Development-General Best Practices

Avoid Changing WOComponent Structure Before It Is Used

One of the more pernicious problems that afflict new WebObjects developers results from creating and sending out a WOComponent in appendToResponse, then changing the structure of that component before the subsequent takeValuesFromRequest and invokeAction methods have completed on that page.

Doing so will confuse WO's interpretation of the page when it's referenced either in the takeValuesFromRequest or invokeAction. The results will make you feel like your WOComponent has become schizophrenic. If it gets confused during the takeValuesFromRequest, it will take values from a field or fields on your form and put them in unrelated instance variables in your corresponding class file. If it gets confused during the invokeAction, it will activate an action method unrelated to the one bound to the WOSubmitButton or WOHyperlink on which you clicked.

In short, if your page starts showing signs of schizophrenic behavior, this is the first place I would look. So what, exactly, is the problem?

From the point that you execute the appendToResponse(), you must make sure that none of the objects used to create the HTML structure that went out as the response, change in any way that would cause that HTML structure (page template) to change before the ensuing takeValuesFromRequest() and invokeAction() methods have completed. In particular, if any boolean values change on which WOConditionals depend, if any array sizes used in WORepetitions change, then you will have created a problem whose manifestations can often be difficult to track down.

When you execute the appendToResponse, WO uses your WOComponents, subcomponents and WOElements to create a hierarchy of objects (an object graph) which it then "walks" to generate the HTML. When the takeValuesFromRequest and invokeAction methods execute, WO recreates this template and re-"walks" it to determine which objects receive the returned values and which object was responsible for invoking the action. If any changes to your object graph would cause that template structure to change between the appendToResponse and the corresponding takeValuesFromRequest and invokeAction, then WO will get lost and probably put wrong values in your form objects and determine that the wrong element (say, a WORadioButton) caused the action.

It's both a common and a pernicious error.

If you have WOConditionals, it's safest to bind them to booleans in your class file that you only change in appendToResponse, before its call to super. appendToResponse. If that is difficult, use whatever logic is natural in your program, but put a boolean (to which you bind your WOConditional) to reflect that logic before the super.appendToResponse, and only change the boolean accordingly at that point in that method.

If you have WORepetitions, confirm that none of your code changes the sizes of those arrays or indexes on which their sizes depend between your activation of super.appendToResponse and the completion of the corresponding invokeAction for that page.

For a deeper explanation of this process and its ramifications, see [In particular, read the last paragraph, and the following page DevGuide-WOClasses.]. In this latter page, pay close attention to the second to last paragraph.

Creating WOComponents

Rather than use

```
MyNewPage nextPage = (MyNewPage)pageWithName("MyNewPage");
```

Prefer this form:

```
MyNewPage nextPage = (MyNewPage)pageWithName(MyNewPage.class.getName());
```

The class.getName() allows Eclipse to do proper refactoring and you can right click=>References=>Workspace your class and truly find all the references vs just having a string references. One other advantage that it has is that if you have two pages with the same name but in different packages, this avoids confusion. If you just use pageWithName("MyNewPage"😉 , WO can return the wrong one. Minor change but really nice benefits.

If you use 1.5, you can use the 1.5 variation:

```
@SuppressWarnings("unchecked")
public <T extends WOComponent> T pageWithName(Class<T> componentClass) {
  return (T) super.pageWithName(componentClass.getName());
}
```

which removes a cast:

```
MyNewPage nextPage = pageWithName(MyNewPage.class);
```

Avoid String Literals

KeyValueCoding (aka KVC) tends to encourage the use of constructs like this

```
website.addObjectToBothSidesOfRelationshipWithKey(newFolder, "folders");
```

This litters code with hard coded strings. Changing the property name breaks code with no compilation warnings. If you use EOGenerator to generate constants for the names:

```
public static final String FOLDERS = "folders";
```

You can use them in place of the hard coded strings and get errors when changes affect code:

```
website.addObjectToBothSidesOfRelationshipWithKey(newFolder, FOLDERS);
```

An EOGenerator template (fragement) to do this:

```
public static final String ENTITY_NAME = "<$name$>";

<$foreach propertyName classPropertyNames.@reversedArray do$>
  public static final String <$propertyName.uppercaseString$> = "<$propertyName$>";<$endforeach do$>
```

Except

- This does not address the related issue when property names are used in bindings in a wod file
- This does not work in the (admittedly rare) case where one class implements multiple entities. EOGenerator assumes one entity == one class. So, MyEO.ENTITY_NAME will return the name of the last entity to get generated.

See the EOGenerator page for more similar tricks.