HibernateTapestrySqueezer

NOTE: This is outdated information that applies only to Tapestry 4.

Tapestry 4 DataSqueezer for use with Hibernate

Withouth a DataSqueezer for your objects, tapestry will seriliaze a hibernate Pojo (assuming your pojo implements java.io.Serializable) and write this into html. For more information, look at the DataSqueezer. This can be quite an inefficient use of resources. To get around this problem, Tapestry lets you provide a DataSqueezer for your objects.

In general you need a generic hibernate method which can fetch objects based on a Class and Object Id. In this case we are using a basic Spring DAO to do this, org.springframework.orm.hibernate.HibernateTemplate provides this method. The Spring DAO here is injected using hivemind.

```
import org.apache.tapestry.util.io.SqueezeAdaptor;

/**
 * Hibernate DataSqueeze Adapter
 */
public interface IHibernateAdapter extends SqueezeAdaptor
{
        public void setDao(IDAO dao);
}
```

Next we implment the DataSqueezer (org.apache.tapestry.util.io.SqueezeAdaptor). When you register a new DataSqueezer, it should define a unique prefix. Tapestry already uses the following ones (Source Mindbridge on Mailing List) [b c d f l s t A B O S Z 0 1 2 3 4 5 6 7 8 9 -]. In this case we are using "C".

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.tapestry.services.DataSqueezer;

public class HibernateAdapter implements IHibernateAdapter
{
    protected final Log log = LogFactory.getLog(getClass());
        private final static String PREFIX = "C";
        private IDAO dao;

        public void setDao(IDAO dao)
        {
                this.dao = dao;
        }

        public IDAO getDao()
        {
                return dao;
        }

        public String getPrefix()
        {
                return PREFIX;
        }

        public Class getDataClass()
        {
                return IPersistent.class;
        }

        public String squeeze(DataSqueezer squeezer, Object data)
        {
                String name = data.getClass().getCanonicalName();
                Persistent p = (IPersistent) data;
                return PREFIX + name + ":" + p.getId();
        }

        public Object unsqueeze(DataSqueezer squeezer, String string)
        {
                // Strip the prefix
                string = string.substring(1);

                String[] strings = string.split("\\:");

                try
                {
                        Integer id = new Integer(strings[1]);
                        Class c = Class.forName(strings[0]);

                        // Get the object using the Dao
                        return getDao().getObject(c,id);
                }
                catch( Throwable t )
                {
                        // Handle error
                }

                return null;
        }
}
```

The interface IPersistent is used to identify these objects. In this case, it looks something like this:

```
/**
 * Implemented by Persistent Objects
 */
public interface IPersistent
{
    public Integer getId();
    public void setId(Integer id);
}
```

Next we need to register this by contributing it to a configuration point, everything else is Tapestry Magic:

```
   <service-point id="HibernateAdapter" interface="IHibernateAdapter">
     <invoke-factory>
       <construct class="HibernateAdapter">
         <set-object property="dao" value="spring:dao"/>
       </construct>
     </invoke-factory>
   </service-point>

  <contribution configuration-id="tapestry.data.SqueezeAdaptors">
    <adaptor object="service:HibernateAdapter"/>
  </contribution>
```