Faking ColdFusion Component Types (and a Quick Way to Confuse ColdFusion)

Posted At : October 14, 2010 2:39 PM | Posted By : Bob Silverberg
Related Categories: MXUnit, ColdFusion

MightyMock (MM) is a mocking framework for ColdFusion that is bundled with MXUnit. It allows you to create mocks, which are *fake* instances of ColdFusion components, for use in unit tests. More information on what MightyMock is and how you can use it to create awesome unit tests can be found on the MXUnit Wiki.

One thing that MM allows you to do is to create *typesafe* mocks, which are mock objects that ColdFusion will recognize as being of a specific type. Why might you need to do that? Let's say you have a setter defined in a component into which you want to pass your mock object, and that setter expects an argument of a certain type. For example, consider this setter in a Customer.cfc component:

```
<cffunction name="setValidator" access="public" returntype="void">

 <cfargument name="validator" type="model.util.validator" />

 <cfset variables.validator = arguments.validator />

</cffunction>
```

If we want to create a mock object and pass it into that method, it needs to have a type of *model.util.validator* or ColdFusion will throw an error as soon as we call *setValidator()*. MM allows us to create such a mock using the optional second parameter of the *mock()* method that is available in MXUnit. The code to create a test for our Customer component using a typesafe Validator mock would look something like this:

```
<cffunction name="someTestThatNeedsTheValidatorInCustomerToReturnTrue" access="public" returntype="void">

 <cfset mockValidator = mock("model.util.validator","typesafe") />

 <cfset mockValidator.validate("{*}").returns(true) />

 <cfset customer = new model.customer() />

 <cfset customer.setValidator(mockValidator) />

 <!--- call a method on the customer object that

  expects the composed validator object to return true --->

 <!--- assert something here --->

</cffunction>
```

For the purposes of this post I'm not going to get into how and why we're using the mock. That may or may not be evident to you from the code. I simply wanted to point out that MM can create a mock of a particular type using the*typesafe* argument. There's just one issue with the current implementation of *typesafe* in MM: the component*model.util.validator* must exist. The way MM works is that it will create an instance of that component and then *empty it out*, so if *model.util.validator* doesn't exist you'll get an error when you try to create the mock.

I wanted to find a way around this limitation, as often one is using a mock precisely because the component being mocked has not been written yet. Sure, you can just create an empty component and you're good to go, but I thought that I could find a way around that requirement. I actually got this idea from a presentation that I saw at CFUnited by Elliott Sprehn, and I wanted to try it out. It turned out to be incredibly simple, but also had a side effect that could be very dangerous.

It turns out that you can change the metadata of a component simply by setting it. What do I mean by that? Here's an example, using the same unit test as above, creating a *typesafe* mock manually:

```
<cffunction name="someTestThatNeedsTheValidatorInCustomerToReturnTrue" access="public" returntype="void">

 <cfset mockValidator = mock() />

 <cfset getMetadata(mockValidator).name = "model.util.validator" />

 <cfset mockValidator.validate("{*}").returns(true) />

 <cfset customer = new model.customer() />

 <cfset customer.setValidator(mockValidator) />

 etc...

</cffunction>
```

Notice what I did there? Rather than using the *typesafe* argument to the *mock()* method, I just created a simple typeless mock. Then, by setting *getMetadata(mockValidator).name* to *model.util.validator*, I changed the actual metadata of the component. Now, even though my component is actually an instance of*mxunit.framework.mightymock.MightyMock*, ColdFusion thinks it's an instance of *model.util.validator*, and I can pass it into my setter. Pretty cool, eh? So what about the side effect?

It turns out that ColdFusion caches this metadata, so after making that initial change to that one instance of mockValidator, any mock that I create will have a type of *model.util.validator*. Of course I can set it back, which is what I may end up doing if this technique finds its way into MightyMock, but until I do that the changes to the metadata will persist. Any intance of*mxunit.framework.mightymock.MightyMock* will be seen by ColdFusion as an instance of *model.util.validator*. So where does the *Quick Way to Confuse ColdFusion* part come in?

When I was experimenting with this technique at one point I tried overwriting the *extends* metadata of the mock. A mock, which, as I mentioned already, is an instance of *mxunit.framework.mightymock.MightyMock*, does not extend anything. This means that it in fact extends the base ColdFusion component, which is *WEB-INF.cftags.component*. The test in which I tried overwriting the extends metadata looked like this:

```
<cffunction name="someTestThatNeedsTheValidatorInCustomerToReturnTrue" access="public" returntype="void">

 <cfset mockValidator = mock() />

 <cfset getMetadata(mockValidator).extends.name = "model.util.validator" />

 etc...

</cffunction>
```

What do you reckon that code did? That's right, it overwrote the metadata for the base ColdFusion component. Now every single component that does not extend anything looks like it extends *model.util.validator*. In addition to that, even components that do extend something still end up looking like they extend *model.util.validator*at the top of their inheritance hierarchy. This broke MXUnit immediately as it expects arguments of type *WEB-INF.cftags.component*. What else this might break would depend on the code that you're writing, but considering that it does change the inheritance hierarchy of all ColdFusion components, it seems like a pretty dangerous thing to do. Sure it's cool, but it's not something I'd recommend doing.