Alternate JPA Providers in WebSphere Application Server

Author: Lisa Walkosz
liwalkos@us.ibm.com
Date: May 28, 2010
THE INFORMATION CONTAINED IN THIS REPORT IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. ALTHOUGH EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BYIBM WITHOUT NOTICE.

IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS REPORT OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS REPORT IS INTENDED TO, OR SHALL HAVE THE EFFECT OF CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.
Trademarks and Service Marks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- WebSphere
- WebSphere Application Server

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Hibernate is licensed under the LGPL v2.1

OpenJPA 2.0 is licensed under Apache License v2.0

Other company, product and service names may be trademarks or service marks of others.
Executive Summary

Database transactions can be made easier when using a persistence architecture. The Java Persistence Architecture, also known as JPA, is a great way to access your database transactions without the need for complex SQL queries, worrying about performance, or other areas of potential programmer error. There are many JPA persistence providers available for application developers to utilize. WebSphere Application Server has the ability to set up and use many of them.
Audience

This report is intended for customers whose Java EE application uses a JPA provider that is not the WebSphere default.

For example, WebSphere JPA 1.0 is the provider for v6.1 with EJB 3.0 Feature Pack, WebSphere JPA 1.2 is the provider for WebSphere v7, and WebSphere JPA 2.0 is the default for the JPA 2.0 Feature Pack.

If for some reason your application needs to take advantage of an alternate JPA provider such as OpenJPA 1.3.x or a 3rd party provider such as Hibernate JPA, it is a simple configuration to get it up and running.

Please note, only the shipped version of JPA in the WebSphere Application Server is supported by IBM, 3rd party providers may not be included in this support.
Scenario Description

A customer may want to have a different JPA provider for the following reasons:

- An application which was made using a different provider needs to work in WebSphere.
- An application developer wants only to use certain features and needs to configure a specific OpenJPA version to accomplish this.
- An application developer wants to compare and contrast different persistence providers on the WebSphere application server.

Our testing purpose is to validate that alternate JPA providers can be used for any of these reasons. We want to see how this is possible without restarting the server (setting them to application scope) as well as server scope, and also configuring the application server to use an alternate default JPA provider.

You can incorporate third-party providers such as Hibernate 3.0 into an application by either embedding jars inside of an application or creating a shared library. This example will use the shared library method. We will also show you how to set the persistence provider to become the default JPA provider of whatever scope you configure.
Topology

We are testing this in a developer's environment on a Windows machine with single server only. Rational Application Developer 7.5 was used to make any changes to the application during our test. We installed WebSphere 7.0.0.9 with the JPA 2.0 Version 1.0 Feature Pack to allow us to utilize the latest JPA has to offer. For JPA 2.0 functionality, you will need to use RAD 8.0.
Import Sample Application

- In lieu of creating an application, we can use a sample that comes with RAD 7.5, and quickly modify the persistence.xml only.
- In RAD, click Help->Samples
- On the left side, navigate to Samples->Application Samples->EJB->EJB 3.0 Counter Sample
- Click Import Sample and let the sample projects be imported to your workspace. Follow the instructions in the “Prepare to import” link.
Some helpful hints using RAD

- Now that you have the sample in your workspace, add a server definition to RAD so that you can develop more effectively.
- Window->Show View->Server->Servers (if Server doesn’t show up, go to Other… for more options)
- Your servers tab should open up; right click New->Server
- Pick your WebSphere Application Server 7.0 and point it to your WebSphere install root, or use the stub that came with RAD. Put any username and password if you have security enabled. From here you can start/stop and add applications to the server definition. Right click on the server and select “Add and Remove Projects…” Add your EJB Counter Sample by moving it to the right side in “Configured Projects:”. When you click Finish the application is basically being installed to the App Server without need for scripting or the administrative console. If you have set up your workspace, application server, and database correctly, you can test out the application.
- Now that you’ve seen the sample is working in its original state, let’s start playing with the code. We want to change the persistence provider that this application uses; there are a few ways we could do this. First, we could go into the persistence.xml and specify exactly what persistence provider to use. This is useful in the case of writing your application with methods calling WebSphere or Hibernate JPA specific APIs, such as using PureQuery. Since this example doesn’t have JPA API specific code, let’s try a different approach. You can configure the application server, WebSphere itself, to pick a default persistence provider.
Changes in the persistence.xml - explanation

Here is what you’ll need to make sure you can use OpenJPA or Hibernate as default persistence providers.

- The WebSphere application server works out of the box with the JPA provider it ships with, but there are still some required configurations to put in your persistence.xml in order to use an alternate one.

- Hibernate requires the configuration of two essential pieces in order to properly run with transactions. The first, hibernate.transaction.factory_class, defines transactional control and the second, hibernate.transaction.manager_lookup_class, defines the mechanism for registration of transaction synchronization so the persistence manager is notified at transaction end when it needs to synchronize changes with the database. For transactional control, both container-managed and bean-managed configurations are supported.

- Modify your persistence.xml. In the counter sample, it’s under EJBCounterSample->ejbModule->META-INF. Add the properties as seen below

- Note: We've changed my persistence version from 1.0 to 2.0 since we have the WebSphere JPA 2.0 feature pack installed and enabled. This test worked with WebSphere JPA 2.0, and OpenJPA 2.0 at the 2.0 version, but Hibernate was kept at Hibernate JPA 1.0.
Changes in the persistence.xml – example code

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <persistence-unit name="Counter">
    <jta-data-source>jdbc/EJB3SampleDatasource</jta-data-source>
    <class>com.ibm.websphere.ejb3sample.counter.JPACounterEntity</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <!-- HIBERNATE properties -->
      <property name="hibernate.transaction.factory_class" value="org.hibernate.transaction.CMTTransactionFactory"/>
      <property name="hibernate.transaction.manager_lookup_class" value="org.hibernate.transaction.WebSphereExtendedJTATransactionLookup"/>
      <property name="java:comp/UserTransaction" value="java:comp/UserTransaction"/>
      OR
      <!-- OPENJPA properties -->
      <property name="openjpa.Log" value="none"/>
      <property name="openjpa.TransactionMode" value="managed"/>
      <property name="openjpa.ConnectionFactoryMode" value="managed"/>
    </properties>
  </persistence-unit>
</persistence>
```
External Jars you will need for alternate providers

Hibernate requires specific 3rd party jars to be added to WebSphere. You can download them from Hibernate’s website. Here are the jars we used to get this sample working.

- hibernate-annotations.jar
- hibernate-commons-annotations.jar
- hibernate-entitymanager.jar
- hibernate-validator.jar (4.0.2)
- hibernate3.jar
- dom4j-1.6.1.jar
- javassist.jar (3.9.0)
- commons-collections-2.1.1.jar (3.1)
- cglib-2.1.3.jar (2.2)
- asm.jar
- asm-attrs.jar
- slf4j-log4j12.jar
- slf4j-api.jar
- slf4j-api-1.5.8.jar
- log4j.jar

Second, for OpenJPA specific jars, download from Apache's website. Here are the jars we used to get the sample working.

- commons-collections-3.2.1.jar
- commons-lang-2.1.jar
- commons-pool-1.5.2.jar
- openjpa-2.0.0.jar
- serp-1.13.1.jar

- OR, use the openjpa all file for ease of use.
- openjpa-all.jar
Creating a shared library

- You can put these jars into the App Server’s lib directory, but for better maintenance, consider creating a shared library. You can give separate servers, nodes, or cluster the ability to use the shared library without overwriting a version someone else may be using. To create a shared library, on the left navigation bar, click Environment->Shared Libraries. Create a New one, and give it a descriptive name. In the classpath section, add your jars by giving the full path name of where they would be located on your application server’s machine. Use WebSphere variables if you’re in a distributed environment.

- For OpenJPA you need to click the Class Loading checkbox for using an isolated class loader for this shared library.
Associate the shared library to your server or application

Choose if you wish to associate the library with a server or your application only. With your server, you can put multiple applications on that server with the same results, whereas application scoped would apply only to that application. Pick whichever scenario works for you.

- **Server association**
  - Now you need to associate a server with that library. Go into your server definition Servers->Server Types->WebSphere application servers. On the right side of the page, expand Java Process Management and select class loader. Create a new classloader, and select parent last. Slide that classloader on the right side, select Shared library references. Inside that, add the library you created in the previous step. That server now will load those jars the next time it is started.

- **Application association**
  - For OpenJPA, you cannot have your server associated with the library, WebSphere's default JPA will just overwrite it. You must associate your application with the shared library. Go into your application definition Applications->Application Types->WebSphere enterprise application->EJBCounterEAR. Now inside your application, go down to References and click Shared Library references. Click each module inside your ear (all if you are unsure of which module is using JPA) and add a shared library you defined earlier to them. Once done, click okay and save.
Set the default JPA provider in your server

- Now that your server or application has access to the JPA provider, you can specify whether or not to use it or the default wsjpa provider. There are a few ways to do this as well. You can either state inside your persistence.xml which provider you want to use, or you can change your default provider on your server.

![General Properties]

- To change the default provider in the administrative console, on the left navigation bar, click Server Types->WebSphere Application servers, on the right side, expand Container Services, and select Default Java Persistence API settings. The screen will show two default providers WebSphere supplies automatically: the WebSphere implementation of JPA, and OpenJPA. You can also select the second radio button and put in a 3rd party provider such as Hibernate.
Verify the right JPA provider is loading

- Now that you’ve selected your default provider, when your persistence.xml without a specified provider is loaded, it will pick the one you’ve configured as default.
- Your EJB Counter should be working just as it has before we switched the default provider, except using a different implementation.

If OpenJPA has loaded, you will see in your SystemOut.log the following.
JPACOMPONENT I CWWJP00281: The Java Persistence API (JPA) component is starting.
JPACOMPONENT I CWWJP00061: The org.apache.openjpa.persistence.PersistenceProviderImpl class is loaded as the default Java Persistence API (JPA) provider.
JPACOMPONENT I CWWJP00171: The Java Persistence API (JPA) component has started.

If Hibernate has loaded, you will see in your SystemOut.log the following.
JPACOMPONENT I CWWJP00281: The Java Persistence API (JPA) component is starting.
JPACOMPONENT I CWWJP00061: The org.hibernate.ejb.HibernatePersistence class is loaded as the default Java Persistence API (JPA) provider.
JPACOMPONENT I CWWJP00171: The Java Persistence API (JPA) component has started.

With no changes you would see the WebSphere implementation loaded, you will see in your SystemOut.log:
JPACOMPONENT I CWWJP00281: The Java Persistence API (JPA) component is starting.
JPACOMPONENT I CWWJP00171: The Java Persistence API (JPA) component has started.
Programmatically change your default JPA provider

- If you wish to programmatically change your persistence provider, you can instead specify it in the persistence.xml file. Just add the provider tag.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence_2_0.xsd">
    <persistence-unit name="Counter">
        <provider>com.ibm.websphere.persistence.PersistenceProviderImpl</provider>
        OR
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        OR
        <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
        <jta-data-source>jdbc/EJB3SampleDataSource</jta-data-source>
        <class>com.ibm.websphere.ejb3sample.counter.JPACounterEntity</class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
            <!-- HIBERNATE properties -->
            <property name="hibernate.transaction.factory_class" value="org.hibernate.transaction.CMTTransactionFactory"/>
            <property name="hibernate.transaction.manager_lookup_class" value="org.hibernate.transaction.WebSphereExtendedJTATransactionLookup"/>
            <property name="jta.UserTransaction" value="java:comp/UserTransaction"/>
            <!-- OPENJPA properties -->
            <property name="openjpa.Log" value="none"/>
            <property name="openjpa.TransactionMode" value="managed"/>
            <property name="openjpa.ConnectionFactoryMode" value="managed"/>
        </properties>
    </persistence-unit>
</persistence>
```
Validate your JPA provider is being used

If you have defined your default JPA provider for your server, you will be required to restart your server. To validate your persistence provider is actually changed, just check the SystemOut.logs after a server restart.

If you have defined your default JPA provider programmatically in your application, you are not required to restart your server. You will have to add code inside your application to indicate which persistence provider is being loaded. It does become apparent when using very different providers, such as Hibernate JPA which will load completely different classes than WebSphere's default JPA.

Make sure you have this function showing your Entity Manager Factory class in the System Out and that it's being called any time JPA is invoked. It will print out the implementation class which will show up as org.apache, com.ibm, or org.hibernate, and so on.

```java
void showJPAPr
```
End

This page intentionally left blank