Shared Flow vs. Inheritance

Table of contents

1. Introduction

Page Flow supports both inheritance and Shared Flow. At first glance the two seem similar; both allow you to share actions and exception handlers. The general guideline for which to use is simple: *use Page Flow inheritance whenever you can*. It allows you to share more than just actions and exception handlers (e.g., you inherit everything in the base class `@Jpf.Controller` annotation), and it uses a familiar Java concept in order to do it. This document mainly explains the (important) cases where you *would* want to use Shared Flow.

2. When to Use Shared Flow

There are three main cases where you would want to use Shared Flow: for accessing shared state, for shared/templated user interface, and when you cannot change your controller class hierarchy.

2.1. Accessing shared state

You want to share actions or exception handlers that use a *single copy* of some shared state. For example, the following shared flow action `switchToLargePictures` sets a single flag that can be used by *many* page flows:

```
@Jpf.Controller
public class MySharedFlow extends SharedFlowController
{
    private boolean _usingLargePictures = false;

    @Jpf.Action(
        forwards={
            @Jpf.Forward(name="cur", navigateTo=Jpf.NavigateTo.currentPage)
        }
    )
    public Forward switchToLargePictures()
    {
        _usingLargePictures = true;
        return new Forward("cur");
    }

    public boolean isUsingLargePictures()
    {
        return _usingLargePictures;
    }
}
```

There is only one instance of a given shared flow per user, so any page flow which references `MySharedFlow` will have access to the *single* value of this flag. For example, the following

page flow references `MySharedFlow` under the name "mySharedFlow":

```
@Jpf.Controller(
    sharedFlowRefs={
        @Jpf.SharedFlowRef(name="mySharedFlow", type=MySharedFlow.class)
    }
)
public class ExamplePageFlow extends PageFlowController
{
}
```

It can access the shared flow's `usingLargePictures` property in one of two ways:

- In its JSPs, through databinding, e.g.,

  ```
  <c:if test="${sharedFlow.mySharedFlow.usingLargePictures}">
      ...
  </c:if>
  ```

- Directly, through an annotated field in the page flow controller class:

  ```
  @Jpf.SharedFlowField(name="mySharedFlow")
  private MySharedFlow _mySharedFlow;    // This field is
  auto-initialized.

  @Jpf.Action(...)
  public Forward someAction()
  {
      if (_mySharedFlow.isUsingLargePictures())
      {
          ...
      }
  }
  ```

There is a simple reason you would not want to put a flag like `isUsingLargePictures` in a base class. If you did, you would end up with a *separate copy* of the value in each derived controller class, thus making it more difficult to share the flag.

2.2. Shared actions and exception handlers for shared user interface

Say you are sharing some bit of user interface, like a menu bar. You may be using the NetUI Template tags, or you may be using Page Flow's support for Tiles. In either case, the user interface you're sharing will likely have its own actions (and possibly exception handlers) associated with it. *It usually does not make sense to be forced to extend a different page flow controller, just to get the shared actions for something like a menu bar.* You may be including *lots* of shared user interface (navigation bar, header, footer, etc.), and it would be bad for each one to require its own base class. Instead, each one can have an associated

shared flow, which you reference in your page flow using a `@Jpf.SharedFlowRef`.

Note:

The NetUI Samples show shared flows being used with both the Template tags and with Tiles.

2.3. You cannot change the inheritance hierarchy for your page flow controller

In some cases, you simply cannot change the base class for your page flow controller. You may have a prescribed base class, yet you still want to share some separate group of actions. When this happens, you can always reference a shared flow, using a `@Jpf.SharedFlowRef`.

Is this a sneaky way to support multiple inheritance? We leave it for you to decide.