SeparationOfLogicAndContent

This RT was send to the cocoon-users mailing list by AlanHodgkinson. It was put here in order to create further discussion on best practices and design pattern for complex Cocoon-based web applications.

Abstract

I present a way of naming links form actions that allow you to separate links and forms from the next page to display. I then raise questions about processing web application events, specifically how to do this within the existing Cocoon pipeline mechanism and maintain a separation between processing of the input from one page from the generation and display of the next page. I then say that Cocoon is great.

Introduction

First Michael Edge wrote:

> I have a question regarding the use and purpose of XSP. I

> believe that XSP is an attempt to separate content/logic

> from presentation

Then Leigh Dodds wrote:

> I've been thinking along the same lines recently, and am

> still in two minds.

Join the club. Cocoon is just fine at separating presentation from content and logic. The problem is the separation of the content and logic.

I am relatively new to Cocoon and from what I've read of the documentation and in the mail archive, it seems that Cocoon is not yet ideal for web applications that have complex behavior based on user input (e.g. the page flow changes based on the data the user enters and the results of state changes in the session and database/business-object layer). It's possible to implement such applications with Cocoon, but it doesn't (yet) seem to be Cocoon's strength.

The yet to be implemented flowmap mentioned in the cocoon-dev mailing list may be the solution we are looking for. (Seach for 'flowmap' in the cocoon-dev mailing list. The flowmap is not yet fully defined, which means that now is the time to provide your input).

It's About Control

Thinking in Model-View-Controller (MVC) terms, the problem is how to separate the controller from the view/model. Note that we don't need to worry too much about mixing up the model and view, given Cocoon's flexible handling of XML data via pipelines.

Mapping MVC to web applications is, in my mind, a priori, a mess, due to the fact that the requests (events) the web application receives must ultimately be encoded as links and form actions in the HTML presented to the user. Thus some mixing of view and control is required. In a sense, we are trying to make the best of a bad situation.

Specifically, you have to embed links and form actions that define the input events to the controller directly in the in web pages. There are ways arund this, but before I present a solution let's look at the types of input events we need to consider.

Links in Content

A typical web application the links can be divided into the following categories:

- Gotos: Links that simply take you to another part of the application or web site. They typically make no state change, except perhaps to note in a sessinon variable that you are now on a different page. These are typically simple links.
- Content Display: Links that display statically or dynamically generated data, but cause no state change. This includes search operations, and links/buttons that display 'additional' information. Typically these are links and buttons that include one or more key values E.g. an id value or form data that identifies the record(s) to display or present in the next page or form.
- State changes: Links that cause a state change. This includes business object and session state changes. These can be quite complex operations. Often these are buttons associated with form data that the user enters.

How can you generate the embedded links and form actions in web content without creating explicit dependencies between the web pages (and also allow reuse of page components via Cocoon's aggregation)? The answer is to change the semantics of links and form actions.

Goto Considered Harmful, Come-From is Fine

Instead of having the link tell the controller where it wants to go, which is the controller's responsibility, it should merely tell the controller which button/link it 'is' or, in other words, where it is 'coming from'. For example, instead of a button having form action named:

```
/cgi-bin/doUserUpdateSubmit.pl
```

which says what the web server should do, you would have:

```
/userUpdateForm/changeName/submitButton
```

Which tells the controller exactly where the button was pressed (and consequently what parameters it should expect) and allows the controller the freedom to decide where to go next. More importantly, the developer is free to rearrange and reuse the pages and page components without changing any of the links or form actions. All that is required is to maintain a mapping of the events and states to their action routines (this might be stored in something called, say, a flowmap 🙂

The whole point of this is to define a 'URI space' that enables you to specify, independent of physical location or file naming, all the application events that the controller must respond to. In other words, all the buttons and links that the web application contains.

Obviously you must also define the fields and parameters associated with the buttons and links, but at least you have turned all the links and form actions into events and dissociated them from the application flow.

So What About Cocoon?

So far this all theory. The practical problem is:

How do you do this in Cocoon?

<disclaimer>
Being a Cocoon, newbie, I quickly reach my limits and
look to others to supply helpful suggestions. That said,
here are some ideas.
</disclaimer>

In the JSP/Servlet world, an obvious answer is the single servlet solution mentioned by Leigh, where you have one servlet that implements all the control logic and dispatches to JSP pages for display. In servlet based applications you're free to write as many servlets as you want. In practice, pretty much everybody only writes one, which acts as a controller.

There's probably going to be a parallel in Cocoon development world. Perhaps we'll all end up writing a single master pipeline that calls 'internal-only' pipeline components to do all the generation and display work.

How to implement the controller? Actions seem a reasonable choice. As explained in the Cocoon documentation, you can implement an action that sets one or more sitemap parameters that enable you to delegate the to the appropriate view based on the processing of the request parameters, session variables and business objects. This would require having the 'master pipeline' described above.

The Controller could be implemented as a generic class implementing Action, which could read an XML config file (called the flowmap or FSM-config). The config file would contain, in some form, the matrix of application states and events and their mappings to the 'action pipelines'.

It might be possible to encode the controller configuration directly in the pipeline portion of the sitemap, but I'm not sure that you want to include all that 'application code', (which is what this information really is, in the sitemap.

Pipelines Are So Cool That I Want Two of Them

How can I separate the processing of the page form data from the page I just left, from the generation and display of the page I am going to present next?

Cocoon, with its powerful pipelines, seduces you into implementing your processing and display in a single pipeline (which might call sub-pipelines, but the point is that it's a single event chain). The problem is, and you see it mentioned in the mailing lists, that once the pipeline is started, and the SAX events are flying towards the user, there's no turning back or possibility for switching to another pipeline if you discover or decide you want to display something else.

When responding to a request I'd like to use one pipeline composed of XSP logicsheets to perform the event processing and then be able to switch to another pipeline to generate the next page. All this should be managed by my controller which gets to pick the second pipeline based on the processing results from the first.

Is this even possible in Cocoon? Are redirects an answer? My reading of the mail archives leads me to believe that this is probably not the correct solution. But perhaps there's some other technique.

It is of course possible to implement all the page processing in Java, in an action class, which chooses the next page, but then I loose all the advantages of the logicsheets. Is there some trick we can use so that we 'have it all'?

I eagerly await your feedback and suggestions.

Best wishes to you all,

Alan Hodgkinson