CF9 ORM Quick Tip - Removing All Items from A Collection

Posted At : October 13, 2009 2:48 PM | Posted By : Bob Silverberg
Related Categories: OO Design, ColdFusion, CF ORM Integration

For this example of using ColdFusion 9's ORM features, let's assume we have a User object and a Department object. The Department has a collection of Users, like so:

```
component persistent="true" output="false" entityname="Department" {

 property name="DeptId" fieldtype="id" generator="native";

 property name="Name";

 property name="Users" fieldtype="one-to-many" type="array"

  cfc="User" fkcolumn="DeptId" singularname="User";

}
```

Let's also say that Department 1 currently has 3 Users assigned to it, call them User1, User2 and User3. Now, we want to write some code that will empty the Department's collection of Users. A first attempt might look something like:

```
Users = Department.getUsers();

for (i = 1; i LTE ArrayLen(Users); i = i + 1) {

 Department.removeUser(Users[i]);

}
```

But this won't work. In fact, if you run that code you'll see that you end up with a Department that still has User2 assigned to it. The reason this happens is that as soon as you remove User1 from the collection, the array shrinks, so User2 now occupies position 1 and User3 occupies position 2. So the next time through the loop, User3 gets removed (because i now equals 2) and Users[2] is User3. Then the loop finishes because the condition is met.

You might want to try doing an array loop using tags, like this:

```
<cfset Users = Department.getUsers() />

<cfloop array="#Users#" index="i">

 <cfset Department.removeUser(i) />

</cfloop>
```

Nope, this throws an error. CF sets the length of the array at the beginning of the loop, so by the time it's inside the third iteration it's trying to access Users[3], which no longer exists.

There are a couple of ways of achieving the objective. The simplest, is to call the setter for the collection and pass in an empty array:

```
Department.setUsers([]);
```

Another is to forget about trying to loop over the array, and instead do a conditional loop based on the current size of the array:

```
Users = Department.getUsers();

while (ArrayLen(Users)) {

 User.removeDept(Users[1]);

}
```

Personally, I prefer the latter approach. The first seems a bit hacky to me, although both of them require that you program to the implementation of the collection (you are writing code expecting the collection to be an array). One way to make that a bit less painful would be to move whichever method you choose into the actual object itself. For example, in the Department object, add the following method:

```
function void removeAllUsers() {

 var Users = this.getUsers();

 while (ArrayLen()) {

  this.removeDept(Users[1]);

 }

}
```

Now you can call removeAllUsers() from outside of the object (e.g., from a Service) and nothing other than the object will have to know that your collection has been implemented as an array.

OK, so I snuck an OO design principle into my Quick Tip. It was hard to resist.