Using Hibernate Entity and Dao in Plugin

Sometime, you may want to have your own Hibernate entity and dao to access extra database table in your plugin. It is very simple to achieve that by adding the following file and class to your plugin.

Similar to development in Spring + Hibernate, a application context file is required for your plugin. In my sample plugin, I created a productsApplicationContext.xml as below

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-2.5.xsd
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">



    <bean id="productSessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource" ref="setupDataSource"/>
        <property name="mappingResources">
            <list>
                <value>/org/joget/sample/products/model/Products.hbm.xml</value>
             </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.show_sql">false</prop>
                <prop key="hibernate.format_sql">false</prop>
            </props>
        </property>
    </bean>



    <bean id="productsDao" class="org.joget.products.dao.ProductsDaoImpl">
        <property name="sessionFactory" ref="productSessionFactory" />
        <property name="localSessionFactory" ref="&amp;productSessionFactory"/>
    </bean>



</beans>
```

In the application context, I created 2 beans. Bean "productSessionFactory" is to initialize a session factory with the hibernate mapping file. Bean "productsDao" is to initialize the dao object of my sample plugin.

Next, we need a Hibernate Mapping file. In my sample plugin, it is /org/joget/sample/products/model/Products.hbm.xml. It mapped the POJO "org.joget.products.model.Product" with "valu_products" table.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.
net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class entity-name="Products" name="org.joget.products.model.Product" table="valu_products">
    <id column="id" name="id" type="string"/>
    <property column="name" name="name" type="string"/>
    <property column="description" name="description" type="string"/>
  </class>
</hibernate-mapping>
```

You need a utility class to initialized your application context and allow you to retrieve the bean object.

```
package org.joget.products;

import org.joget.apps.app.service.AppUtil;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AppContext {

    private static AppContext instance;
    private AbstractApplicationContext appContext;

    public synchronized static AppContext getInstance() {
        if (instance == null) {
            instance = new AppContext();
        }
        return instance;
    }

    private AppContext() {
        Thread currentThread = Thread.currentThread();
        ClassLoader threadContextClassLoader = currentThread.getContextClassLoader();
        try {
            currentThread.setContextClassLoader(this.getClass().getClassLoader());
            this.appContext = new ClassPathXmlApplicationContext(new String[]{"/productsApplicationContext.
xml"}, this.getClass(), AppUtil.getApplicationContext());
        } finally {
            currentThread.setContextClassLoader(threadContextClassLoader);
        }
    }

    public AbstractApplicationContext getAppContext() {
        return appContext;
    }
}
```

After you implemented your POJO and dao class, you should be able to use your dao in your plugin as following. Please refer to the attached sample plugin for POJO and dao implementation.

```
ProductsDao productdao = (ProductsDao) AppContext.getInstance().getAppContext().getBean("productsDao");

Product p = new Product();
p.setId("001");
p.setName("Product A");
p.setDescription("Product A Descpription");

productdao.addProduct(p);
```

In this KB:sample plugin, you are able to add, delete and list product by the following JSON API.

To add,

```
http://localhost:8080/jw/web/json/plugin/org.joget.products.ProductsApi/service?_action=add&name=Product
A&desc=Product A Descpription
```

To delete,

```
http://localhost:8080/jw/web/json/plugin/org.joget.products.ProductsApi/service?_action=delete&id=001
```

To list all products,

```
http://localhost:8080/jw/web/json/plugin/org.joget.products.ProductsApi/service?_action=list
```