

jMaki - Sun's Ajax-JavaScript-Wrapper-Framework

Eine einheitliche Zugriffsschicht für Ajax-JavaScript-Frameworks

Von Andy Bosch

Dojo, Scriptaculus, Yahoo UI Widgets oder DHTML Goodies: Für einen Webentwickler mit Schwerpunkt auf der UI-Entwicklung wird es zunehmend schwerer, den Überblick zu bewahren. Seitdem der Hype um Ajax begonnen hat, sprießen die Ajax- und JavaScript-Frameworks wie Pilze aus dem Boden. Der Aufbau einer eigenen Anwendung mit Hilfe dieser Frameworks stellt dagegen insofern eine Herausforderung dar, dass unter Umständen unterschiedliche Frameworks integriert werden müssen, um die angebotenen Funktionalitäten der verschiedenen Bibliotheken ausnutzen zu können. An dieser Stelle setzt jMaki an. Mit jMaki stellt Sun eine Art „Wrapper-Framework“ zur Verfügung.

Warum überhaupt ein Wrapper-Framework?

Primär soll durch jMaki die Unterstützung von Ajax in JSP- oder JSF-basierten Anwendungen vereinfacht werden. Wenn man in einem Projekt die Entscheidung getroffen hat, die Anwendung mit Hilfe von Ajax anzureichern und damit das UI auch bedienerfreundlicher zu gestalten, stellt sich daran gleich die Frage, mit welchen Frameworks und Bibliotheken dies geschehen soll: Bei Einsatz von JavaServer Faces kann man entweder auf den Ajax-Support von MyFaces bauen, oder man integriert z.B. das OpenSource-Projekt ajax4jsf. Basiert die Anwendung auf „normaler“ JSP-Technologie, können JavaScript-Frameworks wie Dojo oder DWR eingesetzt werden, um nur mal die bekanntesten zu nennen. Tolle Effekte lassen sich auch mit Scriptaculus erzielen. Möchte man jedoch Funktionalitäten oder Widgets der unterschiedlichsten Frameworks integrieren, kann man schon mal die ein oder andere schlaflose Nacht haben, da sie untereinander nicht immer kompatibel und kombinierbar sind.

Genau an diesem Punkt setzt jMaki an. Man könnte jMaki durchaus auch als Integrationsplattform bezeichnen (ok, dieser Begriff ist vielleicht ein wenig „oversized“, drückt jedoch gut aus, wozu jMaki dient). Mit jMaki soll eine einfache Möglichkeit für Entwickler geschaffen werden, um unterschiedlichste Ajax-Funktionalitäten zu nutzen.

Hauptfokus: Ajax-Support

Der Hauptfokus von jMaki liegt eindeutig auf einer Vereinfachung der Nutzung von Ajax-Funktionalität. In den nachfolgenden Listings sind dazu auch ein paar Beispiele aufgezeigt. Mit jMaki werden einem Entwickler Tags für JSP oder JSF 1.1 zur Verfügung gestellt, mit denen sehr einfach Widgets in Webseiten integriert werden können. Der Name jMaki baut sich übrigens zusammen aus „j“ von Java und „Maki“, einem japanischen Wort für „einhüllen“ oder „einwickeln“.

jMaki-Widgets

jMaki bietet die Möglichkeit, Komponenten aus den bekannteren Ajax-JavaScript-Frameworks wie Dojo, Scriptaculus, Yahoo UI Widgets oder DHTML Goodies zu verwenden. Des Weiteren stehen einige native jMaki-Widgets zur Verfügung (z.B. ein Chat-Widget oder ein RSS-Reader-Widget). **Doch was ist überhaupt ein jMaki-Widget?** Ein jMaki-Widget besteht per Definition aus folgenden drei Teilen:

- Einer Komponente, die das notwendige Layout liefert (Html-Datei)
- Einer Komponente, die die notwendigen Styles liefert (Css-Datei)

- Einer Komponente, die die notwendige Funktionalität liefert (JavaScript-Datei)

All diese Komponenten werden in einem jMaki-Widget gebündelt und können sehr einfach durch einen Anwendungsentwickler über JSP- oder JSF-Tags angesprochen und verwendet werden. Somit entfällt der oft mühevoll Weg, CSS- und JS-Dateien anzupassen und zu grübeln, warum im Opera oder Safari irgendeine Funktionalität nicht richtig dargestellt und verarbeitet wird. Der Standardaufbau eines jMaki-Tags schaut wie folgt aus:

```
<a:ajax name="my.widget"/>
```

Zum aktuellen Zeitpunkt der Entwicklung wird bei Verwendung eines jMaki-Tags zunächst im Verzeichnis „/my/widget“ nach den Dateien **component.htm**, **component.js** und **component.css** gesucht. Wird dort nichts gefunden, wird im Verzeichnis WEB-ROOT/resources nach den entsprechenden Definitionen gesucht. Diese Verzeichnisstrukturierung lässt sich sehr schön an den Beispielen nachvollziehen, die aktuell mit jMaki mitausgeliefert werden.

Installation

Momentan ist jMaki noch als Early-Access Version über die Website <https://ajax.dev.java.net/> zu beziehen. Mit enthalten ist gleich eine einsatzbereite war-Datei, um erste konkrete Beispiele zu demonstrieren. Wichtig ist der Hinweis, dass jMaki bereits Java 5 voraussetzt. Um ein eigenes Projekt mit jMaki aufzusetzen, müssen die mit jMaki ausgelieferten Bibliotheken sowie die notwendigen JavaScript- und Css-Ressourcen in das eigene Projektverzeichnis übernommen werden. Danach kann es auch gleich mit der Entwicklung losgehen.

Ein erstes Beispiel

Als erstes einfaches Beispiel wird ein Inplace-Editor verwendet. Damit ist es möglich, in einer Html-Seite bei einem einfachen Ausgabefeld per Klick auf ein Eingabefeld umzuschalten, um den Inhalt zu bearbeiten. Das „coole“ daran ist, dass dazu nicht wie bei Webanwendungen üblich, ein erneuter kompletter Page-Request notwendig ist, sondern per JavaScript / Ajax die Umschaltung in den Edit-Modus erfolgt. Für den Anwender ist dies somit ein flüssigeres Arbeiten, wie er es auch von „normalen“ Fat-Client oder Rich-Client Anwendungen gewohnt ist.

Listing 1: Inplace-Editor mit JSP

```
<%@page contentType="text/html"%>
<%@ taglib prefix="a" uri="http://java.sun.com/jmaki"%>

<html>
  <head>
    <title>jMaki Beispielseite</title>
  </head>
  <body>
    <h1>Inplace-Editor</h1>

    <form>
      <a:ajax type="scriptaculous" name="inplace"
        service="inplaceService.jsp" value="test" />
    </form>
  </body>
</html>
```

Ende Listing

Listing 2: Inplace-Editor mit JSF

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib prefix="a" uri="http://java.sun.com/jmaki-jsf" %>

<html>
  <head>
    <title>jMaki Beispielseite</title>
  </head>
```

```
<body>
  <h1>Inplace-Editor mit JSF</h1>

  <f:view>
    <h:form>
      <a:ajax type="scriptaculous" name="inplace"
              service="inplaceService.jsp" value="test" />
    </h:form>
  </f:view>

</body>
</html>
```

Ende Listing

In Listing 1 ist die Verwendung in einem reinen JSP-Kontext demonstriert. Soll jMaki in einer JSF-Umgebung verwendet werden, genügt es, lediglich eine andere Taglib einzubinden (vgl. Listing 2). Wenn man sich die Tagbibliotheken genau anschaut fällt auf, dass sowohl die JSP- als auch die JSF-Tagbibliothek jeweils genau ein Tag haben, nämlich das ajax-Tag mit entsprechenden Parametern. Dieses eine Tag genügt, um sämtliche Widgets ansprechen zu können. Bei Verwendung des Beispiels aus Listing 1 fällt auf, dass keinerlei JavaScript oder Css programmiert bzw. in der Seite eingebunden werden mussten. Und genau hier liegt der Vorteil, die gesamte Funktionalität in Tags gebündelt zu haben: Es wird ausschließlich mit Widgets gearbeitet. Dass dabei erheblich mehr „im Hintergrund“ bereitsteht, ist für den Anwendungsentwickler primär nicht relevant. Nachdem man dieses Beispiel erfolgreich zum Laufen gebracht hat, sollte man sich auf jeden Fall den daraus erzeugten Html-Quellcode anzeigen lassen. Man kann daraus sehr gut erkennen, wieviel Arbeit einem Entwickler durch Verwendung der jMaki-Tags abgenommen wird, da das ganze (lästige) korrekte Einbinden von JavaScript-Dateien, Stylesheets sowie das korrekte Aufrufen der JavaScript-Funktionen etc. nicht mehr erforderlich ist.

Auch noch schick: Ein Fisheye-Widget

In Listing 3 ist ein weiteres Beispiel zu sehen, wie mit Hilfe eines einfachen Tags z.B. eine Fisheye-Darstellung erzielt werden kann. Und ganz am Rande erwähnt: Der Ausschnitt in Listing 3 ist komplett, es wurde nichts weggelassen. Dieses eine Tag genügt, um eine vollwertige Fisheye-Darstellung zu erzielen!

Listing 3: Fisheye-Widget

```
<a:ajax type="dojo" name="dojo.fisheye"
  args="{items:[
    {iconSrc:'logo_choice_1.jpg',caption:'Auswahl-1', index:1},
    {iconSrc:'logo_choice_2.jpg',caption:'Auswahl-2', index:2},
    {iconSrc:'logo_choice_3.jpg',caption:'Auswahl-3', index:3}
  ]}"/>
```

Ende Listing

>>bosch_jmaki_1.jpg<<

Abb. 1: Fisheye-Beispiel

Auch das Fisheye-Beispiel greift auf das Dojo-Framework zurück, wie am Type-Attribut leicht zu erkennen ist. Natürlich können diese Komponenten auch ohne jMaki verwendet werden. Z.B. kann Dojo ohne Probleme in ein bestehendes (JSP)-Projekt eingebunden werden. Mit jMaki soll sich jedoch ein Entwickler darauf verlassen können, dass er eine einheitliche Basis zum Zugriff auf verschiedenste Ajax-Frameworks hat.

Fazit

Dass man sich heutzutage als Web-Entwickler kaum noch dem Ajax-Virus entziehen kann, scheint offenbar zu sein. Fakt ist leider jedoch auch, dass man es als Entwickler immer schwerer hat, alle neuen und „coolen“ Frameworks zu beherrschen, anzuwenden und vor allem auch innerhalb eines Projektes integrieren zu können. Mit jMaki unternimmt Sun jetzt den Versuch, eine

einheitliche Plattform für diverse Ajax-/JavaScript-Frameworks zu etablieren. Der Ansatz selbst scheint logisch und vernünftig zu sein. Entscheidend für den Erfolg wird jedoch sicherlich sein, wie aktuell jMaki an die Weiterentwicklungen der integrierten und dennoch selbständigen Projekte angepasst wird. Vorerst ist es jedenfalls ein Schritt in die richtige Richtung.

Andy Bosch ist selbständiger IT-Berater und Trainer. Er ist Mitglied von SENS (Software Experts Network e.V.). Schwerpunktmässig beschäftigt er sich mit UI-Themen mit Webumfeld, seit einiger Zeit fast ausschließlich im Umfeld von JavaServer Faces und Ajax/Web2.0. Zudem ist er Betreiber der Plattform www.jsf-forum.de

Links & Literatur

- [1] <https://ajax.dev.java.net/>: Projektseite zu jMaki
- [2] <http://www.javaserver.org/jmaki/>: Beispielseite mit jMaki-Widgets
- [3] http://weblogs.java.net/blog/mode/archive/2006/07/ajax_enabling_j.html: Weblog zu jMaki und JSF-Extensions
- [4] <http://forums.java.net/jive/forum.jspa?forumID=96&start=0>: jMaki-Forum
- [5] JSF-Forum: <http://www.jsf-forum.de>