Red Hat AMQ 2020.Q4

Using the AMQ JMS Pool Library

For Use with AMQ Clients 2.8

Red Hat AMQ 2020.Q4 Using the AMQ JMS Pool Library

For Use with AMQ Clients 2.8

Legal Notice

Abstract

This guide describes how to install and configure the library, run hands-on examples, and use your client with other AMQ components.

Table of Contents

CHAPTER 1. OVERVIEW

AMQ JMS Pool is a library that provides caching of JMS connections, sessions, and message producers. It enables reuse of connection resources beyond the standard lifecycle defined by the JMS API.

AMQ JMS Pool operates as a standard JMS **ConnectionFactory** instance that wraps the **ConnectionFactory** of your chosen JMS provider and manages the lifetime of **Connection** objects from that provider based on the configuration of the JMS pool. It can be configured to share one or more connections among callers to the pool **createConnection()** methods.

AMQ JMS Pool is part of AMQ Clients, a suite of messaging libraries supporting multiple languages and platforms. For an overview of the clients, see AMQ Clients Overview. For information about this release, see AMQ Clients 2.8 Release Notes .

AMQ JMS Pool is based on the Pooled JMS messaging library.

1.1. KEY FEATURES

- JMS 1.1 and 2.0 compatible

- Automatic reconnect

- Configurable connection and session pool sizes

1.2. SUPPORTED STANDARDS AND PROTOCOLS

AMQ JMS Pool supports version 2.0 of the Java Message Service API.

1.3. SUPPORTED CONFIGURATIONS

AMQ JMS Pool supports the OS and language versions listed below. For more information, see Red Hat AMQ 7 Supported Configurations.

- Red Hat Enterprise Linux 7 and 8 with the following JDKs:

  - OpenJDK 8 and 11

  - Oracle JDK 8

  - IBM JDK 8

- Red Hat Enterprise Linux 6 with the following JDKs:

  - OpenJDK 8

  - Oracle JDK 8

- IBM AIX 7.1 with IBM JDK 8

- Microsoft Windows 10 Pro with Oracle JDK 8

- Microsoft Windows Server 2012 R2 and 2016 with Oracle JDK 8

- Oracle Solaris 10 and 11 with Oracle JDK 8

1.4. DOCUMENT CONVENTIONS

The sudo command

In this document, **sudo** is used for any command that requires root privileges. Exercise caution when using **sudo** because any changes can affect the entire system. For more information about **sudo**, see Using the sudo command .

File paths

In this document, all file paths are valid for Linux, UNIX, and similar operating systems (for example, /**home**/**andrea**). On Microsoft Windows, you must use the equivalent Windows paths (for example, **C:\Users\andrea**).

Variable text

This document contains code blocks with variables that you must replace with values specific to your environment. Variable text is enclosed in arrow braces and styled as italic monospace. For example, in the following command, replace ***\<project-dir\>*** with the value for your environment:

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

CHAPTER 2. INSTALLATION

This chapter guides you through the steps to install AMQ JMS Pool in your environment.

2.1. PREREQUISITES

- You must have a subscription to access AMQ release files and repositories.

- To build programs with AMQ JMS Pool, you must install Apache Maven.

- To use AMQ JMS Pool, you must install Java.

2.2. USING THE RED HAT MAVEN REPOSITORY

Configure your Maven environment to download the client library from the Red Hat Maven repository.

Procedure

1. Add the Red Hat repository to your Maven settings or POM file. For example configuration files, see Section B.1, "Using the online repository" .

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

2. Add the library dependency to your POM file.

   ```
   <dependency>
     <groupId>org.messaginghub</groupId>
     <artifactId>pooled-jms</artifactId>
     <version>1.1.1.redhat-00003</version>
   </dependency>
   ```

The client is now available in your Maven project.

2.3. INSTALLING A LOCAL MAVEN REPOSITORY

As an alternative to the online repository, AMQ JMS Pool can be installed to your local filesystem as a file-based Maven repository.

Procedure

1. Use your subscription to download the **AMQ Clients 2.8.0 JMS Pool Maven repository**.zip file.

2. Extract the file contents into a directory of your choosing.
   On Linux or UNIX, use the **unzip** command to extract the file contents.

   ```
   $ unzip amq-clients-2.8.0-jms-pool-maven-repository.zip
   ```

   On Windows, right-click the .zip file and select **Extract All**.

3. Configure Maven to use the repository in the **maven-repository** directory inside the extracted install directory. For more information, see Section B.2, "Using a local repository" .

2.4. INSTALLING THE EXAMPLES

Procedure

1. Use the **git clone** command to clone the source repository to a local directory named **pooled-jms**:

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

2. Change to the **pooled-jms** directory and use the **git checkout** command to switch to the **1.1.1** branch:

   ```
   $ cd pooled-jms
   $ git checkout 1.1.1
   ```

The resulting local directory is referred to as *<source-dir>* throughout this document.

CHAPTER 3. GETTING STARTED

This chapter guides you through the steps to set up your environment and run a simple messaging program.

3.1. PREREQUISITES

- To build the example, Maven must be configured to use the Red Hat repository or a local repository.

- You must install the examples.

- You must have a message broker listening for connections on **localhost**. It must have anonymous access enabled. For more information, see Starting the broker.

- You must have a queue named **queue**. For more information, see Creating a queue.

3.2. RUNNING HELLO WORLD

The Hello World example calls **createConnection()** for each character of the string "Hello World", transferring one at a time. Because AMQ JMS Pool is in use, each call reuses the same underlying JMS **Connection** object.

Procedure

1. Use Maven to build the examples by running the following command in the ***<source-dir>*/pooled-jms-examples** directory.

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

   The addition of **dependency:copy-dependencies** results in the dependencies being copied into the **target/dependency** directory.

2. Use the **java** command to run the example.
   On Linux or UNIX:

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

   On Windows:

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

Running it on Linux results in the following output:

```
$ 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
```

The source code for the example is in the **<source-dir>/pooled-jms-examples/src/main/java** directory. The JNDI and logging configuration is in the **<source-dir>/pooled-jms-examples/src/main/resources** directory.

CHAPTER 4. CONFIGURATION

The AMQ JMS Pool **ConnectionFactory** implementation exposes several configuration options that control the behavior of the pool and the JMS resources it manages.

The configuration options are exposed as **set** methods on the **JmsPoolConnectionFactory** object. For example, the *maxConnections* option is set using the **setMaxConnections(int)** method.

4.1. CONNECTION OPTIONS

These options affect how the JMS pool creates and manages the connections in the pool.

The pooled **ConnectionFactory** creates a pool of connections for each user and password combination used to create a connection, plus a separate pool for those without a username or password. If you require a more fine-grained division of connections into pools, you must explicitly create distinct pool instances.

maxConnections

> The maximum number of connections for a single pool. The default is 1.

connectionIdleTimeout

> The time in milliseconds before a connection not currently on loan can be evicted from the pool. The default is 30 seconds. A value of 0 disables the timeout.

connectionCheckInterval

> The time in milliseconds between periodic checks for expired connections. The default is 0, meaning the check is disabled.

useProviderJMSContext

> If enabled, use the **JMSContext** classes of the underlying JMS provider. It is disabled by default. In normal operation, the pool uses its own generic **JMSContext** implementation to wrap connections from the pool instead of using the provider implementation. The generic implementation might have limitations the provider implementation does not. However, when enabled, connections from the **JMSContext** API are not managed by the pool.

4.2. SESSION OPTIONS

These options affect the behavior of sessions that are created from pooled connections.

maxSessionsPerConnection

> The maximum number of sessions for each connection. The default is 500. A negative value removes any limit.
> If the limit is exceeded, **createSession()** either blocks or throws an exception, depending on configuration.

blockIfSessionPoolIsFull

> If enabled, block **createSession()** until a session becomes available in the pool. It is enabled by default.
> If disabled, calls to **createSession()** throw an **IllegalStateException** if no session is available.

blockIfSessionPoolIsFullTimeout

> The time in milliseconds before a blocked call to **createSession()** throws an **IllegalStateException**. The default is -1, meaning the call blocks forever.

useAnonymousProducers

If enabled, use a single anonymous JMS **MessageProducer** for all calls to **createProducer()**. It is enabled by default.

In rare cases, this behavior is undesirable. If disabled, every call to **createProducer()** results in a new **MessageProducer** instance.

CHAPTER 5. EXAMPLES

This chapter demonstrates the use of AMQ JMS Pool through example programs.

For more examples, see the Pooled JMS examples.

5.1. PREREQUISITES

- To build the examples, Maven must be configured to use the Red Hat repository or a local repository.

- To run the examples, your system must have a running and configured broker.

5.2. ESTABLISHING A CONNECTION

This example creates a new connection pool, binds it to a connection factory, and uses the pool to create a new connection.

Example: Establishing a connection – **Connect.java**

```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. CONFIGURING THE POOL

This example demonstrates setting connection and session configuration options.

Example: Configuring the pool – ConnectWithConfiguration.java

```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. RUNNING THE EXAMPLES

To compile and run the example programs, use the following procedure.

Procedure

1. Create a new project directory. This is referred to as **<project-dir>** in the steps that follow.

2. Copy the example Java listings to the following locations:

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

3. Use a text editor to create a new **<project-dir>/pom.xml** file. Add the following XML to it:

   ```xml
   <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>1.1.1.redhat-00003</version>
       </dependency>
       <dependency>
         <groupId>org.apache.qpid</groupId>
         <artifactId>qpid-jms-client</artifactId>
         <version>${qpid-jms-version}</version>
       </dependency>
     </dependencies>
   </project>
   ```

   Replace **${qpid-jms-version}** with your preferred Qpid JMS version.

4. Change to the project directory and use the **mvn** command to compile the program.

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

   The addition of **dependency:copy-dependencies** results in the dependencies being copied into the **target/dependency** directory.

5. Use the **java** command to run the program.
   On Linux or UNIX:

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

   On Windows:

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

These sample commands run the **Connect** example. To run another example, replace **Connect** with the class name of your desired example.

Running the **Connect** example on Linux results in the following output:

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

APPENDIX A. USING YOUR SUBSCRIPTION

AMQ is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

A.1. ACCESSING YOUR ACCOUNT

Procedure

1. Go to access.redhat.com.

2. If you do not already have an account, create one.

3. Log in to your account.

A.2. ACTIVATING A SUBSCRIPTION

Procedure

1. Go to access.redhat.com.

2. Navigate to **My Subscriptions**.

3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

A.3. DOWNLOADING RELEASE FILES

To access .zip, .tar.gz, and other release files, use the customer portal to find the relevant files for download. If you are using RPM packages or the Red Hat Maven repository, this step is not required.

Procedure

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.

2. Locate the **Red Hat AMQ** entries in the **INTEGRATION AND AUTOMATION** category.

3. Select the desired AMQ product. The **Software Downloads** page opens.

4. Click the **Download** link for your component.

A.4. REGISTERING YOUR SYSTEM FOR PACKAGES

To install RPM packages for this product on Red Hat Enterprise Linux, your system must be registered. If you are using downloaded release files, this step is not required.

Procedure

1. Go to access.redhat.com.

2. Navigate to **Registration Assistant**.

3. Select your OS version and continue to the next page.

4. Use the listed command in your system terminal to complete the registration.

For more information about registering your system, see one of the following resources:

- Red Hat Enterprise Linux 6 – Registering the system and managing subscriptions

- Red Hat Enterprise Linux 7 – Registering the system and managing subscriptions

- Red Hat Enterprise Linux 8 – Registering the system and managing subscriptions

APPENDIX B. USING RED HAT MAVEN REPOSITORIES

This section describes how to use Red Hat-provided Maven repositories in your software.

B.1. USING THE ONLINE REPOSITORY

Red Hat maintains a central Maven repository for use with your Maven-based projects. For more information, see the repository welcome page.

There are two ways to configure Maven to use the Red Hat repository:

- Add the repository to your Maven settings
- Add the repository to your POM file

Adding the repository to your Maven settings
This method of configuration applies to all Maven projects owned by your user, as long as your POM file does not override the repository configuration and the included profile is enabled.

Procedure

1. Locate the Maven **settings.xml** file. It is usually inside the **.m2** directory in the user home directory. If the file does not exist, use a text editor to create it.
   On Linux or UNIX:

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

   On Windows:

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

2. Add a new profile containing the Red Hat repository to the **profiles** element of the **settings.xml** file, as in the following example:

   Example: A Maven settings.xml file containing the Red Hat repository

   ```xml
   <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>
```

For more information about Maven configuration, see the Maven settings reference.

Adding the repository to your POM file

To configure a repository directly in your project, add a new entry to the **repositories** element of your POM file, as in the following example:

Example: A Maven **pom.xml** file containing the Red Hat repository

```
<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>
```

For more information about POM file configuration, see the Maven POM reference.

B.2. USING A LOCAL REPOSITORY

Red Hat provides file-based Maven repositories for some of its components. These are delivered as downloadable archives that you can extract to your local filesystem.

To configure Maven to use a locally extracted repository, apply the following XML in your Maven settings or POM file:

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

${repository-url} must be a file URL containing the local filesystem path of the extracted repository.

Table B.1. Example URLs for local Maven repositories

Operating system	Filesystem path	URL
Linux or UNIX	**/home/alice/maven-repository**	**file:/home/alice/maven-repository**
Windows	**C:\repos\red-hat**	**file:C:\repos\red-hat**

APPENDIX C. USING AMQ BROKER WITH THE EXAMPLES

The AMQ JMS Pool examples require a running message broker with a queue named **queue**. Use the procedures below to install and start the broker and define the queue.

C.1. INSTALLING THE BROKER

Follow the instructions in *Getting Started with AMQ Broker* to install the broker and create a broker instance. Enable anonymous access.

The following procedures refer to the location of the broker instance as **<broker-instance-dir>**.

C.2. STARTING THE BROKER

Procedure

1. Use the **artemis run** command to start the broker.

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

2. Check the console output for any critical errors logged during startup. The broker logs **Server is now live** when it is ready.

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

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


    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. CREATING A QUEUE

In a new terminal, use the **artemis queue** command to create a queue named **queue**.

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

You are prompted to answer a series of yes or no questions. Answer **N** for no to all of them.

Once the queue is created, the broker is ready for use with the example programs.

C.4. STOPPING THE BROKER

When you are done running the examples, use the **artemis stop** command to stop the broker.

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

Revised on 2020-10-08 11:28:47 UTC