OO Principles: Separation of Concerns

Posted At : July 14, 2009 9:15 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 separation of concerns.

In order for separation of concerns to work, your code really needs to be well **encapsulated** first.

What

Separation of concerns dictates that different modules of code overlap in functionality as little as possible. For me, this means that I have .cfm templates for output and .cfc components for logic and data retrieval.

Why

Having different types of operations in different files (output in .cfm and logic in .cfc, for example) it allows me to have different people working on different types of work for the same project at the same time. This allows me to write your display code only knowing what data I will get from my CFC. I can then first write my CFC to give data in the correct format and then go back and have it figure the data correctly while the display code is being written (see **prototyping**).

Once I had certain kinds of operations in certain files, the underlying logic of my operations became clearer. When logical operations and output are mixed together, it is very difficult to even see where reuse makes sense, let alone implement it. Once these sorts of operations are separated, opportunities for reuse become both more clear and easier to implement.

The biggest advantage, for me, has been that it allows me to concentrate more effectively on my code. For example, when my logic code is isolated from my output code, it is easier for me to follow it and concentrate on the logical operations. Similarly, when I am looking at output code that doesn't have any logical operations in it; I can more clearly see the output operations and more easily work with them.

How

When I decided that I would move toward separation of concerns, I ended up taking it in stages. This worked pretty well for me.

First I added a comment to all of my .cfm pages, saying "Output Here" and I put all of my queries and logic above that point and all of my output below it. This allowed me to better concentrate on my code, but hardly allowed for reuse or parallel development.

Once CFCs were available, I put all of my queries and logic in .cfc files and I called the methods I needed above my output comment and kept my output below that line. In addition to making it easier to concentrate on my code, this approach allowed me to take advantage of parallel development (though I rarely did) as well as find (and take advantage of) opportunities for reuse (which I did).

Some people deride that approach as "spaghetti with meatballs". I recommend ignoring such comments. If we wait to write code until our approach is perfect, we won't ever get anywhere.

That worked for a long time, but then I decided that I would rather my .cfm file be more purely output oriented. Doing this allowed me to have HTML people open the file without worrying that they would mess up my data retrieval. So, I started using **page controllers** to call my service components and this works quite well. While this extra layer often isn't needed, it can provide a good translation tool between your model (logic and data) and your view (output). Isolating these translational operations can provide the same benefits as described above.

So, now my CFCs (with the exception of layout components that act basically like .cfm files) can query

the database and contain any required logic. My .cfm files only contain output and very simple loops and conditionals (no "and" or "or" statements in my cfif tags). My page controllers make data available to my .cfm files from my service components and perform any translations between the two.

This works well for me, but this pattern can be taken further to a full multi-tiered environment if your needs dictate that approach. Most important, however, is to understand the development benefits of separation of concerns - even if you aren't "doing OO".