Data binding to NetUI Implicit Objects

Table of contents

1. Introduction

In NetUI, data binding allows JSP tags or other UI technologies to read and write data available in the web-tier environment. This document discusses both the *implicit objects* that are available via NetUI to the JSP author and the *expression languages* that are used to bind UI objects to those implicit objects.

NetUI tags support binding to both *read-only* and *read-write* data. Read-only data is usually bound to tag attributes that simply display information on a page. Read-write data is bound with the intention of being updated from a web browser. Often, read-write data is displayed within an HTML form tag and bound to HTML text boxes, radio buttons, select boxes, and other HTML widget types. Each of these types of binding use the syntax of the JSP 2.0 Expression Language (EL) to express a binding from JSP tag to JavaBean property, Map member, List item, array element, and so on. The JSP 2.0 Expression Language is used to bind read-only data to tag attributes. This language is documented in detail here. For example, this example binds a NetUI `span` tag to a value from a JSP's PageContext attribute map:

```
<netui:span value="${pageScope.fooAttribute}"/>
```

Here, the JSP container evaluates the expression and invokes the `span` tag's `setValue` attribute method to pass the result to the tag. The JSP 2.0 EL is also able to perform simple arithmetic and boolean operations in expressions.

When using the NetUI JSP tags, read-write data is bound to NetUI JSP tags differently. NetUI tags use a derivation of the JSP 2.0 EL to refer to implicit objects in a JSP, but unlike the JSP 2.0 EL, the syntax is slightly different. For example, when binding a NetUI `textBox` tag to data that is meant to be read and then updated during an HTML form POST, the `textBox` tag might look like:

```
<netui:textBox dataSource="actionForm.userName"/>
```

This expression syntax is used on NetUI JSP tag attributes named `dataSource`. The expression syntax is necessary to allow the NetUI tags to know *both* the value of the expression and the expression text. The expression text is needed in order to write the tag's name in the HTML rendered to a web browser. For example, with a `userName` of "foo", the JSP tag above will render:

```
<input type="text" name="{actionForm.userName}" value="foo"/>
```

The expression text is used by the `<netui:textBox>` tag to render the value of the HTML `input`'s `name` attribute, and when the containing HTML form POSTs, this name is used to detect the presence of a NetUI expression that can then be used to update a property on a JavaBean or other data structure.

Best Practice

> In order to prevent POSTing data into JSP implicit objects such as `requestScope` or `sessionScope`, only a few NetUI implicit objects should be used in read/write expressions. These include `pageFlow`, `sharedFlow`, and `actionForm`.

2. JSP Implicit Objects

A JSP 2.0+ container exposes a set of implicit objects for use by JSP authors. These implicit objects are documented here. These can be used on any of the NetUI JSP tag attributes that accept runtime expressions. For example, in a webapp called `foo` the following JSP snippet uses the `pageContext` implicit object as a JavaBean to build a fully-qualified image path:

```
<netui:image value="${pageContext.request.contextPath}/images/banner.png"/>
```

This renders the following HTML markup:

```
<img source="/foo/images/banner.png"/>
```

The JSP container also makes implicit objects available that provide access to the attribute maps for the page context, request, session, and servlet context. By adding attributes to the page context, request, and session, webapp developers can add their own implicit objects. In the following example, a JavaBean of type `Widget` is added to the request in a page flow action:

```
getRequest().setAttribute("widget", fooWidget);
```

Then, this JSP snippet uses the expression language to data bind to the Widget's `density` property:

```
The density is: ${widget.density}
```

This is effectively the same as writing code that does:

```
The density is: <%= ((Widget)request.getAttribute("widget")).getDensity()
%>
```

3. NetUI Implicit Objects

In addition to the implicit objects that the JSP container provides, the NetUI runtime provides an additional set of objects that are available when using certain NetUI features. Not all of the implicit objects are always available -- for example, the `actionForm` implicit object is only available when used inside of a `<netui:form` tag for accessing the form's associated form bean.

A summary of the NetUI implicit objects is shown in the table below; details are available further down this document.

Implicit Object	Context	Description
`actionForm`	Within the `<netui:form>` tag	Provides access to the properties of a JavaBean used as the form bean for an HTML

		form.
`bundle`	Inside of a JSP where the `<netui-data:declareBundl` tag is used to refer to a resource bundle or where a JSP is part of a page flow that has resource bundles declared with the `@Jpf.MessageBundle` annotation.	Provides access to message strings contained in a Java properties file. Strings are referred to by name in the expression.
`container`	The `container` implicit object is available inside of several NetUI JSP tags that "repeat" over data sets including the `<netui-data:dataGrid`, `<netui-data:repeater`, `<netui-data:cellRepeater` `<netui:select`, `<netui:checkBoxGroup`, and `<netui:radioButtonGroup`,	Provides access to the JavaBean properties exposed by the `IDataAccessProvider` interface. Implementations of this interface are made available to the PageContext during rendering so that tag bodies can access information about the current data item, the item's index, and so on.
`pageFlow`	Available to any JSP that is part of a page flow.	The `pageFlow` implicit object provides access to the current page flow controller instance as a JavaBean. This allows a page flow controller to expose properties to JSPs.
`pageInput`	Available to any JSP that was reached by a page flow Forward object that had *page inputs* attached to the Forward.	Page flows allow *action outputs* to be attached to Forward objects as a way to provide a data contract between a page flow action and a page. This ensures that all actions that forward to JSPs provide the JSP with the appropriate data and that all JSPs receive the correct data. This data contract is validated at both the action and at the JSP when using the `<netui-data:declarePageInput>`. tag.
`sharedFlow`	Available to any JSP that is part of a page flow which	The `sharedFlow` implicit object provides access to any

	references shared flows.	JavaBean properties on Shared Flows associated with the current page flow. This allows Shared Flows to expose properties to JSPs.

3.1. NetUI Implicit Object Details

3.2. actionForm

The `actionForm` implicit object is a convenient way to explicitly reference a JavaBean used for authoring HTML forms. This implicit object is available only inside of `<netui:form>` tags with `action` attributes that reference page flow actions accepting a JavaBean. The `actionForm` implicit object allows data binding to JavaBean properties, Map attributes, Lists, and arrays as with any other implicit object. This example shows a JSP that contains a form which POSTs to a page flow action that accepts a JavaBean `NameForm`.

The JavaBean:

```
public class NameForm {
    private String _name;
    public String getName() {
        return _name;
    }

    public void setName(String name) {
    _name = name;
    }
}
```

The JSP:

```
<netui:form action="submitNameform">
    <netui:textBox dataSource="actionForm.name"/><br/>
    <netui:button value="Submit"/>
</netui:form>
```

The page flow action `submitNameForm`:

```
@Jpf.Action()
public Forward submitNameForm(NameForm form) {
    ...
}
```

Here, the `dataSource`'s `actionForm.name` expression refers to the value of the `NameForm`'s `name` property. The result is data bound to the `textBox` tag. When the form is submitted to the server, the request parameter `{actionForm.name}` is applied to the action form which is then passed to the `submitNameForm` action.

3.3. bundle

The `bundle` implicit object is useful for binding UI to localized message strings. The `bundle` implicit object is available in one of two situations:

- when declaring a resource bundle accessible to a JSP via the `<netui-data:declareBundle>` tag.
- when a page flow with which the JSP is associated exposes resource bundles via the `@Jpf.MessageBundle` annotation.

The `declareBundle` JSP tag is used to make a specific resource bundle available to a JSP. For example, given the following resource bundle and JSP, a page can data bind to messages in the resource bundle using the JSP 2.0 EL.

The resource bundle, which is located in `WEB-INF/classes/org/foo/messages.properties`:

```
message1=This is the first message
message2=Another message
```

The JSP can declare this resource bundle to be available to the page using this JSP snippet:

```
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0"
prefix="netui-data"%>

<netui-data:declareBundle name="fooMessages"
bundlePath="org/foo/messages"/>
```

Finally, messages in the JSP can be data bound with JSP literal text and tags:

```
<netui:span value="${bundle.fooMessages.message1}"/>

${bundle.fooMessages.message2}
```

The expressions above contain a reference to the `bundle` implicit object. Then, the specific bundle name is referred to with `fooMessages`; this name must match the value of a `name` attribute of a `declareBundle` tag or the name of a bundle declared in a page flow controller. Finally, the expressions use `message1` and `message2` to refer to message keys in the `messages.properties` file.

Resource bundles can also be registered with the `bundle` implicit object by using the `@Jpf.MessageBundle` class-level annotation. This allows a page flow to also integrate with the implicit message resources object which is available via a Struts module. These resource bundles will be available to all JSPs that are part of a page flow without having to use the `declareBundle` JSP tag. Because a default resource bundle can be associated with a Struts module, the bundle name `default` is reserved for referencing this bundle. For example, the following page flow controller declares the resource bundle above as the default page flow bundle:

```
@Jpf.Controller(
    forwards = {...}
    messageBundles = {
        @Jpf.MessageBundle(bundlePath="org.foo.messages")
    }
)
public class Controller
    extends PageFlowController {
    ...
}
```

Message strings from this bundle can be referred to in JSPs with an expression like:

```
${bundle.default.message1}
```

A resource bundle can also be registered with a specific name by adding an annotation like:

```
@Jpf.Controller(
    forwards = {...}
    messageBundles = {
        @Jpf.MessageBundle(bundleName="jpfBundle",
bundlePath="org.foo.messages")
    }
)
public class Controller
    extends PageFlowController {
    ...
}
```

Then, the same message strings from the previous two examples are available with an expression like

```
${bundle.jpfBundle.message1}
```

Each of these three ways to register a resource bundle (JSP tag, implicit page flow bundle, and explicit page flow bundle) can be used together in a single page flow.

3.4. container

The `container` implicit object is available to a JSP when inside of several NetUI tags that *repeat* over a Map, List, array, or various other kinds of data sets. Generally, JSP tags that can be bound to data sets iterate through them from start to end as in a Java `for` loop. Such JSP tags include:

- `<netui-data:dataGrid>`
- `<netui-data:repeater>`
- `<netui-data:cellRepeater>`
- `<netui:select>`
- `<netui:checkBoxGroup>`
- `<netui:radioButtonGroup>`.

The `container` implicit object provides access to the current item in the data set and to metadata about the current iteration. This access is based on the properties available on the `IDataAccessProvider` interface and includes:

Property Name	Description
`item`	Refers to the current data item. In an array of `Widget` beans, the JavaBean propeties of widget can be accessed with an expression like `${container.item.density}`
`index`	Refers to the current index of iteration. Tags are free to define their own rules for the the value of the `index` property, but in general, this is a zero-based index that increments each time the JSP tag renders its body / iterates to the next data item.
`container`	Refers to an outer repeating container. This value is used when two repeating containers are nested and the inner container needs to access the current item in the outer container. For example, this might be used when rendering hierarchical data sets.

The following example uses the NetUI `<netui-data:repeater>` to iterate over an array of `Widget` beans displaying each Bean's name and density properties and their index in the array.

```
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0"
prefix="netui-data"%>

<table>
<tr><th>Index</th><th>Name</th><th>Density</th></tr>
<netui-data:repeater dataSource="requestScope.widgetBeanArray">
<tr><td>${container.index}</td><td>${container.item.name}</td><td>${container.item.dens
</netui-data:repeater>
</table>
```

Notice in this example how the `repeater` tag has a `dataSource` attribute that references the data set to iterate through. The `dataSource` attribute requires the use of the NetUI Expression Language because the `repeater` can be used to render editing UI for data sets. For example, in the case of rendering a shopping cart, the repeater can be used to render each item in the cart with a `<netui:textBox>` for editing the quantity of each item. An example of this can be found in the Beehive sample webapp called `netui-samples` under the `ui/repeaterediting/` directory.

3.5. pageFlow

The `pageFlow` implicit object is used to refer to the current page flow controller as a JavaBean. If there is no page flow present, the `pageFlow` implicit object will not be available for data binding. For example, if a page flow controller exposes a `username` property as:

```
@Jpf.Controller(
    forwards={@Jpf.Forward(name="index", path="index.jsp")}
)
public class Controller {
    private String _username = null;

    public String getUsername() {
        return _username;
    }

    @Jpf.Action()
    public Forward begin() {
        _username = "Foo Bar";
        return new Forward("index");
    }
}
```

The `username` property can be data bound in `index.jsp` as:

```
${pageFlow.username}
```

Best Practice

Because mutable JavaBean properties can be updated via an HTML form POST, JavaBean properties exposed by a page flow should usually be read-only unless the page flow itself is being used as a form bean.

3.6. pageInput

The `pageInput` implicit object is used to refer to a `Map` of objects that are passed via a `Forward` from a page flow action to a JSP. Use of Page Inputs consists of two parts -- the first are called *action outputs* and the second are called *page inputs*. Action outputs are passed from page flow actions to pages via the action's `Forward` object. page flow actions use Java annotations to declare a validatable data contract that ensures that an action passes the correct data via a `Forward`. At the page, this data is called a *page input* and can again be checked to ensure that the page receives the data necessary to render successfully.

This example shows a page flow action that passes an action output of type `Widget` to a JSP which data binds to the `density` property on the `Widget`.

The page flow controller:

```
@Jpf.Controller()
public class Controller
    extends PageFlowController {
```

```
    @Jpf.Action(
        forwards={@Jpf.Forward(name="success",
                               path="index.jsp",
actionOutputs={@Jpf.ActionOutput(name="theWidget", type=Widget.class,
required=true)}
                              )
        }
    )
    protected Forward begin() {
        Widget widget = new Widget();
        widget.setDensity(3.14);
        Forward f = new Forward("success");
        f.addActionOutput("theWidget", widget);
        return f;
    }
}
```

Notice here that the action has added an action output to the `Forward` via the `addActionOutput` method call. The NetUI runtime will then validate the data contract declared in the annotations against the returned forward object. If validation fails, a runtime error will be displayed in the browser.

The JSP:

```
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0"
prefix="netui-data"%>

<netui-data:declarePageInput name="theWidget" type="org.foo.Widget"/>

${pageInput.theWidget.density}
```

Notice here that the page has used a `declarePageInput` tag to ensure that the `Widget` entering the page is both present in the set of page inputs and is non-null. Given this information, the JSP then refers to properties on the `Widget` via the expression language.

Action outputs and page inputs can be used with or without validation; to disable action output validation, simply remove any action output annotations from a page flow action. To disable page input validation in a JSP, remove any `declarePageInput` tags from the JSP. The APIs to add action outputs to `Forward` objects and to refer to them via the `pageInput` implicit object will continue to work without any data contract validation.

3.7. sharedFlow

The `sharedFlow` implicit object is used to refer to properties of shared flow objects that are associated with the current page flow. If there is no page flow present, the `sharedFlow` implicit object will not be available for data binding. In order for a shared flow to be available for data binding, it must be registered with a page flow by type; additionally, it is registered with a name that will uniquely identify it in the set of shared flows associated with a page flow. More information on shared flows can be found here.

The following example shows a shared flow, a page flow that uses the Shared Flow, and a JSP that uses the JSP 2.0 EL to data bind to a JavaBean propety of the shared flow.

The Shared Flow:

```
package org.foo;

import org.apache.beehive.netui.pageflow.SharedFlowController;

@Jpf.Controller()
public class SharedFlow
   extends SharedFlowController {

    private String _sharedMessage = null;

    public String getSharedMessage() {
        return _sharedMessage;
    }
}
```

The page flow controller:

```
@Jpf.Controller(
    sharedFlowRefs={@Jpf.SharedFlowRef(name="aSharedFlow",
type=org.foo.SharedFlow.class)}
)
public class Controller
   extends PageFlowController {

    ...
}
```

Above, the page flow controller adds an explicit reference to the shared flow controller `org.foo.SharedFlow` defined above.

The JSP:

```
${sharedFlow.aSharedFlow.sharedMessage}
```

In a JSP whose current page flow controller is the `Controller` class defined above, the JSP has access to all of the shared flows associated to the page flow via its `@Jpf.SharedFlowRef` annotation. The shared flow can then be referenced by the `name` attribute of the `SharedFlowRef` annotation. In this case, the name `aSharedFlow` is used in the JSP 2.0 expression to refer to the `sharedFlow`'s `sharedMessage` property.

4. Expression Language Details

Due to limitations in the JSP 2.0 Expression Language specification, expressions can not be used to reference read-write data because the JSP tag itself can never obtain both the expression text and the value of the expression. In NetUI, both the value and expression text are required to bind to editable data because the expression text is written into the JSP as the HTML `name` attribute of HTML form `input` fields.