Red Hat AMQ 2021.Q3

AMQ JMS Pool ライブラリーの使用

AMQ Clients 2.10 向け

Last Updated: 2021-09-02

Red Hat AMQ 2021.Q3 AMQ JMS Pool ライブラリーの使用

AMQ Clients 2.10 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

このガイドでは、ライブラリーをインストールして設定する方法、実践的な例を実行する方法、およびその他の 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。これは大規模な取り組みであるため、これらの変更は今後の複数のリリースで段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージをご覧ください。

第1章 概要

AMQ JMS Pool は、JMS 接続、セッション、およびメッセージプロデューサーのキャッシュを提供するライブラリーです。これにより、JMS API で定義される標準のライフサイクル以外の接続リソースを再利用できます。

AMQ JMS Pool は、選択した JMS プロバイダーの **ConnectionFactory** をラップする標準的な JMS **ConnectionFactory** インスタンスとして動作し、JMS プールの構成に基づいてそのプロバイダーからの **Connection** オブジェクトの寿命を管理します。これは、プール内の **createConnection()** メソッドへの呼び出し間で 1 つ以上の接続を共有するように設定できます。

AMQ JMS Pool は、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である AMQ Clients の一部です。クライアントの概要は、「AMQ Clients Overview」を参照してください。本リリースに関する詳細は、『AMQ Clients 2.10 Release Notes』を参照してください。

AMQ JMS Pool は Pooled JMS メッセージングライブラリーに基づいています。

1.1. 主な特長

- JMS 1.1 および 2.0 との互換
- 自動再接続
- 設定可能な接続およびセッションプールサイズ

1.2. サポート対象の標準およびプロトコル

AMQ JMS Pool は、Java Message Service API のバージョン 2.0 をサポートします。

1.3. サポートされる構成

AMQ JMS Pool でサポートされている設定については、Red Hat カスタマーポータルの「Red Hat AMQ 7 でサポートされる構成」を参照してください。

1.4. 本書の表記慣例

sudo コマンド
本書では、root 権限を必要とするコマンドには **sudo** が使用されています。何らかの変更がシステム全体に影響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
本書には、実際の環境に固有の値に置き換える必要がある変数を含むコードブロックが含まれています。変数テキストは中括弧で囲まれ、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、以下の例では、**<project-dir>** を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

本章では、環境に AMQ JMS Pool 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 JMS Pool 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 JMS Pool 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。設定ファイルの例については、「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. ライブラリー依存関係を POM ファイルに追加します。

```
<dependency>
  <groupId>org.messaginghub</groupId>
  <artifactId>pooled-jms</artifactId>
  <version>2.0.0.redhat-00001</version>
</dependency>
```

これで、クライアントが Maven プロジェクトで利用できるようになりました。

2.3. 「INSTALLING A LOCAL MAVEN REPOSITORY」

オンラインリポジトリーの代わりに、ファイルベースの Maven リポジトリーとして AMQ JMS Pool を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 Clients 2.10.0 JMS Pool Maven リポジトリー**の .zip ファイルをダウンロードします。
2. ファイルの内容を、選択したディレクトリーに展開します。
   Linux または UNIX の場合は、**unzip** コマンドを使用してファイルの内容を展開します。

```
$ unzip amq-clients-2.10.0-jms-pool-maven-repository.zip
```

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

3. 展開し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. **git clone** コマンドを使用して、ソースリポジトリーを **pooled-jms** という名前のローカルディレクトリーにクローンします。

```
$ git clone https://github.com/messaginghub/pooled-jms.git pooled-jms
```

2. **pooled-jms** ディレクトリーに移動し、**git checkout** コマンドを使用して **2.0.0** ブランチに切り替えます。

```
$ cd pooled-jms
$ git checkout 2.0.0
```

作成されるローカルディレクトリーは、本書全体で **<source-dir>** と呼ばれます。

第3章 スタートガイド

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

- サンプルをビルドするには、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。
- **queue** 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. HELLO WORLD の実行

Hello World の例では、文字列「Hello World」の各文字に対して **createConnection()** を呼び出し、一度に転送します。AMQ JMS Pool が使用されているため、各呼び出しは同じ基礎となる JMS **Connection** オブジェクトを再利用します。

手順

1. **<source-dir>/pooled-jms-examples** 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サンプルを実行します。
Linux または UNIX の場合:

```
$ java -cp "target/classes:target/dependency/*" org.messaginghub.jms.example.HelloWorld
```

Windows の場合:

```
> java -cp "target\classes;target\dependency\*" org.messaginghub.jms.example.HelloWorld
```

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.messaginghub.jms.example.HelloWorld
2018-05-17 11:04:23,393 [main           ] - INFO  JmsPoolConnectionFactory       - Provided ConnectionFactory is JMS 2.0+ capable.
2018-05-17 11:04:23,715 [localhost:5672]] - INFO  SaslMechanismFinder           - Best match for SASL auth was: SASL-ANONYMOUS
2018-05-17 11:04:23,739 [localhost:5672]] - INFO  JmsConnection                 - Connection ID:104dfd29-d18d-4bf5-aab9-a53660f58633:1 connected to remote Broker: amqp://localhost:5672
Hello World
```

この例のソースコードは **<source-dir>/pooled-jms-examples/src/main/java** ディレクトリーにあります。JNDI およびロギング設定は **<source-dir>/pooled-jms-examples/src/main/resources** ディレクトリーにあります。

第4章 設定

AMQ JMS Pool **ConnectionFactory** 実装は、プールの動作と、管理する JMS リソースを制御する複数の設定オプションを公開します。

設定オプションは、**JmsPoolConnectionFactory** オブジェクトの **set** メソッドとして公開されます。たとえば、maxConnections オプションは **setMaxConnections(int)** メソッドを使用して設定されます。

4.1. 接続オプション

これらのオプションは、JMS プールがプール内の接続を作成し、管理する方法に影響します。

プールされた **ConnectionFactory** は、接続の作成に使用されたユーザーとパスワードの組み合わせごとに接続のプールを作成し、さらにユーザー名やパスワードのないものについては別のプールを作成します。より多くの接続がプールに細分化する必要がある場合は、個別のプールインスタンスを明示的に作成する必要があります。

maxConnections
: 1 つのプールの最大接続数。デフォルトは 1 です。

connectionIdleTimeout
: 現在貸し出されていない接続をプールから削除できるようになるまでのミリ秒単位の時間。デフォルトは 30 秒です。値 0 を指定すると、タイムアウトが無効になります。

connectionCheckInterval
: 期限切れ接続の定期的なチェックの間隔 (ミリ秒単位)。デフォルトは 0 で、チェックが無効になっていることを意味します。

useProviderJMSContext
: これを有効にすると、基礎となる JMS プロバイダーの **JMSContext** クラスを使用します。これはデフォルトで無効にされます。
通常、プールは独自の汎用 **JMSContext** 実装を使用して、プロバイダー実装ではなく、プールから接続をラップします。汎用の実装には、プロバイダーの実装が制限されない可能性があります。ただし、有効にすると、**JMSContext** API からの接続はプールによって管理されません。

4.2. セッションオプション

これらのオプションは、プールされた接続ファクトリーから作成されるセッションの動作に影響します。

maxSessionsPerConnection
: 各接続のセッションの最大数。デフォルトは 500 です。負の値を指定すると制限が削除されます。制限を超えた場合、**createSession()** は設定に応じて例外をブロックするか、または例外を出力します。

blockIfSessionPoolIsFull
: 有効な場合は、プールでセッションが利用可能になるまで **createSession()** をブロックします。これはデフォルトで有効になっています。
無効な場合は、**createSession()** に対する呼び出しは、セッションが利用できない場合に **IllegalStateException** を出力します。

blockIfSessionPoolIsFullTimeout

createSession() へのブロックされた呼び出しが **IllegalStateException** を出力するまでの時間 (ミリ秒単位)。デフォルトは -1 で、呼び出しが永久にブロックされることを意味します。

useAnonymousProducers

これを有効にすると、**createProducer()** へのすべての呼び出しに単一の匿名 JMS **MessageProducer** を使用します。これはデフォルトで有効になっています。
まれに、この動作が望ましくない場合があります。これを無効にすると、**createProducer()** へのすべての呼び出しは新しい **MessageProducer** インスタンスになります。

第5章 例

本章では、サンプルプログラムで AMQ JMS Pool を使用する方法について説明します。

その他の例は、「プールされた JMS の例」を参照してください。

5.1. 前提条件

- サンプルをビルドするには、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。
- サンプルを実行するには、システムに 稼働して設定されているブローカー が必要になります。

5.2. 接続の確立

この例では、新しい接続プールを作成し、接続ファクトリーにバインドし、プールを使用して新しい接続を作成します。

例: 接続を安定させる - **Connect.java**

```
package net.example;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import org.apache.qpid.jms.JmsConnectionFactory;
import org.messaginghub.pooled.jms.JmsPoolConnectionFactory;

public class Connect {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: Connect <connection-uri>");
            System.exit(1);
        }

        String connUri = args[0];

        ConnectionFactory factory = new JmsConnectionFactory(connUri);
        JmsPoolConnectionFactory pool = new JmsPoolConnectionFactory();

        try {
            pool.setConnectionFactory(factory);

            Connection conn = pool.createConnection();

            conn.start();

            try {
                System.out.println("CONNECT: Connected to '" + connUri + "'");
            } finally {
                conn.close();
            }
        } finally {
            pool.stop();
```

```
        }
    }
}
```

5.3. プールの設定

この例では、接続およびセッション設定オプションを指定する方法を示しています。

例: プールの設定 - **ConnectWithConfiguration.java**

```
package net.example;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import org.apache.qpid.jms.JmsConnectionFactory;
import org.messaginghub.pooled.jms.JmsPoolConnectionFactory;

public class ConnectWithConfiguration {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: ConnectWithConfiguration <connection-uri>");
            System.exit(1);
        }

        String connUri = args[0];

        ConnectionFactory factory = new JmsConnectionFactory(connUri);
        JmsPoolConnectionFactory pool = new JmsPoolConnectionFactory();

        try {
            pool.setConnectionFactory(factory);

            // Set the max connections per user to a higher value
            pool.setMaxConnections(5);

            // Create a MessageProducer for each createProducer() call
            pool.setUseAnonymousProducers(false);

            Connection conn = pool.createConnection();

            conn.start();

            try {
                System.out.println("CONNECT: Connected to '" + connUri + "'");
            } finally {
                conn.close();
            }
        } finally {
            pool.stop();
        }
    }
}
```

5.4. サンプルの実行

サンプルプログラムをコンパイルして実行するには、次の手順を使用します。

手順

1. 新しいプロジェクトディレクトリーを作成します。これは、以降の手順で **<project-dir>** と呼ばれます。
2. Java リストのサンプルを以下の場所にコピーします。

```
<project-dir>/src/main/java/net/example/Connect.java
<project-dir>/src/main/java/net/example/ConnectWithConfiguration.java
```

3. テキストエディターを使用して、新しい **<project-dir>/pom.xml** ファイルを作成します。以下の XML を追加します。

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

  <groupId>net.example</groupId>
  <artifactId>example</artifactId>
  <version>1.0.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.messaginghub</groupId>
      <artifactId>pooled-jms</artifactId>
      <version>2.0.0.redhat-00001</version>
    </dependency>
    <dependency>
      <groupId>org.apache.qpid</groupId>
      <artifactId>qpid-jms-client</artifactId>
      <version>${qpid-jms-version}</version>
    </dependency>
  </dependencies>
</project>
```

${qpid-jms-version} を、希望の Qpid JMS バージョンに置き換えます。

4. プロジェクトディレクトリーに移動し、**mvn** コマンドを使用してプログラムをコンパイルします。

```
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。

5. **java** コマンドを使用してプログラムを実行します。
Linux または UNIX の場合:

```
java -cp "target/classes:target/dependency/*" net.example.Connect amqp://localhost
```

Windows の場合:

```
java -cp "target\classes;target\dependency\*" net.example.Connect amqp://localhost
```

これらのサンプルコマンドは、**Connect** の例を実行します。別の例を実行するには、**Connect** を、任意のサンプルのクラス名に置き換えます。

Linux で **Connect** の例を実行すると、以下の出力が表示されます。

```
$ java -cp "target/classes:target/dependency/*" net.example.Connect amqp://localhost
CONNECT: Connected to 'amqp://localhost'
```

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、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. **INTEGRATION AND AUTOMATION** カテゴリーで **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. パッケージを受信するためのシステムの登録

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、お使いのシステムを登録する必要があります。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 リポジトリーの使用

本セクションでは、ソフトウェアで Red Hat が提供する Maven リポジトリーを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 Maven リポジトリーを維持します。詳細は、リポジトリーの welcome 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
- リポジトリーを POM ファイルに追加する

Maven 設定へのリポジトリーの追加

この設定の手法は、POM ファイルがリポジトリー設定を上書きせず、含まれ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 **settings.xml** ファイルを見つけます。通常、これはユーザーのホームディレクトリー内の **.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. 以下の例のように、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** リポジトリーが含まれる **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 設定に関する詳細は、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

プロジェクトに直接リポジトリーを設定するには、以下の例のように、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** リポジトリーが含まれる **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対してファイルベースの 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。

ローカルに抽出し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 ファイルに以下の XML を適用します。

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

${repository-url} 展開し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の URL でなければなりません。

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 サンプルでの AMQ BROKER の使用

AMQ JMS Pool サンプルには、**queue** という名前の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

『**AMQ Broker の使用**』の説明に従い ブロッカーをインストール して、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、**<broker-instance-dir>** としてブローカーインスタンスの場所を参照します。

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. コンソールの出力で、起動時にログに記録される重要なエラーの有無を確認します。ブローカーは、準備が整う際に **Server is now live** をログに記録します。

```
$ 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 **queue** という名前のキューを作成します。

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

yes または no の質問への回答を求めるプロンプトが表示されます。そのすべてに no (**N**) 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と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: 2021-08-29 15:57:05 +1000