Page Flow Users Guide

Table of contents

1. Advantages of Programming with Page Flows

Page Flows make building Java web applications easy and intuitive. When programming with Page Flows, the developer writes Java files and JSP pages--that's it. There is very little occasion to work with configuration files, or other components. Page Flow programming is not only simple, it is also excells at separating the presentation logic from the data processing logic. This results in uncluttered JSP code which is easy to understand and edit. Moreover, many of the most difficult programming tasks, such as security and validation, are handled with a simple declarative programming model using Java annotations.

2. How Do Page Flows Work?

A Page Flow consists of a single directory containing a Java class, called the "Controller", and any number of JSP pages. The role of the JSP pages is to present a visual interface for users of the web application. The role of the Controller file is to coordinate all of the things that can happen when a user visits a web site. These duties include: handling user requests, fashioning responses to user requests, preserving session state, and coordinating back-end resources (such as databases and web services).

The JSP files use special tags (the "<netui>" tags) and databinding expressions which bind the user interface to objects and actions in the Controller file.

tbd: need diagram here

The action methods in the controller file implement code that can result in site navigation, passing data, or invoking back-end business logic via controls. Significantly, the business logic in the controller class is separate from the presentation code defined in the JSP files. The overall purpose of a page flow is to provide you with an easy-to-use framework for building dynamic, sophisticated web applications. While page flows give you access to advanced features of J2EE, you do not have to be a J2EE expert to quickly develop and deploy Java-based applications built on page flows.

The programming model: annotations, data binding expressions, <netui> tags, etc.

3. Navigation

Introduction

Suppose you have two JSP pages, pageA.jsp and pageB.jsp. Also, suppose you want to place a link on pageA, that will navigate the user to pageB.

One solution would be to place an HTML anchor tag (<a>) on pageA that links to pageB.

But what if you had dozens of pages that needed to link to pageB? You could repeat the anchor tag solution a dozen times, by placing an anchor tag on each page that required a link to pageB. But this solution becomes cumbersome if you ever need to edit those links. At some point in the future, you may want to change those links, so that they navigate to another page, say to pageC. But this would require sifting through all of your web site's JSP pages for anchor tags to edit.

Page Flows solve this problem by placing navigational control within a single file: the Controller file (=JPF file). Methods within the Controller file decide the target page, whether it is pageB, pageC, or somewhere else. The job of the JSP page is to invoke the method, not to directly link to another JSP page. The work of navigation in a Page Flow breaks down to a two step process: (1) The JSP page invokes a method in the Controller file, then (2) the method navigates the user to the target page.

How the Code Works

Methods in the Controller file are invoked through specially designed JSP tags, called "netui" tags. These tags appear with the prefix **<netui:>**. (Tags with the <netui:> reference the netui-tags-html.tld tag library.) Neuti tags like <neuti:anchor>, <netui:button>, and <netui:imageButton> are all capable of invoking methods in the Controller file. The method to invoke is specified by the tag's `action` attribute. For example, the following <netui:anchor> tag invokes the `toPageB` method by referencing the method in it's `action` attribute.

pageA.jsp

```
<%@ taglib uri="netui-tags-html.tld" prefix="netui"%>
    ...
    <netui:anchor action="toPageB">Link to page_B.jsp</netui:anchor>
```

Suppose the link above is clicked. When clicked, the following method, `toPageB`, is invoked.

The method below, when invoked, navigates the user to pageB.jsp. The code within the method body is very simple: upon invocation, the method immediately returns a Forward object, with the String parameter "success". The code *above* the method does the real work of determining the navigational target. The code above the method consists of two metadata annotations. "Metadata" means that the annotation configures or sets a property on some part of the Java code. In the example below, the @Jpf.Action annotation makes the method available to invocation (by netui tags), while the @Jpf.Forward annotation configures the navigation information. In the this case the method it configured to navigate to pageB.jsp whenever it is invoked.

Controller.jpf

```
import org.apache.beehive.netui.pageflow.Forward;
...
    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name = "success", path = "pageB.jsp")
        }
    )
    protected Forward toPageB()
    {
        return new Forward("success");
    }
```

Changing the Navigation Target

To change the navigation target of this action method, simply change the value of the path attribute. For example, if you want this action method to navigate to pageC.jsp, you would make the following change to the controller file (no change to the JSP page is necessary).

```
    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name = "success", path = "pageC.jsp")
        }
    )
    protected Forward navigate()
    {
        return new Forward("success");
    }
```

Conditional Navigation

By placing navigational control in the Controller file, you can control conditional navigation.

The following example is from the Petstore sample. The `begin` method checks to see if the user is logged in or not, and navigates the user appropriately.

`beehive/trunk/samples/petstoreWeb/Controller.jpf`

```
    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name = "shop", path = "/shop/Controller.jpf"),
            @Jpf.Forward(name = "index", path = "/index.jsp")
        }
    )
    protected Forward begin()
    {
        if (_sharedFlow.isUserLoggedIn())
            return new Forward("shop");
        else return new Forward("index");
    }
```

Putting it all together, a Forward object is returned by an action method. The Forward object passes the string "success", indicating that it should behave according to the directions encoded in the annotation @jpf:forward name="success". That annotation's path attribute has the value "page_B.jsp", which causes the page flow controller to load page_B.jsp into the browser.

4. Submitting Data: Form Beans and Data Binding