SpringHibernate

NOTE: This is outdated (circa 2005) information that applies only to Tapestry 4. For the current information on this topic, see

http://wiki.apache.org/tapestry/Tapestry5HowTos#Hibernate

Since this subject is coming often on the mailing list, I created this page for summarizing solutions.

Two approaches are possible for using hibernate with Tapestry: with Spring or write your own engine services to deal with sessions. The Spring approach offers an (almost) transparent session management and declarative transactions.

Hibernate 3 offers several major improvements over the 2.1. A preliminary support for Hibernate 3 and Spring is available at http://opensource.atlassian.com/projects/spring/browse/SPR-300

With Spring, the way to go is to use the "OpenSessionInViewFilter" which opens and closes hibernate sessions for you in a transparent way. These lines are required in the web.xml to enable the session management:

```
<filter>
  <filter-name>hibernateFilter</filter-name>
  <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>hibernateFilter</filter-name>
  <!-- put here your own path for your Tapestry url-pattern -->
  <url-pattern>/exec</url-pattern>
</filter-mapping>

<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

For the applicationContext.xml, my file is based on https://betterpetshop.dev.java.net/ example:

```
<!-- ========================= GENERAL DEFINITIONS ========================= -->
<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="location"><value>classpath:jdbc.properties</value></property>
</bean>

<!-- Message source for this context, loaded from localized "messages_xx" files -->
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basename"><value>messages</value><
/property>                                                                    </bean>

<!-- ========================= RESOURCE DEFINITIONS ========================= -->

<!-- Local DataSource that works in any environment -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName"><value>${jdbc.driverClassName}</value></property>
  <property name="url"><value>${jdbc.url}</value></property>
  <property name="username"><value>${jdbc.username}</value></property>
  <property name="password"><value>${jdbc.password}</value></property>
</bean>

<!-- Hibernate SessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource"><ref local="dataSource"/></property>
  <property name="mappingResources">
    <list>
      <!-- Write here your list of Hibernate files -->
      <value>hibernate/catalogue/Article.hbm.xml</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">${hibernate.dialect}</prop>
      <prop key="hibernate.show_sql">true</prop>
```

```
      <prop key="hibernate.c3p0.minPoolSize">${hibernate.c3p0.minPoolSize}</prop>
      <prop key="hibernate.c3p0.maxPoolSize">${hibernate.c3p0.maxPoolSize}</prop>
      <prop key="hibernate.c3p0.timeout">${hibernate.c3p0.timeout}</prop>
      <prop key="hibernate.c3p0.max_statement">${hibernate.c3p0.max_statement}</prop>
      <prop key="hibernate.generate_statistics">true</prop>
      <prop key="hibernate.cache.use_query_cache">true</prop>
    </props>
  </property>
</bean>

<!-- Transaction manager for a single Hibernate SessionFactory (alternative to JTA) -->
<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
  <property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>

<!-- ========================= BUSINESS OBJECT DEFINITIONS ========================= -->

<!-- Petclinic primary business object: Hibernate implementation -->
<bean id="WebServiceTarget" class="actualis.web.spring.WebManager"
      singleton="false">
  <property name="catalogue"><ref local="catalogueDAO"/></property>
  <property name="commande"><ref local="commandeDAO"/></property>
  <property name="clients"><ref local="clientsDAO"/></property>
</bean>

<bean id="poolTargetSource"
      class="org.springframework.aop.target.CommonsPoolTargetSource">
  <property name="targetBeanName"><value>WebServiceTarget</value></property>
  <property name="maxSize"><value>5</value></property>
</bean>

<bean id="businessObject"
      class="org.springframework.aop.framework.ProxyFactoryBean">
  <property name="targetSource"><ref local="poolTargetSource"/></property>
</bean>

<!-- Transactional proxy for the Petclinic primary business object -->
<bean id="WebService" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="target"><ref local="WebServiceTarget"/></property>
    <property name="transactionManager"><ref local="transactionManager"/></property>
    <property name="transactionAttributes">
      <props>
        <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
        <prop key="find*">PROPAGATION_REQUIRED,readOnly</prop>
        <prop key="load*">PROPAGATION_REQUIRED,readOnly</prop>
        <prop key="store*">PROPAGATION_REQUIRED</prop>
      </props>
  </property>
</bean>

<bean id="catalogueDAO" class="actualis.web.dao.CatalogueDAO">
  <property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>

<bean id="commandeDAO" class="actualis.web.dao.CommandeDAO">
  <property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>

<bean id="clientsDAO" class="actualis.web.dao.ClientsDAO">
  <property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>

<!-- ========================= WEB INTERCEPTORS DEFINITIONS ========================= -->
<bean id="urlMapping"
      class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="interceptors">
    <list>
      <ref bean="openSessionInViewInterceptor"/>
    </list>
  </property>
</bean>
```

```
<bean name="openSessionInViewInterceptor"
      class="org.springframework.orm.hibernate3.support.OpenSessionInViewInterceptor">
  <property name="sessionFactory"><ref bean="sessionFactory"/></property>
</bean>
```

The last trick is to get the lazy mode working. From one page to another, you are going to loose the session in the hibernate objects referenced, so you'll need to update them to reattach them to a session in order to perform operations on them.

One solution:

1. Create an ordinary many-to-one Cat to Kittens, lazy=true.
2. Load Cat in page ListCats, but don't touch Kittens.
3. In page ListCats, instantiate a new Page instance ViewCat.
4. Set the Cat property for page ViewCat (ie - viewCat.setCat(cat)).
5. Page ViewCat is activated and begins rendering.
6. Page ViewCat tries to list this Cat's Kittens, for example in a

Foreach as in .

Page rendering fails with a LazyInitializationException, even though the OpenSessionInViewFilter opened a session, since the Cat instance is not attached to the open session.

I've gotten around this by adding an attach() method to each of my service implementations. This method invokes getSession().lock(object, LockMode. NONE) to reassociate the specified object. I then manually invoke attach() to reassociate the detached objects for the given page. For example, in ViewCat. pageBeginRender(), I perform a getCatService().attach(getCat()).

Another solution consists in tweaking the Richard Hensley's DataSqueezer trick: http://thread.gmane.org/gmane.comp.java.tapestry.user/15398

He creates a custom DataSqueezer that tapestry will invoke automatically. Using this technique, a data object is squeezed into a short form (its id and classname) when tapestry builds a link or a hidden form field. When that url or form field comes back in a later request or form submission, tapestry invokes the datasqueezer to unsqueeze it. The unsqueeze operation re loads the dataobject from the database (or cache) and hence it is in the current session.

There is also a wiki article on the data squeezer: DataSqueezer