OpenJPA Packaging Tricks

By Roger Keays, 29 December 2006

Figbird 2.0, our content management system, has a fairly unique module system which allows us to package extensions into jar files and simply drop them into WEB-INF/lib to install them. Each jar normally also contains a bunch of entities which need to be added to the default persistence unit when the application is loaded. Unfortunately for us, it seems that although the EJB 3.0 expert group imagined almost every possible scenario for packaging persistence units, there is no way of merging persistence units together.

Our original solution was to require the user to manually edit the webapp's persistence.xml to add the mapping files for the new module. This turns out to be quite cumbersome though, especially when you are running a number of sites on a single installation and each site has different modules installed (requiring a separate persistence unit for each).

There had to be a better way.

I had done some digging through the OpenJPA code already so I was fairly comfortable about implementing an OpenJPA specific solution - it would just be a matter of figuring out how to do it. Well, it turns out that OpenJPA's PersistenceUnitInfoImpl class has this great method called addMappingFileName which you can use to add the location of any mapping file on the classpath, just like you do with the <mapping-file/> xml element. All I had to do was create a custom version of Persistence.createEntityManagerFactory:

```
/**
 * Bootstrap an emf just the way we like it. This code is a snippet
 * of openjpa source with a few adjustments to use the mapping files
 * specified in figbird's modules instead of those in persistence.xml
 */
public EntityManagerFactory createEntityManagerFactory() {

    /* create persistence unit */
    PersistenceUnitInfoImpl pui = new PersistenceUnitInfoImpl();
    for (String mapping : config.getLists().get(
            "figbird.core.mappingFiles")) {
        pui.addMappingFileName(mapping);
    }

    PersistenceProductDerivation pd = new PersistenceProductDerivation();
    try {
```

```
        ConfigurationProvider cp = pd.load(pui, config);
        if (cp == null) {
            return null;
        }
         BrokerFactory factory = Bootstrap.newBrokerFactory(cp,
           pui.getClassLoader());
        return OpenJPAPersistence.toEntityManagerFactory(factory);
    } catch (Exception e) {
        throw PersistenceExceptions.toPersistenceException(e);
    }
}
```

In our case, installed modules add their mapping files to a list called figbird.core.mappingFiles when the webapp is started. This method gives us a lot of control over the entities in the persistence unit and allows different sites to have different persistence units even though they are running on the same code base.

You could be wondering why we didn't just use a separate persistence unit for each module. That would be possible in many cases, but for our system, the modules' entities are typically subclasses of the core entities and need to participate in queries and reuse a lot of code from the core module. This wouldn't be quite as easy if they were packaged into separate units.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.