装

ジャーナルの大半は Java で記述されていますが、ファイルシステムとのやりとりは抽象化されており、さまざまなプラグ可能な実装が可能となります。JBoss EAP メッセージングには次の 2 つの実装が標準装備されています。

Java New I/O (NIO)

この実装は、ファイルシステムとのインターフェースに標準の 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できます。また、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。JBoss EAP 7 には Java 8 が必要となります。NIO は JBoss EAP がサポートするオペレーティングシステムで使用できます。

Linux Asynchronous IO (ASYNCIO)

これを実装する場合は、ネイティブコードラッパーを使用して Linux 非同期 IO ライブラリー (ASYNCIO) と通信をします。これを実装すると、明示的に同期をする必要がなくなります。通常 ASYNCIO は Java NIO よりもパフォーマンスが良好です。
使用中のジャーナルタイプを確認するには、以下の CLI 要求を発行します。

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

システムは、以下のいずれかの値を返します。

表10.1 ジャーナルタイプの戻り値

戻り値	説明
NONE	永続性が無効
NIO	Java NIO が使用中
ASYNCIO	**libaio** を使った AsyncIO が使用中
DATABASE	JDBC 永続性が使用中

以下のファイルシステムは、**libaio** ネイティブを使用している場合、Red Hat Enterprise Linux 6、Red Hat Enterprise Linux 7、および Red Hat Enterprise Linux 8 のみでテストされ、サポートされています。これらは他のオペレーティングシステムではテストされておらず、サポートされていません。

- EXT4
- XFS
- NFSv4
- GFS2

以下の表は、**libaio** ネイティブありとなしの両方で、サポートの有無をテストされた HA 共有ストアファイルシステムを示しています。

オペレーティングシステム	ファイルシステム	libaio ネイティブを使用したサポート (journal-type="ASYNCIO")	libaio ネイティブを使用しないサポート (journal-type="NIO")
Red Hat Enterprise Linux 6	NFSv4	あり	あり
Red Hat Enterprise Linux 7 以降	NFSv4	あり	あり
Red Hat Enterprise Linux 6	GFS2	あり	なし
Red Hat Enterprise Linux 7 以降	GFS2	あり	なし

10.2.2. 標準メッセージングジャーナルファイルシステムインスタンス

標準の JBoss EAP メッセージングコアサーバーでは、以下のジャーナルインスタンスを使用します。

バインディングジャーナル

このジャーナルは、サーバーにデプロイされた一連のキューと属性を含むバインディング関連のデータを格納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 ジャーナルであり、通常はメッセージジャーナルと比べて低スループットです。

このジャーナルのファイルには activemq-bindings というプレフィックスが付けられます。各ファイルには bindings 拡張子があります。ファイルサイズは 1048576 で、バインディングフォルダーにあります。

JMS ジャーナル

このジャーナルインスタンスは、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に対する JNDI バインディングなどの JMS 関連のデータをすべて格納します。
管理 API で作成されたすべての JMS リソースは、このジャーナルに永続化されます。設定ファイルで構成したリソースは一切、このジャーナルに永続化されません。JMS ジャーナルは、JMS が使用中の場合のみ作成されます。

このジャーナルのファイルには activemq-jms というプレフィックスが付けられます。各ファイルには jms 拡張子があります。ファイルサイズは 1048576 で、バインディングフォルダーにあります。

メッセージジャーナル

このジャーナルインスタンスは、メッセージ自体と duplicate-id キャッシュを含むすべてのメッセージ関連のデータを格納します。
デフォルトでは、JBoss EAP メッセージングは ASYNCIO ジャーナルの使用を試みます。ASYNCIO が利用できない場合 (たとえば、プラットフォームが正しいカーネルバージョンの Linux ではない、または ASYNCIO がインストールされていない場合など)、Java プラットフォームで利用可能な Java NIO を使用するように自動的にフォールバックします。

このジャーナルのファイルには activemq-data というプレフィックスが付けられます。各ファイルには amq 拡張子があります。ファイルサイズはデフォルトで 10485760 (設定可能) で、ジャーナルフォルダーにあります。

大きなメッセージは、JBoss EAP メッセージングによりメッセージジャーナル外部で永続化されます。詳細は、「大きなメッセージ」のセクションを参照してください。

メモリーが少なくなった場合に、JBoss EAP メッセージングが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可能です。詳細は、「ページング」のセクションを参照してください。

永続性が全く必要ない場合、JBoss EAP メッセージングは、「ゼロ永続化の JBoss EAP メッセージングの設定」のセクションで説明されているように、ストレージに一切のデータを永続化しないように設定することもできます。

10.2.3. バインディングジャーナルと JMS ジャーナルの設定

バインディングジャーナルと JMS ジャーナルは設定を共有しているため、以下の 1 つの管理 CLI コマンドで現在の両方の設定を読み取ることができます。出力も、デフォルト設定を強調表示するように組み込まれています。

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

{
    "outcome" => "success",
```

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

ジャーナルへの **path** のデフォルトは **activemq/bindings** であることに注意してください。以下の管理 CLI コマンドを使用すると **path** の場所を変更できます。

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

また、上記の出力の **relative-to** 属性にも注意してください。**relative-to** が使用された場合、**path** 属性の値は、**relative-to** で指定されたファイルパスに対する相対値とみなされます。デフォルト値は JBoss EAP プロパティーの **jboss.server.data.dir** です。スタンドアロンサーバーの場合、**jboss.server.data.dir** は **EAP_HOME/standalone/data** にあります。ドメインの場合、各サーバーには **EAP_HOME/domain /servers** の下に独自の **serverX/data/activemq** ディレクトリーがあります。以下の管理 CLI コマンドを使用すると **relative-to** の値を変更できます。

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

デフォルトでは、JBoss EAP は、バインディングディレクトリーが存在しなければ自動的に作成するように設定されています。以下の管理 CLI コマンドを使用して、この動作を切り替えます。

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

value を **true** に設定するとディレクトリーの自動作成が有効になります。**value** を **false** に設定すると無効になります。

10.2.4. メッセージジャーナルの場所の設定

以下の管理 CLI コマンドを使用すると、メッセージジャーナルの場所の情報を読み取ることができます。出力も、デフォルト設定を強調表示するように組み込まれています。

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

ジャーナルへの **path** のデフォルトは **activemq/journal** であることに注意してください。以下の管理 CLI コマンドを使用すると **path** の場所を変更できます。

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

注記

Red Hat では、最適なパフォーマンスを得るために、ディスクヘッドの動きを最小限に抑えられるように、ジャーナルを固有の物理ボリュームに配置することを推奨しています。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。

また、上記の出力の **relative-to** 属性にも注意してください。**relative-to** が使用された場合、**path** 属性の値は、**relative-to** で指定されたファイルパスに対する相対値とみなされます。デフォルト値は JBoss EAP プロパティーの **jboss.server.data.dir** です。スタンドアロンサーバーの場合、**jboss.server.data.dir** は **EAP_HOME/standalone/data** にあります。ドメインの場合、各サーバーには **EAP_HOME/domain /servers** の下に独自の **serverX/data/activemq** ディレクトリーがあります。以下の管理 CLI コマンドを使用すると **relative-to** の値を変更できます。

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

デフォルトでは、JBoss EAP は、ジャーナルディレクトリーが存在しなければ自動的に作成するように設定されています。以下の管理 CLI コマンドを使用して、この動作を切り替えます。

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

value を **true** に設定するとディレクトリーの自動作成が有効になります。**value** を **false** に設定すると無効になります。

10.2.5. メッセージジャーナル属性の設定

以下に挙げる属性はすべて、メッセージングサーバーの子プロパティーです。そのため、管理 CLI を使用して値の取得と設定をするコマンド構文は同じです。

指定された属性の現在の値を読み取る構文は以下のようになります。

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

属性の値を書き込む構文は、一致するパターンに従います。

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

- **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-file-open-timeout**
  この属性は、ジャーナルファイルを開く際のタイムアウト値を変更します。デフォルト値は **5** 秒です。

- **journal-buffer-timeout**
  フラッシュを必要とする書き込みで毎回フラッシュするのではなく、内部バッファーに保持し、満杯になるかタイムアウトになる (どちらか早い方) と内部バッファー全体をフラッシュし

ます。これは NIO と ASYNCIO の両方で使用され、フラッシュが必要な多くの同時書き込みがある場合は、システムがより適切な調整を行うことが可能となります。

このパラメーターは、バッファーがまだ満杯になっていない場合、バッファーをフラッシュするタイムアウトの制御を行います。通常、ASYNCIO は NIO より高速のフラッシュレートに対応できるため、システムは NIO と ASYNCIO のデフォルトを別々に保持します。NIO のデフォルト値は **3333333** ナノ秒 (300 回/秒) です。ASYNCIO のデフォルト値は **500000** ナノ秒 (2000 回/秒) です。

注記

タイムアウトを長くすると、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、デフォルトパラメーター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**
  ASYNCIO でタイムアウトになったバッファーのサイズ (バイト単位)。**journal-buffer-size** と **journal-file-size** は、**min-large-message-size** よりも高く設定する必要があります。そのように設定しなければ、メッセージはジャーナルには書き込まれません。詳細は、「大きいメッセージの設定」を参照してください。

- **journal-compact-min-files**
  ジャーナルが圧縮される前のファイルの最小数**journal-compact-min-files** を設定してはじめて圧縮アルゴリズムが起動されます。

  これを **0** に設定すると、圧縮機能は完全に無効になります。ただし、ジャーナルが無制限に増加する危険性があります。有効に使用してください。

  このパラメーターのデフォルトは **10** です。

- **journal-compact-percentage**
  圧縮を開始するしきい値。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。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。

  このパラメーターのデフォルトは **30** です。

- **journal-file-size**
  各ジャーナルファイルのサイズ (バイト単位)。このデフォルト値は **10485760** バイト (10MB) です。**journal-file-size** と **journal-buffer-size** は、**min-large-message-size** より大きく設定する必要があります。そのように設定しなければ、メッセージはジャーナルには書き込まれません。詳細は、「大きいメッセージの設定」を参照してください。

- **journal-max-io**
  書き込み要求は、実行するためにシステムへ送信される前にキューに置かれます。このパラメーターは、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。

  システムは、NIO または ASYNCIO のどちらかによって、パラメーターに別々のデフォルト値を保持します。NIO のデフォルト値は **1** で、ASYNCIO のデフォルト値は **500** です。

  値には制限があります。最大 ASYNCIO の合計値は、OS レベルで設定されている値 (/proc/sys/fs/aio-max-nr、通常 **65536**) より高くすることはできません。

- **journal-min-files**

ジャーナルが保持するファイルの最小数。JBoss EAP が起動され、初期メッセージデータがない場合、JBoss EAP では **journal-min-files** の数のファイルを事前に作成します。デフォルトは **2** です。

ジャーナルファイルを作成してパディングを書き込むことは、かなりコストのかかる操作で、ファイルで一杯になる実行時には、これを最小限に抑える必要があります。ファイルを事前に作成しておくと、ジャーナルが満杯になっても、ジャーナルを作成するために一時停止することなく、次のジャーナルをすぐに再開できます。

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。

- **journal-pool-files**
  再利用可能なジャーナルファイルの数。ActiveMQ では必要に応じてファイルを作成しますが、ファイルを回収する際には、この値まで減少させます。デフォルトは **-1** です。これは、無制限を意味します。

- **journal-sync-transactional**
  true に設定されると、JBoss EAP では、コミット、準備、ロールバックなどのトランザクション限界で、すべてのトランザクションデータが必ずディスクにフラッシュされるようになります。デフォルト値は **true** です。

- **journal-sync-non-transactional**
  true に設定されると、JBoss EAP では、送信や確認などのトランザクション以外のメッセージデータが、毎回必ずディスクにフラッシュされるようになります。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 を選択すると、JBoss EAP に Java NIO ジャーナルの使用を指示します。 **ASYNCIO** は、JBoss EAP に Linux 非同期 IO ジャーナルの使用を指示します。**ASYNCIO** を選択しても Linux が実行されていない場合、または libaio がインストールされていない場合、JBoss EAP は Java NIO ジャーナルを使用します。

10.2.6. ディスク書き込みキャッシュの無効の確認

オペレーティングシステムから **fsync()** を実行した場合でも、Java プログラム内から正しくデータを同期した場合でも、以下のような事象が発生します。

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。

ディスクが高価な非揮発性キャッシュやバッテリー駆動キャッシュを備えておらず、ディスクが冗長アレイ (RAID など) の一部ではなく、データの整合性を重視している場合は、ディスク書き込みキャッシュが無効になっていることを確認してください。

ディスク書き込みキャッシュを無効にすると、突然パフォーマンスが低下する可能性があります。デフォルト設定で書き込みキャッシュが有効な状態でディスクを使用している場合、気付かずにデータの整合性が損なわれる可能性があります。このため、書き込みキャッシュを無効にして、ディスクを実際に確実に機能させて高速で稼働させる方法を考える必要があります。

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検査または変更できます。

Windows の場合は、ディスクを右クリックし、**プロパティー**をクリックすると、設定を確認して変更できます。

10.2.7. libaio のインストール

Java NIO ジャーナルは極めて高性能ですが、Linux カーネル 2.6 以降を使用して JBoss EAP メッセージングを実行している場合は、Red Hat では、非常に最適な永続性パフォーマンスを得るために ASYNCIO ジャーナルを使用することを強く推奨します。

注記

JBoss EAP は、Red Hat Enterprise Linux のバージョン 6、7、または 8 にインストールされ、ext4、xfs、gfs2、nfs4 のいずれかのファイルシステムを使用している場合に限り、ASYNCIO に対応できます。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 ASYNCIO ジャーナルを使用することはできません。

ASYNCIO ジャーナルを使用するには、**libaio** のインストールが必要となります。インストールには以下のコマンドを使用します。

- Red Hat Enterprise Linux 6 および 7 の場合

```
yum install libaio
```

- Red Hat Enterprise Linux 8 の場合

```
dnf install libaio
```

警告

tmpfs ファイルシステム (たとえば **/tmp** ディレクトリーで使用) にメッセージングジャーナルを配置しないでください。ASYNCIO ジャーナルが tmpfs を使用している場合、JBoss EAP は起動できません。

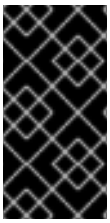
10.2.8. メッセージング用の NFS 共有ストアの設定

専用の共有のストアを使用して、データのレプリケーションに高可用性を確保する場合は、ライブサーバーとバックアップサーバーの両方を NFS クライアントの共有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。あるサーバーが NFS サーバーの共有ディレクトリーを使用し、別のサーバーが NFS クライアントの共有ディレクトリーを使用するように設定すると、ライブサーバーの起動時または稼働中にバックアップサーバーが認識できません。そのため正常に作動させるには、両方のサーバーで NFS クライアントの共有ディレクトリーを指定する必要があります。

NFS クライアントマウントでは、以下のオプションを設定することも必要です。

- **sync**: このオプションは、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**: このオプションは、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 要求を割り込みできるようにします。
- **noac**: このオプションは、属性キャッシングを無効にし、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。
- **soft**: このオプションは、エクスポートしたファイルシステムを提供するホスト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たずにエラーを報告するように指定します。
- **lookupcache=none**: このオプションはルックアップキャッシングを無効にします。
- **timeo=n**: NFS クライアントが NFS 要求を再試行するまで応答を待つ時間 (デシ秒 (10 分の 1 秒) 単位)。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。

高可用性を実現するために共有ファイルシステムを使用する方法の詳細は、本ガイドの「共有ストア」のセクションを参照してください。

10.3. JDBC データベースを使用したメッセージングジャーナルの永続化

デフォルトのファイルベースのジャーナルを使用するのではなく、JDBC を使用してメッセージを永続化し、データをデータベースにバインドするように、JBoss EAP 7 メッセージングを設定できます。この設定を行うには、まず **datasources** サブシステムの **datasource** 要素を設定し、次に **messaging-activemq** サブシステムの **server** 要素で **journal-datasource** 属性を定義して、データソースを使用する必要があります。**journal-datasource** 属性が存在すると、ファイルベースのジャーナルではなくジャーナルエントリーをデータベースに永続化することを、メッセージングサブシステムに通知します。**messaging -activemq** サブシステムの **server** リソースの **journal-database** 属性は、データベースとの通信に使用される SQL ダイアレクトを定義します。これはデータソースメタデータを使用して自動的に設定されます。

メッセージをファイルベースのジャーナルに永続化すると、大きいメッセージのサイズはディスクのサイズによってのみ制限されます。ただし、メッセージをデータベースに永続化すると、大きなメッセージのサイズは、そのデータベースの **BLOB** データ型の最大サイズに制限されます。

重要

JBoss EAP 7.4 では現在、Oracle 12c および IBM DB2 Enterprise データベースのみをサポートします。

10.3.1. メッセージングジャーナル JDBC 永続ストアの設定

以下の手順に従って、JDBC を使用してメッセージを永続化し、データをデータベースにバインドするように、JBoss EAP 7 メッセージングを設定します。

1. **messaging-activemq** サブシステムで使用するために、**datasources** サブシステムの **datasource** を設定します。データソースの作成および設定方法に関する詳細は、JBoss EAP **『設定ガイド』** の「データソース管理」を参照してください。

2. **messaging-activemq** サブシステムを設定して新しいデータソースを使用します。

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

これにより、サーバー設定ファイルの **messaging-activemq** サブシステムに以下の設定が作成されます。

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

JBoss EAP メッセージングが、データベースを使用してメッセージングデータを格納するように設定されました。

10.3.2. メッセージングジャーナルテーブル名の設定

JBoss EAP 7 メッセージングは、個別の JDBC テーブルを使用してバインディング情報、メッセージ、大容量メッセージ、ページング情報を格納します。これらのテーブルの名前は、サーバー設定ファイルの **messaging-activemq** サブシステムにある **server** リソースの **journal-bindings-table**、**journal-jms-bindings-table**、**journal-messages-table**、**journal-large-messages-table**、および **journal-page-store-table** 属性を使用して設定できます。

テーブル名にはいくつかの制限があります。

- JBoss EAP 7 メッセージングでは、**TABLE_NAME** + **GENERATED_ID** のパターンでページングテーブルの識別子を生成します。ここで **GENERATED_ID** は 20 文字まで可能です。Oracle Database 12c ではテーブル名の長さは最大 30 文字であるため、テーブル名を 10 文字に制限する必要があります。これを超えると、**ORA-00972: identifier is too long** というエラーが表示される可能性があり、ページングが機能しなくなります。

- Oracle Database 12c のスキーマオブジェクトネーミング規則に準拠しないテーブル名は、二重引用符で囲む必要があります。引用符付きの識別子は任意の文字で開始でき、任意の文字、句読点、スペースを含めることができます。ただし、引用符で囲まれていてもいなくても、識別子に二重引用符またはヌル文字 (\0) を含めることはできません。引用符で囲まれた識別子は、大文字と小文字が区別されることに注意してください。

- 複数の JBoss EAP サーバーインスタンスが同じデータベースを使用してメッセージを永続化し、データをバインドする場合、テーブル名はサーバーインスタンスごとに一意である必要があります。複数の JBoss EAP サーバーは同じテーブルにアクセスできません。

以下の例は、引用符付き識別子を使用して **journal-page-store-table** 名を設定する管理 CLI コマンドです。

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

これにより、サーバー設定ファイルの **messaging-activemq** サブシステムに以下の設定が作成されます。

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

10.3.3. 管理対象ドメインでのメッセージングジャーナルの設定

前述の「メッセージングジャーナルテーブル名の設定」で説明したように、JDBC を使用してメッセージを永続化してデータをデータベースにバインドする場合は複数の JBoss EAP サーバーが同じデータベーステーブルにアクセスすることはできません。管理対象ドメインの場合、サーバーグループのすべての JBoss EAP サーバーインスタンスは同じプロファイル設定を共有します。このため、メッセージングジャーナル名またはデータソースを設定する式を使用する必要があります。

すべてのサーバーが、同じデータベースを使用してメッセージングデータを格納するように設定されている場合、テーブル名はサーバーインスタンスごとに一意にする必要があります。以下の例は、名前が一意のノード識別子を含む式を使用して、サーバーグループの各サーバーに対して一意の **journal-page-store-table** テーブル名を作成する管理 CLI コマンドです。

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

各サーバーインスタンスが異なるデータベースにアクセスする場合、式を使用して各サーバーのメッセージング設定が異なるデータソースに接続するようにできます。以下の管理 CLI コマンドでは、**connection-url** の **DB_CONNECTION_URL** 環境変数を使用して、異なるデータソースに接続します。

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

10.3.4. メッセージングジャーナルネットワークのタイムアウトの設定

JDBC コネクションがリクエストに応答するまでのデーターベースの待機時間の最大値 (ミリ秒単位) を設定できます。これは、ネットワークが停止するか、JBoss EAP メッセージングとデータベースの間の接続が何らかの理由が閉じられているイベントで役に立ちます。こうしたイベントが発生すると、タイムアウトになるまでクライアントはブロックされます。

journal-jdbc-network-timeout 属性を更新してタイムアウトの設定をします。デフォルト値は **20000** ミリ秒 (**20** 秒) です。

以下の例は、**journal-jdbc-network-timeout** 属性値を **10000** ミリ秒 (**10** 秒) に設定する管理 CLI コマンドです。

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

10.3.5. メッセージング JDBC 永続ストアの HA の設定

JBoss EAP の **messaging-activemq** サブシステムでは、ブローカーがデータベースストアタイプで設定されている場合、JDBC HA 共有ストア機能をアクティブにします。その後、ブローカーは共有デー

タベーステーブルを使用して、ライブサーバーおよびバックアップサーバーが共有 JDBC ジャーナルストアのアクションを確実に連携させます。

以下の属性を使用すると、JDBC 永続ストアの HA の設定が可能となります。

- **journal-node-manager-store-table**: ノードマネージャーを格納する JDBC データベーステーブルの名前。
- **journal-jdbc-lock-expiration**: キープアライブすることなく JDBC ロックが有効とみなされる期間。デフォルトの値は **20000** ミリ秒です。
- **journal-jdbc-lock-renew-period**: JDBC ロックのキープアライブサービスの時間。デフォルトの値は **2000** ミリ秒です。

デフォルト値は、サーバーの **ha-policy** 属性と **journal-datasource** 属性の値に基づいて考慮されています。

後方互換性を保つために、各 Artemis 固有のシステムプロパティーを使用して値を指定することもできます。

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

これらのシステムプロパティーを設定すると、対応する属性のデフォルト値よりも優先されます。

10.4. メッセージングジャーナルの準備済みトランザクションの管理

以下の管理 CLI コマンドを使用して、メッセージングジャーナルの準備済みトランザクションを管理できます。

- 準備済みトランザクションのコミット:

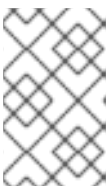
```
/subsystem=messaging-activemq/server=default:commit-prepared-transaction(transaction-as-base-64=XID)
```

- 準備済みトランザクションのロールバック:

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

- すべての準備済みトランザクションの詳細表示:

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



注記

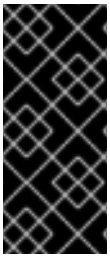
list-prepared-transaction-details-as-html 操作を使用した HTML 形式、または **list-prepared-transaction-details-as-json** 操作を使用した JSON 形式で、準備済みトランザクションの詳細を表示することもできます。

10.5. ゼロ永続化の JBOSS EAP メッセージングの設定

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。ゼロ永続化とは、バインドデータ、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の永続化ができないことです。

messaging-activemq サブシステムがゼロ永続化を実行するように設定するには、**persistence-enabled** パラメーターを **false** に設定します。

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



重要

永続性が無効になっていてページングが有効になっている場合、ページファイルは **paging-directory** 要素で指定された場所に格納したままになることに注意してください。**address-full-policy** 属性が **PAGE** に設定されていると、ページングが有効になります。完全なゼロ永続化が必要な場合は、**address-setting** 要素の **address-full-policy** 属性を **BLOCK**、**DROP**、または **FAIL** を使用するように設定してください。

10.6. ジャーナルデータのインポートとエクスポート

ジャーナルデータのインポートとエクスポートの詳細は、JBoss EAP 7 『移行ガイド』 を参照してください。

第11章 ページングの設定

11.1. ページングについて

JBoss EAP メッセージングは多数のメッセージキューをサポートし、各キューには何百万というメッセージが含まれています。JBoss EAP メッセージングサーバーは制限されたメモリーで実行されるため、一度にすべてのメッセージキューをメモリーに格納することは困難です。

ページングは、JBoss EAP メッセージングサーバーで使用されるメカニズムで、限られたメモリーに大きなメッセージキューを収容するために、必要に応じてメッセージを透過的にメモリー内外にページングします。

JBoss EAP メッセージングでは、特定のアドレスのメモリー内のメッセージサイズが最大設定メッセージサイズを超えると、メッセージのディスクへのページングが開始されます。

注記

JBoss EAP メッセージングページングはデフォルトで有効です。

11.2. ページファイル

複数のファイルにメッセージを格納するファイルシステムの各アドレスには、個別のフォルダーがあります。メッセージを格納するこれらのファイルは、ページファイルと呼ばれます。各ファイルには **page-size-bytes** 属性で設定された最大設定メッセージサイズに達するまでメッセージが収容されます。

システムは必要に応じてページファイルを移動して、ページのメッセージがすべてクライアントに受信されるとすぐにページファイルを削除します。

警告

パフォーマンス上の理由から、JBoss EAP メッセージングではページングされたメッセージをスキャンしません。したがって、メッセージをグループ化するか最新値を提供するように設定されたキューのページングは無効にする必要があります。また、メッセージの優先度付けとメッセージセレクターは、ページングが有効になっているキューでは期待したように動作しません。期待したように動作させるにはこの機能のページングを無効にする必要があります。

たとえば、キューからメッセージを読み込むメッセージセレクターをコンシューマーが持っている場合、そのセレクターと一致するメモリーのメッセージのみがコンシューマーに配信されます。コンシューマーがこれらのメッセージの配信を確認すると、新しいメッセージがページから分離され、メモリーに読み込まれます。ページファイルにおいてディスクのコンシューマーのセレクターと一致するメッセージがあっても、JBoss EAP メッセージングは、別のコンシューマーがメモリー内のメッセージを読み込み、空き領域を利用できるようになるまでメッセージをメモリーに読み込みしません。空き領域が使用できない場合、セレクターを使用しているコンシューマーは新しいメッセージを受信しない場合があります。

11.3. ページングディレクトリーの設定

以下の管理 CLI コマンドを使用すると、ページングディレクトリーの設定を読み込むことができます。この例では、出力にはデフォルト設定が表示されます。

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

paging-directory 設定要素は、ページファイルを格納するファイルシステムの場所を指定します。JBoss EAP は、このページングディレクトリーに各ページングアドレスのフォルダーを作成します。また、ページファイルはこれらのフォルダーに格納されます。デフォルトでは、このパスは **activemq/paging/** です。以下の管理 CLI コマンドを使用すると、パスの場所を変更できます。

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

また、上記の出力例の **relative-to** 属性にも注意してください。**relative-to** が指定されている場合、**path** 属性の値は **relative-to** 属性によって指定されるファイルパスの相対値となります。デフォルトでは、この値は JBoss EAP の **jboss.server.data.dir** プロパティーです。スタンドアロンサーバーの場合、**jboss.server.data.dir** は **EAP_HOME/standalone/data/** にあります。管理対象ドメインの場合、各サーバーは **EAP_HOME/domain /servers/** の下に独自の **serverX/data/activemq/** ディレクトリーを持ちます。以下の管理 CLI コマンドを使用すると **relative-to** の値を変更できます。

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

11.4. ページングモードの設定

アドレスに配信されるメッセージが設定サイズを超える場合、そのアドレスは**ページングモード**になります。

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** を設定すると、一致するアドレスにそれぞれ最大サイズを指定することになります。ただし、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。

page モードであっても、メモリー不足によるエラーが原因でサーバーがクラッシュする可能性があります。JBoss EAP メッセージングは、ディスク上の各ページファイルへの参照を維持します。ページファイルが非常に多くなると、JBoss EAP メッセージングでメモリーを使い切る可能性があります。このリスクを最小限に抑えるには、**page-size-bytes** 属性を適切な値に設定することが重要になります。JBoss EAP メッセージングサーバーのメモリーを **max-size-bytes** の宛先の数倍の 2 倍以上になるよう設定する必要があります。この設定を行わなければ、メモリー不足によるエラーが発生する可能性があります。

以下の管理 CLI コマンドを使用すると、アドレスの現在の最大サイズ (バイト単位) (**max-size-bytes**) を読み取ることができます。

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

以下の管理 CLI コマンドを使用すると、アドレスの最大サイズ (バイト単位) (**max-size-bytes**) を設定できます。

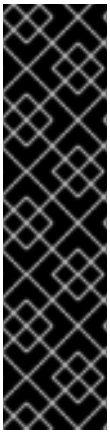
```
/subsystem=messaging-activemq/server=default/address-setting=ADDRESS_SETTING:write-attribute(name=max-size-bytes,value=MAX_SIZE)
```

アドレス設定で別のページング関連属性の値を読み取りまたは書き込みする場合は、同様の構文を使用してください。以下の表は、各属性の説明とデフォルト値を一覧表示しています。

以下の表には、アドレス設定のパラメーターをまとめています。

表11.1 アドレス設定のページング設定

要素	説明
address-full-policy	この属性の値は、ページングの決定に使用されます。有効な値を以下に示します。 **PAGE** ページングを有効にしてディスクに設定された制限を超えたメッセージのページングを可能にします。 **DROP** 設定した制限を超えるメッセージを通知せずにドロップします。 **FAIL** メッセージをドロップし、クライアントメッセージプロデューサーに例外を送信します。 **BLOCK** 設定された制限を超えたメッセージを送信すると、クライアントメッセージプロデューサーをブロックします。 デフォルトは **PAGE** です。
max-size-bytes	これは、ページングモードに入る前に、アドレスが持てる最大メモリーサイズを指定するのに使用されます。デフォルトは **10485760** です。
page-max-cache-size	システムは、ページングナビゲーション中に入出力を最適化するために、メモリー内に最大 **page-max-cache-size** のページファイルを保持します。デフォルトは **5** です。
page-size-bytes	これは、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を指定するために使用されます。デフォルトは **2097152** です。



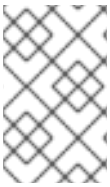
重要

デフォルトでは、アドレスが **max-size-bytes** に達すると、すべてのアドレスがページメッセージに設定されます。最大サイズに達した時点でメッセージのページングをやめる場合は、**address-full-policy** をそれぞれ **DROP**、**FAIL**、および **BLOCK** に設定することで、メッセージをドロップ、クライアント側で例外にしてメッセージをドロップ、それ以上のメッセージ送信からプロデューサーをブロックのいずれかを行うように設定できます。

宛先がメッセージのページングを開始した後に **address-full-policy** を **PAGE** から **BLOCK** に変更すると、ページ化されたメッセージをコンシューマーが消費できなくなることに注意してください。

複数のキューを持つアドレス

複数のキューがバインドされているアドレスへメッセージが転送されると、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。各キューではこのメッセージの元のコピーを参照するだけなので、元の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すると、はじめてメモリーが解放されます。



注記

単一のレイジーキュー/サブスクリプションによって、アドレス全体の入出力のパフォーマンスが軽減されることがあります。これは、ページングシステム上の追加ストレージから送信されたメッセージがすべてのキューにあるためです。

第12章 大きなメッセージの処理

JBoss EAP メッセージングは、クライアントまたはサーバーでメモリーのサイズが制限されていても、大きなメッセージに対応できます。大きなメッセージは、そのままにストリーミングしたり、効率的に転送するためにさらに圧縮したりできます。ユーザーはメッセージのボディーに **InputStream** を設定すると、大きなメッセージを送信できます。メッセージが送信されると、JBoss EAP メッセージングがこの **InputStream** を読み取り、データを断片化してサーバーへ送信します。

クライアントもサーバーも、大きなメッセージのボディー部分を完全な形でメモリーに保存しません。コンシューマーは、最初にボディーが空の大きなメッセージを受け取った後、メッセージに **OutputStream** を設定して、断片的にディスクファイルへストリーミングします。

警告

大きなメッセージを処理する場合、サーバーはメッセージのボディーと同じ方法でメッセージプロパティーを処理しません。たとえば、**journal-buffer-size** より大きい文字列に設定されたプロパティーを持つメッセージは、ジャーナルバッファーが満杯になるため、サーバーで処理できません。

12.1. 大きなメッセージのストリーミング

標準的な方法で大きなメッセージを送信する場合、メッセージを送信するのに必要なヒープサイズはメッセージのサイズの 4 倍以上になる可能性があります (つまり 1 GB のメッセージでは 4 GB のヒープメモリーが必要になる可能性があります)。このため、JBoss EAP メッセージングでは、**java.io.InputStream** と **java.io.OutputStream** クラスを使用してメッセージのボディーの設定をサポートしていますが、これに必要なメモリーはそれほど多くありません。入力ストリームがメッセージと出力ストリームの送信にそのまま使用され、メッセージの受信に使用されます。

メッセージの受信時に、出力ストリームを処理する方法は 2 つあります。

- **ClientMessage.saveToOutputStream(OutputStream out)** メソッドを使用して出力ストリームを復元している間はブロックできます。
- **ClientMessage.setOutputstream (OutputStream out)** メソッドを使用してメッセージをストリームに非同期に書き込むことができます。この方式では、メッセージが完全に受信されるまでコンシューマーをキープアライブする必要があります。

メッセージの送信に **java.io.InputStream** を実装し、メッセージの受信に **java.io.OutputStream** を実装している限り、ストリームの種類に関係なく (ファイル、JDBC Blob、または SocketInputStream など) 使用できます。

コア API を使用した大きなメッセージのストリーミング

以下の表に、オブジェクトプロパティーを使用して JMS で利用可能な **ClientMessage** クラスで使用できるメソッドを示します。

ClientMessage メソッド	説明	JMS 等価プロパティー

ClientMessage メソッド	説明	JMS 等価プロパティー
setBodyInputStream(InputStream)	送信時にメッセージボディーを読み取るために使用される **InputStream** を設定します。	**JMS_AMQ_InputStream**
setOutputStream(OutputStream)	メッセージのボディーを受信する **OutputStream** を設定します。この方法ではブロックしません。	**JMS_AMQ_OutputStream**
saveOutputStream(OutputStream)	メッセージのボディーを **OutputStream** に保存します。これは、コンテンツ全体が **OutputStream** に転送されるまでブロックされます。	**JMS_AMQ_SaveStream**

以下のコード例では、コアメッセージを受信する際に出力ストリームを設定します。

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

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

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

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

以下のコード例では、コアメッセージを送信する際に入力ストリームを設定します。

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

注記

2GiB を超えるメッセージの場合は、**_AMQ_LARGE_SIZE** メッセージプロパティーを使用する必要があります。**getBodySize()** メソッドが最大整数値に制限されているために無効な値を返すためです。

JMS での大きなメッセージのストリーミング

JMS を使用する場合、JBoss EAP メッセージングはオブジェクトプロパティーを設定して、コア API ストリーミングメソッドをマッピングします。**Message.setObjectProperty(String name, Object value)** メソッドを使用して入力ストリームと出力ストリームを設定します。

InputStream は、送信メッセージで **JMS_AMQ_InputStream** プロパティーを使用して設定されます。

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

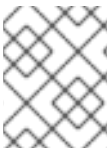
OutputStream は、ブロックの方法で受信されたメッセージの **JMS_AMQ_SaveStream** プロパティーを使用して設定されます。

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

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

JMS_AMQ_OutputStream プロパティーを使用すると、**OutputStream** をブロック以外の方法で設定することもできます。

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



注記

JMS を使用して大きなメッセージをストリーミングする場合、**StreamMessage** オブジェクトと **BytesMessage** オブジェクトのみがサポートされます。

12.2. 大きいメッセージの設定

12.2.1. 大きなメッセージの場所の設定

以下の管理 CLI コマンドを使用すると、大きなメッセージディレクトリーの設定を読み込むことができます。出力も、デフォルト設定を強調表示するように組み込まれています。

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

重要

最適なパフォーマンスを実現するには、大きなメッセージのディレクトリーを、メッセージジャーナルまたはページングディレクトリーとは別の物理ボリュームに格納することを推奨します。

large-messages-directory 設定要素は、大きなメッセージを格納するファイルシステムの場所を指定するために使用されます。デフォルトでは、パスは **activemq/largemessages** であることに注意してください。以下の管理 CLI コマンドを使用するとパスの場所を変更できます。

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

また、上記の出力の **relative-to** 属性にも注意してください。**relative-to** が指定された場合、path 属性

の値は、**relative-to** で指定されたファイルパスに対する相対値とみなされます。デフォルト値は JBoss EAP プロパティーの **jboss.server.data.dir** です。スタンドアロンサーバーの場合、**jboss.server.data.dir** は **EAP_HOME/standalone/data** にあります。ドメインの場合、各サーバーには **EAP_HOME/domain /servers** の下に独自の **serverX/data/activemq** ディレクトリーがあります。以下の管理 CLI コマンドを使用すると **relative-to** の値を変更できます。

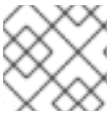
```
/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-attribute(name=relative-to,value=RELATIVE_LOCATION)
```

大きなメッセージのサイズの設定
管理 CLI を使用して、大きなメッセージの現在の設定を表示します。この設定は **connection-factory** 要素の一部であることに注意してください。たとえば、デフォルトの **RemoteConnectionFactory** の現在の設定を読み込むには、以下のコマンドを使用します。

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

同様の構文を使用して属性を設定します。

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



注記

min-large-message-size 属性の値はバイト単位となります。

大きなメッセージの圧縮の設定
高速かつ効率的に転送するために大きなメッセージを圧縮することができます。圧縮/解凍操作はすべて、クライアント側で処理されます。圧縮対象のメッセージが **min-large-message size** より小さい場合は、通常のメッセージとしてサーバーに送信されます。管理 CLI を使用してブール値プロパティー **compress-large-messages** を **true** に設定し、大きなメッセージを圧縮します。

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

12.2.2. コア API を使用した大きなメッセージのサイズの設定

クライアント側でコア API を使用している場合は、**setMinLargeMessageSize** メソッドを使用して、大きなメッセージの最小サイズを指定する必要があります。大きなメッセージの最小サイズ (**min-large-message-size**) はデフォルトで 100KB に設定されています。

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

locator.setMinLargeMessageSize(25 * 1024);

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

第13章 メッセージのスケジューリング

最初にメッセージを送信する未来の時間を指定できます。これは、メッセージを送信する前に、**_AMQ_SCHED_DELIVERY** スケジュール済みデリバリープロパティーを設定すると、指定できます。

指定する値は、メッセージが送信される時間 (ミリ秒単位) に対応する正の **long** にする必要があります。以下は、JMS API を使用してスケジュールされたメッセージを送信する例です。

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

...

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

スケジュールされたメッセージは、メッセージを送信する前に **_AMQ_SCHED_DELIVERY** プロパティーを設定し、コア API を使用して送信することもできます。

第14章 一時キューとランタイムキュー

クライアントがリクエストを送信して応答を待つリクエスト応答パターンを設計する場合、クライアントの各ランタイムインスタンスが応答専用キューを必要とするかどうか、ランタイムインスタンスが共有キューにアクセスできるかどうかを考慮し、適切な属性に基づいて具体的な応答メッセージを選択する必要があります。

複数のキューが必要となる場合、クライアントには動的にキューを作成する機能が必要になります。JMS は、一時キューの概念を使ってこの機能を提供します。**Session** によってリクエストに **TemporaryQueue** が作成されます。**Connection** の有効期間中 (たとえば接続を閉じるまで、または一時キューが削除されるまで) 存在します。つまり、一時キューは特定のセッションで作成されますが、同じ接続で作成された他のセッションで再利用できます。

共有キューと応答独自の一時キューを使用するトレードオフは、アクティブなクライアントインスタンスの見込み数に影響されます。共有キューアプローチでは、一部のプロバイダー固有のしきい値で、キューへのアクセスの競合が懸念される可能性があります。これは、ランタイム時にキューストレージを作成するプロバイダーによるオーバーヘッドの増加と、大量になる可能性のある一時キューを収容するマシンメモリーへの影響を比較する必要があります。

以下の例では、起動時に各クライアントに一時キューとコンシューマーを作成します。各メッセージの **JMSReplyTo** プロパティーを一時キューに設定し、次に各メッセージの相関 ID を設定してリクエストメッセージを応答メッセージと関連付けます。これにより、リクエストごとにコンシューマーを作成して閉じるというオーバーヘッド (非常に大きい) が回避されます。同じプロデューサーとコンシューマーは、多くのスレッドで共有またはプールできます。セッションの終了時に受信したもののまだ確認されていないメッセージはすべて保持され、コンシューマーが次にキューをアクセスすると再送されます。

例: 一時キューのコード

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

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

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

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

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

同様に、一時トピックは **Session.createTemporaryTopic()** メソッドを使用して作成されます。

第15章 フィルター式とメッセージセレクター

JBoss EAP の **messaging-activemq** サブシステムでは、SQL 92 表現構文のサブセットに基づいた強力なフィルター言語が提供されます。

これは JMS セレクターで使用される構文と同じですが、事前定義識別子は異なります。JMS セレクター構文に関するドキュメントは、javax.jms.Message のインターフェース Javadoc を参照してください。

filter 属性は、設定内の複数の場所に置かれます。

- 事前定義されたキュー。キューを事前定義する場合、キューに対してフィルター式で定義できます。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。次の設定スニペットは、フィルターが含まれるキュー定義を示しています。

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

  管理 CLI でセレクターを使用してキューを作成するには、以下のようなコマンドを使用します。

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

- コアブリッジはオプションのフィルター式で定義でき、一致するメッセージのみがブリッジされます。以下は、**messaging-activemq** サブシステムにフィルターを持つブリッジが含まれる設定ファイルの例になります。

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

- 迂回はオプションのフィルター式で定義でき、一致するメッセージのみが迂回されます。詳細は、「迂回」を参照してください。以下のスニペット例は、フィルターを使用した迂回を示しています。

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

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

JMS セレクター式と JBoss EAP メッセージングのコアフィルター式にはいくつかの違いがあります。JMS セレクター式は JMS メッセージで動作しますが、JBoss EAP メッセージングのコアフィルター式はコアメッセージで動作します。

以下の識別子をコアフィルター式で使用することで、コアメッセージの属性を参照できます。

- **AMQPriority**。メッセージの優先度を参照します。メッセージの優先度は、**0 - 9** の有効な値を持つ整数です。**0** の優先度が最も低く、**9** が最も高くなります。たとえば、**AMQPriority** = 3 AND animal = 'aardvark' のようになります。
- **AMQExpiration**。メッセージの有効期限を参照します。値は長整数です。
- **AMQDurable**。メッセージが永続的であるどうかを参照します。値は、**DURABLE** または **NON_DURABLE** という有効な値を持つ文字列です。
- **AMQTimestamp**。メッセージが作成された時点のタイムスタンプです。値は長整数です。
- **AMQSize**。メッセージのサイズ (バイト単位)。値は整数です。

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。

第16章 メッセージ有効期限の設定

送信されたメッセージが指定された期間後にコンシューマーに配信されない場合に、そのメッセージがサーバーで期限切れになるように設定できます。これらの期限切れのメッセージは、さらに調査をするために後で使用できます。

コア API を使用したメッセージ有効期限の設定

コア API を使用すると、**setExpiration** メソッドを使用してメッセージの有効期限を設定できます。

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

JMS を使用したメッセージ有効期限の設定

メッセージの送信時に、JMS **MessageProducer** が使用する持続時間を設定できます。この値は、**setTimeToLive** メソッドを使用してミリ秒単位で指定します。

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

プロデューサーの **send** メソッドで持続時間を設定すると、メッセージの有効期限をメッセージごとに指定することもできます。

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

期限切れアドレスから消費される有効期限切れのメッセージには、以下のプロパティーがあります。

- **_AMQ_ORIG_ADDRESS**
  期限切れメッセージの元のアドレスが含まれる **String** プロパティー。
- **_AMQ_ACTUAL_EXPIRY**
  期限切れのメッセージの実際の有効期間が含まれる **Long** プロパティー。

16.1. 期限切れアドレス

期限切れアドレスを設定して、有効期限切れのメッセージを送信する場所を指定できます。メッセージの有効期限が切れて期限切れアドレスが指定されていない場合、メッセージはキューから削除されドロップされます。

管理 CLI を使用して、**address-setting** の **expiry-address** を設定できます。以下の例では、**jms.queue.exampleQueue** キューの有効期限切れのメッセージが **jms.queue.expiryQueue** 期限切れアドレスに送信されます。

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

16.2. 期限切れリーパースレッド

リーパースレッドは定期的にキューを検査して、メッセージの有効期限が切れているかどうかを確認します。管理 CLI を使用して、リーパースレッドのスキャン期間とスレッド優先度を設定できます。

期限切れリーパースレッドのスキャン期間を設定します。スキャン期間は、有効期限切れのメッセージを検出するためにキューをスキャンする頻度 (ミリ秒単位) です。デフォルトは **30000** です。このパラメーターを **-1** に設定するとリーパースレッドを無効にできます。

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

期限切れリーパースレッドのスレッド優先度を設定します。設定可能な値は **0** から **9** までで、**9** の優先度が最も高くなります。デフォルトは **3** です。

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

第17章 遅延再配信の設定

アドレスへの遅延再配信は、**address-setting** 設定要素の **redelivery-delay** 属性で定義されます。再配信の遅延が指定されると、JBoss EAP はこの遅延期間待機した後にメッセージを再配信します。**redelivery-delay** を **0** に設定すると、再配信の遅延はありません。特定の **address-setting** について **redelivery-delay** の現在値を取得するには、以下の例のような管理 CLI コマンドを使用します。

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

次の表には、メッセージの再配信の設定に使用できる **address-setting** の設定属性が記載されています。以下の例のような管理 CLI コマンドを使用して、特定の属性に値を設定します。

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

表17.1 アドレス設定の配信関連の属性

属性	説明
max-delivery-attempts	**dead-letter-address** に送信する前にキャンセルメッセージを再配信できる回数を定義します。デフォルトは **10** です。
max-redelivery-delay	**redelivery-delay** の最大値 (ミリ秒単位)。**max-redelivery-delay パラメーターを設定して、遅延** が大きすぎるのを防ぐことができます。デフォルトは **redelivery-delay * 10** です。
redelivery-delay	キャンセルメッセージを再配信するまでの待ち時間を定義します (ミリ秒単位)。デフォルトは **0** です。
redelivery-multiplier	**redelivery-delay** パラメーターに適用する乗数。メッセージを再配信するたびに、遅延期間は以前の**redelivery-delay** * **redelivery-multiplier** と等しくなります。デフォルトは **1.0** です。

address-setting の設定の詳細は、「アドレス設定」を参照してください。

第18章 デッドレターアドレスの設定

デッドレターアドレスは、**messaging-activemq** サブシステム設定の **address-setting** 要素で定義されます。特定の **address-setting** の現在の設定を確認するには、以下の例のような管理 CLI コマンドを使用します。

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

dead-letter-address が指定されていないと、**max-delivery-attempts** で指定される回数配信を試みた後、メッセージが削除されます。デフォルトでは 10 回メッセージの配信を試みます。**max-delivery-attempts** を **-1** に設定すると、再試行が無限に試行されます。以下の管理 CLI コマンドの例は、特定の **address-setting** に **dead-letter-address** 属性および **max-delivery-attempts** 属性を設定する方法を示しています。

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

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

たとえば、一致するアドレスのセットに対してグローバルにデッドレターを設定することができ、特定のアドレスの **max-delivery-attempts** を **-1** に設定し、このアドレスのみ再配信が無限に行われるようにすることが可能です。アドレスワイルドカードを使用して、アドレスのセットにデッドレターを設定することもできます。

address-setting の作成と設定に関する詳細は、「アドレス設定」を参照してください。

第19章 フロー制御

フロー制御は、クライアントとサーバー間のメッセージングデータのフローを制限するために使用されます。これはクライアントまたはサーバーがデータで一杯にならないようにより行われます。コンシューマー側とプロデューサー側の両方からデータのフローを管理できます。

19.1. コンシューマーフロー制御

JBoss EAP メッセージングには、コンシューマーのために事前にフェッチするデータ量を定義する設定とコンシューマーがメッセージを消費できる速度を制御する設定が含まれています。

ウィンドウベースのフロー制御
JBoss EAP メッセージングでは、メッセージを各コンシューマーのバッファーに事前フェッチします。バッファーサイズは **connection-factory** の **consumer-window-size** 属性で決定されます。次の設定例は、**consumer-window-size** 属性を明示的に設定した **connection-factory** を示しています。

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

管理 CLI を使用して、特定の **connection-factory** の **consumer-window-size** 属性の値を読み書きします。以下の例は、**InVmConnectionFactory** 接続ファクトリーを使用して実行する方法を示しています。これはサーバーと同じ仮想マシンに存在するコンシューマー (たとえば、ローカルの **MessageDrivenBean**) のデフォルトです。

- 管理 CLI から **InVmConnectionFactory** の **consumer-window-size** 属性を読み取る

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

- 管理 CLI から **consumer-window-size** 属性を書き込む

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

consumer-window-size の値は整数である必要があります。以下の表に記載されているように、特別な意味を持つ値もあります。

表19.1 consumer-window-size の値

Value	説明
n	バッファーサイズを **n** バイトに設定するために使用される整数値。デフォルトは **1048576** で、ほとんどの場合はこれで十分です。デフォルト値が適切でない場合は、ベンチマークをすると、ウィンドウサイズの最適な値が見つかるようになります。
0	バッファーをオフにします。これはコンシューマーが遅い場合に役に立つもので、複数のコンシューマーに確定的に分散することができます。

Value	説明
-1	無制限バッファーを作成します。これにより、メッセージを受信するとすぐにプルして処理する非常に高速のコンシューマーを促進することができます。

警告

consumer-window-size を **-1** に設定すると、コンシューマーがメッセージを受信時にすぐに処理できない場合に、クライアントのメモリーをオーバーフローさせることができます。

コア API を使用している場合、**setConsumerWindowSize()** メソッドを使用して **ServerLocator** からコンシューマーウインドウサイズを設定できます。

JMS を使用している場合、クライアントはインスタンス化された **ConnectionFactory** の **setConsumerWindowSize()** メソッドを使用してコンシューマーウインドウのサイズを指定できます。

レート制限フロー制御

JBoss EAP メッセージングは、メッセージを消費するレートを 1 秒単位で制限できます。これはスロットルと呼ばれるフロー制御メソッドです。適切な **connection-factory** の **consumer-max-rate** 属性を使用して、指定した速度よりも早くコンシューマーがメッセージを消費しないようにします。

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

デフォルト値は **-1** で、レート制限フロー制御を無効にします。

consumer-max-rate 属性の読み取りおよび書き込みには、管理 CLI を使用することが推奨されます。以下の例は、**InVmConnectionFactory** 接続ファクトリーを使用して実行する方法を示しています。これはサーバーと同じ仮想マシンに存在するコンシューマー (たとえば、ローカルの **MessageDrivenBean**) のデフォルトです。

- 管理 CLI を使用して **consumer-max-rate** 属性を読み取る:

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

- 管理 CLI を使用して **consumer-max-rate** 属性を書き込む:

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

JMS を使用している場合、インスタンス化された **ConnectionFactory** の setConsumerMaxRate(int consumerMaxRate) メソッドを使用して最大レートサイズを設定できます。

コア API を使用している場合、速度は **ServerLocator.setConsumerMaxRate(int consumerMaxRate)** メソッドで設定できます。

19.2. プロデューサーフロー制御

JBoss EAP メッセージングは、サーバーの処理能力を超えるメッセージを受信しないように、クライアントから送信されるデータの量を制限することもできます。

ウィンドウベースのフロー制御
JBoss EAP メッセージングは、クレジットの交換を使用して、メッセージプロデューサーを制御します。プロデューサーは、これを行うのに十分なクレジットがある限り、メッセージをアドレスに送信することができます。メッセージを送信するのに必要なクレジットの量は、メッセージのサイズで決定されます。プロデューサーのクレジットが欠乏している場合、プロデューサーはサーバーからさらに多くのクレジットを要求します。サーバーの設定の中では、プロデューサーが一度に要求できるクレジットの量は **producer-window-size** と呼ばれ、次の **connection-factory** 要素の属性です。

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

ウィンドウサイズで一度に送信できるバイト量が決定されるため、リモート接続によってサーバーに過重な負荷がかかるのを防ぎます。

管理 CLI を使用して、特定の接続ファクトリーの **producer-window-size** 属性を読み書きします。以下の例では **RemoteConnectionFactory** を使用しています。これはデフォルト設定に含まれており、リモートクライアントでの使用を目的としています。

- 管理 CLI を使用して **producer-window-size** 属性を読み取る:

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

- 管理 CLI を使用して **producer-window-size** 属性を書き込む:

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

JMS を使用している場合、クライアントは **ConnectionFactory** の **setProducerWindowSize(int producerWindowSize)** メソッドを呼び出してウィンドウサイズを直接設定できます。

コア API を使用している場合、**ServerLocator** の **setProducerWindowSize(int producerWindowSize)** メソッドを使用してウィンドウサイズを設定できます。

プロデューサーウインドウベースのフロー制御のブロック
通常、メッセージングサーバーはリクエストされたのと同じ数のクレジットを提供します。ただし、サーバーから送信されるクレジットの数を制限することは可能で、これにより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してメモリー不足に陥るのを防止できます。

たとえば、**myqueue** という JMS キューがあり、最大メモリーサイズを 10MB に設定すると、サーバー

によってキュー内のメッセージ数が制限され、サイズが 10MB を超えることはなくなります。アドレスが満杯になると、十分な領域がアドレス上で解放されるまで、プロデューサーがクライアント側でブロックされます。

注記

プロデューサーフロー制御のブロックは、ページング (プロデューサーをブロックするのではなく、メッセージをストレージにページングする) の代替アプローチです。詳細は「ページングについて」を参照してください。

address-setting 設定要素には、プロデューサーフロー制御のブロックを管理する設定が含まれます。**address-setting** は、そのアドレスに登録されているすべてのキューに一連の設定を適用するために使用されます。実施方法の詳細は、「アドレス設定の設定」を参照してください。

プロデューサーフロー制御をブロックするのに必要な **address-setting** にはそれぞれ **max-size-bytes** 属性の値を含める必要があります。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が **max-size-bytes** を超えることはできません。JMS トピックの場合、トピックのすべてのサブスクリプションの合計メモリーが **max-size-bytes** を超えることができないことを意味します。

また、**address-full-policy** 属性を **BLOCK** に設定することも必要です。これによってプロデューサーは **max-size-bytes** に達するとブロックする必要があることを認識します。以下の例は、両方の属性セットを持つ **address-setting** です。

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

上記の例では、JMS キューの「myqueue」の最大サイズを **100000** バイトに設定します。プロデューサーは、最大サイズに達した時点で、そのアドレスへの送信がブロックされます。

以下の例のように、管理 CLI を使用してこれらの属性を設定します。

- 指定された **address-setting** の **max-size-bytes** を設定する

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

- 指定された **address-setting** の **address-full-policy** を設定する

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

レート制限フロー制御

JBoss EAP メッセージングでは、以下の例のように、プロデューサーが使用する **connection-factory** の **producer-max-rate** を指定すると、プロデューサーが 1 秒あたりに送信できるメッセージの数を制限します。

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

デフォルト値は **-1** で、レート制限フロー制御を無効にします。

管理 CLI を使用して **producer-max-rate** の値を読み書きします。以下の例では **RemoteConnectionFactory** を使用しています。これはデフォルト設定に含まれており、リモートクライアントでの使用を目的としています。

- **producer-max-rate** 属性の値を読み取る:

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

- **producer-max-rate** 属性の値を書き込む:

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

コア API を使用する場合は、**ServerLocator.setProducerMaxRate(int producerMaxRate)** メソッドを使用してレートを設定します。

JNDI を使用して接続ファクトリーのインスタンス化と検索を行う場合、インスタンス化された接続ファクトリーの **setProducerMaxRate(int producerMaxRate)** メソッドを使用して最大レートをクライアントに設定できます。

第20章 事前承認の設定

JMS は次の 3 つの承認モードを指定します。

- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE

場合によっては、障害発生時にメッセージを失い、メッセージをクライアントに配信する前にサーバーでメッセージを承認することが適切なことがあります。この特別なモードは JBoss EAP メッセージングによってサポートされ、pre-acknowledge モードと呼ばれます。

配信前にサーバーで事前承認する欠点は、サーバーでメッセージを承認してからクライアントへ配信されるまでにサーバーのシステムがクラッシュした場合に、メッセージが失われることです。この場合、システムの再起動時にメッセージが失われ、復元されません。

メッセージングのケースに応じて、事前承認モードでは、メッセージが失われることがある代わりに、余分なネットワークトラフィックと CPU 使用率を回避できます。

事前承認の使用例は、株価更新メッセージです。これらのメッセージでは、クラッシュ時にメッセージが失われる適切な場合があります。これは、次の株価更新メッセージがすぐに到達し、前の株価を上書きするためです。

注記

事前承認モードを使用する場合は、消費されるメッセージのトランザクションセマンティクスが失われます。これは、メッセージが、トランザクションをコミットしたときではなく、サーバーで最初に承認されるためです。

20.1. サーバーの設定

以下のように管理 CLI を使用して **pre-acknowledge** 属性を **true** に設定すると、接続ファクトリーを事前承認モードを使用するように設定できます。

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

20.2. クライアントの設定

事前承認モードは、**jndi.properties** ファイルなどでクライアントの JNDI コンテキスト環境で設定することが可能です。

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

または、JMS API を使って事前確認モードを使用するには、**ActiveMQSession.PRE_ACKNOWLEDGE** 定数で JMS セッションを作成します。

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

第21章 インターセプター

JBoss EAP メッセージングは、サーバーを出入りするパケットをインターセプトするインターセプターをサポートします。着信インターセプターと発信インターセプターは、それぞれサーバーに入ったり出たりするたびに呼び出されます。パケットの監査やフィルタリングなどのカスタムコードを実行することができます。インターセプターはインターセプトするパケットを修正できます。これによりインターセプターは強力になりますが、危険にさらされる可能性もあります。

21.1. インターセプターの実装

インターセプターでは次のインターセプターインターフェースを実装する必要があります。

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

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

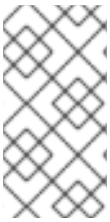
返されるブール値は重要です。

- true が返されると、通常どおりプロセスが続行されます。
- false が返されると、プロセスは中止されます。また、他のインターセプターは呼び出されず、パケットはそれ以上サーバーで処理されません。

インターセプタークラスをモジュールとして JBoss EAP に追加する必要があります。詳細は、JBoss EAP **『設定ガイド』** の「 カスタムモジュールの作成」を 参照してください。

21.2. インターセプターの設定

JBoss EAP にカスタムモジュールとしてモジュールを追加した後、着信インターセプターと発信インターセプターは共に、管理 CLI を使用してメッセージングサブシステムの設定に追加されます。



注記

新しいインターセプター設定を許可する前に、JBoss EAP を**管理者専用**モードで起動する必要があります。詳細は、JBoss EAP **『設定ガイド』** の「 JBoss EAP の管理専用モードでの実行 」を参照してください。新しい設定の処理が完了してから、通常モードでサーバーを再起動します。

以下の管理 CLI コマンドの例にしたがって、各インターセプターを追加する必要があります。この例では、各インターセプターをカスタムモジュールとして JBoss EAP に追加済みであると仮定しています。

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

発信インターセプターを追加する場合は、以下の例に示すものと同様の構文に従います。

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

第22章 メッセージのグループ化

メッセージグループは、共通した特定の特性を持つメッセージのグループです。

- メッセージグループのすべてのメッセージは、共通のグループ ID でグループ化されます。そのため、これらのメッセージは共通のグループプロパティーで特定できます。
- メッセージグループのメッセージはすべて、キューのカスタマーの数に関係なく同じコンシューマーによって順次処理され、消費されます。そのため、一意のグループ ID を持つ特定のメッセージグループは、常にそのメッセージグループを開く 1 つのコンシューマーによって処理されます。コンシューマーがメッセージグループを閉じると、メッセージグループ全体がキューの別のコンシューマーに移動されます。

メッセージグループは、特定のプロパティーの値 (グループ ID など) を持つメッセージが単一コンシューマーによって順次処理される必要がある場合に特に便利です。

重要

キューのページングが有効になっていると、メッセージのグループ化は期待どおりに動作しません。メッセージをグループ化するためにキューを設定する前に、ページングを無効にしてください。

メッセージングサーバーのクラスター内におけるメッセージグループ化の設定の詳細については、Part III、「複数のメッセージングシステムの設定」の「クラスター化されたメッセージのグループ化」を参照してください。

22.1. コア API を使用したメッセージグループの設定

_AMQ_GROUP_ID プロパティーは、クライアント側でコア API を使用してメッセージグループを特定するために使用されます。ランダムな一意のメッセージグループ識別子を選択するために、**SessionFactory** で **auto-group** プロパティーを **true** に設定することもできます。

22.2. JMS を使用したメッセージグループの設定

JMSXGroupID プロパティーは、Java Messaging Service (JMS) クライアントのメッセージグループを特定するために使用されます。別のメッセージを持つメッセージグループを 1 つのコンシューマーに送信する場合は、異なるメッセージに対して同じ **JMSXGroupID** を設定することができます。

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

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

別の方法としては、クライアントによって使用される **connection-factory** の **auto-group** または **group-id** のいずれかの属性を使用することができます。

auto-group を **true** に設定すると、その **connection-factory** を介して送信されたすべてのメッセージにランダムな一意のメッセージグループ識別子を使用するようになります。管理 CLI を使用して **auto-group** 属性を設定できます。

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

その接続ファクトリーを介して送信されたすべてのメッセージに対して、**group-id** 属性によって **JMSXGroupID** プロパティーが指定された値に設定されます。接続ファクトリーに特定の **group-id** を設定するには、管理 CLI を使用します。

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

第23章 迂回

迂回は、JBoss EAP メッセージングで設定されるオブジェクトです。メッセージをあるアドレスから別のアドレスに迂回する場合に役に立ちます。迂回は、以下のタイプに分類できます。

特別な迂回

メッセージは新しいアドレスにのみ迂回され、古いアドレスには送信されません。

特別でない迂回

メッセージは古いアドレスに送信され、そのコピーが新しいアドレスにも送信されます。特別でない迂回はメッセージのフローを分割するために使用できます。

迂回により、メッセージは同じサーバー内のアドレスにのみ迂回されます。別のサーバーのアドレスにメッセージを迂回する必要がある場合、一般的なパターンでは、メッセージをローカルストアと転送キューに迂回し、そのキューから消費して別のサーバーのアドレスに転送するブリッジをセットアップします。

そのため、迂回は非常に高度な概念です。ブリッジと組み合わせると、迂回は興味深く複雑なルーティングを作成するのに使用することができます。サーバーの迂回のセットは、一種のメッセージのルーティングテーブルと考えることができます。迂回とブリッジを組み合わせると、複数の地理的に分散したサーバー間に、信頼性の高いルーティング接続の分散ネットワークを作成し、グローバルなメッセージジング網が作成できます。ブリッジの使用方法についての詳細は、「コアブリッジの設定」を参照してください。

迂回は、**トランスフォーマー**とオプションのメッセージフィルターを適用するように設定できます。オプションのメッセージフィルターは、指定されたフィルターに一致するメッセージのみを迂回するのに役立ちます。フィルターの詳細は、「フィルター式とメッセージセレクター」を参照してください。

トランスフォーマーは、メッセージを別のフォームに変換するために使用されます。トランスフォーマーが指定されている場合、迂回されたメッセージはすべて**トランスフォーマー**によって変換されます。すべてのトランスフォーマーは **org.apache.activemq.artemis.core.server.cluster.Transformer** インターフェースを実装する必要があります。

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

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

JBoss EAP メッセージングを有効にしてトランスフォーマー実装のインスタンスをインスタンス化できるようにするには、JBoss EAP モジュールにその実装を追加し、そのモジュールをエクスポートされた依存関係として **org.apache.activemq.artemis** モジュールに追加する必要があります。カスタムモジュールの 作成方法の詳細は、JBoss EAP **『設定ガイド』** の「カスタムモジュールの作成」を参照してください。**org.apache.activemq.artemis** モジュールに依存関係を追加するには、以下の例のように **EAP_HOME/modules/system/layers/base/org/apache/activemq/artemis/main/module.xml** ファイルをテキストエディターで開いて **<module>** を **<dependencies>** のリストに追加します。

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

```
      <module name="YOUR_MODULE_NAME" export="true"/>
    </dependencies>
</module>
```

23.1. 特別な迂回

以下の例は、設定で見られる可能性がある特別な迂回を示しています。

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

この例では、**prices-divert** と呼ばれる迂回が、アドレス **jms.topic.priceUpdates** に送信されたメッセージを迂回するように設定されています。これは **priceUpdates** という JMS トピックに送信されたすべてのメッセージを、**priceForwarding** というローカル JMS キューに対応する別のローカルアドレス **jms.queue.priceForwarding** にマッピングしています。

また、値が **New York** のメッセージプロパティー **office** を持つメッセージのみを迂回するようにメッセージフィルター文字列を指定します。その他のメッセージはすべて、今までどおり通常のアドレスへルーティングされます。フィルター文字列はオプションです。指定されていない場合は、すべてのメッセージが一致したとみなされます。

トランスフォーマークラスが指定されていることに注意してください。この例では、トランスフォーマーは迂回が発生した時間を記録するヘッダーを追加するだけです。

この例では、実際にメッセージをローカルストアと転送キューに迂回しています。これは、メッセージを別のサーバーのアドレスに転送するブリッジで設定されています。詳細は、「コアブリッジの設定」を参照してください。

23.2. 特別でない迂回

以下は、特別でない迂回の例です。特別でない迂回は、オプションのフィルターとトランスフォーマーを使用して特別な迂回と同じ方法で設定できます。

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

上記の迂回は、アドレス **jms.queue.orders** に送信されたすべてのメッセージのコピーを取ります。そして **orders** という JMS キューにマッピングされ、**SpyTopic** という JMS トピックに対応する **jms.topic.SpyTopic** というローカルアドレスに送信されます。

迂回の作成

管理 CLI を使用して、必要な迂回のタイプを作成します。

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

デフォルトでは、特別でない迂回が作成されます。特別な迂回を作成するには、**exclusive** 属性を使用します。

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

以下の表は、迂回の属性とその説明を示しています。この情報は、管理 CLI で以下のコマンドを使用して表示できます。

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

属性	説明
divert-address	迂回元のアドレス。必須。
exclusive	迂回が特別であるかどうか。メッセージが新しいアドレスに迂回され、古いアドレスにはまったく送信されないことを意味します。デフォルトは false です。
filter	オプションのフィルター文字列。指定すると、フィルター式に一致するメッセージのみが迂回されます。
forwarding-address	迂回先のアドレス。必須。
routing-name	迂回のルーティング名。
transformer-class-name	迂回前にメッセージの本文またはプロパティーを変更するために使用するクラスの名前。

第24章 スレッド管理

JBoss EAP の各メッセージングサーバーは、一般的な使用のために単一のスレッドプールを維持し、スケジュール使用のためにスケジュールスレッドプールを維持します。Java のスケジュールスレッドプールは標準のスレッドプールを使用するように設定できません。それ以外は、スケジュールされたアクティビティーとスケジュールされていないアクティビティーの両方に単一のスレッドプールを使用できます。

JBoss EAP は新しい非ブロッキング NIO を使用することに注意してください。デフォルトでは、JBoss EAP のメッセージングは、受信パケットの処理に、**.getRuntime().availableProcessors()** によって報告されるコア (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には、トランスポート設定で **nio-remoting-threads** パラメーターを指定してスレッド数を設定します。詳細は、「メッセージングトランスポートの設定」を参照してください。

24.1. サーバースケジュールスレッドプール

サーバースケジュールスレッドプールは、定期的に実行する必要がある、または遅れて実行する必要があるサーバーサイドの多くのアクティビティーに使用されます。内部的には、java.util.concurrent.ScheduledThreadPoolExecutor インスタンスにマッピングします。

このプールによって使用されるスレッドの最大数は、**scheduled-thread-pool-max-size** パラメーターを使用して設定します。デフォルト値は 5 スレッドです。通常、このプールには少ない数のスレッドで十分です。デフォルトの JBoss EAP メッセージングサーバーのこの値を変更するには、以下の管理 CLI コマンドを使用します。

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

24.2. サーバー汎用スレッドプール

汎用スレッドプールは、サーバー側でほとんどの非同期操作に使用されます。内部的には、**java.util.concurrent.ThreadPoolExecutor** インスタンスにマッピングします。

このプールによって使用されるスレッドの最大数は、**thread-pool-max-size** 属性を使用して設定します。

thread-pool-max-size が **-1** に設定された場合、スレッドプールには上限がなく、要求を満たすのに十分なスレッドが利用できない場合は、新しいスレッドが要求に応じて作成されます。その後アクティビティーがおさまると、スレッドはタイムアウトになり、閉じられます。

thread-pool-max-size がゼロより大きい正の整数に設定されている場合、スレッドプールはバインドされます。要求を受け取り、プールに利用可能な空きスレッドがない場合、要求はスレッドが利用可能になるまでブロックされます。バインドされたスレッドプールは、上限が非常に低く設定された場合にデッドロックの状況になることがあるため、注意して使用することが推奨されます。

thread-pool-max-size のデフォルト値は **30** です。デフォルトの JBoss EAP メッセージングサーバーに新しい値を設定するには、以下の管理 CLI コマンドを使用します。

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

バインドされない (キャッシュ) スレッドプールとバインドされる (固定) スレッドプールの詳細については、**ThreadPoolExecutor** の Javadoc を参照してください。

24.3. サーバースレッド使用状況の監視

スレッドの使用状況をチェックして、スレッドプールのサイズを適切に設定していることを確認します。

スレッドの使用状況をチェックするには、以下の CLI コマンドを実行します。

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

システムは以下のような結果を返します。

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

表24.1 スレッド使用状況データ

使用状況属性	説明
global-client-scheduled-thread-pool-active-count	現在タスクを実行しているすべての ActiveMQ クライアントが使用しているスケジュールプールスレッドの数。
global-client-scheduled-thread-pool-completed-task-count	サーバーが起動してからすべての ActiveMQ クライアントによって実行された、スケジュールプールスレッドを使用するタスクの数。

使用状況属性	説明
global-client-scheduled-thread-pool-current-thread-count	スケジュールプール内で、すべての Active MQ クライアントが使用しているスレッドの現在の数。
global-client-scheduled-thread-pool-keepalive-time	スケジュールスレッドプール内で、アイドル状態のときに実行中のスレッドを保持する時間。
global-client-scheduled-thread-pool-largest-thread-count	スケジュールプール内で、すべての Active MQ クライアントによって同時に使用されたスレッドの最大数。
global-client-scheduled-thread-pool-max-size	スケジュールプール内で、このサーバー内で稼働しているすべての ActiveMQ クライアントによって使用されるスレッドの最大数。
global-client-scheduled-thread-pool-task-count	スケジュールスレッドプール内で、すべての Active MQ クライアントによってスケジュールされたタスクの合計数。
global-client-thread-pool-active-count	現在タスクを実行しているすべての ActiveMQ クライアントによって使用される汎用プールスレッドの数。
global-client-thread-pool-completed-task-count	すべての ActiveMQ クライアントによって実行された汎用プールスレッドを使用するタスクの数。
global-client-thread-pool-current-thread-count	汎用プール内で、すべての Active MQ クライアントが使用しているスレッドの現在の数。
global-client-thread-pool-keepalive-time	汎用スレッドプール内で、アイドル状態のときにスレッドを実行し続ける時間。
global-client-thread-pool-largest-thread-count	汎用プール内で、すべての Active MQ クライアントによって同時に使用されたスレッドの最大数。
global-client-thread-pool-max-size	汎用プール内で、このサーバー内で稼働しているすべての ActiveMQ クライアントによって使用されるスレッドの最大数。
global-client-thread-pool-task-count	汎用スレッドプール内で、すべての Active MQ クライアントによってスケジュールされたタスクの合計数。

24.4. 期限切れリーパースレッド

キュー内の期限が切れたメッセージをスキャンするためにサーバー側でも単一のスレッドが使用されます。このスレッドは、独自の設定可能な優先順位で実行する必要があるため、このためにスレッドプールのいずれかを使用することはできません。

24.5. 非同期 IO

非同期 IO には、ネイティブレイヤーからイベントを送受信するためのスレッドプールがあります。これは、接頭辞が **ArtemisMQ-AIO-poller-pool** のスレッドダンプ上にあります。JBoss EAP メッセージングは、ジャーナル上で開かれたファイルごとに 1 つのスレッドを使用します (通常は 1 つあります)。

また、libaio への書き込みを呼び出すために使用される単一のスレッドもあります。これは、libaio 上で、パフォーマンスの問題を引き起こすコンテキスト切り替えを回避するために行われます。このスレッドは、接頭辞が **ArtemisMQ-AIO-writer-pool** のスレッドダンプ上で見つかります。

24.6. クライアントスレッド管理

JBoss EAP には、クライアント接続の作成に使用されるクライアントスレッドプールが含まれています。このプールは、本章で前述した静的プールとは別のもので、クライアントのように動作するときに JBoss EAP によって使用されます。たとえば、クライアントスレッドプールクライアントは、同じクラスター内の他のノードとのクラスター接続として作成されます。または Artemis リソースアダプターが JBoss EAP のリモートインスタンス内で統合されたリモート Apache ActiveMQ Artemis メッセージングサーバーに接続するときに作成されます。スケジュールクライアントスレッドにもプールがあります。

注記

JBoss EAP 7.1 のリリースでは、クライアントスレッドが 60 秒間何もアクティビティーがなかった後にタイムアウトするようになりました。

管理 CLI を使用したクライアントスレッドプールサイズの設定

管理 CLI を使用して、クライアントスレッドプールとクライアントスケジュールスレッドプールの両方のサイズを設定します。管理 CLI を使用して設定したプールサイズは、システムプロパティーを使用して設定するサイジングよりも優先されます。

以下のコマンドは、クライアントスレッドプールを設定します。

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

この属性に定義されるデフォルト値はありません。属性が定義されていない場合、プールの最大サイズは、CPU コアプロセッサー数の 8 倍になるように決定されます。

現在の設定を確認するには、以下のコマンドを使用します。

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

クライアントスケジュールスレッドプールのサイズは、以下のコマンドを使用して設定します。

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

以下は、クライアントスケジュールスレッドプールの現在のプールのサイズを表示します。デフォルト値は **5** です。

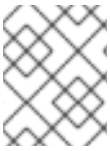
```
/subsystem=messaging-activemq:read-attribute(name=global-client-scheduled-thread-pool-max-size)
```

システムプロパティーを使用したクライアントスレッドプールサイズの設定

以下のシステムプロパティーは、クライアントのグローバルおよびグローバルスケジュールスレッドプールのサイズをそれぞれ設定するために使用できます。

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

以下の例のように、システムプロパティーは XML 設定で参照できます。



注記

管理 CLI を使用して設定したプールサイズは、システムプロパティーで設定したサイズよりも優先されます。

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

独自のスレッドプールを使用するようにクライアントを設定する

クライアントは、各 **ClientSessionFactory** インスタンスが JBoss EAP によって提供されるプールを使用せず、独自のクライアントスレッドプールを使用するように設定できます。この **ClientSessionFactory** から作成されるセッションはどれも、新規作成されるプールを使用します。

ClientSessionFactory インスタンスが独自のプールを使用するように設定するには、ファクトリーの作成後すぐに適切なセッターメソッドを呼び出します。例を以下に示します。

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

JMS API を使用している場合は、**ClientSessionFactory** に同じパラメーターを設定できます。例を以下に示します。

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

JNDI を使用して **ActiveMQConnectionFactory** インスタンスをインスタンス化する場合は、JBoss EAP のスタンドアロンインスタンスの以下の例にあるように管理 CLI を使用してこれらのパラメーターを設定することもできます。

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

/subsystem=messaging-activemq/server=default/connection-factory=myConnectionFactory:write-
```

```
attribute(name=scheduled-thread-pool-max-size,value=10)

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

上記の各コマンドを実行すると、インスタンスのリロードが必要であることが管理 CLI により通知されることに注意してください。

第25章 重複メッセージ検出の設定

送信者が別のサーバーにメッセージを送信する場合、メッセージの送信後、ただし、プロセスが成功したことを示す応答を送信者に送信する前に、ターゲットサーバーまたは接続でエラーが発生する状況があります。このような状況では、メッセージが目的の受信者に正常に送信されたかどうかを送信者が判断することは非常に困難です。送信者が最後のメッセージの再送信を決定する場合、そのアドレスに重複メッセージが送信される可能性があります。

アプリケーションが重複したメッセージをフィルターするロジックを提供しなくてもよいように、JBoss EAP メッセージングでは重複メッセージ検出を設定できます。

25.1. メッセージ送信に重複メッセージ検出を使用する

送信されたメッセージに対して重複メッセージ検出を有効にするには、**org.apache.activemq.artemis.api.core.Message.HDR_DUPLICATE_DETECTION_ID** プロパティーの値を設定する必要があります。これにより、**_AMQ_DUPL_ID** に解決され、一意の値になります。ターゲットサーバーがメッセージを受信すると、**_AMQ_DUPL_ID** プロパティーが設定されている場合は、メモリーキャッシュをチェックして、そのヘッダーの値が含まれるメッセージをすでに受信したかどうかを確認します。すでに受信した場合、このメッセージは無視されます。詳細は、「重複 ID キャッシュの設定」を参照してください。

コア API を使用している場合は、**_AMQ_DUPL_ID** プロパティーの値を **byte[]** または **SimpleString** 型にすることができます。JMS を使用している場合は、**String** である必要があります。

以下の例は、コア API のプロパティーを設定する方法を示しています。

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

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

以下の例は、JMS クライアントのプロパティーを設定する方法を示しています。

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

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

重要

重複メッセージは、**HDR_DUPLICATE_DETECTION_ID** プロパティーを使用して同じトランザクション内に送信されたときは検出され**ません**。

25.2. 重複 ID キャッシュの設定

サーバーは、各アドレスに送信される **_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。各アドレスは独自のアドレスキャッシュを維持します。

キャッシュのサイズは固定です。キャッシュの最大サイズは **id-cache-size** 属性を使用して設定します。このパラメーターのデフォルト値は **20000** 要素です。キャッシュの最大サイズが **n** 要素の場合は、(**n + 1**) 番目の ID が保存され、キャッシュ内の要素 **0** が上書きされます。この値は、以下の管理 CLI コマンドを使用して設定します。

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

キャッシュは、ディスクに永続的に残るように設定することもできます。これを設定するには、以下の管理 CLI コマンドを使用して **persist-id-cache** 属性を設定します。

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

この値を **true** に設定すると、各 ID は受信時に永続ストレージに永続的に残ります。このパラメーターのデフォルト値は **true** です。

注記

メッセージを再送信した場合にキャッシュに保存されている送信済みのメッセージが上書きされないようにするには、重複 ID キャッシュのサイズを大きなサイズに設定します。

第26章 低速なコンシューマーの処理

サーバー側のキューがある低速なコンシューマーは、サーバーのパフォーマンスに大きな問題を引き起こす可能性があります。メッセージがコンシューマーのサーバー側キューで構築されるため、メモリー使用量が増えます。その結果、サーバーがページングモードに入り、パフォーマンスに悪影響を与える可能性があります。別の大きな問題として、**consumer-windows-size** 属性が **0** よりも大きい場合は、クライアントのメッセージバッファーに送信されたメッセージが消費されるのを待ち続ける可能性があります。メッセージをすぐに消費しないコンシューマーをサーバーから切断できるように基準を設定できます。

低速なコンシューマーが MDB の場合、JBoss EAP サーバーが接続を管理します。MDB が切断されると、MDB が消費していたキューにメッセージが返され、MDB が自動的に再接続します。この時点で、メッセージはキュー上のすべての MDB に再度負荷分散されます。この同じプロセスは、永続的なトピックでリッスンする MDB にも当てはまります。JMS コンシューマーの場合は、速度が遅いと切断され、**reconnects-attempts** が **-1** に設定されている場合や、**0** よりも大きい場合にのみ再接続します。

非永続的な JMS サブスクライバーまたは非永続的なサブスクリプションを持つ MDB の場合、接続は切断されます。その結果、サブスクリプションが削除されます。非永続的なサブスクライバーが再接続すると、新しい非永続的なサブスクリプションが作成され、トピックに送信された新しいメッセージのみ消費が開始されます。

コンシューマーが遅いかどうかを判断する計算では、特定のコンシューマーが確認応答したメッセージの数のみを確認します。たとえば、フロー制御がコンシューマーで有効になっているか、またはコンシューマーが大きいメッセージをストリーミングしているかどうかは考慮しません。低速なコンシューマー検出を設定する場合はこれに留意してください。

低速なコンシューマーのチェックは、スケジュールスレッドプールを使用して実行されます。低速なコンシューマー検出が有効になっているサーバー上の各キューにより、内部の **java.util.concurrent.ScheduledThreadPoolExecutor** インスタンスで新しいエントリーが発生します。キューが多数あり、**slow-consumer-check-period** が比較的小さい場合は、チェックの一部の実行に遅延が生じる可能性があります。ただし、検出アルゴリズムで使用される計算の精度には影響しません。このスレッドプールに関する詳細は、「スレッド管理」を参照してください。

低速なコンシューマー処理は、**address-setting** に基づきます。**address-setting** 設定に関する詳細は、「アドレス設定」を参照し、付録の **address-setting** 属性のリストを参照してください。低速なコンシューマーの処理を設定するために使用される属性は 3 つあります。以下のとおりです。

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。同じ接続を使用するどのクライアントスレッドにも影響を与えます。
  - **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。デフォルトは **-1** で、バインドされません。

管理 CLI を使用して、属性の現在の値を読み取ります。たとえば、以下のコマンドを使用して、**myAddress** という名前の **address-setting** の現在の **slow-consumer-policy** を読み取ります。

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

同様に、以下の例を使用して同じ **slow-consumer-policy** を設定します。

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

パート III. 複数ノードのメッセージングシステムの設定

第27章 JMS ブリッジの設定

JBoss EAP メッセージングには、完全な機能を備えた JMS メッセージブリッジが含まれます。JMS ブリッジの機能は、ソースキューまたはトピックからメッセージを消費し、通常は別のサーバーにあるターゲットキューまたはトピックに送信することです。

ソース宛先がトピックの場合、JMS ブリッジはそのサブスクリプションを作成します。**client-id** および **subscription-name** 属性が JMS ブリッジに設定されていると、サブスクリプションが永続化されます。これは、JMS ブリッジが停止後に再起動されてもメッセージがなくならないことを意味します。

ソースサーバーとターゲットサーバーは同じクラスターに置く必要はありません。このため、たとえば WAN を介して、接続が信頼できない場合に、あるクラスターから別のクラスターに確実にメッセージを送信するのにブリッジングが適しています。

注記

JMS ブリッジをコアブリッジと混同しないようにしてください。JMS ブリッジは、JMS 1.1 準拠の 2 つの JMS プロバイダーをブリッジするのに使用でき、JMS API を使用します。コアブリッジは、2 つの JBoss EAP メッセージングインスタンスをブリッジするのに使用され、コア API を使用します。可能な場合は、JMS ブリッジではなく、コアブリッジを使用することが推奨されます。

以下の例は、JBoss EAP JMS ブリッジの設定を示しています。

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

管理 CLI を使用した JMS ブリッジの追加

以下の管理 CLI コマンドを使用して JMS ブリッジを追加できます。ソースとターゲットの宛先が設定にすでに定義されている必要があることに注意してください。設定可能な属性の完全なリストは、付録の表を参照してください。

```
/subsystem=messaging-activemq/jms-bridge=my-jms-bridge:add(quality-of-
service=AT_MOST_ONCE,module=org.apache.activemq.artemis,failure-retry-interval=500,max-
retries=1,max-batch-size=10,max-batch-time=100,source-connection-
factory=ConnectionFactory,source-destination=jms/queue/InQueue,source-context={},target-
```

```
connection-factory=jms/RemoteConnectionFactory,target-destination=jms/queue/OutQueue,target-
context=
{java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory,java.naming.provider.u
=http-remoting://192.168.40.1:8080})
```

以下の例のように、管理 CLI で **read-resource** コマンドを使用して JMS ブリッジの設定を確認できます。

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

以下の例にあるように、**write-attribute** を使用して JMS ブリッジに設定を追加します。

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

管理コンソールを使用した JMS ブリッジの追加

以下の手順に従うことで、管理コンソールを使用して JMS ブリッジを追加することもできます。

1. ブラウザーで管理コンソールを開き、**設定 → サブシステム → Messaging (ActiveMQ) → JMS ブリッジ**に移動します。
2. 追加 (+) ボタンをクリックし、プロンプトが表示されたら必要な情報を入力します。
3. 終了したら **追加** をクリックします。

27.1. サービス品質

JBoss EAP の **quality-of-service** は、メッセージが消費され、確認応答される方法を決定する設定可能な属性です。**quality-of-service** の有効な値とその説明を以下に示します。JMS ブリッジ属性の完全なリストは、付録の表を参照してください。

AT_MOST_ONCE
: メッセージはソースから宛先に最大 1 回到達します。メッセージはソースから消費され、確認応答された後、宛先に送信されます。そのため、ソースで削除され、宛先に到達するまでの間に障害が発生した場合に、メッセージが失われる可能性があります。このモードはデフォルト値です。
このモードは永続メッセージと非永続メッセージの両方に使用できます。

DUPLICATES_OK
: メッセージは、宛先に正常に送信された後、ソースから消費され、確認応答されます。そのため、最初のメッセージが送信された後で、確認応答される前にエラーが発生した場合、メッセージを再送信できる可能性があります。
このモードは永続メッセージと非永続メッセージの両方に使用できます。

ONCE_AND_ONLY_ONCE
: メッセージはソースから宛先に 1 回のみ到達します。ソースと宛先の両方が同じサーバーインスタンスにある場合は、同じローカルトランザクションでメッセージを送信して確認応答することで、これを実現できます。ソースと宛先が異なるサーバー上にある場合は、JTA トランザクションで送信セッションと消費セッションを登録することで実現されます。JTA トランザクションは JTA トランザクションマネージャーによって制御されます。これは、ブリッジで **setTransactionManager()** メソッドを使用して設定する必要があります。
このモードは永続メッセージのみに使用できます。

警告

quality-of-service 属性が **ONCE_AND_ONLY_ONCE** に設定されているデプロイ済みの JMS ブリッジを持つサーバーをシャットダウンする場合は、予期しないエラーを防ぐために、JMS ブリッジを持つサーバーを最初にシャットダウンするようにしてください。

ONCE_AND_ONLY_ONCE ではなく **DUPLICATES_OK** モードを使用し、宛先で重複を確認して破棄することにより、1 回限りのセマンティクスを提供できる可能性があります。詳細は、「重複メッセージ検出の設定」を参照してください。ただし、キャッシュは一定期間のみ有効になります。そのため、このアプローチは **ONCE_AND_ONLY_ONCE** を使用するほどは完璧ではありませんが、特定のアプリケーションのニーズによっては適切な選択肢となる可能性があります。

27.2. タイムアウトと JMS ブリッジ

ターゲットサーバーまたはソースサーバーは、ある時点で利用できなくなることがあります。これが発生すると、ブリッジは **max-retries** の値と同じ回数、再接続を試みます。試行までの待機時間は、**failure-retry-interval** で設定します。

警告

JMS ブリッジごとに固有の **max-retries** パラメーターがあるため、**reconnect-attempts** パラメーターを設定しない接続ファクトリーを使用する必要があります。または、**0** に設定します。これにより、再接続時間が長くなる可能性がある競合が回避されます。また、JMS ブリッジによって参照される接続ファクトリーが **quality-of-service** が **ONCE_AND_ONLY_ONCE** に設定されているとします。その場合は、**factory-type** を **XA_GENERIC**、**XA_TOPIC**、または **XA_QUEUE** に設定する必要があります。

第28章 コアブリッジの設定

ブリッジの機能は、ある宛先からメッセージを消費し、通常は異なる JBoss EAP メッセージングサーバー上にある別の宛先に転送します。

ソースサーバーとターゲットサーバーは同じクラスターに置く必要はありません。このため、たとえば WAN やインターネットを介して、接続が信頼できない場合に、あるクラスターから別のクラスターに確実にメッセージを送信するのにブリッジングが適しています。

ブリッジには障害に対する回復力が組み込まれているため、ネットワーク障害などの理由でターゲットサーバー接続が失われた場合、ブリッジは、オンラインに戻るまでターゲットへの接続を再試行します。オンラインに戻ると、通常どおりに操作を再開します。

ブリッジとは、2 つの別個の JBoss EAP メッセージングサーバーを確実に相互接続する方法です。コアブリッジでは、ソースサーバーとターゲットサーバーの両方が JBoss EAP 7 メッセージングサーバーである必要があります。

注記

コアブリッジを JMS ブリッジと混同しないようにしてください。コアブリッジは、2 つの JBoss EAP メッセージングインスタンスをブリッジするのに使用され、コア API を使用します。JMS ブリッジは、JMS 1.1 準拠の 2 つの JMS プロバイダーをブリッジするのに使用でき、JMS API を使用します。可能な場合は、JMS ブリッジではなく、コアブリッジを使用することが推奨されます。

以下の例は、JBoss EAP メッセージングコアブリッジの設定を示しています。

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

以下の管理 CLI コマンドを使用して、このコアブリッジを追加できます。コアブリッジを定義する際は、**queue-name** を定義し、**static-connectors** または **discovery-group** のいずれかを定義する必要があることに注意してください。設定可能な属性の完全なリストは、付録の表を参照してください。

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

28.1. 重複検出のためのコアブリッジの設定

コアブリッジは、メッセージをターゲットに転送する前に、一意の重複 ID 値 (メッセージ内にまだない場合) を自動的に追加するように設定できます。重複メッセージ検出にコアブリッジを設定するには、**use-duplicate-detection** 属性を **true** (デフォルト値) に設定します。

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

第29章 クラスターの概要

JBoss EAP メッセージングクラスターを使用すると、メッセージ処理負荷を共有するために JBoss EAP メッセージングサーバーのグループをグループ化できます。クラスター内のアクティブな各ノードは、独自のメッセージを管理し、独自の接続を処理するアクティブな JBoss EAP メッセージングサーバーです。

警告

異なるバージョンの JBoss EAP で構成される混合クラスターは **messaging-activemq** サブシステムによってサポートされていません。メッセージングクラスターを形成するサーバーはすべて、同じバージョンの JBoss EAP を使用する必要があります。

クラスターは、JBoss EAP 設定ファイルの他のノードへクラスター接続を宣言する各ノードによって形成されます。ノードが別のノードへのクラスター接続を形成すると、内部的には、そのノードと他のノード間のコアブリッジ接続を作成します。これは、背後で透過的に実行されます。各ノードに明示的なブリッジを宣言する必要はありません。これらのクラスター接続により、クラスターのノード間でメッセージが流れ、負荷分散を行います。

クラスタリングの重要な部分が、クライアントまたは他のサーバーが最小設定で接続できるようにサーバーが接続の詳細をブロードキャストできるサーバー検出です。

このセクションでは、クラスターのノード間でクライアント接続のバランスを取るためのクライアント側のロードバランシング、JBoss EAP メッセージングが枯渇を回避するためにノード間でメッセージを再分配するメッセージ再分配についても説明します。

警告

クラスターノードを設定したら、その設定を他のノードに単純にコピーしてシンメトリッククラスターを生成することが一般的です。

実際、予期しないエラーを防ぐために、クラスターの各ノードは、次の要素に対して同じ設定を共有する必要があります。

- cluster-connection
- broadcast-group
- discovery-group
- address-settings (キューとトピックを含む)

ただし、JBoss EAP のメッセージングファイルをコピーする際は注意が必要です。メッセージングデータ、bindings、journal、large-messages のディレクトリーをノード間でコピーしないでください。クラスターノードが初めて起動され、そのジャーナルファイルを初期化するとき、特別な識別子がジャーナルディレクトリーに保持されます。識別子は、クラスターが適切に形成できるようにノード間で一意である必要があります。

29.1. サーバー検出

サーバー検出は、サーバーが以下の対象に接続詳細を伝播できるメカニズムです。

- メッセージングクライアント
  メッセージングクライアントは、ある時点でクラスター内で稼働しているサーバーの具体的な情報を持たずにクラスターのサーバーに接続することができます。
- 他のサーバー
  クラスター内のサーバーは、クラスター内の他のすべてのサーバーについて事前の情報を持たずに相互のクラスター接続を作成することができます。

この情報、またはクラスタートポロジーは、通常の JBoss EAP メッセージング接続をクライアントに送信、およびクラスター接続を介して他のサーバーに送信されます。ただし、最初の接続を確立する方法が必要です。これを行うには、UDP や JGroups などの動的な検出技術を使用するか、初期コネクターの静的リストを指定します。

29.1.1. ブロードキャストグループ

ブロードキャストグループは、サーバーがネットワーク経由でコネクターをブロードキャストする手段です。コネクターは、クライアントまたは他のサーバーがサーバーに接続できる方法を定義します。

ブロードキャストグループはコネクターのセットを取得し、ネットワーク上でブロードキャストします。クラスターを設定するブロードキャスト技術に応じて、UDP または JGroups を使用してコネクターのペア情報をブロードキャストします。

ブロードキャストグループは、サーバー設定の **messaging-activemq** サブシステム内に定義されます。JBoss EAP メッセージングサーバーごとに多くのブロードキャストグループが存在します。

UDP を使用したブロードキャストグループの設定
以下は、UDP ブロードキャストグループを定義するメッセージングサーバーの設定例です。この設定は、**messaging-group** ソケットバインディングに依存することに注意してください。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <broadcast-group name="my-broadcast-group" connectors="http-connector" socket-
binding="messaging-group"/>
    ...
  </server>
</subsystem>
...
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
  ...
  <socket-binding name="messaging-group" interface="private" port="5432" multicast-
address="231.7.7.7" multicast-port="9876"/>
  ...
</socket-binding-group>
```

この設定は、以下の管理 CLI コマンドを使用して実現できます。

1. **messaging-group** ソケットバインディングを追加します。

```
/socket-binding-group=standard-sockets/socket-binding=messaging-
group:add(interface=private,port=5432,multicast-address=231.7.7.7,multicast-port=9876)
```

2. ブロードキャストグループを追加します。

```
/subsystem=messaging-activemq/server=default/broadcast-group=my-broadcast-
group:add(socket-binding=messaging-group,broadcast-period=2000,connectors=[http-
connector])
```

JGroups を使用したブロードキャストグループの設定
以下は、デフォルトの JGroups ブロードキャストグループを使用するブロードキャストグループを定義するメッセージングサーバーの設定例で、UDP を使用します。JGroups を使用してブロードキャストできるようにするには、**jgroups-channel** を設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <broadcast-group name="my-broadcast-group" connectors="http-connector" jgroups-
cluster="activemq-cluster"/>
    ...
  </server>
</subsystem>
```

これは、以下の管理 CLI コマンドを使用して設定できます。

```
/subsystem=messaging-activemq/server=default/broadcast-group=my-broadcast-
group:add(connectors=[http-connector],jgroups-cluster=activemq-cluster)
```

ブロードキャストグループの属性

以下の表は、ブロードキャストグループの設定可能な属性を示しています。

属性	説明
broadcast-period	連続するブロードキャストの間隔 (ミリ秒単位)。
connectors	ブロードキャストされるコネクターの名前。
jgroups-channel	**jgroups** サブシステムに定義されているチャネルの名前。クラスターを形成するために **jgroups-cluster** 属性と組み合わせて使用されます。定義されていない場合は、**default** のチャネルが使用されます。**jgroups-channel** は、**jgroups-cluster** 属性によって識別される個別の論理グループ間のグループ通信を多重化することに注意してください。
jgroups-cluster	ブロードキャストグループと検出グループ間の通信に使用される論理名。特定のブロードキャストグループからのメッセージを受信しようとする検出グループは、ブロードキャストグループで使用されるものと同じクラスター名を使用する必要があります。
jgroups-stack	クラスターの形成に使用される **jgroups** サブシステムに定義されているスタックの名前。この属性は非推奨です。代わりに、**jgroups-channel** を使用してクラスターを形成します。各 **jgroups-stack** はすでに **jgroups-channel** に関連付けられているため、そのチャネルを使用するか、新しい **jgroups-channel** を作成して **jgroups-stack** に関連付けることができます。  **重要** **jgroups-stack** と **jgroups-channel** の両方を指定すると、新しい **jgroups-channel** が生成され、**jgroups-stack** および **jgroups-channel** と同じ名前空間に登録されます。そのため、**jgroups-stack** と **jgroup-channel** の名前は一意である必要があります。
socket-binding	ブロードキャストグループソケットバインディング。

29.1.2. 検出グループ

ブロードキャストグループは、コネクター情報がサーバーからブロードキャストされる方法を定義し、検出グループは、UDP マルチキャストアドレスや JGroup チャネルなど、ブロードキャストエンドポイントからコネクター情報を受け取る方法を定義します。

検出グループは、個別のサーバー別にブロードキャストごとに 1 つずつ、コネクターのリストを維持します。特定のサーバーからブロードキャストエンドポイントでブロードキャストを受け取ると、そのサーバーのリスト内のエントリーが更新されます。特定のサーバーから一定時間ブロードキャストを受信していない場合は、そのサーバーのエントリーがリストから削除されます。

検出グループは JBoss EAP メッセージングの 2 つの場所で使用されます。

- クラスター接続により、トポロジーをダウンロードするために初期接続を取得する方法を把握します。
- メッセージングクライアントにより、トポロジーをダウンロードするために初期接続を取得する方法を把握します。

検出グループは常にブロードキャストを受け付けますが、現在の利用可能なライブおよびバックアップサーバーの一覧は、初期接続が確立された場合にのみ使用されます。その後、サーバー検出は通常の JBoss EAP メッセージング接続で行われます。

注記

各検出グループは、対応するブロードキャストグループと一致するブロードキャストエンドポイント (UDP または JGroups) で設定する必要があります。たとえば、ブロードキャストグループが UDP を使用して設定されている場合、検出グループも UDP を使用し、同じマルチキャストアドレスを使用する必要があります。

29.1.2.1. サーバー上の検出グループの設定

検出グループは、サーバー設定の **messaging-activemq** サブシステムで定義します。JBoss EAP メッセージングサーバーごとに多くの検出グループが存在します。

UDP を使用した検出グループの設定
以下は、UDP 検出グループを定義するメッセージングサーバーの設定例です。この設定は、**messaging-group** ソケットバインディングに依存することに注意してください。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <discovery-group name="my-discovery-group" refresh-timeout="10000" socket-
binding="messaging-group"/>
    ...
  </server>
</subsystem>
...
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
  ...
  <socket-binding name="messaging-group" interface="private" port="5432" multicast-
address="231.7.7.7" multicast-port="9876"/>
  ...
</socket-binding-group>
```

この設定は、以下の管理 CLI コマンドを使用して実現できます。

1. **messaging-group** ソケットバインディングを追加します。

```
/socket-binding-group=standard-sockets/socket-binding=messaging-
group:add(interface=private,port=5432,multicast-address=231.7.7.7,multicast-port=9876)
```

2. 検出グループを追加します。

```
/subsystem=messaging-activemq/server=default/discovery-group=my-discovery-
group:add(socket-binding=messaging-group,refresh-timeout=10000)
```

JGroups を使用した検出グループの設定
以下は、JGroups 検出グループを定義するメッセージングサーバーの設定例です。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <discovery-group name="my-discovery-group" refresh-timeout="10000" jgroups-cluster="activemq-cluster"/>
    ...
  </server>
</subsystem>
```

これは、以下の管理 CLI コマンドを使用して設定できます。

```
/subsystem=messaging-activemq/server=default/discovery-group=my-discovery-group:add(refresh-timeout=10000,jgroups-cluster=activemq-cluster)
```

検出グループの属性
以下の表は、検出グループの設定可能な属性を示しています。

属性	説明
initial-wait-timeout	最初のブロードキャストがクラスター内の少なくとも 1 つのノードを提供するまで待機する時間 (ミリ秒単位)。
jgroups-channel	**jgroups** サブシステムに定義されているチャネルの名前。クラスターを形成するために **jgroups-cluster** 属性と組み合わせて使用されます。定義されていない場合は、**default** のチャネルが使用されます。**jgroups-channel** は、**jgroups-cluster** 属性によって識別される個別の論理グループ間のグループ通信を多重化することに注意してください。
jgroups-cluster	ブロードキャストグループと検出グループ間の通信に使用される論理名。特定のブロードキャストグループからのメッセージを受信しようとする検出グループは、ブロードキャストグループで使用されるものと同じクラスター名を使用する必要があります。
jgroups-stack	クラスターの形成に使用される **jgroups** サブシステムに定義されているスタックの名前。この属性は非推奨です。代わりに、**jgroups-channel** を使用してクラスターを形成します。各 **jgroups-stack** はすでに **jgroups-channel** に関連付けられているため、そのチャネルを使用するか、新しい **jgroups-channel** を作成して **jgroups-stack** に関連付けることができます。  **重要** **jgroups-stack** と **jgroups-channel** の両方を指定すると、新しい **jgroups-channel** が生成され、**jgroups-stack** および **jgroups-channel** と同じ名前空間に登録されます。そのため、**jgroups-stack** と **jgroup-channel** の名前は一意である必要があります。

属性	説明
refresh-timeout	特定のサーバーから最後のブロードキャストを受信した後、そのサーバ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検出グループが待機する時間。
socket-binding	検出グループソケットバインディング。

警告

上記の JGroups 属性と UDP 固有属性は相互排他的です。検出グループ設定で指定できるのは、1 つのセットのみです。

29.1.2.2. クライアント側の検出グループの設定

JMS またはコア API を使用して JBoss EAP メッセージングクライアントを設定し、接続できるサーバーのリストを検出できます。

JMS を使用したクライアント検出の設定
JMS を使用するクライアントは、関連する **ConnectionFactory** を JNDI を使用して検索できます。**connection-factory** または **pooled-connection-factory** の **entries** 属性は、ファクトリーが公開される JNDI 名を指定します。以下は、JNDI を使用してルックアップするためにリモートクライアントに設定された **ConnectionFactory** の例です。

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

注記

java:jboss/exported 名前空間にバインドされた JNDI 名のみがリモートクライアントに使用できることに注意してください。**connection-factory** が **java:jboss/exported** 名前空間にバインドされたエントリーを持っている場合、リモートクライアントは **java:jboss/exported** の後のテキストを使用して **connection-factory** を検索します。たとえば、**RemoteConnectionFactory** はデフォルトで **java:jboss/exported/jms/RemoteConnectionFactory** にバインドされています。これは、リモートクライアントが **jms/RemoteConnectionFactory** を使用してこの **connection-factory** をルックアップすることを意味します。**pooled-connection-factory** はリモートクライアントに適していないため、**pooled-connection-factory** では **java:jboss/exported** 名前空間内にエントリーがバインドされないようにする必要があります。

JMS 2.0 以降、デフォルトの JMS 接続ファクトリーは、JNDI 名 **java:comp/DefaultJMSConnectionFactory** で Java EE アプリケーションを利用できます。JBoss EAP の **messaging-activemq** サブシステムでは、このデフォルトの接続ファクトリーを提供するために使用される **pooled-connection-factory** を定義します。この **pooled-connection-factory** のパラメーターの変更は、JNDI 名 **java:comp/DefaultJMSConnectionFactory** で、デフォルトの JMS プロバイダーを検索する Java EE アプリケーションによって考慮されます。以下は、***-full** プロファイルおよび ***-full-ha** プロファイルに定義されているデフォルトのプールされた接続ファクトリーです。

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

コア API を使用したクライアント検出の設定
コア API を使用して **ClientSessionFactory** インスタンスを直接インスタンス化する場合、セッションファクトリーの作成時に検出グループパラメーターを直接指定できます。例を以下に示します。

```
final String groupAddress = "231.7.7.7";
final int groupPort = 9876;

ServerLocator factory =
  ActiveMQClient.createServerLocatorWithHA(new DiscoveryGroupConfiguration(
      groupAddress,
      groupPort,
      new UDPBroadcastGroupConfiguration(groupAddress, groupPort, null, -1)));
ClientSessionFactory factory = locator.createSessionFactory();
ClientSession session1 = factory.createSession();
ClientSession session2 = factory.createSession();
```

DiscoveryGroupConfiguration で **setDiscoveryRefreshTimeout()** セッターメソッドを使用して **refresh-timeout** 値を設定できます。デフォルトは **10000** ミリ秒です。

また、**DiscoveryGroupConfiguration** で **setDiscoveryInitialWaitTimeout()** セッターメソッドを使用して **initial-wait-timeout** 値を設定することもできます。これは、最初のセッションを作成する前にセッションファクトリーが待機する時間を決定します。デフォルト値は、**10000** ミリ秒です。

29.1.3. 静的検出

ネットワークで UDP を使用できない、または使用しない場合は、1 つ以上のサーバーの初期リストで接続を設定できます。

これは、すべてのサーバーがホストされる場所を、認識しなければならないという意味ではありません。これらのサーバーが、信頼できるサーバーに接続するように設定すると、そのサーバーを使用して接続の詳細を伝播できます。

クラスター接続の設定
クラスター接続の場合、追加の設定は必要ありません。コネクターが通常の方法で定義されていることを確認するだけです。これらは、クラスター接続の設定によって参照されます。

クライアント接続の設定
使用可能なサーバーの静的リストは、クライアントからも使用できます。

JMS を使用したクライアント検出の設定
JMS で静的検出を使用するには、複数のコネクター (それぞれがクラスター内の一意のノードを指す) で **connection-factory** を設定し、JNDI を使用してクライアントで ConnectionFactory を検索する方法が推奨されます。以下は、このような **connection-factory** を示す設定のスニペットです。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <connection-factory name="MyConnectionFactory" entries="..." connectors="http-connector http-
node1 http-node2"/>
    ...
  </server>
</subsystem>
```

上記の例では、**http-connector** はローカルサーバーを指す HTTP コネクター (**<http-connector>**)、**http-node1** はサーバー **node1** を指す HTTP コネクターというようになります。サーバー設定にコネクターを設定するには、「コネクターとアクセプター」セクションを参照してください。

コア API を使用したクライアント検出の設定
コア API を使用している場合は、クラスター内の各サーバーに一意の **TransportConfiguration** を作成し、以下のサンプルコードのように **ServerLocator** を作成するメソッドに渡します。

```
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("host", "myhost");
map.put("port", "8080");

HashMap<String, Object> map2 = new HashMap<String, Object>();
map2.put("host", "myhost2");
map2.put("port", "8080");

TransportConfiguration server1 = new
TransportConfiguration(NettyConnectorFactory.class.getName(), map);
TransportConfiguration server2 = new
TransportConfiguration(NettyConnectorFactory.class.getName(), map2);

ServerLocator locator = ActiveMQClient.createServerLocatorWithHA(server1, server2);
ClientSessionFactory factory = locator.createSessionFactory();
ClientSession session = factory.createSession();
```

29.2. サーバー側のメッセージロードバランシング

クラスター接続がクラスターのノード間で定義されている場合、JBoss EAP メッセージングはクライアントから特定ノードに到達するメッセージを負荷分散します。

メッセージングクラスター接続は、一致するコンシューマーが他のノードにあるかどうかに関係なく、ラウンドロビン方式でメッセージを負荷分散するように設定できます。一致するコンシューマーが存在する場合にのみ、他のノードに分散するように設定することもできます。詳細は、「メッセージ再分配」セクションを参照してください。

クラスター接続の設定
クラスター接続はサーバーをクラスターにグループ化します。このため、クラスターのノード間でメッセージの負荷を分散できます。クラスター接続は、**cluster-connection** 要素を使用して JBoss EAP サーバー設定に定義します。

警告

Red Hat は、**messaging-activemq** サブシステム内で 1 つの **cluster-connection** の使用のみをサポートします。

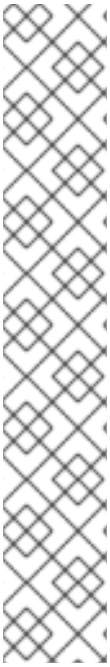
以下は、***-full** プロファイルおよび ***-full-ha** プロファイルで定義されているデフォルトの **cluster-connection** です。属性の詳細リストは、「クラスター接続の属性」を参照してください。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <cluster-connection name="my-cluster" discovery-group="dg-group1" connector-name="http-
connector" address="jms"/>
    ...
  </server>
</subsystem>
```

上記の場合、クラスター接続は「jms」から始まるアドレスに送信されるメッセージを負荷分散します。このクラスター接続は、サブ文字列「jms」から始まるコアキューにマップされるので、すべての JMS キューおよびトピックに適用されます。

注記

パケットがクラスター接続を使用して送信され、ブロックされた状態で確認応答を待機している場合、**call-timeout** 属性は例外をスローする前に応答を待機する時間を指定します。デフォルト値は **30000** です。たとえば、リモート JMS ブローカーがネットワークから切断され、トランザクションが不完全な場合、スレッドは接続が再確立されるまでスタック状態のままになる可能性があります。この状況を回避するには、**call-failover-timeout** 属性を **call-timeout** 属性と一緒に使用することが推奨されます。**call-failover-timeout** 属性は、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されます。デフォルト値は **-1** で、タイムアウトなしを意味します。クライアントフェイルオーバーに関する詳細は、「自動クライアントフェイルオーバー」を参照してください。



注記

または、クラスター接続で、検出にサーバーの静的リストを使用する場合は、**static-connectors** 属性を使用することができます。例を以下に示します。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <cluster-connection name="my-cluster" static-connectors="server0-connector
server1-connector" .../>
    ...
  </server>
</subsystem>
```

この例には 2 つのサーバーが定義され、少なくとも 1 つが使用可能になることがわかります。クラスターにはさらに多くのサーバーが存在する場合があります。しかし、これらは、初期接続が確立されると、これらのコネクターのいずれかを使用して検出されます。

重複検出のためのクラスター接続の設定

クラスター接続は、クラスターのノード間でメッセージを移動するために内部的にコアブリッジを使用します。重複メッセージ検出にクラスター接続を設定するには、**use-duplicate-detection** 属性を **true** (デフォルト値) に設定します。

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

クラスターユーザーの認証情報

クラスターのノード間で接続を作成し、クラスター接続を形成するとき、JBoss EAP メッセージングはクラスターのユーザーとパスワードを使用します。

以下の管理 CLI コマンドを使用して、クラスターのユーザーとパスワードを設定できます。

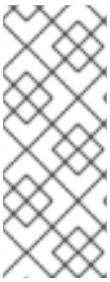
```
/subsystem=messaging-activemq/server=default:write-attribute(name=cluster-
user,value="NewClusterUser")
/subsystem=messaging-activemq/server=default:write-attribute(name=cluster-
password,value="NewClusterPassword123")
```

これにより、以下の XML コンテンツが JBoss EAP 設定ファイルの **messaging-activemq** サブシステムに追加されます。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <cluster user="NewClusterUser" password="NewClusterPassword123"/>
    ...
  </server>
</subsystem>
```

警告

cluster-user のデフォルト値は **ACTIVEMQ.CLUSTER.ADMIN.USER** で、**cluster-password** のデフォルト値は **CHANGE ME!!** です。これらの値はデフォルトから変更するか、リモートクライアントがデフォルト値を使用してサーバーへ接続できるようにする必要があります。デフォルトから変更されない場合、JBoss EAP メッセージングはこれを検出し、起動するたびに警告を表示します。



注記

cluster-credential-reference 属性を使用して、クラスターパスワードを設定する代わりにクレデンシャルストアを参照することもできます。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=cluster-credential-reference,value={clear-text=SecretStorePassword})
```

29.3. クライアント側のロードバランシング

JBoss EAP メッセージングのクライアント側のロードバランシングでは、単一のセッションファクトリーを使用して作成される後続のセッションをクラスターの異なるノードに接続できます。これにより、セッションがクラスターのノードにスムーズに広がり、特定のノードに集中することがありません。

クライアントファクトリーによって使用されるロードバランシングポリシーを宣言するための推奨される方法は、**<connection-factory>** リソースの **connection-load-balancing-policy-class-name** 属性を設定することです。JBoss EAP メッセージングでは、以下の追加設定なしのロードバランシングポリシーが提供され、独自のロードバランシングも実装できます。

ラウンドロビン

このポリシーでは、最初のノードがランダムに選択され、後続の各ノードが同じ順序で順番に選択されます。
たとえば、ノードの選択順序が、**B**、**C**、**D**、**A**、**B**、**C**、**D**、**A**、**B** や **D**、**A**、**B**、**C**、**D**、**A**、**B**、**C** のようになります。

org.apache.activemq.artemis.api.core.client.loadbalance.RoundRobinConnectionLoadBalancingPolicy を **connection-load-balancing-policy-class-name** として使用します。

ランダム

このポリシーでは、各ノードはランダムに選択されます。
org.apache.activemq.artemis.api.core.client.loadbalance.RandomConnectionLoadBalancingPolicy を **connection-load-balancing-policy-class-name** として使用します。

ランダムスティッキー

このポリシーでは、最初のノードがランダムに選択され、後続の接続に再利用されます。
org.apache.activemq.artemis.api.core.client.loadbalance.RandomStickyConnectionLoadBalancingPolicy を **connection-load-balancing-policy-class-name** として使用します。

最初の要素

このポリシーでは、最初 (0 番目) のノードが常に返されます。
org.apache.activemq.artemis.api.core.client.loadbalance.FirstElementConnectionLoadBalancingPolicy を **connection-load-balancing-policy-class-name** として使用します。

また、**org.apache.activemq.artemis.api.core.client.loadbalance.ConnectionLoadBalancingPolicy** インターフェースを実装して、独自のポリシーを実装することもできます。

29.4. メッセージ再分配

メッセージ再分配では、コンシューマーのないキューから、クラスター内で、一致するコンシューマーを持つ他のノードにメッセージを自動的に再分配するように JBoss EAP メッセージングを設定できます。この機能を有効にするには、クラスター接続の **message-load-balancing-type** を **ON_DEMAND** (デフォルト値) に設定する必要があります。これは、以下の管理 CLI コマンドを使用して設定できます。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:write-attribute(name=message-load-balancing-type,value=ON_DEMAND)
```

メッセージ再分配は、キューの最後のコンシューマーが閉じられた直後に起動するように、または、キューの最後のコンシューマーが閉じられてから再分配するまで、設定可能な遅延時間を待機するように設定できます。これは **redistribution-delay** 属性を使用して設定します。

redistribution-delay 属性を使用して、キューで最後のコンシューマーが閉じられてから、一致するコンシューマーを持つクラスターの他のノードにそのキューからメッセージを再分配するまでの待機時間 (ミリ秒) を設定します。**-1** (デフォルト値) は、メッセージが再分配されないことを意味します。値が **0** の場合は、メッセージが即座に再分配されることを意味します。

デフォルトの JBoss EAP 設定の **address-setting** は **redistribution-delay** 値を **1000** に設定します。これは、メッセージを再分配する前に 1000 ミリ秒待機することを意味します。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <address-setting name="#" redistribution-delay="1000" message-counter-history-day-limit="10"
page-size-bytes="2097152" max-size-bytes="10485760" expiry-address="jms.queue.ExpiryQueue"
dead-letter-address="jms.queue.DLQ"/>
    ...
  </server>
</subsystem>
```

コンシューマーが閉じても、同じキューに別のコンシューマーがすぐに作成されることがよくあるため、多くの場合、再分配前に遅延を導入するのが理にかなっています。この場合、新しいコンシューマーがすぐに到達するため、即座の再分配は好ましくないでしょう。

以下は、「jms」から始まるアドレスにバインドされているキューまたはトピックに対して **0** の **redistribution-delay** を設定する **address-setting** の例です。この場合、メッセージは即座に再分配されます。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <address-setting name="jms.#" redistribution-delay="0"/>
```

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

このアドレス設定は、以下の管理 CLI コマンドを使用して追加できます。

```
/subsystem=messaging-activemq/server=default/address-setting=jms.#:add(redistribution-
delay=1000)
```

29.5. クラスター化されたメッセージのグループ化

重要

この機能はサポートされていません。

クラスター化されたグループ化では、通常のメッセージのグループ化に対して異なる方法を実行します。クラスターの中では、特定のグループ ID を持つメッセージグループが、ノードのいずれかに到達できます。ノードでは、どのノードの、どのコンシューマーに、どのグループ ID がバインドされるかを判断することが重要です。各ノードは、メッセージグループがデフォルトで到達する場所に関係なく、それらのグループ ID を処理するコンシューマーを持つノードにメッセージグループを正しくルーティングする役割を担います。指定したグループ ID を持つメッセージが、クラスター内の指定したノードに接続されている特定のコンシューマーに送信されます。この場合、これらのメッセージはコンシューマーが切断されても別のノードに送信されることはありません。

この状況は、グルーピングハンドラーによって処理されます。各ノードにグルーピングハンドラーがあります。また、このグルーピングハンドラー (と他のハンドラー) は、メッセージグループを正しいノードにルーティングする役割を担います。グルーピングハンドラーには、**LOCAL** と **REMOTE** の 2 つのタイプがあります。

ローカルハンドラーは、メッセージグループが取るルートを決定する役割を担います。リモートハンドラーはローカルハンドラーと通信し、適切に動作します。各クラスターは、特定のノードを選択してローカルグルーピングハンドラー指定するようにします。また、その他のすべてのノードには、リモートハンドラー指定する必要があります。

警告

メッセージのグループ化がクラスター内で使用された場合は、リモートグルーピングハンドラーとして設定されたノードで障害が発生すると、中断します。リモートグルーピングハンドラーのバックアップの設定では、これは修正されません。

最初にメッセージグループを受信するノードが、通常のクラスタールーティング条件 (ラウンドロビンキューを利用できるか) に基づいてルーティングの決定を行います。ノードは、この決定をそれぞれのグルーピングハンドラーに提案します。グルーピングハンドラーが受け入れた場合は、提案されたキューにメッセージをルーティングします。

グルーピングハンドラーが提案を拒否した場合は、他のルートを提案し、それに応じてルーティングを行います。その他のノードも先例に従い、選択したキューにメッセージグループを転送します。メッセージがキューに到達すると、そのキューのカスタマーに固定されます。

管理 CLI を使用してグルーピングハンドラーを設定できます。以下のコマンドは、**news.europe.#** アドレスを使用して **LOCAL** グルーピングハンドラーを追加します。

```
/subsystem=messaging-activemq/server=default/grouping-handler=my-group-handler:add(grouping-handler-address="news.europe.#",type=LOCAL)
```

サーバーをリロードする必要があります。

```
reload
```

以下の表は、**grouping-handler** の設定可能な属性を示しています。

属性	説明
group-timeout	**REMOTE** ハンドラーでは、この値は、ルートが使用されたことを **REMOTE** が **LOCAL** に通知する頻度を指定します。**LOCAL** ハンドラーでは、指定された時間でルートが使用されていない場合、削除されます。また、新しいパスが確立される必要があります。値はミリ秒単位です。
grouping-handler-address	クラスター接続と、使用するアドレスへの参照。
reaper-period	タイムアウトグループバインディングのチェックを実行するためにリーパーが実行される頻度 (**LOCAL** ハンドラーにのみ有効)。
timeout	処理が決定されるまで待機する時間。このタイムアウトに達すると、順序付けが厳格に保持された状態で、送信中に例外がスローされます。
type	ハンドラーが、処理の決定を行うクラスターの単一のローカルハンドラーであるか、ローカルハンドラーと対話するリモートハンドラーであるか。使用できる値は **LOCAL** または **REMOTE** です。

29.5.1. クラスター化されたメッセージのグループ化のベストプラクティス

クラスター化されたグループ化のベストプラクティスには、以下のものがあります。

- コンシューマーを定期的に作成して閉じる場合は、コンシューマーが別のノードに均等に分散されていることを確認します。キューが固定されると、キューからカスタマーを削除しなくても、そのキューにメッセージが自動的に転送されます。
- メッセージグループがバインドされているキューを削除する場合は、メッセージを送信しているセッションによってキューが削除されていることを確認してください。これを実行することで、他のノードは削除後にこのキューにメッセージをルーティングしないようにします。
- フェイルオーバーメカニズムとして、ローカルグルーピングハンドラーを持つノードを常に複製します。

29.6. メッセージングクラスターの起動と停止

JBoss EAP 7.4 サーバーを設定して ActiveMQ Artemis クラスターを形成する場合は、実行中のクラスター化サーバーに接続されているその他のサーバーとクライアントが存在する可能性があります。接続されているクライアントおよびサーバーを最初にシャットダウンしてから、クラスター内で実行してい

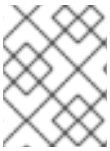
る JBoss EAP 7.4 サーバーをシャットダウンすることが推奨されます。これは順番に行う必要があり、サーバーがすべての接続を切断するのに十分な時間を設け、切断の際に、一貫性のない状態につながる可能性のある障害が発生するのを回避するためにも、並行して行ってはいけません。ActiveMQ Artemis はクラスターノードの自動スケールダウンに対応しておらず、すべてのクラスターノードが再起動されることを想定しています。

サーバーを起動するときにも同じことが当てはまります。ActiveMQ Artemis クラスターの JBoss EAP 7.4 サーバーを最初に起動する必要があります。起動が完了すれば、そのクラスターに接続する他のサーバーおよびクライアントを起動できます。

第30章 高可用性

高可用性とは、1つまたは複数のサーバーで障害が発生した後もシステムが機能を継続できることです。

高可用性の一部がフェイルオーバーです。これは、サーバーで障害が発生した場合に、クライアントアプリケーションが動作を継続できるように、クライアント接続をサーバー間で移行する機能です。

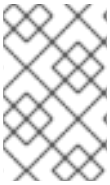


注記

永続メッセージデータのみが、フェイルオーバー後も存続します。フェイルオーバー後、非永続メッセージデータは使用できなくなります。

30.1. ライブ/バックアップのペア

JBoss EAP 7 メッセージングにより、各ライブサーバーがバックアップを持つライブ/バックアップペアとしてサーバーをリンクできます。ライブサーバーはクライアントからメッセージを受信し、バックアップサーバーはフェイルオーバーが発生するまで機能しません。バックアップサーバーは1つのライブサーバーのみが所有でき、パッシブモードで留まり、ライブサーバーの作業の引き継ぎを待ちます。



注記

ライブサーバーとバックアップサーバーの間には1対1の関係があります。ライブサーバーにはバックアップサーバーを 1つだけ含めることができ、バックアップサーバーは 1つのライブサーバーのみが所有できます。

ライブサーバーがクラッシュするか、正しいモードで停止した場合、現在パッシブモードのバックアップサーバーが新しいライブサーバーになります。新しいライブサーバーが、自動フェイルバックを許可するように設定されている場合は、古いライブサーバーが復旧することを検出して自動的に停止し、古いライブサーバーがメッセージの受信を再び開始できるようにします。

注記

ライブ/バックアップサーバーのペアを1つだけデプロイする場合は、バックアップインスタンスがメッセージをアクティブに処理していないため、ペアの前でロードバランサーを効果的に使用できません。さらに、JNDI や Undertow Web サーバーなどのサービスは、バックアップサーバーでもアクティブではありません。これらの理由により、バックアップメッセージングサーバーとして使用されている JBoss EAP のインスタンスへの JEE アプリケーションのデプロイはサポートされていません。

30.1.1. ジャーナルの同期

HA をレプリケートされたジャーナルで設定する場合は、ライブサーバーとバックアップの同期に時間がかります。

同期が完了したかどうかを確認するには、CLI で以下のコマンドを実行します。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:read-
attribute(name=synchronized-with-backup)
```

結果が **true** であれば、同期は完了です。

ライブサーバーを安全にシャットダウンできるかどうかを確認するには、CLI で以下のコマンドを送信します。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-slave:read-
attribute(name=synchronized-with-live)
```

結果が **true** であれば、ライブサーバーをシャットダウンしても安全です。

30.2. HA ポリシー

JBoss EAP メッセージングは、サーバーをバックアップする 2 つの異なるストラテジーであるレプリケーションと共有ストアをサポートしています。**server** 設定要素の **ha-policy** 属性を使用して、指定したサーバーに選択したポリシーを割り当てます。**ha-policy** には、4 つの有効な値があります。

- **replication-master**
- **replication-slave**
- **shared-store-master**
- **shared-store-slave**

ご覧のとおり、この値はサーバーがデータのレプリケーションまたは共有ストア の ha ポリシーを使用するかどうか、マスターまたはスレーブの役割を引き受けるかどうかを指定します。

管理 CLI を使用して、選択したサーバーに **ha-policy** を追加します。

注記

以下の例は、**standalone-full-ha** 設定プロファイルを使用して JBoss EAP を実行していることを前提とします。

```
/subsystem=messaging-activemq/server=SERVER/ha-policy=POLICY:add
```

たとえば、以下のコマンドを使用して **replication-master** ポリシーを**デフォルト**のサーバーに追加します。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:add
```

replication-master ポリシーはデフォルト値で設定されます。ポリシーの追加時に、デフォルトの設定をオーバーライドする値を含めることができます。現在の設定を読み取る管理 CLI コマンドは、以下の基本構文を使用します。

```
/subsystem=messaging-activemq/server=SERVER/ha-policy=POLICY:read-resource
```

たとえば、以下のコマンドを使用して、上記を **default** のサーバーに追加された **replication-master** ポリシーの現在の設定を読み取ります。また、デフォルト設定を強調表示するために出力が含まれています。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:read-resource

{
    "outcome" => "success",
```

```
    "result" => {
        "check-for-live-server" => true,
        "cluster-name" => undefined,
        "group-name" => undefined,
        "initial-replication-sync-timeout" => 30000L
    }
}
```

各ポリシーで利用可能な設定オプションの詳細については、「データのレプリケーション」と「共有ストア」を参照してください。

30.3. データのレプリケーション

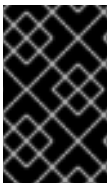
レプリケーションを使用すると、ライブとバックアップサーバーのペアが同じデータディレクトリーを共有せず、すべてのデータ同期がネットワーク経由で行われます。したがって、ライブサーバーが受信したすべての (永続的な) データがバックアップに複製されます。

ライブサーバーが正常にシャットダウンされると、バックアップサーバーがアクティブになり、クライアントはバックアップにフェイルオーバーします。この動作は事前に決定し、データのレプリケーション使用時に設定することはできません。

バックアップサーバーは、最初にライブサーバーのすべての既存データを同期してから置き換える必要があります。したがって、共有ストレージとは異なり、複製バックアップは、起動直後に完全に機能する訳ではありません。同期にかかる時間は、同期されるデータ量とネットワーク速度によって異なります。また、バックアップが起動すると、クライアントは **initial-replication-sync-timeout** の間ブロックされることに注意してください。このタイムアウトが過ぎると、同期が完了しない場合でもクライアントのブロックが解除されます。

フェイルオーバーが成功すると、バックアップのジャーナルは、ライブサーバーのデータよりも新しいデータの保持を開始します。フェイルバックを実行し、再起動後にライブサーバーになるように元のライブサーバーを設定できます。フェイルバックは、ライブサーバーがオンラインに戻る前に、バックアップサーバーとライブサーバーの間でデータを同期します。

両方のサーバーがシャットダウンしている場合、管理者はどのサーバーのジャーナルに最新データがあるかを判断する必要があります。バックアップジャーナルに最新のデータがある場合は、そのジャーナルをライブサーバーにコピーします。最新データがない場合は、再度アクティブになるたびに、バックアップはライブサーバーから古いジャーナルデータを複製し、独自のジャーナルデータを削除します。ライブサーバーのデータが最新の場合、アクションは不要で、サーバーを正常に起動できます。



重要

レイテンシーが高く、データセンター間のネットワークが信頼できない可能性があるため、データセンター間の高可用性に複製ジャーナルの設定と使用は推奨されず、サポートされていません。

ライブとバックアップのペアの複製はクラスターの一部である必要があります。**cluster-connection** 設定要素は、バックアップサーバーがそのライブ一致を見つける方法を定義します。

レプリケーションには、ネットワーク分離のリスクを低減するために少なくとも 3 つ以上のライブ/バックアップのペアが必要ですが、このリスクはなくす。3 つ以上のライブ/バックアップペアを使用する場合、クラスターはクォーラムと割り当てを使用して 2 つのライブブローカーを使用できます。

cluster-connection を設定する場合は、以下の詳細を確認してください。

- ライブサーバーとバックアップサーバーはいずれも同じクラスター内に含まれる必要があります。単純なライブ/バックアップ複製ペアでも、クラスター設定が必要なことに注意してください。
- クラスターユーザーとパスワードは、ペアの各サーバーで一致している必要があります。

<master> と <slave> **要素** の両方で **group-name** 属性を設定して、ライブ/バックアップサーバーのペアを指定します。バックアップサーバーは、同じ **group-name** を共有するライブサーバーにのみ接続します。

group-name を使用する例として、3 つのライブサーバーと 3 つのバックアップサーバーがあるとします。ライブサーバーはそれぞれ独自のバックアップとペアする必要があるため、以下のグループ名を割り当てます。

- live1 および backup1 は、**pair1** の **group-name** を使用します。
- live2 および backup2 は、**pair2** の **group-name** を使用します。
- live3 と backup3 は、**group-name** の **pair3** を使用します。

この例では、サーバー **backup1** は同じ **group-name** (**pair1**)でライブサーバーを検索します。この場合は、サーバーは **live1** です。

共有ストアの場合と同様に、ライブサーバーが停止またはクラッシュすると、複製するペアのバックアップがアクティブになり、その役割を引き継ぎます。特に、ライブサーバーへの接続が失われると、ペアのバックアップがアクティブになります。これは、一時的なネットワークの問題が原因でも発生することがあるため、問題となる可能性があります。この問題に対処するために、ペアのバックアップは、クラスター内の他のサーバーにまだ接続できるかどうかを判断しようとします。半分を超えるサーバーに接続できる場合、アクティブになります。ライブサーバーのほか、クラスター内の半分を超える他のサーバーとの通信が失われた場合、ペアのバックアップは待機し、ライブサーバーとの再接続を試みます。これにより、バックアップサーバーとライブサーバーの両方がメッセージを処理し、一方がそれに気付かない「スプリットブレイン」状況のリスクが軽減されます。

重要

これは、ライブサーバーが見つからず、ジャーナルのファイルロックが解放された場合にバックアップがアクティブになり、クライアント要求の処理を開始する共有ストアバックアップとの重要な違いです。また、レプリケーションでは、バックアップサーバーは、保持している可能性のあるデータが最新であるかどうかがわからないため、自動的にアクティブ化することを決定できないことに注意してください。保持しているデータを使用して複製バックアップサーバーをアクティブにするには、管理者はスレーブをマスターに変更することで、設定を変更してライブサーバーにする必要があります。

参考情報

- クラスター接続の設定

30.3.1. データのレプリケーションの設定

以下の 2 つの例は、**my-cluster** という名前のクラスターと **group1** という名前のバックアップグループに存在するライブサーバーとバックアップサーバーの両方の基本設定を示しています。

以下の手順では、管理 CLI を使用して **my-cluster** という名前のクラスターと **group1** という名前のバックアップグループに存在するライブサーバーとバックアップサーバーの両方の基本設定を提供します。

注記

以下の例は、**standalone-full-ha** 設定プロファイルを使用して JBoss EAP を実行していることを前提とします。

管理 CLI コマンドを使用して、データレプリケーションにライブサーバーを設定する

1. **ha-policy** をライブサーバーに追加します

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(check-for-live-server=true,cluster-name=my-cluster,group-name=group1)
```

check-for-live-server 属性は、指定された ID を持つ他のサーバーがクラスター内にないことを確認するようにライブサーバーに指示します。JBoss EAP 7.0 では、この属性のデフォルト値は **false** でした。JBoss EAP 7.1 以上では、デフォルト値は **true** です。

2. **ha-policy** をバックアップサーバーに追加します

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-slave:add(cluster-name=my-cluster,group-name=group1)
```

3. 共有された **cluster-connection** が存在することを確認します。
ライブサーバーとバックアップサーバーの間の適切な通信には **cluster-connection** が必要です。以下の管理 CLI コマンドを使用して、同じ **cluster-connection** がライブサーバーとバックアップサーバーの両方に設定されていることを確認します。この例では、**standalone-full-ha** 設定プロファイルにあるデフォルトの **cluster-connection** を使用しています。ほとんどの場合、これで十分なはずです。クラスター接続の設定方法の詳細は、「クラスター接続の設定」を参照してください。

以下の管理 CLI コマンドを使用して、ライブおよびバックアップサーバーの両方で同じ cluster-connection を使用していることを確認します。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:read-resource
```

cluster-connection が存在する場合、出力は現在の設定を提供します。存在しない場合は、エラーメッセージが表示されます。

すべての設定属性の詳細は、「すべてのレプリケーション設定」を参照してください。

30.3.2. すべてのレプリケーション設定

管理 CLI を使用して、追加された後、ポリシーに設定を追加できます。これを実行するコマンドは、以下の基本構文に従います。

```
/subsystem=messaging-activemq/server=default/ha-policy=POLICY:write-attribute(name=ATTRIBUTE,value=VALUE)
```

たとえば、**restart-backup** 属性の値を **true** に設定するには、以下のコマンドを使用します。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-slave:write-attribute(name=restart-backup,value=true)
```

以下の表は、**replication-master** ノードと **replication-slave** 設定要素の HA 設定属性を示しています。

表30.1 replication-master の属性

属性	説明
check-for-live-server	**true** に設定して、起動時に同じサーバー ID を使用して別のサーバーのクラスターをチェックするようにこのサーバーに指示します。JBoss EAP 7.0 のデフォルト値は **false** です。JBoss EAP 7.1 以降のデフォルト値は **true** です。
cluster-name	レプリケーションに使用されるクラスターの名前。
group-name	設定されている場合、バックアップサーバーは、**group-name** が一致するライブサーバーとのみペアになります。
initial-replication-sync-timeout	開始レプリケーションが同期されるまでの待機時間 (ミリ秒単位)。デフォルトは **30000** です。
synchronized-with-backup	ライブサーバーとレプリケーションサーバーのジャーナルが同期されているかどうかを示します。

表30.2 replication-slave の属性

属性	説明
allow-failback	別のサーバーがこのサーバーを引き継ぐ要求を送信したときに、このサーバーが自動的に停止するかどうか。通常のユースケースは、再起動または障害復旧後に、ライブサーバーがアクティブなプロセスの再開を要求する場合です。**allow-failback** が **true** に設定されたバックアップサーバーは、クラスターに再参加してプロセスの再開を要求すると、ライブサーバーに移行します。デフォルトは **true** です。
cluster-name	レプリケーションに使用されるクラスターの名前。
group-name	設定されている場合、バックアップサーバーは、**group-name** が一致するライブサーバーとのみペアになります。
initial-replication-sync-timeout	開始レプリケーションが同期されるまでの待機時間 (ミリ秒単位)。デフォルトは **30000** です。
max-saved-replicated-journal-size	起動時にファイルを移動した後に、レプリケートされたバックアップサーバーが再起動できる回数を指定します。最大値に達した後、サーバーはフェイルバックすると永久に停止します。デフォルトは **2** です。

属性	説明
restart-backup	**true** に設定して、フェイルバックが原因で停止したら再起動するようにこのバックアップサーバーに指示します。デフォルトは **true** です。
synchronized-with-live	レプリケーションサーバーのジャーナルがライブサーバーと同期されているかどうか (ライブサーバーを安全にシャットダウンできるかどうか) を示します。

30.3.3. クラスター接続タイムアウトの防止

ライブとバックアップのペアはそれぞれ **cluster-connection** を使用して通信します。**cluster-connection** の **call-timeout** 属性は、クラスターの別のサーバーへの呼び出し後にサーバーが応答を待機する時間を設定します。**call-timeout** のデフォルト値は 30 秒で、ほとんどの場合、これで十分です。ただし、バックアップサーバーが、ライブサーバーから送信されるレプリケーションパケットを処理できない状況があります。これは、たとえば、ディスク操作が遅いため、または **journal-min-files** の値が大きいために、ジャーナルファイルの最初の事前作成に時間がかかりすぎる場合に発生することがあります。このようなタイムアウトが発生すると、ログに次のような行が表示されます。

```
AMQ222207: The backup server is not responding promptly introducing latency beyond the limit. Replication server being disconnected now.
```

警告

上記のような行がログに表示された場合は、レプリケーションのプロセスが停止していることを意味します。レプリケーションを再度開始するには、バックアップサーバーを再起動する必要があります。

クラスター接続のタイムアウトを防ぐには、以下のオプションを考慮してください。

- **cluster-connection** の **call-timeout** を増やします。詳細は、「クラスター接続の設定」を参照してください。
- **journal-min-files** の値を減らします。詳細は、「永続性の設定」を参照してください。

30.3.4. 古いジャーナルディレクトリーの削除

バックアップサーバーは、ライブサーバーとの同期を開始すると、ジャーナルを新しい場所に移動します。デフォルトでは、ジャーナルディレクトリーは **EAP_HOME/standalone** の **data/activemq** ディレクトリーに置かれます。ドメインの場合、各サーバーは **EAP_HOME/domain/servers** の下に独自の **serverX/data/activemq** ディレクトリーを持っています。ディレクトリーの名前は、**bindings**、**journal**、**largemessages**、**paging** です。これらのディレクトリーの詳細は、「永続性の設定」と「ページングの設定」を参照してください。

移動すると、新しいディレクトリーの名前が **oldreplica.X** に変更されます。**X** は数字の接尾辞です。新しいフェイルオーバーにより別の同期が開始されると、「moved」ディレクトリーの接尾辞が 1 増えます。たとえば、最初の同期では、ジャーナルディレクトリーは **oldreplica.1** に移動され、2 回目は **oldreplica.2**、といったように移動されます。元のディレクトリーは、ライブサーバーから同期したデータを保存します。

デフォルトでは、バックアップサーバーは、フェイルオーバーとフェイルバックの 2 つの発生を管理するように設定されています。その後、クリーンアッププロセスがトリガーされ、**oldreplica.X** ディレクトリーが削除されます。バックアップサーバーの **max-saved-replicated-journal-size** 属性を使用して、クリーンアッププロセスをトリガーするフェイルオーバーの発生回数を変更できます。

注記

ライブサーバーには、**max-saved-replicated-journal-size** が **2** に設定されます。この値は変更できません

30.3.5. 専用ライブサーバーとバックアップサーバーの更新

ライブサーバーとバックアップサーバーが専用のトポロジーにデプロイされ、そこで各サーバーが JBoss EAP の独自のインスタンスで実行されている場合は、以下の手順に従ってクラスターのスムーズな更新と再起動を行ってください。

1. バックアップサーバーを正常にシャットダウンします。
2. ライブサーバーを正常にシャットダウンします。
3. ライブサーバーとバックアップサーバーの設定を更新します。
4. ライブサーバーを起動します。
5. バックアップサーバーを起動します。

30.3.6. データのレプリケーションの制限: スプリットブレイン処理

「スプリットブレイン」状態は、ライブサーバーとそのバックアップの両方が同時にアクティブなときに発生します。両方のサーバーがクライアントおよびプロセスメッセージを処理し、一方はそれに気付きません。この場合、ライブサーバーとバックアップサーバーの間でメッセージのレプリケーションは行われなくなります。2 つのサーバー間でネットワーク障害が発生すると、スプリット状況が発生する可能性があります。

たとえば、ライブサーバーとネットワークルーター間の接続が切断されると、バックアップサーバーはライブサーバーへの接続を失います。ただし、バックアップはクラスター内の半分以上のサーバーに引き続き接続できるため、アクティブになります。ライブ/バックアップのペアが 1 つのみで、バックアップサーバーがライブサーバーへの接続を失った場合にも、バックアップがアクティブになることに注意してください。両方のサーバーがクラスター内でアクティブな場合、2 つの望ましくない状況が発生する可能性があります。

1. リモートクライアントはバックアップサーバーにフェイルオーバーしますが、MDB などのローカルクライアントはライブサーバーを使用します。両方のノードでのジャーナルが完全に別になるため、スプリットブレイン処理が生じます。
2. ライブサーバーへの接続の切断は、リモートクライアントがバックアップサーバーにフェイルオーバーした後に修正されます。古いクライアントは引き続きバックアップを使用し続け、新しいクライアントはライブサーバーに接続するので、これもスプリットブレインの状況が生じます。

お客様は、ライブサーバーとバックアップサーバーの各ペア間に信頼できるネットワークを実装して、データのレプリケーション使用時のスプリットブレイン処理のリスクを軽減する必要があります。たとえば、複製したネットワークインターフェースカードや、その他のネットワーク冗長性を使用します。

30.4. 共有ストア

このスタイルの高可用性は、ライブノードとバックアップノードの両方からアクセス可能な共有ファイルシステムを必要とするという点で、データのレプリケーションとは異なります。つまり、サーバーのペアは、設定のページング、メッセージジャーナル、バインディングジャーナル、および大きいメッセージに同じ場所を使用します。

注記

共有ストアの使用は Windows ではサポートされません。Red Hat バージョンの GFS2 または NFSv4 を使用する場合、Red Hat Enterprise Linux でサポートされます。さらに、GFS2 は ASYNCIO ジャーナルタイプでのみサポートされますが、NFSv4 は ASYNCIO と NIO の両方のジャーナルタイプでサポートされます。

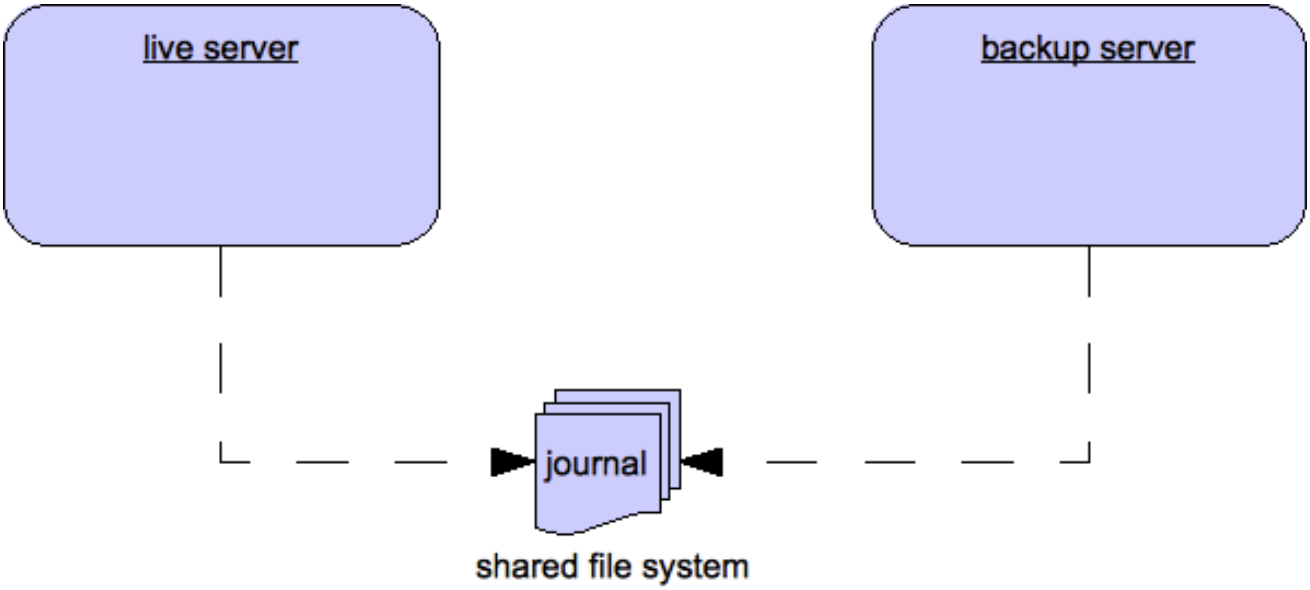
また、クラスターに含まれていない場合でも、ペアの各参加サーバー (ライブとバックアップ) には **cluster-connection** が定義されている必要があります。これは、**cluster-connection** で、バックアップサーバーがその存在をライブサーバーと他のすべてのノードに通知する方法を定義するからです。これがどのように実行されるかについては、「クラスター接続の設定」を参照してください。

フェイルオーバーが発生し、バックアップサーバーが引き継ぐ場合、クライアントが接続できるようになるには、その前に、共有ファイルシステムから永続ストレージを読み込む必要があります。このスタイルの高可用性は、ライブとバックアップの両方のペアからアクセス可能な共有ファイルシステムを必要とするという点で、データのレプリケーションとは異なります。これは通常、ある種の高パフォーマンスストレージエリアネットワーク (SAN) になります。Red Hat は、ストレージソリューションに NAS と呼ばれるネットワーク接続ストレージの使用を推奨していません。

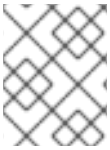
共有ストアの高可用性の利点は、ライブノードとバックアップノード間でレプリケーションが発生しないことです。これは、通常の操作中にレプリケーションのオーバーヘッドによるパフォーマンスの低下が発生しないことを意味します。

共有ストアのレプリケーションの欠点は、バックアップサーバーがアクティブになったときに、共有ストアからジャーナルをロードする必要があり、ストア内のデータ量によっては時間がかかる場合があることです。また、JBoss EAP でサポートされる共有ストレージソリューションが必要です。

通常の操作中に最高のパフォーマンスが必要な場合、Red Hat では、高性能な SAN にアクセスし、若干遅いフェイルオーバーコストを受け入れることを推奨しています。正確なコストは、データ量によって異なります。



30.4.1. 共有ストアの設定



注記

以下の例は、**standalone-full-ha** 設定プロファイルを使用して JBoss EAP を実行していることを前提とします。

1. **ha-policy** をライブサーバーに追加します。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-master:add
```

2. **ha-policy** をバックアップサーバーに追加します。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:add
```

3. 共有された **cluster-connection** が存在することを確認します。
ライブサーバーとバックアップサーバーの間の適切な通信には **cluster-connection** が必要です。以下の管理 CLI コマンドを使用して、同じ **cluster-connection** がライブサーバーとバックアップサーバーの両方に設定されていることを確認します。この例では、**standalone-full-ha** 設定プロファイルにあるデフォルトの **cluster-connection** を使用しています。ほとんどの場合、これで十分なはずです。クラスター接続の設定方法の詳細は、「クラスター接続の設定」を参照してください。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:read-resource
```

cluster-connection が存在する場合、出力は現在の設定を提供します。存在しない場合は、エラーメッセージが表示されます。

共有ストアポリシーのすべての設定属性の詳細は、「すべての共有ストア設定」を参照してください。

30.4.2. すべての共有ストア設定

管理 CLI を使用して、追加されたポリシーに設定を追加します。これを実行するコマンドは、以下の基本構文に従います。

```
/subsystem=messaging-activemq/server=default/ha-policy=POLICY:write-
attribute(name=ATTRIBUTE,value=VALUE)
```

たとえば、**restart-backup** 属性の値を **true** に設定するには、以下のコマンドを使用します。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-
attribute(name=restart-backup,value=true)
```

表30.3 shared-store-master 設定要素の属性

属性	説明
failover-on-server-shutdown	**true** に設定して、通常シャットダウンしたときにフェイルオーバーするようにこのサーバーに指示します。デフォルトは **false** です。

表30.4 shared-store-slave 設定要素の属性

属性	説明
allow-failback	**true** に設定して、別のサーバーがこのサーバーを引き継ぐ要求を送信したときに、自動的に停止するようにこのサーバーに指示します。ユースケースは、通常サーバーが停止し、そのバックアップが役割を引き継ぎ、その後、メインサーバーが再起動し、サーバー (元のバックアップ) の動作停止を要求する場合です。デフォルトは **true** です。
failover-on-server-shutdown	**true** に設定して、通常シャットダウンしたときにフェイルオーバーするようにこのサーバーに指示します。デフォルトは **false** です。
restart-backup	**true** に設定して、フェイルバックまたはスケールダウンが原因で停止したら再起動するようにこのサーバーに指示します。デフォルトは **true** です。

30.5. ライブサーバーへのフェイルバック

ライブサーバーに障害が発生し、バックアップがその役割を引き継いだ後は、ライブサーバーを再起動して、クライアントをフェイルバックさせることができます。

共有ストアの場合は、単に元のライブサーバーを再起動し、プロセス自体を強制終了して新しいライブサーバーを強制終了します。または、スレーブで **allow-fail-back** を **true** に設定して、マスターがオンラインに戻ったら自動的に停止するように強制できます。**allow-fail-back** を設定する管理 CLI コマンドは以下のようになります。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-
attribute(name=allow-fail-back,value=true)
```

レプリケーション HA モードでは、マスター設定で **check-for-live-server** 属性が **true** に設定されていることを確認する必要があります。JBoss EAP 7.1 より、これがデフォルト値になります。

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:write-attribute(name=check-for-live-server,value=true)
```

true に設定すると、ライブサーバーは、その nodeID を使用して別のサーバーの起動時にクラスターを検索します。見つかった場合は、このサーバーにアクセスし、「フェイルバック」を試行します。これはリモートのレプリケーションのシナリオであるため、元のライブサーバーは、その ID で実行しているバックアップとデータを同期する必要があります。同期されると、アクティブな処理を引き継ぐことができるようにするため、シャットダウンするようにバックアップサーバーに要求します。この動作により、元のライブサーバーは、フェイルオーバーがあったかどうかを判断して、フェイルオーバーがあった場合は、その役割を果たしたサーバーがまだ実行中かどうかを判断できます。

警告

バックアップへのフェイルオーバーが発生した後にライブサーバーを再起動する場合は、**check-for-live-server** 属性を **true** に設定する必要があります。再起動しない場合は、バックアップサーバーが実行していることを確認せずに、ライブサーバーが直ちに起動します。これにより、ライブとバックアップが同時に実行され、新しく接続されたすべてのクライアントに重複メッセージが配信されるようになります。

共有ストアの場合、通常のサーバーシャットダウンでフェイルオーバーを発生させることができ、次のようにマスターまたはスレーブのいずれかの HA 設定でこの **failover-on-server-shutdown** を **true** に設定することもできます。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-attribute(name=failover-on-server-shutdown,value=true)
```

また、**allow-failback** を **true** に設定することにより、元のライブサーバーが復旧したときに実行中のバックアップサーバーを強制的にシャットダウンし、元のライブサーバーが自動的に引き継げるようにできます。

```
/subsystem=messaging-activemq/server=default/ha-policy=shared-store-slave:write-attribute(name=allow-failback,value=true)
```

30.6. コロケートバックアップサーバー

JBoss EAP では、バックアップメッセージングサーバーを、別のライブサーバーと同じ JVM にコロケートすることもできます。たとえば、各ライブサーバーが他のバックアップをコロケートするスタンドアロンサーバーの単純な 2 ノードクラスターの例を考えましょう。このようにサーバーをコロケートする場合、共有ストアまたは複製された HA ポリシーのいずれかを使用できます。コロケートするようにメッセージングサーバーを設定する際に覚えておくべき 2 つの重要なことがあります。

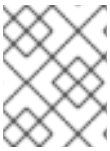
第 1 に、設定の各 **server** 要素には、独自の **remote-connector** および **remote-acceptor**、または **http-connector** および **http-acceptor** が必要です。たとえば、**remote-acceptor** を使用するライブサーバーは、ポート **5445** でリッスンするよう設定できますが、コロケートされたバックアップからの **remote-acceptor** はポート **5446** を使用します。ポートは、デフォルトの **socket-binding-group** に追加する必要がある **socket-binding** 要素に定義します。**http-acceptors** の場合、ライブおよびコロケートバック

アップは、同じ **http-listener** を共有できます。各 **server** 設定のクラスター関連の設定要素は、サーバーによって使用される **remote-connector** または **http-connector** を使用します。関連する設定が、後述の各例に含まれています。

第 2 に、ジャーナル関連のディレクトリーのパスを適切に設定する点に注意します。たとえば、共有ストアのコロケートトポロジーでは、ライブサーバーとそのバックアップの両方が、別のライブサーバーにコロケーションされ、バインディングとメッセージジャーナル用、大きいメッセージ用、ページング用にディレクトリーの場所を共有するように設定する必要があります。

30.6.1. コロケート HA トポロジーの手動作成の設定

以下の手順で使用する管理 CLI コマンドの例は、コロケートトポロジーを使用する単純な 2 ノードクラスターを設定する方法を示しています。この例では、2 ノードのコロケートクラスターを設定します。ライブサーバーとバックアップサーバーは各ノードに存在します。**ノード 1** のコロケートバックアップは、**ノード 2** のコロケートされたライブサーバーとペアになり、**ノード 2** のバックアップサーバーは **ノード 1** のライブサーバーとペアになります。共有ストアとデータレプリケーション HA ポリシーの両方の例が含まれます。



注記

以下の例は、**full-ha** 設定プロファイルを使用して JBoss EAP を実行していることを前提としています。

1. 各インスタンスのデフォルトのサーバーを HA ポリシーを使用するように変更します。各ノードのデフォルトサーバーはライブサーバーになります。従う手順は、共有ストアポリシーまたははデータレプリケーションポリシーを設定したかどうかによって異なります。

   - **共有ストアポリシーの説明:** 以下の管理 CLI コマンドを使用して、推奨 HA ポリシーを追加します。

     ```
     /subsystem=messaging-activemq/server=default/ha-policy=shared-store-master:add
     ```

   - **データレプリケーションポリシーの説明:** 各ノードのデフォルトサーバーは、一意の **group-name** を使用して設定する必要があります。以下の例では、1 番目のコマンドが **ノード 1** で実行され、2 番目が **ノード 2** で実行されます。

     ```
     /subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(cluster-name=my-cluster,group-name=group1,check-for-live-server=true)

     /subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(cluster-name=my-cluster,group-name=group2,check-for-live-server=true)
     ```

2. 新しいバックアップサーバーを各ライブサーバーとコロケートします。

   a. JBoss EAP の各インスタンスに新しいサーバーを追加して、デフォルトのライブサーバーとコロケートします。新しいサーバーは、他のノードでデフォルトのサーバーをバックアップします。以下の管理 CLI コマンドを使用して、**backup** という名前の新しいサーバーを作成します。

      ```
      /subsystem=messaging-activemq/server=backup:add
      ```

   b. 次に、推奨 HA ポリシーを使用するように新しいサーバーを設定します。従う手順は、共有ストアポリシーまたははデータレプリケーションポリシーを設定したかどうかによって異なります。

- **共有ストアポリシーの説明:** 以下の管理 CLI コマンドを使用して HA ポリシーを追加します。

```
/subsystem=messaging-activemq/server=backup/ha-policy=shared-store-slave:add
```

- **データレプリケーションポリシーの説明:** 他のノード上のライブサーバーの **group-name** を使用するようにバックアップサーバーを設定します。以下の例では、1 番目のコマンドが**ノード 1** で実行され、2 番目のコマンドが**ノード 2** で実行されます。

```
/subsystem=messaging-activemq/server=backup/ha-policy=replication-slave:add(cluster-name=my-cluster,group-name=group2)

/subsystem=messaging-activemq/server=backup/ha-policy=replication-slave:add(cluster-name=my-cluster,group-name=group1)
```

3. すべてのサーバーに対してディレクトリーの場所を設定します。
サーバーが HA 用に設定されたら、バインディングジャーナル、メッセージジャーナル、大きいメッセージのディレクトリーの場所を設定する必要があります。ページングを使用する計画がある場合は、ページングのディレクトリーも設定する必要があります。従う手順は、共有ストアポリシーまたはデータレプリケーションポリシーを設定したかどうかによって異なります。

- **共有ストアポリシーの説明:** **ノード 1** のライブサーバーの **path** 値は、**ノード 2** のバックアップサーバーとしてサポート対象のファイルシステムと同じ場所を指す必要があります。**ノード 2** のライブサーバーと**ノード 1** のバックアップでも同様です。

  a. 以下の管理 CLI コマンドを使用して、**ノード 1** のディレクトリーの場所を設定します。

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

/subsystem=messaging-activemq/server=default/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-A)

/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-A)

/subsystem=messaging-activemq/server=default/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-A)

/subsystem=messaging-activemq/server=backup/path=bindings-directory:write-attribute(name=path,value=/PATH/TO/shared/bindings-B)

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-B)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-B)

/subsystem=messaging-activemq/server=backup/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-B)
```

  b. 以下の管理 CLI コマンドを使用して、**ノード 2** のディレクトリーの場所を設定します。

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

/subsystem=messaging-activemq/server=default/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-B)

/subsystem=messaging-activemq/server=default/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-B)

/subsystem=messaging-activemq/server=default/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-B)

/subsystem=messaging-activemq/server=backup/path=bindings-directory:write-attribute(name=path,value=/PATH/TO/shared/bindings-A)

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=/PATH/TO/shared/journal-A)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=/PATH/TO/shared/largemessages-A)

/subsystem=messaging-activemq/server=backup/path=paging-directory:write-attribute(name=path,value=/PATH/TO/shared/paging-A)
```

- **データレプリケーションポリシーの説明:** 各サーバーは独自のディレクトリーを使用し、他のサーバーとは共有しません。以下のコマンド例では、**path** の場所の各値は、ファイルシステム上の一意の場所であると仮定します。ライブサーバーでは、デフォルトの場所を使用するため、ディレクトリーの場所を変更する必要はありません。ただし、バックアップサーバーは、依然として一意の場所で設定する必要があります。

  a. 以下の管理 CLI コマンドを使用して、**ノード 1** のディレクトリーの場所を設定します。

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

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=activemq/journal-B)

/subsystem=messaging-activemq/server=backup/path=large-messages-directory:write-attribute(name=path,value=activemq/largemessages-B)

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

  b. 以下の管理 CLI コマンドを使用して、**ノード 2** のディレクトリーの場所を設定します。

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

/subsystem=messaging-activemq/server=backup/path=journal-directory:write-attribute(name=path,value=activemq/journal-B)

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

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

4. 新しいアクセプターおよびコネクターをバックアップサーバーに追加します。
各バックアップサーバーは、**http-connector** と、デフォルトの **http-listener** を使用する **http-acceptor** を使用して設定する必要があります。これにより、サーバーが HTTP ポート経由で通信を送受信できるようになります。以下の例は、**http-acceptor** と **http-connector** をバックアップサーバーに追加します。

```
/subsystem=messaging-activemq/server=backup/http-acceptor=http-acceptor:add(http-listener=default)

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

5. バックアップサーバーの **cluster-connection** を設定します。
各メッセージングサーバーは、適切な通信に **cluster-connection**、**broadcast-group**、および **discovery-group** が必要です。以下の管理 CLI コマンドを使用して、これらの要素を設定します。

```
/subsystem=messaging-activemq/server=backup/broadcast-group=bg-group1:add(connectors=[http-connector],jgroups-cluster=activemq-cluster)

/subsystem=messaging-activemq/server=backup/discovery-group=dg-group1:add(jgroups-cluster=activemq-cluster)

/subsystem=messaging-activemq/server=backup/cluster-connection=my-cluster:add(connector-name=http-connector,cluster-connection-address=jms,discovery-group=dg-group1)
```

これで、コロケートサーバー設定が完了しました。

30.7. フェイルオーバーモード

JBoss EAP のメッセージングは、2 つのタイプのクライアントフェイルオーバーを定義します。

- 自動クライアントフェイルオーバー
- 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

また、JBoss EAP メッセージングでは、同じサーバーに対して 100% 透過的な自動再割り当てを提供します (一時的なネットワークの問題の場合など)。これは、同じサーバーに再接続される点を除くと、フェイルオーバーと似ています。詳細は、「クライアントの再接続とセッションの再割り当て」を参照してください。

フェイルオーバー中に、クライアントのコンシューマーが非永続的または一時的なキューにある場合、これらのキューはバックアップノード上でフェイルオーバー中に自動的に再作成されます。これは、バックアップノードが非永続キューを認識しないためです。

30.7.1. 自動クライアントフェイルオーバー

JBoss EAP メッセージングクライアントは、すべてのライブサーバーとバックアップサーバーの情報を受け取るように設定できます。そのため、クライアントで接続 (ライブサーバー接続) に失敗すると、ク

ライアントが障害を検出し、バックアップサーバーに再接続します。次に、バックアップサーバーはフェイルオーバーの前に各接続に存在したセッションおよびコンシューマーを自動的に再作成します。そのため、ユーザーは再接続ロジックを手動でコーディングする必要がなくなります。

JBoss EAP メッセージングクライアントは、「使用済みの接続の検出」で説明されているように、**client-failure-check-period** で指定した時間内にサーバーからパケットを受信しないと、接続障害を検出します。

割り当てられた時間内にクライアントがデータを受信しない場合、接続が失敗したと見なし、フェイルオーバーを試みます。ソケットがオペレーティングシステムによって閉じられているとします。たとえば、サーバーハードウェア自体がクラッシュしたのではなく、サーバープロセスが強制終了され可能性がり、クライアントはすぐにフェイルオーバーします。

JBoss EAP メッセージングクライアントは、さまざまな方法でライブサーバーとバックアップサーバーのペアのリストを検出できるように設定できます。たとえば、明示的なエンドポイントを使用して設定できますが、最も一般的な方法は、クライアントが初めてクラスターに接続するときにクラスタートポロジーについての情報を受け取ることです。詳細は、「サーバー検出」を参照してください。

デフォルトの HA 設定には、クラスター通信に推奨される **http-connector** を使用する **cluster-connection** が含まれます。これは、リモートクライアントがデフォルトの **RemoteConnectionFactory** を使用してサーバーに接続するときに使用する **http-connector** と同じです。推奨されていませんが、別のコネクターを使用することもできます。独自のコネクターを使用する場合は、リモートクライアントが使用する **connection-factory** とクラスターノードが使用する **cluster-connection** の両方の設定の一部に含まれていることを確認してください。コネクターとクラスター接続に関する詳細は、「メッセージングトランスポートの設定」と「クラスター接続」を参照してください。

警告

JMS クライアントにより使用される **connection-factory** に定義する **connector** は、クラスターにより使用される **cluster-connection** に定義されているものと同じである必要があります。同じでない場合、クライアントはベースとなるライブ/バックアップのペアのトポロジーを更新できないため、バックアップサーバーの場所を認識できません。

CLI コマンドを使用して、**connection-factory** と **cluster-connection** の両方の設定を確認します。たとえば、**RemoteConnectionFactory** という名前の **connection-factory** の現在の設定を読み取るには、以下のコマンドを使用します。

```
/subsystem=messaging-activemq/server=default/connection-
factory=RemoteConnectionFactory:read-resource
```

同様に、以下のコマンドは **my-cluster** という名前の **cluster-connection** の設定を読み込みます。

```
/subsystem=messaging-activemq/server=default/cluster-connection=my-cluster:read-resource
```

自動クライアントフェイルオーバーを有効にするには、ゼロでない再接続試行を許可するようにクライアントを設定する必要があります。詳細は、「クライアントの再接続とセッションの再割り当て」を参照してください。デフォルトでは、フェイルオーバーは、ライブサーバーへの接続が少なくとも 1 つ確立された後にのみ発生します。つまり、クライアントが、ライブサーバーへの最初の接続を確立できな

いと、フェイルオーバーは発生しません。クライアントは、最初の試行に失敗すると、**reconnect-attempts** プロパティーに従ってライブサーバーへ接続を再試行し、設定された試行回数の後に失敗します。

```
/subsystem=messaging-activemq/server=default/connection-factory=RemoteConnectionFactory:write-attribute(name=reconnect-attempts,value=<NEW_VALUE>)
```

このルールの例外は、ライブ/バックアップサーバーのペアが 1 つのみで、他のライブサーバーはなく、リモート MDB が正常にシャットダウンされたときにライブサーバーに接続される場合です。MDB が **@ActivationConfigProperty(propertyName = "rebalanceConnections", propertyValue = "true")** を設定している場合、別のライブサーバーへの接続を再度調整し、バックアップにフェイルオーバーしません。

初期接続時のフェイルオーバー
クライアントは最初の接続が確立されるまで完全なトポロジーについてわからないため、バックアップについて不明な時間枠があります。この時点で失敗した場合、クライアントは元のライブサーバーにのみ再接続を試みます。クライアントが試行する回数を設定するには、**ClientSessionFactoryImpl** または **ActiveMQConnectionFactory** でプロパティー **initialConnectAttempts** を設定します。

または、サーバー設定で、クライアントが使用する接続ファクトリーの **initial-connect-attempts** 属性を設定できます。これのデフォルトは **0** で、つまり、1 回のみ試行します。試行がその回数行われると、例外がスローされます。

```
/subsystem=messaging-activemq/server=default/connection-factory=RemoteConnectionFactory:write-attribute(name=initial-connect-attempts,value=<NEW_VALUE>)
```

サーバーレプリケーションについて
JBoss EAP メッセージングは、ライブサーバーとバックアップサーバーの間で完全なサーバー状態を複製しません。新しいセッションがバックアップ上に自動的に再作成される場合、そのセッション中に送信済みまたは確認応答済みのメッセージについては認識されません。フェイルオーバー時の in-flight の送信と確認応答も失われる可能性があります。

完全なサーバー状態を複製することで、JBoss EAP メッセージングは、理論的に 100% 透過的なシームレスなフェイルオーバーを提供して、メッセージや確認応答を失うことを防ぐことができます。ただし、これには、キューやセッションを含め、完全なサーバー状態を複製するため、多大なコストがかかります。これには、サーバー状態マシン全体のレプリケーションが必要です。つまり、ライブサーバー上のすべての操作は、一貫したレプリカ状態にするためにまったく同じグローバル順序でレプリカサーバーに複製される必要があります。これは、特に複数のスレッドがライブサーバー状態を同時に変更していることを考慮すると、パフォーマンスの優れたスケーラブルな方法で行うことは非常に困難です。

仮想同期などの手法を使用して完全な状態マシンのレプリケーションを提供することは可能です。しかし、これは拡張性が低く、すべての操作を単一スレッドに効果的にシリアライズするため、同時実行性が大幅に低下します。ロック状態の複製やスレッドスケジューリングの複製など、マルチスレッドのアクティブなレプリケーションには他の手法が存在しますが、これを Java レベルで達成することは非常に困難です。

したがって、100% 透過的なフェイルオーバーのために、パフォーマンスと同時実行性を低下させる価値はありませんでした。100% 透過的なフェイルオーバーがなくても、トランザクションの重複検出と再試行の組み合わせを使用することで、障害が発生した場合でも、一度だけの配信を保証するのは簡単です。ただし、これは、クライアントコードに対して 100% 透過的ということではありません。

30.7.1.1. フェイルオーバー中のブロック呼び出しの処理

クライアントコードがサーバーへのブロック呼び出しの中にある場合 (フェイルオーバー中に実行を継

続する応答を待機している場合)、新しいセッションでは、進行中の呼び出しに関する情報がありません。ブロックされた呼び出しは、それ以外では、決して受け取ることのない応答を待機してハングする可能性があります。

これを防ぐため、JBoss EAP メッセージングは、JMS を使用する場合は **javax.jms.JMSException** をスローし、コア API を使用する場合はエラーコード **ActiveMQException.UNBLOCKED** を設定して **ActiveMQException** をスローすることにより、フェイルオーバー時に進行中のブロッキング呼び出しのブロックを解除します。この例外をキャッチし、必要に応じて操作を再試行するかどうかはクライアントコード次第です。

ブロック解除されるメソッドが commit() または prepare() の呼び出しである場合、トランザクションは自動的にロールバックされ、JBoss EAP メッセージングは、JMS を使用する場合は **javax.jms.TransactionRolledBackException** をスローし、コア API を使用する場合はエラーコード **ActiveMQException.TRANSACTION_ROLLED_BACK** を設定して **ActiveMQException** をスローします。

30.7.1.2. トランザクションによるフェイルオーバーの処理

セッションがトランザクションであり、現在のトランザクションでメッセージがすでに送信または確認応答されている場合、サーバーはフェイルオーバー中にメッセージまたは確認応答が失われたかどうかがわかりません。

その結果、トランザクションはロールバック専用としてマークされ、その後コミットしようとすると、JMS を使用する場合は **javax.jms.TransactionRolledBackException** をスローし、コア API を使用する場合はエラーコード **ActiveMQException.TRANSACTION_ROLLED_BACK** を設定して **ActiveMQException** をスローします。

警告

このルールに関する注意点は、XA が JMS またはコア API を介して使用される場合です。2 フェーズコミットが使用され、**prepare()** がすでに呼び出されている場合、ロールバックによって **HeuristicMixedException** が発生する可能性があります。このため、コミットは **XAException.XA_RETRY** 例外をスローします。これは、後のある時点でコミットを再試行する必要があることをトランザクションマネージャーに通知します。これの副作用は、非永続メッセージが失われることです。これを回避するには、XA の使用時に永続メッセージを使用するようにしてください。確認応答では、**prepare()** が呼び出される前にサーバーにフラッシュされるため、これは問題ではありません。

例外をキャッチし、必要に応じてクライアント側のローカルロールバックコードを実行するのはユーザー次第です。すでにロールバックされているため、セッションを手動でロールバックする必要はありません。ユーザーは、同じセッションに対してトランザクション操作を再試行できます。

コミット呼び出しの実行中にフェイルオーバーが発生すると、サーバーは、前述したように応答を返さないため、ハングを防ぐために呼び出しのブロックを解除します。この場合、障害が発生する前にトランザクションのコミットがライブサーバーで実際に処理されたかどうかをクライアントが判断するのは容易ではありません。

注記

XA が JMS またはコア API を介して使用されている場合、**XAException.XA_RETRY** がスローされます。これは、ある時点で再試行が必要であることをトランザクションマネージャーに通知するためです。後のある時点で、トランザクションマネージャーはコミットを再試行します。元のコミットが発生していない場合、それはまだ存在し、コミットされます。存在しない場合、コミットされたと見なされますが、トランザクションマネージャーが警告を記録する可能性があります。

これを修正するには、クライアントはトランザクションで重複検出を有効にし、呼び出しがブロック解除された後にトランザクション操作を再試行します。検出をサーバーに設定する方法は、「重複メッセージ検出」を参照してください。フェイルオーバー前にトランザクションが実際にライブサーバーで正常にコミットされた場合、重複検出により、トランザクションで再送信される永続メッセージがサーバーで無視され、トランザクションの再試行時にメッセージが複数回送信されないようにします。

30.7.1.3. 接続失敗の通知

JMS は、接続障害の非同期通知を送信する標準メカニズムである **java.jms.ExceptionListener** を提供します。このクラスに関する詳細は、JMS javadoc を参照してください。また、コア API は、**org.apache.activemq.artemis.core.client.SessionFailureListener** クラスの形式で同様の機能を提供します。

ExceptionListener または **SessionFailureListener** インスタンスは、接続が正常にフェイルオーバー、再接続、または再割り当てされたかどうかに関係なく、接続障害が発生した場合に JBoss EAP によって常に呼び出されます。ただし、**SessionfailureListener** の **connectionFailed()** に渡される **failedOver** フラグの値、または **javax.jms.JMSException** のエラーコードを調べると、再接続または再割り当てが発生したかどうかを確認できます。エラーコードは次のいずれかになります。

JMSException エラーコード

エラーコード	説明
FAILOVER	フェイルオーバーが発生し、正常に再割り当てまたは再接続しました。
DISCONNECT	フェイルオーバーは発生せず、切断されています。

30.7.2. アプリケーションレベルのフェイルオーバー

場合によっては、自動クライアントフェイルオーバーは行わず、接続障害を自分で処理し、障害ハンドラーに再接続ロジックを手動でコーディングすることもできます。フェイルオーバーはユーザーアプリケーションレベルで処理されるため、これをアプリケーションレベルのフェイルオーバーとして定義します。

JMS を使用する場合にアプリケーションレベルのフェイルオーバーを実装するには、JMS 接続で **ExceptionListener** クラスを設定します。**ExceptionListener** は、接続障害が検出された場合に JBoss EAP メッセージングによって呼び出されます。**ExceptionListener** では、古い JMS 接続を閉じ、場合によっては、JNDI から新しい接続ファクトリーインスタンスを検索し、新しい接続を作成します。

コア API を使用している場合、手順は非常に似ています。コア **ClientSession** インスタンスで **FailureListener** を設定します。

30.8. 使用済みの接続の検出

このセクションでは、接続の Time-to-Live (TTL) について説明します。また、JBoss EAP メッセージングが、クラッシュしたクライアントおよびリソースを正常にクローズせずに終了したクライアントを処理する方法についても説明します。

サーバー上の使用済みの接続リソースのクリーンアップ

JBoss EAP クライアントアプリケーションが終了する前に、**finally** ブロックを使用して制御された方法で、そのリソースを閉じる必要があります。

以下は、コアクライアントが **finally** ブロックでセッションとセッションファクトリーを適切に閉じる例です。

```
ServerLocator locator = null;
ClientSessionFactory sf = null;
ClientSession session = null;

try {
   locator = ActiveMQClient.createServerLocatorWithoutHA(..);

   sf = locator.createClientSessionFactory();;

   session = sf.createSession(...);

   ... do some stuff with the session...
}
finally {
   if (session != null) {
      session.close();
   }

   if (sf != null) {
      sf.close();
   }

   if(locator != null) {
      locator.close();
   }
}
```

以下は、適切に動作する JMS クライアントアプリケーションの例です。

```
Connection jmsConnection = null;

try {
   ConnectionFactory jmsConnectionFactory =
ActiveMQJMSClient.createConnectionFactoryWithoutHA(...);

   jmsConnection = jmsConnectionFactory.createConnection();

   ... do some stuff with the connection...
}
finally {
   if (connection != null) {
      connection.close();
   }
}
```

ただし、クライアントがクラッシュし、リソースを消去する機会がない場合もあります。これが発生すると、サーバー側のリソースがサーバー上でハングしたままになる可能性があります。これらのリソースを削除しないと、サーバーでリソースリークが発生し、時間が経つにつれてサーバーでメモリーまたは他のリソースが不足する可能性があります。

使用済みのクライアントリソースを消去しようとする場合、重要なことは、クライアントとサーバー間のネットワークに障害が発生した後に復帰し、クライアントが再接続できるという事実を認識することです。JBoss EAP はクライアントの再接続に対応しているため、サーバー側の「使用済みの」リソースをすぐに消去しないようにすることが重要です。そうしないと、クライアントがサーバー上の古いセッションに再接続してリソースを取り戻すことができなくなります。

JBoss EAP は、これらすべてを設定可能にします。**ClientSessionFactory** の設定ごとに、Time-To-Live (TTL) プロパティーを使用して、クライアントからのデータがない場合にサーバーが接続を有効に保つ時間 (ミリ秒単位) を設定できます。クライアントは、サーバーが接続を閉じないように、定期的に「ping」パケットを自動的に送信します。サーバーが TTL 時間の間に接続でパケットを受信しない場合、その接続に関連するサーバー上のすべてのセッションを自動的に閉じます。

JMS を使用している場合、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。または、JMS 接続ファクトリーインスタンスをサーバー側の JNDI に直接デプロイする場合は、パラメーター **connectionTtl** を使用して xml 設定に指定できます。

http-connector などのネットワークベース接続の **ConnectionTTL** のデフォルト値は **60000** (1 分) です。**in-vm** など、内部接続の接続 TTL のデフォルト値は **-1** です。**ConnectionTTL** の **-1** の値は、サーバーが、サーバー側で接続をタイムアウトしないことを意味します。

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、サーバー側でグローバル値を設定します。これを行うに、サーバー設定に **connection-ttl-override** 属性を指定します。**connection-ttl-override** のデフォルト値は **-1** です。これは「オーバーライドしない」、つまり、クライアントが独自の値を使用できることを意味します。

コアセッションまたは JMS 接続を閉じる

すべてのコアクライアントセッションと JMS 接続は、使用の終了時に常に **finally** ブロックで明示的に閉じられていることが重要です。

これができない場合、JBoss EAP はガベージコレクション時にこれを検出します。その後、接続を閉じ、以下のような警告をログに記録します。

```
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession]
I'm closing a ClientSession you left open. Please make sure you close all ClientSessions explicitly
before let
ting them go out of scope!
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession]
The session you didn't close was created here:
java.lang.Exception
   at org.apache.activemq.artemis.core.client.impl.DelegatingSession.<init>
(DelegatingSession.java:83)
   at org.acme.yourproject.YourClass (YourClass.java:666)
```

JMS を使用している場合、警告は、クライアントセッションではなく、JMS 接続を伴うことに注意してください。また、ログには、閉じられていない JMS 接続またはコア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が示されます。これにより、コード内のエラーを特定し、適切に修正できます。

クライアント側からの障害の検出

クライアントは、サーバーからデータを受信している限り、接続は有効であると見なします。クライア

ントが **client-failure-check-period** ミリ秒のパケットを受信しない場合は、接続が失敗したと見なし、フェイルオーバーを開始するか、クライアントの設定方法に応じて、**FailureListener** インスタンスまたは **ExceptionListener** インスタンスを呼び出します。

JMS を使用している場合、この動作は、**ActiveMQConnectionFactory** インスタンスの **ClientFailureCheckPeriod** 属性で定義します。

HTTP 接続など、ネットワーク接続におけるクライアント障害チェックの間隔のデフォルト値は **30000** (30 秒) です。in-vm 接続のクライアント障害チェックの間隔のデフォルト値は **-1** です。**-1** の値は、サーバーからデータが受信されない場合、クライアントがクライアント側で接続に失敗しないことを意味します。接続のタイプに関係なく、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、チェックの間隔は通常、サーバー上の接続 TTL の値よりもはるかに短くします。

非同期接続実行の設定
サーバー側で受信されたパケットの多くは、**リモーティング**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、デフォルトでは、一部のパケットはスレッドプールからスレッドを使用して実行され、**リモーティング**スレッドが長時間固定されないようにします。別のスレッドで操作を非同期に処理すると、待ち時間が少し長くなることに注意してください。これらのパケットは、以下のとおりです。

```
org.apache.activemq.artemis.core.protocol.core.impl.wireformat.RollbackMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionCloseMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionCommitMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionXACommitMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionXAPrepareMessage

org.apache.activemq.artemis.core.protocol.core.impl.wireformat.SessionXARollbackMessage
```

非同期接続実行を無効にするには、パラメーター **async-connection-execution-enabled** を **false** に設定します。デフォルト値は **true** です。

30.9. クライアントの再接続とセッションの再割り当て

JBoss EAP メッセージングクライアントは、クライアントとサーバー間の接続で障害が検出された場合に、自動的にサーバーに再接続または再割り当てするように設定できます。

透過的なセッションの再割り当て
ネットワークの一時停止のように一時的な原因で障害が発生し、ターゲットサーバーが再起動されなかった場合、**connection-ttl** の値より長い間、クライアントが切断されることはないと仮定して、セッションはサーバー上に残ります。「使用済みの接続の検出」を参照してください。

このシナリオでは、再接続が確立されたときに、JBoss EAP がクライアントセッションをサーバーセッションに自動的に再割り当てします。これは 100% 透過的に行われ、クライアントは何も起こらなかったようにそのまま続行できます。

JBoss EAP メッセージングクライアントがサーバーにコマンドを送信すると、送信された各コマンドはインメモリーバッファーに保存されます。接続が失敗し、その後クライアントが再割り当てプロトコルの一部として同じサーバーに再割り当てを試みると、サーバーは正常に受信した最後のコマンドの ID をクライアントに提供します。

クライアントがフェイルオーバー前に受信したよりも多くのコマンドを送信した場合、クライアントとサーバーが状態を回復できるように、送信されたコマンドをバッファーから再生できます。

このバッファーのサイズ (バイト単位) は、**confirmationWindowSize** プロパティーで設定します。サーバーが **confirmationWindowSize** バイトのコマンドを受信して処理すると、コマンド確認をクライアントに送り返し、クライアントはバッファー内のスペースを解放できます。

サーバー上で JMS サービスを使用して JMS 接続ファクトリーインスタンスを JNDI にロードする場合、選択した **connection-factory** の **confirmation-window-size** 属性を設定して、サーバー設定にこのプロパティーを設定できます。JMS を使用し、JNDI を使用しない場合、適切なセッターメソッド **setConfirmationWindowSize** を使用して **ActiveMQConnectionFactory** インスタンスにこれらの値を直接設定できます。コア API を使用している場合、**ServerLocator** インスタンスには、公開された **setConfirmationWindowSize** メソッドもあります。

confirmationWindowSize を **-1** (デフォルト) に設定すると、バッファリングが無効になり、再割り当てが発生しないようになり、代わりに再接続が強制されます。

セッション再接続

または、サーバーがクラッシュ後に実際に再起動されたか、停止されている可能性があります。このような場合、セッションはサーバー上に存在しなくなり、100% 透過的にそれらに再割り当てることができなくなります。

この場合、JBoss EAP は自動的に接続を再接続し、クライアント上のセッションおよびコンシューマーに対応するサーバー上でセッションとコンシューマーを再作成します。このプロセスは、バックアップサーバーにフェイルオーバーする際に発生するプロセスとまったく同じです。

また、クライアントの再接続は、コアブリッジなどのコンポーネントがターゲットサーバーに再接続できるように、内部的に使用されます。

再接続中にトランザクションセッションと非トランザクションセッションが再接続される方法と、1 回限りの配信保証を維持するために必要な事項を完全に理解するには、「自動クライアントフェイルオーバー」のセクションを参照してください。

再接続属性の設定

クライアントの再接続を設定するには、以下のプロパティーを設定します。

- retryInterval。このオプションのパラメーターは、ターゲットサーバーへの接続に失敗した場合、後続の再接続試行の間隔 (ミリ秒単位) を設定します。デフォルトの値は **2000** ミリ秒です。

- retryIntervalMultiplier。このオプションのパラメーターは、最後の再試行から次の再試行までの時間を計算するために時間に適用する乗数を設定します。これにより、再試行の間隔に指数バックオフを実装することができます。
  たとえば、**retryInterval** を **1000** ミリ秒に設定し、retryIntervalMultiplier を **2.0** に設定すると、最初の再接続試行が失敗した場合、クライアントは、後続の再接続試行を **1000** ミリ秒、**2000** ミリ秒、**4000** ミリ秒の順に待機します。

  デフォルト値は **1.0** で、各再接続試行が等間隔であけられることを意味します。

- maxRetryInterval。このオプションのパラメーターは、使用される最大再試行間隔を設定します。**retryIntervalMultiplier** を設定しないと、後続の再試行が指数関数的に増加して、途方もなく大きい値になる可能性があります。このパラメーターを設定すると、値の上限を設定できます。デフォルトの値は **2000** ミリ秒です。

- reconnectAttempts。このオプションのパラメーターには、断念してシャットダウンするまでの再接続試行の総回数を設定します。**-1** の値は、無制限の試行回数を示します。デフォルト値は **0** です。

クライアント上で JMS および JNDI を使用して JMS 接続ファクトリーインスタンスを検索する場合は、これらのパラメーターを JNDI コンテキスト環境で指定できます。たとえば、**jndi.properties** ファイルは以下のようになります。

```
java.naming.factory.initial = ActiveMQInitialContextFactory
connection.ConnectionFactory=tcp://localhost:8080?
retryInterval=1000&retryIntervalMultiplier=1.5&maxRetryInterval=60000&reconnectAttempts=1000
```

JMS を使用し、JMS 接続ファクトリーを直接インスタンス化する場合は、作成直後に **ActiveMQConnectionFactory** で適切なセッターメソッドを使用してパラメーターを指定できます。

コア API を使用して、**ServerLocator** インスタンスを直接インスタンス化する場合は、作成直後に ServerLocator で適切なセッターメソッドを使用してパラメーターを指定することもできます。

クライアントは再接続できるが、サーバーが再起動したかタイムアウトした場合など、サーバーでセッションが使用できなくなった場合、クライアントは再割り当てできず、接続またはセッションに登録されている **ExceptionListener** または **FailureListener** インスタンスが呼び出されます。

ExceptionListeners と SessionFailureListeners

クライアントが再接続または再割り当てすると、登録されている JMS **ExceptionListener** またはコア API **SessionFailureListener** が呼び出されます。

第31章 リソースアダプター

Jakarta Connectors Resource Adapter を使用すると、アプリケーションはどのメッセージングプロバイダーとも通信できます。MDB や他の Jakarta Enterprise Beans、さらには Servlet などの Jakarta JEE コンポーネントがメッセージを送受信する方法を設定します。

31.1. 統合された ARTEMIS リソースアダプターについて

JBoss EAP 7 には統合された Artemis リソースアダプターが含まれており、これは **pooled-connection-factory** 要素を使用してリソースアダプターの送受信接続を設定します。

送信接続
送信接続は、**pooled-connection-factory** 要素を使用して定義します。これは、Jakarta Enterprise Beans およびサーブレットによって Jakarta EE デプロイメントで使用され、キューまたはトピックからメッセージを送受信します。接続ファクトリーから作成される接続はアプリケーションサーバーのスコープ内に作成されるため、以下のようにアプリケーションサーバー機能を使用できます。

- 接続プール
- アプリケーションサーバーによって定義されるセキュリティードメインを使用した認証
- トランザクションマネージャーを使用した XA トランザクションへの参加

これらの機能は、**InVmConnectionFactory** や **RemoteConnectionFactory** のような基本の **connection-factory** では使用できないため、これは **pooled-connection-factory** との大きな違いです。また、**pooled-connection-factory** を使用して定義される接続ファクトリーでは、外部スタンドアロン JMS クライアントから JNDI を使用してルックアップを行うことができないことに注意してください。

受信接続
受信接続は、メッセージ駆動 Bean (MDB) がキューまたはトピックからメッセージを受信するためにのみ使用されます。MDB は、キューまたはトピックをリッスンするステートレスセッション Bean です。**onMessage(Message message)** 公開メソッドを実装する必要があります。これは、メッセージがキューまたはトピックに送信されるときに呼び出されます。Artemis リソースアダプターは、キューまたはトピックからメッセージを受信して **onMessage(Message message)** メソッドに渡す役割を担います。この目的のため、受信接続を設定して、統合された Artemis サーバーの場所と追加要素を定義します。

各 MDB セッション Bean は、クライアントスレッドプールのスレッドを使用して、宛先からのメッセージを消費します。最大プールサイズが定義されていない場合は、CPU コアプロセッサー数の 8 倍になるように決定されます。テストスイートなど、多くの MDB セッションを持つシステムの場合、これによりスレッドが枯渇し、MDB がプールからの空きスレッドを待機するように強制する可能性があります。管理 CLI を使用して、クライアントスレッドプールの最大プールサイズを増やすことができます。以下のコマンドは、最大クライアントスレッドプールサイズを **128** に設定します。

```
/subsystem=messaging-activemq:write-attribute(name=global-client-thread-pool-max-size,value=128)
```

クライアントスレッドプールサイズを設定する方法は、「クライアントスレッド管理」を参照してください。MDB に関する詳細は、JBoss EAP 『**Developing EJB Applications**』の「Message Driven Beans」を参照してください。

31.2. リモート接続の統合された ARTEMIS リソースアダプターの使用

JBoss EAP には、統合 ActiveMQ Artemis メッセージングサーバーに接続するためのリソースアダプターが含まれています。デフォルトで、**messaging-activemq** サブシステムに定義された **pooled-**

connection-factory はアダプターを使用して接続を確立します。ただし、同じリソースアダプターを使用して、JBoss EAP のリモートインスタンス内で実行している Artemis サーバーへの接続も確立できます。

重要

messaging-activemq サブシステムにデフォルトで設定されている **activemq-ra** のプールされた接続ファクトリーには、**java:jboss/DefaultJMSConnectionFactory** エントリーが割り当てられます。このエントリーは **messaging-activemq** サブシステムに必要です。**activemq-ra** のプールされた接続ファクトリーを削除する場合は、このエントリーを別の接続ファクトリーに割り当てる必要があります。そうしないと、デプロイメントのサーバーログに以下のエラーが表示されます。

```
WFLYCTL0412: Required services that are not installed:" =>
["jboss.naming.context.java.jboss.DefaultJMSConnectionFactory"]
```

JBoss EAP のリモートインスタンス内で実行している Artemis サーバーに接続するには、以下の手順に従って新しい **pooled-connection-factory** を作成します。

1. リモートメッセージングサーバーを指す outbound-socket-binding を作成します。

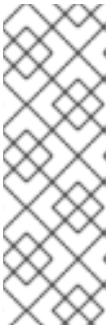
```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-
binding=remote-server:add(host=<server host>, port=8080)
```

2. 手順 1 で作成した outbound-socket-binding を参照する remote-connector を作成します。

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

3. 手順 2 で作成した remote-connector を参照する pooled-connection-factory を作成します。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=remote-
artemis:add(connectors=[remote-http-connector], entries=[java:/jms/remoteCF])
```



注記

Artemis 1.x では、宛先名の接頭辞が必要でした (トピックの場合は jms.topic、キューの場合は jms.queue)。Artemis 2.x では接頭辞は必要ありませんが、Artemis 1.x との互換性のために、EAP は引き続き接頭辞を追加し、互換性モードで実行するように Artemis に指示します。リモート Artemis 2.x サーバーに接続する場合は、互換性モードではなく、接頭辞は不要になる可能性があります。接頭辞なしで宛先を使用する場合は、**enable-amq1-prefix** 属性を **false** に設定して、接頭辞を含まないように接続ファクトリーを設定できます。

MDB を pooled-connection-factory を使用するように設定する

pooled-connection-factory をリモート Artemis サーバーに接続するよう設定した後、リモートサーバーからメッセージを読み取るメッセージ駆動 Bean (MDB) では、**pooled-connection-factory** リソースの名前を使用して **@ResourceAdapter** アノテーションを付加する必要があります。

```
import org.jboss.ejb3.annotation.ResourceAdapter;

@ResourceAdapter("remote-artemis")
```

```
@MessageDriven(name = "MyMDB", activationConfig = { ... })
public class MyMDB implements MessageListener {
    public void onMessage(Message message) {
        ...
    }
}
```

MDB がリモートサーバーにメッセージを送信する必要がある場合は、JNDI **エントリー**の 1 つを使用して検索することにより、**pooled-connection-factory** を挿入する必要があります。

```
@Inject
@JMSConnectionFactory("java:/jms/remoteCF")
private JMSContext context;
```

JMS 宛先の設定

MDB は、メッセージを消費する宛先も指定する必要があります。これを実行するための標準的な方法は、ローカルサーバーの JNDI ルックアップに対応する **destinationLookup** アクティベーション設定プロパティーを定義することです。

```
@ResourceAdapter("remote-artemis")
@MessageDriven(name = "MyMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "myQueue"),
    ...
})
public class MyMDB implements MessageListener {
    ...
}
```

ローカルサーバーにリモート Artemis サーバーの JNDI バインディングが含まれない場合は、**destination** アクティベーション設定プロパティーを使用して、リモート Artemis サーバーに設定されているように宛先の名前を指定し、**useJNDI** アクティベーション設定プロパティーを **false** に設定します。これは、JNDI ルックアップを必要とせずに JMS 宛先を自動的に作成するように Artemis リソースアダプターに指示します。

```
@ResourceAdapter("remote-artemis")
@MessageDriven(name = "MyMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "myQueue"),
    ...
})
public class MyMDB implements MessageListener {
    ...
}
```

上記の例では、アクティベーション設定プロパティーは、リモート Artemis サーバーでホストされている **myQueue** という名前の JMS キューからメッセージを消費するように MDB を設定します。ほとんどの場合、MDB は、消費されたメッセージを処理するために他の宛先をルックアップする必要がなく、メッセージに定義されている場合は、**JMSReplyTo** 宛先を使用できます。

MDB でリモートサーバーに定義された他の JMS 宛先が必要な場合は、クライアント側 JNDI を使用する必要があります。詳細は、「サーバーへの接続」を参照してください。

31.3. RED HAT AMQ に接続するように ARTEMIS リソースアダプターを設定する

統合された Artemis リソースアダプターを設定し、JBoss EAP 7.4 アプリケーションの JMS プロバイダーとなる Red Hat AMQ 7 のリモートインストールに接続するように設定できます。これにより、JBoss EAP がリモート Red Hat AMQ 7 サーバーのクライアントになることができます。

AMQP や STOMP などの他のメッセージングプロトコルのサポートが必要な場合は、Red Hat AMQ 7 をメッセージングブローカーとして設定する必要があります。JBoss EAP サーバーに統合された Artemis リソースアダプターは、デプロイされたアプリケーションのメッセージを処理するために使用できます。

統合されたリソースアダプターの制限

キューとトピックの動的作成

JBoss EAP 7.4 に統合されている Artemis リソースアダプターは、Red Hat AMQ 7 ブローカーのキューとトピックの動的な作成をサポートしていないことに注意してください。すべてのキューとトピック宛先を、リモート Red Hat AMQ 7 サーバーに直接設定する必要があります。

接続ファクトリーの作成

Red Hat AMQ では、接続ファクトリーは **pooled-connection-factory** と **external-context** の両方を使用して設定できますが、各接続ファクトリーの作成方法には違いがあります。**external-context** を使用して接続ファクトリーを作成すると、JMS 仕様に定義されている単純な JMS 接続ファクトリーが作成されます。新規作成された接続ファクトリーは **RemoteConnectionFactory** と同じで、これは **messaging-activemq** サブシステムにデフォルトで定義されています。この接続ファクトリーはアプリケーションサーバー内の他のコンポーネントから独立しているため、トランザクションマネージャーやセキュリティーマネージャーなどの他のコンポーネントを認識せず、使用することができません。このため、JBoss EAP 7 で接続ファクトリーを作成するには、**pooled-connection-factory** のみを使用できます。**external-context** は、ローカルデプロイメントが検索または挿入できるように、リモート AMQ 7 ブローカーですでに設定されている JMS 宛先を JBoss EAP 7 サーバーの JNDI ツリーに登録するためにのみ使用できます。

Artemis リソースアダプターを使用しないため、**external-context** または **connection-factory** 要素を設定して作成された接続ファクトリーを使用してリモート AMQ 7 ブローカーに接続することはできません。リモート AMQ 7 ブローカーに接続する場合は、**pooled-connection-factory** 要素を設定して作成される接続ファクトリーのみが、使用がサポートされます。

リモート Red Hat AMQ Server を使用するように JBoss EAP を設定する

管理 CLI を使用して、以下の手順に従って、メッセージングプロバイダーとして Red Hat AMQ 7 のリモートインストールを使用するように JBoss EAP を設定できます。

1. Red Hat AMQ 7 の **broker.xml** デプロイメント記述子ファイルにキューを設定します。

```
<configuration xmlns="urn:activemq"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="urn:activemq /schema/artemis-configuration.xsd">

   <core xmlns="urn:activemq:core" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        xsi:schemaLocation="urn:activemq:core ">
       ...
       <acceptors>
          <acceptor name="netty-acceptor">tcp://localhost:61616?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.
          </acceptor>
       </acceptors>
       <addresses>
          <address name="MyQueue">
             <anycast>
                <queue name="MyQueue" />
```

```
            </anycast>
        </address>
        <address name="MyOtherQueue">
            <anycast>
                <queue name="MyOtherQueue" />
            </anycast>
        </address>
        <address name="MyTopic">
            <multicast/>
        </address>
    <addresses>
    ...
  </core>
</configuration>
```

注記

JBoss EAP に含まれる Artemis リソースアダプターは、ActiveMQ Artemis JMS クライアント 2.x を使用します。このクライアントは、アドレスに **anycastPrefix** と **multicastPrefix** の接頭辞が必要です。また、キュー名がアドレス名と同じであることも必要です。

2. リモートコネクターを作成します。

```
/subsystem=messaging-activemq/remote-connector=netty-remote-throughput:add(socket-binding=messaging-remote-throughput)
```

これにより、**messaging-activemq** サブシステムに以下の **remote-connector** が作成されます。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    ...
    <remote-connector name="netty-remote-throughput" socket-binding="messaging-remote-throughput"/>
    ...
</subsystem>
```

3. リモート宛先アウトバウンドソケットバインディングを追加します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=messaging-remote-throughput:add(host=localhost, port=61616)
```

これにより、**outbound-socket-binding** 要素設定に以下の **remote-destination** が作成されます。

```
<outbound-socket-binding name="messaging-remote-throughput">
    <remote-destination host="localhost" port="61616"/>
</outbound-socket-binding>
```

4. プールされた接続ファクトリーをリモートコネクターに追加します。

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-
remote:add(transaction=xa,entries=[java:/RemoteJmsXA,
java:jboss/RemoteJmsXA],connectors=[netty-remote-throughput])
```

これにより、**messaging-activemq** サブシステムに以下の **pooled-connection-factory** が作成されます。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    ...
    <pooled-connection-factory name="activemq-ra-remote" entries="java:/RemoteJmsXA
java:jboss/RemoteJmsXA" connectors="netty-remote-throughput"/>
    ...
</subsystem>
```

注記

Artemis 1.x では、宛先名の接頭辞が必要でした (トピックの場合は jms.topic、キューの場合は jms.queue)。Artemis 2.x では接頭辞は必要ありませんが、Artemis 1.x との互換性のために、EAP は引き続き接頭辞を追加し、互換性モードで実行するように Artemis に指示します。リモート Artemis 2.x サーバーに接続する場合は、互換性モードではなく、接頭辞は不要になる可能性があります。接頭辞なしで宛先を使用する場合は、**enable-amq1-prefix** 属性を **false** に設定して、接頭辞を含まないように接続ファクトリーを設定できます。

5. キューとトピックの **external-context** バインディングを作成します。

```
/subsystem=naming/binding=java\:global\/remoteContext:add(binding-type=external-context,
class=javax.naming.InitialContext, module=org.apache.activemq.artemis, environment=
[java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory,
java.naming.provider.url=tcp://127.0.0.1:61616, queue.MyQueue=MyQueue,
queue.MyOtherQueue=MyOtherQueue, topic.MyTopic=MyTopic])
```

これにより、**naming** サブシステムに以下の **external-context** バインディングが作成されます。

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    ...
    <bindings>
        <external-context name="java:global/remoteContext"
module="org.apache.activemq.artemis" class="javax.naming.InitialContext">
            <environment>
                <property name="java.naming.factory.initial"
value="org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory"/>
                <property name="java.naming.provider.url" value="tcp://127.0.0.1:61616"/>
                <property name="queue.MyQueue" value="MyQueue"/>
                <property name="queue.MyOtherQueue" value="MyOtherQueue"/>
                <property name="topic.MyTopic" value="MyTopic"/>
            </environment>
        </external-context>
    </bindings>
    ...
</subsystem>
```

6. JNDI 名を Red Hat AMQ 7 アドレス名の値に設定して、JMS キューおよびトピックのルックアップエントリーを作成します。これにより、JNDI 名と Red Hat AMQ 7 アドレス名の間のマッピングが作成されます。

```
/subsystem=naming/binding=java\:\/MyQueue:add(lookup=java:global/remoteContext/MyQueue
,binding-type=lookup)
/subsystem=naming/binding=java\:\/MyOtherQueue:add(lookup=java:global/remoteContext/My
OtherQueue,binding-type=lookup)
/subsystem=naming/binding=java\:\/MyTopic:add(lookup=java:global/remoteContext/MyTopic,bi
nding-type=lookup)
```

これにより、**naming** サブシステムに以下の **lookup** 設定が作成されます。

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    ...
    <lookup name="java:/MyQueue" lookup="java:global/remoteContext/MyQueue"/>
    <lookup name="java:/MyOtherQueue"
lookup="java:global/remoteContext/MyOtherQueue"/>
    <lookup name="java:/MyTopic" lookup="java:global/remoteContext/MyTopic"/>
    ...
</subsystem>
```

または、naming サブシステムを設定する代わりに、**/subsystem=messaging-activemq/external-jms-queue** または **/subsystem=messaging-activemq/external-jms-topic** リソースを定義します。例を以下に示します。

```
/subsystem=messaging-activemq/external-jms-queue=MyQueue:add(entries=
[java:/MyQueue])
```

これにより、以下のリソースが作成されます。

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    ...
    <external-jms-queue name="MyQueue" entries="java:/MyQueue"/>
    ...
</subsystem>
```

注記

external-jms-queue リソースは、キュー管理および統計の操作を提供しません。

これで、JBoss EAP が Red Hat AMQ 7 のリモートインストールをメッセージングプロバイダーとして使用するよう設定されました。

31.4. リモート ARTEMIS ベースのブローカーの JMS リソース設定

管理 CLI から、**@JMSConnectionFactoryDefinition** アノテーションまたは **@JMSDestinationDefinition** アノテーションを使用して、Red Hat AMQ 7 などのリモート Artemis ベースのブローカーの JMS リソースを設定できます。管理コンソールからリソースを設定することもできます。

リモート ActiveMQ サーバーリソースは、Artemis のローカルインスタンスを必要としません。これにより、JBoss EAP イメージのメモリーと CPU フットプリントが削減されます。

31.4.1. JMSConnectionFactoryDefinition アノテーションを使用した JMS リソース設定

JBoss EAP は **@JMSConnectionFactoryDefinition** アノテーションを使用して接続ファクトリーを定義します。この接続ファクトリーは、ローカルまたはリモートの Artemis ブローカーに接続できます。**@JMSConnectionFactoryDefinition** アノテーションの **resourceAdapter** 要素は、リモート Artemis ブローカーに接続できる **messaging-subsystem** に定義される **pooled-connection-factory** の名前を参照します。**resourceAdapter** 要素は、接続ファクトリーの作成に使用されるリソースアダプター、または接続ファクトリーが定義されるリソースアダプターを定義します。

resourceAdapter 要素が **@JMSConnectionFactoryDefinition** アノテーションに定義されていない場合、**messaging-activemq** サブシステムはデフォルトで接続ファクトリーの JNDI 名を使用します。これは、デフォルトのバインディングとして知られています。デフォルトのバインディングは、**/subsystem=ee/service=default-bindings** の **jms-connection-factory** 属性を使用して定義されます。**resourceAdapter** 要素が指定されているか、または **jms-connection-factory** のデフォルトのバインディングから定義できる場合、およびリモートブローカーへの **pooled-connection-factory** である場合、リモートブローカーへの接続に使用できます。

resourceAdapter が **messaging-activemq** サブシステムに定義されていない場合、または **jms-connection-factory** のデフォルトのバインディングから取得できない場合、JMS リソースの作成タスクは、リソースアダプターの **admin-objects** および **connection-definitions** リソースに基づいて **resource-adapters** サブシステムに委任されます。

以下のセクションでは、**@JMSConnectionFactoryDefinition** アノテーションを設定して使用する方法の例を紹介します。

デフォルトのリソースアダプターを使用した @JMSConnectionFactoryDefinition の設定

1. コネクターを作成します。

```
/subsystem=messaging/remote-connector=remote-amq:add(socket-binding="messaging-remote-throughput")
```

2. プールされた接続ファクトリーを作成します。

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-amq"]
```

3. **ee** サブシステムのデフォルトの JMS 接続ファクトリーを定義します。

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory, value="java:/jms/remote-amq/JmsConnectionFactory")
```

4. アプリケーションコードで **@JMSConnectionFactoryDefinition** アノテーションを使用します。

```
@JMSConnectionFactoryDefinition(
        name="java:app/myCF"
```

リモート Artemis ブローカーを使用した @JMSConnectionFactoryDefinition の設定

1. コネクターを作成します。

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-
binding="messaging-remote-throughput")
```

2. プールされた接続ファクトリーを作成します。

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-
remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-
amq"])
```

3. **ee** サブシステムのデフォルトの JMS 接続ファクトリーを定義します。

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory,
value="java:/jms/remote-amq/JmsConnectionFactory")
```

4. アプリケーションコードで **@JMSConnectionFactoryDefinition** アノテーションを使用します。

```
@JMSConnectionFactoryDefinition(
        name="java:app/myCF"
        resourceAdapter="myPCF"
)
```

サードパーティー JMS リソースアダプターを使用した @JMSConnectionFactoryDefinition の設定

1. コネクターを作成します。

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-
binding="messaging-remote-throughput")
```

2. プールされた接続ファクトリーを作成します。

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-
remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-
amq"])
```

3. **ee** サブシステムのデフォルトの JMS 接続ファクトリーを定義します。

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory,
value="java:/jms/remote-amq/JmsConnectionFactory")
```

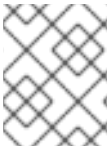
4. アプリケーションコードで **@JMSConnectionFactoryDefinition** アノテーションを使用します。

```
@JMSConnectionFactoryDefinition(
        name="java:app/myCF"
        resourceAdapter="wsmq"
)
```

31.4.2. JMSDestinationDefinition アノテーションを使用した JMS リソース設定

サーバーリソースを使用して、ローカルブローカーへの **pooled-connection-factory** に必要な宛先を作成できます。

resourceAdapter 要素が **pooled-connection-factory** 名を指し、ローカルブローカー (例: **/subsystem/messaging-activemq/server=default**) に定義されている場合、ローカル Artemis ブローカーに宛先を作成します。



注記

リモート Artemis ベースのブローカーに宛先を作成する必要がある場合は、**pooled-connection-factory** を **messaging-activemq** サブシステムに定義する必要があります。

@JMSDestinationDefinition アノテーションに設定された **resourceAdapter** が、**messaging-activemq** サブシステムの **server** に定義された **resourceAdapter** 要素と一致する場合、**pooled-connection-factory** のコネクターがローカル Artemis ブローカーを指すか、リモート Artemis ブローカーを指すかに関係なく、このブローカーに宛先が作成されます。

JMSDestinationDefinition アノテーションを使用した JMS リソースの設定

1. コネクターを作成します。

```
/subsystem=messaging-activemq/remote-connector=remote-amq:add(socket-binding="messaging-remote-throughput")
```

2. プールされた接続ファクトリーを作成します。

```
/subsystem=messaging-activemq/pooled-connection-factory=activemq-ra-remote:add(entries=["java:/jms/remote-amq/JmsConnectionFactory"],connectors=["remote-amq"])
```

3. **ee** サブシステムのデフォルトの JMS 接続ファクトリーを定義します。

```
/subsystem=ee/service=default-bindings:write-attribute(name=jms-connection-factory, value="java:/jms/remote-amq/JmsConnectionFactory")
```

4. アプリケーションコードに **@JMSDestinationDefinition** アノテーションを使用します。

```
@JMSDestinationDefinition(
    name = "java:/jms/queue/MessageBeanQueue",
    interfaceName = "javax.jms.Queue",
    destinationName = "MessageBeanQueue"
    properties= {
        "management-address=remote-activemq.management"
    }
)
```

31.4.3. 管理コンソールを使用したリモート ActiveMQ サーバーリソースの設定

管理コンソールから以下のリモート ActiveMQ サーバーリソースを設定できます。

- 汎用コネクター
- VM コネクターで

- HTTP コネクター
- リモートコネクター
- 検出グループ
- 接続ファクトリー
- プールされた接続ファクトリー
- 外部 JMS キュー
- 外部 JMS トピック

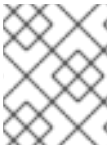
管理コンソールからのリモート ActiveMQ サーバーリソースを設定するには、以下を行います。

1. 管理コンソールにアクセスし、**設定 → サブシステム → Messaging (ActiveMQ) → リモート ActiveMQ Server** を選択し、**表示**をクリックします。
2. ナビゲーションペインで、設定するリソースをクリックします。

31.5. RED HAT JBOSS A-MQ リソースアダプターのデプロイ

Red Hat JBoss A-MQ 製品が提供するリソースアダプターをデプロイし、たとえば Red Hat JBoss A-MQ 6.3.0 を JBoss EAP の外部 JMS プロバイダーにすることができます。

Red Hat JBoss A-MQ リソースアダプターをデプロイおよび設定する方法の詳細は、Red Hat JBoss A-MQ ドキュメントスイート内にある『**Integrating with JBoss Enterprise Application Platform**』の「Install the ActiveMQ Resource Adapter」を参照してください。



注記

製品名が、Red Hat JBoss A-MQ 6.x リリースから Red Hat AMQ 7.x リリースに変更されたことに注意してください。

31.5.1. Red Hat JBoss A-MQ 6 リソースアダプターの問題

- JBoss EAP はアプリケーションを追跡、監視して、閉じられていないリソースを探します。このような監視は多くの場合、有用ですが、アプリケーションが単一のメソッド内の **UserTransaction** の閉じたインスタンスを再利用しようとすると、予期しない動作を引き起こす可能性があります。アプリケーションがこの方法で接続を再利用する場合、Red Hat JBoss A-MQ リソースアダプターを設定するときに **<connection-definition/>** 要素に **tracking="false"** 属性を追加します。

```
<connection-definition class-name="..." tracking="false" ... />
```

- Red Hat JBoss A-MQ 6 リソースアダプターは、JBoss EAP で使用される、Narayana API の **XAResourceWrapper** を実装しません。そのため、トランザクションマネージャーがすべての XA トランザクションの参加者にコミットを送信し、応答の待機中にクラッシュすると、コミットされたトランザクションのレコードがオブジェクトストアから削除されるまで、警告が無期限に記録されます。
- コミットメソッドプロトコルの呼び出し中にネットワークの切断などのエラーが発生すると、Red Hat JBoss A-MQ 6 リソースアダプターはコード **XAER_RMERR** を返します。正しい戻りコードは **XAER_RMFAIL** または **XAER_RETRY** であるため、この動作は XA 仕様に違反しま

す。そのため、トランザクションはメッセージブローカー側で不明な状態のままになり、場合によってはデータの不整合が生じることがあります。予期しないエラーコードが返されると、以下のようなメッセージが記録されます。

```
WARN [com.arjuna.ats.jtax] ...: XAResourceRecord.rollback caused an XA error:
ARJUNA016099: Unknown error code:0 from resource ... in transaction ...:
javax.transaction.xa.XAException: Transaction ... has not been started.
```

- Red Hat JBoss A-MQ 6.x は、Java EE 6 に含まれる JMS 1.1 仕様をサポートします。Java EE 7 で導入され、JBoss EAP 7 でサポートされている JMS 2.0 仕様はサポートしていません。メッセージをリモート Red Hat JBoss A-MQ ブローカーに送信する必要がある場合は、アプリケーションコード内で JMS 1.1 API を使用する必要があります。Red Hat JBoss A-MQ 6.x でサポートされる標準仕様の詳細は、「Red Hat JBoss A-MQ Supported Standards and Protocols」を参照してください。

31.6. IBM MQ リソースアダプターのデプロイ

IBM MQ について

IBM MQ は、IBM のメッセージ指向ミドルウェア (MOM) 製品で、分散システム上のアプリケーションが互いに通信できるようにします。これは、メッセージとメッセージキューを使用して実行できます。IBM MQ は、メッセージキューにメッセージを配信し、メッセージチャネルを使用してデータを他のキューマネージャーへ転送します。IBM MQ の詳細は、IBM の製品 Web サイトの「IBM MQ」を参照してください。

概要

IBM MQ は、JBoss EAP 7.4 の外部 JMS プロバイダーとして設定できます。このセクションでは、JBoss EAP で IBM MQ リソースアダプターをデプロイして設定する手順を説明します。このデプロイメントと設定は、管理 CLI ツールまたは Web ベースの管理コンソールを使用して実行できます。IBM MQ のサポートされる設定に関する最新情報は、「JBoss EAP supported configurations」を参照してください。

注記

設定の変更を有効にするには、IBM MQ リソースアダプターの設定後にシステムを再起動する必要があります。

前提条件

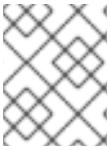
作業を開始する前に、IBM MQ リソースアダプターのバージョンを検証し、その設定プロパティーを理解する必要があります。

- IBM MQ リソースアダプターは、**wmq.jmsra.rar** というリソースアーカイブ (RAR) ファイルとして提供されます。**wmq.jmsra.rar** ファイルは、**/opt/mqm/java/lib/jca/wmq.jmsra.rar** から取得できます。JBoss EAP の各リリースでサポートされる特定のバージョンに関する詳細は、「JBoss EAP supported configurations」を参照してください。
- 以下の IBM MQ 設定値を知っている必要があります。これらの値については、IBM MQ 製品のドキュメントを参照してください。
  - **MQ_QUEUE_MANAGER**: IBM MQ キューマネージャーの名前
  - **MQ_HOST_NAME**: IBM MQ キューマネージャーへの接続に使用されるホスト名
  - **MQ_CHANNEL_NAME**: IBM MQ キューマネージャーへの接続に使用されるサーバーチャネル

- **MQ_QUEUE_NAME**: 宛先キューの名前
- **MQ_TOPIC_NAME**: 宛先トピックの名前
- **MQ_PORT**: IBM MQ キューマネージャーへの接続に使用されるポート
- **MQ_CLIENT**: トランスポートタイプ

- 送信接続の場合は、以下の設定値も理解している必要があります。
  - **MQ_CONNECTIONFACTORY_NAME**: リモートシステムへの接続を提供する接続ファクトリーインスタンスの名前

手順



注記

以下は、IBM のデフォルト設定であり、変更される可能性があります。詳細は IBM MQ のドキュメントを参照してください。

1. まず、**wmq.jmsra.rar** ファイルを **EAP_HOME/standalone/deployments/** ディレクトリーにコピーして、リソースアダプターを手動でデプロイします。
2. 次に、管理 CLI を使用してリソースアダプターを追加し、設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar:add(archive=wmq.jmsra.rar,
transaction-support=XATransaction)
```

transaction-support 要素が **XATransaction** に設定されたことに注意してください。トランザクションを使用する場合は、以下の例にあるように、XA リカバリーデータソースのセキュリティードメインを提供してください。

```
/subsystem=resource-adapters/resource-adapter=test/connection-definitions=test:write-
attribute(name=recovery-security-domain,value=myDomain)
```

XA リカバリーに関する詳細は、JBoss EAP **『設定ガイド』** の「**XA リカバリーの設定**」を参照してください。

トランザクション以外のデプロイメントの場合は、**transaction-support** の値を **NoTransaction** に変更します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar:add(archive=wmq.jmsra.rar,
transaction-support=NoTransaction)
```

3. リソースアダプターが作成されたので、必要な設定要素を追加できます。
   a. キューの **admin-object** を追加し、そのプロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=queue-
ao:add(class-name=com.ibm.mq.connector.outbound.MQQueueProxy, jndi-
name=java:jboss/MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=queue-
ao/config-properties=baseQueueName:add(value=MQ_QUEUE_NAME)
```

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=queue-ao/config-properties=baseQueueManagerName:add(value=MQ_QUEUE_MANAGER)
```

b. トピックの **admin-object** を追加し、そのプロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=topic-ao:add(class-name=com.ibm.mq.connector.outbound.MQTopicProxy, jndi-name=java:jboss/MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=topic-ao/config-properties=baseTopicName:add(value=MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/admin-objects=topic-ao/config-properties=brokerPubQueueManager:add(value=MQ_QUEUE_MANAGER)
```

c. 管理接続ファクトリーの接続定義を追加し、そのプロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd:add(class-name=com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl, jndi-name=java:jboss/MQ_CONNECTIONFACTORY_NAME, tracking=false)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=hostName:add(value=MQ_HOST_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=port:add(value=MQ_PORT)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=channel:add(value=MQ_CHANNEL_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=transportType:add(value=MQ_CLIENT)

/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar/connection-definitions=mq-cd/config-properties=queueManager:add(value=MQ_QUEUE_MANAGER)
```

4. JBoss EAP の EJB3 メッセージングシステムでデフォルトのプロバイダーを JBoss EAP 7 メッセージングから IBM MQ に変更する場合は、以下のように管理 CLI を使用して **ejb3** サブシステムを変更します。

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-name,value=wmq.jmsra.rar)
```

5. 以下のように、MDB コードに **@ActivationConfigProperty** アノテーションと **@ResourceAdapter** アノテーションを設定します。

```
@MessageDriven(name="IbmMqMdb", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType",propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
    @ActivationConfigProperty(propertyName = "hostName", propertyValue = "MQ_HOST_NAME"),
```

```
    @ActivationConfigProperty(propertyName = "port", propertyValue = "MQ_PORT"),
    @ActivationConfigProperty(propertyName = "channel", propertyValue =
"MQ_CHANNEL_NAME"),
    @ActivationConfigProperty(propertyName = "queueManager", propertyValue =
"MQ_QUEUE_MANAGER"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue =
"MQ_QUEUE_NAME"),
    @ActivationConfigProperty(propertyName = "transportType", propertyValue =
"MQ_CLIENT")
})

@ResourceAdapter(value = "wmq.jmsra-VERSION.rar")
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class IbmMqMdb implements MessageListener {
}
```

@ResourceAdapter 値の **VERSION** は、RAR の名前に含まれる実際のバージョンに置き換えてください。

6. リソースアダプターをアクティベートします。

```
/subsystem=resource-adapters/resource-adapter=wmq.jmsra.rar:activate()
```

31.6.1. IBM MQ リソースアダプターの制限と既知の問題

以下の表に、IBM MQ リソースアダプターの既知の問題をまとめています。バージョン列のチェックマーク (✔) は、そのバージョンのリソースアダプターに問題があることを示しています。

表31.1 IBM MQ リソースアダプターの既知の問題

JIRA	問題の説明	IBM MQ 8	IBM MQ 9
JBEAP-503	IBM MQ リソースアダプターは、**Queue.toString()** メソッドと **QueueBrowser.getQueue().toString()** メソッドに異なる文字列の値を返します。**Queue** は **com.ibm.mq.connector.outbound.MQQueueProxy** クラスのインスタンスで、**QueueBrowser.htmlQueueBrowser.getQueue()** メソッドから返される **com.ibm.mq.jms.MQQueue** クラスとは異なります。これらのクラスには、**toString()** メソッドの異なる実装が含まれます。これらの **toString()** メソッドを使用して同じ値を返すことができないことに注意してください。	✔	✔

JIRA	問題の説明	IBM MQ 8	IBM MQ 9
JBEAP-511、JBEAP-550、JBEAP-3686	以下の制限は、IBM MQ のメッセージプロパティー名に適用されます。 • デプロイメント記述子の **activation-config** セクションでは、_、**&**、| などの特殊文字を使用して **destinationName** プロパティーを設定しないでください。これらの文字を使用すると、MDB デプロイメントは **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 例外で失敗します。 • デプロイメント記述子の **activation-config** セクションで、**java:/** 接頭辞を使用して **destinationName** プロパティーを設定しないでください。この接頭辞を使用すると、MDB デプロイメントは **com.ibm.msg.client.jms.DetailedInvalidDestinationException** 例外で失敗します。 • IBM MQ JMS クラスで使用するために予約されているため、プロパティーを「JMS」または「usr.JMS」で開始してはいけません。例外は、IBM Knowledge Center の Web サイトに記載されています。 リソースアダプターの各バージョンに対するメッセージプロパティー名の制限の完全なリストは、IBM Knowledge Center の Web サイトの「IBM MQバージョン 8.0 のプロパティー名の制限」と「IBM MQバージョン 9.0 のプロパティー名の制限」を参照してください。	✔	✔
JBEAP-549	**@ActivationConfigProperty** アノテーションを使用して MDB の **destination** プロパティー名の値を指定するときは、すべて大文字を使用する必要があります。例を以下に示します。 `@ActivationConfigProperty(propertyName = "destination", propertyValue = "QUEUE")`	✔	✔
JBEAP-624	IBM MQ リソースアダプターを使用して **@JMSConnectionFactoryDefinition** アノテーションを使用して Jakarta EE デプロイメントに接続ファクトリーを作成する場合は、**resourceAdapter** プロパティーを指定する必要があります。指定しないと、デプロイメントは失敗します。 `@JMSConnectionFactoryDefinition( name = "java:/jms/WMQConnectionFactory", interfaceName = "javax.jms.ConnectionFactory", resourceAdapter = "wmq.jmsra", properties = { "channel=<channel>", "hostName=<hostname_wmq_broker>", "transportType=<transport_type>", "queueManager=<queue_manager>" } )`	✔	✔

JIRA	問題の説明	IBM MQ 8	IBM MQ 9
JBEAP-2339	IBM MQ リソースアダプターは、接続開始前からキューとトピックからメッセージを読み取ることができます。これは、接続開始前にコンシューマーがメッセージを消費できることを意味します。この問題を回避するには、**external-context** を使用してリモート IBM MQ ブローカーが作成した接続ファクトリーを使用し、IBM MQ リソースアダプターが作成した接続ファクトリーは使用しないでください。	✔	✔

JBEAP-3685	**<transaction-support>XATransaction</transaction-support>** が設定されると、インジェクションを使用して作成されたか、手動で作成されたかに関係なく、**JMSContext** は常に **JMSContext.SESSION_TRANSACTED** です。
	
	以下のコード例では、**@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)** は無視され、**JMSContext** は **JMSContext.SESSION_TRANSACTED** のままになります。
	
	`@Inject`
	`@JMSConnectionFactory("jms/CF")`
	`@JMSPasswordCredential(userName="myusername", password="mypassword")`
	`@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)`
	`transient JMSContext context3;`
		✔	✔

JBEAP-14633	JMS 仕様によると、**QueueSession** インターフェースを使用してパブリッシュ/サブスクライブドメインに固有のオブジェクトを作成することできず、**Session** から継承する特定のメソッドは、**javax.jms.IllegalStateException** をスローする必要があります。このようなメソッドの 1 つが、**QueueSession.createTemporaryTopic()** です。**javax.jms.IllegalStateException** をスローする代わりに、IBM MQ リソースアダプターは **java.lang.NullPointerException** をスローします。	✔	✔
JBEAP-14634	**MQTopicProxy.getTopicName()** は、IBM MQ ブローカーによって設定されていたものとは異なるトピック名を返します。たとえば、トピック名が **topic://MYTOPIC?XMSC_WMQ_BROKER_PUBQ_QMGR=QM** に設定された場合、**MQTopicProxy** は **topic://MYTOPIC** を返します。	✔	✔
JBEAP-14636	JMSContext のデフォルトの **autoStart** 設定は **false** です。これは、**JMSContext** が使用するベースとなる接続が、コンシューマーの作成時に自動的に開始さ れないことを意味します。この設定はデフォルトの **true** にする必要があります。	✔	✔

JIRA	問題の説明	IBM MQ 8	IBM MQ 9
JBEAP-14640	無効な認証情報が使用されると、IBM MQ リソースアダプターは、**JMSSecurityException** ではなく **DetailedJMSException** をスローし、以下のエラーをサーバーコンソールに記録します。 > WARN [org.jboss.jca.core.connectionmanager.pool.strategy.PoolByCri] (EJB default - 7) IJ000604: Throwable while attempting to get a new connection: null: com.ibm.mq.connector.DetailedResourceException: MQJCA1011: Failed to allocate a JMS connection., error code: MQJCA1011 An internal error caused an attempt to allocate a connection to fail. See the linked exception for details of the failure. 以下は、この問題を引き起こす可能性があるコードの例です。 ``` QueueConnection qc = queueConnectionFactory.createQueueConnection("invalidUserName", "invalidPassword"); ```	✔	✔
JBEAP-14642	**MQMessageProducer.send(Destination destination, Message message)** メソッドと **MQMessageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive, CompletionListener completionListener)** メソッドのリソースアダプターによる無効なクラスキャスト変換により、IBM MQ リソースアダプターは **JMSException** をスローし、サーバーコンソールに以下のエラーメッセージをログに記録します。 > SVR-ERROR: Expected JMSException, received com.ibm.mq.connector.outbound.MQQueueProxy cannot be cast to com.ibm.mq.jms.MQDestination これは、キューまたはトピックルックアップで使用される JNDI 名が **com.ibm.mq.connector.outbound.MQQueueProxy/MQTopicProxy** であるためです。	✔	✔
JBEAP-14643	**JMSProducer** インターフェースの **setDeliveryDelay(expDeliveryDelay)** メソッドは設定を変更しません。このメソッドを呼び出しても、デフォルト設定の **0** のままです。	✔	✔
JBEAP-14670	**UserTransaction.begin()** の前に作成される **QueueSession** で作業が行われる場合、その作業はトランザクションの一部とは見なされません。これは、このセッションを使用してキューに送信されたメッセージは **UserTransaction.commit()** によってコミットされず、**UserTransaction.rollback()** の後、メッセージがキューに残ることを意味します。	✔	✔

JIRA	問題の説明	IBM MQ 8	IBM MQ 9
JBEAP-14675	接続を閉じた直後に、同じ **clientID** で **JMSContext** を作成すると、IBM MQ リソースアダプターはサーバーコンソールに以下のエラーを断続的に記録します。 > ERROR [io.undertow.request] (default task-1) UT005023: Exception handling request to /jmsServlet-1.0-SNAPSHOT/: com.ibm.msg.client.jms.DetailedJMSRuntimeException: MQJCA0002: An exception occurred in the IBM MQ layer. See the linked exception for details. A call to IBM MQ classes for Java(tm) caused an exception to be thrown. この問題は、同じ **clientID** の接続が閉じられた後に新しい **JMSContext** の作成で遅延がある場合は発生しません。	✔	✔
JBEAP-15535	コンテナー管理トランザクション (CMT) において、ステートフルセッション Bean がトピックにメッセージを送信しようとすると、メッセージ送信が失敗し、以下のメッセージが示されます。 > SVR-ERROR: com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a message to destination '**MDB_NAME TOPIC_NAME**' スタックトレースは、以下の例外によって発生したことを示しています。 > com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED') reason '2072' ('MQRC_SYNCPOINT_NOT_AVAILABLE')	✔	✔

JIRA	問題の説明	IBM MQ 8	IBM MQ 9
JBEAP-20758	IBM MQ 8 または IBM MQ 9 リソースアダプターの **wmq.jmsra.rar** を JBoss EAP にデプロイすると、以下のエラーメッセージがサーバーコンソールに表示されます。 WARN [org.jboss.as.connector.deployers.RADeployer] (MSC service thread 1-8) IJ020017: Invalid archive: file:/<path-to-jboss>/jboss-eap-7.4.0.Beta/standalone/tmp/vfs/temp/tempa02bdd5ee254e590/content-135e13d4f38704fc/contents/ IBM MQ v9.0.0.4 リソースアダプターは、JBoss EAP 7.4 の Jakarta Messaging プロバイダーテストの一部としてテストされました。この警告メッセージを無視するか、**enabled** 属性を **false** に設定してアーカイブ検証を無効にすることができます。例を以下に示します。 /subsystem=jca/archive-validation=archive-validation:write-attribute(name=enabled, value=false)	✔	✔

31.7. 汎用 JAKARTA メッセージングリソースアダプターのデプロイ

JBoss EAP は、サードパーティーの Jakarta Messaging プロバイダーと連携するように設定できます。ただし、Jakarta アプリケーションプラットフォームとの統合のために、すべての Jakarta Messaging プロバイダーが Jakarta Messaging Jakarta Connectors リソースアダプターを生成するわけではありません。この手順では、JBoss EAP に含まれる汎用 Jakarta Messaging リソースアダプターを設定して Jakarta Messaging プロバイダーに接続する手順を説明します。この手順では、Tibco EMS 8 を Jakarta Messaging プロバイダーの例として使用します。他の Jakarta Messaging プロバイダーでは、異なる設定が必要になる場合があります。

重要

汎用 Jakarta Messaging リソースアダプターを使用する前に、Jakarta Messagingf プロバイダーについて、JBoss EAP と一緒に使用できる固有のリソースアダプターがあるかどうかを確認します。汎用 Jakarta Messaging Jakarta Connectors リソースアダプターは、Jakarta Messaging プロバイダーが独自のリソースアダプターを提供しない場合にのみ使用してください。

汎用リソースアダプターを設定する前に、以下を行う必要があります。

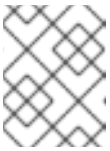
- Jakarta Messaging プロバイダーサーバーが設定されており、使用できる状態である必要があります。プロバイダーの JMS Messaging 実装に必要なバイナリーが必要になります。
- 接続ファクトリー、キュー、トピックなどの Jakarta Messaging リソースを検索できるようにするには、以下の Jakarta Messaging プロバイダープロパティーの値を知る必要があります。
  - **java.naming.factory.initial**
  - **java.naming.provider.url**
  - **java.naming.factory.url.pkgs**

この手順で使用する XML の例では、これらのパラメーターはそれぞれ、**PROVIDER_FACTORY_INITIAL**、**PROVIDER_URL**、**PROVIDER_CONNECTION_FACTORY** として記述されています。これらのプレースホルダーを、お使いの環境の Jakarta Messaging の値に置き換えてください。

31.7.1. サードパーティー Jakarta メッセージングプロバイダーで使用する汎用 Jakarta Messaging リソースアダプターの設定

1. リソースアダプターモジュールを作成および設定します。
   Jakarta Messaging プロバイダーへの接続および通信に必要なすべてのライブラリーが含まれる JBoss EAP モジュールを作成します。このモジュールの名前は **org.jboss.genericjms.provider** です。

   - 以下のディレクトリー構造を作成します。**EAP_HOME/modules/org/jboss/genericjms/provider/main**
   - プロバイダーの Jakarta Messaging 実装に必要なバイナリーを **EAP_HOME/modules/org/jboss/genericjms/provider/main** にコピーします。

     注記

     Tibco EMS の場合、必要なバイナリーは、Tibco インストールの **lib** ディレクトリーにある **tibjms .jar** と **tibcrypt.jar** です。

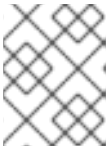
   - 以下のように、**EAP_HOME/modules/org/jboss/genericjms/provider/main** に **module.xml** ファイルを作成し、前の手順の JAR ファイルをリソースとして表示します。

     ```
     <module xmlns="urn:jboss:module:1.5" name="org.jboss.genericjms.provider">
       <resources>
         <!-- all jars required by the Jakarta Messaging provider, in this case Tibco -->
         <resource-root path="tibjms.jar"/>
         <resource-root path="tibcrypt.jar"/>
       </resources>
       <dependencies>
         <module name="javax.api"/>
         <module name="javax.jms.api"/>
       </dependencies>
     </module>
     ```

   - 以下の CLI コマンドを使用して **ee** サブシステムにモジュールを追加します。

     ```
     /subsystem=ee:list-add(name=global-modules, value={"name" =>
     "org.jboss.genericjms.provider", "slot" =>"main"}
     ```

2. Jakarta Messaging プロバイダーに対して Java Naming and Directory Interface 外部コンテキストを作成および設定します。
   接続ファクトリーや宛先などの Jakarta Messaging リソースは、Jakarta Messaging プロバイダーでルックアップされます。JBoss EAP インスタンスに外部コンテキストを追加し、このリソースの**ローカル**ルックアップがリモート Jakarta Messaging プロバイダー上のリソースを自動的に検索できるようにします。



注記

この手順では、**EAP_HOME/standalone/configuration/standalone-full.xml** を JBoss EAP 設定ファイルとして使用します。

管理 CLI を使用して外部の Java Naming and Directory Interface コンテキストを作成し、その設定プロパティーを追加します。以下の例のプロパティーは、リモート Jakarta Messaging プロバイダーに接続するために正しい値に置き換える必要があります。たとえば、Tibco EMS などの Jakarta Messaging プロバイダーは Java Naming and Directory Interface **lookup(Name)** メソッドをサポートしません。このような場合、この問題を回避するには、値が **true** の **org.jboss.as.naming.lookup.by.string** プロパティーを追加します。必要なプロパティーとその値に関する情報は、アダプターのドキュメントを参照してください。

```
/subsystem=naming/binding="java:global/remoteJMS":add(binding-type=external-
context,module=org.jboss.genericjms.provider,class=javax.naming.InitialContext,environment=
[java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory,java.naming.prc
vider.url=tcp://<hostname>:7222,org.jboss.as.naming.lookup.by.string=true])
```

外部コンテキストが適切に設定された状態で、**java:global/remoteJMS/** で始まるリソースへの Java Naming and Directory Interface ルックアップはリモート Jakarta Messaging プロバイダーで実行されます。たとえば、メッセージ駆動 Bean が **java:global/remoteJMS/Queue1** の Java Naming and Directory Interface ルックアップを実行する場合、外部コンテキストは、リモート Jakarta Messaging プロバイダーに接続し、**Queue1** リソースのルックアップを実行します。

Java Naming and Directory Interface 名を検索する際に、**external-context** を使用せずにリモートサーバーへの Java Naming and Directory Interface ルックアップを行うことができます。これを実行するには、以下の例にあるように CLI を使用して **external-context** を参照する新しいバインディングを作成します。

```
/subsystem=naming/binding=java\:\/jms\/queue\/myQueue:add(binding-type=lookup,
lookup=java:global/remoteJMS/jms/queue/myQueue)
```

上記の例では、**java:/jms/queue/myQueue** の Java Naming and Directory Interface ルックアップを実行するアプリケーションは、リモートサーバーの **myQueue** という名前のキューを見つけます。

3. 汎用 Jakarta Messaging リソースアダプターを作成します。
管理 CLI を使用してリソースアダプターを作成します。

```
/subsystem=resource-adapters/resource-adapter=generic-
ra:add(module=org.jboss.genericjms,transaction-support=XATransaction)
```

4. 汎用 Jakarta Messaging リソースアダプターを設定します。
管理 CLI を使用してリソースアダプターの **connection-definition** とその他の要素を設定します。

```
/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
cd:add(class-name=org.jboss.resource.adapter.jms.JmsManagedConnectionFactory, jndi-
name=java:/jms/XAQCF)

/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
cd/config-properties=ConnectionFactory:add(value=XAQCF)

/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-
```

```
cd/config-
properties=JndiParameters:add(value="java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory;java.naming.provider.url=tcp://<hostname>:7222")

/subsystem=resource-adapters/resource-adapter=generic-ra/connection-definitions=tibco-cd:write-attribute(name=security-application,value=true)
```

5. 汎用リソースアダプターを使用するように **ejb3** サブシステムのデフォルトのメッセージ駆動 Bean プールを設定します。

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-name, value=generic-ra)
```

汎用 Jakarta Messaging リソースアダプターが設定され、使用できる状態になりました。以下は、新しいメッセージ駆動 Bean を作成する際にリソースアダプターを使用する例です。

例: 汎用リソースアダプターを使用したコード

```
@MessageDriven(name = "HelloWorldQueueMDB", activationConfig = {
  // The generic Jakarta Messaging resource adapter requires the  Java Naming and Directory Interface bindings
  // for the actual remote connection factory and destination
  @ActivationConfigProperty(propertyName = "connectionFactory", propertyValue = "java:global/remoteJMS/XAQCF"),
  @ActivationConfigProperty(propertyName = "destination", propertyValue = "java:global/remoteJMS/Queue1"),
  @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
  @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge") })
  public class HelloWorldQueueMDB implements MessageListener {
  public void onMessage(Message message) {
  // called every time a message is received from the _Queue1_ queue on the Jakarta Messaging provider.
  }
}
```

重要

汎用 Jakarta Messaging リソースアダプターを使用する場合は、潜在的な **NullPointerException** エラーを回避するために、必ずセッションが処理されるように設定してください。このエラーが発生するのは、Jakarta EE 仕様で、パラメーターが処理され**ない**ことが示されているとき、汎用 Jakarta Messaging リソースアダプターがパラメーターの処理を試みるためです。これは、**connection.createSession(true, Session.SESSION_TRANSACTED);** を実行して達成できます。

プールされた接続ファクトリーをリソースアダプターから使用することもできます。

```
@Resource(lookup = "java:/jms/XAQCF")
private ConnectionFactory cf;
```

外部コンテキストから直接リソースを挿入することはできませんが、外部コンテキストを挿入してからルックアップを実行することはできます。たとえば、Tibco EMS ブローカーにデプロイされたキューのルックアップは以下のようになります。

```
@Resource(lookup = "java:global/remoteJMS")
private Context context;
...
Queue queue = (Queue) context.lookup("Queue1")
```

31.8. リソースアノテーションの使用

@Resource アノテーションを使用すると、Jakarta Enterprise Beans は Jakarta Messaging リソースまたは接続ファクトリーを直接挿入できます。**@Resource** アノテーションを使用して以下のパラメーターを指定できます。

- **lookup**
- **name**
- **mappedName**

リソースを挿入するには、これらのいずれかのパラメーターにリソースの Java Naming and Directory Interface (JNDI) 名を指定する必要があります。

31.8.1. Jakarta メッセージングリソースの挿入

1. 以下に示すようにキューを定義します。

```
<jms-queue name="OutQueue" entries="jms/queue/OutQueue
java:jboss/exported/jms/queue/OutQueue"/>
```

2. このキューは、**@Resource** アノテーションの **lookup**、**name**、または **mappedName** パラメーターに Java Naming and Directory Interface 名を指定して挿入します。例を以下に示します。

```
@Resource(lookup = "java:jboss/exported/jms/queue/OutQueue")
public Queue myOutQueue;
```

31.8.2. 接続ファクトリーの挿入

1. 以下に示すように接続ファクトリーを定義します。この例は、**JmsXA** プールされた接続ファクトリーを示しています。

```
<pooled-connection-factory name="activemq-ra" entries="java:/JmsXA
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm" transaction="xa"/>
```

2. 以下に示すように、デフォルトの **activemq-ra** プールされた接続ファクトリーを挿入します。

```
@Resource(lookup = "java:/JmsXA")
private ConnectionFactory cf;
```

31.8.3. 汎用 JMS リソースアダプターの制限と既知の問題

- 汎用 JMS リソースアダプターは、JMS サーバーと通信するために JMS API に依存します。JMS API は JMS リソースを作成するためのプログラムを使用した方法を提供しないため、「Java™ Platform, Enterprise Edition (Java EE) Specification, v7 」に定義されている以下の新機

能はサポートされません。

- EE.5.18.4 JMS 接続ファクトリーリソース定義
  これは、アプリケーションが JMS **ConnectionFactory** リソースを定義する機能です。

- EE.5.18.5 JMS 宛先定義
  これは、アプリケーションが JMS **Destination** リソースを定義する機能です。

第32章 後方互換性と前方互換性

JBoss EAP は JBoss EAP 6 のように HornetQ をメッセージングブローカーとして使用していた古いバージョンの JBoss EAP との後方互換性および前方互換性をサポートしています。この 2 つの互換性モードは、HornetQ のコアプロトコルをサポートする、JBoss EAP の組み込みメッセージングサーバーの ActiveMQ Artemis によって提供されます。

- 前方互換性: HornetQ を使用するレガシー JMS クライアントは、ActiveMQ Artemis を実行している JBoss EAP 7 サーバーに接続できます。
- 後方互換性: JBoss EAP メッセージングを使用する JBoss EAP 7 の JMS クライアントは、HornetQ を実行しているレガシー JBoss EAP 6 サーバーに接続できます。

32.1. 前方互換性

前方互換性では、レガシー JBoss EAP 6 の JMS クライアントへのコード変更は必要ありません。サポートは JBoss EAP **messaging-activemq** サブシステムおよびそのリソースによって提供されます。前方互換性のサポートを有効にするには、JBoss EAP 7 サーバーの設定に以下の変更を行います。手順ごとに、スタンドアロンサーバーの管理 CLI コマンドの例を示します。

- リモートレガシークライアントのポート 4447 でリッスンする **socket-binding** を作成します。

```
/socket-binding-group=standard-sockets/socket-binding=legacy-remoting:add(port=4447)
```

- 前の手順で作成した **socket-binding** を使用するレガシー **remote-connector** を作成します。これは JNDI ルックアップに必要です。

```
/subsystem=remoting/connector=legacy-remoting-connector:add(socket-binding=legacy-remoting)
```

- ポート 5445 でリッスンするレガシーメッセージング **socket-binding** を設定します。

```
/socket-binding-group=standard-sockets/socket-binding=legacy-messaging:add(port=5445)
```

- 前の手順のバインディングを使用する **messaging-activemq** サブシステムに **remote-connector** と **remote-acceptor** を設定します。

```
/subsystem=messaging-activemq/server=default/remote-connector=legacy-messaging-connector:add(socket-binding=legacy-messaging)

/subsystem=messaging-activemq/server=default/remote-acceptor=legacy-messaging-acceptor:add(socket-binding=legacy-messaging)
```

- **messaging-activemq** サブシステムの **legacy-connection-factory** 要素にレガシー HornetQ JMS ConnectionFactory を作成します。

```
/subsystem=messaging-activemq/server=default/legacy-connection-factory=legacy-discovery:add(entries=[java:jboss/exported/jms/LegacyRemoteConnectionFactory], connectors=[legacy-messaging-connector])
```

- レガシー HornetQ JMS 宛先を作成し、**legacy-entries** 属性を **jms-queue** または **jms-topic** リソースに含めます。

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

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

以下の例に従って、既存のキューまたはトピックに **legacy-entries** を追加できます。

```
/subsystem=messaging-activemq/server=default/jms-queue=myQueue:write-attribute(name=legacy-entries,value=[java:jboss/exported/jms/myQueue])
```

entries 属性が JBoss EAP メッセージング JMS クライアントによって使用されるのに対して、**legacy-entries** はレガシー HornetQ JMS クライアントによって使用されます。レガシー JMS クライアントは、このレガシー JMS リソースを検索し、JBoss EAP 7 と通信します。

注記

レガシー JMS クライアントでのコード変更を回避するには、**messaging-activemq** サブシステムに設定されたレガシー JNDI エントリーが、レガシー JMS クライアントによって想定されるルックアップと一致する必要があります。

管理 CLI の移行操作
管理 CLI の **migrate** 操作を実行して **messaging** サブシステム設定を更新する場合、ブール値の引数 **add-legacy-entries** が **true** に設定されていると、**messaging-activemq** サブシステムは **legacy-connection-factory** リソースを作成し、**legacy-entries** を **jms-queue** および **jms-topic** リソースに追加します。移行された **messaging-activemq** サブシステムのレガシーエントリーは、レガシー **messaging** サブシステムに指定されたエントリーに対応し、通常のエントリーは **-new** 接尾辞を付加して作成されます。

migrate 操作の実行時にブール値の引数 **add-legacy-entries** が **false** に設定されていると、**messaging-activemq** サブシステムにはレガシーリソースが作成されず、レガシー JMS クライアントは JBoss EAP 7 サーバーと通信できません。

32.2. 後方互換性

後方互換性では、レガシー JBoss EAP 7 サーバーの設定変更は必要ありません。JBoss EAP 7 の JMS クライアントはレガシーサーバー上でリソースを検索せず、代わりにクライアント側の JNDI を使用して JMS リソースを作成します。JBoss EAP 7 の JMS クライアントはこれらのリソースを使用して、HornetQ コアプロトコルを使用してレガシーサーバーと通信できます。

警告

JBoss EAP 7 の JBoss EAP 5 サーバーへのクライアント接続は現在サポートされていません。

JBoss EAP メッセージングは、クライアント側 JNDI をサポートし、JMS **ConnectionFactory** および **Destination** リソースを作成します。

たとえば、JBoss EAP 7 の JMS クライアントが「myQueue」という名前の JMS キューを使用してレガシーサーバーと通信する場合は、以下のプロパティーを使用して JNDI **InitialContext** を設定する必要があります。

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connectionFactory.jms/ConnectionFactory=tcp://<legacy server address>:5445? \

protocolManagerFactoryStr=org.apache.activemq.artemis.core.protocol.hornetq.client.HornetQClientProtocolManagerFactory
queue.jms/myQueue=myQueue
```

次に、クライアントは **jms/ConnectionFactory** 名を使用して JMS **ConnectionFactory** を作成し、**jms/myQueue** を使用して JMS **Queue** を作成できます。レガシー接続ファクトリーの URL を指定する場合、プロパティー **protocolManagerFactoryStr=org.apache.activemq.artemis.core.protocol.hornetq.client.HornetQClientProtocolManagerFactory** が必須であることに注意します。これにより、JBoss EAP メッセージング JMS クライアントはレガシーサーバーで HornetQ ブローカーと通信できます。

パート IV. パフォーマンスチューニング

第33章 メッセージング統計の監視

messaging-activemq サブシステムのメッセージングサーバーの統計収集が有効になっている場合、メッセージングサーバーのリソースに関するランタイム統計を表示できます。

33.1. メッセージング統計の有効化

パフォーマンスに悪影響を及ぼす可能性があるため、**messaging-activemq** サブシステムの統計収集はデフォルトで有効になって**いません**。キューのメッセージ数やキューに追加されたメッセージ数など、基本情報を取得するためにキュー統計を有効にする必要はありません。これらの統計は、**statistics-enabled** を **true** に設定しなくてもキュー属性を使用して利用できます。

追加の統計収集は、管理 CLI または管理コンソールを使用して有効にできます。

管理 CLI を使用したメッセージング統計の有効化

以下の管理 CLI コマンドは、**default** メッセージングサーバーの統計の収集を有効にします。

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

プールされた接続ファクトリーの統計は、他のメッセージングサーバー統計とは別に有効になります。以下のコマンドを使用して、プールされた接続ファクトリーの統計を有効にします。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra:write-attribute(name=statistics-enabled,value=true)
```

変更を反映するためにサーバーをリロードします。

管理コンソールを使用したメッセージング統計の有効化

管理コンソールを使用してメッセージングサーバーの統計収集を有効にするには、以下の手順に従います。

1. **設定** → **サブシステム** → **Messaging (ActiveMQ)** → **サーバー**に移動します。
2. サーバーを選択し、**表示**をクリックします。
3. **統計**タブの下の**編集**をクリックします。
4. **Statistics Enabled** フィールドを **ON** に設定し、**保存**をクリックします。

プールされた接続ファクトリーの統計は、他のメッセージングサーバー統計とは別に有効になります。プールされた接続ファクトリーの統計収集を有効にするには、以下の手順に従います。

1. **設定** → **サブシステム** → **Messaging (ActiveMQ)** → **サーバー**に移動します。
2. サーバーを選択し、**接続**を選択し、**表示**をクリックします。
3. **Pooled Connection Factory** タブを選択します。
4. プールされた接続ファクトリーを選択し、**属性**タブで **編集**をクリックします。
5. **Statistics Enabled** フィールドを **ON** に設定し、**保存**をクリックします。
6. 変更を反映するためにサーバーをリロードします。

33.2. メッセージング統計の表示

管理 CLI または管理コンソールを使用して、メッセージングサーバーのランタイム統計を表示できます。

管理 CLI を使用したメッセージング統計の表示

以下の管理 CLI コマンドを使用するとメッセージング統計を表示できます。統計はラインタイム情報であるため、必ず **include-runtime=true** 引数を指定してください。

- キューの統計を表示します。

```
/subsystem=messaging-activemq/server=default/jms-queue=DLQ:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "consumer-count" => 0,
        "dead-letter-address" => "jms.queue.DLQ",
        "delivering-count" => 0,
        "durable" => true,
        ...
    }
}
```

- トピックの統計を表示します。

```
/subsystem=messaging-activemq/server=default/jms-topic=testTopic:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "delivering-count" => 0,
        "durable-message-count" => 0,
        "durable-subscription-count" => 0,
        ...
    }
}
```

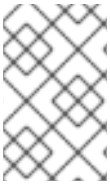
- プールされた接続ファクトリーの統計を表示します。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra/statistics=pool:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => 1,
        "AvailableCount" => 20,
        "AverageBlockingTime" => 0L,
        "AverageCreationTime" => 13L,
        "AverageGetTime" => 14L,
        ...
    }
}
```

注記

プールされた接続ファクトリーの統計は、他のメッセージングサーバー統計とは別に有効になります。手順は、「メッセージング統計の有効化」を参照してください。

管理コンソールを使用したメッセージング統計の表示
管理コンソールからメッセージング統計を表示するには、**ランタイム**タブで Messaging (ActiveMQ) サブシステムに移動し、サーバーを選択します。統計を表示する宛先を選択します。



注記

準備済みトランザクションページでは、準備済みトランザクションを表示、コミット、およびロールバックできます。詳細は、「メッセージングジャーナルの準備済みトランザクションの管理」を参照してください。

利用可能なすべての統計の詳細リストは、「メッセージング統計」を参照してください。

33.3. メッセージカウンターの設定

メッセージングサーバーの以下のメッセージカウンター属性を設定できます。

- **message-counter-max-day-history**: メッセージカウンター履歴が保持される日数。
- **message-counter-sample-period**: キューのサンプルとなる頻度 (ミリ秒単位)。

これらのオプションを設定する管理 CLI コマンドは以下の構文を使用します。**STATISTICS_NAME** と **STATISTICS_VALUE** は、設定する統計の名前と値に置き換えてください。

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

たとえば、以下のコマンドを使用して **message-counter-max-day-history** を 5 日に設定し、**message-counter-sample-period** を 2 秒に設定します。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=message-counter-max-day-history,value=5)
/subsystem=messaging-activemq/server=default:write-attribute(name=message-counter-sample-period,value=2000)
```

33.4. キューのメッセージカウンターと履歴の表示

以下の管理 CLI 操作を使用して、キューのメッセージカウンターとメッセージカウンター履歴を表示できます。

- **list-message-counter-as-json**
- **list-message-counter-as-html**
- **list-message-counter-history-as-json**
- **list-message-counter-history-as-html**

これらの値の表示に使用する管理 CLI コマンドは、以下の構文を使用します。**QUEUE_NAME** と **OPERATION_NAME** は、使用するキュー名と操作に置き換えてください。

```
/subsystem=messaging-activemq/server=default/jms-queue=QUEUE_NAME:OPERATION_NAME
```

たとえば、以下のコマンドを使用して、JSON 形式で **TestQueue** キューのメッセージカウンターを表示します。

```
/subsystem=messaging-activemq/server=default/jms-queue=TestQueue:list-message-counter-as-
json
{
    "outcome" => "success",
    "result" => "
{\"destinationName\":\"TestQueue\",\"destinationSubscription\":null,\"destinationDurable\":true,\"count\":
0,\"countDelta\":0,\"messageCount\":0,\"messageCountDelta\":0,\"lastAddTimestamp\":\"12/31/69
7:00:00 PM\",\"updateTimestamp\":\"2/20/18 2:24:05 PM\"}"
}
```

33.5. キューのメッセージカウンターのリセット

reset-message-counter 管理 CLI 操作を使用して、キューのメッセージカウンターをリセットできます。

```
/subsystem=messaging-activemq/server=default/jms-queue=TestQueue:reset-message-counter
{
    "outcome" => "success",
    "result" => undefined
}
```

33.6. 管理コンソールを使用したランタイム操作

管理コンソールを使用して、以下を操作できます。

- 別のメッセージングサーバーへの強制的フェイルオーバーを実行する
- メッセージングサーバーのすべてのメッセージカウンターをリセットする
- メッセージングサーバーのすべてのメッセージカウンター履歴をリセットする
- メッセージングサーバーに関連した情報を表示する
- メッセージングサーバーの接続を閉じる
- トランザクションをロールバックする
- トランザクションをコミットする

別のメッセージングサーバーへの強制的なフェイルオーバーの実行

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**

2. **メッセージング** ActiveMQ → **サーバー**をクリックします。
3. **表示**の横にある矢印ボタンをクリックして、Force Failover をクリックします。
4. Force Failover ウィンドウで、Yes をクリックします。

メッセージングサーバーのすべてのメッセージカウンターのリセット

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング** ActiveMQ → **サーバー**をクリックします。
3. **表示**の横にある矢印ボタンをクリックして、**リセット**をクリックします。
4. **リセット**ウィンドウで、**すべてのメッセージカウンターをリセットする**の横にあるトグルボタンをクリックして機能を有効にします。
   ボタンが青色の背景で **ON** になりました。
5. **リセット**をクリックします。

メッセージングサーバーのメッセージカウンター履歴のリセット

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング** ActiveMQ → **サーバー**をクリックします。
3. **表示**の横にある矢印ボタンをクリックして、**リセット**をクリックします。
4. **リセット**ウィンドウで、**すべてのメッセージカウンター履歴をリセットする**の横にあるトグルボタンをクリックして機能を有効にします。
   ボタンが青色の背景で **ON** になりました。
5. **リセット**をクリックします。

メッセージングサーバーに関連する情報の表示

管理コンソールを使用して、メッセージングサーバーに関連する以下の情報の一覧を表示できます。

- 接続
- コンシューマー
- プロデューサー
- コネクター
- ロール
- トランザクション

メッセージングサーバーに関連する情報を表示するには、以下を行います。

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインの該当する項目をクリックして、右側のペインの項目の一覧を表示します。

メッセージングサーバーの接続を閉じる

IP アドレス、ActiveMQ アドレス一致、またはユーザー名を指定して接続を閉じることができます。

メッセージングサーバーの接続を閉じるには、以下を行います。

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインで、**接続**をクリックします。
4. **閉じる**ウィンドウで、閉じる接続に基づいて該当するタブをクリックします。
5. 選択内容に基づいて、IP アドレス、ActiveMQ アドレス一致、またはユーザー名を入力し、**閉じる**をクリックします。

メッセージングサーバーのロールバックトランザクション

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインで、**Transactions** をクリックします。
4. ロールバックするトランザクションを選択し、**ロールバック**をクリックします。

メッセージングサーバーのトランザクションのコミット

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインで、**Transactions** をクリックします。
4. コミットするトランザクションを選択し、**コミット**をクリックします。

第34章 JMS の調整

JMS API を使用する場合は、パフォーマンスを改善するためのヒントについて、以下の情報を確認してください。

- メッセージ ID を無効にします。
  メッセージ ID が必要ない場合は、**MessageProducer** クラスで **setDisableMessageID()** メソッドを使用して無効にします。値を **true** に設定すると、一意の ID 作成のオーバーヘッドがなくなり、メッセージのサイズが小さくなります。

- メッセージのタイムスタンプを無効にします。
  メッセージのタイムスタンプが必要ない場合は、**MessageProducer** クラスで **setDisableMessageTimeStamp()** メソッドを使用して無効にします。値を **true** に設定すると、タイムスタンプ作成のオーバーヘッドがなくなり、メッセージのサイズが減少します。

- **ObjectMessage** を使用しません。
  ObjectMessage は、シリアライズされたオブジェクトを含むメッセージを送信するために使用されます。つまり、メッセージの本文 (ペイロード) が、バイトストリームとしてネットワーク経由で送信されます。オブジェクトが小さくても Java のシリアライズ形式は非常に大きく、ネットワーク上の多くのスペースを占有します。また、カスタムマーシャリング手法と比較すると低速です。**ObjectMessage** は、他のメッセージタイプの 1 つを使用できない場合にのみ使用します。たとえば、ランタイムまでペイロードタイプがわからない場合に使用します。

- **AUTO_ACKNOWLEDGE** を使用しません。
  コンシューマーで確認応答モードを選択すると、ネットワーク経由で送信される確認応答メッセージの送信により発生する追加のオーバーヘッドとトラフィックによってパフォーマンスに影響します。**AUTO_ACKNOWLEDGE** でこのオーバーヘッドが発生するのは、クライアント上で受信される各メッセージについてサーバーから確認応答の送信が必要であるためです。可能であれば **DUPS_OK_ACKNOWLEDGE** を使用して、レイジー方法でメッセージを確認応答します。**CLIENT_ACKNOWLEDGE** は、クライアントコードがメソッドを呼び出してメッセージを確認応答するか、またはトランザクションセッションの 1 つの確認応答またはコミットとして数多くの確認応答をまとめます。

- 永続メッセージを使用しません。
  デフォルトでは、JMS メッセージは永続化されます。永続メッセージが必要ない場合は、それらを **non-durable** に設定します。永続メッセージは、ストレージに永続化されるために多くのオーバーヘッドが発生します。

- **TRANSACTED_SESSION** モードを使用して、単一のトランザクションでメッセージを送受信します。
  単一のトランザクションでメッセージをバッチ処理することで、JBoss EAP に統合された ActiveMQ Artemis サーバーで必要なネットワークラウンドトリップは、送信または受信ごとではなく、コミット時に 1 つのみになります。

第35章 永続性の調整

- メッセージジャーナルを独自の物理ボリュームに配置します。
  追加のみのジャーナルの利点の1つは、ディスクヘッド移動が最小限に抑えられることです。ディスクが共有されている場合は、この利点は失われます。トランザクションコーディネーター、データベース、その他のジャーナルなど、複数のプロセスが同じディスクから読み書きされる場合、ディスクヘッドが異なるファイル間でスキップする必要があるため、パフォーマンスに影響を及ぼします。ページングまたは大きいメッセージを使用している場合は、それらが別のボリュームに配置されていることを確認します。

- **journal-min-files** 値を調整します。
  journal-min-files パラメーターを、平均的に持続可能な割合に一致するファイル数に設定します。ジャーナルデータディレクトリーに新しいファイルが頻繁に作成される場合は、大量のデータが保持されるため、ファイルの最小数を増やす必要があります。これにより、新規データファイルを作成せず、ジャーナルを再利用できます。

- ジャーナルファイルサイズを最適化します。
  ジャーナルファイルのサイズは、ディスク上のシリンダーの容量に合わせて調整する必要があります。多くのシステムでは、デフォルト値の **10 MB** で十分です。

- **AIO** ジャーナルタイプを使用します。
  Linux オペレーティングシステムの場合、ジャーナルタイプは **AIO** のままにします。**AIO** の拡張性は、Java **NIO** よりも優れています。

- **journal-buffer-timeout** 値を調整します。
  journal-buffer-timeout 値を増やすと、スループットは向上しますが、待ち時間が犠牲になります。

- **journal-max-io** 値を調整します。
  AIO を使用している場合は、**journal-max-io** パラメーター値を増やして、パフォーマンスを向上させることができる可能性があります。**NIO** を使用している場合は、この値を変更しないでください。

第36章 その他の調整オプション

このセクションでは、JBoss EAP メッセージングで、調整できる他の箇所を説明します。

- 非同期送信の確認応答を使用します。
  トランザクション以外の永続メッセージを送信する必要があり、**send()** の呼び出しが返るまでにメッセージがサーバーに到達したという保証が必要ない場合は、ブロック送信するように設定しないでください。代わりに、非同期送信の確認応答を使用して、別のストリームで返される送信の確認応答を取得します。ただし、サーバーがクラッシュした場合に、一部のメッセージが失わる可能性があります。

- **pre-acknowledge** モードを使用します。
  pre-acknowledge モードでは、メッセージはクライアントに送信される前に確認応答されます。これにより、ネットワーク上の確認応答トラフィックの量が減少します。ただし、クライアントがクラッシュすると、クライアントが再接続した場合にメッセージは再配信されません。

- セキュリティーを無効にします。
  security-enabled 属性を false に設定してセキュリティーを無効にすると、パフォーマンスが若干改善されます。

- 永続性を無効にします。
  persistence-enabled を **false** に設定すると、メッセージ永続性を完全にオフにできます。

- トランザクションを遅れて同期します。
  journal-sync-transactional を **false** に設定すると、トランザクションの永続的なパフォーマンスが向上しますが、障害時にトランザクションが失われる可能性がいくらかあります。

- 非トランザクションを遅れて同期します。
  journal-sync-non-transactional を **false** に設定すると、非トランザクションの永続的なパフォーマンスが向上しますが、障害時に永続メッセージが失われる可能性がいくらかあります。

- メッセージを非ブロックで送信します。
  送信されるすべてのメッセージのネットワークラウンドトリップの待機を回避するには、JMS および JNDI を使用している場合は、**block-on-durable-send** と **block-on-non-durable-send** を **false** に設定するか、**setBlockOnDurableSend()** メソッドと **setBlockOnNonDurableSend()** メソッドを呼び出して **ServerLocator** に直接設定します。

- **consumer-window-size** を最適化します。
  高速なコンシューマーがある場合には、**consumer-window-size** を増やしてコンシューマーフロー制御を効果的に無効にできます。

- JMS API の代わりにコア API を使用します。
  JMS 操作は、サーバーが処理する前にコア操作に変換する必要があるため、コア API を使用する場合よりもパフォーマンスが低下します。コア API を使用する場合は、可能な限り **SimpleString** を取得するメソッドを使用するようにします。**SimpleString** は、**java.lang.String** とは異なり、ネットワークに書き込まれる前にコピーする必要がないため、呼び出し間で **SimpleString** インスタンスを再利用する場合に、不要なコピーを避けることができます。コア API は他のブローカーに移植できないことに注意してください。

第37章 アンチパターンの回避

- 可能な場合は、接続、セッション、コンシューマー、プロデューサーを再利用します。
  メッセージングの最も一般的なアンチパターンは、送信または消費されるメッセージごとに新しい接続、セッション、プロデューサーの作成です。これらのオブジェクトは作成に時間がかかり、複数のネットワークラウンドトリップを伴う可能性があるため、リソースの使用率が低くなります。常に再利用します。

注記

Spring Messaging Template テンプレートなどの一般的なライブラリーは、これらのアンチパターンを使用します。Spring Messaging Template を使用している場合は、パフォーマンスが低下する可能性があります。Spring Messaging Template は、たとえば JCA を使用して JMS セッションをキャッシュするアプリケーションサーバーでのみ安全に使用でき、その後メッセージを送信するためにのみ使用できます。アプリケーションサーバーであっても、同期的に消費するメッセージに安全に使用することはできません。

- サイズの大きいメッセージを使用しません。
  XML のような詳細形式は、ネットワーク上の多くのスペースを占有し、結果としてパフォーマンスが低下します。可能な場合は、メッセージ本文では XML を使用しません。

- 各リクエストに一時キューを作成しません。
  この一般的なアンチパターンには、一時キューの要求/応答パターンが含まれます。一時キュー要求/応答パターンにより、メッセージはターゲットに送信され、返信先ヘッダーはローカルの一時キューのアドレスで設定されます。受信側がメッセージを受信すると、メッセージを処理し、返信先ヘッダーに指定されたアドレスに応答を返します。このパターンでよくある間違いは、送信されるメッセージごとに新しい一時キューを作成することです。これにより、パフォーマンスが大幅に低下します。代わりに、一時キューを多くのリクエストに再利用する必要があります。

- 必要でない限り、メッセージ駆動 Bean は使用しないでください。
  MDB を使用したメッセージの消費は、単純な JMS メッセージコンシューマーを使用するメッセージの消費よりも遅くなります。

付録A リファレンス資料

A.1. アドレス設定の属性

表A.1 アドレス設定の属性

名前	説明
address-full-policy	**max-size-bytes** が指定されているアドレスがいっぱいになるとどうなるかを決定します。許可される値は、**PAGE**、**DROP**、**FAIL**、または **BLOCK** 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** がデフォルトです。ページングの詳細は、「ページングについて」を参照してください。
auto-create-jms-queues	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、JBoss EAP 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。デフォルトは **false** です。**非推奨: 代わりに auto-create-queues を使用してください。**
auto-create-jms-topics	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、JBoss EAP 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。デフォルトは **false** です。**非推奨: 代わりに auto-create-addresses を使用してください。**
auto-create-addresses	メッセージが送信されるか、コンシューマーがアドレス **match** に合う名前のキューに接続を試行するとき、ブローカーがアドレスを自動的に作成するかどうかを決定します。自動作成されるキューは永続性 (非一時的) です。デフォルトは **true** です。
auto-create-queues	メッセージが送信される、またはコンシューマーがアドレス**match** に合う名前のキューに接続しようとするとき、ブローカーがキューを自動的に作成するかどうかを決定します。自動作成されるキューは永続性 (非一時的) です。デフォルトは **true** です。
auto-delete-jms-queues	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、JBoss EAP が自動的に削除するかどうかを決定します。デフォルトは **false** です。**非推奨: 代わりに auto-delete-queues を使用してください。**
auto-delete-jms-topics	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、JBoss EAP が自動的に削除するかどうかを決定します。デフォルトは **false** です。**非推奨: 代わりに auto-delete-addresses を使用してください。**
auto-delete-addresses	自動作成されたアドレスにキューがなくなったら、ブローカーが自動的に削除するかどうかを決定します。デフォルトは **true** です。

名前	説明
auto-delete-queues	自動作成されたキューのコンシューマーが **0** で、メッセージも **0** の場合は、ブローカーが自動的に削除するかどうかを決定します。デフォルトは **true** です。
dead-letter-address	デッドメッセージの送信先アドレス。詳細は、「デッドレターアドレスの設定」を参照してください。
expiry-address	期限切れのメッセージを受信するアドレス。詳細は、「メッセージ有効期限の設定」を参照してください。
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。デフォルトは **-1** です。
last-value-queue	キューが最後の値のみを使用するかどうかを定義します。詳細は、「last-value キュー」を参照してください。
max-delivery-attempts	キャンセルされたメッセージを dead-letter-address に送信する前に再配信できる回数を定義します。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)。デフォルトは **0** です。
max-size-bytes	このアドレスの最大サイズ (バイト単位)。デフォルトは **-1** です。
message-counter-history-day-limit	メッセージカウンター履歴の日数制限。デフォルトは **0** です。
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メモリー内に保持するページファイルの数。デフォルトは **5** です。
page-size-bytes	ページングサイズ (バイト単位)。デフォルトは **10485760** です。
redelivery-delay	キャンセルされたメッセージの再配信を試行するまでの待ち時間 (ミリ秒単位) を定義します。デフォルトは **0** です。詳細は、「遅延再配信の設定」を参照してください。
redelivery-multiplier	redelivery-delay パラメーターに適用する乗数。デフォルトは **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メッセージを再分配するまでの待機時間 (ミリ秒単位) を定義します。デフォルトは **-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、設定済みのデッドレターアドレスアドレスに送信されます。デフォルトは **false** です。

名前	説明
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。同じ接続を使用するどのクライアントスレッドにも影響を与えます。**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。デフォルトは **-1** です。

A.2. 接続ファクトリーの属性

表A.2 接続ファクトリーの属性

属性	説明
auto-group	メッセージのグループ化が自動的に使用されるかどうか。
block-on-acknowledge	確認応答時にブロックするかどうか。
block-on-durable-send	永続送信時にブロックするかどうか。
block-on-non-durable-send	非永続送信時にブロックするかどうか。
cache-large-message-client	大きいメッセージをキャッシュするかどうか。
call-failover-timeout	フェイルオーバーの処理時に使用するタイムアウト (ミリ秒単位)。
call-timeout	呼び出しのタイムアウト (ミリ秒単位)。
client-failure-check-period	クライアント障害チェックの間隔 (ミリ秒単位)。
client-id	クライアント ID。
compress-large-messages	大きいメッセージを圧縮すべきかどうか。
confirmation-window-size	確認ウィンドウサイズ (バイト単位)。
connection-load-balancing-policy-class-name	クライアントがクラスター内の異なるノード間でセッションをロードバランシングするために使用できるクライアント側のロードバランシングポリシーを実装するクラスの名前。
connection-ttl	接続 Time-to-Live (ミリ秒単位)。

属性	説明
connectors	コネクターを定義します。これは、コネクター名 (未定義値を含む) でマップに保存されます。この属性を書き込むとき、コネクター名のリストを渡すことが可能です。
consumer-max-rate	コンシューマーの 1 秒あたりの最大レート。
consumer-window-size	コンシューマーウィンドウサイズ (バイト単位)。
deserialization-black-list	デシリアライズが許可されないクラスまたはパッケージのリストを定義します。
deserialization-white-list	デシリアライズが許可されるクラスまたはパッケージのリストを定義します。
discovery-group	検出グループ名。
dups-ok-batch-size	重複 OK のバッチサイズ。
enable-amq1-prefix	JMS 宛先名で接頭辞を含めるかどうかを指定します。値が false の場合、接頭辞は含まれません。
entries	接続ファクトリーがバインドされる必要がある JNDI 名。
factory-type	接続ファクトリーのタイプ。有効な値は次のとおりです。 • **GENERIC** • **TOPIC** • **QUEUE** • **XA_GENERIC** • **XA_QUEUE** • **XA_TOPIC** **GENERIC** はブローカーへの汎用接続に使用し、**TOPIC** と **QUEUE** はそれぞれの JMS タイプへの接続に使用する必要があります。XA で始まるタイプは、トランザクションメッセージングに使用する必要があります。
failover-on-initial-connection	初期接続でフェイルオーバーするかどうか。
group-id	グループ ID。
ha	接続ファクトリーが高可用性に対応しているかどうか。
max-retry-interval	最大再試行間隔 (ミリ秒単位)。

属性	説明
min-large-message-size	大きいメッセージの最小サイズ (バイト単位)。
pre-acknowledge	事前確認応答を行うかどうか。
producer-max-rate	プロデューサーの 1 秒あたりの最大レート。
producer-window-size	プロデューサーウィンドウサイズ (バイト単位)。
protocol-manager-factory	この接続ファクトリーによって使用されるプロトコルマネージャーファクトリー。
reconnect-attempts	再接続試行回数。
retry-interval	再試行の間隔 (ミリ秒単位)。
retry-interval-multiplier	再試行間隔の乗数。
scheduled-thread-pool-max-size	スケジュールスレッドプールの最大サイズ。
thread-pool-max-size	スレッドプールの最大サイズ。
transaction-batch-size	トランザクションのバッチサイズ。
use-global-pools	グローバルプールを使用するかどうか。
use-topology-for-load-balancing	メッセージングクライアントがクラスタートポロジーを使用してクラスターに接続するか、またはすべての接続に初期コネクターを使用するかどうかを指定します。

A.3. プールされた接続ファクトリーの属性

表A.3 プールされた接続ファクトリーの属性

属性	説明
allow-local-transactions	アウトバウンド JMS セッションにローカルトランザクションを許可するかどうか。 **true** に設定すると、JMS プロデューサーが以下の状況でメッセージを送信できます。 ● トランザクションセッションのサーブレットから。 ● トランザクションタイプ属性が **NOT_SUPPORTED** に設定されたトランザクションセッションの MDB から。 この属性は、明示的に禁止する **JMSContext** には適用されません。

属性	説明
auto-group	メッセージのグループ化が自動的に使用されるかどうか。
block-on-acknowledge	確認応答時にブロックするかどうか。
block-on-durable-send	永続送信時にブロックするかどうか。
block-on-non-durable-send	非永続送信時にブロックするかどうか。
cache-large-message-client	大きいメッセージをキャッシュするかどうか。
call-failover-timeout	フェイルオーバーの処理時に使用するタイムアウト (ミリ秒単位)。
call-timeout	呼び出しのタイムアウト (ミリ秒単位)。
client-failure-check-period	クライアント障害チェックの間隔 (ミリ秒単位)。
client-id	クライアント ID。
compress-large-messages	大きいメッセージを圧縮すべきかどうか。
confirmation-window-size	確認ウィンドウサイズ (バイト単位)。
connection-load-balancing-policy-class-name	クライアントがクラスター内の異なるノード間でセッションをロードバランシングするために使用できるクライアント側のロードバランシングポリシーを実装するクラスの名前。
connection-ttl	接続 Time-to-Live (ミリ秒単位)。
connectors	コネクターを定義します。これは、コネクター名 (未定義値を含む) でマップに保存されます。この属性を書き込むとき、コネクター名のリストを渡すことが可能です。
consumer-max-rate	コンシューマーの 1 秒あたりの最大大レート。
consumer-window-size	コンシューマーウィンドウサイズ (バイト単位)。
credential-reference	プールされた接続ファクトリーを認証するための、クレデンシャルストアの認証情報。
deserialization-black-list	デシリアライズが許可されないクラスまたはパッケージのリストを定義します。
deserialization-white-list	デシリアライズが許可されるクラスまたはパッケージのリストを定義します。

属性	説明
discovery-group	検出グループ名。
dups-ok-batch-size	重複 OK のバッチサイズ。
enable-amq1-prefix	JMS 宛先名で接頭辞を含めるかどうかを指定します。値が false の場合、接頭辞は含まれません。
enlistment-trace	IronJacamar で、このプールされた接続ファクトリーのエンリストメントトレースを記録できるようにします。この属性はデフォルトでは定義されず、**ironjacamar.disable_enlistment_trace** システムプロパティーが存在することで動作が実行されます。
entries	接続ファクトリーがバインドされる必要がある JNDI 名。
failover-on-initial-connection	初期接続でフェイルオーバーするかどうか。
group-id	グループ ID。
ha	接続ファクトリーが高可用性に対応しているかどうか。
initial-connect-attempts	最初にこのファクトリーで接続を試行する回数。
initial-message-packet-size	このファクトリーから作成されるメッセージの初期サイズ。
jndi-params	受信接続の宛先を見つけるために使用する JNDI パラメーター。
managed-connection-pool	このプールされた接続ファクトリーによって使用される管理対象の接続プールのクラス名。
max-pool-size	プールの最大サイズ。
max-retry-interval	最大再試行間隔 (ミリ秒単位)。
min-large-message-size	大きいメッセージの最小サイズ (バイト単位)。
min-pool-size	プールの最小サイズ。
password	この接続ファクトリーで使用するデフォルトのパスワード。これは、接続ファクトリーをリモートホストへポイントする場合にのみ必要です。
pre-acknowledge	事前確認応答を行うかどうか。
producer-max-rate	プロデューサーの 1 秒あたりの最大レート。

属性	説明
producer-window-size	プロデューサーウィンドウサイズ (バイト単位)。
protocol-manager-factory	このプールされた接続ファクトリーによって使用されるプロトコルマネージャーファクトリー。
reconnect-attempts	再接続試行回数。デフォルトで、プールされた接続ファクトリーは、メッセージングサーバーに無限に再接続を試みます。
retry-interval	再試行の間隔 (ミリ秒単位)。
retry-interval-multiplier	再試行間隔の乗数。
scheduled-thread-pool-max-size	スケジュールスレッドプールの最大サイズ。
setup-attempts	MDB エンドポイントを設定する回数。
setup-interval	MDB エンドポイントの設定を試みる間隔 (ミリ秒単位)。
statistics-enabled	ランタイム統計が有効になっているかどうか。
thread-pool-max-size	スレッドプールの最大サイズ。
transaction	トランザクションモード。
transaction-batch-size	トランザクションのバッチサイズ。
use-auto-recovery	自動リカバリーを使用するかどうか。
use-global-pools	グローバルプールを使用するかどうか。
use-jndi	JNDI を使用して、受信接続の宛先を見つけます。
use-local-tx	受信セッションにローカルトランザクションを使用します。
use-topology-for-load-balancing	メッセージングクライアントがクラスタートポロジーを使用してクラスターに接続するか、またはすべての接続に初期コネクターを使用するかどうかを指定します。
user	この接続ファクトリーで使用するデフォルトのユーザー名。これは、接続ファクトリーをリモートホストへポイントする場合にのみ必要です。

A.4. コアブリッジの属性

表A.4 コアブリッジの属性

属性	説明
check-period	クライアント障害チェックの間隔 (ミリ秒単位)。
confirmation-window-size	メッセージのターゲットノードへの転送に使用される接続に使用するサイズ。
connection-ttl	ハートビートがない場合に、ブリッジで使用される接続が動作していると見なされる最大時間 (ミリ秒単位)。
credential-reference	ブリッジを認証するための、クレデンシャルストアの認証情報。
discovery-group	このブリッジによって使用される検出グループの名前。この属性は、**static-connectors** が定義されている場合は設定できません。
filter	オプションのフィルター文字列。指定した場合、指定したフィルター式に一致するメッセージのみが転送されます。
forwarding-address	メッセージが転送されるターゲットサーバーのアドレス。転送アドレスを指定しない場合は、メッセージの元の宛先が保持されます。
ha	このブリッジが高可用性をサポートするべきかどうか。**true** の場合、クラスターの利用可能なサーバーに接続し、フェイルオーバーをサポートします。デフォルト値は **false** です。
initial-connect-attempts	最初にこのブリッジで接続を試みる回数。
max-retry-interval	接続を再試行するために使用される最大間隔。
min-large-message-size	このサイズ (バイト単位) 以上のメッセージが、大きいメッセージと見なされます。
password	リモートサーバーへのブリッジ接続の作成時に使用するパスワード。これを指定しない場合、**messaging-activemq** サブシステムの **cluster-password** 属性によって指定されたデフォルトのクラスターパスワードが使用されます。
cluster-credential-reference	クラスターへの認証に使用されるクレデンシャルストア参照。これは、**password** の代わりに使用できます。
queue-name	ブリッジが消費するローカルキューの一意の名前。このキューは、起動時にブリッジがインスタンス化されるときにはすでに存在している必要があります。
reconnect-attempts	断念してシャットダウンするまでにブリッジが行う再接続試行の総回数。デフォルト値は **-1** で、無制限の試行回数を示します。

属性	説明
reconnect-attempts-on-same-node	断念してシャットダウンするまでにブリッジが同じノード上で行う再接続試行の総回数。**-1** の値は、無制限の試行回数を示します。デフォルトは **10** です。
retry-interval	ターゲットサーバーへの接続に失敗した場合の後続の再接続試行の間隔 (ミリ秒単位)。
retry-interval-multiplier	最後の再試行から次の再試行までの時間を計算するために時間に適用する乗数。これにより、再試行の間隔に指数バックオフを実装することができます。
static-connectors	このブリッジによって使用される静的に定義されたコネクターの一覧。**discovery-group** が定義されている場合、この属性は設定できません。
transformer-class-name	**org.apache.activemq.artemis.core.server.cluster.Transformer** インターフェースを実装するユーザー定義クラスの名前。
use-duplicate-detection	ブリッジが、転送する各メッセージに重複 ID プロパティーを自動的に挿入するかどうか。
user	リモートサーバーへのブリッジ接続の作成時に使用するユーザー名。指定しない場合、**messaging-activemq** サブシステムの **cluster-user** 属性に指定されたデフォルトのクラスターユーザーが使用されます。

A.5. JMS ブリッジの属性

表A.5 JMS ブリッジの属性

属性	説明
add-messageID-in-header	これを **true** に設定すると、元のメッセージのメッセージ ID が、ヘッダー **AMQ_BRIDGE_MSG_ID_LIST** 内の宛先に送信されるメッセージに付加されます。メッセージが複数回ブリッジされる場合は、毎回メッセージ ID が付加されます。
client-id	永続性があり、ソース宛先がトピックである場合は、サブスクリプションを作成し、検索する際に使用する JMS クライアント ID。
failure-retry-interval	ブリッジで障害が発生したことを検出したときに、ソースサーバーまたはターゲットサーバーへの接続の再作成を試行するまで待機する時間 (ミリ秒単位)。
max-batch-size	バッチでターゲット宛先に送信するまでソース宛先から消費するメッセージの最大数。値は **1** 以上である必要があります。

属性	説明
max-batch-time	消費されたメッセージの数が max-batch-size に達していない場合を含め、メッセージのバッチをターゲットに送信するまで待機する最大時間 (ミリ秒単位)。**-1** を値として指定すると、永久に待機します。
max-retries	ブリッジで障害が発生したことを検出したときに、ソースサーバーまたはターゲットサーバーへの接続の再作成を試行する回数。ブリッジは、この回数を試行した後に試行を断念します。**-1** を値として指定すると、永久に試行します。
module	ソースおよびターゲット JMS リソースを検索するのに必要なリソースが含まれる JBoss EAP モジュールの名前。
paused	JMS ブリッジが一時停止されているかどうかを報告する読み取り専用プロパティー。
quality-of-service	望ましいサービス品質モード。使用できる値は、**AT_MOST_ONCE**、**DUPLICATES_OK**、または **ONCE_AND_ONLY_ONCE** です。さまざまなモードの詳細は、「サービス品質」を参照してください。
selector	ソース宛先からのメッセージ消費に使用される JMS セレクターの式。セレクターの式に一致するメッセージのみがソースからターゲット宛先にブリッジされます。
subscription-name	永続性があり、ソース宛先がトピックである場合のサブスクリプションの名前。
source-connection-factory	ソースメッセージングサーバーで検索するソース接続ファクトリーの名前。
source-context	ソース JNDI 初期コンテキストの設定に使用されるプロパティー。
source-credential-reference	ソース接続の認証に使用されるクレデンシャルストア参照。これは **source-password** の代わりに使用できます。
source-destination	ソースメッセージングサーバーで検索するソース宛先の名前。
source-password	ソース接続を作成するためのパスワード。
source-user	ソース接続を作成するためのユーザーの名前。
target-connection-factory	ターゲットメッセージングサーバーで検索するターゲット接続ファクトリーの名前。

属性	説明
target-context	ターゲット JNDI 初期コンテキストを設定するために使用されるプロパティー。
target-credential-reference	ターゲット接続の認証に使用されるクレデンシャルストア参照。これは、**target-password** の代わりに使用できます。
target-destination	ターゲットメッセージングサーバーで検索するターゲット宛先の名前。
target-password	ターゲット接続を作成するためのパスワード。
target-user	ターゲット接続を作成するためのユーザーの名前。

A.6. クラスター接続の属性

表A.6 クラスター接続の属性

属性	説明
allow-direct-connections-only	**true** に設定した場合、このノードは、1 ホップより離れていると、クラスター内の別のノードへの接続を作成**しません**。**static-connectors** 属性が定義される場合にのみ使用されます。デフォルトは **false** です。
call-failover-timeout	クラスター接続によって行われるリモート呼び出しのフェイルオーバーが実行中の場合に使用するタイムアウト (ミリ秒単位)。デフォルトは **-1** で、バインドされません。
call-timeout	クラスター接続によって行われるリモート呼び出しのタイムアウト (ミリ秒単位)。デフォルトは **30000** (30 秒) です。
check-period	クライアント障害チェックの間隔 (ミリ秒単位)。デフォルトは **30000** (30 秒) です。
cluster-connection-address	各クラスター接続は、この値で始まるアドレスに送信されるメッセージのみに適用されます。
confirmation-window-size	メッセージをターゲットノードに転送するために使用される接続のウィンドウサイズ (バイト単位)。デフォルトは **1048576** です。
connection-ttl	ハートビートがない場合に、クラスター接続で使用される接続が動作していると見なされる最大時間 (ミリ秒単位)。デフォルトは **60000** (60 秒) です。
connector-name	クラスター接続に使用するコネクターの名前。

属性	説明
discovery-group	このクラスター接続が接続を確立するクラスター内の他のサーバーのリストを取得するために使用される検出グループ。**static-connectors** が定義されている場合は、未定義 (null) である必要があります。
initial-connect-attempts	最初にこのクラスター接続で接続を試みる回数。デフォルトは **-1** で、バインドされません。
max-hops	メッセージの転送最大回数。JBoss EAP は、チェーンの中間として他の ActiveMQ Artemis メッセージングサーバーと間接的にのみ接続される可能性のあるノードにメッセージを負荷分散するように設定することもできます。デフォルトは **1** です。
max-retry-interval	接続を再試行するために使用される最大時間間隔 (ミリ秒単位)。デフォルトは **2000** (2 秒) です。
message-load-balancing-type	このパラメーターは、クラスター内の他のノード間でメッセージを分配する方法を決定します。非推奨の **forward-when-no-consumers** に取って代わります。有効な値は、**OFF**、**STRICT**、または **ON_DEMAND** です。 **OFF** メッセージはクラスター内の別のノードへ転送されません。 **STRICT** クラスターの他のノードの同じキューにコンシューマーがない場合や、一致しないメッセージフィルターまたはセレクターを持つコンシューマーがある場合でも、メッセージはラウンドロビン方式で分配されます。このパラメーターが **STRICT** に設定されていても、他のノードに同じ名前のキューがない場合は、JBoss EAP は他のノードへメッセージを転送しません。**STRICT** の使用は、レガシーの **forward-when-no-consumers** パラメーターを **true** に設定するのと同様になります。 **ON_DEMAND** 転送アドレスにコンシューマーを持つキューがある場合、メッセージはクラスターの他のノードに転送されます。このコンシューマーにメッセージフィルターまたはセレクターがある場合、これらのセレクターの少なくとも 1 つがメッセージと一致する必要があります。**ON_DEMAND** の使用は、レガシーの **forward-when-no-consumers** パラメーターを **false** に設定するのと同様になります。 デフォルトは **ON_DEMAND** です。
min-large-message-size	このサイズ (バイト単位) 以上のメッセージが、大きいメッセージと見なされます。デフォルトは **102400** です。
node-id	このクラスター接続によって使用されるノード ID。この属性は読み取り専用です。

属性	説明
notification-attempts	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。デフォルトは **2** です。
notification-interval	通知の間隔 (ミリ秒単位)。デフォルトは **10000** (10 秒) です。
reconnect-attempts	断念してシャットダウンするまでにブリッジが行う再接続試行の総回数。デフォルトは **-1** で、無制限の試行回数を示します。
retry-interval	ターゲットサーバーへの接続に失敗した場合のターゲットサーバーへの後続の再接続試行の間隔 (ミリ秒単位)。デフォルトは **500** です。
retry-interval-multiplier	最後の再試行から次の再試行までの時間を計算するために時間に適用する乗数。これにより、再試行の間隔に指数バックオフを実装することができます。デフォルトは **1.0** です。
static-connectors	このクラスター接続が接続を確立するコネクターの静的に定義されたリスト。**discovery-group-name** が定義されている場合は、未定義である必要があります。
topology	このクラスター接続が認識しているノードのトポロジー。この属性は読み取り専用です。
use-duplicate-detection	ブリッジが、転送する各メッセージに重複 ID プロパティーを自動的に挿入するかどうか。デフォルトは **true** です。

A.7. メッセージング統計

キューの統計

表A.7 キューの統計

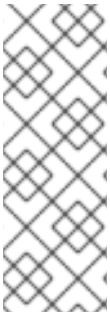
統計	説明
コンシューマー数	このキューからメッセージを消費するコンシューマーの数。
メッセージ数	このキューに現在あるメッセージの数。
メッセージの追加数	作成以降、このキューに追加されたメッセージの数。
スケジュールされた数	このキューにスケジュールされているメッセージの数。

トピックの統計

表A.8 トピックの統計

統計	説明
配信数	このトピックが現在コンシューマーに配信しているメッセージの数。
永続メッセージ数	このトピックのすべての永続的サブスクライバーに対するメッセージの数。
永続性のあるサブスクリプション数	このトピックの永続的サブスクライバーの数。
メッセージ数	このトピックに現在あるメッセージの数。
メッセージの追加	作成以降、このトピックに追加されたメッセージの数。
サブスクリプション数	このトピックの永続的サブスクライバーと非永続的サブスクライバーの数です。

プールされた接続ファクトリーの統計



注記

プールされた接続ファクトリーの統計収集は、メッセージングサーバー用に収集される他の統計とは別に有効になります。以下の管理 CLI コマンドを使用して、プールされた接続ファクトリーの統計収集を有効にします。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra:write-attribute(name=statistics-enabled,value=true)
```

表A.9 プールされた接続ファクトリーの統計

統計	説明
ActiveCount	アクティブな数。
AvailableCount	利用可能な数。
AverageBlockingTime	プールの平均ブロック時間。
AverageCreationTime	物理接続の作成に費やされた平均時間。
AverageGetTime	物理接続の取得に費やされた平均時間。
AveragePoolTime	プールの物理接続によって費やされた平均時間。
AverageUsageTime	物理接続の使用に費やされた平均時間。
BlockingFailureCount	物理接続の取得に失敗した回数。

統計	説明
CreatedCount	作成された数。
DestroyedCount	破棄された数。
IdleCount	現在アイドル状態の物理接続の数。
InUseCount	現在使用中の物理接続の数。
MaxCreationTime	物理接続作成の最大時間。
MaxGetTime	物理接続取得の最大時間。
MaxPoolTime	プールの物理接続の最大時間。
MaxUsageTime	物理接続使用の最大時間。
MaxUsedCount	使用される接続の最大数。
MaxWaitCount	接続を待機するスレッドの最大数。
MaxWaitTime	接続の最大待機時間。
TimedOut	タイムアウトした回数。
TotalBlockingTime	ブロック合計時間。
TotalCreationTime	物理接続の作成に費やされた合計時間。
TotalGetTime	物理接続の取得に費やされた合計時間。
TotalPoolTime	プールの物理接続によって費やされた合計時間。
TotalUsageTime	物理接続の使用に費やされた合計時間。
WaitCount	物理接続の取得を待機する必要があったリクエストの数。
XACommitAverageTime	XAResource commit 呼び出しの平均時間。
XACommitCount	XAResource commit 呼び出しの数。
XACommitMaxTime	XAResource commit 呼び出しの最大時間。
XACommitTotalTime	すべての XAResource commit 呼び出しの合計時間。
XAEndAverageTime	XAResource end 呼び出しの平均時間。

統計	説明
XAEndCount	XAResource end 呼び出しの数。
XAEndMaxTime	XAResource end 呼び出しの最大時間。
XAEndTotalTime	すべての XAResource end 呼び出しの合計時間。
XAForgetAverageTime	XAResource forget 呼び出しの平均時間。
XAForgetCount	XAResource forget 呼び出しの数。
XAForgetMaxTime	XAResource forget 呼び出しの最大時間。
XAForgetTotalTime	すべての XAResource forget 呼び出しの合計時間。
XAPrepareAverageTime	XAResource prepare 呼び出しの平均時間。
XAPrepareCount	XAResource prepare 呼び出しの数。
XAPrepareMaxTime	XAResource prepare 呼び出しの最大時間。
XAPrepareTotalTime	すべての XAResource prepare 呼び出しの合計時間。
XARecoverAverageTime	XAResource recover 呼び出しの平均時間。
XARecoverCount	XAResource recover 呼び出しの数。
XARecoverMaxTime	XAResource recover 呼び出しの最大時間。
XARecoverTotalTime	すべての XAResource recover 呼び出しの合計時間。
XARollbackAverageTime	XAResource rollback 呼び出しの平均時間。
XARollbackCount	XAResource rollback 呼び出しの数。
XARollbackMaxTime	XAResource rollback 呼び出しの最大時間。
XARollbackTotalTime	すべての XAResource rollback 呼び出しの合計時間。
XAStartAverageTime	XAResource start 呼び出しの平均時間。
XAStartCount	XAResource start 呼び出しの数。
XAStartMaxTime	XAResource start 呼び出しの最大時間。
XAStartTotalTime	すべての XAResource start 呼び出しの合計時間。

Revised on 2021-05-17 08:55:47 CEST