AppLoader: Loading Application-Scoped Components

Posted At : October 1, 2007 12:57 PM | Posted By : Steve
Related Categories: Frameworks, ColdFusion

Early in my use of Application-scoped CFCs, I realized that I would have to have some mechanism to reload them when I changed the code.

I started with code that should look familiar to most people using CFCs:

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

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

   <!--- load and init yourcomp --->
   <cfset Application.YourComp = CreateObject("component","YourComp").init(mycomp)>

   <cfset Application.init = true>
</cfif>
```

If you aren't already familiar with the topic, you can read an **introduction to CFC instantiation**.

As my application grew in complexity, so did my instantiation code:

```
<cfif isDefined("URL.reinit") OR NOT isDefined("Application.init")>
   <!--- load and init DataMgr --->
   <cfset Application.DataMgr = CreateObject("component","DataMgr").init("mydsn","MSSQL")>

   <!--- load and init SessionMgr --->
   <cfset Application.SessionMgr = CreateObject("component","SessionMgr").init("Session")>

   <!--- load and init users --->
   <cfset Application.Users = CreateObject("component","Users").init(DataMgr)>

   <!--- load and init permissions --->
   <cfset Application.Permissions = CreateObject("component","Permissions").init(DataMgr,Application.Users)>

   <!--- load and init security --->
   <cfset Application.Security = CreateObject("component","Security").init(DataMgr,SessionMgr,Users)>

   <cfset Application.init = true>
</cfif>
```

In practice, this would actually be a very short example but hopefully demonstrative of basic situation.

Despite the growing length of the code, this actually works well overall.

The major problem that I ran into with the approach however, was the lack of targetted reload. By that I mean that I would use the "reinit" variable in the URL, but I couldn't re-instantiate just one component with this code.

So, I could add a few conditionals to take care of this.

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

   <cfif ListFindNoCase(URL.reinit,"DataMgr")>
      <!--- load and init DataMgr --->
      <cfset Application.DataMgr = CreateObject("component","DataMgr").init("mydsn","MSSQL")>
```

```
   </cfif>

   <cfif ListFindNoCase(URL.reinit,"SessionMgr")>
      <!--- load and init SessionMgr --->
      <cfset Application.SessionMgr = CreateObject("component","SessionMgr").init("Session")>
   </cfif>

   <cfif ListFindNoCase(URL.reinit,"Users")>
      <!--- load and init users --->
      <cfset Application.Users = CreateObject("component","Users").init(DataMgr)>
   </cfif>

   <cfif ListFindNoCase(URL.reinit,"Permissions")>
      <!--- load and init permissions --->
      <cfset Application.Permissions = CreateObject("component","Permissions").init(DataMgr,Application.Users)>
   </cfif>

   <cfif ListFindNoCase(URL.reinit,"Security")>
      <!--- load and init security --->
      <cfset Application.Security = CreateObject("component","Security").init(DataMgr,SessionMgr,Users)>
   </cfif>

   <cfset Application.init = true>

</cfif>
```

The conditionals add a lot of code but doesn't completely solve the problem. I also need to add code to reload all of the components if needed. More significantly, however, is another problem.

If I refresh the DataMgr component in the example, the Users and Security components will still be using the previous instantiation of the DataMgr component. I need a way to make sure that every time I initialize a component I also reinitialize any component to which that component is passed.

Enter AppLoader

I created a component to handle just this problem. I takes an XML definition of my component structure and manages targetted reloading of the those components.

```
<?xml version="1.0"?>
<site>
   <components>
      <component name="DataMgr" path="DataMgr">
         <argument name="datasource" arg="datasource" />
         <argument name="database" arg="dbtype" />
      </component>
      <component name="SessionMgr" path="SessionMgr">
         <argument name="scope" arg="SessionScope" />
      </component>
         <component name="Users" path="Users">
         <argument name="DataMgr" component="DataMgr" />
      </component>
      <component name="Permissions" path="Permissions">
         <argument name="DataMgr" component="DataMgr" />
         <argument name="Admins" component="Admins" />
      </component>
      <component name="Security" path="Security">
         <argument name="DataMgr" component="DataMgr" />
         <argument name="SessionMgr" component="SessionMgr" />
      </component>
```

```
    </components>
</site>
```

To instantiate the components with AppLoader, use the following code:

```
<cfparam name="URL.reinit" default="false">
<cfset Loader = CreateObject("component","AppLoader").init(XmlFilePath=ExpandPath("components.xml"))>

<cfinvoke component="#Loader#" method="setArgs">
   <cfinvokeargument name="datasource" value="mydsn">
   <cfinvokeargument name="dbtype" value="MSSQL">
   <cfinvokeargument name="SessionScope" value="Session">
</cfinvoke>

<cfinvoke component="#Loader#" method="load">
   <cfinvokeargument name="refresh" value="#url.reinit#">
</cfinvoke>
```

The setArgs methods allows you to pass arguments into AppLoader to be reference in the arg attribute of the argument tag.

The load method tells AppLoader to load the components. AppLoader will instantiate any component that doesn't already exist or any component that is indicated in the refresh argument (or all of them if the argument is true) as well as any components that depend on it (continuing down any number of levels).

The XML is pretty basic. The component element has two attributes:

- name: The name of the Application-scoped variable that will hold the reference to the component.
- path: The path to the component that you want to instantiate (the second argument of the CreateObject() function.

The component element has an argument element for each argument of the init() method of the component. The argument element has a few attributes:

- name: The name of the argument (AppLoader passes arguments in by name)
- component: If a component is being passed in, use this to indicate the name of the component to be passed in.
- arg: Use this attribute to indicate a value passed in to AppLoader through the setArgs method.
- value: Use this attribute to pass in a static value to the argument.

While the initial code examples require that the components be listed in order of dependency, the XML passed in to AppLoader can be in any order and it will still accurately determine the order of dependencies.

If you are using components as Application-scoped services, then AppLoader can simplify your development and allow for targetted reloading of your components.

If you have more complicated needs, then you might check out **ColdSpring** or **LightWire** (neither of which had I heard of when I first started AppLoader a few years ago).

Feel free to download **AppLoader** and use it as you see fit. It is open-source and free for any use.