Tapestry5SpringIntegrationAlternative1

Integrating Spring & T5

At the time of writing, T5 (version 5.0.2) doesn't have built-in support for Spring, but this will certainly be included in one of the upcoming releases. In the meantime, you can use one of the following options to inject Spring beans into T5 components:

- Use Patch provided by Massimo in TAPESTRY-1284 https://issues.apache.org/jira/browse/TAPESTRY-1284
- Follow the next steps

Please note that this method was inspired from the solution presented by Massimo

Add the following methods to your module class

```
package foo.corp.services;

@Id("app")
public class AppModule {
  ...

    /**
     * Contributes the {@link ObjectProvider} provided by {@link Infrastructure#getObjectProvider()}
     * mapped to the provider prefix "spring".
     */
     @Contribute("tapestry.ioc.MasterObjectProvider")
     public static void contributeSpringToMasterObjectProvider(
                 MappedConfiguration<String, ObjectProvider> configuration,
                 @InjectService("Spring")
                 ObjectProvider spring) {
                 configuration.add("spring", spring);
     }


     /**
      * Build the actual provider for Spring beans
      * @param log logging facility
      * @return the {@link ObjectProvider} delegated to lookup objects
      */
     public static ObjectProvider buildSpring(Log aLog,
      @Inject("infrastructure:ApplicationGlobals") ApplicationGlobals aGlobals) {

         if (aGlobals == null) {
                 throw new RuntimeException(
                         "Cannot build Spring ApplicationContext without configurations");
         }

         return new SpringObjectProvider(aLog, aGlobals);
    }

}
```

Create the SpringObjectProvider class

The class listing is presented below:

```
package foo.corp.services;

import javax.servlet.ServletContext;

import org.apache.commons.logging.Log;
import org.apache.tapestry.ioc.ObjectProvider;
```

```
import org.apache.tapestry.ioc.ServiceLocator;
import org.apache.tapestry.services.ApplicationGlobals;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

/**
 * Spring based {@link ObjectProvider} which will serve Spring
 * beans through Tapestry5 IoC context.
 *
 * Adapted from Massimo Lusetti's (http://meridio.blogspot.com) implementation
 *
 *
 */
public class SpringObjectProvider implements ObjectProvider {
    private final Log _log;

    private ApplicationContext _context;

    private ServletContext _servletContext;

    public SpringObjectProvider(Log aLog, ApplicationGlobals aGlobals) {
        _log = aLog;
        _servletContext = aGlobals.getServletContext();
    }

    /* (non-Javadoc)
     * @see org.apache.tapestry.ioc.ObjectProvider#provide(java.lang.String, java.lang.Class, org.apache.
tapestry.ioc.ServiceLocator)
     */
    public <T> T provide(String expression, Class<T> objectType, ServiceLocatorlocator)
    {
        initialize();

        if (_context == null) {
            throw new RuntimeException("Spring ApplicationContext not configured");
        }
        try {
            Object service = _context.getBean(expression, objectType);
            // NOTE: If above succeed this cannot throw a ClassCastException
            return objectType.cast(service);
        }
        catch (BeansException are){
            String msg = "Impossible to locate the service " +
                         expression + " of type " + objectType.getCanonicalName() +
                         " from Spring application context";
            _log.error(msg, are);
            throw new RuntimeException(msg, are);
        }
    }


    private void initialize()
    {
        if (_context == null){
            _log.debug("Initializing Spring ApplicationContext from servlet context");
            _context = WebApplicationContextUtils.
                       getRequiredWebApplicationContext(_servletContext);
        }
    }

}
```

Update your web.xml

Now, update your web.xml configuration file to add the Spring context parameters and listener

```
<!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>Integration Test App 1</display-name>
    <context-param>
        <param-name>tapestry.app-package</param-name>
        <param-value>org.apache.tapestry.integration.app1</param-value>
    </context-param>

    <!-- Spring Stuff -->
    <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/applicationContext-*.xml</param-value>
    </context-param>

   <!-- Tap 5 -->
    <filter>
        <filter-name>app</filter-name>
        <filter-class>org.apache.tapestry.TapestryFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>app</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- Spring Open Session In View Pattern filter -->
    <filter>
      <filter-name>hibernateFilter</filter-name>
      <filter-class>
         org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
      </filter-class>
    </filter>

    <!-- Spring/Hibernate filter mappings -->
    <filter-mapping>
    <filter-name>hibernateFilter</filter-name>
    <url-pattern>/*</url-pattern>
    </filter-mapping>

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


</web-app>
```

Inject Spring beans

We are done! it's now time to inject some Spring beans. For illustration purpose, I am updating an example from Tapestry 5 integration test source code:

First, here is sample applicationContext-hibernate.xml config file. Only the relevant parts are presented (I am using Spring/Hibernate stack)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-2.0.xsd">

    <!-- Hibernate SessionFactory -->
        <bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.
AnnotationSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="configLocation" value="${hibernate.config}"/>
        <property name="hibernateProperties">
            <value>
                hibernate.dialect=${hibernate.dialect}
                hibernate.show_sql=${hibernate.show_sql}
                hibernate.query.substitutions=true 'Y', false 'N'
            </value>
        </property>
    </bean>

    ...

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

    <bean id="toDoDatabase" class="org.apache.tapestry.integration.app1.services.ToDoDatabaseHibernateImpl">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

</beans>
```

As you can see, the toDoDatabase is now mapped to the ToDoDatabaseHibernateImpl class.

And then, the ToDoList class:

```
    package org.apache.tapestry.integration.app1.pages;

    import java.util.List;

    import org.apache.tapestry.PrimaryKeyEncoder;
    import org.apache.tapestry.annotations.Component;
    import org.apache.tapestry.annotations.Inject;
    import org.apache.tapestry.corelib.components.Form;
    import org.apache.tapestry.integration.app1.data.ToDoItem;
    import org.apache.tapestry.integration.app1.services.ToDoDatabase;
    import org.apache.tapestry.util.DefaultPrimaryKeyEncoder;

    public class ToDoList
    {
        @Inject("spring:toDoDatabase")
        private ToDoDatabase _database;

        private ToDoItem _item;

        private DefaultPrimaryKeyEncoder<Long, ToDoItem> _encoder;

        @Component
        private Form _form;

        public List<ToDoItem> getItems()
        {
            return _encoder.getValues();
        }

        public ToDoItem getItem()
```

```
        {
            return _item;
        }

        public void setItem(ToDoItem item)
        {
            _item = item;
        }

        public ToDoDatabase getDatabase()
        {
            return _database;
        }

        public PrimaryKeyEncoder getEncoder()
        {
            return _encoder;
        }

        void onPrepare()
        {
            List<ToDoItem> items = _database.findAll();

            _encoder = new DefaultPrimaryKeyEncoder<Long, ToDoItem>();

            for (ToDoItem item : items)
            {
                _encoder.add(item.getId(), item);
            }
        }

        void onSuccess()
        {
            int order = 0;

            for (ToDoItem item : _encoder.getValues())
            {
                item.setOrder(order++);
                _database.update(item);
            }
        }

        void onSelectedFromAddNew()
        {
            if (_form.isValid())
            {
                ToDoItem item = new ToDoItem();
                item.setTitle("<New To Do>");
                item.setOrder(_encoder.getValues().size());

                _database.add(item);
            }
        }

        void onActionFromReset()
        {
            _database.reset();
        }
}
```

Note that the only change to this class the the @Inject annotation

Congrats! you are now using injected Spring beans in T5