Red Hat AMQ 2021.Q1

AMQ OpenWire JMS クライアントの使用

AMQ Clients 2.9 向け

Last Updated: 2023-02-27

Red Hat AMQ 2021.Q1 AMQ OpenWire JMS クライアントの使用

AMQ Clients 2.9 向け

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

法律上の通知

概要

本ガイドでは、クライアントをインストールして設定する方法、実例を実行し、他の AMQ コンポーネントでクライアントを使用する方法を説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社 の CTO、Chris Wright のメッセージを参照してください。

第1章 概要

AMQ OpenWire JMS は、OpenWire メッセージを送受信するメッセージングアプリケーションで使用する Java Message Service (JMS) 1.1 クライアントです。

AMQ OpenWire JMS は AMQ Clients (複数の言語やプラットフォームをサポートするメッセージングライブラリースイート) に含まれています。クライアントの概要は、AMQ Clients の概要 を参照してください。本リリースに関する詳細は、『 AMQ Clients 2.9 リリースノート』 を参照してください。

AMQ JMS は、Apache Qpid からの JMS 実装に基づいています。JMS API の詳細は、「 JMS API reference」および「JMS tutorial」を参照してください。

1.1. 主な特長

- JMS 1.1 との互換性
- セキュアな通信用の SSL/TLS
- 自動再接続およびフェイルオーバー
- 分散トランザクション (XA)
- Pure-Java 実装

1.2. サポートされる標準およびプロトコル

AMQ OpenWire JMS は、以下の業界標準およびネットワークプロトコルをサポートします。

- Java Message Service API のバージョン 1.1
- IPv6 での最新の TCP

1.3. サポートされる構成

AMQ OpenWire JMS は、以下に挙げる OS および言語のバージョンをサポートします。詳細は、「Red Hat AMQ 7 Supported Configurations」 を参照してください。

- 以下の JDK を備えた Red Hat Enterprise Linux 7 および 8:
  - OpenJDK 8 および 11
  - Oracle JDK 8
  - IBM JDK 8
- IBM JDK 8 と IBM AIX 7.1
- Microsoft Windows 10 Pro と Oracle JDK 8
- Microsoft Windows Server 2012 R2 および 2016 with Oracle JDK 8
- Oracle JDK 8 での Oracle Solaris 10 および 11

AMQ OpenWire JMS は、以下の AMQ コンポーネントおよびバージョンと組み合わせてサポートされます。

- AMQ Broker の最新バージョン
- A-MQ 6 バージョン 6.2.1 以降

1.4. 用語および概念

本セクションでは、コア API エンティティーを紹介し、コア API が連携する方法を説明します。

表1.1 API の用語

エンティティー	説明
ConnectionFactory	接続を作成するエントリーポイント。
接続	ネットワーク上の 2 つのピア間の通信チャネル。これにはセッションが含まれます。
Session	メッセージを生成および消費するためのコンテキスト。メッセージプロデューサーとコンシューマーが含まれます。
MessageProducer	メッセージを宛先に送信するためのチャネル。ターゲットの宛先があります。
MessageConsumer	宛先からメッセージを受信するためのチャネル。ソースの宛先があります。
宛先	メッセージの名前付きの場所 (キューまたはトピックのいずれか)。
Queue	メッセージの保存されたシーケンス。
トピック	マルチキャスト配布用のメッセージの保存されたシーケンス。
メッセージ	情報のアプリケーション固有の部分。

AMQ OpenWire JMS は **メッセージ** を送受信します。メッセージは、**メッセージプロデューサー** と **コンシューマー** を使用して接続されたピア間で転送されます。プロデューサーとコンシューマーは **セッション** 上で確立されます。セッションは**接続**上で確立されます。接続は **接続ファクトリー** によって作成されます。

送信ピアは、メッセージ送信用のプロデューサーを作成します。プロデューサーには、リモートピアでターゲットキューまたはトピックを識別する **宛先** があります。受信ピアは、メッセージ受信用のコンシューマーを作成します。プロデューサーと同様に、コンシューマーにはリモートピアでソースキューまたはトピックを識別する宛先があります。

宛先は、**キュー** または **トピック** のいずれかです。JMS では、キューとトピックはメッセージを保持する名前付きブローカーエンティティーのクライアント側表現です。

キューは、ポイントツーポイントセマンティクスを実装します。各メッセージは 1 つのコンシューマーによってのみ認識され、メッセージは読み取り後にキューから削除されます。トピックはパブリッシュ/サブスクライブセマンティクスを実装します。各メッセージは複数のコンシューマーによって認識され、メッセージは読み取り後も他のコンシューマーで利用できるままになります。

詳細は、JMS tutorial を参照してください。

1.5. 本書の表記慣例

sudo コマンド
本書では、root 権限を必要とするすべてのコマンドに対して **sudo** が使用されています。すべての変更がシステム全体に影響する可能性があるため、**sudo** を使用する場合は注意が必要です。**sudo** の詳細は、sudo コマンドの使用を参照してください。

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

変数テキスト
本書では、変数を含むコードブロックが紹介されていますが、これは、お客様の環境に固有の値に置き換える必要があります。可変テキストは矢印の中括弧で囲まれ、斜体の等幅フォントとしてスタイル設定されます。たとえば、以下のコマンドでは **<project-dir>** は実際の環境の値に置き換えます。

```
$ cd <project-dir>
```

第2章 インストールシステム

本章では、環境に AMQ OpenWire JMS をインストールする手順を説明します。

2.1. 前提条件

- AMQ リリースファイルおよびリポジトリーにアクセスするには、サブスクリプション が必要です。
- AMQ OpenWire JMS でプログラムを構築するには、Apache Maven をインストールする必要があります。
- AMQ OpenWire JMS を使用するには、Java をインストールする必要があります。

2.2. 「RED HAT MAVEN リポジトリーの使用」

Red Hat Maven リポジトリーからクライアントライブラリーをダウンロードするように Maven 環境を設定します。

手順

1. Red Hat リポジトリーを Maven 設定または POM ファイルに追加します。設定ファイルの例は、「オンラインリポジトリーの使用」 を参照してください。

```
<repository>
  <id>red-hat-ga</id>
  <url>https://maven.repository.redhat.com/ga</url>
</repository>
```

2. ライブラリーの依存関係を POM ファイルに追加します。

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-client</artifactId>
  <version>5.11.0.redhat-630416</version>
</dependency>
```

これで、Maven プロジェクトでクライアントを使用できるようになります。

2.3. ローカル MAVEN リポジトリーのインストール

オンラインリポジトリーの代わりに、AMQ OpenWire JMS をファイルベースの Maven リポジトリーとしてローカルファイルシステムにインストールできます。

手順

1. サブスクリプションを使用して、**AMQ Broker 7.9.0 Maven リポジトリー**の .zip ファイルをダウンロードします。
2. 選択したディレクトリーにファイルの内容を抽出します。
   Linux または UNIX では、**unzip** コマンドを使用してファイルの内容を抽出します。

```
$ unzip amq-broker-7.9.0-maven-repository.zip
```

Windows では、.zip ファイルを右クリックして、**Extract All** を選択します。

3. 抽出されたインストールディレクトリー内の **maven-repository** ディレクトリーにあるリポジトリーを使用するように Maven を設定します。詳細は、「「ローカルリポジトリーの使用」」を参照してください。

2.4. 「サンプルのインストール」

手順

1. サブスクリプションを使用して **AMQ Broker 7.9.0**.zip ファイルをダウンロードします。
2. 選択したディレクトリーにファイルの内容を抽出します。
Linux または UNIX では、**unzip** コマンドを使用してファイルの内容を抽出します。

```
$ unzip amq-broker-7.9.0.zip
```

Windows では、.zip ファイルを右クリックして、**Extract All** を選択します。

.zip ファイルの内容を抽出すると、**amq-broker-7.9.0** という名前のディレクトリーが作成されます。これはインストールの最上位ディレクトリーであり、本書では **<install-dir>** と呼びます。

第3章 はじめに

本章では、環境を設定して簡単なメッセージングプログラムを実行する手順を説明します。

3.1. 前提条件

- 例を作成するには、Red Hat リポジトリー または ローカルリポジトリー を使用するように Maven を設定する必要があります。
- サンプルをインストール する必要があります。
- **localhost** での接続をリッスンするメッセージブローカーが必要です。匿名アクセスを有効にする必要があります。詳細は、ブローカーの開始を参照してください。
- **exampleQueue** という名前のキューが必要です。詳細は、キューの作成 を参照してください。

3.2. 最初のサンプルの実行

この例では、**exampleQueue** という名前のキューにコンシューマーおよびプロデューサーを作成します。テキストメッセージを送信してから受信し、受信したメッセージをコンソールに出力します。

手順

1. **<install-dir>/examples/protocols/openwire/queue** ディレクトリーで以下のコマンドを実行し、Maven を使用してサンプルを構築します。

```
$ mvn clean package dependency:copy-dependencies -DincludeScope=runtime -DskipTests
```

dependency:copy-dependencies を追加すると、依存関係が **target/dependency** ディレクトリーにコピーされます。

2. **java** コマンドを使用して例を実行します。
Linux または UNIX の場合:

```
$ java -cp "target/classes:target/dependency/*"
org.apache.activemq.artemis.jms.example.QueueExample
```

Windows の場合:

```
> java -cp "target\classes;target\dependency\*"
org.apache.activemq.artemis.jms.example.QueueExample
```

Linux で実行すると、以下のような出力になります。

```
$ java -cp "target/classes:target/dependency/*"
org.apache.activemq.artemis.jms.example.QueueExample
Sent message: This is a text message
Received message: This is a text message
```

この例のソースコードは **<install-dir>/examples/protocols/openwire/queue/src** ディレクトリーにあります。追加の例は、**<install-dir>/examples/protocols/openwire** ディレクトリーにあります。

第4章 設定

本章では、AMQ OpenWire JMS 実装を JMS アプリケーションにバインドし、設定オプションを設定するプロセスについて説明します。

JMS は Java Naming Directory Interface (JNDI) を使用して、API 実装およびその他のリソースを登録し、検索します。これにより、特定の実装に固有のコードを作成せずに JMS API にコードを作成できます。

設定オプションは、接続 URI でクエリーパラメーターとして公開されます。

AMQ OpenWire JMS の設定に関する詳細は、ActiveMQ ユーザーガイド を参照してください。

4.1. JNDI 初期コンテキストの設定

JMS アプリケーションは **InitialContextFactory** から取得した JNDI **InitialContext** オブジェクトを使用して、接続ファクトリーなどの JMS オブジェクトを検索します。AMQ OpenWire JMS は、**org.apache.activemq.jndi.ActiveMQInitialContextFactory** クラスで **InitialContextFactory** の実装を提供します。

InitialContextFactory の実装は、**InitialContext** オブジェクトがインスタンス化されると検出されます。

```
javax.naming.Context context = new javax.naming.InitialContext();
```

実装を見つけるには、お使いの環境で JNDI を設定する必要があります。これには、**jndi.properties** ファイルの使用、システムプロパティーの使用、または初期コンテキスト API の使用の 3 つの方法があります。

jndi.properties ファイルの使用

jndi.properties という名前のファイルを作成し、Java クラスパスに配置します。**java.naming.factory.initial** キーでプロパティーを追加します。

例: jndi.properties ファイルを使用した JNDI 初期コンテキストファクトリーの設定

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory
```

Maven ベースのプロジェクトでは、**jndi.properties** ファイルは **<project-dir>/src/main/resources** ディレクトリーに配置されます。

システムプロパティーの使用

java.naming.factory.initial システムプロパティーを設定します。

例: システムプロパティーを使用した JNDI 初期コンテキストファクトリーの設定

```
$ java -Djava.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory ...
```

初期コンテキスト API の使用

JNDI 初期コンテキスト API を使用してプロパティーをプログラム的に設定します。

例: プログラムでの JNDI プロパティーの設定

```
Hashtable<Object, Object> env = new Hashtable<>();
```

```
env.put("java.naming.factory.initial", "org.apache.activemq.jndi.ActiveMQInitialContextFactory");

InitialContext context = new InitialContext(env);
```

同じ API を使用して、接続ファクトリー、キュー、およびトピックの JNDI プロパティーを設定できます。

4.2. 接続ファクトリーの設定

JMS 接続ファクトリーは、接続を作成するためのエントリーポイントです。これは、アプリケーション固有の設定をエンコードする接続 URI を使用します。

ファクトリー名と接続 URI を設定するには、以下の形式でプロパティーを作成します。この設定は、**jndi.properties** ファイルに保存するか、対応するシステムプロパティーを設定できます。

接続ファクトリーの JNDI プロパティー形式

```
connectionFactory.<lookup-name> = <connection-uri>
```

たとえば、以下のように **app1** という名前のファクトリーを設定します。

例: jndi.properties ファイルでの接続ファクトリーの設定

```
connectionFactory.app1 = tcp://example.net:61616?jms.clientID=backend
```

その後、JNDI コンテキストを使用して、**app1** の名前を使用して設定済みの接続ファクトリーを検索できます。

```
ConnectionFactory factory = (ConnectionFactory) context.lookup("app1");
```

4.3. 接続 URI

コネクションは接続 URI を使用して設定されます。接続 URI は、クエリーパラメーターとして設定されるリモートホスト、ポート、および設定オプションのセットを指定します。利用可能なオプションの詳細は、5章*設定オプション* を参照してください。

接続 URI 形式

```
<scheme>://<host>:<port>[?<option>=<value>[&<option>=<value>...]]
```

暗号化されていない接続のスキームは **tcp** で、SSL/TLS 接続の場合は **ssl** です。

たとえば、以下はポート **61616** でホスト **example.net** に接続する接続 URI で、クライアント ID を **backend** に設定します。

例: 接続 URI

```
tcp://example.net:61616?jms.clientID=backend
```

フェイルオーバー URI
再接続およびフェイルオーバーに使用される URI には複数の接続 URI を含めることができます。URI の形式は、以下のとおりです。

フェイルオーバー URI 形式

```
failover:(<connection-uri>[,<connection-uri>])[?<option>=<value>[&<option>=<value>...]]
```

nested. で始まるトランスポートオプションは、リストの各接続 URI に適用されます。

4.4. キューおよびトピック名の設定

JMS は、JNDI を使用してデプロイメント固有のキューとトピックリソースを検索するオプションを提供します。

JNDI でキューおよびトピック名を設定するには、以下の形式でプロパティーを作成します。この設定を **jndi.properties** ファイルに置くか、対応するシステムプロパティーを設定します。

キューおよびトピックの JNDI プロパティー形式

```
queue.<lookup-name> = <queue-name>
topic.<lookup-name> = <topic-name>
```

たとえば、以下のプロパティーは、2 つのデプロイメント固有のリソースの名前、**jobs** および **notifications** を定義します。

例: jndi.properties ファイルでのキューおよびトピック名の設定

```
queue.jobs = app1/work-items
topic.notifications = app1/updates
```

その後、JNDI 名でリソースを検索できます。

```
Queue queue = (Queue) context.lookup("jobs");
Topic topic = (Topic) context.lookup("notifications");
```

第5章 設定オプション

本章では、AMQ OpenWire JMS で利用可能な設定オプションについて説明します。

JMS 設定オプションは、接続 URI でクエリーパラメーターとして設定されます。詳細は、「「接続 URI」」を参照してください。

5.1. JMS オプション

jms.username

クライアントが接続を認証するために使用するユーザー名。

jms.password

クライアントが接続を認証するために使用するパスワード。

jms.clientID

クライアントが接続に適用するクライアント ID。

jms.closeTimeout

JMS 終了操作のタイムアウト（ミリ秒単位）。デフォルトは 15000 (15 秒) です。

jms.connectResponseTimeout

JMS 接続操作のタイムアウト（ミリ秒単位）。デフォルトは 0 で、タイムアウトなしを意味します。

jms.sendTimeout

JMS 送信操作のタイムアウト（ミリ秒単位）。デフォルトは 0 で、タイムアウトなしを意味します。

jms.checkForDuplicates

有効にすると、重複メッセージを無視します。これは、デフォルトで有効になっています。

jms.disableTimeStampsByDefault

有効な場合は、タイムスタンプのメッセージは使用しないでください。これはデフォルトでは無効にされます。

jms.useAsyncSend

有効にすると、確認応答を待たずにメッセージを送信します。これはデフォルトでは無効にされます。

jms.alwaysSyncSend

有効にすると、送信はすべての配信モードで承認を待ちます。これはデフォルトでは無効にされます。

jms.useCompression

有効にすると、メッセージの本文を圧縮します。これはデフォルトでは無効にされます。

jms.useRetroactiveConsumer

有効にすると、非永続サブスクライバーはサブスクリプションの開始前に公開されたメッセージを受信できます。これはデフォルトでは無効にされます。

Prefetch ポリシーオプション

Prefetch ポリシーは、各 **MessageConsumer** がリモートピアから取得し、ローカルの「prefetch」バッファーに保持するメッセージの数を決定します。

jms.prefetchPolicy.queuePrefetch

キューの事前フェッチするメッセージの数を指定。デフォルトは 1000 です。

jms.prefetchPolicy.queueBrowserPrefetch

キューブラウザーの事前フェッチするメッセージの数を指定。デフォルトは 500 です。

jms.prefetchPolicy.topicPrefetch

非永続トピック用に事前にフェッチするメッセージの数。デフォルトは 32766 です。

jms.prefetchPolicy.durableTopicPrefetch

永続トピック用に事前にフェッチするメッセージの数。デフォルトは 100 です。

jms.prefetchPolicy.all

これは、すべての事前にフェッチされた値を一度に設定するために使用できます。

prefetch の値は、キューの複数のコンシューマーへのメッセージの分散に影響します。値が大きいと、各コンシューマーに一度に送信されるバッチが大きくなる可能性があります。コンシューマーが異なるレートで動作している場合に、より均等にラウンドロビンの分散を実現するには、小さい値を使用します。

再配信ポリシーオプション

再配信ポリシーは、クライアント上で再配信されたメッセージの処理方法を制御します。

jms.redeliveryPolicy.maximumRedeliveries

メッセージがデッドレターキューに送信される前に再配信を試行する回数。デフォルトは 6 で、-1 は制限がないことを意味します。

jms.redeliveryPolicy.redeliveryDelay

再配信試行の間隔 （ミリ秒単位）。**initialRedeliveryDelay** が 0 の場合に使用されます。デフォルトは 1000 (1 秒) です。

jms.redeliveryPolicy.initialRedeliveryDelay

最初の再配信試行までの時間 （ミリ秒単位）。デフォルトは 1000 (1 秒) です。

jms.redeliveryPolicy.maximumRedeliveryDelay

再配信の試行間隔の最大時間 （ミリ秒単位）。これは、**useExponentialBackOff** が有効な場合に使用されます。デフォルトは 1000 （1 秒） です。-1 は無制限を意味します。

jms.redeliveryPolicy.useExponentialBackOff

有効にすると、後続の試行ごとに再配信の遅延を増やします。これはデフォルトでは無効にされます。

jms.redeliveryPolicy.backOffMultiplier

再配信の遅延を増やす乗数。デフォルトは 5 です。

jms.redeliveryPolicy.useCollisionAvoidance

有効にすると、再配信の遅延を若干上またはスケールダウンして、競合を回避します。これはデフォルトでは無効にされます。

jms.redeliveryPolicy.collisionAvoidanceFactor

再配信の遅延を増やす乗数。デフォルトは 0.15 です。

nonBlockingRedelivery

有効にすると、インラインブロックを回避するために、再配信を順序付けます。これはデフォルトでは無効にされます。

5.2. TCP オプション

closeAsync

有効にすると、別のスレッドでソケットを閉じます。これは、デフォルトで有効になっています。

connectionTimeout

TCP 接続操作のタイムアウト （ミリ秒単位）。デフォルトは 30000 (30 秒) です。0 はタイムアウトなしを意味します。

dynamicManagement

有効にすると、トランスポートロガーの JMX 管理が許可されます。これはデフォルトでは無効にされます。

ioBufferSize

I/O バッファーサイズ (バイト単位)。デフォルトは 8192 (8 KiB) です。

jmxPort

JMX 管理のポート。デフォルトは 1099 です。

keepAlive

有効な場合は TCP keepalive を使用します。これは、**KeepAliveInfo** メッセージに基づく keepalive メカニズムとは異なります。これはデフォルトでは無効にされます。

logWriterName

org.apache.activemq.transport.LogWriter 実装の名前。名前とクラスのマッピングは **resources/META-INF/services/org/apache/activemq/transport/logwriters** ディレクトリーに保存されます。デフォルトは **default** です。

soLinger

socket linger オプション。デフォルトは 0 です。

soTimeout

ソケット読み取り操作のタイムアウト （ミリ秒単位）。デフォルトは 0 で、タイムアウトなしを意味します。

soWriteTimeout

ソケット書き込み操作のタイムアウト （ミリ秒単位）。デフォルトは 0 で、タイムアウトなしを意味します。

startLogging

有効にされ、**trace** オプションも有効になっている場合、ログトランスポートの起動イベントが有効になります。これは、デフォルトで有効になっています。

tcpNoDelay

有効な場合、TCP 送信の遅延やバッファーを行いません。これはデフォルトでは無効にされます。

threadName

設定されている場合は、トランスポートスレッドに割り当てられた名前。リモートアドレスが名前に追加されます。これは、デフォルトで未設定になっています。

trace

有効にすると、転送イベントを **log4j.logger.org.apache.activemq.transport.TransportLogger** にログを記録します。これはデフォルトでは無効にされます。

useInactivityMonitor

有効にすると、**KeepAliveInfo** メッセージの送信に失敗する接続がタイムアウトします。これは、デフォルトで有効になっています。

useKeepAlive

有効にすると、**KeepAliveInfo** メッセージを定期的に送信して、接続がタイムアウトしないようにします。これは、デフォルトで有効になっています。

useLocalHost

有効にすると、現在のホスト名の代わりに **localhost** の名前を使用してローカル接続を行います。これはデフォルトでは無効にされます。

5.3. SSL/TLS オプション

socket.keyStore

SSL/TLS キーストアへのパス。キーストアは相互 SSL/TLS 認証に必要です。設定しないと、**javax.net.ssl.keyStore** システムプロパティーの値が使用されます。

socket.keyStorePassword

SSL/TLS キーストアのパスワード。設定しないと、**javax.net.ssl.keyStorePassword** システムプロパティーの値が使用されます。

socket.keyStoreType

トラストストアタイプの文字列名。デフォルトは **java.security.KeyStore.getDefaultType（）** の値です。

socket.trustStore

SSL/TLS トラストストアへのパス。設定しないと、**javax.net.ssl.trustStore** システムプロパティーの値が使用されます。

socket.trustStorePassword

SSL/TLS トラストストアのパスワード。設定しないと、**javax.net.ssl.trustStorePassword** システムプロパティーの値が使用されます。

socket.trustStoreType

トラストストアタイプの文字列名。デフォルトは **java.security.KeyStore.getDefaultType（）** の値です。

socket.enabledCipherSuites

有効にする暗号スイートのコンマ区切りリスト。未設定の場合は、context-default 暗号が使用されます。

socket.enabledProtocols

有効にする SSL/TLS プロトコルのコンマ区切りリスト。未設定の場合は、JVM デフォルトプロトコルが使用されます。

5.4. OPENWIRE オプション

wireFormat.cacheEnabled

有効にすると、頻繁に使用される値をキャッシュして、過剰なマーシャリングおよび帯域幅の消費を回避します。これは、デフォルトで有効になっています。

wireFormat.cacheSize

キャッシュエントリーの数。キャッシュは接続ごとに設定されます。デフォルトは 1024 です。

wireFormat.maxInactivityDuration

アクティビティーのない接続が停止されたとみなされる最大時間（ミリ秒単位）。デフォルトは 30000 (30 秒) です。

wireFormat.maxInactivityDurationInitalDelay

非アクティブチェックが開始するまでの初期の遅延（ミリ秒単位）。**Inital** は誤字になっていることに注意してください。デフォルトは 10000 (10 秒) です。

wireFormat.maxFrameSize

最大フレームサイズ (バイト単位)。デフォルトは **java.lang.Long.MAX_VALUE** の値です。

wireFormat.sizePrefixDisabled

true に設定すると、パケットにサイズのプレフィックスを付けないでください。デフォルトは false です。

wireFormat.stackTraceEnabled

有効にすると、サーバーの例外からクライアントにスタックトレースを送信します。これは、デフォルトで有効になっています。

wireFormat.tcpNoDelayEnabled

有効な場合は、サーバーに **TCP_NODELAY** をアクティベートするように指示します。これは、デフォルトで有効になっています。

wireFormat.tightEncodingEnabled

有効にすると、ネットワーク上の小さいエンコーディングを最適化します。これにより、CPU の使用率が増加します。これは、デフォルトで有効になっています。

5.5. フェイルオーバーオプション

maxReconnectAttempts

接続が失敗したと報告するまでに許可される再接続試行回数。デフォルトは -1 で、無制限を意味します。0 は再接続を無効にします。

maxReconnectDelay

2 回目以降の再接続試行の最大時間 （ミリ秒単位）。デフォルトは 30000 (30 秒) です。

randomize

有効にすると、フェイルオーバーエンドポイントのいずれかをランダムに選択します。これは、デフォルトで有効になっています。

reconnectDelayExponent

再接続の遅延バックオフを増やす乗数。デフォルトは 2.0 です。

useExponentialBackOff

有効にすると、後続の試行ごとに再接続の遅延を増やします。これは、デフォルトで有効になっています。

timeout

再接続を待機する送信操作のタイムアウト （ミリ秒単位）。デフォルトは -1 で、タイムアウトなしを意味します。

第6章 メッセージ配信

6.1. ストリーミングされた大きなメッセージへの書き込み

大きなメッセージに書き込むには、**BytesMessage.writeBytes（）** メソッドを使用します。以下の例では、ファイルからバイトを読み取り、メッセージに書き込みます。

例: ストリーミングされた大きなメッセージへの書き込み

```
BytesMessage message = session.createBytesMessage();
File inputFile = new File(inputFilePath);
InputStream inputStream = new FileInputStream(inputFile);

int numRead;
byte[] buffer = new byte[1024];

while ((numRead = inputStream.read(buffer, 0, buffer.length)) != -1) {
    message.writeBytes(buffer, 0, numRead);
}
```

6.2. ストリームされた大きなメッセージからの読み取り

大きなメッセージから読み取るには、**BytesMessage.readBytes（）** メソッドを使用します。以下の例では、メッセージからバイトを読み取り、ファイルに書き込みます。

例: ストリームされた大きなメッセージからの読み取り

```
BytesMessage message = (BytesMessage) consumer.receive();
File outputFile = new File(outputFilePath);
OutputStream outputStream = new FileOutputStream(outputFile);

int numRead;
byte buffer[] = new byte[1024];

for (int pos = 0; pos < message.getBodyLength(); pos += buffer.length) {
    numRead = message.readBytes(buffer);
    outputStream.write(buffer, 0, numRead);
}
```

付録A サブスクリプションの使用

AMQ は、ソフトウェアサブスクリプションから提供されます。サブスクリプションを管理するには、Red Hat カスタマーポータルでアカウントにアクセスします。

A.1. アカウントへのアクセス

手順

1. access.redhat.com に移動します。
2. アカウントがない場合は、作成します。
3. アカウントにログインします。

A.2. サブスクリプションのアクティベート

手順

1. access.redhat.com に移動します。
2. **サブスクリプション** に移動します。
3. **Activate a subscription** に移動し、16 桁のアクティベーション番号を入力します。

A.3. リリースファイルのダウンロード

.zip、.tar.gz およびその他のリリースファイルにアクセスするには、カスタマーポータルを使用してダウンロードする関連ファイルを検索します。RPM パッケージまたは Red Hat Maven リポジトリーを使用している場合は、この手順は必要ありません。

手順

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **JBOSS INTEGRATION AND AUTOMATION** カテゴリーの **Red Hat AMQ** エントリーを見つけます。
3. 必要な AMQ 製品を選択します。**Software Downloads** ページが開きます。
4. コンポーネントの **Download** リンクをクリックします。

A.4. パッケージ用システムの登録

この製品の RPM パッケージを Red Hat Enterprise Linux にインストールするには、システムが登録されている必要があります。ダウンロードしたリリースファイルを使用している場合は、この手順は必要ありません。

手順

1. access.redhat.com に移動します。

2. **Registration Assistant** に移動します。
3. ご使用の OS バージョンを選択し、次のページに進みます。
4. システムの端末に一覧表示されたコマンドを使用して、登録を完了します。

システムを登録する方法は、以下のリソースを参照してください。

- Red Hat Enterprise Linux 7 - システム登録およびサブスクリプション管理
- Red Hat Enterprise Linux 8 - システム登録およびサブスクリプション管理

付録B RED HAT MAVEN リポジトリーの追加

このセクションでは、Red Hat が提供する Maven リポジトリーをソフトウェアで使用する方法を説明します。

B.1. オンラインリポジトリーの使用

Red Hat は、Maven ベースのプロジェクトで使用する中央の Maven リポジトリーを維持しています。詳細は、リポジトリーのウェルカムページ を参照してください。

Red Hat リポジトリーを使用するように Maven を設定する方法は 2 つあります。

- Maven 設定にリポジトリーを追加する
- POM ファイルにリポジトリーを追加する

Maven 設定へのリポジトリーの追加

この設定方法は、POM ファイルがリポジトリー設定をオーバーライドせず、含まれているプロファイルが有効になっている限り、ユーザーが所有するすべての Maven プロジェクトに適用されます。

手順

1. Maven の **settings.xml** ファイルを見つけます。これは通常、ユーザーのホームディレクトリーの **.m2** ディレクトリー内にあります。ファイルが存在しない場合は、テキストエディターを使用して作成します。
   Linux または UNIX の場合:

   ```
   /home/<username>/.m2/settings.xml
   ```

   Windows の場合:

   ```
   C:\Users\<username>\.m2\settings.xml
   ```

2. 次の例のように、Red Hat リポジトリーを含む新しいプロファイルを **settings.xml** ファイルの **profiles** 要素に追加します。

   例: Red Hat リポジトリーを含む Maven settings.xml ファイル

   ```
   <settings>
     <profiles>
       <profile>
         <id>red-hat</id>
         <repositories>
           <repository>
             <id>red-hat-ga</id>
             <url>https://maven.repository.redhat.com/ga</url>
           </repository>
         </repositories>
         <pluginRepositories>
           <pluginRepository>
             <id>red-hat-ga</id>
             <url>https://maven.repository.redhat.com/ga</url>
             <releases>
               <enabled>true</enabled>
   ```

```
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
 </profiles>
 <activeProfiles>
  <activeProfile>red-hat</activeProfile>
 </activeProfiles>
</settings>
```

Maven 設定の詳細は、Maven 設定リファレンス を参照してください。

POM ファイルへのリポジトリーの追加
プロジェクトで直接リポジトリーを設定するには、次の例のように、POM ファイルの **repositories** 要素に新しいエントリーを追加します。

例: Red Hat リポジトリーを含む Maven pom.xml ファイル

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>example-app</artifactId>
  <version>1.0.0</version>

  <repositories>
    <repository>
      <id>red-hat-ga</id>
      <url>https://maven.repository.redhat.com/ga</url>
    </repository>
  </repositories>
</project>
```

POM ファイル設定の詳細は、Maven POM リファレンス を参照してください。

B.2. ローカルリポジトリーの使用

Red Hat は、そのコンポーネントの一部にファイルベースの Maven リポジトリーを提供します。これらは、ローカルファイルシステムに抽出できるダウンロード可能なアーカイブとして提供されます。

ローカルに抽出されたリポジトリーを使用するように Maven を設定するには、Maven 設定または POM ファイルに次の XML を適用します。

```
<repository>
  <id>red-hat-local</id>
  <url>${repository-url}</url>
</repository>
```

${repository-url} は、抽出されたリポジトリーのローカルファイルシステムパスを含むファイル URL である必要があります。

表B.1 ローカル Maven リポジトリーの URL の例

オペレーティングシステム	ファイルシステムパス	URL
Linux または UNIX	**/home/alice/maven-repository**	**file:/home/alice/maven-repository**
Windows	**C:\repos\red-hat**	**file:C:\repos\red-hat**

付録C 例で AMQ ブローカーの使用

AMQ OpenWire JMS の例では、名前が **examples**というキューが含まれる実行中のメッセージブローカーが必要です。以下の手順に従って、ブローカーをインストールして起動し、キューを定義します。

C.1. ブローカーのインストール

『 **Getting Started with AMQ Broker**』の手順に従って、ブローカーをインストールし、ブローカーインスタンスを作成 します。匿名アクセスを有効にします。

以下の手順では、ブローカーインスタンスの場所を **<broker-instance-dir>** と呼びます。

C.2. ブローカーの起動

手順

1. **artemis run** コマンドを使用してブローカーを起動します。

```
$ <broker-instance-dir>/bin/artemis run
```

2. 起動時にログに記録された重大なエラーがないか、コンソールの出力を確認してください。ブローカーでは、準備が整うと **Server is now live** とログが記録されます。

```
$ example-broker/bin/artemis run

        __  __  ____    ____          _
      /\   |  \/  |/ __ \  |  _ \        | |
     /  \  | \  / | |  | | | |_) |_ __ ___ | | _____ _ __
    / /\ \ | |\/| | |  | | |  _ <| '__/ _ \| |/ / _ \ '__|
   / ____ \| |  | | |__| | | |_) | | | (_) |   <  __/ |
  /_/    \_\_|  |_|\____/  |____/|_|  \___/|_|\_\___|_|

   Red Hat AMQ <version>

2020-06-03 12:12:11,807 INFO  [org.apache.activemq.artemis.integration.bootstrap]
AMQ101000: Starting ActiveMQ Artemis Server
...
2020-06-03 12:12:12,336 INFO  [org.apache.activemq.artemis.core.server] AMQ221007:
Server is now live
...
```

C.3. キューの作成

新しいターミナルで、**artemis queue** コマンドを使用して **exampleQueue** という名前のキューを作成します。

```
$ <broker-instance-dir>/bin/artemis queue create --name exampleQueue --address exampleQueue
--auto-create-address --anycast
```

プロンプトで質問に Yes または No で回答するように求められます。すべての質問に **N** (いいえ) と回答します。

キューが作成されると、ブローカーはサンプルプログラムで使用できるようになります。

C.4. ブローカーの停止

サンプルの実行が終了したら、**artemis stop** コマンドを使用してブローカーを停止します。

```
$ <broker-instance-dir>/bin/artemis stop
```

改訂日時： 2022-09-08 16:29:54 +1000