OJBWithJDO

Using OJB with JDO in Cocoon

Note: Since Cocoon 2.1.3 OJBBlock is part of Cocoon.

This is a "howto" about using OJB-JDO in Cocoon.

In this Howto we will show how to set a simple example of using OJB-JDO with Cocoon. The example uses PostgreSQL database, but you can change it to use another RDMS.

Introduction

ObJectRelationalBridge (OJB) is an Object/Relational mapping tool that allows transparent persistence for Java Objects against relational databases.

OJB has 3 methods for access to Databases:

1. PersistentBroker NewbieGuideToOJB
2. ODMG
3. JDO

Since JDO is the recommended successor of ODMG, we will try to show how to start using JDO with Cocoon.

Scenario

- We need to interact with a table that manage user's info. The table is called **auth_user**. The table primary key is autofilled with a sequence called **auth_users_seq**.
- We have already installed and running Cocoon.

Environment

- Red Hat Linux 9
- Sun J2SDK 1.4.2
- Tomcat 4.1.24
- Cocoon 2.1 (CVS 28-Jul-3003)
- PostgreSQL 7.3.2
- OJB-JDO 1.0rc4 (CVS 28-Jul-2003)

Solution

1. Download OJB 1.0.0 from http://db.apache.org/builds/ojb/ (approx 8MB)
    - Be sure the binaries have JDORI support.
    - The command to build from sources with JDORI support is:

      ```
      bin/build.sh with-jdori jar
      ```

    - **Note**: *Before building see **2.** and check requirements from OJB Quickstart*

2. Download the JDO libraries:

- jdo.jar - for the OjbStore JDORI Plugin only.
- jdori.jar - for the OjbStore JDORI Plugin only.

3. Create the **auth_user** table and the **auth_user_seq** sequence in your **test** database.

- See **posgres.sql** for PostgreSQL

4. Create a file **auth_user.jdo**

- This file contains the mapping between the beans and database columns for JDO

5. Compile and **enhance** the **auth_user.java** bean.

- See **auth_user.java**.
- The classes imported in step 1. and 2. are needed, include it in your classpath.
- Put the ONLY the output class into "TOMCAT_HOME/webapps/cocoon/WEB-INF/classes/test"
- to enhance the file, put the auth_user.class and the auth_user.jdo files in a directory called "test" and from the directory above "test" run the following
  $ java -cp "./;c:\db-ojb-1.0.0\lib\jdo.jar;c:\db-ojb-1.0.0\lib\jdori.jar;c:\db-ojb-1.0.0\lib\db-ojb-1.0.0.jar" com.sun.jdori.enhancer.Main -v -f test\auth_user.class test\auth_user.jdo

Note: in auth_user.jdo, first change the DTD location to: <!DOCTYPE jdo SYSTEM "http://java.sun.com/dtd/jdo_1_0.dtd">

6. Create a file **repository.xml** in "TOMCAT_HOME/webapps/cocoon/WEB-INF/classes"

- This file contains info about your DB and credentials (jdbc driver, username and password).

7. Copy files **repository.dtd** and **OJB.properties** in "TOMCAT_HOME/webapps/cocoon/WEB-INF/classes"

8. Copy the following files from the OJB distribution to "TOMCAT_HOME/webapps/cocoon/WEB-INF/lib":

```
antlr.jar
jdori.jar
commons-dbcp.jar
commons-pool.jar
db-ojb-1.0.rc4.jar
jdo.jar
```

- Please also include your database driver. As we are working with postgres...
  ***pg73jdbc3.jar**

9. Copy **adduser.xml** and **sitemap.xmap** into "TOMCAT_HOME/webapps/cocoon/ojbdemo"

Test

Open *http://localhost:8080/cocoon/ojbdemo/adduser*

If all goes well, a record is added to your database.

If you press the refresh button in your browser you will get an error because you are trying to insert a new record with the same usr_login. (The column is UNIQUE).

Links

- **OJB Website:** http://db.apache.org/ojb/
- **Quickstart:** http://db.apache.org/ojb/quickstart.html
- **Using JDO API in OJB**: http://db.apache.org/ojb/tutorial4.html
- **Hands-on Java Data Objects**: http://www-106.ibm.com/developerworks/java/edu/j-dw-javajdo-i.html

Notes

- This is a first attempt to integrate OJB-JDO with Cocoon. I am also a OJB newbie. But after a week studing about OJB and JDO I think OJB with JDO is a good way to fill the current "model" gap in the MVC design.
- Currently I am trying to create a component that will create a **PersistentManagerFactory** just once for all the application. **Done**: See OJBBlock

The idea is that any request can "get" the global factory and ask his own PersistentManager. This is the correct way. After that we can use JDO as the "model" using java classes.

Readers comments.

Also, for new OJB people, you need to get j2ee.jar to build OJB (JD Daniels)

Attachment: Cocoon-OJB-JDO.zip