Customizing Model-Glue by Overriding Framework Components

Posted At : January 11, 2010 2:23 PM | Posted By : Bob Silverberg
Related Categories: Model-Glue, ColdFusion

One of the cool features of Model-Glue version 3 is the ability to replace many of the components that make up the framework with your own version. This allows you to change the default behaviour of the framework (which can be useful) without having to alter any of the framework's internal code (which is a bad idea). Let's look at a situation in which we might want to customize Model-Glue and how we would do it.

Customizing Generated Code

Scaffolding is a feature of the framework that allows Model-Glue to generate working CRUD code for an object when you're using an ORM. Model-Glue 3.1 supports Transfer and Reactor, and Model-Glue 3.2 (coming soon) will also support ColdFusion 9's built-in ORM features. When you ask Model-Glue to do scaffolding for an object it will create event handlers and view templates for the operations that you request, which can include List, View, Edit, Commit and Delete. The code that is generated is based on cfcs that, by default, are found in ModelGlue/gesture/modules/scaffold/beans. The cfcs are called View.cfc, List.cfc, etc.

If we want to change the way the view template is generated for the List action, we can simply edit the file ModelGlue/gesture/modules/scaffold/beans/List.cfc, changing the code to reflect how we want our List view template to be generated.

But wait, didn't we discuss earlier that changing files within the framework itself is a Bad Thing©? No problem. Model-Glue allows you to override its own internal components by pointing to a component that belongs to your own application, which lives outside of Model-Glue's code base.

How Model-Glue's Components are Defined

All of the components that can be overridden can be found in Model-Glue's own Coldspring config file, which can be found in *ModelGlue/gesture/configuration/ModelGlueConfiguration.xml*. Let's take a look at a snippet from that file to see how we'd change the behaviour of our List scaffold:

```
<bean id="modelglue.scaffoldManager"

 class="ModelGlue.gesture.modules.scaffold.ScaffoldManager">

 <constructor-arg name="modelGlueConfiguration">

  <ref bean="modelglue.modelGlueConfiguration" />

 </constructor-arg>

 <constructor-arg name="scaffoldBeanRegistry">

  <map>

   <entry key="Commit"><ref bean="modelglue.scaffoldType.Commit" /></entry>

   <entry key="Delete"><ref bean="modelglue.scaffoldType.Delete" /></entry>

   <entry key="Edit"><ref bean="modelglue.scaffoldType.Edit" /></entry>

   <entry key="List"><ref bean="modelglue.scaffoldType.List" /></entry>

   <entry key="View"><ref bean="modelglue.scaffoldType.View" /></entry>

  </map>

 </constructor-arg>

 <property name="modelGlue">

  <ref bean="modelglue.ModelGlue" />

 </property>

</bean>



<bean id="modelglue.scaffoldType.List"

 class="coldspring.beans.factory.config.MapFactoryBean">

 <property name="SourceMap">

  <map>

   <entry key="class">

    <value>ModelGlue.gesture.modules.scaffold.beans.List</value>

   </entry>

   <event key="hasXMLGeneration"><value>true</value></event>

   <event key="hasViewGeneration"><value>true</value></event>

   <entry key="prefix"><value>List.</value></entry>

   <entry key="suffix"><value>.cfm</value></entry>

  </map>

 </property>

</bean>
```

The first Coldspring bean definition we see, for a bean called *modelglue.scaffoldManager* controls all of the scaffolding behaviour for the framework. Looking at that bean definition we can see that a number of other beans are registered as scaffold beans by putting them into the *scaffoldBeanRegistry* map. So we don't actually have to replace that bean (the *modelglue.scaffoldManager* bean), we just have to replace the bean for the List action, which we can see points to a bean definition with an id of *modelglue.scaffoldType.List*. That bean definition is the second one in the code snippet above. So, how do we tell Model-Glue that when it creates a List scaffold we want it to use our own List.cfc component, rather than the default ModelGlue/gesture/modules/scaffold/beans/List.cfc? We have to*override* the bean definition for *modelglue.scaffoldType.List*.

Overriding Model-Glue's Bean Definitions

We can override the bean definition for *modelglue.scaffoldType.List* by simply copying the code from *ModelGlueConfiguration.xml* into our application's own local Coldspring.xml file. After doing that, our local Coldspring.xml file will contain the following code (in addition to all of the other beans in there already):

```
<bean id="modelglue.scaffoldType.List"
 class="coldspring.beans.factory.config.MapFactoryBean">
 <property name="SourceMap">
  <map>
   <entry key="class">
    <value>ModelGlue.gesture.modules.scaffold.beans.List</value>
   </entry>
   <event key="hasXMLGeneration"><value>true</value></event>
   <event key="hasViewGeneration"><value>true</value></event>
   <entry key="prefix"><value>List.</value></entry>
   <entry key="suffix"><value>.cfm</value></entry>
  </map>
 </property>
</bean>
```

What Have We Accomplished?

All we've done at this point is to tell Model-Glue (or more accurately Coldspring) that when it needs to find the bean definition for *modelglue.scaffoldType.List* it should use the definition in our local Coldspring.xml file, rather than the one in *ModelGlueConfiguration.xml*. But the bean definition is still the same, so we've overridden the bean definition but we haven't changed the behaviour.

Changing Behaviour

How do we tell Model-Glue to use our version of List.cfc rather than ModelGlue/gesture/modules/scaffold/beans/List.cfc? We just change the value of the *class* key in our local *modelglue.scaffoldType.List* bean definition. The new version will look something like this:

```
<bean id="modelglue.scaffoldType.List"
 class="coldspring.beans.factory.config.MapFactoryBean">
 <property name="SourceMap">
  <map>
   <entry key="class"><value>myApp.scaffold.beans.List</value></entry>
   <event key="hasXMLGeneration"><value>true</value></event>
   <event key="hasViewGeneration"><value>true</value></event>
   <entry key="prefix"><value>List.</value></entry>
   <entry key="suffix"><value>.cfm</value></entry>
  </map>
 </property>
</bean>
```

Where my custom version of List.cfc lives in */myApp/scaffold/beans/*, which is a folder in my application. It's as simple as that. Now, when I ask Model-Glue to scaffold an object, the event handler and view template for the List action will be generated based on code in my*myApp.scaffold.beans.List.cfc* component, rather than the default *ModelGlue.gesture.modules.scaffold.beans.List.cfc* component.

We can further customize the List behaviour by changing other keys in the *modelglue.scaffoldType.List* bean definition. For example, the default filename that will be generated for a List action for a Product object would be *List.Product.cfm*. This is because the *prefix* is defined as "List." and the *suffix* is defined as ".cfm". If we want to generate a file called *dspProductList.cfm*, we could further change our bean definition like so:

```
<bean id="modelglue.scaffoldType.List"
 class="coldspring.beans.factory.config.MapFactoryBean">
 <property name="SourceMap">
  <map>
   <entry key="class"><value>myApp.scaffold.beans.List</value></entry>
   <event key="hasXMLGeneration"><value>true</value></event>
   <event key="hasViewGeneration"><value>true</value></event>
   <entry key="prefix"><value>dsp</value></entry>
   <entry key="suffix"><value>List.cfm</value></entry>
  </map>
 </property>
</bean>
```

You can use this technique to override any of the bean definitions in the ModelGlueConfiguration.xml file. For example, if you want to support SES urls you can override Model-Glue's default UrlManager by placing a bean definition for *modelglue.UrlManager* into your local Coldspring config. You can also override the default ValidationService that Model-Glue uses for Generic Database Messages (GDMs) by placing a bean definition for *modelglue.ValidationService* into your local Coldspring config. You could use that, for example, to enable the use of ValidateThis to provide validations for GDMs. I am doing that in conjunction with the new CF9 ORM Adapter in an application right now, and I plan to write a more detailed post about it in the near future.