Ajax with Java
- Java Webparts -

Manfred Wolff, wolff@manfred-wolff.de
Oktober 2006 / January 2007

Contents

1 Einleitung

Ajax (Asynchronous JavaScript and XML) ist, wie der Name schon sagt, verbunden mit der JavaScript-Technologie. Mit Hilfe von JavaScript werden XML-Fragmente an den Server gesendet und nach Empfang der Nachricht geparst, um dann die Ergebnisse an die entsprechende Stelle zu legen.

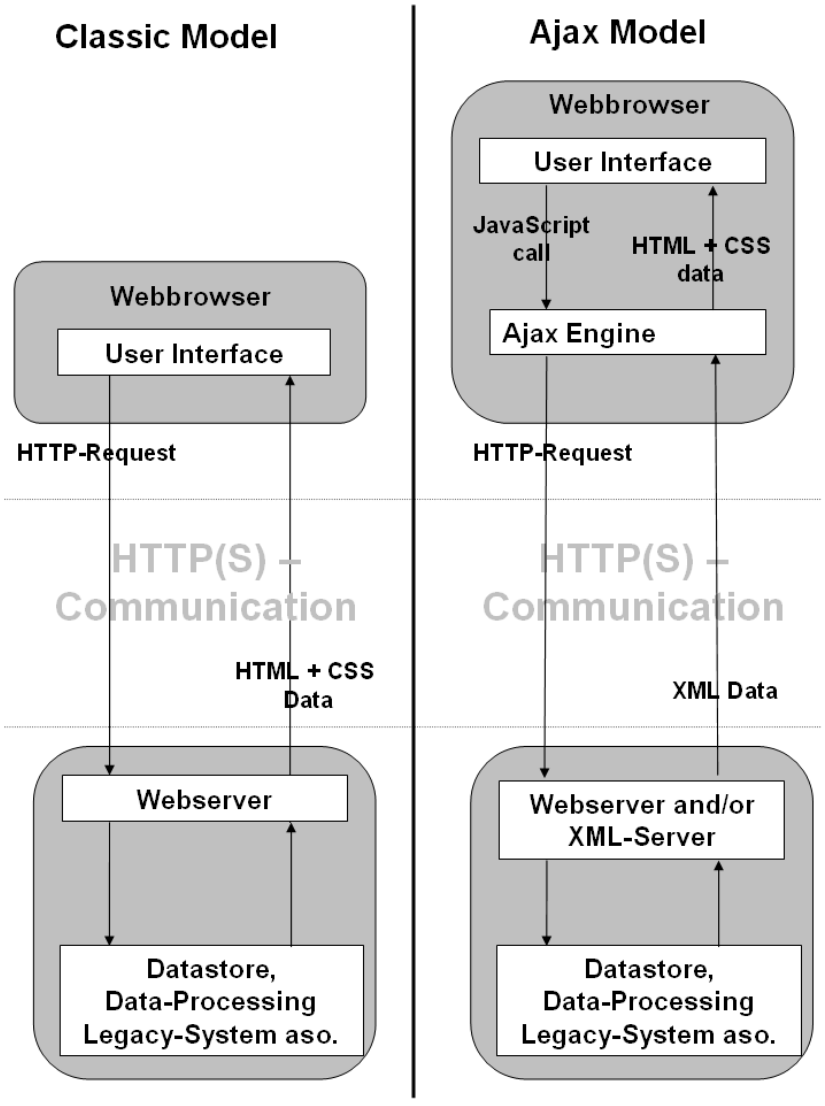

Figure 1: Basics of the ajax programming

Das Problem ist, dass diese JavaScript-Codefragmente sich immer sehr stark auf eine bestimmte Webseite beziehen. Mit dem Java-Webparts Projekt gibt es eine Möglichkeit sich entsprechende JavaScript-Bereiche auf seiner Webseite generieren zu lassen. Es ist also möglich Ajax zu verwenden ohne eine

einzige Zeile Code schreiben zu müssen.
Mit Java-Webparts (`http://javawebparts.sourceforge.net/`)wird eine Tag-Bibliothek ausgeliefert, die **A**jax**P**arts **T**aglib (APT), mit dessen Hilfe eine Menge Ajax-Funktionalitäten realisiert werden können. Die Bibliothek ist unabhängig von einem bestimmten Webframework, ich habe die Bibliothek im Zusammenhang mit Struts (1.3) getestet. Das Gute an dieser Bibliothek ist, dass sie überall eingesetzt werden kann, wo JSP Seiten genutzt werden. Es können auch nachträglich nur bestimmte Teile der Webseite "'ajaxifiziert"' werden. Ich habe mich auf folgende Features konzentriert:

- Einfügen von beliebigen HTML-Fragmenten in eine JSP-Seite. In diesem Fall werden über den `Writer` des Responses die gewünschten Teile ausgegeben. Diese Methode erinnert ein wenig an die "'native Servlet-Programmierung"'. Mit dieser Methode können aber schon die gängigsten Ajax-Effekte implementiert werden wie z.B.
  - Füllen von Comboboxen in Abhängigkeit von anderen Comboboxen.
  - Anzeigen von möglichen Alternativen bei Buchstabeneingabe in einer Textbox.
  - Hinzufügen oder entfernen von Teilbereichen einer JSP-Seite, z.B. von Hilfetexten.
- Einfügen von zeitgesteuerten Events, z.B. zum Einfügen einer Uhr in die JSP-Seite oder zum "'pollen"' auf einen bestimmten Status.
- Manipulation von Inhalten **und** Eigenschaften von Form-Elementen wie z.B. Vorbelegung von Texten in Eingabezeilen, disablen von Buttons, Farbveränderungen etc.

Dieses Dokument dient als Einführung in die Ajax-Programmierung mit APT.

2 Die Philosophie von APT

APT arbeiten mit vordefinierten Handlern. Grundsätzlich wird zwischen Request-, Response- und Errorhandlern unterschieden. Die Abfolge ist dabei immer die gleiche:

- In der JSP Seite wird über das `<ajax:event>` Tag definiert, wo das entsprechende Ajax-Feature ausgelöst werden soll. Grundsätzlich sind alle JavaSript Events möglich wie z.B. Änderung in einer Combobox, Klick auf ein Button, Platzieren der Maus über einen bestimmten Bereich.

- Jedem Event wird, konfiguriert in einer XML-Datei, ein bestimmter Request-, Response- und Errorhandler zugewiesen. Dabei können Events in Gruppen komponiert werden.
- Der Request wird dann einem Ziel zugeführt, z.B. einem bestimmten Servlet, oder einer Struts-Action.
- Das Ziel muss dann die Daten in den Response schreiben.
- Die Daten werden dann in einem vorher definierten Block zurückgeschrieben (z.B. in einen `<div>` Block). Für die Manipulation von Formulardaten und -eigenschaften müssen die Daten wiederum in einem bestimmten XML-Format zur Verfügung gestellt werden.

Der Entwicklungszyklus ist also grob wie folgt:

1. Entscheide, wie das Ajax-Event ausgelöst werden soll und markiere diese Stelle mit dem `<ajax:event>` Tag.
2. Entscheide, wo der Response ausgegeben werden soll und markiere diese Stelle z.B. mit einem `<div>` - Block.
3. Definiere die Form des Requests und Responses in der zentralen XML-Konfigurationsdatei.
4. Implementiere die Logik, die den Request entgegennimmt und den Response erzeugt.

3 APT Konfiguration

Um APT einzusetzen sind einige Konfigurationen durchzuführen.

1. Für Ajax werden zwei Bibliotheken benötigt:

   `javawebparts-ajaxparts-xxxx.jar` und

   `javawebparts-core-xxxx.jar`,

   wobei xxxx die Version der Bibliothek entspricht. Aktuell ist im Moment die Version 1.0Beta6. Die Bibliotheken werden mit der Webanwendung im `WEB-INF/lib` Verzeichnis zur Verfügung gestellt.

2. In der `web.xml` wird konfiguriert, wo die XML-Konfigurationsdatei für die verschiedenen Handler zu finden ist.

```
<context-param>
   <param-name>AjaxPartsTaglibConfig</param-name>
   <param-value>/WEB-INF/ajax-config.xml </param-value>
</context-param>
```

3. In der JSP-Seite wird die Tag-Bibliothek referenziert. Die Bibliothek selber wird im `jar` mitgeliefert, sodass sich hier keine weiteren Gedanken gemacht werden muss.

   ```
   <%@ taglib prefix="ajax" uri="javawebparts/ajaxparts/taglib" %>
   ```

4. Für jedes Element, welches mit Ajax verbunden werden soll, muss ein Eventhandler definiert werden, der direkt dem Form-Element folgen muss. Direkt heißt, dass kein Zeichen (auch keine Blanks etc.) vor dem Eventhandler eingefügt werden dürfen.

   ```
   <ajax:event ajaxRef="xxxx/yyyy"/>
   ```

   xxxx entspricht einer Gruppe und yyyy einem Element, welches in der Ajax Konfigurationsdatei konfiguriert werden muss.

5. Nachdem alle Events definiert wurden ist ein abschließendes

   ```
   <ajax:enable/>
   ```

   nötig. Die beste Stelle hierfür ist kurz vor dem schließenden `</body>` - Tag.

4 Die APT Request-Handler

Ein Requesthandler definiert, auf welche Art und Weise der Request übertragen werden soll. Folgende Handler werden unterschieden:

- **QueryString:** Erzeugt einen Anfragestring und überträgt ihn zum Ziel. Dabei ist es möglich Schlüssel / Wertepaare zu übertragen, wobei die Werte Form-Felder Werte sein müssen.

  Beispiel: Übertragen werden soll der Inhalt einer Textbox mit dem Bezeichner `name`

  ```
  <requestHandler type="std:QueryString"
                  target="/ajax.do">
     <parameter>value=name</parameter>
  </requestHandler>
  ```

  In der korrespondierenden Java-Klasse kann dann der Parameter als Requestparameter ausgelesen werden:

  ```
  String value = (String) request.getParameter("value");
  ```

- **SendById:** Mit diesem Request werden explizite DOM-IDs an den Server übertragen. Es können sowohl Werte von Formelementen als auch Body-Elemente übertragen werden. Außerdem kann kann die Übertragung auch per XML erfolgen. In diesem Fall muss das gewünschte Root-Element mit spezifiziert werden.

Beispiel: Übertragen werden soll der Inhalt einer Textbox mit dem Bezeichner `firstName` und der Inhalt eines `<div>` - Blocks namens `lastName`. Das XML-Rootelement soll Daten heißen:

```
<requestHandler type="std:SendByID"
                target="/sendByIDTest.do">
 <parameter>xml.person,
            lastName.innerHTML,
            firstName.value</parameter>
</requestHandler>
```

- **SimpleRequest:** Dieser Requesthandler erzeugt einen Request an den Server ohne dabei Daten zu übertragen.

```
<requestHandler type="std:SimpleRequest"
                target="/textReturner.do">
   <parameter />
</requestHandler>
```

- **SimpleXML:** Erzeugt ein einfaches XML-Dokument und überträgt die Werte im Body des POST requests. Als Parameter werden die Werte als Schlüssel / Wertepaare übertragen, wobei der erste Wert das Root-Tag des XMLs beschreibt.
- **Poster:** Poster arbeitet wie SimpleXML, nur dass die Werte nicht im XML-Format übertragen werden sondern als String.

5 Der APT Responsehandler

Responsehandler definieren, auf welche Art und Weise der Response aufgearbeitet werden soll. Folgende Responsehandler sind möglich:

- **Alerter:** Der Response-String wird in einer Messagebox ausgegeben.
- **CodeExecuter:** Diese Handler nimmt an, dass der Response ein JavaScript Snipplet ist. Dieses wird ausgeführt. Dabei muss der Code nicht in `<script>` Tags eingebettet sein.
- **IFrameDisplay:** Der Response wird so wie er ist in einen IFrame eingebettet, der bereits im Dokument vorhanden sein muss. Als Parameter wird der IFrame Name angegeben. Falls in mehrern IFrames das gleiche Ergebnis eingetragen werden soll, so können diese kommasepariert angegeben werden.

- **InnerHTML:** InnerHTML arbeitet exact wie IFrameDisplay, nur das in diesem Fall eine ID von einem `<div>` Block angegeben werden muss.

  Beispiel: Die Response soll in einem `<div>`-Block mit der ID `result` eingesetzt werden:

```
<responseHandler type="std:InnerHTML">
   <parameter>result</parameter>
</responseHandler>
```

- **DoNothing:** Dieser Handler tut nichts. Er wird gebraucht, wenn die Response irrelevant ist, im XML aber ein Responsehandler als Mandantory erwartet wird.
- **Redirector:** Dieser Handler geht davon aus, dass die Response eine URL ist. Der Handler führt diese URL auf dem Client aus. Mit diesem Handler ist es also möglich dynamisch auf verschiedene URLs zu verzweigen.
- **SelectBox:** Dieser Handler kann das Ergebnis in einer SelectBox darstellen. **In meiner Umgebung funktionierte der Handler nicht!**. Er kann genauso gut mit InnerHTML dargestellt werden.
- **TextboxArea:** Dieser Handler kann das Ergebnis in einer Textbox darstellen. **In meiner Umgebung funktionierte der Handler nicht!** Er kann genauso gut mit InnerHTML dargestellt werden.
- **WindowOpener:** Dieser Handler öffnet ein neues Fenster und stellt hier die Response dar. Als Parameter werden zwei Teile durch eine Tilde komponiert. Der vordere Teil ist der Titel des neuen Fenster, im hinteren Teil können Fensterparameter angegeben werden. Die möglichen Parameter können durch Google gefunden werden (`window.open()`)

```
<responseHandler type="std:WindowOpener">
<parameter>MyWindow~height=300,width=150</parameter>
</responseHandler>
```

- **XSLT:** Dieser Handler ist dafür gedacht XML zurück zu geben um es dann mit Hilfe von XSLT nach HTML zu transformieren. Dieser Handler benötigt die Sarissa Bibliothek (`http://sarissa.sourceforget.net`)
- **FormManipulator:** Mit Hilfe dieses Handlers können Form-Element sowohl vom Inhalt als auch von den Eigenschaften verändert werden. Zurückgeben werden muss ein XML, welches wie folgt aufgebaut sein muss:

```
<form name="AA">
  <formproperty name="BB" value="CC"/>
    <element name="DD" value="EE">
    <property name="FF" value="GG"/>
    </element>
 </form>
```

Wobei *AA* der Name der Form ist, *BB* der Name einer Formeigenschaft (z.B. action), *CC* der neue Wert ist, *DD* der Name eines Form-Elements, *EE* der neue Wert, *FF* eine Eigenschaft eines Elements ist und *GG* wiederum den neuen Wert darstellt. Diese Funktion ist sehr mächtig, weil alle Eigenschaften von Form-Elementen zur Laufzeit geändert werden können.

```
<responseHandler type="std:FormManipulator">
</responseHandler>
```

6 Die Ajax XML Konfiguration

Die genaue Beschreibung der Konfiguration ist der Original Dokumentation zu übernehmen. Grundsätzlich können Ajax Funktionen gruppiert werden und werden dann mit Request-, Response- und ggf. mit einem Errorhandler konfiguriert werden.
[1]

```
<ajaxConfig>
  <handler name="" type="">
    <function></function>
    <location></location>
  </handler>
  <group ajaxRef="" async="" method="" form="" preProc="" postProc="">
    <errroHandler code="" type="" />
    <element ajaxRef="" async="" method="" form="" preProc="" postProc="">
      <errroHandler code="" type="" />
       <event type="" async="" method="" form="" preProc="" postProc="">
        <errroHandler code="" type="" />
         <requestHandler type="" target="">
           <parameter></parameter>
        </requestHandler>
        <responseHandler type="" matchPattern="">
          <parameter> </parameter>
```

[1] Standardmäßig wird nur ein Errorhandler, der AlertErrorhandler zur Verfügung gestellt. Dieser Handler gibt Fehler in einer Alertbox aus. Außerdem können eigene Request-, Response- oder Errorhandler eingefügt werden. Dieses würde aber dieses Dokument sprengen

```
        </responseHandler>
      </event>
    </element>
  </group>
</ajaxConfig>
```

- **handler:** Über das Handler-Tag ist es möglich eigene Request-, Response- oder Errorhandler zu definieren. Dieses ist nicht im Fokus dieses Dokuments.
- **group:** Über dieses Tag können mehrere Ajax-Events gruppiert werden, typischerweise sind die Events, die in einer JSP-Seite zur Verfügung gestellt werden. Die einzelnen Attribute sind selbsterklärend, wobei nur das ajaxRef Element ein "'required" Feld ist.
- **element:** Die Ajax-Referenz des Elements, welches das Ajax-Event auslöst, z.B. eine Eingabezeile, eine Combobox oder ein Button.
- **event:** Der Typ des Events. Dabei sind alle JavaScript Events möglich: onblur, onfocus, onchange, onclick, oncontextmenu, onkeydown, onkeypress, onkeyup, onmousemove, onmouseout, onmouseover, onmouseup, onmousedown, onresize, onscroll, onselect.
- **requestHandler:** Der Requesthandler, der für dieses Event zuständig ist.
- **responseHandler:** Der Responsehandler, der für dieses Event zuständig ist.

7 Beispiel Step by Step

Zunächst wird über ein `ajax:event` Tag die Combobox markiert, die den Event auslöst:

```
<form name="SelectForm">
Suche Dir eine Science Fiction Sendung heraus und ich zeige
Dir die Darsteller:<br/>
<select name="showTitleSelect">
<option value="B5">Babylon 5</option>
  <option value="BSG">Battlestar Galactica</option>
  <option value="STTNG">Star Trek The Next Generation</option>
  <option value="STTOS">Star Trek The Original Series</option>
  <option value="SGA">Stargate Atlantis</option>
  <option value="SG1">Stargate SG-1</option>
  </select><ajax:event ajaxRef="SelectForm/showTitleChange"/>
</form>
```

Der Zweite Schritt ist die XML-Konfiguration für dieses Evernt in der Ajax Konfigurationsdatei:

```
<group ajaxRef="SelectForm">
  <element ajaxRef="showTitleChange">
    <event type="onchange">
       <requestHandler type="std:QueryString" method="get">
          <target>/ajax.do</target>
          <parameter>showTitle=showTitleSelect</parameter>
       </requestHandler>
       <responseHandler type="std:InnerHTML">
          <parameter>characters</parameter>
       </responseHandler>
   </event>
  </element>
</group>
```

Bei jedem onchange Event der Listbox wird ein Ajax-Event generiert. Der std:QueryString Requesthandler sorgt dafür, das der Wert des eingestellt Listboxeintrages an den Server addressiert wird. In diesem Fall wird die Struts Action /ajax.do angesprungen.
Über ein div-Element wird jetzt die Stelle markiert, in der die Response gerendert werden soll, die ID heißt characters.

```
<br/>
 Hier ist die Liste der Darsteller:
 <div id="characters">
   <select>
   <option> </option>
   </select>
 </div>
```

Der gesamte Body innerhalb des div-Blocks wird durch die Response ersetzt. Zum Schluss muss nur noch die Funktionalität ausimplementiert werden, anbei die Funktion, welche die Werte in die Response schreibt

```
/**
 * Füllt eine Combobox je nachdem, wie die erste gefüllt ist.
 * @param response HTTPServlet Response
 * @param showTitle Wert der ersten Combobox
 * @throws IOException
 */
private void fillComboBox(HttpServletResponse response,
                          String showTitle)
   throws IOException {

   PrintWriter out = response.getWriter();
   out.println("<select>\n");
   String[] characters = null;

   if (showTitle.equalsIgnoreCase("B5")) {
       characters = StaticStrings.Babylon5;
   }
   //... und so weiter
   if (showTitle.equalsIgnoreCase("SG1")) {
```

```
      characters = StaticStrings.StargateSG1;
   }

   if (characters != null) {
      for (int i = 0; i < characters.length; i++) {
          out.println("  <option>" + characters[i] + "</option>\n");
      }
   }
   out.println("</select>\n");
}
```

8 Taglib Referenz

Die Tag-Bibliothek besteht aus nur vier Tags. Hier die vollständig Referenz der Taglib.

8.1 ajax:event

Dieses Tag markiert ein Ajax-Auslöser. Dieses Tag ist unmittelbar nach dem Auslöser zu platzieren. Nehmen wir einmal an, wir wollen nach Änderung einer Combobox weitere Boxen füllen. Das Event-Tag muss dann wie folgt platziert werden:

```
<select name="showTitleSelect">
  <option value="B5">Babylon 5</option>
  ...
  <option value="SG1">Stargate SG-1</option>
</select><ajax:event ajaxRef="ComboForm/showTitleChange"/>
```

Folgende parameter sind möglich:

- **ajaxRef [required]:** Die ID besteht aus zwei Teilen: Der erste Teil ist die ID der Gruppe, zudem dieses Element gehört, der zweite Teil entspricht der Element ID.

8.2 ajax:enable

Dieses Tag muss nach dem letzten `ajax:event` Tag eingeführt werden. Mit diesem Tag wird Ajax auf dieser Seite möglich gemacht.
Folgende parameter sind möglich:

- **debug [optional, default: error]:** Hiermit wird der debug-level festgelegt. Debugausgaben werden standardmäßig über ein Alert-Fenster angezeigt (siehe nächsten Parameter). Folgende Werte sind möglich: `trace`, `debug`, `info`, `error` und `fatal`.

- **logger [optional, default: JWPAlertLogger]:** Definiert den Logger für das Ajax-Logging. Zurzeit sind zwei Logger implementiert: Der JWPAlertLogger, der die Ausgaben über die JavaScript Funktion `alert()` ausgibt und den JWPWindowLogger, der die Ausgaben in ein neues Fenster schreibt.

- **suppress [optional, default: false]:** Dieser Schalter gibt an, wie der Logger und das JavaSript dargestellt wird. Wenn der Response des Ajax-Elemtents aus einer JSP kommt, dann muss dieser Schalter auf true gesetzt werden um zu verhindern, dass JavaScript Funktionen doppelt auf dem Client dargestellt werden.

8.3 ajax:timer

Mit diesem Tag kann ein Timer aufgebaut werden mit dem periodisch ein Teil der Seite aufgefrischt werden kann, z.B. zum Einblenden einer Uhr. Das Tag muss in einer Form eingebettet sind.
Mit diesem Tag werden zwei JavaScript Funktinen erstellt die `startXXXX()` und `stopXXXX()` heißen, wobei `XXXX` die `ajaxRef` ist, wobei der slash mit einem underscore ersetzt wird. Wenn z.B. die `ajaxRef` "'MyPage/MyButton"' ist, dann würden die Funktionen

`startMyPage_MyButton()` und
`stopMyPage_MyButton()`

heißen. Somit ist es möglich einen Timer zu starten und zu stoppen.
Folgende parameter sind möglich:

- **ajaxRef [required]:** Die ID besteht aus zwei Teilen: Der erste Teil ist die ID der Gruppe, zudem dieses Element gehört, der zweite Teil entspricht der Element ID.

- **frequency [required]:** Beschreibt das Intervall des Timers in Millisekunden.

- **startOnLoad [optional, default: false]:** Dieser Parameter entscheidet, ob der Timer sofort nach dem Laden der Seite gestartet wird (`true`) oder mit einem Aufruf der entsprechenden JavaScript-Funktion manual gestartet werden soll (`false`).

8.4 ajax:manual

Dieses Tag kann genutzt werden, wenn AJAX JavaScript Code generiert werden soll, es aber nicht mit einem bestimmten Event verknüpft werden soll. Die erstellen JavaScript Funktionen können dann mit normalen Aufruf angesprochen werden.
Folgende parameter sind möglich:

- **ajaxRef [required]:** Die ID besteht aus zwei Teilen: Der erste Teil ist die ID der Gruppe, zudem dieses Element gehört, der zweite Teil entspricht der Element ID.
- **function [required]:** Der Name der JavaScript Funktion, die generiert werden soll. Diese Funktion kann dann von jeder Stelle aus aufgerufen werden.