APS OpenJPA Provider

User Guide

Version: 0.9.1

Author: Tommy Svensson

Table of Contents

1 APSJPAService

This provides JPA to services and applications. It has a slightly more OSGi friendly API than the org.osgi.service.jpa.EntityManagerFactoryBuilder. The APSOpenJPAProvider however returns an APSJPAService instance that also implements EntityManagerFactoryBuilder. For some reason I haven't figured out yet, it cannot be registered as a service with the EntityManagerFactoryBuilder interface! The bundle fails to deploy if that is done.

The provided service is using OpenJPA. The service works partly as an extender inspecting deployed bundles for a META-INF/persistence.xml file. When found this is read and some setup is done already there. The persistenceUnitName from the persistence.xml file is used to connect the client later with its configuration. When a JPA using bundle is shut down its JPA setup is automatically cleaned.

Here is an example of usage:

```
        private APSJPAEntityManagerProvider emp = null;
        ...
        private APSJPAEntityManagerProvider getEMP() {
            if (this.emp == null || !this.emp.isValid()) {
                DataSourceDef dsDef = this.dataSourceDefService.lookupByName(”MyDS");
                if (dsDef == null) {
                    throw new SomeException("Could not find an ’MyDs’ in
'persistence/datasources' configuration!");
                }
                Map<String, String> props = new HashMap<String, String>();
                props.put("javax.persistence.jdbc.user", dsDef.getConnectionUserName());
                props.put("javax.persistence.jdbc.password",
dsDef.getConnectionPassword());
                props.put("javax.persistence.jdbc.url", dsDef.getConnectionURL());
                props.put("javax.persistence.jdbc.driver",
dsDef.getConnectionDriveName());
                this.emp = this.jpaService.initialize(this.bundleContext,
”myPersistenceUnitName”, props);
            }
            return this.emp;
        }
        ...
        EntityManager em = getEMP().createEntityManager();
        em.getTransaction().begin();

        try {
            RoleEntity role = new RoleEntity(id);
            role.setDescription(description);
            em.persist(role);
            em.getTransaction().commit();
        }
        catch (RuntimeException re) {
            em.getTransaction().rollback();
            throw re;
        }
        finally {
            em.close();
        }
```

This code example handles the APSJPAService having been restared or redeployed. When `emp.isValid()` returns false then all you need to do is to call `jpaService.initialize(...)` again. The rest is just POJPA (Plain Old JPA :-)).

1.1 APIs

public *interface* **APSJPAService** [se.natusoft.osgi.aps.api.data.jpa.service] {

This service allows an JPA EntityManager to be gotten for a persistent unit name.

So why is this done this way ? Why is not an EntityManagerFactory returned?

The answer to that is that the EntityManagerFactory is internal to the service who is responsible for creating it and for closing it at sometime (stopping of bundle). The client only needs an EntityManager for which the client is responsible after its creation.

The creation of the EntityManagerFactory is delayed until the call to initialize(...). Creating the EMF along with the persistence provider at persistence bundle discovery would limit database connection properties to the persistence.xml file which is less than optimal to put it mildly. The whole point with the APS project is to provide a configured platform into which you can drop applications and they adapt to their surrounding. Not unlike what JEE does, but does it milder and more flexibly being OSGi and also provides application and service specific configuration with a web gui for editing configuration. Thereby providing database connection properties from clients allows clients more flexibility in how they want to handle that. The APSDataSourceDef service can for example be used to lookup a JDBC connection definition. The default provider implementation of this service uses OpenJPA which provides its own connection pooling.

APSJPAEntityManagerProvider initialize(BundleContext bundleContext, String persistenceUnitName, Map<String, String> props) throws APSResourceNotFoundException

Initializes and returns a provider from the specified properties.

Returns

A configured EntityManager.

Parameters

bundleContext - The context of the client bundle. It is used to locate its persistence provider.

persistenceUnitName - The name of the persistent unit defined in persistence.xml.

props - Custom properties to configure database, etc.

public *static interface* **APSJPAEntityManagerProvider** [se.natusoft.osgi.aps.api.data.jpa.service] {

Once you get this it is valid until the APSJPAService is stopped (which will happen if the service is redeployed!).

public boolean isValid()

Returns true if this instance is valid. If not call APSJPAService.initialize(...) again to get a new instance. It will be invalid if the APSJPAService provider have been restarted.

EntityManager createEntityManager()

Creates a new EntityManager. You are responsible for closing it!

Please note that the EntityManager caches all referenced entities. If you keep and reuse it for a longer time it can use more memory. For example at <a href='http://docs.jboss.org/ejb3/app-

server/tutorial/extended_pc/extended.html' http://docs.jboss.org/ejb3/app-server/tutorial/extended_pc/extended.html a it says that "Usually, an EntityManager in JBoss EJB 3.0 lives and dies within a JTA transaction". This indicates how long-lived the EntityManager should preferably be.

Returns

A configured EntityManager.

EntityManagerFactory getEntityManagerFactory()

Returns the underlaying entity manager factory. This will return null if isValid() return false!

}
