Table of Contents

Motivation

Motivation: Why Ajax?

- **HTML and HTTP are weak**
  - Non-interactive
  - Coarse-grained updates
- **Everyone wants to use a browser**
  - Not a custom application
- **Real browser-based active content**
  - Failed: Java Applets
    (not universally supported)
  - Serious alternative: Flash (and Flex)
    Not universally supported; limited power

Installation and Setup

Installation and Setup

Installing Ajax4jsf

- **Download the latest binary version**
  - `http://labs.jboss.com/jbossajax4jsf/downloads/`
- **Unzip**
  - Into any location
- **Install ajax4jsf.jar**
  - Copy `/lib/ajax4jsf.jar` into `WEB-INF/lib` directory
- **Add filter to** `web.xml`
  - See next page

Filter Settings for web.xml

```
<?xml version="1.0"?>
<web-app ...>
  ...
  <filter>
    <display-name>Ajax4jsf Filter</display-name>
    <filter-name>ajax4jsf</filter-name>
    <filter-class>org.ajax4jsf.Filter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>ajax4jsf</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
  </filter-mapping>
</web-app>
```

Using Ajax4jsf

- **Use xhtml**
  - Most Ajax applications use xhtml, not HTML 4
- **Add taglib for ajax4jsf**

  <%@ taglib uri="https://ajax4jsf.dev.java.net/ajax"
  prefix="a4j" %>

- **Give ids to sections that you want to update**

  <h:outputText ... id="someName" />

  Note: ids within a page must be unique
- **Use a4j: tags**
  - Almost all must go inside h:form
    Even the ones that don't use form elements (e.g., a4j:poll)

Basic Template

```
<?xml version="1.0" encoding="UTF-8"?>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax"
           prefix="a4j"%>
<f:view>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ↘
→Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.↘
  →dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<meta http-equiv="content-type"
      content="text/html; charset=UTF-8" />
<head><title>Some Title</title>
</head>
<body>
```

XHTML: Case

- **In HTML 4, case does not matter for tag names and attribute names**
  - `<BODY>`, `<Body>`, and `<body>` are equivalent
  - `<H1 ALIGN="...">` is equivalent to `<H1 aLiGn="...">`
- **In xhtml, tag names and attribute names must be in lower case**
  - `<body>`, `<h1 align="...">`

XHTML: Quotes

- **In HTML 4, quotes are optional if attribute value contains only alphanumeric values**
  - `<H1 ALIGN="LEFT">` or `<H1 ALIGN=LEFT>`
- **In xhtml, you must always use single or double quotes**
  - `<h1 align="left">` or `<h1 align='left'>`

XHTML: End Tags

- **HTML 4**
  - Some tags are containers
    `<H1>...</H1>`, `<A HREF...>...</A>`
  - Some tags are standalone
    `<BR>`, `<HR>`
  - Some tags have optional end tags
    `<P>`, `<LI>`, `<TR>`, `<TD>`, `<TH>`
- **XHTML**
  - All tags are containers. End tags always required.
    `<p>...</p>`, `<li>...</li>`
  - If there is no body content, start/end tags can be merged
    `<br></br>` or just `<br/>`

Ajax and the Firefox JavaScript Console

- **Invoke with Control-Shift-J**
- **Also see Venkman JavaScript debugger**
  - `http://www.mozilla.org/projects/venkman/`
  - `https://addons.mozilla.org/firefox/216/`

Main Ajax4jsf Tags

Main Ajax4jsf Tags

Tag Summary

- `a4j:commandButton` **and** `a4j:commandLink`
  - Run code on the server, then update specified JSF element (or comma separated element list) after.

```
<a4j:commandButton action="#{bean.method}"
                   value="Button label" reReRender="some-id"↘
                   →/>
... <h:outputText value="#{bean.prop}" id="some-id"↘
→/>
```

- **a4j:poll**
  - Run code periodically on server, then update specified JSF element(s)
- **a4j:support**
  - Capture JavaScript event in any existing JSF control and invoke server-side code, then update specified element(s)

```
<h:inputText ...>
  <a4j:support event="onkeyup" reRender="some-id" />
</h:inputText>
```

a4j:commandButton and a4j:commandLink

a4j:commandButton: Basic Syntax

```
<a4j:commandButton
        action="#{bean.method}"
        value="Button label"
        reRender="some-id" />
```

- When you press the button, send a behind-the-scenes, asynchronous HTTP call to server, and run this method. This method should look like a normal action controller (i.e., no arguments and returns a String), but the return value is ignored (i.e., is not used to match a navigation rule in faces config).
- After action method is executed, re-evaluate and re-display the JSF element that has this id. The point is that this JSF element should output something that changed as a result of the call to the action controller.

Reminder: Template

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j↘
→"%>
<f:view>
<!DOCTYPE html PUBLIC "-//W3C//DTD␣XHTML␣1.0␣↘
→Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.↘
    →dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;↘
→charset=UTF-8""␣/>
<link␣rel="stylesheet"
␣␣␣␣href="./css/styles.css"
␣␣␣␣type="text/css" />
<title>Ajax4JSF Examples</title>
```

h:commandButton: Example

```
<tr><td width="100">
        <h:form>
          <a4j:commandButton
                    action="#{numBean.makeResult}"
                    value="Show Random Number"
                    reRender="region1" />
        </h:form>
      </td>
      <td width="100">
        <h:outputText value="#{numBean.result}"
                      id="region1" />
      </td>
 </tr>
```

h:commandLink: Example

```
<tr><td width="100">
        <h:form>
          <a4j:commandLink
                    action="#{numBean.makeResult}"
                    value="Show Random Number"
                    reRender="region2" />
        </h:form>
      </td>
      <td width="100">
        <h:outputText value="#{numBean.result}"
                      id="region2" />
      </td>
 </tr>
```

a4j:commandButton and a4j:commandLink: Bean

```
public class RandomNumberBean {
  private int range = 1;
  private double result;
  public int getRange() {
    return(range);
  }
  public void setRange(int range) {
    this.range = range;
  }
  public String makeResult() {
    result = Math.random() * range;
    return(null);
  }
  public double getResult() {
    return(result);
```

a4j:commandButton and a4j:commandLink: faces-config

```
<managed-bean>
  <managed-bean-name>
    numBean
  </managed-bean-name>
  <managed-bean-class>
    coreservlets.RandomNumberBean
  </managed-bean-class>
  <managed-bean-scope>
    request
  </managed-bean-scope>
</managed-bean>
```

a4j:commandButton: Example 2

- **a4j:commandButton results in normal JSF request processing cycle**
  - So, form is still submitted
  - Bean setter methods still execute
  - Action controller runs after setter methods, so controller has access to results of all form fields

a4j:commandButton: Example 2

```
<tr><td width="100">
        <h:form>
          Range:
          <h:inputText value="#{numBean.range}"
                       size="5" /><br/>
          <a4j:commandButton
                  action="#{numBean.makeResult}"
                  value="Show Random Number"
                  reRender="region3" />
        </h:form>
      </td>
      <td width="100">
        <h:outputText value="#{numBean.result}"
                      id="region3" />
      </td>
 </tr>
```

a4j:commandButton and a4j:commandLink: Bean

```
public class RandomNumberBean {
  private int range = 1;
  private double result;
  public int getRange() {
    return(range);
  }
  public void setRange(int range) {
    this.range = range;
  }
  public String makeResult() {
    result = Math.random() * range;
    return(null);
  }
  public double getResult() {
    return(result);
```

Example

```
http://localhost:
8080/ajax4jsf-coreservlets/welcome.faces
```

Limitations on Use of h:outputText with Ajax4jsf

- **In JSF, the following is perfectly legal**

```
<body bgcolor="<h:outputText␣#{myBean.fgColor}" />"↘
→>
␣␣
```

result is

```
<body bgcolor="red">
```

- **But the following is illegal**

```
 <body bgcolor=
        "<h:outputText␣#{myBean.fgColor}" id="foo" />↘
        →">
␣␣
```

Results in

```
 <body␣bgcolor="<span␣id="foo">red</span>">
```

a4j:poll

a4j:poll: Basic Syntax

```
<a4j:poll
    interval="x"
    reRender="some-id" />
```

- **Interpretation**
  - Every x milliseconds, send off an asynchronous HTTP request to the server.
  - Re-evaluate and re-display the JSF element with the id some-id.
    Key point: this element should result from code that gives different values at different times
    Unlike with `a4j:commandButton`, there is no explicit additional server-side method to run

a4j:poll: Example

```
<h:form>
  <a4j:poll interval="5000"
            reRender="timeDisplay" />
  <h2>
    <h:outputText value="#{timeBean.time}"
                  id="timeDisplay" />
  </h2>
</h:form>
```

- **The `getTime` method returns different results at different times.**

a4j:poll: Bean

```
public class TimeBean {
   public Date getTime() {
      return(new Date());
   }
}
```

a4j:poll: faces-config.xml

```
<managed-bean>
  <managed-bean-name>
    timeBean
  </managed-bean-name>
  <managed-bean-class>
    coreservlets.TimeBean
  </managed-bean-class>
  <managed-bean-scope>
    application
  </managed-bean-scope>
</managed-bean>
```

- **TimeBean has no state, so for efficiency, reuse the same instance.**

a4j:poll: Results

```
http://localhost:
8080/ajax4jsf-coreservlets/welcome.faces
```

a4j:support

a4j:support: Basic Syntax

```
<h:someTag ...>
  <a4j:support event="javascript-event"
               reRender="some-id">
</h:someTag>
```

- **Interpretation**
  - Do whatever someTag normally does, but if the specified JavaScript event (onclick, onchange, onkeypress, etc.) occurs, send a behind-the-scenes asynchronous HTTP request to the server that results in the specified JSF element being re-evaluated and re-displayed
  - Note: form values are sent and setter methods are run

JavaScript Event Handlers used with a4j:support

- `onchange` User changes element (IE: and element loses focus)
- `onclick/ondblclick` User single/double clicks form element or link
- `onfocus/onblur` Element receives/loses focus
- `onkeydown/onkeypress/onkeyup` User presses/presses-or-holds/releases a key
- `onmousedown/onmouseup` User presses/releases mouse
- `onmousemove` User moves mouse
- `onmouseover/onmouseout` User moves mouse onto/off area or link
- `onselect` User selects text within a textfield or textarea
- `onsubmit` User submits form

a4j:support: Example 1

- **Idea**
  - Use `h:inputText` to make a textfield
  - As the user types into the textfield, copy the value into regular text
- **Approach**
  - Textfield
    Stores result in `myBean.message`
    ```
    <a4j:support event="onkeyup"
    reRender="output-region"/>
    ```
  - Separate output field

    ```
    <h:outputText value="#{myBean.message}"
                  id="output-region" />
    ```

a4j:support: Example

```
<h:form>
    <table border="1">
      <tr><th>Textfield</th>
          <th>Ajax Value</th>
      </tr>
      <tr><td width="100">
          <h:inputText value="#{myBean.message}">
             <a4j:support event="onkeyup"
                          reRender="output-region" />
          </h:inputText></td>
          <td width="100">
          <h:outputText value="#{myBean.message}"
                        id="output-region" /></td>
      </tr>
    </table>
 </h:form>
```

a4j:support: Bean

```
public class MessageBean {
   private String message;
   public String getMessage() {
      return(message);
   }
   public void setMessage(String message) {
      this.message = message;
   }
}
```

a4j:support: faces-config.xml

```
<managed-bean>
    <managed-bean-name>
      myBean
    </managed-bean-name>
    <managed-bean-class>
      coreservlets.MessageBean
    </managed-bean-class>
    <managed-bean-scope>
      request
    </managed-bean-scope>
 </managed-bean>
```

a4j:support: Example 2

- **Idea**
  - Use `h:selectOneMenu` to make a list of US states
  - When the user selects a state, a list of corresponding cities is shown (again, using `h:selectOneMenu`)
  - When city selected, population of that city is displayed
- **Approach**
  - State list: `<a4j:support event="onchange" reRender="city-list"/>`
  - City List: `<a4j:support event="onchange" reRender="pop-field"/>`
  - Bean: **Make it session scoped** so values persist across across multiple submissions (since there are at least two)

a4j:support: Example

```
<h:form>
  State:
  <h:selectOneMenu value="#{locationBean.state}">
    <f:selectItems value="#{locationBean.states}" />
    <a4j:support event="onchange" reRender="cityList" />
  </h:selectOneMenu><br/>
  City:
  <h:selectOneMenu value="#{locationBean.city}"
                   disabled="#{locationBean.cityListDisabled}"
                   id="cityList">
    <f:selectItems value="#{locationBean.cities}" />
    <a4j:support event="onchange" reRender="population" />
  </h:selectOneMenu><br/>
  Population:
  <h:outputText value="#{locationBean.city}"
                      escape="false"
                      id="population" /></h:form>
```

a4j:support: Bean

```
public class LocationBean implements Serializable {
 private String state;
 private String city;
 // Make city list disabled initially.
 private boolean isCityListDisabled = true;
 public String getState() { return (state); }
 public void setState(String state) {
    this.state = state;
    isCityListDisabled = false;
 }
 public String getCity() { return(city); }
 public void setCity(String city) { this.city = city; }
 public boolean isCityListDisabled() {
    return(isCityListDisabled);
 }
```

a4j:support: Bean (Continued)

```
...
  public List<SelectItem> getStates() {
      List<SelectItem> states =
        new ArrayList<SelectItem>();
      states.add(
          new SelectItem("---␣Select␣State␣---"));
      for(StateInfo stateData:
           StateInfo.getNearbyStates()) {
        states.add(
            new SelectItem(stateData.getStateName()));
      }
      return(states);
   }
```

- **Put dummy value at the top of the list so that any real user selection is considered a change**

a4j:support: Bean (Continued)

```
public SelectItem[] getCities() {
    SelectItem[] cities =
      { new SelectItem("---␣Choose␣City␣---")};
    if(!isCityListDisabled && (state != null)) {
      for(StateInfo stateData:
            StateInfo.getNearbyStates()) {
        if(state.equals(stateData.getStateName())) {
          cities = stateData.getCities();
          break;
        }
      }
    }
    return(cities);
 }
```

- `state` = Result of form submission (i.e., value form the state list)

a4j:support: Supporting Class (StateInfo)

```
public class StateInfo {
  private String stateName;
  private SelectItem[] cities;
  public StateInfo(String stateName,
                   SelectItem[] cities) {
    this.stateName = stateName;
    this.cities = cities;
  }
  public String getStateName() {
    return(stateName);
  }
  public SelectItem[] getCities() {
    return(cities);
  }
```

a4j:support: Supporting Class (StateInfo) Continued

```
private static StateInfo[] nearbyStates =
  { new StateInfo("Maryland",
          new SelectItem("<i>unknown</i>",
                         "---␣Choose␣City␣---"),
          new SelectItem("635815", "Baltimore"),
          new SelectItem("57907", "Frederick"),
          new SelectItem("57698", "Gaithersburg"),
          new SelectItem("57402", "Rockville")),
    ...
```

- **Number** = Values that are used when an entry is selected and form is submitted. These are the values displayed in the text when the city field changes
- **Name** = Values that are displayed in the list

a4j:support: faces-config.xml

```
<managed-bean>
  <managed-bean-name>
    locationBean
  </managed-bean-name>
  <managed-bean-class>
    coreservlets.LocationBean
  </managed-bean-class>
  <managed-bean-scope>
    session
  </managed-bean-scope>
 </managed-bean>
```

a4j:support: Results

```
http://localhost:
8080/ajax4jsf-coreservlets/welcome.faces
```

Other Ajax Tools

Other Ajax Tools

JavaServer Faces (JSF) component libraries

- **Trinidad (formerly Oracle ADF)**
  - `http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/` (also `myfaces.apache.org`)
- **Tomahawk**
  - `http://myfaces.apache.org/tomahawk/`
- **Ajax4jsf**
  - `http://labs.jboss.com/jbossajax4jsf/`
- **IceFaces**
  - `http://www.icefaces.org/`

Summary

- `a4j:commandButton / a4j:commandLink`

```
<a4j:commandButton action="#{bean.method}"
                   value="Button label"
                   reRender="some-id" />
<a4j:commandLink action="#{bean.method}"
                 value="Link text"
                  reRender="some-id" />
```

- `a4j:poll`

```
<a4j:poll interval="x-milliseconds"
          reRender="some-id" />
```

- `a4j:support`

```
<h:someTag ...>
  <a4j:support event="javascript-event"
               reRender="some-id">
</h:someTag>
```

Conclusion

- **Ajax4jsf can be installed without modifying the existing pages**
  - Download a jar
  - modify your `web.xml` to install filters
  - Insert new tags (or properties) in existing pages
- **Ajax4Jsf provides 4 tags**
  - `a4j:commandButton`, `a4j:commandLink`, `a4j:poll`, `a4j:support`.
- **Uses the Event handling procedures of JSF**
  - Events are fired even when no action is done.
  - Components can be modified without action and navigation

References

- `http://www.coreservlets.com/JSF-Tutorial/`