Page Flow Inheritance

Table of contents

1. Introduction

Page Flow inheritance is a powerful way to share actions, exception handlers, configuration, etc. among controller classes. The basic idea is simple: you use Java inheritance to share pieces of controller classes. This document shows ways in which you might use the feature, and also shows areas that go beyond what you might expect from standard Jave inheritance.

Note:

Both inheritance and Shared Flow offer ways to share actions and exception handlers. See Shared Flow vs. Inheritance for some guidelines on when to use each one.

2. Basic Inheritance

2.1. Inheriting Plain Members

If you derive from a base class, you inherit all its public/protected member variables and methods. Just as usual.

2.2. Inheriting Annotated Members

If you derive from a base class, then you inherit all its public/protected *annotated* members, like action methods (`@Jpf.Action`), exception handler methods (`@Jpf.ExceptionHander`), or shared flow fields (`@Jpf.SharedFlowField`). This is not surprising, but it is important to point out. Inheriting an action method means that you *inherit the action*. In the following example, `DerivedFlow` inherits its `begin` action from `BaseFlow`.

```
package base;
...
@Jpf.Controller
public class BaseFlow extends PageFlowController
{
    @Jpf.Action(
        forwards={
            @Jpf.Forward(name="index", path="index.jsp")
        }
    )
    public Forward begin()
    {
        return new Forward("index");
    }
}
```

```
package derived;
...
@Jpf.Controller
public class DerivedFlow extends BaseFlow
{
}
```

As usual, hitting /derived/DerivedFlow.jpf in your browser will execute the `begin` action on `DerivedFlow`. In this case, it executes the *inherited* `begin` action. Pretty simple.

> **Note:**
> You may have noticed that "index.jsp" is a local path, and you may have wondered whether hitting /derived/DerivedFlow.jpf will take you to /base/index.jsp or /derived/index.jsp. The page flow can actually be configured to work either way; see Local Paths, below.

2.3. Inheriting the @Jpf.Controller annotation

When you extend a base class controller, you inherit its `@Jpf.Controller` annotation in a special way: it is *merged* with the `@Jpf.Controller` annotation on your derived class. Here is a very simple example:

```
package base;
...
@Jpf.Controller(nested=true)
public class BaseFlow extends PageFlowController
{
    ...
}

package derived;
...
@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class DerivedFlow extends BaseFlow
{
}
```

The controller `DerivedFlow` inherits `nested=true` from the base class, and still keeps its `simpleActions`. It is as if `DerivedFlow` defined the following `@Jpf.Controller` annotation:

```
@Jpf.Controller(
    nested=true,
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
```

```
    }
)
```

A more common example involves merging of arrays of annotations, like `@Jpf.SimpleAction` or `@Jpf.Forward` . In the following example, `ChildFlow` inherits the simple action `baseAction` and a catch for `LoginException`.

```
package parent;
...
@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="page1.jsp"),
        @Jpf.SimpleAction(name="baseAction",
navigateTo=Jpf.NavigateTo.previousPage)
    },
    catches={
        @Jpf.Catch(type=LoginException.class, path="loginError.jsp")
    }
)
public class ParentFlow extends PageFlowController
{
}

package child;
...
@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    },
    catches={
        @Jpf.Catch(type=Exception.class, path="error.jsp")
    }
)
public class ChildFlow extends ParentFlow
{
}
```

It is as if `ChildFlow` was defined with the following `@Jpf.Controller` annotation:

```
@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="baseAction",
navigateTo=Jpf.NavigateTo.previousPage)
    },
    catches={
        @Jpf.Catch(type=LoginException.class, path="loginError.jsp"),
        @Jpf.Catch(type=Exception.class, path="error.jsp")
    }
)
```

The basic rule is that attributes (e.g, `nested=true`) are inherited, while arrays (e.g., `simpleActions={...}`) are inherited and merged. When there are conflicts, values

from the derived controller *override* values from the base controller, as you would expect.

3. Advanced Inheritance

3.1. Overriding

You may have noticed from the `ParentFlow`/`ChildFlow` example above that the `begin` action in `ChildFlow` overrode the one in `ParentFlow`. The simple rule is that any annotation or attribute within your `@Jpf.Controller` will override one of the same name/type in the base class. In the following example, the `@Jpf.Catch` for `LoginException` is overridden in `DerivedFlow`.

```
package base;
...
@Jpf.Controller(
    catches={
        @Jpf.Catch(type=LoginException.class, path="loginError.jsp")
    }
)
public class BaseFlow extends PageFlowController
{
}

package derived;
...
@Jpf.Controller(
    catches={
        @Jpf.Catch(type=LoginException.class,
method="handleLoginException")
    }
)
public class DerivedFlow extends BaseFlow
{
    @Jpf.ExceptionHandler(
        forwards={
            @Jpf.Forward(name="errorPage", path="error.jsp")
        }
    )
    public Forward handleLoginException(LoginException ex, String
actionName, String message, Object formBean)
    {
        ...
        return new Forward("errorPage");
    }
}
```

3.2. Abstract base classes

If you make your base controller class `abstract`, then you are free from some usual

restrictions:

- Even if it is a page flow controller, it does not need to have a `begin` action.
- Even if it has `nested=true` in `@Jpf.Controller`, it does not have to have at least one `@Jpf.Forward` or `@Jpf.SimpleAction` with a `returnAction` attribute. This would normally be required.
- If you have a local path (e.g., "index.jsp", which does not start with "/"), you will *not* receive a warning if the file does not exist. A derived page flow may have a local file with this name.
- It is not required to have the `@Jpf.Controller` annotation.

4. Local Paths

In a derived controller class, you may inherit an action that forwards to a local path (a path that does not begin with a "/"). In the following example, `DerivedFlow` inherits a `begin` action that forwards to "index.jsp":

```
package base;
...
@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class BaseFlow extends PageFlowController
{
}

package derived;
...
@Jpf.Controller
public class DerivedFlow extends BaseFlow
{
}
```

If you hit /derived/DerivedFlow.jpf, where do you end up? /base/index.jsp or /derived/index.jsp? By default, you end up at /derived/index.jsp; the local path is relative to the current page flow (/derived/DerivedFlow.jpf). You can change this behavior, though, by setting `inheritLocalPaths=true` in your derived class's `@Jpf.Controller` annotation, e.g.,

```
package derived;
...
@Jpf.Controller(inheritLocalPaths=true)
public class DerivedFlow extends BaseFlow
{
}
```

Now, if you hit /derived/DerivedFlow.jpf, you will see the content of /base/index.jsp.

Note:

Even when `inheritLocalPaths=true`, you won't leave (destroy) the current page flow by going to a path that's inherited from a base class.