How I Use Transfer - Part VIII - My Abstract Transfer Decorator Object - Simple Methods

Posted At : July 21, 2008 12:11 PM | Posted By : Bob Silverberg
Related Categories: OO Design, How I Use Transfer, ColdFusion, Transfer

In the past two posts in this series about Transfer (an ORM for ColdFusion) I discussed how I have implemented my Abstract and Concrete Gateways. As you may recall from Part II, that leaves my AbstractTransferDecorator Object as the final abstract object in my model. There is a lot of code in this object, and I plan to provide a lot of explanation, so I'm going to break this up into a few posts.

My AbstractTransferDecorator acts as a base object for all of my Concrete Decorators, and, because most of my Business Objects are created for me by Transfer, it really acts as an Abstract Business Object. For a basic overview of what a decorator is, and how one is used with Transfer, check out this page from the Transfer docs.

I have blogged about this object a few times before, but the posts in this series will, for now, supercede all of those previous posts, as this object continues to change. In fact, this object has a few methods and techniques the design of which I'm not crazy about right now. There will definitely be changes coming, at some point down the road.

I think it's a good idea to start at a high level, so here is a description of the various methods in my AbstractTransferDecorator:

- **configure()** - This method is called by Transfer after creating an instance of a Transfer Object. It's basically a decorator's version of the standard Init().
- **save()** - This allows the object to save itself.
- **delete()** - This allows the object to delete itself.
- **onNewProcessing()** - This allows the object to perform actions when a new instance of the object is about to be persisted to the database.
- **populate()** - This is used to fill a Transfer Object, and its related objects, with data. It's kind of like a setMemento() method, if you're familiar with that terminology.
- **copyToStruct()** - This allows the object to return a simple struct that contains keys for all of its properties. It's kind of like a getMemento() method, if you're familiar with that terminology.
- **validate()** - This contains generic validation routines that are shared by all Business Objects.
- **getValidations()** - This returns an array of validation rules that apply to the Business Object.
- **addValidation()** - This is a helper method to make defining validations simpler.

To keep this post reasonably short, we'll address the first four items, starting with configure():

```
<cffunction name="configure" access="private" returntype="void" hint="I am like init() but for decorators">

  <cfset variables.myInstance = StructNew() />

  <cfset variables.myInstance.CleanseInput = true />

 </cffunction>
```

The Configure method will be called by Transfer whenever a new instance of the object is created. This behaviour is built into Transfer itself. In here I create a struct to hold any private variables that the Business Object may need, and I set a default value of "true" for CleanseInput. CleanseInput is used by the Populate() method to determine whether or not to cleanse HTML from an object's properties prior to persisting it. This is a technique that I'm still kind of on the fence about.

The save() and delete() methods are both short and sweet:

```
<cffunction name="save" access="public" output="false" returntype="void" hint="Calls Transfer.save().">

  <cfset getTransfer().save(this) />

 </cffunction>



 <cffunction name="delete" access="public" output="false" returntype="void" hint="Calls Transfer.delete().">

  <cfset getTransfer().delete(this) />

 </cffunction>
```

As you can see, save() just calls getTransfer().save(), and delete() just calls getTransfer().delete(). I like this approach as it allows me to minimize references to Transfer in my service layer. I also personally prefer myObject.save() to myService.save(myObject).

onNewProcessing() is called by the service layer prior to saving a new object. Because most objects don't require this functionality this method remains empty in the AbstractDecorator:

```
<cffunction name="onNewProcessing" access="public" output="false" returntype="void" hint="Extra processing required for a new record.">

  <cfargument name="args" type="any" required="true" />

 </cffunction>
```

I originally tried to use the BeforeCreate event of Transfer's Event Model for this, but the processing that I wanted to do required data submitted by the user, and the Event Model does not provide a way to pass that in. That is why this method accepts the args argument, which will contain all of the data from the attributes scope (everything the user submitted).

In the next post, I'll walk through the populate() method, which is the largest method in this object.