

ADF Code Corner

011. ADF Faces RC - How-to use the Client and Server Listener Component

ORACLE
CODE CORNER



twitter.com/adfcodecorner

Abstract:

One of the most interesting features in Ajax is the ability of an application to send user information to the server without user recognition. A positive use of this feature is the auto suggest feature, which provides the user with a list of possible select values based on the user input into a text field. Another positive use is auto completion of strings, in which case the user enters a shortcut into a text field e.g. "NY" for the system to complete the field with New York. This how-to document shows how this Ajax functionality is achieved in ADF Faces RC using the client and server listener component.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
14-JUN-2008

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: <http://forums.oracle.com/forums/forum.jspa?forumID=83>

Introduction

In this how-to example, we explore a negative use of the Ajax asynchronous messaging functionality, which is the ability of an application to spy on the user input.

Before you can shout "security breach", let me reassure that this application doesn't cause any harm. I just chose this example because its cool. The sample code, as usual, can be downloaded at the end of this how-to document.

The sample application consist of two text fields that each intercepts the user input to send it to the server as the user types. The difference between the fields is that the first field has the functionality configured at designtime, whereas the second has it configured dynamically through a managed bean. The latter thus allows you to provide this functionality dynamically: your application may contain a checkbox that if checked provides text auto completion as otherwise it wont.

Let me spy on you: Please enter your mail password

I am a spy too!

As soon as the user types into one of the fields, the characters are send to the server where they can be accessed from a managed bean method.

Configuring a clientListener and serverListener at designtime

At designtime you use the following artifacts to configure asynchronous messages to be send to the server

- af:clientListener
- af:serverListener
- JavaScript method called by the client listener

```
<af:document>  
  <f:facet name="metaContainer">
```

```
<af:group>
  <![CDATA[
    <script>
      function clientMethodCall(event) {
        component = event.getSource();
        AdfCustomEvent.queue(component,
          "customEvent",
          {payload:component.getSubmittedValue()} ,
          true);
        event.cancel();
      }
    </script> ]]>
</af:group>
</f:facet>
<af:form>
  <af:panelFormLayout>
    <f:facet name="footer"/>
    <af:inputText label="Let me spy on you: Please enter your
      mail password">
      <af:clientListener method="clientMethodCall" type="keyUp"/>
      <af:serverListener type="customEvent"
        method="#{customBean.handleRequest}"/>
    </af:inputText>
  </af:panelFormLayout>
</af:form>

```

...

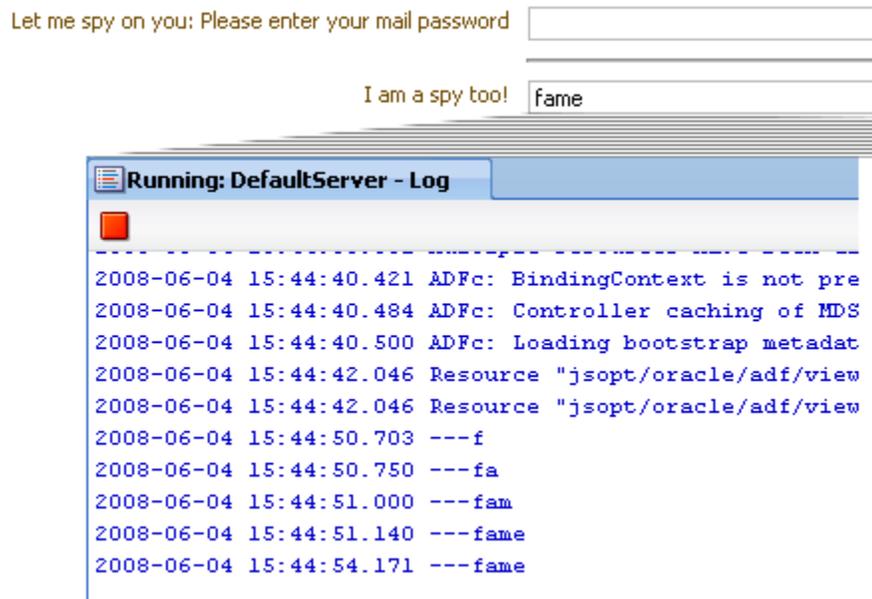
The `af:clientListener` calls the JavaScript method on the page whenever the `keyUp` JavaScript event occurs on the text field. The JavaScript function then gets a hold onto the event source, the text field, to read its submitted text value. The text then is passed to the server listener, which listens under the name of "customEvent".

The bean method, which is mapped by the `serverListener` component, is `customBean.handleRequest`:

```
public void handleRequest(ClientEvent event){

    System.out.println("---"+event.getParameters().get("payload"));

}
```



The managed bean can be configured either in the faces-config.xml file or the adfc-config.xml file, which is the Oracle taskflow configuration. All that the method does in the above example is to print out the user entered values. However, as you see you can access the payload by requesting it as a parameter on the event object passed into the method. Note that the term "payload" is arbitrarily chosen by me. You are free to choose whatever name pleases you.

Configuring a clientListener and serverListener at runtime

To apply a client and server listener dynamically, an additional step is required, which is to create a binding reference of the text field to the managed bean that adds the listeners to it.

```
<af:inputText label="I am a spy too!"
    binding="#{customBean.secondSpyField}"/>
```

As you can see, the definition of the text field doesn't contain the clientListener and serverListener elements, but instead has the binding property set to the managed bean. To add the client and server listener on page load, the following code in the managed bean is required

```
public void setSecondSpyField(RichInputText secondSpyField) {
    this.secondSpyField = secondSpyField;
    if (this.secondSpyField!=null){
        ClientListenerSet cls = new ClientListenerSet();
        cls.addListener("keyUp", "clientMethodCall");

        cls.addCustomServerListener(
            "customEvent",
            getMethodExpression("#{customBean.handleRequest}"));
        this.secondSpyField.setClientListeners(cls);
    }
}
```

```
public RichInputText getSecondSpyField() {
    return secondSpyField;
}

public MethodExpression getMethodExpression(String s){
    FacesContext fc = FacesContext.getCurrentInstance();
    ELContext elctx = fc.getELContext();
    ExpressionFactory elFactory =
        fc.getApplication().getExpressionFactory();
    MethodExpression methodExpr =
        elFactory.createMethodExpression(elctx, s, null, new
        Class[]{ClientEvent.class});
    return methodExpr;
}
```

Within the setter of the text field binding, the two listeners are added. The JavaScript function that handles the client method call is assumed to be on the page. There are ways to dynamically add JavaScript to a page using the extended renderkit service, but for this example we assume the JavaScript function to exist.

The key for making the dynamic approach work is within the `getMethodExpression` function. When building the method expression you need to tell the method's input arguments, which in the case of the server listener is of type `ClientEvent`.

Download

You can download the sample explained in this article from the ADF Code Corner site:

<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html>

RELATED DOCUMENTATION

	af:clientListener - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_clientListener.html
	af:serverListener - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e12419/tagdoc/af_serverListener.html