Tapestry3HiveMind

NOTE: This is outdated information that applies only to Tapestry 3. For the Tapestry 5 equivalent, see

http://tapestry.apache.org/ioc.html

Using HiveMind 1.0 with Tapestry 3.0 can be quite powerful. Although not as well integrated as Tapestry 4.0, you can still extract HiveMind services from a common registry and inject them into pages and components.

Note that HiveMind 1.1 uses a version of the Javassist library that is **not** compatible with Tapestry 3.0. Likewise, Tapestry 4.0 uses a version of Javassist compatible with HiveMind 1.1 ... but not compatible with HiveMind 1.0.

Adding the HiveMind Filter

Tapestry 3.0 does not include *any* logic for managing the HiveMind Registry; you will need to use the HiveMindFilter and wrap it around your Tapestry ApplicationServlet.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>MyApp</display-name>

  <filter>
    <filter-name>HiveMindFilter</filter-name>
    <filter-class>org.apache.hivemind.servlet.HiveMindFilter</filter-class>
  </filter>

   <filter>
    <filter-name>redirect</filter-name>
    <filter-class>org.apache.tapestry.RedirectFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>redirect</filter-name>
    <url-pattern>/</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>MyApp</servlet-name>
    <servlet-class>org.apache.tapestry.ApplicationServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>MyApp</servlet-name>
    <url-pattern>/app</url-pattern>
  </servlet-mapping>

  <filter-mapping>
    <filter-name>HiveMindFilter</filter-name>
    <servlet-name>MyApp</servlet-name>
  </filter-mapping>
</web-app>
```

Custom Engine Class

The eventual goal is for the HiveMind Registry to be the Global object. To accomplish this, we need a custom engine subclass for the application:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application
      PUBLIC "-//Apache Software Foundation//Tapestry Specification 3.0//EN"
      "http://jakarta.apache.org/tapestry/dtd/Tapestry_3_0.dtd">

<application name="MyApp" engine-class="com.examples.myapp.RegistryEngine" >

</application>
```

The engine class does two things: override how the Global object is created, and registry an OGNL PropertyAccessor for the Registry:

```
package com.examples.myapp;

import javax.servlet.http.HttpServletRequest;

import ognl.OgnlRuntime;

import org.apache.hivemind.servlet.HiveMindFilter;
import org.apache.tapestry.engine.BaseEngine;
import org.apache.tapestry.request.RequestContext;

public class RegistryEngine extends BaseEngine
{
    static
    {
        OgnlRuntime.setPropertyAccessor(Registry.class, new RegistryPropertyAccessor());
    }

    protected Object createGlobal(RequestContext context)
    {
        HttpServletRequest request = context.getRequest();

        return HiveMindFilter.getRegistry(request);
    }

}
```

Registry PropertyAccessor

An OGNL PropertyAccessor is an object that tells OGNL how to access properties of an object. It's useful for creating "synthetic" properties on objects. We want the services inside the Registry to appear as properties.

```
package com.myapp.examples;

import java.util.Map;

import org.apache.hivemind.Registry;

import ognl.ObjectPropertyAccessor;
import ognl.OgnlException;

public class RegistryPropertyAccessor extends ObjectPropertyAccessor
{

    public Object getProperty(Map context, Object target, Object name) throws OgnlException
    {
        String serviceId = (String) name;

        Registry registry = (Registry) target;

        return registry.getService(serviceId, Object.class);
    }
}
```

Injecting services into pages

In your page specification, you define a new property, using the <property-specification> element:

```
    <property-specification name="timeService" type="com.examples.myapp.services.TimeService">
        global['app.TimeService']
    </property-specification>
```

The initial value of the property, here defined as the body of the element, is the OGNL expression. We've set up the global object as the HiveMind Registry. The RegistryPropertyAccessor allows us to access services by their fully-qualified service id (you can actually use single or double quotes; for some reason, single quotes just *look right*).

We can access this property in Java code as well:

```
  public abstract TimeService getTimeService();
```

Injecting services into components

The process is the same, except that component's don't have a global property, only pages do. So we must navigate to the page, then get the global:

```
    <property-specification name="timeService">
        page.global['app.TimeService']
    </property-specification>
```