How I Use Transfer - Part VII - A Concrete Gateway Object

Posted At : July 15, 2008 1:27 PM | Posted By : Bob Silverberg
Related Categories: OO Design, How I Use Transfer, ColdFusion, Transfer

In the previous post, I described my Abstract Gateway Object. As I did with the Service Objects, I'm going to take a moment to describe a Concrete Gateway Object as an illustration of how I use the Abstract Gateway. I'll start with a recap of the differences between the Abstract Gateway and Concrete Gateways, followed by a look at the code of a specific Concrete Gateway.

- **The Abstract Gateway Object**
- Is never instantiated as an object.
- Cannot be used as is.
- Is only ever used as a base object for Concrete Gateway Objects.
- There is only one Abstract Gateway Object, called AbstractGateway.cfc.
- Does not have any Transfer classes associated with it.

- **Concrete Gateway Objects**
- Are instantiated as objects.
- Methods on them are called by Service Objects.
- All extend AbstractGateway.cfc.
- There are many Concrete Gateway Objects, e.g., UserGateway.cfc, ProductGateway.cfc, ReviewGateway.cfc, etc.
- Have one "main" Transfer class associated with them, but can interact with others via code specific to the Concrete Gateway.

One thing to note here is that my Gateway Objects are all injected into Service Objects via Coldspring, and are only called by Service Objects. So the Service acts as an API to the entire model. If a Business Object needs to call a method on a gateway, it calls it via a Service Object that is injected into the Business Object.

Let's take a look at an example of a Concrete Gateway Object, ProductGateway.cfc. To start, here's the Bean definition of this gateway from my Coldspring config file:

```
<bean id="ProductGateway" class="model.Gateway.ProductGateway">

 <property name="TransferClassName"><value>product.product</value></property>

 <property name="DescColumn"><value>ProductCode</value></property>

</bean>
```

In here I indicate that the main Transfer class with which this service interacts is *product.product*. That means that calls to GetList(), GetActiveList() and ReInitActiveList() will be directed at the table defined to Transfer as product.product. I can write additional methods in my gateway that will interact with other Transfer classes, but the default methods, inherited from the AbstractGateway, will be directed at product.product.

I also indicate that the property that represents the description of the main class is *ProductCode*. That is used as a default sort sequence for the default methods.

And here's what's inside ProductGateway.cfc:

```
<cffunction name="GetList" access="public" output="false" returntype="any" hint="Interface to the getByAttributesQuery method">

 <cfargument name="Level1CategoryId" type="any" required="false" />

 <cfargument name="Level2CategoryId" type="any" required="false" />

 <cfargument name="Level3CategoryId" type="any" required="false" />

 <cfargument name="ProductName" type="any" required="false" />

 <cfargument name="SKU" type="any" required="false" />

 <cfargument name="ProductCode" type="any" required="false" />


 <cfset var TQL = "" />

 <cfset var TQuery = "" />

 <cfsavecontent variable="TQL">

 <cfoutput>

  SELECT Product.ProductId, Product.ProductCode, Product.SKU, Product.ActiveFlag,

    Product.ProductName, Product.Level1CategoryName, Product.Level2CategoryName,

    Product.Level3CategoryName

  FROM product.catalog_admin AS Product

  WHERE Product.ProductId IS NOT NULL

 <cfif structKeyExists(arguments,"Level1CategoryId") and Val(arguments.Level1CategoryId)>

  AND  Product.Level1CategoryId = :Level1CategoryId

 </cfif>

 <cfif structKeyExists(arguments,"Level2CategoryId") and Val(arguments.Level2CategoryId)>

  AND  Product.Level2CategoryId = :Level2CategoryId

 </cfif>

 <cfif structKeyExists(arguments,"Level3CategoryId") and Val(arguments.Level3CategoryId)>

  AND  Product.Level3CategoryId = :Level3CategoryId

 </cfif>

 <cfif structKeyExists(arguments,"ProductName") and Len(arguments.ProductName)>

  AND  Product.ProductName like :ProductName

 </cfif>

 <cfif structKeyExists(arguments,"SKU") and Len(arguments.SKU)>

  AND  Product.SKU like :SKU

 </cfif>
 <cfif structKeyExists(arguments,"ProductCode") and Len(arguments.ProductCode)>
  AND  Product.ProductCode like :ProductCode
 </cfif>
  ORDER BY Product.ActiveFlag DESC, Product.ProductName
 </cfoutput>
 </cfsavecontent>
 <cfset TQuery = getTransfer().createQuery(TQL) />
 <cfif structKeyExists(arguments,"Level1CategoryId") and Val(arguments.Level1CategoryId)>
  <cfset TQuery.setParam("Level1CategoryId",arguments.Level1CategoryId) />
 </cfif>
 <cfif structKeyExists(arguments,"Level2CategoryId") and Val(arguments.Level2CategoryId)>
  <cfset TQuery.setParam("Level2CategoryId",arguments.Level2CategoryId) />
 </cfif>
 <cfif structKeyExists(arguments,"Level3CategoryId") and Val(arguments.Level3CategoryId)>
  <cfset TQuery.setParam("Level3CategoryId",arguments.Level3CategoryId) />
 </cfif>
 <cfif structKeyExists(arguments,"ProductName") and Len(arguments.ProductName)>
  <cfset TQuery.setParam("ProductName","%" & arguments.ProductName & "%") />
 </cfif>
 <cfif structKeyExists(arguments,"SKU") and Len(arguments.SKU)>
  <cfset TQuery.setParam("SKU","%" & arguments.SKU & "%") />
 </cfif>
 <cfif structKeyExists(arguments,"ProductCode") and Len(arguments.ProductCode)>
  <cfset TQuery.setParam("ProductCode","%" & arguments.ProductCode & "%") />
 </cfif>

 <cfset TQuery.setDistinctMode(true) />
 <cfreturn getTransfer().listByQuery(TQuery) />

</cffunction>
```

Here I'm overriding the default GetList() method to allow for criteria to be passed in by a user. This function is based on one automatically generated for me by **Brian Rinaldi's** most excellent **Illudium PU-36 Code Generator**. One thing to note in here is the Transfer Class that this TQL is referring to. It's called product.catalog_admin, and is actually pointing at a view in my database. The product information in this application is spread across many tables, and rather than having to write a complicated TQL statement with multiple inner and outer joins, I just write my TQL against a view that already joins everything together.

Let's look at another method on the Concrete Gateway:

```
<cffunction name="getApprovedReviews" access="public" output="false" returntype="any" hint="Returns a query of Approved Reviews for a given Product">
 <cfargument name="ProductId" type="any" required="true" />
 <cfset var TQuery = 0 />
 <cfset var TQL = "" />
 <cfsavecontent variable="TQL">
  <cfoutput>
   SELECT Review.ReviewId, TUser.Nickname, Review.Rating, Review.Comments, Review.LastUpdateTimestamp
   FROM product.product AS Product
   JOIN product.review AS Review
   JOIN product.reviewstatus AS ReviewStatus
   JOIN user.user AS TUser
   WHERE Product.ProductId = :ProductId
   AND  ReviewStatus.ReviewStatusDesc = :ReviewStatus
   ORDER BY Review.LastUpdateTimestamp
  </cfoutput>
 </cfsavecontent>
 <cfset TQuery = getTransfer().createQuery(TQL) />
 <cfset TQuery.setParam("ProductId",Val(arguments.ProductId)) />
 <cfset TQuery.setParam("ReviewStatus","Approved") />
 <cfreturn getTransfer().listByQuery(TQuery) />
</cffunction>
```

This is basically just another query that I need within the application. My Concrete Gateways are quite simple in that they only include methods that return query objects.

As I alluded to in a **previous post**, I have one "special" Concrete Gateway Object, called ValueListGateway, which I use to interact with all of my "code" or "lookup" objects. It is used for objects like UserGroup, OrderStatus, ProductCategory, Colour, etc. It too extends AbstractGateway, but is built itself in an abstract way so that I can use it to interact with all of those "code" objects, without having to write a Concrete Gateway Object for each one. I plan on discussing that in a future post.

In the next installment I'm going to start looking at my AbstractTransferDecorator, which can be thought of as an Abstract Business Object.