OO Principles: Composition

Posted At : July 30, 2009 11:15 AM | Posted By : Steve
Related Categories: ColdFusion, OO Principles

I don't "do OO" development in ColdFusion. I'm starting with that statement not to spark another debate about whether to use OO in ColdFusion, but rather to clarify that while this post is about a principle of object oriented development, you don't need to "Do OO" in order to learn, use, and benefit from composition.

In the last "**OO Principles**" entry, I talked about **encapsulating CFCs**. The example that I use was the need to have a datasource in a component. It should be clear from that entry that you could pass in more methods as well.

Just like in the example, I pass in a datasource to my **DataMgr** component (for our purposes, it only matters that this is a component that only requires a datasource argument).

```
<cfset Application.DataMgr = CreateObject("component","DataMgr").init(datasource=request.dsn)>
```

For most of my components, I don't want to access the database directly, but rather use DataMgr to do so. I could instantiate DataMgr from within my component. In the case of DataMgr, this would be relatively easy (as it only has one argument), but I would still have a separate instance of DataMgr for each component in my site. For more complicated components (requiring several arguments), however, this quickly becomes untenable.

I already have DataMgr instantiated in Application scope, so I may as well use that. The first temptation here may be to just reference Application.DataMgr from within my component. As we have seen, however, it is to our **advantage to use encapsulation**.

If I want to create a Users component that takes DataMgr as an argument, I can use the following "init" method:

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

<cfset variables.DataMgr = arguments.DataMgr>

<cfreturn This>
</cffunction>
<cfset Application.DataMgr = CreateObject("component","DataMgr").init(request.dsn)>
<cfset Application.Users = CreateObject("component","Users").init(Application.DataMgr)>
```

The only difference from the previous example is that I have passed in a component instead of a string. Now I have composed DataMgr into Users.

This becomes even more advantageous as you have several components and realize that one needs to get data from another. For example, if you have an Organizations component from which your Users component may need data, you can have your Users component additionally compose your Organizations component:

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

<cfset variables.DataMgr = arguments.DataMgr>
<cfset variables.Organizations = arguments.Organizations>

<cfreturn This>
```

```
</cffunction>
<cfset Application.DataMgr = CreateObject("component","DataMgr").init(request.dsn)>
<cfset Application.Organizations = CreateObject("component","Organizations").init(Application.DataMgr)>
<cfset Application.Users = CreateObject("component","Users").init(Application.DataMgr,Application.Organizations)>
```

Now your Users component can interact with your Organizations component without having to know how it was instantiated.

Beyond keeping your code encapsulated, composition also helps you see how different components related to each other. Using composition to tie your components together will also set you up for future growth.

Using composition in this way can open up new challenges (such as circular dependencies) as well as new opportunities for organization and automation (using a Dependency Injection Engine, for example). If you are using components, however, I would highly recommend using composition over breaking encapsulation.