

# Ajax mit DWR und Dojo

Axel Rauschmayer

2006-08-04

## Ziele dieses Vortrags

- Begriff *Ajax* verstehen.
- Grundlegendes Ajax kennenlernen.
- Werkzeuge verstehen, die beim Implementieren helfen
  - der Server-Seite (DWR, Java) und
  - der Browser-Seite (Dojo, JavaScript).

# Was ist Ajax?

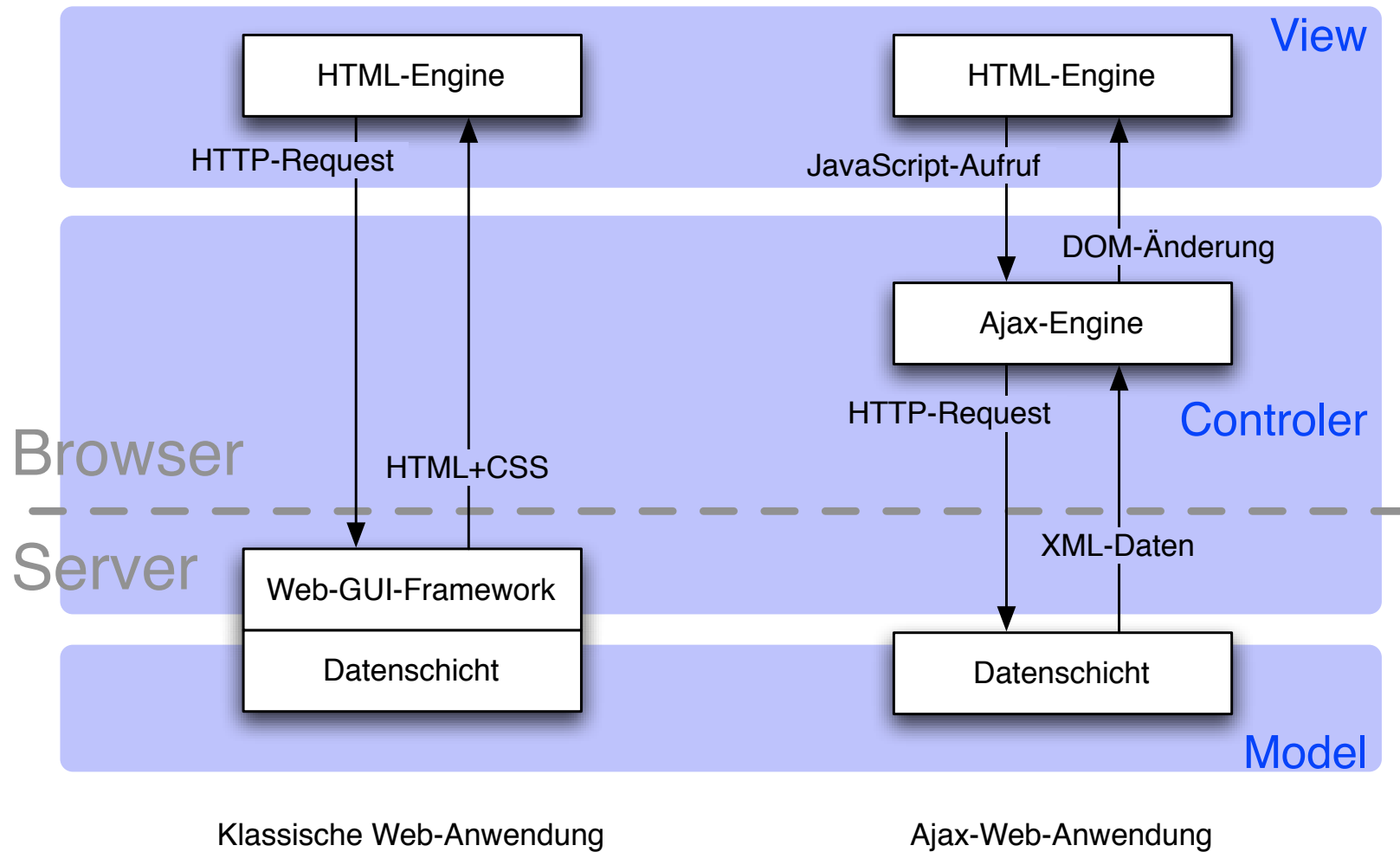
Ajax: Asynchronous JavaScript and XML.

- Idee: Dehne die globale Verfügbarkeit von passiven Daten auf globale Verfügbarkeit von interaktiven Anwendungen aus.
- Konkurrenten: Java-Applets, Flash, Mozilla XUL.  
Sind weitgehend gescheitert, da nicht standardisiert und/oder anwendbar genug.
- Erfolgsgründe: Ajax ist
  - standard-basiert: Die Anwendungen laufen üblicherweise auf dem Internet Explorer, Firefox, Safari, Opera.
  - benutzerfreundlich: Benutzer kennen Web-Browser als Benutzerschnittstelle für globale Daten. Ajax fügt sich sehr sensibel in diese Umgebung ein.

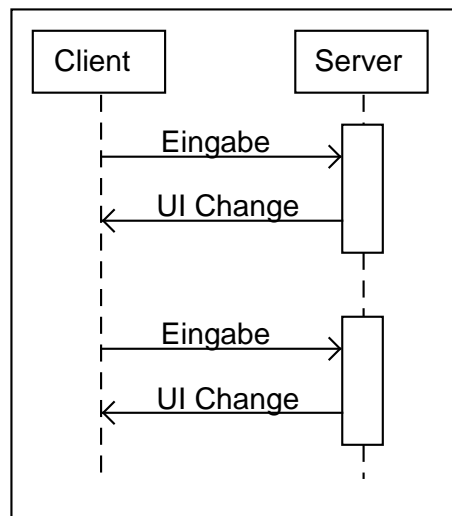
# Die Folgen von Ajax

- Verwaltung der Benutzeroberfläche wird verlagert vom Server auf den Browser.
- Der Server ist blosser Daten-Lieferant. Server-APIs werden funktional, wie Web-Services (wenn sie nicht sogar als Web-Services implementiert werden).
- Desktop- und Web-Anwendungen werden sich immer ähnlicher
  - Desktop: Updates übers Internet.
  - Web: Interaktive Widgets, Drag and Drop etc.

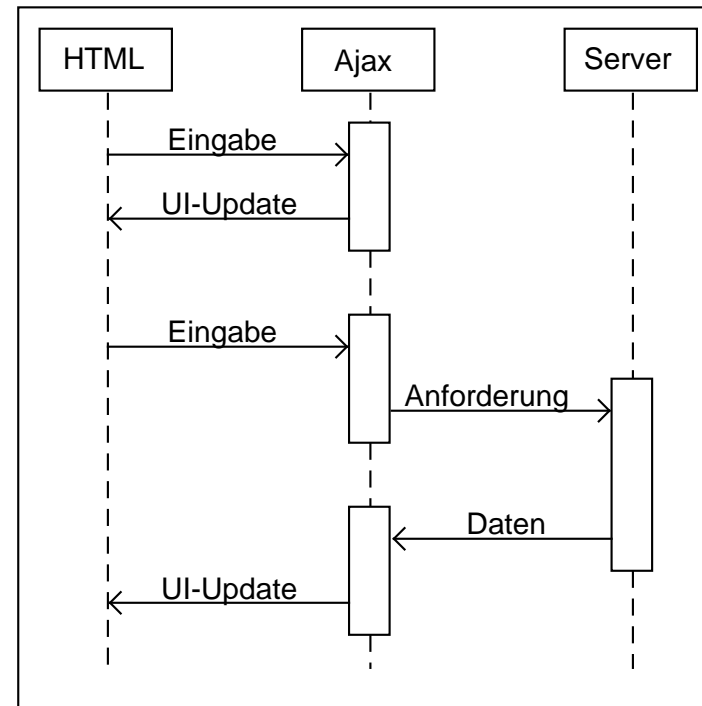
# Architektur



# Interaktion



klassisch (synchron)



Ajax (asynchron)

# Ajax-Basistechnologien

Ajax: Asynchronous JavaScript and XML

- Darstellung der Benutzeroberfläche: (X)HTML
- Programmierung von Verhalten: JavaScript
  - Datenmodell für HTML: DOM (Document Object Model), objekt-orientierte Sicht auf HTML.
  - Asynchroner Austausch von Daten mit Server: XMLHttpRequest-Objekt. Übliche Datenformate: XML oder JSON (siehe nächste Folie).
- Unterstützend: CSS, das auch im DOM repräsentiert wird.

---

## JSON (JavaScript Object Notation)

---

```
{ "menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      { "value": "New", "onclick": "CreateNewDoc()" },
      { "value": "Open", "onclick": "OpenDoc()" },
      { "value": "Close", "onclick": "CloseDoc()" } ] ] }
```

---

---

## XML

---

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

---

# Geschichte

- Sep 1995: Mit Netscape 2 wird *LiveScript* eingeführt, u.a. um Formular-Eingaben zu überprüfen. Später: Umbenennung zu *JavaScript*.
- 1997: DHTML, client-side Scripting, mit der Möglichkeit, HTML zu verändern. In Netscape 4 (Jun 1997) und Explorer 4 (Okt 1997)
- 8. Feb 2005: Google Maps
- 18. Feb 2005: „Ajax“-Artikel von Jesse James Garrett [2].
- 1. Feb 2006: Gründung der „Open Ajax“-Initiative.  
Mitglieder: BEA Systems, Borland, IBM, Dojo Foundation, Eclipse Foundation, Google, Laszlo Systems, Mozilla Foundation, Novell, Openwave Systems, Oracle, Red Hat, Yahoo!, Zend Technologies, Zimbra.  
*Websites, IDEs, Server-Software, Datenbanken, Ajax-Toolkits, Web-Browser.*

# DOM: Document Object Model

`localhost:8080/ajax-demo/dom.jsp`

Grundlegende Mittel:

- Events: um auf Dinge wie Mausklicks zu reagieren.
- Dargestelltes HTML ändern:
  - Quellcode: Wird geparkt und eingesetzt.
  - Struktur: Beliebige Umstellungen selbst vornehmen.
  - CSS: Sichtbarkeit, Farbe etc.
- Markierungen, um Abschnitte im HTML (unformatiert) zu kennzeichnen:
  - `div`: mehrere Absätze.
  - `span`: innerhalb eines Absatzes.

## DOM: HTML per Source ändern

---

```
<a onClick="document.getElementById('a').innerHTML='Hello <b>World</b>';  
    return false;"  
    href="#"  
>Inner HTML</a>:  
<span id="a"></span>
```

---

- Event `click`: DOM-Event, der gefeuert wird, wenn auf ein Element geklickt wird.
- `document`-Objekt: Globaler Einstiegspunkt in die DOM-API.
- Attribut `innerHTML`: Enthält den HTML-Quellcode eines Elementes, kann geschrieben und gelesen werden.
- `return false`: Verhindert, dass zur `href`-Location gegangen wird.

# DOM: HTML per Struktur ändern

---

```
<a onClick="document.getElementById('b').appendChild(  
            document.createTextNode(' Huhu! '));  
            return false;"  
    href="#"  
>Raw DOM</a>:  
<span id="b"></span>
```

---

# DOM: CSS ändern

---

```
<a onClick="document.getElementById('c').style.fontSize='200%';  
        return false;"  
        href="#"  
>CSS</a>:  
<span id="c">Text</span>
```

---

- Attribut `style`: Hat seinerseits alle üblichen CSS-Eigenschaften als Attribute: background, border etc.

# DOM: Aus- und Einblenden per CSS

---

```
<a onClick="document.getElementById('d').style.visibility='hidden' "  
  href="#">Hide</a> |  
<a onClick="document.getElementById('d').style.visibility='visible' "  
  href="#">Show</a>:  
<div id="d">  
  This  
  <ul>  
    <li> is a  
    <li> bullet list.  
  </ul>  
</div>
```

---

# Was ist DWR?

DWR: Direct Web Remoting.

- Einfacher Ajax-Einstieg für Java-Programmierer.
- Versteckt die Komplexität der Client-Server-Kommunikation („RMI zwischen Java und JavaScript“).
- JavaScript-Hilfsfunktionen für schmerzloses, minimales Ajax.

# Vorbereitung für DWR: Tomcat installieren

- Tomcat Core herunterladen.
- Web-Anwendung (mit der für Enterprise-Java typischen Ordnerstruktur) in `$TOMCAT/webapps` verschieben.
  - Unter Unix genügt auch ein symbolischer Link.
  - Die Beispiele für diesen Vortrag sind sowohl ein Eclipse-Projekt, als auch eine standardgemäße Web-Anwendung.
- `cd $TOMCAT`
  - **Start:** `bin/startup.sh`
  - **Stop:** `bin/shutdown.sh`

# 1. Schritt: DWR-Servlet konfigurieren

---

TOMCAT/webapps/ajax-demo/WEB-INF/web.xml

---

```
<servlet>
  <servlet-name>dwr-servlet</servlet-name>
  <display-name>DWR Servlet</display-name>
  <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-class>
  <init-param>
    <param-name>classes</param-name>
    <param-value>
      de.hypergraphs.ajax.TimeServer,
    </param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>dwr-servlet</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

## 2. Schritt: Java-Klasse für den Server schreiben

```
@Create // Ein Service, vgl.: @Convert für zu übertragende Daten
public class TimeServer {
    @RemoteMethod
    public String getData() {
        return "Time: "+new Date();
    }
}
```

Überprüfen, ob DWR funktioniert: [localhost:8080/ajax-demo/dwr/](http://localhost:8080/ajax-demo/dwr/)

## 3. Schritt: JavaScript für den Browser schreiben

localhost:8080/ajax-demo/string.jsp

---

Header

---

```
<script type='text/javascript'>
function updateData() {
    // getData(): Keine Argumente, Rückgabewert per Continuation
    TimeServer.getData(function(data) {
        DWRUtil.setValue("reply", data);
    });
}
</script>
```

---

Body

---

```
Server Information:
<span id="reply">NOT YET LOADED</span>
<input value="Update" type="button" onclick="updateData()" />
```

---

# Komplexeres DWR-Beispiel: Table

`localhost:8080/ajax-demo/table.jsp`

## Table-Beispiel: Zu übertragende Daten

```
@Convert // Neu: Kein Service, sondern konvertierte Daten
public class Student {
    private String id;
    private String name;

    /** Empty constructor needed for DWR */
    public Student() {
    }
}
```

....

[Getters, Setters etc.]

## Table-Beispiel: Services

```
@Create(scope=ScriptScope.APPLICATION)
public class StudentDatabase {
    private List<Student> students = new ArrayList<Student>();
    {
        students.add(new Student("333", "Max"));
        students.add(new Student("5004", "Moritz"));
    }
    @RemoteMethod
    public List<Student> getStudents() {
        return this.students;
    }
    @RemoteMethod
    public void addStudent(Student student) {
        removeStudent(student.getId()); // avoid duplicate IDs
        this.students.add(student);
    }
    [...]
}
```

## Table-Beispiel: Daten anfordern

```
<table border="1">
<thead>
  <tr><th>ID</th><th>Name</th><th></th></tr>
</thead>
<tbody id="students">
</tbody>
</table>
```

```
function refreshTable() {
  StudentDatabase.getStudents(function(students) { // asynchrones Ergebnis
    DWRUtil.removeAllRows("students");
    DWRUtil.addRows("students", students, [
      function(student) { return student.id; },
      function(student) { return student.name; },
      function(student) { return '...'; } // UI-HTML
    ]));
}
```

## Table-Beispiel: Server-Daten ändern

```
<table>
<tr><td>ID:</td><td><input id="id" type="text"></td></tr>
<tr><td>Name:</td><td><input id="name" type="text"></td></tr>
<tr><td colspan="2" align="right">
    <input value="Add" onClick="addStudent()" type="button">
</td></tr>
</table>
```

```
function addStudent() {
    var student = { name: "", id: 0 };
    DWRUtil.getValues(student);
    StudentDatabase.addStudent(student, function() { // arbeite asynchron weiter
        refreshTable();
    });
}
```

# Was ist Dojo?

[localhost:8080/ajax-demo/dojo/demos/demoEngine.html](http://localhost:8080/ajax-demo/dojo/demos/demoEngine.html)

- Toolkit für JavaScript.
- Behebt einige Defizite von JavaScript (keine Module, schwaches Event-System, . . . ).
- Kapselt Browser-Unterschiede.
- Viele mächtige Widgets. Werden wahlweise definiert:
  - per HTML, deklarativ.
  - per JavaScript, prozedural (z.B.: füllen des Widget-Inhalts mit Daten vom Server).

# Dojo: Tabs

`localhost:8080/ajax-demo/tabs.jsp`

# Dojo: Tabs

Head

---

```
<script type="text/javascript" src="/ajax-demo/dojo/dojo.js"></script>
<script type="text/javascript">
  dojo.require("dojo.widget.TabContainer");
  dojo.require("dojo.widget.ContentPane");
  dojo.require("dojo.widget.LinkPane");
</script>
```

---

Body

---

```
<div id="mainTabContainer" dojoType="TabContainer" selectedTab="tab1">
  <div id="tab1" dojoType="ContentPane" label="Tab 1">
    <h1>Tab Eins</h1>
    Inhalt ist in Seite eingebettet. Tab 2 verweist auf externe Datei.
  </div>
  <a dojoType="LinkPane" href="index.jsp">Tab 2</a>
</div>
```

---

# Dojo: Tree

`localhost:8080/ajax-demo/tree.jsp`

# Dojo: Tree

---

Head

```
<script type="text/javascript" src="/ajax-demo/dojo/dojo.js"></script>
<script type="text/javascript">
  dojo.require("dojo.widget.Tree");
  dojo.require("dojo.widget.ContentPane");
</script>
```

---

Body

```
<div dojoType="Tree">
  <div dojoType="TreeNode" title="Item 1">
    <div dojoType="TreeNode" title="Item 1.1"></div>
    <div dojoType="TreeNode" title="Item 1.2">
      <div dojoType="TreeNode" title="Item 1.2.1"></div>
      <div dojoType="TreeNode" title="Item 1.2.2"></div>
    </div>
  </div></div>
```

---

## Weitere Möglichkeiten

- Client-side Data Storage: Über eine Brücke zu Flash (das bei fast allen Browsern mit dabei ist, selbst älteren) kann man per JavaScript bis zu mehreren Megabyte Daten im Browser ablegen und somit autarke Anwendungen schreiben.
- Vektorgrafik [4]: Da neuere Browser zunehmend SVG unterstützen, bekommt Ajax hier die Mittel für echte, auch animierte Vektorgrafik.

# Technische Herausforderungen

- Bookmark-ability, History: URLs können (aus Sicherheitsgründen) nicht beliebig verändert werden, sollen aber dennoch jederzeit den aktuellen Zustand der Anwendung widerspiegeln.  
Teillösung: Der Fragment-Teil einer URL ist änderbar.
- Accessibility: Lesesoftware für Blinde muss gezielt unterstützt werden.
- Durchsuchbarkeit für Suchmaschinen. Hier kann man selten das Anlegen separater, statischer Seiten vermeiden.

# Die Zukunft

- Comet/Reverse Ajax [5]: Der Server benachrichtigt den Client. Verbindungen bleiben länger offen, was für geringere Latenzzeiten sorgt.
- JavaScript-Erweiterungen: Optionale Typen, Module etc. [6]
- Bessere Java-Tools: Z.T. schon vorhanden bei IntelliJ, Aptana, MyEclipse, NetBeans. Bisher aber nicht auf dem Stand von Java.

## Zum Mitnehmen

- Ajax-Anwendungen sind traditionellen Desktop-Anwendungen sehr ähnlich.
- Die Benutzeroberfläche läuft im Browser und nimmt gelegentlich Verbindung zum Server auf.
- Die Server-Seite wird angenehm schlank und aufgeräumt.
- Mit den richtigen Werkzeugen bleibt der Programmieraufwand verhältnismäßig gering.

## Referenzen

- [1] *Ajax Notes*, Axel Rauschmayer. Ressourcen-Sammlung: Tutorials, Tipps und Tools. <http://www.pst.ifi.lmu.de/~rauschma/cgi-bin/wiki/wiki.cgi?AjaxNotes>
- [2] *Ajax: A New Approach to Web Applications*, Jesse James Garrett. <http://www.adaptivepath.com/publications/essays/archives/000385.php>
- [3] *Surveying open-source AJAX toolkits*, Peter Wayner. [http://www.infoworld.com/article/06/07/31/31FEajax\\_1.html](http://www.infoworld.com/article/06/07/31/31FEajax_1.html)
- [4] *No Flash Required: Interactive Browser Graphics*, Gavin Doughtie. [http://www.xdraw.org/oscon\\_slides/](http://www.xdraw.org/oscon_slides/)

- [5] *Comet: Low Latency Data for the Browser*, Alex Russell <http://alex.dojotoolkit.org/?p=545>
- [6] *JavaScript 2 and the Future of the Web*, Brendan Eich. <http://developer.mozilla.org/presentations/xtech2006/javascript/>