Managing Bi-directional Relationships in ColdFusion ORM - Array-based Collections Follow-up

Posted At : April 2, 2010 9:01 AM | Posted By : Bob Silverberg
Related Categories: ColdFusion, CF ORM Integration

Barney Boisvert made a comment on my last post on this topic, suggesting that a nice way to reduce the amount of code one has to write, and to boost performance, would be to have one side in a bi-directional relationship simply delegate to the other side. This way you are only writing the code to do the work in one object, and it also reduces the number of *hasX()* calls.

I changed my code to try this technique out, and I quite like it. Because the code is slightly different I figured I might as well write about it here, to keep my posts on this topic up to date.

Taking the same Country and Language cfcs that we looked at last time:

```
component persistent="true" hint="This is Country.cfc"

{

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

 property name="CountryCode" length="2" notnull="true";

 property name="CountryName" notnull="true";

 property name="Languages" fieldtype="many-to-many" cfc="Language"

  type="array" singularname="Language" linktable="CountryLanguage";

}


component persistent="true" hint="This is Language.cfc"

{

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

 property name="LanguageName" notnull="true";

 property name="Countries" fieldtype="many-to-many" cfc="Country"

 type="array" singularname="Country" linktable="CountryLanguage"

 inverse="true";

}
```

I'm going to add the "real" code to the Language.cfc, like in the last post, and I'll also show the "deferring" code in Country.cfc. The Country.cfc code will be much smaller, so let's just take a look at that first:

```
component persistent="true" hint="This is Country.cfc"

{

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

 property name="CountryCode" length="2" notnull="true";

 property name="CountryName" notnull="true";

 property name="Languages" fieldtype="many-to-many" cfc="Language"

  type="array" singularname="Language" linktable="CountryLanguage";


 public Country function init() {

  if (isNull(variables.Languages)) {

   variables.Languages = [];

  }

  return this;

 }


 public void function addLanguage(required Language Language)

  hint="set both sides of the bi-directional relationship" {

  arguments.Language.addCountry(this);

 }


 public void function removeLanguage(required Language Language)

  hint="set both sides of the bi-directional relationship" {

  arguments.Language.removeCountry(this);

 }


}
```

So, when I call *addLanguage()* on a Country object all that's going to happen is that the *addCountry()* method on the Language object that has been passed in will be called. All of the processing that must be done to set both sides of the relationship will be contained in that *addCountry()* method. The same applies to calling

removeLanguage() on a Country object. Note also that I still need to default the Languages property to an empty array or I risk errors when working with brand new objects. OK, let's move on and take a look at the new version of Language.cfc:

```
component persistent="true" hint="This is Language.cfc"

{

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

 property name="LanguageName" notnull="true";

 property name="Countries" fieldtype="many-to-many" cfc="Country"

 type="array" singularname="Country" linktable="CountryLanguage"

 inverse="true";


 public Language function init() {

  if (isNull(variables.Countries)) {

   variables.Countries = [];

  }

  return this;

 }


 public void function addCountry(required Country Country)

  hint="set both sides of the bi-directional relationship" {

  if (not hasCountry(arguments.Country)) {

   // set this side

   arrayAppend(variables.Countries,arguments.Country);

   // set the other side

   arrayAppend(arguments.Country.getLanguages(),this);

  }

 }


 public void function removeCountry(required Country Country)

  hint="set both sides of the bi-directional relationship" {

  if (hasCountry(arguments.Country)) {

   // set this side

   var index = arrayFind(variables.Countries,arguments.Country);

   if (index gt 0) {

    arrayDeleteAt(variables.Countries,index);

   }

   // set the other side

   index = arrayFind(arguments.Country.getLanguages(),this);

   if (index gt 0) {

    arrayDeleteAt(arguments.Country.getLanguages(),index);

   }

  }

 }

}
```

This is pretty much the same as the code that we looked at in the last post, with the difference being that we're going to check to see whether the add or remove is required, using the *hasCountry()* method, and then, if required, we're going to manually do the add and remove on both sides. As Barney pointed out in his comment, this reduces the number of times that a *hasX()* method has to be called to one, and also isolates all of the code in one object, which could potentially make maintenance simpler.

Again, I just want to point out that this new code is based on an idea that Barney shared with me via a comment on my last post, so the credit for this cleverness belongs to him.