ResourceLoader12

Resource Loader

The Resource Loader is a facility that allows easy configuration and management of resources. Pools configured via the Resource Loader will be created automatically during the transaction manager's startup and will be closed during its shutdown.

It consists of an extra layer above the JDBC and JMS pools which configures them via a properties file. It is not mandatory to use it but sometimes it is better than manually creating resource pools. Another good example of Resource Loader usage is when BTM is embedded in a servlet container which configuration file cannot easily be extended, like Tomcat12 for instance.

Contents

- Enabling the Resource Loader
- Configuring a JDBC pool
- Configuring a JMS pool
- Ant-like references
- JNDI binding

Enabling the Resource Loader

You can enable it by setting the `bitronix.tm.resource.configuration` property in the configuration file or by calling `setResourceConfigurationFilename()` on the Configuration object.

This property must contain a path to a properties file that will be loaded by the Resource Loader to create pools. Its default value is `null` which means the Resource Loader is disabled and resources should be manually created via the API.

Configuring a JDBC pool

You can configure a JDBC connection pool that wraps the driver's `javax.sql.XADataSource` implementation and presents it to the user as a `javax.sql.DataSource`, much like with the API.

To configure a JDBC pool, create a set of properties:

```
-   (A) .(B). (C)   -
resource.ds1.className=oracle.jdbc.xa.client
.OracleXADataSource
resource.ds1.uniqueName=oracle
resource.ds1.maxPoolSize=5
resource.ds1.driverProperties.user=users1
resource.ds1.driverProperties.password=users
1
resource.ds1.driverProperties.URL=jdbc:oracl
e:thin:@localhost:1521:XE
```

`(A)` is constant and never changes.
`(B)` is an arbitrary name used to group a set of properties together.
`(C)` is a javabean property of bitronix.tm.resource.jdbc.PoolingDataSource.

Refer to the `PoolingDataSource` javadoc or to the JDBC configuration page to know what properties can be set.

Configuring a JMS pool

You can configure a JMS connection pool that wraps JMS server's `javax.jms.XAConnectionFactory` impleme ntation and presents it to the user as a `javax.jms.ConnectionFactory`, much like with the API or for JDBC.

To configure a JMS pool, create a set of properties:

```
-   (A) .(B). (C)   -
resource.mq1.className=org.activemq.ActiveMQ
XAConnectionFactory
resource.mq1.uniqueName=activemq

resource.mq1.maxPoolSize=2

resource.mq1.driverProperties.userName=defau
ltUser
resource.mq1.driverProperties.password=defau
ltPassword
resource.mq1.driverProperties.brokerURL=tcp:
//localhost:61616
```

`(A)` is constant and never changes.
`(B)` is an arbitrary name used to group a set of properties together.
`(C)` is a javabean property of bitronix.tm.resource.jms.PoolingConnectionFactory.

Refer to the `PoolingConnectionFactory` javadoc or to the JMS configuration page to know what properties can be set.

Ant-like references

Unlike the transaction manager, the Resource Loader does not support ant-like references (`${...}`) in its configuration file.

JNDI binding

You can optionally have the Resource Loader bind resources to JNDI after configuration by setting this property to true:

```
bitronix.tm.resource.bind=true
```

Resources are then bound in the default JNDI context using their unique name as the JNDI name.