CFC Instantiation

Posted At : October 1, 2007 6:25 AM | Posted By : Steve
Related Categories: ColdFusion

I was looking for a good introduction to instantiating a CFC to link to from a blog entry. I couldn't find one that covered what I wanted, so I decided I would just write one myself.

CreateObject()

A CFC can be instantiated a number of ways, but CreateObject is the most common.

Here is the most basic syntax for instantiating a CFC:

```
<cfset MyComp = CreateObject("component","MyComp")>
```

This code instantiates MyComp.cfc from the same directory as the current file (or from the root of that site). The variable "MyComp" holds a reference to the instantiated component. Unlike traditional ColdFusion variables that only hold data, this variable actually has the logic (think, "the code") from the component as well.

Significantly, it holds the logic at the time the component is instantiated. This means that if the code in the file changes, that change won't be reflected in the variable unless it is instantiated again.

init()

I will often want to pass some data and/or perform some logic when I instantiate a component. By convention, the init method is used for this purpose (this could properly be called a "psuedo-constructor").

A method in a CFC looks just like a UDF in tradition ColdFusion code:

```
<cffunction name="init" access="public" returntype="any" output="no">
   <cfreturn this>
</cffunction>
```

The "this" variable is a reference to the component (the CFC) itself. The "this" variable generally should not be used for any purpose other than returning from the init() method, but must be used there.

Here is the basic code to call the init() method:

```
<cfset MyComp = CreateObject("component","MyComp")>
<cfset MyComp.init()>
```

As you can see, the CreateObject returns a reference to the component to the "MyComp" variable. Then I can call a method on the component referenced by the "MyComp" variable. Note that while the "MyComp.cfc" component is referenced by a variable of the same name, this is by convention rather than necessity.

Since the init() method is acting in a special capacity (that of a psuedo-constructor), it is generally called using method-chaining. That is, that it is called directly against the results of the CreateObject function.

```
<cfset MyComp = CreateObject("component","MyComp").init()>
```

This code does the same thing as the above code, but ensures that no code can be placed between the CreateObject call and the call to the init method.

Arguments

I will often want to pass arguments into my components as I instantiate them. For example, I need to pass in my datasource to the component. In order to do so, I first need to set that up in the init method:

```
<cffunction name="init" access="public" returntype="any" output="no">
   <cfargument name="datasource" type="string" required="yes">

   <cfset variables.datasource = arguments.datasource>

   <cfreturn this>
</cffunction>
```

As you can see, I copied the datasource variable from arguments scope to variables scope. This ensures that it can be used from other methods in the component. In a .cfm file, variables scope persists for the life of that file (one request). In a CFC, the the variables scope persists with the component. If the component is stored in a persistent scope, then it will last as long the referring variable.

The calling code could reference arguments ordinally or by name. To pass the argument in ordinally (that is, in the order specified by the cfargument tags), use this syntax:

```
<cfset MyComp = CreateObject("component","MyComp").init("mydsn")>
```

In order to pass the arguments in by name, use this syntax:

```
<cfset MyComp = CreateObject("component","MyComp").init(datasource="mydsn")>
```

Although this example passes in a string argument, another component can (and often should) be passed in as an argument to another component.

Application Scope

Instantiating a component can be somewhat resource intensive. The code for most components is relatively static and therefore doesn't need to be changed on every request.

As such, it often makes sense to store components in Application scope and only initialize them as needed.

Changing to Application scope itself is very simple:

```
<cfset Application.MyComp = CreateObject("component","MyComp").init("mydsn")>
```

The next step is to make sure that the component only gets instantiate if it doesn't already exist or if I manually reload the component.

```
<cfif isDefined("URL.reinit") OR NOT isDefined("Application.init")>

   <!--- load and init mycomp --->
   <cfset Application.MyComp = CreateObject("component","MyComp").init(datasource="mydsn")>

   <cfset Application.init = true>

</cfif>
```

The Application.init variable is just an indicator that the code inside the conditional has executed.

This approach to loading components is common and very effective for using components as services. If your application grows substantially, you may run into limitations of this approach, but should

provide a good foundation approach and one that should be easy to adapt to a more robust approach when and if one becomes necessary.