Nested Page Flows

Table of contents

1. Introduction

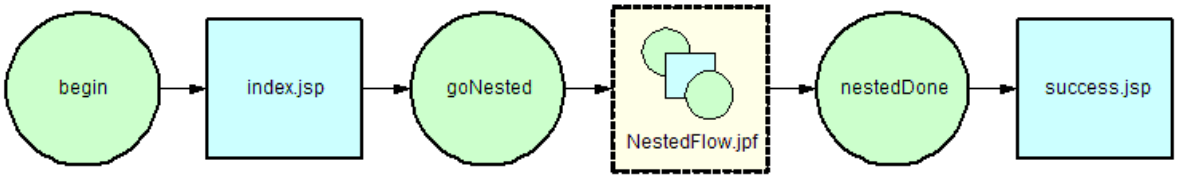
By default, executing an action in a new page flow causes the current page flow to be discarded. This behavior allows you to create separate controllers for different sections of your project, and it minimizes the amount of data kept in the user session at one time. Each page flow manages its own state and logic. "Nested page flows" give you an even greater ability to break up your project into separate, self-contained bits of functionality. At its heart, "nesting" is a way of pushing aside the current page flow temporarily and transferring control to another page flow with the intention of coming back to the original one.

So when would you use this? Nesting is useful when you want to do one of the following tasks:

- gather data from the user, for use in the current page flow;
- allow the user to correct errors or supply additional information en route to executing a desired action;
- show an alternate view of data represented in the current page flow;
- bring the user through a "wizard";
- show the user information that will be useful in the current page flow (e.g., help screens can be easily implemented as nested page flows); and
- in general, to further break up your application into separate (and in many cases reusable) pieces.

2. Basics

Let's start with a very simple example. You can find the code for this under `basicNesting` in the NetUI Samples application. Here, we have a "main" page flow which forwards to a nested page flow, which later returns to the main page flow. The flow looks like this:



main page flow

Here is the code for the main page flow:

```
@Jpf.Controller(
```

```
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="goNested",
path="../nested/NestedFlow.jpf"),
        @Jpf.SimpleAction(name="nestedDone", path="success.jsp")
    }
)
public class MainFlow extends PageFlowController
{
}
```

As you can see, the `begin` action forwards to `index.jsp`, which allows you to raise the `goNested` action. This action enters the nested page flow *simply by forwarding to it*. Any time you hit the URL for a nested page flow (or any one of its actions or pages), you enter the nested page flow, and the current one is pushed aside.

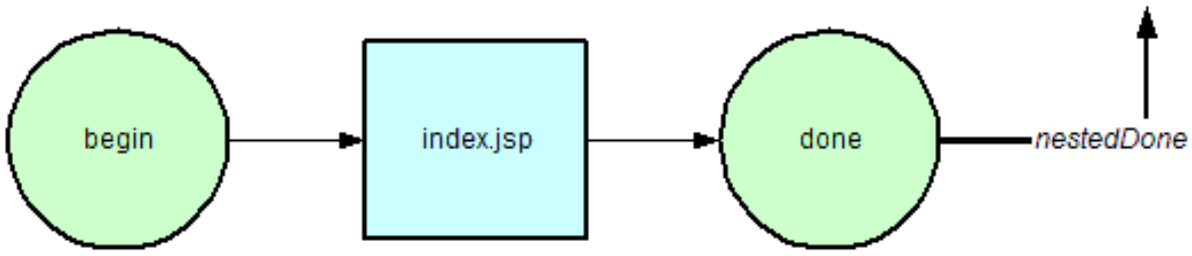
When the nested page flow returns, it causes the `nestedDone` action to run, and this action simply forwards to `success.jsp`

So how does the nested page flow return to the current one, and raise the `nestedDone` action? Here is the code for the nested flow:

```
@Jpf.Controller(
    nested=true,
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="done", returnAction="nestedDone")
    }
)
public class NestedFlow extends PageFlowController
{
}
```

Note the `nested` =`true`, which defines this as a nested page flow. Also note the `returnAction` attribute on the simple action `done`. When this action is executed, it returns to the original page flow (`MainFlow`) and raises its `nestedDone` action. This is called an **exit point** of the nested page flow.

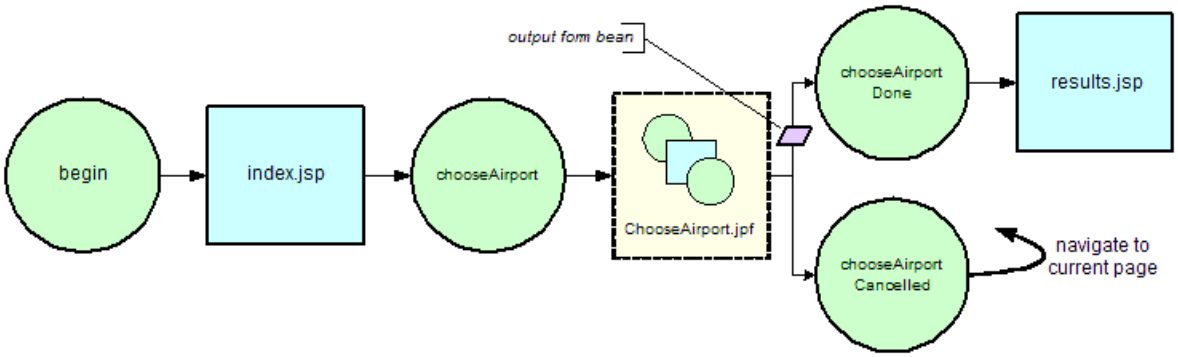
The nested flow looks like this:



nested page flow

3. More Nested Page Flow Features

Often, you want to do more than simply invoke and return from a nested page flow. For instance, you may want to gather data from a nested page flow, for use in the current page flow. In the example below (found under `nesting` in the NetUI Samples project), the user is forwarded to a nested page flow (`/nesting/chooseAirport/chooseAirport.jpf`), which is a wizard that helps the user find an airport. The nested page flow returns the chosen airport to the original page flow, which continues with its sequence. First, here is a diagram of the main page flow:



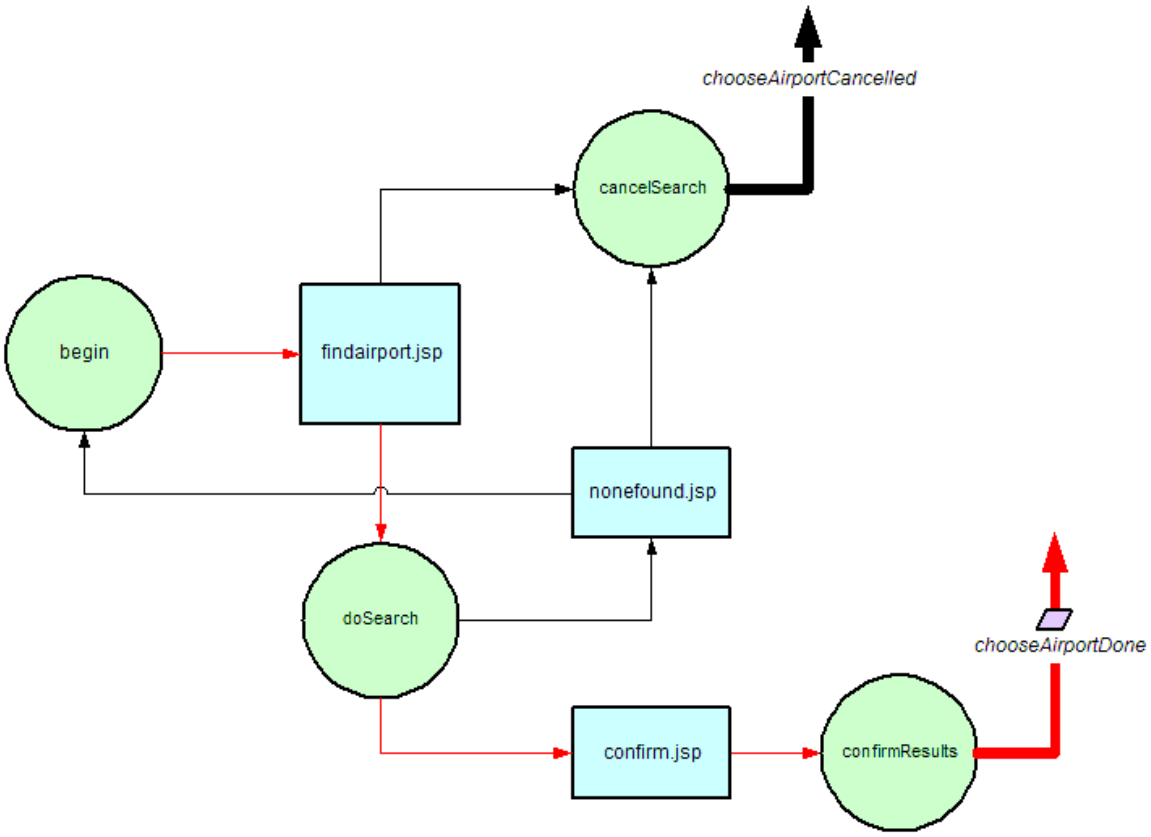
main page flow

This page flow demonstrates two new features related to nesting:

- The nested page flow returns a form bean (`ChooseAirport.Results`) when it raises the `chooseAirportDone` action.
- If the nested page flow raises a `chooseAirportCancelled` action, the page flow will go back to *the most recent page shown to the user.*, whatever that page is.

3.1. Returning data from a nested page flow

Here is a diagram of the nested page flow `ChooseAirport.jpf`, with the "happy path" to the `chooseAirportDone` return-action hightlighted in red:



Choose Airport nested page flow

During the course of this page flow, a member variable called `_currentResults`, of type `ChooseAirport.Results` is kept. As the user moves through the page flow, possibly re-trying and changing the desired result, this member variable is kept up-to-date. In the end, it is *returned* as an "output form bean" along with the `chooseAirportDone` return-action, using the `outputFormBean` attribute. This is what it looks like in the annotations:

```
    @Jpf.SimpleAction(name="confirmResults",
returnAction="chooseAirportDone", outputFormBean="_currentResults")
```

This annotation simply specifies that the value of `_currentResults` will be sent along with the return-action `chooseAirportDone`.

Note:

You do not have to return a member variable as an output form bean. If you just want to return a local variable, you would use the `outputFormBeanType` attribute instead, and you would pass the bean on the `Forward` object returned from an action method, like this:

```
    @Jpf.Action(
        forwards={
             @Jpf.Forward(name="done", returnAction="chooseAirportDone",
outputFormBeanType=Results.class)
        }
    )
    public Forward confirmResults()
    {
        Results results = initialize a Results object
        return new Forward("done", results);
    }
```

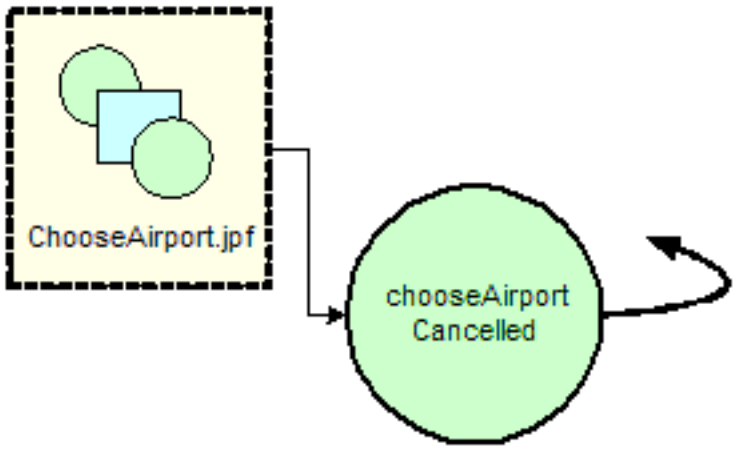
In the original page flow, there is a `chooseAirportDone` method that accepts this form bean as an argument, like this:

```
@Jpf.Action(
    ...
)
protected Forward chooseAirportDone(ChooseAirport.Results results)
{
    ...
}
```

As you can see, the page flow handles this returned form bean just like it would handle a form bean posted from a page.

3.2. Navigating back to the original page of the main flow

The the current example, the main page flow goes back to the most recent page whenever the nested page flow raises the `chooseAirportCancelled` action. We are referring to this section of the diagram of the main page flow:



chooseAirportCancelled

To navigate to the most recent page, the page flow uses the `navigateTo` attribute on a `@Jpf.SimpleAction` (or a `@Jpf.Forward`), like this:

```
@Jpf.SimpleAction(name="chooseAirportCancelled",
navigateTo=Jpf.NavigateTo.currentPage)
```

Or could also go back to the *previous* page:

```
@Jpf.SimpleAction(name="chooseAirportCancelled",
navigateTo=Jpf.NavigateTo.previousPage)
```

Or it could re-run the most recent action in the current page flow:

```
@Jpf.SimpleAction(name="chooseAirportCancelled",
navigateTo=Jpf.NavigateTo.previousAction)
```

Note:

The `navigateTo` feature is not only used with nested page flows. You can use it on any `@Jpf.SimpleAction` or `@Jpf.Forward` annotation in your page flow. It just happens to be a very valuable feature when used in conjunction with nested page flows.

3.3. Passing Data to a Nested Page Flow

Sometimes you will want to pass data into a nested page flow. While this may be less common than returning data from a nested page flow, it is useful for initializing data in the flow. To do this, you will add a form bean to the `begin` action of the nested page flow, and in the calling page flow you will pass an instance of this bean on the `Forward` object that is sent to the nested page flow.

3.3.1. Add a form bean to the nested page flow's begin action

To add a form bean, simply add a single argument to a `begin` action method:

```
@Jpf.Action(
    forwards={
        @Jpf.Forward(name="index", path="index.jsp")
    }
)
public Forward begin(InitBean initBean)
{
    ...
}
```

The bean type can be any class of your choosing; for instance, you can make it a `String`, which means that the nested page flow requires a `String` to be initialized:

```
@Jpf.Action(
    forwards={
        @Jpf.Forward(name="index", path="index.jsp")
    }
)
public Forward begin(String initString)
{
    ...
}
```

3.3.2. Pass the form bean from the calling page flow to the nested page flow

You can pass the initialization bean to the nested page flow by adding it to the `Forward` object that is sent to the nested page flow:

```
@Jpf.Action(
    forwards={
        @Jpf.Forward(name="nestedFlow", path="/nested/NestedFlow.jpf",
outputFormBeanType=InitBean.class)
    }
)
public Forward goNested()
{
    InitBean initBean = initialize the bean
    return new Forward("nestedFlow", initBean);
}
```

Note that the `outputFormBeanType` annotation attribute is optional; it mainly helps tools understand the output of the action, and it ensures that an incompatible type will not be passed.

4. Other Notes

Here are some other notes about using nested page flows:

- Aside from the `nested=true` on the `@Jpf.Controller` annotation, and defining exit points through `returnAction`, nested page flows are built just like other non-nested page flows.
- You enter a nested page flow by hitting its URL, by hitting the URL for any of its actions, or by hitting the URL for any of its pages (pages in the same directory path). When nesting occurs, the original page flow is pushed onto the "nesting stack", and is popped off the stack when the nested page flow hits an exit point (through a `returnAction` attribute).
- Nested page flows can nest themselves.
- While in a nested page flow, you can get a reference to the calling page flow through `PageFlowUtils.getNestingPageFlow()`.