Red Hat AMQ 7.3

Using the AMQ JMS Pool Library

For Use with AMQ Clients 2.4

Last Updated: 2019-06-18

Red Hat AMQ 7.3 Using the AMQ JMS Pool Library

For Use with AMQ Clients 2.4

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 2.4 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 following OS and language versions:

- Red Hat Enterprise Linux 6, 7, and 8 with the following JDKs:
  - OpenJDK 8 and 11
  - Oracle JDK 8
  - IBM JDK 8
- HP-UX 11i with HP-UX JVM 8
- IBM AIX 7.1 with IBM JDK 8
- Oracle Solaris 10 and 11 with Oracle JDK 8
- Microsoft Windows 10 Pro with Oracle JDK 8
- Microsoft Windows Server 2012 R2 and 2016 with Oracle JDK 8

For more information, see Red Hat AMQ Supported Configurations .

1.4. DOCUMENT CONVENTIONS

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

CHAPTER 2. INSTALLATION

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

2.1. USING THE RED HAT MAVEN REPOSITORY

The library uses Apache Maven as its build tool. You can configure your Maven environment to download the 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>1.0.4.redhat-00004</version>
</dependency>
```

The library is now available in your Maven project.

2.2. 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 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 jboss-amq-pooled-jms-1.0.4.redhat-00004-maven-repository.zip
```

   On Windows, right-click on 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” .

2.3. INSTALLING THE ZIP FILE

The AMQ JMS Pool zip file contains the examples and a distribution of the client libraries for those not using Maven. If you are using Maven and do not require the examples, you do not need to install the zip file.

Procedure

1. Use your subscription to download the **AMQ JMS Pool** 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 packaged-pooled-jms-1.0.4.redhat-00004-bin.zip
   ```

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

   When you extract the contents of the zip file, a directory named **packaged-pooled-jms-1.0.4.redhat-00004** is created. This is the top-level directory of the installation and is referred to as **<install-dir>** throughout this document.
3. (Optional) If you are not using Maven, add the jar files in the **<install-dir>/lib** directory to your Java classpath.

CHAPTER 3. GETTING STARTED

This chapter guides you through a simple exercise to help you get started using AMQ JMS Pool.

3.1. PREREQUISITES

- The Hello World example program is located in the AMQ JMS Pool zip file. To get started, you must install the zip file.
- To build the example, Maven must be configured to use the Red Hat repository or a local repository.

3.2. PREPARING THE BROKER

The example programs require a running broker with a queue named **queue**. Follow these steps to define the queue and start the broker:

Procedure

1. Install the broker.
2. Create a broker instance. Enable anonymous access.
3. Start the broker instance and check the console for any critical errors logged during startup.

```
$ <broker-instance-dir>/bin/artemis run
...
14:43:20,158 INFO  [org.apache.activemq.artemis.integration.bootstrap] AMQ101000: Starting ActiveMQ Artemis Server
...
15:01:39,686 INFO  [org.apache.activemq.artemis.core.server] AMQ221020: Started Acceptor at 0.0.0.0:5672 for protocols [AMQP]
...
15:01:39,691 INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
```

4. Use the **artemis queue** command to create a queue called **queue**.

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

You are prompted to answer a series of questions. For yes or no questions, type **N**. Otherwise, press Enter to accept the default value.

3.3. RUNNING HELLO WORLD

Procedure

1. Use Maven to build the examples by running the following command in the **<install-dir>/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
```

The 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.

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 **<install-dir>/examples/src/main/java** directory. The JNDI and logging configuration is in the **<install-dir>/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.

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

```
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

```
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:

```
<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>1.0.4.redhat-00004</version>
    </dependency>
    <dependency>
      <groupId>org.apache.qpid</groupId>
      <artifactId>qpid-jms-client</artifactId>
      <version>0.42.0.redhat-00002</version>
    </dependency>
  </dependencies>
</project>
```

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.

Accessing your account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

Activating a subscription

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

Downloading ZIP and TAR files

To access ZIP or TAR files, use the customer portal to find the relevant files for download. If you are using RPM packages, this step is not required.

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 **JBOSS INTEGRATION AND AUTOMATION** category.
3. Select the desired AMQ product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

Registering your system for packages

To install RPM packages on Red Hat Enterprise Linux, your system must be registered. If you are using ZIP or TAR files, this step is not required.

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.

To learn more see How to Register and Subscribe a System to the Red Hat Customer Portal .

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

```
<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**

Revised on 2019-06-18 17:15:19 UTC