JndiXaConnectionFactory2x

XA Connection factories only available from JNDI

The JMS specification does not specify that `XAConnectionFactory` implementations should be javabeans. Because of this, it is not always possible to build a `PoolingConnectionFactory` directly.

Contents

- The JndiXAConnectionFactory
- JNDI credentials
- Other JNDI properties

The JndiXAConnectionFactory

When you have no way to create a `XAConnectionFactory` in a javabean way, you can specify the `className` argument of `PoolingConnectionFactory` to be bitronix.tm.resource.jms.JndiXAConnectionFactory. This class is an implementation of `XAConnectionFactory` that will wrap another implementation fetched from JNDI.

Set the `name`, `initialContextFactory` and `providerUrl` properties respectively to the JNDI name of the `XAConnectionFactory` implementation, the initial context factory and provider URL used to access the JNDI server. The last two parameters can be omitted if the `XAConnectionFactory` implementation is available in the default JNDI context (inside an application server for instance).

Here is what your code would look like for creating a SwiftMQ `PoolingConnectionFactory`:

```
PoolingConnectionFactory myConnectionFactory
= new PoolingConnectionFactory ();
myConnectionFactory.setClassName("bitronix.t
m.resource.jms.JndiXAConnectionFactory");
myConnectionFactory.setUniqueName("swiftmq")
;
myConnectionFactory.setMaxPoolSize(5);
myConnectionFactory.getDriverProperties().se
tProperty("name",
"QueueConnectionFactory@router1");
myConnectionFactory.getDriverProperties().se
tProperty("initialContextFactory",
"com.swiftmq.jndi.InitialContextFactoryImpl"
);
myConnectionFactory.getDriverProperties().se
tProperty("providerUrl",
"smqp://localhost:4001/timeout=10000");

Connection c =
myConnectionFactory.createConnection();
 // create some session then send or receive
some message
c.close();

myConnectionFactory.close();
```

JNDI credentials

It is often required to use credentials to be able to connect to a JNDI repository. The bitronix.tm.resource.jms.JndiXAConnectionFactory class fully supports this via its `securityPrincipal` and `securityCredentials` properties as illustrated bellow:

```
PoolingConnectionFactory myConnectionFactory
= new PoolingConnectionFactory ();
...
myConnectionFactory.getDriverProperties().se
tProperty("name",
"QueueConnectionFactory@router1");
myConnectionFactory.getDriverProperties().se
tProperty("initialContextFactory",
"com.swiftmq.jndi.InitialContextFactoryImpl"
);
myConnectionFactory.getDriverProperties().se
tProperty("providerUrl",
"smqp://localhost:4001/timeout=10000");

// set JNDI username / password
myConnectionFactory.getDriverProperties().se
tProperty("securityPrincipal", "jndiUser");
myConnectionFactory.getDriverProperties().se
tProperty("securityCredentials",
"jndiPassword");
...
```

Other JNDI properties

There are some other properties to make the `JndiXAConnectionFactory` more flexible:

- **urlPkgPrefixes** Use it to set the Context#URL_PKG_PREFIXES.
- **narrowJndiObject** Use it to indicate if PortableRemoteObject.narow() should be applied on the object fetched from JNDI.
- **extraJndiProperties** Extra JNDI environment properties added the the InitialContext.