How I Use Transfer - Part VI - My Abstract Gateway Object

Posted At : July 14, 2008 6:35 AM | Posted By : Bob Silverberg
Related Categories: OO Design, How I Use Transfer, ColdFusion, Transfer

In the past few posts in the series I discussed, at length, how I have implemented my service layer, which consists of an Abstract Service Object and Concrete Service Objects. I decided to take it easy this time around and just look at my AbstractGateway Object, as it's pretty simple in comparison.

As with the AbstractService, this object is not meant to be instantiated on its own, but rather acts as a base object which my concrete Gateway Objects extend. There are only three public methods in my AbstractGateway:

- GetList(), which returns a listing for the default entity
- GetActiveList(), which returns a listing for the default entity, but only includes Active records
- ReInitActiveList(), which is used to reinitialize the cached query for the Active List (see below for details)

Here's the Init() method:

```
<cffunction name="Init" access="Public" returntype="any" output="false" hint="I build a new Gateway">

  <cfset variables.Instance = StructNew() />

  <cfreturn this />

 </cffunction>
```

All I do in the Init() method is set up a struct to hold all of my private variables.

GetList() is currently set up to use Transfer's list() method to return a list of all records for the default entity:

```
<cffunction name="GetList" access="public" output="false" returntype="query" hint="Returns a query containing all records in the default table, ordered by the DescColumn">

  <cfreturn getTransfer().list(getTransferClassName(),getDescColumn()) />

 </cffunction>
```

TransferClassName and DescColumn are specified in the Coldspring config file for each Gateway bean. For example, for the UserGateway TransferClassName is user.user and DescColumn is UserName. I do in fact often override this method, but for the times that I don't need to I'm glad that I have a default method defined in this object.

Most of my Business Objects have a property called ActiveFlag, which is used for logical deletions. I rarely delete anything from the database, rather I set ActiveFlag to false and then it is up to the application to treat those items as deleted. So I often want to generate a list of entities which only includes active records (e.g., to populate a select box), in which case I ask the Gateway to GetActiveList(). Because this is something that I often need, I cache this query in the Gateway itself:

```
<cffunction name="GetActiveList" access="public" output="false" returntype="query" hint="Returns a cached query containing all ACTIVE records in the default table">

  <cfif NOT StructKeyExists(variables.Instance,"qryActiveList")>

   <cfset ReInitActiveList() />

  </cfif>

  <cfreturn variables.Instance.qryActiveList>

 </cffunction>



 <cffunction name="ReInitActiveList" access="public" output="false" returntype="void" hint="ReInitializes the data in the cached query">

  <cfset variables.Instance.qryActiveList = GetActiveListData() />

 </cffunction>



 <cffunction name="GetActiveListData" access="private" output="false" returntype="query" hint="Returns a query with all active records in the default table">

  <cfreturn GetTransfer().listByProperty(getTransferClassName(), "ActiveFlag", 1, getDescColumn()) />

 </cffunction>
```

When I call GetActiveList(), it checks to see if the query has already been cached, and if not it requests that it be done via the ReInitActiveList() method. Then it simply returns the cached query. ReInitActiveList() can also be called by the service layer after a database change, to ensure that the cached query remains up to date. I've encapsulated the actual running of the query into its own method, GetActiveListData(), so that I can vary the implementation of ReInitActiveList() and GetActiveListData() independently. For example, in some concrete gateways I may override both methods, while in others I may only override one of those methods. In the AbstractGateway, GetActiveListData() uses Transfer's listByProperty() method to retrieve only Active records, and sorts them by the DescColumn.

And that's pretty much it for the AbstractGateway. Most of the interesting gateway code is in the concrete gateways, an example of which I'll describe in the next post.

The only thing that I might change about this object is to use java.lang.ref.SoftReference for the cache, instead of just caching the query in the gateway (which in turn is cached in the application scope). Transfer's object cache uses soft references, and I have read a couple of interesting articles about this. I'm not sure that this would be an appropriate use of soft references as I don't really want these queries to fall out of the cache. I only cache queries for certain gateways, and the size of those queries is quite limited, so for now I think it's best to leave it as is. If I do go that route I'll be sure to document what I've done in a blog post.

One final note, during the writing of this series I've been evaluating the code base and noticing things that I'd like to change. I've also received a number of useful comments from people that are making me reconsider some things. One change that will probably be coming will be to move a lot of the logic that is currently in my service layer into my gateways. This will allow me to decouple Transfer from the service layer, something that I'm already trying to do to a certain extent (e.g., by the use of the Get() method in the service). For now I'm going to continue documenting what I'm doing at this moment, and when I do start to refactor I'll post about that as well.