Jetspeed-2 RDBMS Components v.2.1.2

Project Documentation

Apache Software Foundation
15 July 2007

Table of Contents

1 Jetspeed-2 RDBMS Documentation

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

1.1 RDBMS Overview

RDBMS Overview

Jetspeed-2 RDBMS component provide a layer of abstraction from the persistence mechanism used by Jetspeed-2. It provides facilities for datasource configuration as well as data access.

Datasource Configuration

Jetspeed-2 uses OJB `PersistenceBroker` API as its default persistence mechanism. The `ConnectionRepositoryEntry` component configures OJB for Jetspeed-2 as well as the properties available under `/etc/db-ojb` in the Jetspeed-2 source repository or `WEB-INF/classes` in a deployed instance of Jetspeed-2.

The `datasource.xml` spring assembly configuration file configures `ConnectionRepositoryEntry` and is located in `WEB-INF/assembly/boot`.

The `ConnectionRepositoryEntry` configures an entry in OJB's ConnectionRepository according to its properties. The properties `driverClassName`, `url`, `username` and `password` are used only if no `jndiName` is set, i.e. if the connection factory uses the driver to create data sources. The platform settings are derived from the configured data source or database driver using OJB's `JdbcMetadataUtils` class. The default Jetspeed-2 `ConnectionRepositoryEntry` configuration expose a datasource.

```
    <bean id="JetspeedDS"
class="org.apache.jetspeed.components.rdbms.ojb.ConnectionRepositoryEntry">
      <property name="jndiName">
        <value>java:comp/env/jdbc/jetspeed</value>
      </property>
    </bean>
```

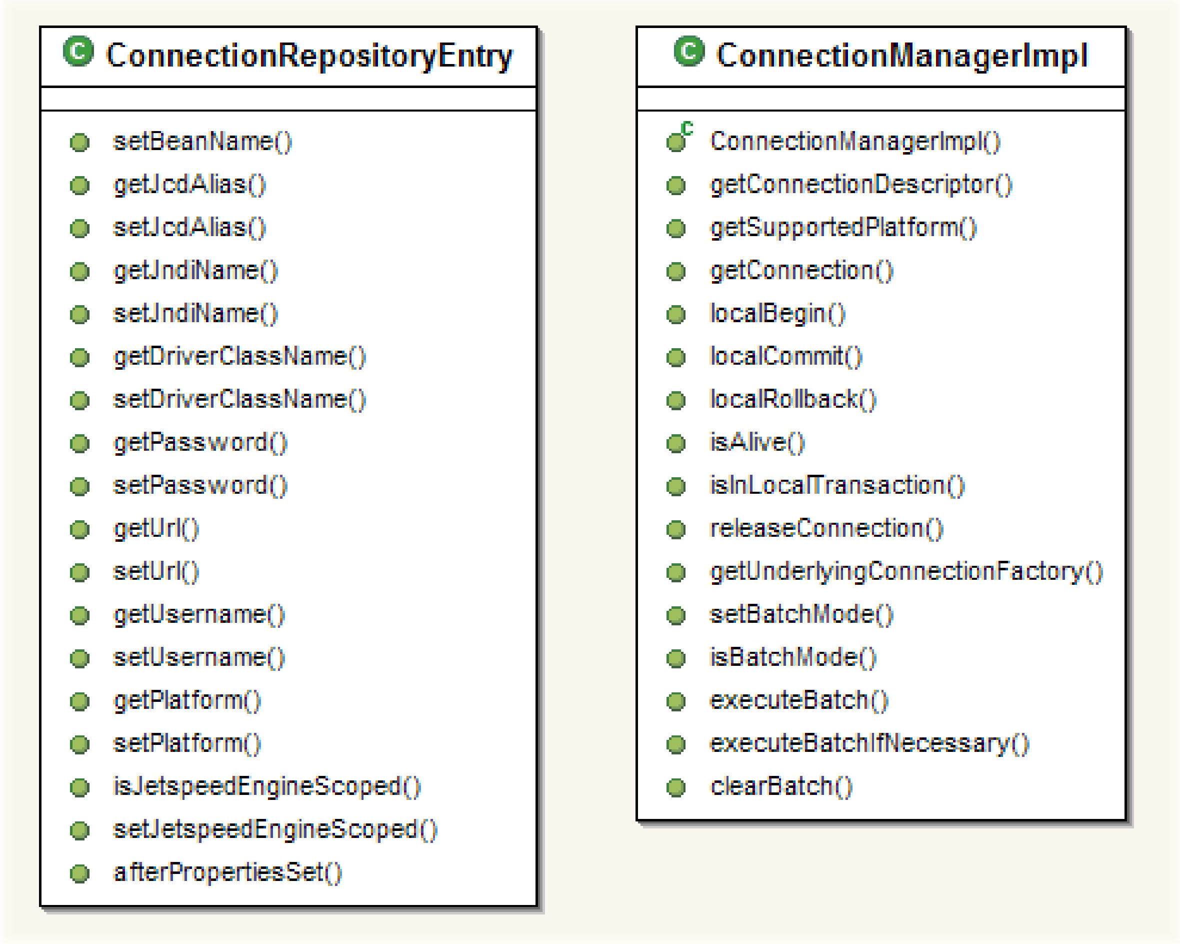
In order for OJB to be configured properly with Jetspeed-2, the `OJB.properties` file (located under `/etc/db-ojb/OJB.properties` in the source tree and `WEB-INF/classes` in the deployed application) must set:

```
ConnectionManagerClass=org.apache.jetspeed.components.rdbms.ojb.ConnectionManagerImpl
```

instead of:

```
ConnectionFactoryClass=org.apache.ojb.broker.accesslayer.ConnectionFactoryManagedImpl
```

A class diagram of `ConnectionRepositoryEntry` and `ConnectionManagerImpl` is provided below:



OJB Datasource Configuration

The bean name provided in `datasource.xml` must match the `jdbc-connection-descriptor jcd-alias` property (by default `JetspeedDS`) located in OJB `repository_database.xml` as illustrated below.

```
<jdbc-connection-descriptor
    jcd-alias="JetspeedDS"
    default-connection="true"
    batch-mode="false">
```

Jetspeed-2 Datasource Configuration in Tomcat

Jetspeed-2 configure the following datasource in Tomcat. In the source tree, the Tomcat datasource configuration is located under `/etc/conf/tomcat`. When deployed Jetspeed-2 in a Tomcat instance, the Jetspeed-2 datasource configuration are deployed under `${tomcat_home}/conf/Catalina/localhost/jetspeed.xml`. If a different portal name is being used for Jetspeed-2, the configuration file will be named accordingly.

```
<Resource name="jdbc/jetspeed" auth="Container"
             factory="org.apache.commons.dbcp.BasicDataSourceFactory"
             type="javax.sql.DataSource" username="" password=""
             driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
             url="jdbc:derby:/tmp/productiondb;create=true"
             maxActive="100" maxIdle="30" maxWait="10000"/>
```

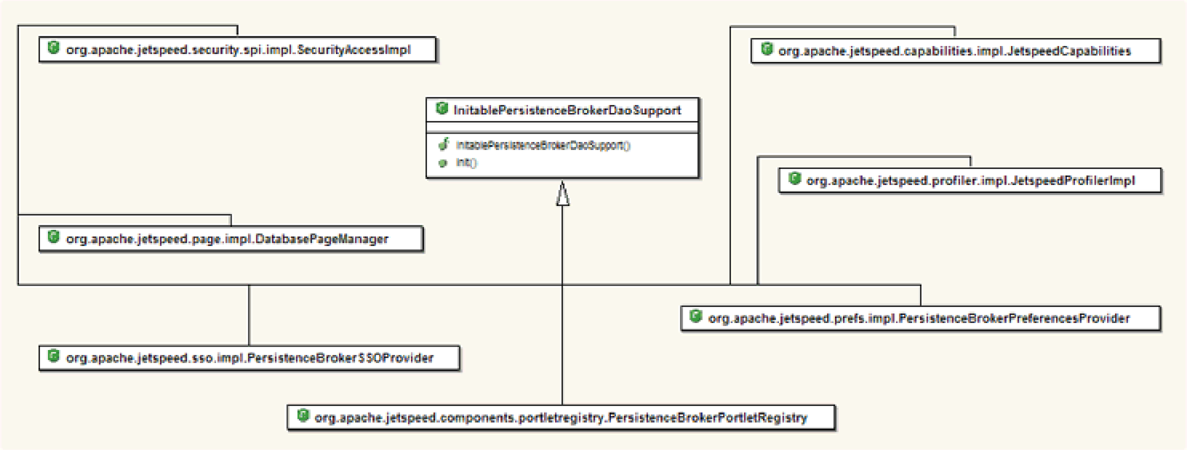
1.2 Data Access in Jetspeed-2

Data Access Overview

Jetspeed-2 RDBMS component provide some of level of abstraction from the underlying persistence mechanism.

Data Access Using Object Relational Mapping

Jetspeed-2 uses object relational mapping as the underlying technology for persistence. By default, Apache OJB is used as an ORM engine. In order to minimize Jetspeed-2 OJB dependencies, the `InitablePersistenceBrokerDaoSupport` provides a layer of abstraction that minimizes the dependencies on a specific ORM engine. The class diagram below illustration the Jetspeed-2 implementation classes that leverage `InitablePersistenceBrokerDaoSupport`:


org.apache.jetspeed.security.spi.impl.SecurityAccessImpl
org.apache.jetspeed.capabilities.impl.JetspeedCapabilities
InitablePersistenceBrokerDaoSupport
InitablePersistenceBrokerDaoSupport()
init()
org.apache.jetspeed.profiler.impl.JetspeedProfilerImpl
org.apache.jetspeed.page.impl.DatabasePageManager
org.apache.jetspeed.prefs.impl.PersistenceBrokerPreferencesProvider
org.apache.jetspeed.sso.impl.PersistenceBrokerSSOProvider
org.apache.jetspeed.components.portletregistry.PersistenceBrokerPortletRegistry

The `InitablePersistenceBrokerDaoSupport` extends `org.springframework.orm.ojb.support.PersistenceBrokerDaoSupport`.

Spring ORM Support

Spring's adds significant support when using the O/R mapping layer of your choice to create data access applications. The `InitablePersistenceBrokerDaoSupport` extends spring layer of abstraction for persistence support, specifically the OJB `PersistenceBroker` API support .

Using such a layer of abstraction has many advantages. Some of the advantages outlined in Spring's documentation are:

- Ease of testing. Spring's inversion of control approach makes it easy to swap the implementations and config locations of persistence manager instances, JDBC DataSources, transaction managers, and mapper object implementations (if needed). This makes it much easier to isolate and test each piece of persistence-related code in isolation.

- Common data access exceptions. Spring can wrap exceptions from you O/R mapping tool of choice, converting them from proprietary (potentially checked) exceptions to a common runtime DataAccessException hierarchy. This allows you to handle most persistence exceptions, which are non-recoverable, only in the appropriate layers, without annoying boilerplate catches/throws, and exception declarations. You can still trap and handle exceptions anywhere you need to. Remember that JDBC exceptions (including DB specific dialects) are also converted to the same hierarchy, meaning that you can perform some operations with JDBC within a consistent programming model.
- General resource management. Spring application contexts can handle the location and configuration of persistence managers instances, JDBC DataSources, and other related resources. This makes these values easy to manage and change. Spring offers efficient, easy and safe handling of persistence resources.
- Integrated transaction management. Spring allows you to wrap your O/R mapping code with either a declarative, AOP style method interceptor, or an explicit 'template' wrapper class at the Java code level. In either case, transaction semantics are handled for you, and proper transaction handling (rollback, etc) in case of exceptions is taken care of. As discussed below, you also get the benefit of being able to use and swap various transaction managers, without your ORM specific code being affected: for example, between local transactions and JTA, with the same full services (such as declarative transactions) available in both scenarios. As an additional benefit, JDBC-related code can fully integrate transactionally with the code you use to do O/R mapping. This is useful for data access that's not suitable for O/R mapping, such as batch processing or streaming of BLOBs, which still needs to share common transactions with O/R mapping operations.
- To avoid vendor lock-in, and allow mix-and-match implementation strategies.