JBoss Enterprise Application Platform 5

Messaging ユーザーガイド

JBoss Enterprise Application Platform 5 向け
エディッション 5.1.2

Last Updated: 2017-10-26

JBoss Enterprise Application Platform 5 Messaging ユーザーガイド

JBoss Enterprise Application Platform 5 向け
エディッション 5.1.2

Tim Fox

Jeff Mesnil

Andy Taylor

Clebert Suconic

Howard Gao

編集者

Laura Bailey

Jared Morgan

法律上の通知

概要

JBoss Enterprise Application Platform 5.0 およびそのパッチリリースで JBoss Messaging 1.4 を使用する方向けのガイド

目次

第1章 JBOSS MESSAGING 1.4 について

JBoss Messaging は JBoss によるエンタープライズメッセージングシステムです。レガシな JBoss Java Message Service (JMS) プロバイダーの JBossMQ を完全に書き直しました。

JBoss Messaging は JBoss Enterprise Application Platform 4.3 および 5 のデフォルトの JMS プロバイダーです。

JBoss Messaging は Red Hat のメッセージング戦略に不可欠です。JBoss Messaging は単一のノードとクラスター環境の両方のパフォーマンスを向上し、モジュラーアーキテクチャーの機能を備えているため、今後簡単にさらなる機能を追加していくことができます。

本ガイドでは、JBoss Enterprise Application Platform 用の JBoss Messaging のインストール、設定、構成方法について記載しています。

第2章 はじめに

JBoss Messaging はオープンソースな標準ベースのメッセージングプラットフォームを提供し、エンタープライズクラスのメッセージングを一般市場にもたらします。

JBoss Messaging はパフォーマンス性の高い堅牢なメッセージングのコアを実装し、サービス指向アーキテクチャー (SOA)、エンタープライズサービスバス (ESB)、その他の統合要件などあらゆるレベルの要求に対応できるように設計されています。

JBoss Messaging により、クラスター全体でアプリケーションの負荷を均等に分散できます。Single Point of Failure (SPOF) がないため、各ノードの CPU サイクルのバランスをとり、非常に拡張性のある高性能なクラスタリング実装を実現します。

JBoss Messaging には、標準規格に準拠した形式でメッセージが配信されるよう Java Messaging Service (JMS) フロントエンドが含まれ、将来的に他のメッセージングプロトコルにも対応できるようになっています。

2.1. JBOSS MESSAGING の特長

JBoss Messaging には次のような特長があります。

- 高スループットと短い待ち時間でパフォーマンス、信頼性、拡張性を重要視しています。
- SOA の構想を視野に入れた JBoss ESB の基盤となります (JBoss ESB は JBoss Messaging をそのデフォルト JMS プロバイダーとして使用します)。

JBoss Messaging のその他の特長は以下のとおりです。

- publish-subscribe メッセージングモデルと point-to-point メッセージングモデル
- 永続メッセージと非永続メッセージ
- メッセージが必要な場合に必ず一度だけ届くようなメッセージ配信を保証
- トランザクションの信頼できるインターフェース、ACID セマンティクスに対応
- JAAS を基にしたカスタマイズ可能なセキュリティフレームワーク
- JBoss Transactions (旧 Arjuna JTA) と完全統合し、トランザクションの完全復旧に対応
- 拡張型 JMX 管理インターフェース
- Oracle、DB2、Sybase、Microsoft SQL Server、PostgreSQL、MySQL などほとんどの主要なデータベースに対応
- HTTP トランスポート、HTTP トラフィックのみを許可するファイアウォールと併用
- 専用サーブレットを介したメッセージングを許可するサーブレットトランスポート
- SSL トランスポート
- 設定可能な DLQ (Dead Letter Queues) と Expiry Queues
- メッセージ統計、キューやサブスクリプションに配信されたメッセージの履歴ビューをスクロール表示

- ストレージへのメッセージの自動ページング、これによりシステムのメモリに収納するには大きすぎる大規模なキューの使用が可能
- 厳密なメッセージの順序、目的のキューに到着した順に配信される特定の 1 メッセージグループに属する複数のメッセージ

JBoss Messaging には次のようなクラスタリング機能も含まれます。

- **完全にクラスター化したキューおよびトピック**

  論理 キューおよびトピックがクラスター全体に分散されます。クラスターにあるいずれのノードからでもキューやトピックの送受信が可能です。

- **完全クラスター化した持続性のあるサブスクリプション**

  特定の持続性のあるサブスクリプションはクラスターのどのノードからでもアクセスが可能なため、クラスター全体に渡りそのサブスクリプションからの処理負荷を拡散できます。

- **完全クラスター化した一時的なキュー**

  送信メッセージに一時的なキューの **`replyTo`** が含まれている場合は、クラスター内のいずれのノードへも返信が可能です。

- **インテリジェントなメッセージの再分散**

  ノードによって異なるコンシューマー速度の利点を活用するためメッセージはクラスター内のノード間を自動的に移動します。これにより、特定のノード上でのメッセージのスタベーションや堆積を防止します。

- **メッセージ順の保護**

  プロデューサーによって生成されたメッセージ順がコンシューマーによってコンシュームされる順と必ず同じになるようにする場合に有効にします。これはメッセージの再分散がアクティブな場合にも機能します。

- **完全に透過的なフェールオーバー**

  サーバーに障害が発生した場合、例外なく新規ノード上でセッションが継続します。完全に設定が可能です。このフェールオーバー動作を実装したくない場合には、この機能を無効にして例外が送出されるようフォールバックして、新規ノードで手作業による接続を再作成できます。

- **高可用性でシームレスなフェールオーバー**

  ノードに障害が発生した場合、永続メッセージを失うことなく自動的に別のノードにフェールオーバーして、シームレスにセッションを継続することができます。永続メッセージは常に **必ず 1 回のみ** 配信されます。

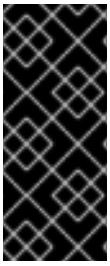
- **メッセージブリッジ**

  JBoss Messaging にはメッセージブリッジのコンポーネントが含まれており、2 つの JMS 1.1 デスティネーション間でメッセージの橋渡しができます。地理的に離れたクラスタ郡をつなぎ、グローバルに分散された大規模な論理キューやトピックを形成することができます。

2.2. JBOSS MQ との互換性

JBoss MQ は Enterprise Application Platform 4.2 同梱の JMS 実装でした。JBoss Messaging は JMS 1.1 および JMS 1.0.2b と互換性があるため、JBoss MQ に対して書かれた JMS コードは変更を加えなくても JBoss Messaging で稼働します。

JBoss Messaging には JBoss MQ に対してワイヤー形式の互換性がありません。そのため、JBoss MQ クライアントを JBoss Messaging のクライアント JAR でアップグレードする必要があります。



重要

JBoss Messaging のデプロイメント記述子は JBoss MQ のデプロイメント記述子に類似していますが、その両者は **同一ではありません** 。このため、JBoss Messaging で動作させるためには簡単な調節が必要になります。データベースのデータモデルは全く異なるため、JBoss MQ データスキーマで JBoss Messaging を使用したりその逆はしないでください。

2.3. JBOSS MESSAGING で使用されるシステムのプロパティ

2.3.1. support.bytesId

このシステムプロパティは **JBossMessage** オブジェクトを異質のメッセージオブジェクトから構成する場合にデフォルトの動作を制御します。サーバーの起動時にコマンドラインで **-D** オプションを使用して、このプロパティを設定します。

このプロパティを **true** に設定すると **JBossMessage** コンストラクタは異質のメッセージヘッダからネイティブの byte[] 相関 ID の抽出を試行します。**false** に設定すると、通常の文字列タイプ **JMSCorrelationID** を使用します。このプロパティを設定しない場合、または **true** もしくは **false** 以外の別のものに設定すると、プロパティは **true** にデフォルト設定されます。

2.3.2. retain.oldxabehaviour

このシステムプロパティは、接続の切断後に **prepare()** メソッドが呼び出された場合に、JMS XAResource により送出される例外タイプを制御します。サーバーの起動時にコマンドラインで **-D** オプションを使用して、このプロパティを設定します。

このプロパティを定義しないと、**XA_RBCOMMFAIL** エラーコードがある **XAException** が送出されます。これ以外は、**XA_RETRY** エラーコードがある **XAException** が送出されます。JBoss Messaging はデフォルトではこのプロパティを定義しないことに注意してください。

第3章 JBOSS MESSAGING のインストール

JBoss Enterprise Application Platform (EAP) には JBoss Messaging がデフォルトの JMS プロバイダーとして既にインストールされています。EAP バージョン 4.3 またはそれ以降を使用している場合は JBoss Messaging を手作業でインストールする必要はありません。

第4章 サンプル

JBoss Messaging にはダウンロードできるサンプルが多数あります。https://access.redhat.com からサンプルのアーカイブファイルをダウンロードしてください。

タスク：JBoss Messaging Examples Zip のダウンロード

このタスクに従い JBoss Messaging Example Zip バンドルをダウンロードします。サンプルは、当プラットフォームの文書バンドルの中に含まれています。

前提条件：

- access.redhat.com にて JBoss Enterprise Application Platform への正しいエンタイトルメントを有すること

1. Red Hat カスタマーポータル にログインします。
2. **Downloads (ダウンロード)** → **JBoss Enterprise Middleware (JBoss Enterprise ミドルウェア製品)** → **Downloads (ダウンロード)** を選択します。
3. ソフトウェアダウンロードのページで、**`Product` (製品)** のドロップダウンメニューから**Application Platform (アプリケーションプラットフォーム)** を選択します。

   バージョンのドロップダウンメニューは、最新リリースにデフォルト設定されています。
4. **Application Platform *[version]* Documentation (文書)** を探し、**`Download` (ダウンロード)** をクリックします。
5. 文書バンドルのダウンロードが開始します。

関連情報

- タスク：サンプルの解凍とデプロイ

タスク：サンプルの解凍とデプロイ

プラットフォームの文書バンドルから JBoss Messaging のサンプルを解凍し、サンプルを実行するために必要な基本設定要件をすべてクリアし、このタスクを完了します。

- タスク：JBoss Messaging Examples Zip のダウンロード
- デフォルト設定で JBoss Enterprise Application Server インスタンスを実行

1. お使いのオペレーティングシステムに合ったアーカイブユーティリティを使い、Zip アーカイブを開きます。
2. Zip アーカイブマネージャーにて、**jboss-eap-5.1** → **doc**に移動します。
3. サンプルのディレクトリを **`$JBOSS_HOME/docs/examples`**に解凍します。
4. ファイルブラウザーで **`$JBOSS_HOME/docs/examples/jboss-messaging-examples/destinations/`** を開きます。

5. **jbm-examples-destinations-service.xml** を **$JBOSS_HOME/server/default/deploy** にコピーし、サンプルで必要な対象の設定ディレクティブをデプロイします。

関連情報

- クラスター化されないサンプル
- クラスター化されたサンプル

クラスター化されないサンプル

重要

JBoss Enterprise Application Platform のクラスター化されていないプロファイルでクラスター化されていないサンプルを実行する必要があります。**All** プロファイルと **Production** プロファイルはサポートされていません。

各サンプルの **readme.html** では、設定の詳細、予想される出力、簡単なトラブルシューティングが記載されています。

queue

このサンプルは、JMS クライアントを使用したリモートキューへの簡単な送受信を示しています。

topic

このサンプルは JMS クライアントを使用したリモートトピックへの簡単な送受信を示しています。

mdb

このサンプルは JBoss Messaging での EJB2.1 MDB の使用を示しています。

ejb3mdb

このサンプルは JBoss Messaging での EJB3 MDB の使用を示しています。

stateless

このサンプルは JBoss Messaging と通信する EJB2.1 ステートレスセッション Bean を示しています。

mdb-failure

このサンプルは EJB2.1 MDB でのロールバックと再配信を示しています。

secure-socket

このサンプルは SSL 暗号化トランスポートを介して JBoss Messaging サーバーと通信する JMS クライアントを示しています。

http

このサンプルは HTTP を介してトラフィックをトンネルして JBoss Messaging サーバーと通信する JMS クライアントを示しています。

web-service

このサンプルは JBoss Messaging と通信する JBoss web-service を示しています。

stateless-clustered

このサンプルはクラスター化 EJB2.1 ステートレスセッション Bean と通信する JMS クライアントを示します (EJB2.1 ステートレスセッション Bean は JBoss Messaging と通信します)。 サンプルは、接続ファクトリーをルックアップするために HAJNDI を使用します。

bridge

このサンプルはメッセージブリッジの使用について示しています。JBoss AS 内でメッセージブリッジをデプロイして、メッセージをソースから目的のキューに移動させます。

servlet

このサンプルは JBoss Messaging を使ったサーブレットトランスポートの使用方法について示しています。サーブレットとサーブレットトランスポートを使用する ConnectionFactory をデプロイします。

ordering-group

このサンプルは、JBoss Messaging での厳密なメッセージ順序の使用について示しています。JBoss Messaging の順序グループ API を使用して、メッセージの優先度に関係なく厳密な順序でメッセージを配信します。

クラスター化されたサンプル

重要

クラスター化されたサンプルでは、ports-01 と ports-02 に設定されたポートを使用して 2 つの JBoss Application Server インスタンスが実行されている必要があります。

サンプルは、Enterprise Application Platform の**`All`** サーバープロファイルと **`Production`** サーバープロファイルでの使用向けにサポートされています。

各サンプルの **`readme.html`** では、設定の詳細、予想される出力、簡単なトラブルシューティングが記載されています。

distributed-topic

このサンプルは JBoss Messaging 分散型トピックと通信する JMS クライアントを示しています。この場合、2 つの JBoss AS インスタンスが実行されている必要があります。

distributed-queue

このサンプルは JBoss Messaging 分散型キューと通信する JMS クライアントを示しています。この場合、2 つの JBoss AS インスタンスが実行されている必要があります。

queue-failover

このサンプルは JMS コンシューマーの透過的なフェールオーバーを示しています。

第5章 設定

Java Message Service (JMS) API は、メッセージングクライアントがメッセージングサーバーと通信する方法を指定します。メッセージのデスティネーションや接続ファクトリなどのメッセージングサービスがどのように定義され、実装されるかは、JMS プロバイダーによって異なります。JBoss Messaging にはサービス設定用の独自のファイルがあります。

本章では JBoss Messaging で使用可能な各種サービスの設定方法について説明します。これらのサービスはクライアントのアプリケーションに JMS API レベルのサービスを提供するため連携して動作します。

JBoss Messaging の設定は数種の設定ファイルで分かれています。提供されるサービスの種類に応じて、設定情報は **messaging-service.xml**、**remoting-bisocket-service.xml**、**<your database type>-persistence-service.xml**、**connection-factories-service.xml**、**destinations-service.xml** 間で分かれています。これらのファイルはすべて **$JBOSS_HOME/server/$PROFILE/deploy/messaging** のディレクトリにあります。

AOP インターセプターのスタックは **aop-messaging-client.xml** (クライアント側の動作用) と **aop-messaging-server.xml** (サーバー側の動作用) で設定することができます。通常はこれらのファイルを変更する必要はありませんが、必要がないインターセプターがある場合はそれを削除してパフォーマンスを向上させることができます。セキュリティインターセプターを削除する場合は、セキュリティの影響を十分に考慮してください。

5.1. SERVERPEER の設定

ServerPeer は JBoss Messaging JMS ファサードの中核です。**$JBOSS_HOME/server/$PROFILE/deploy/messaging/messaging-service.xml** を編集してその動作を設定できます。

すべての JBoss Messaging サービスは **ServerPeer** を基にしています。

以下は Server Peer の設定例です。この例では Server Peer の属性のすべての値は指定されていません。

```
<!-- ServerPeer MBean configuration
     ============================== -->

<mbean code="org.jboss.jms.server.ServerPeer"
  name="jboss.messaging:service=ServerPeer"
  xmbean-dd="xmdesc/ServerPeer-xmbean.xml">

  <!--The unique id of the server peer - in a cluster each node
      MUST have a unique value - must be an integer-->

  <attribute name="ServerPeerID">
    ${jboss.messaging.ServerPeerID:0}
  </attribute>

  <!--The default JNDI context to use for queues when they are
      deployed without specifying one-->

  <attribute name="DefaultQueueJNDIContext">/queue</attribute>

  <!--The default JNDI context to use for topics when they are
      deployed without specifying one -->
```

```
<attribute name="DefaultTopicJNDIContext">/topic</attribute>

<attribute name="PostOffice">
  jboss.messaging:service=PostOffice
</attribute>

<!-- The default Dead Letter Queue (DLQ) to use for destinations.
     This can be overridden on a per destinatin basis -->

<attribute name="DefaultDLQ">
  jboss.messaging.destination:service=Queue,name=DLQ
</attribute>

<!--The default maximum number of times to attempt delivery of a
    message before sending to the DLQ (if configured).
    This can be overridden on a per destination basis-->

<attribute name="DefaultMaxDeliveryAttempts">10</attribute>

<!--The default Expiry Queue to use for destinations. This can
    be overridden on a per destinatin basis-->

<attribute name="DefaultExpiryQueue">
  jboss.messaging.destination:service=Queue,name=ExpiryQueue
</attribute>

<!--The default redelivery delay to impose. This can be overridden
    on a per destination basis -->

<attribute name="DefaultRedeliveryDelay">0</attribute>

<!--The periodicity of the message counter manager enquiring on
    queues for statistics-->

<attribute name="MessageCounterSamplePeriod">5000</attribute>

<!--The maximum amount of time for a client to wait for failover
    to start on the server side after it has detected failure-->

<attribute name="FailoverStartTimeout">60000</attribute>

<!--The maximum amount of time for a client to wait for failover
    to complete on the server side after it has detected failure-->

<attribute name="FailoverCompleteTimeout">300000</attribute>

<attribute name="StrictTck">false</attribute>

<!--The maximum number of days results to maintain in the message
    counter history-->

<attribute name="DefaultMessageCounterHistoryDayLimit">-1</attribute>

<!--The name of the connection factory to use for creating
    connections between nodes to pull messages-->
```

```
  <attribute name="ClusterPullConnectionFactoryName">
    jboss.messaging.connectionfactory:service=ClusterPullConnectionFactory
  </attribute>

  <!--When redistributing messages in the cluster. Do we need to
      preserve the order of messages received
      by a particular consumer from a particular producer? -->

  <attribute name="DefaultPreserveOrdering">false</attribute>

  <!-- Max. time to hold previously delivered messages back waiting for
       clients to reconnect after failover -->

  <attribute name="RecoverDeliveriesTimeout">300000</attribute>

  <!-- Set to true to enable message counters that can be viewed via JMX -
->

  <attribute name="EnableMessageCounters">false</attribute>

  <!-- The password used by the message sucker connections to create
connections.
       THIS SHOULD ALWAYS BE CHANGED AT INSTALL TIME TO SECURE SYSTEM
  <attribute name="SuckerPassword"></attribute>
  -->

  <!-- The name of the server aspects configuration resource
  <attribute name="ServerAopConfig">aop/jboss-aop-messaging-
server.xml</attribute>
  -->
  <!-- The name of the client aspects configuration resource
    <attribute name="ClientAopConfig">aop/jboss-aop-messaging-
client.xml</attribute>
  -->

  <depends optional-attribute-name="PersistenceManager">
    jboss.messaging:service=PersistenceManager
  </depends>

  <depends optional-attribute-name="JMSUserManager">
    jboss.messaging:service=JMSUserManager
  </depends>

  <depends>jboss.messaging:service=Connector,transport=bisocket</depends>
  <depends optional-attribute-name="SecurityStore"
    proxy-type="org.jboss.jms.server.SecurityStore">
      jboss.messaging:service=SecurityStore
  </depends>
</mbean>
```

5.2. SERVERPEER の属性

本セクションでは **ServerPeer** 管理 Bean 属性について説明します。

StopServerPeerOnDBFailure

`true` に設定している場合、データベースにエラーが発生すると(例：database connection is lost - allowing other nodes to continue work)サーバーは停止します。デフォルト値は `false` です。

ServerPeerID

`ServerPeer` の一意の識別子です。デプロイするノードがクラスタを形成するかメッセージブリッジによってリンクされるかに関わらず、それぞれ一意の識別子が必要です。識別子は有効な整数である必要があります。

DefaultQueueJNDIContext

キューをバインドする時に使用するデフォルトの JNDI コンテキストです。デフォルト値は `/queue` です。

DefaultTopicJNDIContext

トピックをバインドする時に使用するデフォルトの JNDI コンテキストです。デフォルト値は `/topic` です。

PostOffice

`ServerPeer` が使用するポストオフィスです。通常はこの属性を変更する必要はありません。ポストオフィスはメッセージをキューにルーティングしてキューとアドレス間のマッピングを維持します。

DefaultDLQ

サーバーがデスティネーションに使用するデフォルトの DLQ (Dead Letter Queue) です。DLQ はデスティネーション単位でオーバーライドできます。詳細は 「デスティネーションの設定」 を参照してください。DLQ はサーバーが特定の回数以上配信を失敗したメッセージのデスティネーションです。DLQ が指定されていない場合、配信が最大回数試行されるとメッセージは削除されます。配信試行の最大回数のグローバルな既定値は `DefaultMaxDeliveryAttempts` 属性で指定するか、デスティネーション単位で個別に指定することができます。

DefaultMaxDeliveryAttempts

設定されている場合、メッセージが DLQ に送信または削除されるまでに試行される配信最大回数のグローバルな既定値です。デフォルト値は `10` です。この値はデスティネーション単位で無効にできます。

DefaultExpiryQueue

`ServerPeer` がデスティネーションに使用するデフォルトの Expiry Queue です。デスティネーション管理 Bean 設定に関するセクションで記載されているとおり、この値はデスティネーション単位で無効にできます。Expiry Queue は期限切れとなったメッセージを保持します。メッセージの有効期限は `Message::getJMSExpiration()` の値によって決まります。Expiry Queue が指定されていないと、メッセージは期限切れになった時に削除されます。

DefaultRedeliveryDelay

この属性により再配信の試行を遅延させることができます。これにより過度に配信障害が発生するのを防止するのに役立ちます。デフォルト値は `0` です (遅延なし)。この値はデスティネーション単位で無効にできます。

MessageCounterSamplePeriod

この属性はサーバーがキューの統計に対しキューをクエリする間の期間を定義します。デフォルト値は **5000** ミリ秒です。

FailoverStartTimeout

問題が検知された時に、サーバー側でフェールオーバーが開始するのをクライアントが待機する最長期間です (ミリ秒単位)。デフォルト値は **60000** (1 分) です。

FailoverCompleteTimeout

フェールオーバーが開始された時点で、サーバー側でフェールオーバーが終了するのをクライアントが待機する最長期間です (ミリ秒単位)。デフォルト値は **300000** (5 分) です。

DefaultMessageCounterHistoryDayLimit

JBoss Messaging は、メッセージカウンター履歴を提供し、特定の日数の間で各キューに到着するメッセージの数を示します。この属性はメッセージカウンター履歴を保存する最大日数を表します。この値はデスティネーション単位で無効にできます。

ClusterPullConnectionFactoryName

キュー間でメッセージをプルまたは吸い込むために使用する接続ファクトリです。この属性を省略して、フェールオーバーを保持している間にメッセージの吸い込みを無効にすることができます。

DefaultPreserveOrdering

true に設定すると、JMS の順序がクラスター内で維持されます。詳細は 6章 *クラスタリングに関する注意点* を参照してください。デフォルト値は **false** です。

RecoverDeliveriesTimeout

フェールオーバーが発生すると、クライアントが再接続している間に配信済みのメッセージは保存されます。クライアントが再接続しないと (クライアントが停止など)、これらのメッセージは最終的には時間切れとなりキューに追加されます。この属性はタイムアウトになるまでの期間をミリ秒単位で設定します。デフォルト値は **300000** (5 分) です。

EnableMessageCounters

true に設定すると、サーバーの開始時にメッセージカウンターを有効にします。

SuckerPassword

JBoss Messaging はクラスター化したデスティネーション間でメッセージを再分散するためにノード間の接続を内部的に作成します。こうした接続は特別に確保しているユーザー名で作成されます。こうした接続を作成する場合に使用するパスワードをこの属性で定義します。

JBoss Messaging 1.4.1.GA 以降のバージョンについては、**SecurityMetadataStore** で **SuckerPassword** を定義する必要があります。

警告

SuckerPassword はインストール時点で変更する必要があります。変更しないとデフォルトのパスワードが使用され、デフォルトのパスワードを知っているユーザーは誰でもサーバーのどのデスティネーションにもアクセスできてしまいます。

SuckerConnectionRetryTimes

これは、障害の発生時に吸い込む側の接続を再試行できる最大回数です。デフォルト値は **-1** で、「無期限に試行」を表します。

SuckerConnectionRetryInterval

これは失敗した吸い込む側が接続を再度試みる間のミリ単位で表される間隔です。デフォルト値は **5000** です。

StrictTck

厳密な JMS テクノロジ互換性キット (TCK) セマンティクスを有効にするには、この属性を **true** に設定します。

Destinations

現在デプロイされているデスティネーション（キューとトピック）の一覧を返します。

MessageCounters

特定のキューのメッセージカウンターです。

MessageStatistics

各キューのそれぞれのメッセージカウンターの統計です。

SupportsFailover

この属性を **false** に設定すると、クラスター内でノードがクラッシュした場合、サーバー側のフェールオーバーは発生しません。

PersistenceManager

ServerPeer によって使用される永続マネージャーです (通常はこの属性を変更する必要はありません)。

JMSUserManager

ServerPeer によって使用される JMS ユーザーマネージャーです (通常はこの属性を変更する必要はありません)。

SecurityStore

プラグ可能な **SecurityStore** です。この属性を再定義する場合、**MessageSucker** ユーザー (**JBM.SUCKER**) をクラスタリングで必要になるすべての特別権限をもって認証する必要があります。

SupportsTxAge

トランザクションの作成時間をトランザクションの記録に保存するか指定します。**true** に設定すると、トランザクションの記録は保存されます。デフォルト値は **false** です。

5.3. SERVERPEER METHODS

次の方法は ServerPeer 管理 Bean に使用できます。

deployQueue

プログラムによってキューをデプロイするために使用します。キューは存在しているがデプロイされていない場合は、そのキューはデプロイされます。そうでない場合は、キューを作成してからデプロイします。

name のパラメーターは、デプロイするデスティネーションと一致します。

オプションの **jndiName** のパラメーターはデスティネーションがバインドされている場所の完全な JNDI の名前を示しています。これが指定されていない場合は、デスティネーションは **<DefaultQueueJNDIContext>/<name>** でバインドされます。

この操作にはオーバーロードした 2 つのバージョンがあります。ひとつはデフォルトのページングパラメーターを持つデスティネーションをデプロイします。もうひとつは、指定したページングパラメーターを持つデスティネーションをデプロイします。ページングパラメーターの詳細は 「デスティネーションの設定」 を参照してください。

undeployQueue

プログラムによってキューをデプロイ解除するために使用します。キューは永続ストレージから削除されません。キューが正しくデプロイ解除されるとこの動作は **true** を返します。これ以外は **false** を返します。

destroyQueue

プログラムによってキューを破棄するために使用します。キューはデプロイ解除され、その全データはデータベースから削除され、破棄されます。

警告

この方法を使用する場合、キューのデータはすべて削除されるため注意が必要です。

この動作はキューが正しく破棄されると **true** を返します。これ以外は **false** を返します。

deployTopic

プログラムによってトピックをデプロイするために使用します。この操作にはオーバーロードした 2 つのバージョンがあります。ひとつはデフォルトのページングパラメーターを持つ既存のトピックをデプロイします。もうひとつは、指定したページングパラメーターを持つトピックを作成、デプロイします。詳細は 「デスティネーションの設定」 を参照してください。

name のパラメーターはデプロイするデスティネーションの名前を示しています。

jndiName は、デスティネーションがバインドされている場所の完全な JNDI の名前を示しています。これが指定されていない場合は、デスティネーションは **<DefaultTopicJNDIContext>/<name>** でバインドされます。

undeployTopic

プログラムによってトピックをデプロイ解除するために使用します。トピックはデプロイ解除されますが、永続ストレージからは削除されません。この動作はトピックが正しくデプロイ解除されると **true** を返します。これ以外は **false** を返します。

destroyTopic

プログラムによってトピックを破棄するために使用します。トピックはデプロイ解除され、全データはデータベースから削除され、破棄されます。この動作はトピックが正しく破棄されると **true** を返します。これ以外は **false** を返します。

警告

この方法を使用する場合、トピックのデータはすべて削除されるため注意が必要です。

listMessageCountersHTML

簡易表示の HTML 形式でメッセージカウンターを返します。

resetAllMesageCounters

全メッセージカウンターをゼロにリセットします。

enableMessageCounters

全デスティネーションのメッセージカウンターをすべて有効にします。メッセージカウンターはデフォルトでは無効になっています。

disableMessageCounters

全デスティネーションのメッセージカウンターをすべて無効にします。メッセージカウンターはデフォルトでは無効になっています。

retrievePreparedTransactions

現在、ノードで準備済みの状態にある全トランザクションの XID の一覧を取り込みます。

showPreparedTransactions

現在、ノードで準備済みの状態にある全トランザクションの XID の一覧を簡単に表示する HTML 形式で示します。

listAllPreparedTransactions

準備済みの全トランザクションの詳細を表示します。

listPreparedTransactions

トランザクションの長さが指定した時間と同じかそれよりも長い準備済みの全トランザクションの詳細を表示します。

showMessageDetails

メッセージの詳細を表示します。メッセージ ID を使用して、表示するメッセージを指定します。

commitPreparedTransaction

準備済みのトランザクションを手作業でコミットします。トランザクション ID を使用して、コミットするトランザクションを指定します。

rollbackPreparedTransaction

準備済みのトランザクションを手作業でロールバックします。トランザクション ID を使用して、ロールバックするトランザクションを指定します。

5.4. データベースの変更

JMS は、**`jboss-as/server/$PROFILE/deploy/messaging/<DATABASE_TYPE>-persistence-service.xml`** にて永続ストレージとして定義されたデータベースを利用します。デフォルトの永続ストレージタイプは **`hsqldb-persistence-service.xml`** にて定義された Hypersonic (HSQLDB) です。この設定はクラスター環境向けに定義されています (**`<attribute name="Clustered">true</attribute>`**)。

警告

Hypersonic 設定がデフォルトの永続設定として利用されていますが、**Hypersonic は実稼動環境ではサポートされていません。**以下の既知の問題が理由です。

- トランザクション分離がない
- スレッドおよびソケットリーク (**`connection.close()`** はリソースを整理しません)
- 永続性の質 (通常ログはエラー後破損し、自動回復しません)
- データベースの破損
- 負荷時の安定性 (データベースのプロセスは、過度のデータを処理すると停止します)
- クラスター環境で実行不可能

Hypersonic データベースは、開発やテストを目的として設計されており、実稼動環境で利用するべきではありません。推奨データベースの詳細情報は、『スタートガイド』の『その他のデータベースの利用』を参照してください。

Persistence Manager、Post Office、JMS User Manager はすべて永続ストレージと通信します。

Persistence Manager はメッセージ関連の永続化を処理します。Post Office はバインディング関連の永続化を処理します。JMS User Manager はユーザー関連の永続化を処理します。こうした管理 Bean のすべての設定は **<your database type>-persistence-service.xml** ファイルで処理されます。

MySQL、Oracle、PostgreSQL、Microsoft SQL Server、Sybase のデータベースのサンプル設定ファイルは、リリースバンドルの **jboss-as/docs/examples/jms** ディレクトリにあります。

これらデータベースのひとつにサポートを有効にするには、デフォルトの **jboss-as/server/$PROFILE/deploy/messaging/hsqldb-persistence-service.xml** 設定ファイルをご使用のデータベースの種類向けの設定ファイルと置き換えてください。新規永続設定を適用するために、サーバーを再起動します。

重要

未対応の Hypersonic 設定以外に、提供のサンプル設定はデフォルトではクラスター化されていない環境向けに設定されています。お使いのデータベース設定向けにクラスター環境を有効にするには、 ***<your database type>-persistence-service.xml*** の Clustered 属性を**true** に設定します (デフォルト設定は<attribute name="Clustered">false</attribute>)。

デフォルトでは、データストアに依存するメッセージングサービスは、データソースに関して **java:/DefaultDS** を参照します。別の JNDI 名でデータソースをデプロイするためには、永続設定ファイルのすべての **DataSource** 属性を更新する必要があります。ディストリビューションにはデータソース設定のサンプルが含まれています。

5.5. ポストオフィスの設定

ポストオフィスは、メッセージをそのデスティネーションまたは複数のデスティネーションにルーティングします。メッセージの送信が可能なアドレスと最終的なキューの間のマッピングを管理します。例えば、JMS キューを表すアドレスでメッセージを送信すると、ポストオフィスはこのメッセージをその JMS キューにルーティングします。JMS トピックを表すアドレスでメッセージを送信すると、ポストオフィスはこのメッセージを一連のキューにルーティングします。このセットは各 JMS サブスクリプション毎に 1 つです。

また、ポストオフィスはアドレスマッピングの永続化処理も行います。

JBoss Messaging ポストオフィスはクラスター対応です。クラスターでは、完全分散の JMS キューとトピックを提供するためにポストオフィスは自動的にノード間でメッセージのルーティング (プッシュ) とプルを行います。

<database type>-persistence-service.xml ファイルでポストオフィスを設定します。例えば以下のとおりです。

```
<mbean code="org.jboss.messaging.core.jmx.MessagingPostOfficeService"
  name="jboss.messaging:service=PostOffice"
  xmbean-dd="xmdesc/MessagingPostOffice-xmbean.xml">

  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>

  <depends>
    jboss.jca:service=DataSourceBinding,name=DefaultDS
  </depends>
```

```
<depends optional-attribute-name="TransactionManager">
  jboss:service=TransactionManager
</depends>

<!-- The name of the post office -->

<attribute name="PostOfficeName">JMS post office</attribute>

<!-- The datasource used by the post office to access it's
     binding information -->

<attribute name="DataSource">java:/DefaultDS</attribute>

<!-- If true will attempt to create tables and indexes on
     every start-up -->

<attribute name="CreateTablesOnStartup">true</attribute>

<!-- If true then we will automatically detect and reject
     duplicate messages sent during failover -->

<attribute name="DetectDuplicates">true</attribute>

<!-- The size of the id cache to use when detecting duplicate
     messages -->

<attribute name="IDCacheSize">500</attribute>

<attribute name="SqlProperties">
  CREATE_POSTOFFICE_TABLE=CREATE TABLE JBM_POSTOFFICE
    (POSTOFFICE_NAME VARCHAR(255),
    NODE_ID INTEGER, QUEUE_NAME VARCHAR(255), COND VARCHAR(1023),
    SELECTOR VARCHAR(1023), CHANNEL_ID BIGINT, CLUSTERED CHAR(1),
    ALL_NODES CHAR(1),
    PRIMARY KEY(POSTOFFICE_NAME, NODE_ID, QUEUE_NAME)) ENGINE = INNODB

  INSERT_BINDING=INSERT INTO JBM_POSTOFFICE
    (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR,
     CHANNEL_ID, CLUSTERED, ALL_NODES)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?)

  DELETE_BINDING=DELETE FROM JBM_POSTOFFICE WHERE
    POSTOFFICE_NAME=? AND NODE_ID=? AND QUEUE_NAME=?

  LOAD_BINDINGS=SELECT QUEUE_NAME, COND, SELECTOR,
    CHANNEL_ID, CLUSTERED, ALL_NODES FROM
 JBM_POSTOFFICE WHERE POSTOFFICE_NAME=? AND NODE_ID=?
</attribute>

<!-- This post office is clustered. If you don't want a clustered post
     office then set to false -->

<attribute name="Clustered">true</attribute>

<!-- All the remaining properties only have to be specified if the post
     office is clustered. You can safely comment them out if your post
```

```
      office is non clustered -->

<!-- The JGroups group name that the post office will use -->

<attribute name="GroupName">
  ${jboss.messaging.groupname:MessagingPostOffice}
</attribute>

<!-- Max time to wait for state to arrive when the post office
     joins the cluster -->

<attribute name="StateTimeout">5000</attribute>

<!-- Max time to wait for a synchronous call to node members using
     the MessageDispatcher -->

<attribute name="CastTimeout">50000</attribute>

<!-- Set this to true if you want failover of connections to occur
     when a node is shut down -->

<attribute name="FailoverOnNodeLeave">false</attribute>

<!-- JGroups stack configuration for the data channel - used for sending
     data across the cluster -->

<!-- By default we use the TCP stack for data -->
<attribute name="DataChannelConfig">
  <config>
    <TCP start_port="7900"
      loopback="true"
      recv_buf_size="20000000"
      send_buf_size="640000"
      discard_incompatible_packets="true"
      max_bundle_size="64000"
      max_bundle_timeout="30"
      use_incoming_packet_handler="true"
      use_outgoing_packet_handler="false"
      down_thread="false" up_thread="false"
      enable_bundling="false"
      use_send_queues="false"
      sock_conn_timeout="300"
      skip_suspected_members="true"/>
    <MPING timeout="4000"
      bind_to_all_interfaces="true"
      mcast_addr="${jboss.messaging.datachanneludpaddress:228.6.6.6}"
      mcast_port="${jboss.messaging.datachanneludpport:45567}"
      ip_ttl="8"
      num_initial_members="2"
      num_ping_requests="1"/>
    <MERGE2 max_interval="100000"
      down_thread="false" up_thread="false" min_interval="20000"/>
    <FD_SOCK down_thread="false" up_thread="false"/>
    <VERIFY_SUSPECT timeout="1500" down_thread="false"
     up_thread="false"/>
    <pbcast.NAKACK max_xmit_size="60000"
```

```
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
      <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
      <pbcast.GMS print_local_addr="true" join_timeout="3000"
        down_thread="false" up_thread="false"
        join_retry_timeout="2000" shun="false"
        view_bundling="true"/>
    </config>
  </attribute>

  <!-- JGroups stack configuration to use for
       the control channel - used for control messages -->

  <!-- We use udp stack for the control channel -->
  <attribute name="ControlChannelConfig">
    <config>
      <UDP
        mcast_addr="${jboss.messaging.controlchanneludpaddress:228.7.7.7}"
        mcast_port="${jboss.messaging.controlchanneludpport:45568}"
        tos="8"
        ucast_recv_buf_size="20000000"
        ucast_send_buf_size="640000"
        mcast_recv_buf_size="25000000"
        mcast_send_buf_size="640000"
        loopback="false"
        discard_incompatible_packets="true"
        max_bundle_size="64000"
        max_bundle_timeout="30"
        use_incoming_packet_handler="true"
        use_outgoing_packet_handler="false"
        ip_ttl="2"
        down_thread="false" up_thread="false"
        enable_bundling="false"/>
      <PING timeout="2000"
        down_thread="false" up_thread="false" num_initial_members="3"/>
      <MERGE2 max_interval="100000"
        down_thread="false" up_thread="false" min_interval="20000"/>
      <FD_SOCK down_thread="false" up_thread="false"/>
      <FD timeout="10000" max_tries="5" down_thread="false"
        up_thread="false" shun="true"/>
      <VERIFY_SUSPECT timeout="1500" down_thread="false"
       up_thread="false"/>
      <pbcast.NAKACK max_xmit_size="60000"
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
      <UNICAST timeout="300,600,1200,2400,3600"
        down_thread="false" up_thread="false"/>
      <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
```

```
      <pbcast.GMS print_local_addr="true" join_timeout="3000"
        use_flush="true" flush_timeout="3000"
        down_thread="false" up_thread="false"
        join_retry_timeout="2000" shun="false"
        view_bundling="true"/>
      <FRAG2 frag_size="60000" down_thread="false" up_thread="false"/>
      <pbcast.STATE_TRANSFER down_thread="false" up_thread="false"
        use_flush="true" flush_timeout="3000"/>
      <pbcast.FLUSH down_thread="false" up_thread="false" timeout="20000"
        auto_flush_conf="false"/>
    </config>
  </attribute>

</mbean>
```

5.5.1. MessagingPostOffice 属性

MessagingPostOffice サービス属性について以下の一覧で説明しています。

DataSource

ポストオフィスがマッピングデータの永続化に使用すべきデータソース

SQLProperties

特定のデーターベースの DDL と DML を指定します。特定の DDL や DML ステートメントがオーバーライドされていない場合は、デフォルトの Hypersonic 設定がそのステートメントに使用されます。

CreateTablesOnStartup

ポストオフィスが起動時にテーブル（およびインデックス）の作成を試行するようにするには、これを **true** に設定します。テーブルまたはインデックスが既に存在している場合は、**SQLException** は JDBC ドライバーによって送出されますが、永続マネージャーはこれを無視するためその動作は継続します。

デフォルトでは、**CreateTablesOnStartup** 属性の値は **true** に設定されています。

DetectDuplicates

ポストオフィスが、サーバー障害の後、別ノードから送信の再試行が行われた際に送信されたなどで、重複したメッセージを検出するようにしたい場合は、これを **true** に設定します。

デフォルトでは、**DetectDuplicates** 属性の値は **true** に設定されています。

IDCacheSize

重複検出が有効になっている場合 (**DetectDuplicates** 参照)、サーバーは最後から **n** 個の送信メッセージ ID を記憶し、障害が起こった後にメッセージが重複して送信されないようにします。**n** の値は、この属性で決定します。

デフォルトでは、**IDCacheSize** 属性は **500** に設定されています。

PostOfficeName

ポストオフィスの名前

NodeIDView

これは、クラスター内のすべてのノードのノード ID を 1 セットとして返します。

GroupName

クラスター内で同じグループ名を持つ全ポストオフィスで 1 つのクラスターを作成します。クラスターを形成したいと考えるクラスターにあるノードすべてと、グループ名が一致するようにしてください。

Clustered

True に設定している場合、ポストオフィスはクラスター 1 つに参加し、分散したキュートトピックを形成します。false に設定すると、クラスターには参加せず、クラスター関連の属性はすべて無視されます。

StateTimeout

ノードが既存のクラスターに参加する際にグループのステータスが到着になるまで待機する最大時間

デフォルト値は、**`5000`** ミリ秒です。

CastTimeout

返信キャストメッセージの同期まで待機する最大時間

デフォルト値は、**`5000`** ミリ秒です。

FailoverOnNodeLeave

ノードがクリーンにシャットダウンした場合、ノードに保存されているメッセージがどのように再配信されるかを指定します。デフォルト値は **`false`** になっています。**`true`** の場合は、サーバーノードがクリーンにシャットダウンされる (ターミナルで**`Ctrl`**+**`C`** を押す) と、そのノードに保存されている全メッセージが同じクラスターにある別のノードに移動されます。

重要

クリーンシャットダウンされたノードに元々接続されていたクライアントは自動的にクラスター内でフェールオーバーしたノードに再接続されません。クライアントは、メッセージのフェールオーバーが発生すると例外を返します。

MaxConcurrentReplications

返信が戻ってこないようにするまでに、同時複製リクエストを最大で作成できる数。これにより、JGroups が膨大にならないようにします。これは変更しないほうが良いでしょう。

デフォルト値は **`50`** となっています。

ControlChannelConfig

JBoss Messaging は全グループの管理に JGroups を使用します。これにはコントロールチャンネルの JGroups スタック設定が含まれています。

コントロールチャンネルを使い、クラスター内の別のノードからリクエストを送信／応答を受信します。

JGroups 設定の詳細は、標準の JGroups 設定であるため、ここでは説明しません。JGroups の詳細情報については、JGroups リリース文書か、オンライン (http://www.jgroups.org あるいは http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups) で入手していただけます。

DataChannelConfig

JBoss Messaging は全グループの管理に JGroups を使用します。これにはデータチャンネルの JGroups スタック設定が含まれています。

データチャンネルを使い、クラスター内の別のノードからメッセージを送信／応答を受信し、セッションデータをレプリケートします。

JGroups 設定の詳細は、標準の JGroups 設定であるため、ここでは説明しません。JGroups の詳細情報については、JGroups リリース文書か、オンライン (http://www.jgroups.org あるいは http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroups) で入手していただけます。

Database Connection Retry パラメーターは、接続失敗が検出された場合接続を再確立するか、何度再接続を試みるか、接続再試行の間隔はどの程度かを制御します。

RetryOnConnectionFailure

MBean がデータベースへの再接続を試行すべきか指定します。デフォルト値は **`false`** です。

MaxRetry

DataSource 接続失敗の上限を指定します。デフォルトは **`25`** です。パラメーターを **`-1`** に設定し、「Retry forever」モードを有効にします。このパラメーターは ***`RetryOnConnectionFailure`*** が **`true`** に設定されている場合有効です。

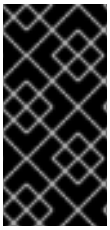
重要

クラスター化されたデスティネーションからメッセージをコンシュームするクライアントは、クローズされると応答しなくなります。これは、ノードの ***`MaxRetry`*** 値が **`-1`** に設定されている場合に発生し、データベースへの接続が失われます。この問題を回避するには、ノードの***`MaxRetry`*** パラメーターを**`-1`**よりも大きな値に設定してください。 **`[database]-persistence-service.xml`**ファイルにある **`PersistenceManager`**、**`PostOffice`**、**`JMSUserManager`** MBeanの属性値を設定してください。

RetryInterval

連続する再試行の間隔を指定します。デフォルトは、1000 (ミリ秒) となっています。このパラメーターは、***`RetryOnConnectionFailure`*** が **`true`** に設定されている場合に有効です。

安定性を保ちつつもノードが数秒応答しなくなった場合 (例：マイナーなネットワーク障害)、以前のフェールオーバーの動作によりノードがクラスターを退出したことを報告します。MessagePostOffice bean の以下のパラメーターを指定することでこのシナリオを回避できます。以下のパラメーターは、MessagePostOffice がクラスターの timestamp テーブルにアクセスできるようにします。この timestamp テーブルを使いより正確にノードの状態を判断します。



重要

timestamp テーブルのパラメーターは、MessagingClusterHealthMBean MBean と連携して使う必要があります。ServerPeer MBean にこの MBean をデプロイします。対応のパラメーターと操作に関する包括的な説明については、「ServerPeer の設定」 を参照してください。

KeepOldFailoverMode

timestamp テーブルのフェールオーバーモードを使うかどうかを指定します。デフォルトは **true** (新規のフェールオーバー動作を無効化) となっています。

NodeStateRefreshInterval

クラスターがノードを無効とするまでに、ノードによるタイムスタンプ更新の最大待機時間 (ミリ秒) を指定します。デフォルトは 30000 です (30 秒)。

5.6. MESSAGINGCLUSTERHEALTHMBEAN の設定

MessagingPostOffice は ***KeepOldFailoverMode*** と ***NodeStateRefreshInterval*** パラメーターを使い、ノードがクラスターの一部としてとどまるためにタイムスタンプを更新する時間を制御します。これらのパラメーターは、ノードがデータベースとの接続切断にどのように対応するのかを制御するわけではありません。

ノードとデータベースとの接続が切断された場合、**MessagePostOffice** 向けのタイムスタンプの情報を更新できません。ノードの状態が問題なくても、クラスターは実際のノードの状態を把握できません。結果、クラスターはノードはダウンし、重複メッセージの送信が発生する可能性があります。

MessagingClusterHealthMBean MBean はノードの状態を監視し、データベースへの接続が切れた場合ノードを開始／停止します。ノードがクラスターからシュンされた結果、データベースの接続がなくなった場合、MBean はノードを即座にシャットダウンします。MBean は、ノードがダウンしている間、JGroups のステータスとデータベースのステータスを監視し、JGroups を検出し、データベース接続が通常機能するようにリストアされた時点でノードを再起動します。

この機能を有効にするには、<depends> ディレクティブの ***optional-attribute-name*** 属性として、ServerPeer MBean の **MessagingClusterHealthMBean** MBean を宣言します。

```
<!-- ServerPeer MBean configuration ============================== -->
<mbean code="org.jboss.jms.server.ServerPeer"
name="jboss.messaging:service=ServerPeer"
       xmbean-dd="xmdesc/ServerPeer-xmbean.xml">

<!-- The unique id of the server peer - in a cluster each node MUST have a
unique value - must be an integer -->

   <attribute name="ServerPeerID">0</attribute>

<!-- The default JNDI context to use for queues when they are deployed
without specifying one -->

   <attribute name="DefaultQueueJNDIContext">/queue</attribute>

<!-- The default JNDI context to use for topics when they are deployed
without specifying one -->
```

```
    <attribute name="DefaultTopicJNDIContext">/topic</attribute>

<!-- XML CONFIG REMOVED FOR READABILITY -->


    <depends optional-attribute-name="PersistenceManager">
       jboss.messaging:service=PersistenceManager
    </depends>

<!-- XML CONFIG REMOVED FOR READABILITY -->

    <depends optional-attributename="MessagingClusterHealthMBean">
       jboss.messaging:service=MessagingClusterHealthMBean
    </depends>

</mbean>
```

5.7. PERSISTENCE MANAGER の設定

JBoss Messaging には JDBC Persistence Manager が同梱されており、JDBC を介してアクセスされるリレーショナルデータベースのメッセージデータの永続化処理を行います。Persistence Manager は Messaging サーバーにプラグが可能です。これにより、非リレーショナルのストア、ファイルストアにあるメッセージデータを永続化する追加の実装が可能になります。

永続サービス設定の詳細は **<database type>-persistence-service.xml** にグループ化されています。JBoss Messaging にはデフォルトで **hsqldb-persistence-service.xml** ファイルが同梱されています。これにより、デフォルトで JBoss Enterprise Application Server インスタンスに付属している Hypersonic データベースインスタンスを使用するように Messaging サーバーを設定します。

警告

Hypersonic は実稼働環境の使用ではサポートされていません。

JBoss Messaging には MySQL、Oracle、PostgreSQL、Sybase、Microsoft SQL Server、DB2 の Persistence Manager 設定も同梱しています。サンプルの設定ファイル (**mysql-persistence-service.xml** や **ndb-persistence-service.xml**) は、リリースバンドルの **jboss-as/docs/examples/jms** ディレクトリにあります。

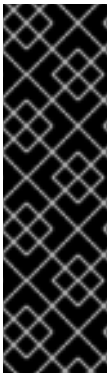
JDBC Persistence Manager は標準 SQL をその Data Manipulation Language (DML) として使用します。そのため、Persistence Manager 設定を別のデータベースの種類に書き込むことは、設定の Data Definition Language (DDL) の変更の問題です。これは通常、データベースごとに異なります。

JBoss Messaging には Null Persistence Manager 設定オプションも同梱されており、永続化が必要ない場合に使用できます。

以下のコードはデフォルトの Hypersonic 永続マネージャー設定です。

```
<mbean code="org.jboss.messaging.core.jmx.JDBCPersistenceManagerService"
  name="jboss.messaging:service=PersistenceManager"
  xmbean-dd="xmdesc/JDBCPersistenceManager-xmbean.xml">
```

```
  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The datasource to use for the persistence manager -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on every start-up
-->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then will use JDBC batch updates -->

  <attribute name="UsingBatchUpdates">false</attribute>

  <!-- The maximum number of parameters to include in a prepared statement
-->

  <attribute name="MaxParams">500</attribute>
</mbean>
```



重要

Sybase データベーステキストとイメージデータのタイプの最大サイズはデフォルトでは **2 kilobytes** に設定されています。この制限を越えるすべてのメッセージは、通知や警告なく切り捨てられます。切り捨てられないようにするには **@@TEXTSIZE** データーベースのパラメータをそれより高く設定します。

@@TEXTSIZE 値をデフォルト値より低く設定すると、Microsoft SQL Server でもこの切り捨て動作が起こることがあります。詳しくは http://jira.jboss.com/jira/browse/SOA-554 と、『管理設定ガイド』の『Sybaseに関する特記』 の項を参照してください。

重要

データがデーターベースから削除される場合、Microsoft SQL Server はハードドライブの領域の割り振り解除を自動的に行うことはありません。多くの記録を一時的に格納するサービスのデータストアとしてハードドライブのデータベース領域を使用する場合 (メッセージングサービスなど)、そのディスク領域は実際に格納されているデータ量より非常にかつ素早く大きくなります。

未使用領域が必ず再利用されるようデーターベース管理者は、データーベースのメンテナンス計画を実施する必要があります。未使用領域を使用するガイダンスに関しては、DBCC コマンドの **ShrinkDatabase** と **UpdateUsage** に関するご使用の Microsoft SQL Server のドキュメントを参照してください。この問題の詳細については https://issues.jboss.org/browse/SOA-629 を参照してください。

5.7.1. JDBCPersistenceManager MBean 属性

JDBCPersistenceManager 属性が以下の一覧で説明されています。

CreateTablesOnStartup

永続マネージャーが起動時にテーブル（およびインデックス）の作成を試行するようにするには、これを **true** に設定します。テーブルまたはインデックスが既に存在している場合は、**SQLException** は JDBC ドライバーによって送出されますが、永続マネージャーはこれを無視するためその動作は継続します。

デフォルトでは、**CreateTablesOnStartup** 属性の値は **true** に設定されています。

UsingBatchUpdates

データベースが JDBC バッチ更新に対応している場合は、これを **true** に設定します。JDBC 永続マネージャーは、バッチ内のデータベースアップデートを複数まとめ、パフォーマンスを向上します。

デフォルトでは、**UsingBatchUpdates** 属性の値は **false** に設定されています。

UsingBinaryStream

getBytes(), setBytes() ではなく JDBC バイナリストリームを使いメッセージを保存、読み込みたい場合は、これを **true** に設定します。データベースによっては、getBytes()/setBytes() を使った ゲット／セットにはバイトの上限があります。

デフォルトでは、**UsingBinaryStream** 属性の値は **true** に設定されています。

UsingTrailingByte

末尾にゼロがある場合、Sybase のバージョンによっては、blob を切り捨てると言われています。これを防ぐには、この属性を **true** に設定し、末尾に 0 がこないバイト数を追加し、永続前と後で各 blob を削除し、データベースから blob が切り捨てられないようにします。現在、Sybase のみに必要であるとわかっています。

デフォルトでは、**UsingTrailingByte** 属性の値は **false** に設定されています。

SupportsBlobOnSelect

Oracle (および、他のデータベース) では、INSERT INTO ... SELECT FROM 文字列を使い、BLOB を挿入できないため、２段階のメッセージ条件挿入が必要となります。この値が false の場合、このような２段階の挿入を使います。

デフォルトでは、**SupportsBlobOnSelect** 属性の値は **true** に設定されています。

SQLProperties

特定のデーターベースの DDL と DML を指定します。特定の DDL や DML ステートメントがオーバーライドされていない場合は、デフォルトの Hypersonic 設定がそのステートメントに使用されます。

MaxParams

メッセージをロードすると、永続マネージャーは多数のパラメーターで準備している文字列を生成します。この値により、永続マネージャーは、用意した文字列毎に許容されるパラメーターの絶対最大数はどれくらいであるか判断します。

デフォルトでは、**MaxParams** 属性値は **100** に設定されています。

UseNDBFailoverStrategy

クラスター化されたデータベース環境で実行している場合、データベーストランザクションのコ

ミット中にMySQL などのデータベースで問題が発生することがあります。これは、データベースのノードがコミット中に切断された場合 (つまり、トランザクションの最終段階が未知の状態の場合) 発生します。この属性を true に設定すると、上記が発生し、SQL ステートメントは再度実行されます。ただし、さらにエラーが発生すると、以前のトランザクションのコミットが正常に終了したためと推測され、このエラーは無視されます。

デフォルトでは、**`UseNDBFailoverStrategy`** 属性は **`false`** に設定されています。

Database Connection Retry パラメーターは、接続失敗が検出された場合接続を再確立するか、何度再接続を試みるか、接続再試行の間隔はどの程度かを制御します。

RetryOnConnectionFailure

MBean がデータベースへの再接続を試行すべきか指定します。デフォルト値は **`false`** です。

MaxRetry

DataSource 接続失敗の上限を指定します。デフォルトは **`25`** です。パラメーターを **`-1`** に設定し、「Retry forever」モードを有効にします。このパラメーターは ***`RetryOnConnectionFailure`*** が **`true`** に設定されている場合有効です。

重要

クラスター化されたデスティネーションからメッセージをコンシュームするクライアントは、クローズされると応答しなくなります。これは、ノードの ***`MaxRetry`*** 値が **`-1`** に設定されている場合に発生し、データベースへの接続が失われます。この問題を回避するには、ノードの***`MaxRetry`*** パラメーターを**`-1`**よりも大きな値に設定してください。 **`[database]-persistence-service.xml`**ファイルにある **`PersistenceManager`**、**`PostOffice`**、**`JMSUserManager`** MBeanの属性値を設定してください。

RetryInterval

連続する再試行の間隔を指定します。デフォルトは、1000 (ミリ秒) となっています。このパラメーターは、***`RetryOnConnectionFailure`*** が **`true`** に設定されている場合に有効です。

CreateTablesOnStartup

Persistence Manager を開始する時に、テーブルとインデックスの作成を試行するか設定します。**`true`** (デフォルト) に設定すると、永続マネージャーは起動時にテーブル（およびインデックス）を作成するよう試行します。テーブルまたはインデックスが既に存在している場合は、**`SQLException`** は JDBC ドライバーによって送出されますが、永続マネージャーはこれを無視するためその動作は妨害されずに継続します。

UsingBatchUpdates

パフォーマンスを向上させるために複数のデーターベース更新をバッチでグループ化するか指定します。ご使用のデータベースが JDBC のバッチ更新をサポートする場合は、この値を **`true`** に設定します。デフォルト値は **`false`** です。

UsingBinaryStream

メッセージが **`getBytes()`** や **`setBytes()`** を介してではなく、JDBC バイナリストリームを使用して保存や読み取りを行うか指定します。ご使用のデーターベースが **`getBytes()`** や **`setBytes()`** を使用する必要がある場合は、この値を **`false`** に設定します。デフォルト値は **`true`** です。

UsingTrailingByte

末尾のゼロが含まれる Sybase database BLOB の処理方法を指定します。**true** に設定すると、末尾がゼロでないバイトは、永続化の前にそれぞれの BLOB に追加され、永続化に続く BLOB から削除されます。これによりデーターベースによる切り捨てを防止します。デフォルト値は **false** です。

注記

Sybase の特定のバージョンは末尾にゼロがある BLOB を切り捨てます。この属性は Sybase データーベースを実行している場合のみ必要です。

SupportsBlobOnSelect

BLOB をどのように特定のデーターベースの種類に挿入するか設定します。**false** に設定すると、2 ステージ挿入が使用されます。デフォルト値は **true** です。

注記

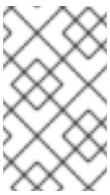
特に Oracle などの特定のデーターベースでは、**INSERT INTO ... SELECT FROM** ステートメントを介した BLOB 挿入を許可しないため、2 ステージの条件メッセージの挿入が必要です。Oracle のデーターベース、またはこの要件が付いた他のデーターベースを実行している場合は、この属性を **false** に設定します。

SQLProperties

特定のデーターベースの DDL と DML を指定します。特定の DDL や DML ステートメントが無効になっていない場合は、デフォルトの Hypersonic 設定がそのステートメントに使用されます。

UseNDBFailoverStrategy

クラスター環境でデーターベースのトランザクションのコミットが失敗した場合に、SQL ステートメントを再度実行するか指定します。**true** に設定すると、コミットが失敗した場合 SQL ステートメントは再度実行されます。さらにエラーが発生する場合は、永続マネージャーはそのエラーは以前のトランザクションが正しくコミットされているためと仮定し、エラーを無視します。デフォルトでは、この属性は **false** に設定されています。



注記

クラスター環境で実行する MySQL などの一部のデータベースは、データーベースのトランザクションのコミット中に失敗することがあります。これが発生すると、最終的なトランザクションの状態は不明です。

MaxParams

メッセージのロード中に、準備済みステートメントごとに許可されたパラメーターの最大数を指定します。デフォルト値は **500** です。

5.8. JMS ユーザーマネージャーの設定

JMS User Manager は事前設定したクライアント ID をユーザーにマッピングします。設定したログインモジュールに応じてユーザーとロールのテーブルの管理も行います。

JMSUserManager の設定例は以下のとおりです。

```
<mbean code="org.jboss.jms.server.plugin.JDBCJMSUserManagerService"
  name="jboss.messaging:service=JMSUserManager"
  xmbean-dd="xmdesc/JMSUserManager-xmbean.xml">
  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>
  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>
  <attribute name="DataSource">java:/DefaultDS</attribute>
  <attribute name="CreateTablesOnStartup">true</attribute>
  <attribute name="SqlProperties">
    CREATE_USER_TABLE=CREATE TABLE JBM_USER (USER_ID VARCHAR(32) NOT NULL,
      PASSWD VARCHAR(32) NOT NULL, CLIENTID VARCHAR(128),
      PRIMARY KEY(USER_ID)) ENGINE = INNODB

    CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL,
      USER_ID VARCHAR(32) NOT NULL, PRIMARY KEY(USER_ID, ROLE_ID))
      ENGINE = INNODB

    SELECT_PRECONF_CLIENTID=SELECT CLIENTID FROM JBM_USER WHERE USER_ID=?

    POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID)
      VALUES ('jdoe','jdoepw','jdoe-id')
  </attribute>
</mbean>
```

5.8.1. JMSUserManager MBean 属性

CreateTablesOnStartup

JMS ユーザーマネージャーが起動時にテーブル（およびインデックス）の作成を試行するようにするには、これを **true** に設定します。テーブルまたはインデックスが既に存在している場合は、**SQLException** は JDBC ドライバーによって送出されますが、永続マネージャーはこれを無視するためその動作は継続します。

デフォルトでは、**CreateTablesOnStartup** 属性の値は **true** に設定されています。

UsingBatchUpdates

データベースが JDBC バッチ更新に対応している場合は、これを **true** に設定します。JDBC 永続マネージャーは、バッチ内のデータベースアップデートを複数まとめ、パフォーマンスを向上します。

デフォルトでは、**UsingBatchUpdates** 属性の値は **false** に設定されています。

SQLProperties

特定のデーターベースの DDL と DML を指定します。特定の DDL や DML ステートメントがオーバーライドされていない場合は、デフォルトの Hypersonic 設定がそのステートメントに使用されます。

デフォルトユーザーとロールのデータをここでも指定可能です。挿入データは、上記の例のように **POPULATE.TABLES** で開始するプロパティー名で指定する必要があります。

Database Connection Retry パラメーターは、接続失敗が検出された場合接続を再確立するか、何度再接続を試みるか、接続再試行の間隔はどの程度かを制御します。

RetryOnConnectionFailure

MBean がデータベースへの再接続を試行すべきか指定します。デフォルト値は **false** です。

MaxRetry

DataSource 接続失敗の上限を指定します。デフォルトは **25** です。パラメーターを **-1** に設定し、「Retry forever」モードを有効にします。このパラメーターは ***RetryOnConnectionFailure*** が **true** に設定されている場合有効です。

重要

クラスター化されたデスティネーションからメッセージをコンシュームするクライアントは、クローズされると応答しなくなります。これは、ノードの ***MaxRetry*** 値が **-1** に設定されている場合に発生し、データベースへの接続が失われます。この問題を回避するには、ノードの***MaxRetry*** パラメーターを**-1**よりも大きな値に設定してください。 **[database]-persistence-service.xml**ファイルにある **PersistenceManager**、**PostOffice**、**JMSUserManager** MBeanの属性値を設定してください。

RetryInterval

連続する再試行の間隔を指定します。デフォルトは、1000 (ミリ秒) となっています。このパラメーターは、***RetryOnConnectionFailure*** が **true** に設定されている場合に有効です。

5.9. デスティネーションの設定

5.9.1. 事前設定したデスティネーション

JBoss Messaging にはサーバーの起動時にデプロイされる事前設定したデスティネーションの一連のデフォルトが同梱されています。これらのデスティネーションの設定情報は **destinations-service.xml** のセクションにあります。

```
<!--
      The Default Dead Letter Queue. This destination is a dependency of
an EJB MDB container.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=DLQ"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-
name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>

   <!--
      The Default Expiry Queue.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=ExpiryQueue"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
```

```
      <depends optional-attribute-
name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>
```

5.9.2. キューの設定

5.9.2.1. キュー MBean の属性

Name

キュー名を定義します。

JNDIName

キューをバインドする JNDI 名を定義します。

DLQ

このキューの DLQ (Dead Letter Queue) を定義して、Server Peer 設定ファイルで設定されているすべての値を無効にします。

ExpiryQueue

Expiry Queue を定義して、Server Peer 設定ファイルで設定されているすべての値を無効にします。

RedeliveryDelay

このキューに適用される再配信の遅延を定義して、Server Peer 設定ファイルで設定されているすべての値を無効にします。

MaxDeliveryAttempts

設定されている場合、メッセージが DLQ に送信されるまでに試行されるメッセージの配信最大回数を定義します。デフォルト値は **-1** です。これは、Server Peer 設定ファイルからの値が使用されていることを意味します。他のどの設定も Server Peer 設定ファイルで設定されている値を無効にします。

CreatedProgrammatically

キューがプログラムによって作成された場合は **true** を返します。

MessageCount

キューのメッセージ合計数を返します。スケジュールされたメッセージ数に、配信されている数と配信されていない数が加わります。

ScheduledMessageCount

キューの **スケジュールされたメッセージ** の数を返します。これは後日に配信が予定されているメッセージの数です。

スケジュールされた配信により、特定のメッセージが配信される最も早い時刻を指定できます。例えば、今から 2 時間は配信されないように指定して、今メッセージを送ることができます。詳しくはメッセージのヘッダーで以下を設定してください。

```
long now = System.currentTimeMillis();

Message msg = sess.createMessage();

msg.setLongProperty(JBossMessage.JMS_JBOSS_SCHEDULED_DELIVERY_PROP_NAME,
  now + 1000 * 60 * 60 * 2);

prod.send(msg);
```

MaxSize

キューで保留できるメッセージの最大数を指定します。過剰なメッセージは破棄されます。デフォルト値は **-1** で、バインドされません。

Clustered

デスティネーションがクラスター化されている場合、この属性は **true** に設定する必要があります。

MessageCounter

各キューはメッセージカウンターを維持します。

MessageCounterStatistics

メッセージカウンターの統計です。

MessageCounterHistoryDayLimit

メッセージカウンターの履歴を保持する最長日数です。Server Peer 設定ファイルで設定されているすべての値を無効にします。

ConsumerCount

現在キューからコンシュームしているコンシューマーの数です。

DropOldMessageOnRedeploy

以前にデプロイした属性とは異なるクラスター属性でキューサービスをどのように処理するか指定します。**true** に設定すると、キューサービスの属性に異なるクラスター属性が含まれている場合はキューサービスが再デプロイした後に、キューに残っているすべてのメッセージは削除されます。**false** (デフォルト) に設定すると、すべてのメッセージは保持されます。

警告

デスティネーションを再デプロイする場合は、クラスター内のすべてのノードをシャットダウンし、適切な設定変更を加え、ノードを再起動する必要があります。

非クラスター化のキューからクラスター化のキューに再デプロイするには、クラスター属性を **true** に設定し、キューサービス設定を各ノードに追加する必要があります。

クラスター化のキューから非クラスター化のキューに再デプロイするには、キュー設定の 1 つでクラスター属性を **false** に設定し、クラスター内のすべての他のキューを削除する必要があります。

5.9.2.1.1. Destination Security Configuration

<SecurityConfig> は、デスティネーションに読み取り、書き込み、作成を行うことができるロールを決定します。JBossMQ デスティネーションのセキュリティ設定と同じ構文とセマンティクスを使用します。

```
<SecurityConfig>
  <security>
    <role read="true" write="true" create="true"/>
  </security>
<SecurityConfig>
```

<SecurityConfig> 要素には 1 つの <security> 要素を含む必要があります。それには、複数の <role> 要素を含むことができます。<role> 要素は以下の属性を使用してその特定のロールに対しアクセスの種類を定義します。

read

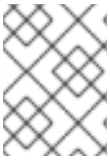
ロールがコンシューマーを作成、メッセージを受信、デスティネーションの閲覧を行うことができると指定します。

write

ロールがプロデューサーを作成、デスティネーションにメッセージを送信できると指定します。

create

ロールがこのデスティネーションで持続性のあるサブスクリプションを作成できると指定します。



注記

デスティネーションのセキュリティ設定はオプションです。**SecurityConfig** 要素を指定しないと、Server Peer からのデフォルトのセキュリティ設定が使用されます。

5.9.2.1.2. Destination paging parameters

Pageable Channels (ページング可能なチャンネル) は、キュー単位またはトピック単位でメモリに一度に保存できるメッセージの最大数を指定できる JBoss Messaging の機能です。JBoss Messaging

は、ブロック単位で透過的にメッセージをストレージ間でページングします。これによって、チャンネルサイズが大きくなると、パフォーマンスを低下させることなくキューとサブスクリプションのサイズを非常に大きくすることができます。

ページング可能なチャンネルに関連するパラメーターは以下のとおりです。

FullSize

一度にメモリでキューまたはトピックのサブスクリプションが保持するメッセージの最大数を指定します。実際のキューはこれより多くのメッセージを保持できますが、メッセージは追加されたりコンシュームされたりするためストレージ間でページングされます。値を指定しない場合のデフォルト値は **75000** です。

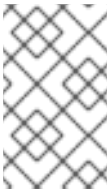
PageSize

キューまたはサブスクリプションからメッセージを読み込む場合の、動作ごとに事前に読み込んだメッセージの最大数を指定します。値を指定しない場合のデフォルト値は **2000** です。

DownCacheSize

メッセージがストレージにフラッシュされるまでに Down Cache が保持するメッセージの最大数を指定します。デフォルト値は **2000** メッセージです。

メッセージがキューからストレージにページングされる場合、メッセージはストレージに書き込まれる前に **Down Cache** に入ります。これにより書き込みを単独の動作として発生させ、パフォーマンスを向上させます。



注記

一時的なキューのページングパラメーターは適切な接続ファクトリで指定する必要があります。使用可能な様々な接続ファクトリについての詳細は 「接続ファクトリの設定」 を参照してください。

5.9.2.1.3. キュー管理 Bean の動作

RemoveAllMessages

キューからすべてのメッセージを削除（および消去）します。

重要

これによりキューからすべてのメッセージが永久に消去されますので、使用する場合はご注意ください。

ListAllMessages

現在キューにあるすべてのメッセージを一覧表示します。この動作で JMS セレクターを引数として使用すると、所定の基準に一致するキュー内のメッセージのサブセットを取得することができます。

ListDurableMessages

キューにあるすべての **持続性のある** メッセージを一覧表示します。この動作で JMS セレクターを引数として使用すると、所定の基準に一致するキュー内のメッセージのサブセットを取得することができます。

ListNonDurableMessages

キューにあるすべての **持続性のない** メッセージを一覧表示します。この動作で JMS セレクターを引数として使用すると、所定の基準に一致するキュー内のメッセージのサブセットを取得することができます。

ResetMessageCounter

メッセージカウンターをゼロにリセットします。

ResetMessageCounterHistory

メッセージカウンター履歴をリセットします。

ListMessageCounterAsHTML

メッセージカウンターを HTML 形式で一覧表示します。

ListMessageCounterHistoryAsHTML

メッセージカウンター履歴を HTML 形式で一覧表示します。

5.9.3. トピックの設定

5.9.3.1. トピック管理 Bean の属性

Name

トピック名を定義します。

JNDIName

トピックがバインドされる JNDI の場所を定義します。

DLQ

このトピックに使用する Dead Letter Queue (DLQ) を定義して、Server Peer 設定ファイルで設定されているすべての値を無効にします。

ExpiryQueue

このトピックに使用する Expiry Queue を定義して、Server Peer 設定ファイルで設定されているすべての値を無効にします。

RedeliveryDelay

このトピックに対し再配信を試行する間の遅延期間を定義して、Server Peer 設定ファイルで設定されているすべての値を無効にします。

MaxDeliveryAttempts

設定されている場合は、メッセージが DLQ に送信されるまでに試行されるメッセージの配信最大回数を定義します。デフォルト値は `-1` であり、Server Peer 設定ファイルからの値を使用することを指定します。他のどの設定も Server Peer 値を無効にします。

CreatedProgrammatically

トピックがプログラムによって作成された場合は `true` を返します。

MaxSize

トピックのサブスクリプションで保留できるメッセージの最大数を指定します。過剰なメッセージはトピックから破棄されます。デフォルト値は **-1** で、サイズ制限は適用しません。

Clustered

デスティネーションがクラスター化されている場合は、これを **true** に設定します。

MessageCounterHistoryDayLimit

メッセージカウンター履歴を保持する最大日数を定義し、Server Peer 設定ファイルで設定されているすべての値を無効にします。

MessageCounters

トピックのサブスクリプションに対するメッセージカウンターの一覧を返します。

AllMessageCount

トピックに属するすべてのサブスクリプションにあるメッセージの合計数を返します。

DurableMessageCount

このトピックに属するすべてのサブスクリプションにある **持続性のある** メッセージの合計数を返します。

NonDurableMessageCount

このトピックに属するすべてのサブスクリプションにある **持続性のない** メッセージの合計数を返します。

DropOldMessageOnRedeploy

以前にデプロイした属性とは異なるクラスター属性でキューサービスをどのように処理するか指定します。**true** に設定すると、キューサービスの属性に異なるクラスター属性が含まれている場合はキューサービスが再デプロイした後に、キューに残っているすべてのメッセージは削除されます。**false** (デフォルト) に設定すると、すべてのメッセージは保持されます。

警告

デスティネーションを再デプロイする場合は、クラスター内のすべてのノードをシャットダウンし、適切な設定変更を加え、ノードを再起動する必要があります。

非クラスター化のキューからクラスター化のキューに再デプロイするには、クラスター属性を **true** に設定し、キューサービス設定を各ノードに追加する必要があります。

クラスター化のキューから非クラスター化のキューに再デプロイするには、キュー設定の 1 つでクラスター属性を **false** に設定し、クラスター内のすべての他のキューを削除する必要があります。

AllSubscriptionsCount

このトピックに属するすべてのサブスクリプションの数を返します。

DurableSubscriptionsCount

このトピックに属するすべての持続性のあるサブスクリプションの数を返します。

NonDurableSubscriptionsCount

このトピックに属するすべての持続性のないサブスクリプションの数を返します。

5.9.3.1.1. Destination Security Configuration

<SecurityConfig> は、デスティネーションに読み取り、書き込み、作成を行うことができるロールを決定します。JBossMQ デスティネーションのセキュリティ設定と同じ構文とセマンティクスを使用します。

<SecurityConfig> 要素には 1 つの <security> 要素を含む必要があります。それには、複数の <role> 要素を含むことができます。<role> 要素は以下の属性を使用してその特定のロールに対しアクセスの種類を定義します。

read

ロールがコンシューマーを作成、メッセージを受信、デスティネーションの閲覧を行うことができると指定します。

write

ロールがプロデューサーを作成、デスティネーションにメッセージを送信できると指定します。

create

ロールがこのデスティネーションで持続性のあるサブスクリプションを作成できると指定します。

注記

デスティネーションのセキュリティ設定はオプションです。**`SecurityConfig`** 要素を指定しないと、Server Peer からのデフォルトのセキュリティ設定が使用されます。

5.9.3.1.2. Destination paging parameters

以前はキューまたはサブスクリプションをサポートするアプリケーションについては、キューはメモリに完全に保存されている必要がありました。しかしこれは非常に大きなキューまたはサブスクリプションには常に可能ではありませんでした。

Pageable Channels (ページング可能なチャンネル) は、キュー単位またはトピック単位でメモリに一度に保存できるメッセージの最大数を指定することができる JBoss Messaging の新しい機能です。JBoss Messaging は、ブロック単位で透過的にメッセージをストレージ間でページングします。これにより、チャンネルサイズが大きくなると、パフォーマンスを低下させることなくキューやサブスクリプションのサイズを非常に大きくすることができます。これは、非常に基本的なハードウェアで 2 キロバイトのメッセージを 1 千万回以上キューで検証されているため、さらに多くのメッセージ数に拡大する可能性があります。

ページング可能なチャンネルに関連するパラメーターは以下のとおりです。

FullSize

一度にメモリでキューまたはトピックのサブスクリプションが保持するメッセージの最大数を指定します。実際のキューはこれより多くのメッセージを保持できますが、メッセージは追加されたりコンシュームされたりするためストレージ間でページングされます。値を指定しない場合のデフォルト値は **75000** です。

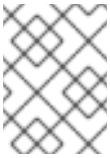
PageSize

キューまたはサブスクリプションからメッセージを読み込む場合の、動作ごとに事前に読み込んだメッセージの最大数を指定します。値を指定しない場合のデフォルト値は **2000** です。

DownCacheSize

メッセージがストレージにフラッシュされるまでに Down Cache が保持するメッセージの最大数を指定します。デフォルト値は **2000** メッセージです。

メッセージがキューからストレージにページングされる場合、メッセージはストレージに書き込まれる前に **Down Cache** に入ります。これにより書き込みを単独の動作として発生させ、パフォーマンスを向上させます。



注記

一時的なキューのページングパラメーターは適切な接続ファクトリで指定する必要があります。詳細は「接続ファクトリの設定」セクションを参照してください。

5.9.3.2. トピック管理 Bean の動作

RemoveAllMessages

このトピックに属するサブスクリプションからすべてのメッセージを削除（および消去）します。

重要

これによりトピックからすべてのメッセージが永久に消去されますので、使用する場合はご注意ください。

ListAllMessages

指定のサブスクリプションに属するすべてのメッセージを一覧表示します。この動作で JMS セレクターを引数として使用すると、所定の基準に一致するキュー内のメッセージのサブセットを取得することができます。

ListDurableMessages

指定のサブスクリプションに属するすべての持続性のあるメッセージを一覧表示します。この動作で JMS セレクターを引数として使用すると、所定の基準に一致するキュー内のメッセージのサブセットを取得することができます。

ResetMessageCounter

メッセージカウンターをゼロにリセットします。

ResetMessageCounterHistory

メッセージカウンター履歴をリセットします。

ListAllSubscriptionsAsHTML

このトピックに属するすべてのサブスクリプションを HTML 形式で一覧表示します。

ListDurableSubscriptionsAsHTML

このトピックに属するすべての持続性のあるサブスクリプションを HTML 形式で一覧表示します。

ListNonDurableSubscriptions

指定のサブスクリプションに属するすべての持続性のないメッセージを一覧表示します。この動作で JMS セレクターを引数として使用すると、所定の基準に一致するキュー内のメッセージのサブセットを取得することができます。

ListNonDurableSubscriptionsAsHTML

このトピックに属するすべての持続性のないサブスクリプションを HTML 形式で一覧表示します。

5.10. 接続ファクトリの設定

JBoss Messaging はデフォルトでは起動時に JNDI 内で 2 つの接続ファクトリをバインドするように設定されています。

1 つ目の接続ファクトリはデフォルトの非クラスター化した接続ファクトリです。この接続ファクトリにより元々 JBossMQ に対して書き込みされたアプリケーションとの互換性を維持することができ、自動フェールオーバーまたは負荷分散は含まれていません。クライアント側の自動フェールオーバーまたは負荷分散が必要でない場合は、この 1 つ目の接続ファクトリを使用してください。

1 つ目の接続ファクトリは以下の JNDI コンテキストにバインドされています。

- **/ConnectionFactory**
- **/XAConnectionFactory**
- **java:/ConnectionFactory**
- **java:/XAConnectionFactory**.

2 つ目の接続ファクトリはデフォルトのクラスター化した接続ファクトリで、以下の JNDI コンテキストにバインドされています。

- **/ClusteredConnectionFactory**
- **/ClusteredXAConnectionFactory**
- **java:/ClusteredConnectionFactory**
- **java:/ClusteredXAConnectionFactory**

ある接続ファクトリにデフォルトのクライアント ID を提供したり、接続ファクトリを別の JNDI の場所にバインドしたい場合は、よく考慮した上で、追加の接続ファクトリを設定およびデプロイします。新しい接続ファクトリをデプロイするには **connection-factories-service.xml** で、新規の **ConnectionFactory** 管理 Bean を設定します。

また、新しいサービスデプロイメント記述子 ***<name>*-service.xml** を作成したり、**$JBOSS_HOME/server/messaging/deploy** でそれをデプロイすることもできます。

ご使用の接続ファクトリで関連する属性を設定することで、自動フェールオーバーまたは負荷分散に対応できます。

例5.1 接続ファクトリ

この例の接続ファクトリは、事前設定したクライアント ID **myClientID** を持つ接続ファクトリを作成し、JNDI ツリーの **/MyConnectionFactory** と **/factories/cf** の 2 つの場所にバインドされます。

例では以下のデフォルト値を無効にします。

- **PreFetchSize**
- **DefaultTempQueueFullSize**
- **DefaultTempQueuePageSize**
- **DefaultTempQueueDownCacheSize**
- **DupsOKBatchSize**
- **SupportsFailover**
- **SupportsLoadBalancing**
- **LoadBalancingFactory**

この接続ファクトリは、デフォルトのリモートコネクターを使用します。その接続ファクトリを持つ別のリモートコネクターを使用するには、**Connector** 属性を変更して、使用したいコネクターのサービス名を指定します。

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory"
  name="jboss.messaging.connectionfactory:service=MyConnectionFactory"
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>jboss.messaging:service=PostOffice</depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/factories/cf</binding>
    </bindings>
  </attribute>

  <attribute name="ClientID">myClientID</attribute>

  <attribute name="SupportsFailover">true</attribute>

  <attribute name="SupportsLoadBalancing">false</attribute>

  <attribute name="LoadBalancingFactory">
```

```
      org.acme.MyLoadBalancingFactory
    </attribute>

    <attribute name="PrefetchSize">1000</attribute>

    <attribute name="SlowConsumers">false</attribute>

    <attribute name="StrictTck">true</attribute>

    <attribute name="SendAcksAsync">false</attribute>

    <attribute name="DefaultTempQueueFullSize">50000</attribute>

    <attribute name="DefaultTempQueuePageSize">1000</attribute>

    <attribute name="DefaultTempQueueDownCacheSize">1000</attribute>

    <attribute name="DupsOKBatchSize">10000</attribute>
  </mbean>
```

5.10.1. ConnectionFactory 管理 Bean の属性

ClientID

接続ファクトリはクライアント ID を付けて事前設定することができます。この接続ファクトリで作成された接続はすべてこのクライアント ID を取得します。

JNDIBindings

この接続ファクトリに使用できる JNDI バインディングを一覧表示します。

PrefetchSize

コンシューマーフロー制御の目的で一度にウィンドウが保持できるメッセージ数を指定します。このウィンドウのサイズはサーバーがブロックされずにコンシューマーに送信できるメッセージ数を決定します。各コンシューマはコンシュームするメッセージのバッファーを管理します。

Transmission Control Protocol (TCP) は、独自のフロー制御を別に実装します。TCP ウィンドウのサイズが ***`PrefetchSize`*** パラメーターより小さいと、メッセージ消費がブロックされる可能性もあります。

SlowConsumers

速度の遅いコンシューマー用の許容範囲のバッファサイズを小さくするか指定します。速度の遅いコンシューマと速いコンシューマが混在している場合は、速度の遅いコンシューマーのバッファーサイズを小さくすることにより、速度の速いコンシューマーはより多くのメッセージをコンシュームすることになります。バッファリングを完全に無効にすることは不可能ですが、**`SlowConsumers`** 属性を **`true`** に設定することで、バッファーサイズは小さくなります。この属性を **`true`** に設定することは、**`PrefetchSize`** を **`1`** に設定することと同等であり、これは使用できる最小値です。

StrictTck

属性を **`true`** に設定すると、厳密な JMS 動作が有効になります。厳密な JMS 動作はテクノロジ互換性キット (TCK) で必要です。

SendAcksAsync

属性を **true** に設定すると、非同期に受信確認が送信されます。これにより、特に **auto_acknowledge** モードがアクティブな場合に、パフォーマンスを向上させることができます。

DefaultTempQueueFullSize

一時的なフルサイズのキューのデスティネーションにページングパラメーターを指定するオプションの属性であり、この接続ファクトリで作成される接続にスコープされます。デフォルト値は **200000** です。これらの属性に関する詳細は 「Destination paging parameters」 を参照してください。

DefaultTempQueuePageSize

一時的なページサイズのデスティネーションにページングパラメーターを指定するオプションの属性であり、この接続ファクトリで作成される接続にスコープされます。デフォルト値は **2000** です。これらの属性に関する詳細は 「Destination paging parameters」 を参照してください。

DefaultTempQueueDownCacheSize

一時的な Down Cache サイズのデスティネーションにページングパラメーターを指定するオプションの属性であり、この接続ファクトリで作成される接続にスコープされます。デフォルト値は **2000** です。これらの属性に関する詳細は 「Destination paging parameters」 を参照してください。

DupsOKBatchSize

ローカルにバッファされた **DUPS_OK_ACKNOWLEDGE** の受信確認の数をそれらが送信される前に指定します。デフォルト値は **2000** です。

SupportsLoadBalancing

クライアント側の負荷分散がクラスター化したインストール上の接続ファクトリに有効であるか指定します。負荷分散が有効な場合、その接続ファクトリで作成された接続はクラスターのノード全体で負荷分散されるようになります。特定のノードで作成された接続はそのノードに留まります。デフォルト値は **false** です。

SupportsFailover

クライアント側の自動フェールオーバーがクラスター化したインストール上の接続ファクトリに有効であるか指定します。自動フェールオーバーが有効な場合は、接続に関する問題が検出された時に、JBoss Messaging は自動的にクラスター内の別のノードに透過的にフェールオーバーします。デフォルト値は **false** です。

注記

自動フェールオーバーが無効な場合、同期の JMS 動作で接続の例外を検出することがユーザーコードの役目です。JMS **ExceptionListener** をインストールして、非同期に例外を検出する必要があります。例外が検出されると、クライアント側のコードは HAJNDI を介して新しい接続ファクトリを検索し、接続を再作成する必要があります。

DisableRemotingChecks

接続ファクトリが対応する JBoss Remoting Connector が適切な値を使用していることを確認するかどうか設定します。JBoss Messaging はこうした値に非常に精度の高い確認を行うため、変更する必要はほとんどありません。このサニティチェックを無効にするには、**DisableRemotingChecks** を **false** に設定します。デフォルト値は **true** です。

警告

リモートチェックを無効にしないでください。システムが不安定になることがあります。

LoadBalancingFactory

接続ファクトリが使用するクライアント側の負荷分散ファクトリの実装を指定します。その値は、インターフェースの **`org.jboss.jms.client.plugin.LoadBalancingFactory`** を実装するクラス名に対応している必要があります。

デフォルト値は **`org.jboss.jms.client.plugin.RoundRobinLoadBalancingFactory`** であり、クラスター全体でラウンドロビン式に接続の負荷分散を行います。

Connector

接続ファクトリが使用するリモートコネクターを指定します。異なる接続ファクトリに違うコネクターを使用することができるため、HTTP トランスポートを使用してサーバーと通信を行う接続ファクトリと、bisocket トランスポートを使用して通信を行う別の接続ファクトリをデプロイすることができます。

EnableOrderingGroup

厳密なメッセージ順を **`ConnectionFactory`** で有効にするか指定します。**`true`** に設定すると、有効な接続ファクトリから作成されるプロデューサーが送信したメッセージはすべて順序グループのメッセージになります。このパラメーターのデフォルト値は **`false`** です。

DefaultOrderingGroupName

メッセージ順序グループのデフォルト名を指定します。その指定された名前は **`EnableOrderingGroup`** パラメーターが **`true`** に設定された時点で有効になります。この属性がない場合は、グループ名は自動的に生成されます。

5.11. リモートコネクターの設定

JBoss Messaging は JBoss Remoting を使用して、クライアントとサーバー間のすべての通信を行います。

JBoss Remoting の設定と機能に関する詳細は 『管理設定ガイド』 の [リモーティング] の章を参照してください。

デフォルト設定には 1 つのリモートコネクターが含まれており、単一のデフォルト接続ファクトリで使用されます。各接続ファクトリは異なるコネクターを使用するように設定することができます。

デフォルトのコネクターは、サーバー側でのみ接続を待機して受け取る TCP ソケットベースのトランスポートであるリモート bisocket トランスポートを使用するように設定されています。つまり、接続は常にクライアント側から開始します。受信接続だけがサーバー上で許可されている場合、または発信接続だけがクライアントから許可されている場合に、一般的なファイアウォールのシナリオに理想的なコネクターです。

より高度なセキュリティが要求される場合に、SSL を使用するように bisocket トランスポートを設定できます。

その他のサポートされているトランスポートには HTTP トランスポートがあります。Hypertext Transfer Protocol を使用して、クライアントとサーバー間で通信を行います。クライアントはメッセージがないか定期的にサーバーをポーリングして、データを受信します。このトランスポートはサーバーとクライアント間のファイアウォールがサーバー上の着信 HTTP トラフィックだけを許可する場合に理想的です。そのポーリング動作と HTTP の制限のため、このトランスポートは bisocket トランスポートほどパフォーマンスはよくありません。高負荷となるような状況を処理するようには設計されていません。

現在これ以外のリモートトランスポートは JBoss Messaging ではサポートされていません。

リモート設定に関する詳細は、***$JBOSS_HOME*/server/*$SERVER*/deploy/messaging/remoting-bisocket-service.xml** を参照してください。以下のコードは、bisocket リモート設定の一例です。

例5.2 Bisocket リモート設定

```
        <?xml version="1.0" encoding="UTF-8"?>

<!--
     Standard bisocket-based Remoting service deployment descriptor.

     $Id: remoting-bisocket-service.xml 3981 2008-03-28 18:00:41Z timfox
$
 -->

<server>

   <!-- Standard bisocket connector - the bisocket transport only opens
connection from client->server
        so can be used with firewalls where only outgoing connections
are allowed.
        For examples of HTTP and SSL transports see docs/examples -->
   <mbean code="org.jboss.remoting.transport.Connector"
          name="jboss.messaging:service=Connector,transport=bisocket"
          display-name="Bisocket Transport Connector">
      <attribute name="Configuration">
         <config>
            <invoker transport="bisocket">

               <!-- There should be no reason to change these
parameters - warning!
                    Changing them may stop JBoss Messaging working
correctly -->
               <attribute name="marshaller"
isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="unmarshaller"
isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="dataType"
isParam="true">jms</attribute>
               <attribute name="socket.check_connection"
isParam="true">false</attribute>
               <attribute
name="serverBindAddress">${jboss.bind.address}</attribute>
               <attribute
```

```
name="serverBindPort">${jboss.messaging.connector.bisocket.port:4457}
</attribute>
               <attribute name="clientSocketClass"
isParam="true">org.jboss.jms.client.remoting.ClientSocketWrapper</attrib
ute>
               <attribute
name="serverSocketClass">org.jboss.jms.server.remoting.ServerSocketWrapp
er</attribute>
               <attribute
name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</
attribute>

               <!-- the following parameters are useful when there is a
firewall between client and server. Uncomment them if so.-->
               <!--
               <attribute name="numberOfCallRetries"
isParam="true">1</attribute>
               <attribute name="pingFrequency"
isParam="true">214748364</attribute>
               <attribute name="pingWindowFactor"
isParam="true">10</attribute>
               <attribute name="generalizeSocketException"
isParam="true">true</attribute>
               -->

               <!-- Now remoting supports socket write timeout
configuration. Uncomment this if you need it. -->
               <!--
               <attribute name="writeTimeout"
isParam="true">30000</attribute>
               -->

               <!-- End immutable parameters -->

               <attribute name="stopLeaseOnFailure"
isParam="true">true</attribute>

               <!-- Periodicity of client pings. Server window by
default is twice this figure -->
               <attribute name="clientLeasePeriod"
isParam="true">10000</attribute>
               <attribute name="validatorPingPeriod"
isParam="true">10000</attribute>
               <attribute name="validatorPingTimeout"
isParam="true">5000</attribute>

               <attribute name="failureDisconnectTimeout"
isParam="true">0</attribute>
               <attribute name="callbackErrorsAllowed">1</attribute>
               <attribute
name="registerCallbackListener">false</attribute>
               <attribute name="useClientConnectionIdentity"
isParam="true">true</attribute>

             <attribute name="timeout" isParam="true">0</attribute>
```

```
                <!-- Max Number of connections in client pool. This
should be significantly higher than
                     the max number of sessions/consumers you expect -->
                <attribute name="JBM_clientMaxPoolSize"
isParam="true">200</attribute>

                <!-- The maximum time to wait before timing out on
trying to write a message to socket for delivery -->
                <attribute name="callbackTimeout">10000</attribute>

                <!-- Use these parameters to specify values for binding
and connecting control connections to
                     work with your firewall/NAT configuration
                <attribute name="secondaryBindPort">xyz</attribute>
                <attribute name="secondaryConnectPort">abc</attribute>
                -->

            </invoker>
            <handlers>
               <handler
subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler
</handler>
            </handlers>
         </config>
      </attribute>
   </mbean>

</server>
```

変更による影響を完全に理解していない限りは変更すべきでない制限された属性があります。以下の属性は、ご使用のプロジェクトの要件に変更、設定可能です。

clientLeasePeriod

クライアントは定期的に **ハートビート** をサーバーに返して、それらがアクティブであることを確認します。サーバーがある一定の期間ハートビートを受信しない場合は、接続を終了し、クライアントのセッションに対応するすべてのリソースを削除します。**clientLeasePeriod** は、ハートビート間の期間をミリ秒単位で決定します。デフォルト値は **10000** です。

デフォルトでは、サーバーは **clientLeasePeriod** の 2 倍の時間内にハートビートを受信しないとクライアントを閉じます。実際にはこの期間はシステムの負荷に応じて自動的に変更します。

numberOfRetries

接続が使用可能になるのを待機している間に JBoss Remoting がクライアントプールでブロックする秒数です。1 クライアントからサーバーに同時にアクセスしているセッションが非常に多く、プールから接続を取得できない場合は、この値を高くすると良いかもしれません。

clientMaxPoolSize

JBoss Remoting は要求に対応するクライアント側の TCP 接続プールを維持します。1 クライアントからサーバーに同時にアクセスしているセッションが非常に多く、プールから接続を取得できない場合は、この値を高くすると良いかもしれません。

secondaryBindPort

bisocket トランスポートは制御接続を使用して、サーバーとクライアント間で制御メッセージを渡し合います。この属性は 第 2 **ServerSocket** のバインド先となるアドレスを定義します。ファイアウォールの内側で動作させるためには、ご使用のファイアウォールの設定に応じて特定の値を指定する必要があることがあります。

secondaryConnectPort

クライアントが接続に使用するポートです。これを指定することで、クライアントが NAT ルーターと動作できるようになります。

maxPoolSize

要求に対応するためにサーバー側で使用するスレッド数です。

デフォルトでは、JBoss Messaging は **${jboss.bind.address}** にバインドしており、**./run.sh -c *[yourconfig]* -b *[yourIP]*** コマンドを実行することで定義できます。

必要であれば **remoting-bisocket-service.xml** を変更して、別の通信ポートを使用することができます。

警告

上述したもの以外のコネクター設定の値は変更しないでください。他の値を変更すると、JBoss Messaging が正しく機能しなくなる恐れがあります。

5.12. SERVICEBINDINGMANAGER

SeviceBindingManager は、別のポート範囲を使用して同じ IP で実行している複数のアプリケーションサーバーのインスタンスを提供するため、開発中に使用すると便利です。別の方法でこれを行うこともできますが、**ServiceBindingManager** を使用する方が非常に簡単です。

5.13. メッセージ駆動 BEAN

メッセージ駆動 Bean は、J2EE アプリケーションがメッセージの非同期処理を可能にするエンタープライズ Bean であり、JMS メッセージとして動作します。イベントの代わりにメッセージを受信するという点を除いてイベントリスナーに似ています。メッセージは任意の J2EE コンポーネント (アプリケーションクライアント、別のエンタープライズ Bean、または Web コンポーネント) または JMS アプリケーションあるいは J2EE テクノロジーを使用しないシステムで送信できます。この定義は『 http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts5.html 』 に存在し、メッセージ駆動 Bean (MDB :Message Driven Bean) の詳細もここで提供されます。

MDB は、デプロイメント記述子またはアノテーションを使用して指定できます。

記述子の使用

```
            <enterprise-beans>
  <message-driven>
    <ejb-name>MDBExample</ejb-name>
    <destination-jndi-name>queue/@QUEUE_NAME@</destination-jndi-name>
  </message-driven>
</enterprise-beans>
```

アノテーションの使用

```
            @MessageDriven(mappedName="jms/Queue")
public class SimpleMessageBean implements MessageListener {
    @Resource
    private MessageDrivenContext mdc;
    ...
```

MDB はプロパティを使用して設定します。これらのプロパティは JCA 仕様で指定されたものと JBoss 拡張機能として利用可能なものに分割されます。

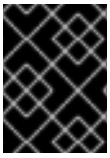
重要

Remarks の欄で明示的に呼び出さない限り、表5.1「JCA 仕様で提供される MDB プロパティ」 に記載の MDB プロパティは必須ではありません。

表5.1 JCA 仕様で提供される MDB プロパティ

Name	タイプ	デフォルト値	リマーク
定義	java.lang.String	いいえ	このプロパティーは必須です。 Queue または Topic の JNDI 名
destinationType	java.lang.String	いいえ	宛先有効値のタイプは javax.jms.Queue または javax.jms.Topic
messageSelector	java.lang.String	いいえ	サブスクリプションのメッセージセレクター

Name	タイプ	デフォルト値	リマーク
acknowledgeMode	int	AUTO_ ACKNOWLEDGE	トランザクション jms を使用しない場合の受信確認のタイプ - 有効値 AUTO_ ACKNOWLEDGE または DUPS_OK_ ACKNOWLEDGE
clientID	java.lang.String		接続のクライアント id
subscriptionDurability	String	NonDurable	トピックサブスクリプションが Durable かどうか。有効値は Durable または NonDurable
subscriptionName	String	いいえ	トピックサブスクリプションのサブスクリプション名



重要

Remarks の欄で明示的に呼び出さない限り、表5.2「JBoss 拡張機能として提供される MDB プロパティ」 に記載の MDB プロパティは必須ではありません。

表5.2 JBoss 拡張機能として提供される MDB プロパティ

Name	タイプ	デフォルト値	リマーク
isTopic	boolean	false	destinationType を設定
providerAdapterJNDI	java.lang.String	DefaultJMSProvider	JMS プロバイダーの JNDI 名
user	java.lang.String	いいえ	JMS サーバーに接続するのに使用されるユーザー ID
pass	java.lang.String	いいえ	ユーザーのパスワード
maxMessages	int	1	メッセージが MDB に送信される前にメッセージを上記の回数読み込みます。 コンテキスト過多によるコンテキストの切り替えを回避するために、各メッセージは別々に同じスレッド上で送信されます。

Name	タイプ	デフォルト値	リマーク
minSession	int	1	この mdb にメッセージを同時に送信できる JMS セッションの最小数
maxSession	int	15	この mdb にメッセージを同時に送信できる JMS セッションの最大数
reconnectInterval	long	10 秒	JMS プロバイダーへの (再) 接続試行を行う間隔 (秒単位)
keepAlive	long	60 秒	セッションがアライブ状態である時間 (ミリ秒)
sessionTransacted	boolean	true	セッションがトランザクション処理されたかどうか
useDLQ	boolean	true	Dead Letter Queue (DLQ) ハンドラーを使用するかどうか
dLQJNDIName	java.lang.String	queue/DLQ	DLQ の JNDI 名
dLQHandler	java.lang.String	org.jboss.resource.adapter.jms.inflow.dlq.GenericDLQHandler	org.jboss.resource.adapter.jms.inflow.DLQHandler 実装クラス名
dLQUser	java.lang.String	いいえ	JMS サーバーに dlq 接続を行うために使用されるユーザー ID
dLQPassword	java.lang.String	いいえ	dLQUser のパスワード
dLQClientID	java.lang.String	いいえ	DLQ 接続のクライアント id
dLQMaxResent	int	5	メッセージが DLQ に送信される前に再送信される最大回数

Name	タイプ	デフォルト値	リマーク
redeliverUnspecified	boolean	true	未指定のトランザクションコンテキストでメッセージを再送信するかどうか
transactionTimeout	int	デフォルト値はリソースマネージャーに設定されるタイムアウト値	トランザクションタイムアウトの時間 (秒単位)
DeliveryActive	boolean	true	MDB が最初のデプロイメントでサブスクリプションを行うか、あるいは対応する MBean の start() または stopDelivery() を待機するかどうか。サーバー起動時に MDB (起動中) にメッセージが送信されないようにする場合はこの値を false にします。

デフォルトの MDB プロパティの設定

`@org.jboss.ejb3.annotation.DefaultActivationSpecs` アノテーションを使用して MDB がデフォルトのプロパティを持つよう設定できます。

第6章 クラスタリングに関する注意点

クラスタリング関連情報を見つけ出すためには、JBoss Messaging の関連するコンポーネントへのリンクと併せて本ガイドのこの部分のそれぞれの考慮事項の要約を参照してください。

6.1. 一意の SERVER PEER ID

ほとんどの場合、JBoss Messaging は設定に最小限の変更を加えたクラスター環境で動作します。必ず行うべき重要な変更は、すべてのノードそれぞれに一意のサーバー ID を割り当てることです。

LAN クラスターを形成するノードやメッセージブリッジでリンクされているノードを含め、デプロイされるすべてのノードには一意の ID が必要です。

ServerPeerID の属性を使用して、この情報を設定します。詳しくは 「ServerPeer の属性」 を参照してください。

6.2. クラスター化したデスティネーション

JBoss Messaging はクラスター全体で Java Message Service (JMS) のキューとトピックを透過的にクラスター化します。あるノード上の分散キューまたはトピックに送信されるメッセージは、他のノードでコンシューム可能です。特定のデスティネーションをクラスター化するためには、**clustered** 属性を使用して、この機能を設定します。詳しくは 「MessagingPostOffice 属性」 を参照してください。

JBoss Messaging は、ノード間でメッセージのバランスをとり、クラスター全体に効率的に処理の負荷が分散されるよう速度の異なるコンシューマに対応します。

クラスター化したデスティネーションの他の特性を維持しながらノード間でメッセージの再分散を無効にするためには、Server Peer で **ClusterPullConnectionFactoryName** 属性を指定しないようにします。この属性に関する全詳細は 「ServerPeer の属性」 を参照してください。

6.3. クラスター化した持続性のあるサブスクリプション

JBoss Messaging の持続性のあるサブスクリプションは、複数のノード上の複数のサブスクライバが 1 つの持続性のあるサブスクリプションからコンシュームできるような方法でクラスター化されます。持続性のあるサブスクリプションは、そのトピックがクラスター化されていると自動的にクラスター化されます。

クラスター化したトピックとキューの設定に関する詳細は、「MessagingPostOffice 属性」 の ***Clustered*** 属性を参照してください。

6.4. クラスター化した一時的なデスティネーション

JBoss Messaging は、一時的なトピックとキューのクラスター化に対応します。ポストオフィスがクラスター化されると、すべての一時的なトピックとキューはクラスター化されます。

クラスター化したトピックとキューの設定に関する詳細は、「MessagingPostOffice 属性」 の ***Clustered*** 属性を参照してください。

6.5. 非クラスター化のサーバー群

すべてのノードをクラスター内で参加させたくない場合やサーバーをクラスター化させたくない場合は、***PostOffice*** のクラスター属性を **false** に設定します。

非クラスター化のサーバーの設定に関する詳細は、「MessagingPostOffice 属性」 の様々な属性を参照してください。

6.6. クラスター内のメッセージ順序

メッセージが生成された順序と同じ順序でコンシュームされるようにするためには、**`DefaultPreserveOrdering`** Server Peer 属性を **`true`** に設定することで、厳密な JMS 順を設定します。**`true`** に設定している間はメッセージをクラスター全体で自由に分散させることはできません。デフォルト値は **`false`** です。

6.7. べき等のオペレーション

永続のデスティネーションに送信されたメッセージが例外なく返されると、そのメッセージは必ず永続化されます。

例外が送出されると、メッセージが永続化され **なかった** ことは保証されません。これは、メッセージの永続化から呼び出し側に応答を返す途中で障害が発生した可能性があるためです。

したがってアプリケーションをコード化して、動作が **べき等** となるようにする必要があります — つまり、システムが矛盾することなく動作を反復することが可能ということです。

オリジナルのメッセージが正常に送信されている場合は、重複するメッセージをチェックしてそれらを破棄することで、アプリケーションレベルでこの動作を実装することができます。この **重複チェック** の機能は、有力な技術で XA トランザクションの必要がなくなります。

クラスター環境では、JBoss Messaging はデフォルトで重複チェックを行うように設定されています。

永続化に関する考慮事項は 「ServerPeer methods」、「データベースの変更」、「Persistence Manager の設定」、「メッセージブリッジの概要」 に記載されています。

6.8. クラスター化した接続ファクトリ

接続ファクトリで **`supportsLoadBalancing`** を **`true`** に設定すると、接続作成の連続試行は、使用可能なサーバー間でラウンドロビン式になります。最初のノードはランダムに選択されます。

`supportsFailover` を **`true`** に設定すると、接続エラーが検出されるたびに、フェールオーバーが透過的かつ自動的に発生するようになります。

接続ファクトリの設定に関する詳細は、「ConnectionFactory 管理 Bean の属性」 を参照してください。

第7章 JBOSS MESSAGING XA 復元の設定

このセクションでは JBoss Enterprise Application Platform の JBoss Messaging リソースに対して XA 復元を処理するための JBoss Transactions の設定方法について説明しています。

JBoss Transactions Recovery Manager は継続的に JBoss Messaging XA リソースをポーリングして復元するように設定できます。これにより、トランザクションで高い耐久性が実現します。

JBoss Transactions Recovery Manager を有効にするには、**$JBOSS_HOME/server/$PROFILE/conf/jbossts-properties.xml** に 1 行追加します。次のコードの断片に必要な行が含まれています。

```
<properties depends="arjuna" name="jta">
  <!--
  Support subtransactions in the JTA layer?
  Default is NO.
  -->
  <property name="com.arjuna.ats.jta.supportSubtransactions" value="NO"/>
  <property name="com.arjuna.ats.jta.jtaTMImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManag
erImple"/>
  <property name="com.arjuna.ats.jta.jtaUTImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionI
mple"/>
  <!--
      *** Add this line to enable recovery for JMS resources using
DefaultJMSProvider ***
  -->
  <property
name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1"

value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/Def
aultJMSProvider"/>

</properties>
```

上記の例では、Recovery Manager は JMS プロバイダローダーの **DefaultJMSProvider** を使用して JMS リソースの復元を試行しています。

DefaultJMSProvider には JBoss Enterprise Application Platform が同梱しています。これは **$JBOSS_HOME/server/$PROFILE/conf/jms-ds.xml** (または、クラスタ環境 **hajndi-jms-ds.xml**) で定義されています。異なる JMS プロバイダローダー(リモート JMS プロバイダに対応するものなど)で復元を実行するには、プロパティファイルにもう 1 行追加し、**DefaultJMSProvider** の代わりにご使用のリモートプロバイダを指定します。ご使用のプロバイダ名は、その管理 Bean 設定ファイルに一覧表示されているはずです。

プロバイダにはそれぞれ固有の名前が必要です。例えば、**com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1**、**com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING2** などです。

復元は復元可能な XAResources を実装する JMS プロバイダならいずれでも動作するはずです (つまり **XAResource.recover()** を正しく実装しているということです)。

クラスタのノードから復元を行う Recovery Manager の場合、クラスタ内のすべてのノードの **`hajndi-jms-ds.xml`** でそれぞれ 1 行追加する必要があります。

第8章 JBOSS MESSAGING のメッセージブリッジの設定

8.1. メッセージブリッジの概要

JBoss Messaging には完全に機能するメッセージブリッジが収納されています。

ブリッジは、ソースキューやトピックからメッセージをコンシュームして、一般的にはそれらを別のサーバー上にある目的のキューやトピックに送信します。ソースのサーバーと目的のサーバーは同じクラスター内にある必要はないため、ブリッジングは接続が不確かな状況下においてあるクラスターから別のクラスターにメッセージを送信する場合に確実な方法となります (WAN 全体など）。

ブリッジは JBoss Enterprise Application Platform インスタンス内に管理 Bean としてデプロイされます。その方法は、JBoss Messaging を含む Enterprise Application Platform 設定の **deploy** ディレクトリに管理 Bean 記述子を追加することです。

$JBOSS_HOME/docs/examples/jboss-messaging-examples/bridge/ にあるサンプルでは、シンプルなブリッジが JBoss Enterprise Application Platform でデプロイされ、メッセージをソースから目的のディレクトリまで移動する例が見られます。

JMS 1.1 準拠であれば、JBoss Messaging JMS サーバー以外の他のサーバーからメッセージを取得するためにもこのブリッジを使用できます。

ブリッジには障害に対するビルトインの耐性があります。ソースまたは目的のサーバーの接続が失われると、ブリッジはそのソースまたは目的のサーバーがオンラインに復帰するまで再接続を試みます。オンラインに復帰すると、通常通りに動作を再開します。

ブリッジは特定の JMS セレクターに一致するメッセージをコンシュームするように設定することができます。

キューまたはトピックからコンシュームするように設定が可能です。ブリッジがトピックからコンシュームする場合、非持続的なサブスクリプションまたは持続的なサブスクリプションでコンシュームするように設定できます。

ブリッジは 3 種類の *サービスの質* (QoS) レベルのうちひとつでメッセージを処理するように設定することができます。

ブリッジ QoS レベル

QOS_AT_MOST_ONCE

このモードでは、メッセージが最大で 1 回デスティネーションに届くように指定します。メッセージはソースからコンシュームされ、デスティネーションに送信される前に確認されます。このため、メッセージがソースから離れてデスティネーションに到着するまでの間に障害が発生すると、メッセージは失われる可能性があります。したがって、メッセージは最大で 1 回配信されることになります。

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。

QOS_DUPLICATES_OK

このモードは、メッセージがデスティネーションに正しく送信された後に、ソースからコンシュームされ確認されるように指定します。メッセージがデスティネーションに到着して確認されるまでの間に障害が発生すると、そのメッセージはシステムが復帰した時に 2 回目の送信が行われます。

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。

QOS_ONCE_AND_ONLY_ONCE

このモードでは、メッセージが 1 回だけ届くように指定します。メッセージのソースとデスティネーションが同じ JBoss Messaging サーバーインスタンス上にある場合は、同じローカルトランザクション内でメッセージの送受信を行うことができます。

ソースとデスティネーションが異なるサーバー上にある場合は、JBoss Transactions の JTA 実装で制御される JTA トランザクションを使用して、高い持続性を持つメッセージを実装できます。JTA が必要な場合は、両方の接続ファクトリとも **XAConnectionFactory** 実装である必要があります。

このモードは永続メッセージにしか使用できません。

このモードは、復元に対応するようトランザクションマネージャとリソース側の両方でロギングが必要です。このレベルの QOS を必要とする場合は、JBoss Transactions で XA Recovery を有効にする必要があります。

注記

特定のアプリケーションでは **QOS_ONCE_AND_ONLY_ONCE** を設定することなく **一度限り** セマンティックを適用できる場合があります。**QOS_DUPLICATES_OK** モードに設定し、デスティネーションで重複メッセージをチェックして破棄します。

ディスクで受信したメッセージ ID のキャッシュを維持し、受信したメッセージとこのキャッシュを比較することで、アプリケーションレベルで **QOS_ONCE_AND_ONLY_ONCE** 動作を実装できます。キャッシュは一定期間しか有効でないためこの方法は完璧ではありませんが、ご使用のアプリケーションによっては便利な選択肢となる場合があります。

8.2. ブリッジのデプロイメント

メッセージブリッジは、JBoss Messaging を含む JBoss Enterprise Application Platform インストールの **deploy** ディレクトリに管理 Bean 記述子を追加するとデプロイできます。

8.3. ブリッジの設定

以下のコードは、すべての属性を示すメッセージブリッジの設定例です。一度にすべての属性を指定すべきではないため、この設定では一部コメントアウトしている属性があります。

例8.1 メッセージブリッジの設定

```
<mbean code="org.jboss.jms.server.bridge.BridgeService"
  name="jboss.messaging:service=Bridge,name=TestBridge"
  xmbean-dd="xmdesc/Bridge-xmbean.xml">

  <!-- The JMS provider loader that is used to lookup the source
       destination
  -->
  <depends optional-attribute-name="SourceProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JMS provider loader that is used to lookup the target
       destination
  -->
```

```
  <depends optional-attribute-name="TargetProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JNDI lookup for the source destination -->
  <attribute name="SourceDestinationLookup">/queue/A</attribute>

  <!-- The JNDI lookup for the target destination -->
  <attribute name="TargetDestinationLookup">/queue/B</attribute>

  <!-- The username to use for the source connection
  <attribute name="SourceUsername">bob</attribute>
  -->

  <!-- The password to use for the source connection
  <attribute name="SourcePassword">BobSecur3</attribute>
  -->

  <!-- The username to use for the target connection
  <attribute name="TargetUsername">mary</attribute>
  -->

  <!-- The password to use for the target connection
  <attribute name="TargetPassword">MaryS3cur3</attribute>
  -->

  <!-- Optional: The Quality Of Service mode to use, one of:
    QOS_AT_MOST_ONCE = 0;
    QOS_DUPLICATES_OK = 1;
    QOS_ONCE_AND_ONLY_ONCE = 2;
  -->
  <attribute name="QualityOfServiceMode">0</attribute>

  <!-- JMS selector to use for consuming messages from the source
  <attribute name="Selector">specify jms selector here</attribute>
  -->

  <!-- The maximum number of messages to consume from the source
       before sending to the target
  -->
  <attribute name="MaxBatchSize">5</attribute>

  <!-- The maximum time to wait (in ms) before sending a batch to the
       target even if MaxBatchSize is not exceeded. -1 means wait
forever
  -->
  <attribute name="MaxBatchTime">-1</attribute>

  <!-- If consuming from a durable subscription this is the subscription
       name
       <attribute name="SubName">mysub</attribute>
  -->

  <!-- If consuming from a durable subscription this is the client ID to
       use
       <attribute name="ClientID">myClientID</attribute>
  -->
```

```
  <!-- The number of ms to wait between connection retrues in the event
       connections to source or target fail
  -->
  <attribute name="FailureRetryInterval">5000</attribute>

  <!-- The maximum number of connection retries to make in case of
failure,
       before giving up -1 means try forever
  -->
  <attribute name="MaxRetries">-1</attribute>

  <!-- If true then the message ID of the message before bridging will
be
       added as a header to the message so it is available to the
       receiver. Can then be sent as correlation ID to correlate in a
       distributed request-response
  -->
  <attribute name="AddMessageIDInHeader">false</attribute>

    </mbean>
```

メッセージブリッジ設定の属性

SourceProviderLoader, TargetProvider Loader

JMSProviderLoader 管理 Bean は、ブリッジによってソースの接続ファクトリとソースのデスティネーションの検索に使用されます。デフォルトでは、JBoss Enterprise Application Platform には **JMSProviderLoader** が 1 つ同梱され、**$JBOSS_HOME/server/$PROFILE/deploy/messaging/jms-ds.xml** ファイルでデプロイされ、デフォルトのローカル **JMSProviderLoader** として動作します。クラスター化した設定では、**hajndi-jms-ds.xml** が同じ役割を果たします。

ソースまたは目的のデスティネーションが異なるサーバーにある場合、または JBoss JMS Provider 以外に対応する場合、リモート JMS Provider のデスティネーションとコンタクトするためにブリッジが使用できる別の **JMSProviderLoader** 管理 Bean インスタンスをデプロイすることができます。

JBoss Messaging 以外のリモートのソースまたは目的のデスティネーションで **QOS_ONCE_AND_ONLY_ONCE** の配信を使用するためには、リモート JMS Provider は、サーバーからリモートで動作する完全機能の JMS XA リソース実装を提供する必要があります。

SourceDestinationLookup

/queue/mySourceQueue など **SourceProviderLoader** によるソースのデスティネーションの完全な JNDI 検索です。

TargetDestinationLookup

/topic/myTargetTopic など **TargetProviderLocator** による目的のデスティネーションの完全な JNDI 検索です。

SourceUsername

ソース接続の作成時に使用するユーザー名を指定するオプションの属性です。

SourcePassword

ソース接続の作成時に使用するパスワードを指定するオプションの属性です。

TargetUsername

目的の接続の作成時に使用するユーザー名を指定するオプションの属性です。

TargetPassword

目的の接続の作成時に使用するパスワードを指定するオプションの属性です。

QualityOfServiceMode

目的の **サービスの質** のモードを表す整数です。可能性のある値は次のとおりです。

- **0** は **QOS_AT_MOST_ONCE** を表します。
- **1** は **QOS_DUPLICATES_OK** を表します。
- **2** は **QOS_ONCE_AND_ONLY_ONCE** を表します。

これらモードの詳細な説明については 「メッセージブリッジの概要」 を参照してください。

Selector

ソースのデスティネーションからメッセージをコンシュームする時に、JMS セレクター式を与えることができるオプションの属性です。セレクター式に一致するメッセージだけがソースから目的のデスティネーションまでブリッジされます。セレクター式は、http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html で指定されている JMS セレクターの構文に従う必要があります。

最適なパフォーマンスを得るには、ソーストピックサブスクリプションのセレクターをソースキューのコンシューマーに適用します。

MaxBatchSize

目的のデスティネーションにメッセージのバッチを送信するまでにソースのデスティネーションからコンシュームする最大メッセージ数を指定します。値は **1** かそれ以上である必要があります。

MaxBatchTime

MaxBatchSize に到達していなくても、メッセージのバッチを目的に送信するまでに待機する最長時間（ミリ秒単位）を指定します。値は **-1** (永久に待機)、または時間を指定する **1** かそれ以上である必要があります。

SubName

ソースのデスティネーショントピックからコンシュームする持続性のあるサブスクリプションの名前を表します。

ClientID

ソースのデスティネーショントピックからコンシュームする持続性のあるサブスクリプションの作成または検索時に使用する JMS クライアント ID を表します。

FailureRetryInterval

障害が検出されてからソースまたは目的のサーバーに接続の再作成を試行する間の待機時間（ミリ秒単位）です。

MaxRetries

障害が検出されてからソースまたは目的のサーバーに接続の再作成を試行する回数です。この後、ブリッジは接続の再作成の試行を停止します。**-1** の値は、ブリッジが再接続の試行を永久に続けるという意味です。

AddMessageIDInHeader

true に設定すると、オリジナルのメッセージ ID はデスティネーションに送信されたメッセージの **JBossMessage.JBOSS_MESSAGING_BRIDGE_MESSAGE_ID_LIST** ヘッダに追加されます。メッセージが複数回ブリッジされると、それぞれのメッセージ ID はヘッダに追加されます。これにより、分散型要求応答パターンが使用できるようになります。

第9章 JBOSS MESSAGING 順序グループの有効化

このセクションでは、厳密なメッセージ順序を実現するための JBoss Messaging の順序グループの使用方法について説明します。

メッセージ順序グループは、厳密なメッセージ順序の JBoss Messaging 実装です。順序グループの機能が有効な場合、メッセージの優先度はメッセージが配信された順序に対し影響力がなくなります。特定の順序グループ内のメッセージは、目的のキューに到着した順序と全く同じ順序で配信されます (FIFO)。

前のメッセージの配信が完了してからでないと、順序グループにある次のメッセージが配信されません。正常なメッセージ配信を表すシグナルを送信するには、受信確認メカニズムを利用します (「受領確認メカニズム」 を参照のこと)。

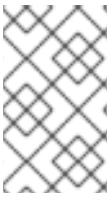
トラザクション的な受領

トランザクション的にメッセージを受け取る場合、現在のメッセージの受領確認も含むトランザクションがコミットされるまで次のメッセージは配信されません。トランザクションがロールバックすると、メッセージは取り消され、JMS サーバーに戻されて次の配信に備えます。

9.1. 受領確認メカニズム

順序グループの一部を形成するメッセージは、1 通ずつ配信されます。前のメッセージの配信が完了するまで、次のメッセージは配信されません。メッセージ配信の完了は、確認モードの設定により様々な方法で伝えられます。

- **`CLIENT_ACKNOWLEDGE`** モードでは、**`Message.acknowledge()`** メソッドが完了状態を伝えることになります。
- **`AUTO_ACKNOWLEDGE`** と **`DUPS_OK_ACKNOWLEDGE`** モードでは、次のいずれかでメッセージの完了が伝えられます。
  - **`MessageConsumer.receive()`** メソッドのいずれか 1 つから成功が返される場合、または
  - **`MessageListener()`** の **`onMessage()`** 呼び出しから成功が返される場合



注記

メッセージコンシューマーが閉じられると、閉じられた時点で処理されたメッセージは完了とみなされます。コンシューマーが閉じられる前に **`*_ACKNOWLEGE`** が呼び出されたかどうかは関係ありません。

9.2. メッセージ順序グループを有効にする方法

接続ファクトリか、プロデューサーのいずれかで順序グループを有効化できます。

- 接続ファクトリで順序グループを定義した場合、接続ファクトリにあるプロデューサーはすべて、同じ順序グループを利用します (「プロデューサー上の順序グループを有効化」 参照)。
- プロデューサーで順序グループを定義すると (接続ファクトリ上でプロデューサーは定義)、プロデューサーは定義した順序グループを利用します。プロデューサーの接続ファクトリが順序グループも定義している場合は、プロデューサーの順序グループがその設定をオーバーライドし、独自の順序グループを利用します (「接続ファクトリで順序グループを有効化」 参照)。

9.2.1. プロデューサー上の順序グループを有効化

プロデューサーで順序グループ機能を有効にするには、以下を行います。

1. セッションに **JBossMessageProducer** を追加します。

```
JBossMessageProducer producer=
(JBossMessageProducer)session.createProducer(queue);
```

2. enableOrderingGroup メソッドを追加し順序グループを設定します。

```
producer.enableOrderingGroup(String ogrpName) throws JMSException
```

このメソッドは順序グループを作成します。つまり、このメソッドが呼び出されると、JBossMessageProducer は順序グループの代わりにメッセージを送信します。**null** パラメーターを渡すと順序グループは自動的に生成されます。このメソッドへ新たに呼び出しを行うと、以前の呼び出しをオーバーライドします。

3. オプションで、**disableOrderingGroup()** を追加し順序グループを無効にすることができます。

```
public void disableOrderingGroup() throws JMSException
```

このメッセージが呼び出されると、JBossMessageProducer は順序グループのメッセージ送信を停止して、デフォルトの動作を再開します。

OrderingGroupExample.java クラスの **$EAPHOME/doc/examples/jboss-messaging-examples/ordering-group/** にサンプルが用意されています (**src/org/jboss/example/jms/ordering/OrderingGroupExample.java**)。このサンプルは **jboss-eap-docs** の一部として提供されています。

9.2.2. 接続ファクトリで順序グループを有効化

接続ファクトリで順序グループ機能を有効にするには、以下の属性をファクトリサービスの設定ファイルに追加します。

EnableOrderingGroup

true に設定されると、順序グループの機能が有効になります (デフォルト値は **false** です)。

DefaultOrderingGroupName

メッセージ順序グループのデフォルト名を設定します。この属性がないとグループ名は自動的に生成されます。

注記

接続ファクトリで順序グループの機能を有効にするように設定した時点で、接続ファクトリから作成されるプロデューサから送信されるメッセージはすべて順序グループのメッセージとなります。

次のファクトリサービスの設定ファイルのサンプルでは、順序グループの機能を有効にする方法を示しています。

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory";
  name="jboss.messaging.connectionfactory:service=ConnectionFactory";
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>
    jboss.messaging:service=PostOffice
  </depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/XAConnectionFactory</binding>
    <binding>java:/MyConnectionFactory</binding>
    <binding>java:/XAConnectionFactory</binding>
    </bindings>
  </attribute>

  <!-- The two OrderingGroup properties -->
  <attribute name="EnableOrderingGroup">true</attribute>
  <attribute name="DefaultOrderingGroupName">MyOrderingGroup</attribute>
</mbean>
```

設定に変更を加えて順序グループの機能を有効にする利点は、コードを変更する必要がなくメッセージ順序の機能を容易に利用できるという点です。

9.3. 注意点と制限

順序グループの機能に関して次の点に注意してください。

- 順序グループの機能ではキューを使用する必要があります。この機能はトピックでは動作しません。
- 順序グループの機能は、メッセージセレクタおよびスケジュールされた配信とは併用しないでください。
- オリジナルのメッセージがデッドメッセージまたは期限切れのメッセージの場合、メッセージは完了したとみなされ次のメッセージが配信可能となります。デッドメッセージは **DLQ** に移動し、期限切れのメッセージは **ExpiryQueue** に移動します。
- **ConnectionConsumer** を使用すると、メッセージの順序は監視されるようになります。ただし、**ConnectionConsumer** は次のメッセージを受け取るセッションを制御するわけではありません。
- 順序グループ機能はクラスター化されたキューでは機能しません。しかし、ユーザーが HASingleton としてクラスター化キューをデプロイすると（つまり、クラスター化キューで一度に有効なキューが 1 つしかない）、順序グループは予想通りに機能します。

付録A 改訂履歴

改訂 5.1.2-2.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

改訂 5.1.2-2 **2012-07-18** **Anthony Towns**
Rebuild for Publican 3.0

改訂 5.1.2-100 **Thu 8 December 2011** **Russell Dickenson**
JBoss Enterprise Application Platform 5.1.2 GA 向けの変更が含まれます。このガイドの内容の変更については、『 リリースノート 5.1.2』を参照してください。