OO Principles: Encapsulation and Decoupling

Posted At : July 9, 2009 10:30 AM | Posted By : Steve
Related Categories: 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 encapsulation and decoupling.

This will be the first in a few entries about how to take advantage of OO principles even if you aren't using OO.

Encapsulation is hiding the inner workings of a software component behind a defined interface. Decoupling is ensuring that two different software components are not tightly dependent on one another.

I'm covering these two principles together because, for me, they basically cover the same concept. So, I will now colloquially use the term "encapsulation" to cover both concepts. After all, we're trying to get some work done here, not pass a computer science class.

The basic idea here is that a module of code (CFC, UDF, or Custom Tag) should not know anything about the outside world except what it is explicitly told and should not expose its internals to the outside world except by way of its coded interface.

Here are some rough guidelines to know if your code is encapsulated:

- You should never reference a shared scope variable from within a module (again, defined here as CFC, UDF, or Custom Tag).
- You should never set a shared scope variable from within a module.
- Custom Tags should never set CALLER scope unless the name of the variable to set has explicitly been passed in to the custom tag as an attribute.

The big advantage of well encapsulated code is that it is very clear where and how it interacts with other parts of the system. Keeping all code well encapsulated helps reduce the chances that changes in one part of the system will cause problems in other parts of the system.

This also opens up opportunities for reuse. When every module of code is dependent on environmental variables (and/or creates them), it is difficult to find way to use that module outside of its original purpose. Many modules of code do not, of course, lend themselves to reuse, but this is helpful for those that do (and makes it easier to tell the difference).

Encapsulated code is more easily testable than non-encapsulated code. If your module is encapsulated, you can test it in isolation by providing different values, rather than have to test the whole system. You can even use existing unit testing frameworks to test your code.

This approach will also allow you to change implementation decisions. For example, let's say you want to nicely formatted weather information. If you create a custom tag that looks for a URL variable to determine the zip code for the weather, then it won't work on pages that don't have that information in the URL. If the outside code decides to pass that information in from a Form (or in session data) then the custom tag will no longer work. If, on the other hand, you have a zipcode attribute for the custom tag then the tag need not care where that information comes from.

```
<cf_MyWeatherData zipcode="#URL.zip#">
```

Now, if you decide on your page to get the zip from session scope instead, you just change the code calling the custom tag.

```
<cf_MyWeatherData zipcde="#Session.Zip#">
```

You could even call it different ways on different pages.

So, even if you aren't doing Object Oriented Programming, you can still benefit from encapsulation. If I have missed anything or gotten anything wrong, please let me know in the comments.

More Reading:

- **Introduction to Object Oriented Programming**
- **Thought for the Day: Encapsulation**
- **Why you should, and should not, break encapsulation in methods**