Ajax – hype oder hilfreich?

Jörn Clausen
joern@TechFak.Uni-Bielefeld.DE

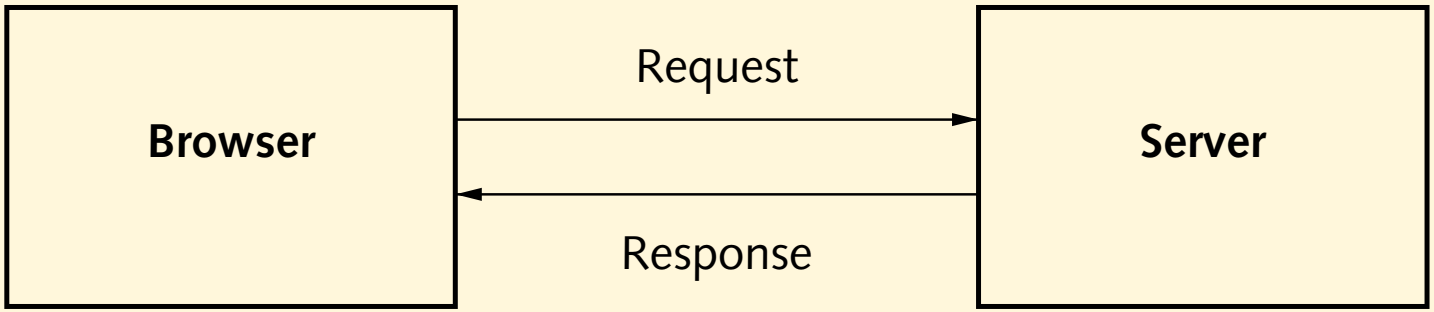
Übersicht

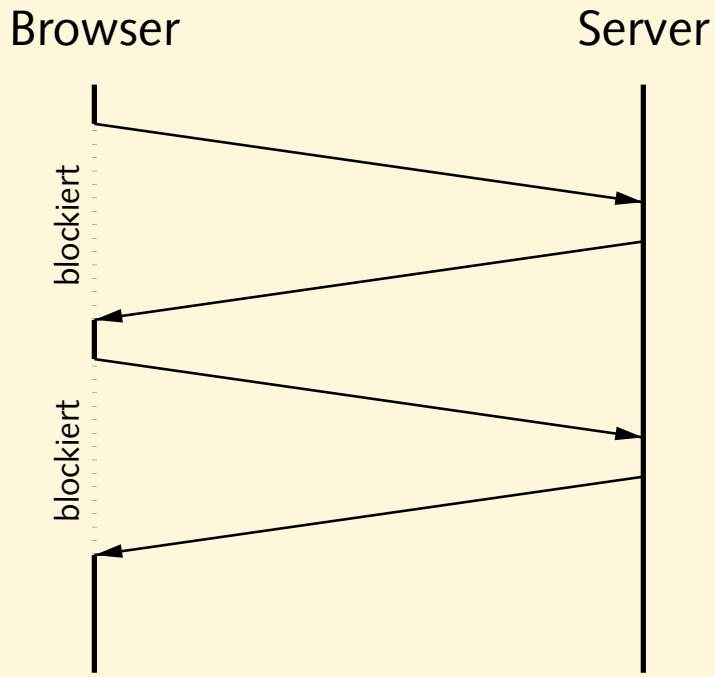
- das World Wide Web im Wandel der Zeit
- XMLHttpRequest
- typische Probleme und Lösungen
- Vor- und Nachteile
- Anwendung: Mashups mit Google Maps

This is not the Web of your Youth

- Standards: XHTML + CSS (+ ECMAScript)
- Webservices, APIs
- Blogs, RSS, syndication
- Wikis, social networks, folksonomies
- RIA (Rich Internet Applications), Mashups
- Web 2.0
- Beispiele: Wikipedia, Google Mail/Maps, Flickr, del.icio.us, . . .

Web 1.0

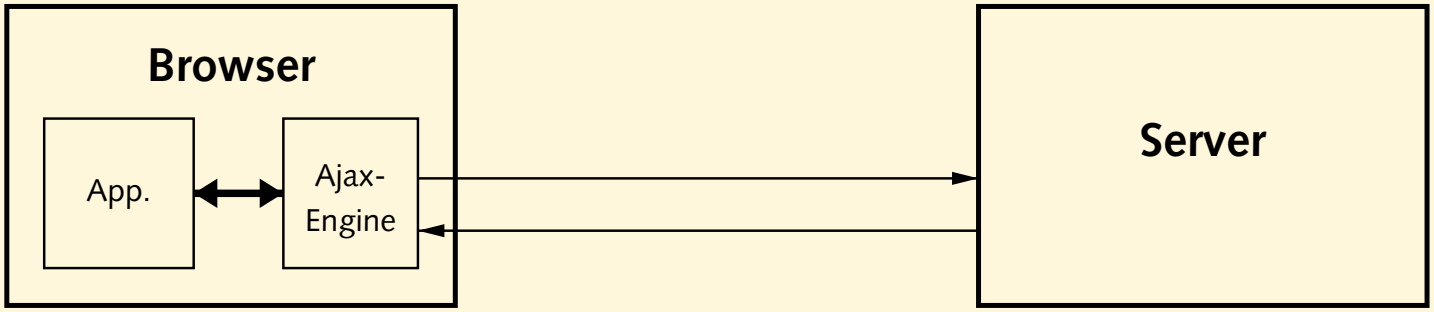

Browser
Request
Response
Server

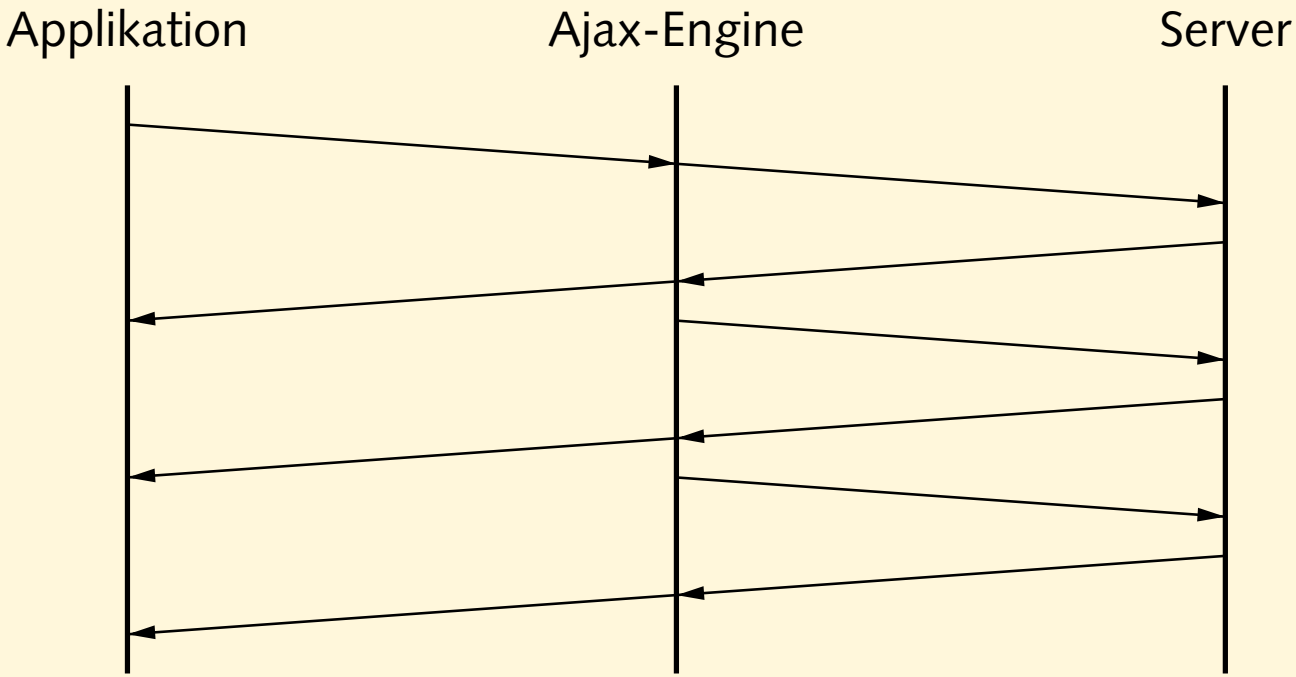

Browser
Server
blockiert
blockiert

Interaktive Web-Anwendungen

- JavaScript + Frames (1995, Netscape 2.0)
- Dynamic HTML + DOM (1996, Internet Explorer 4.0)
- Iframes (1997, HTML 4.0)
- XMLHTTP (2001, Internet Explorer 5.0)
- XMLHttpRequest (2002, Mozilla 1.0)
- Ajax: A New Approach to Web Applications (2005, Jesse James Garrett)

Asynchronous JavaScript and XML





XMLHttpRequest

- verschiedene Implementierungen von XHR, aber identisches API
- Unterschiede z.B. beim Caching von Antworten
- Standardisierung durch Web APIs WG innerhalb des W3C
- Frameworks und Toolkits

JavaScript	Prototype, dojo, Sarissa, Adobe Spry, ...
Java	Google Web Toolkit (GWT), Apache Struts, ...
PHP	Sajax, Xajax
Python	TurboGears
Ruby	Ruby on Rails
.NET	Microsoft Atlas
Perl	CGI::Ajax, Nutzung von Prototype oder dojo

Beispiel, Server-Seite

- einfacher „case converter“ 
- Server-Seite per Shell-Skript
  - lies Text von STDIN
  - wandle Groß- in Kleinbuchstaben und umgekehrt
  - schreibe nach STDOUT
- kann per CGI/POST benutzt werden

```
#!/bin/sh
echo "Content-Type: text/plain"
echo
tr a-zA-Z A-Za-z
```

Beispiel, HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Simple AJAX Test</title>
    <script type="text/javascript" src="chcase1.js"></script>
  </head>
  <body>
    <form action="" method="post" onsubmit="chcase();return false">
      <div><input id="input" size="50"/></div>
      <div id="output"></div>
    </form>
  </body>
</html>
```

Beispiel, Client-Seite

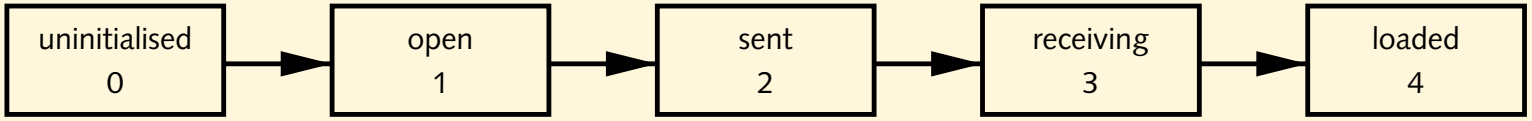
```
var http = null;
try { http = new ActiveXObject('Msxml2.XMLHTTP'); }
catch (err) {
    try { http = new ActiveXObject('Microsoft.XMLHTTP'); }
    catch (err) { http = null; }
}
if (!http && typeof XMLHttpRequest != 'undefined') {
    http = new XMLHttpRequest();
}

function chcase() {
    var text = document.getElementById('input').value;
    http.open('POST', '/cgi-bin/chcase.sh');
    http.onreadystatechange = handleResponse;
    http.send(text);
}

function handleResponse() {
    if (http.readyState == 4 && http.status == 200) {
        document.getElementById('output').innerHTML = http.responseText;
    }
}
```

Währenddessen . . .

- Request durchläuft mehrere Phasen



- handler wird bei jeder Zustandsänderung aufgerufen

```
if (http.readyState == 4 && http.status == 200) ...
```

- HTTP return code

```
if (http.readyState == 4 && http.status == 200) ...
var msg = http.statusText;
```

- HTTP header setzen und lesen

```
http.setRequestHeader('Pragma', 'no-cache');
var bestbefore = http.getResponseHeader('Expires');
```

Arbeiten mit XHR

- zunächst ungewohnt: Asynchronität
- ähnlich zu Signal-Verarbeitung oder GUI-Programmierung
- Die Antwort ist 42! Was war die Frage?

```
<form action="" method="post" onsubmit="chcase(1);return false">
  <div><input id="input1" size="50"/></div>
  <div id="output1"></div>
</form>
<hr/>
<form action="" method="post" onsubmit="chcase(2);return false">
  <div><input id="input2" size="50"/></div>
  <div id="output2"></div>
</form>
```

Arbeiten mit XHR, cont.

- funktioniert nicht:

```
function chcase(num) {
    var text = document.getElementById('input'+num).value;
    http.open('POST', '/cgi-bin/chcase.sh');
    http.onreadystatechange = handleResponse(num);
    http.send(text);
}

function handleResponse(num) {
    if (http.readyState == 4 && http.status == 200) {
        document.getElementById('output'+num).innerHTML = http.responseText;
    }
}
```

- handler nur Funktionsname
- mögliche Lösung: Identifikation in Request/Response

Arbeiten mit XHR, cont.

- funktioniert nicht:

```
function chcase(num) {
    var text = document.getElementById('input'+num).value;
    http.open('POST', '/cgi-bin/chcase.sh');
    http.onreadystatechange = handleResponse(num);
    http.send(text);
}

function handleResponse(num) {
    if (http.readyState == 4 && http.status == 200) {
        document.getElementById('output'+num).innerHTML = http.responseText;
    }
}
```

- handler nur Funktionsname
- mögliche Lösung: Identifikation in Request/Response
- elegantere Lösung: closure

```
    http.onreadystatechange = function() { handleResponse(num); };
```

Latenz

- „find as you type“: Google Suggest
- erster Versuch: 

```
<form action="" method="post" onkeyup="chcase();return false">
  <div><input id="input" size="50"/></div>
  <div id="output"></div>
</form>
```

- neue Requests, bevor Responses eingetroffen sind
- XHR kann nur eine R/R-Transaktion gleichzeitig verarbeiten
- Lösung: `abort()` und ein frisches XHR-Objekt 

Latenz, cont.

```
var http=null;

function mkXHR () {
    // browser sniffer
    return XHR;
}

function chcase() {
    if (http && http.readyState != 0) {
        http.abort();
    }
    http = mkXHR();
    if (http) {
        http.open('POST', '/cgi-bin/chcase.sh');
        http.onreadystatechange = handleResponse;
        http.send(document.getElementById('input').value);
    }
}
```

Response verarbeiten

- JSON
  - JavaScript Object Notation
  - mit `eval()` oder Parser in Datenstruktur umwandeln
- XML
  - muss nicht, kann aber
  - `http.responseXML` enthält XML-Antwort als DOM-Objekt
  - mit XSLT in (X)HTML umwandeln und darstellen
  - Beispiel mit XSLT-Prozessor von Firefox/Mozilla/Sarissa 

Film-Datenbank, HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Simple AJAX Film Database</title>
    <script type="text/javascript" src="prototype.js"></script>
    <script type="text/javascript" src="sarissa.js"></script>
    <script type="text/javascript" src="films.js"></script>
    <link href="films.css" type="text/css" rel="stylesheet"/>
  </head>
  <body onload="showFilms()">
    <div id="films"></div>
    <hr/>
    <div id="roles"></div>
    <hr/>
    <div id="person"></div>
  </body>
</html>
```

Film-Datenbank, Client-Seite

```
var xml = mkXHR();
var xslt = mkXHR();

function showFilms () {
    xml.open('GET', '/cgi-bin/filmdb.pl?q=films');
    xml.onreadystatechange = showFilms_H;
    xml.send(null);
    xslt.open('GET', 'films2html.xsl');
    xslt.onreadystatechange = showFilms_H;
    xslt.send(null);
}

function showFilms_H {
    if (xml.readyState == 4 && xml.status == 200 &&
        xslt.readyState == 4 && xslt.status == 200) {
        var transformer = new XSLTProcessor();
        transformer.importStylesheet(xslt.responseXML);
        html = transformer.transformToFragment(xml.responseXML, document);
        $('films').innerHTML = ''; $('films').appendChild(html);
    } else {
        $('films').innerHTML = 'please wait';
    }
}
```

Film-Datenbank, XML-Response

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<films>
  <film>
    <filmid>1</filmid>
    <runtime>136</runtime>
    <title>The Matrix</title>
    <year>1999</year>
  </film>
  <film>
    <filmid>2</filmid>
    <runtime>251</runtime>
    <title>The Lord of the Rings - The Return of the King</title>
    <year>2003</year>
  </film>
  ...
</films>
```

Film-Datenbank, XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">

  <xsl:template match="/films">
    <table>
      <xsl:apply-templates/>
    </table>
  </xsl:template>

  <xsl:template match="film">
    <tr>
      <td><a href="javascript:showRoles({filmid})">
          <xsl:value-of select="title"/>
      </a></td>
      <td><xsl:value-of select="year"/></td>
      <td><xsl:value-of select="runtime"/> min.</td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

Kritik

- Benutzbarkeit
  - Bruch mit klassischer Navigation (back button, bookmarks, ...)
  - Anwendung fast immer RESTless (Zustand nicht an URL gebunden)
  - aber: gilt für nicht-Web-Anwendungen genauso
  - kürzere Reaktionszeiten, geringerer Datentransfer
- Barrierefreiheit
  - Ziel ist oft Immitation klassischer GUIs
  - inkrementeller Seitenaufbau problematisch für Vorleser, Braille-Zeilen
  - aber: vermutlich besser als Java-Applets oder Flash-Filme
- Sicherheit:
  - relativ neue Technik
  - Cross Site Scripting, JSON + `eval()`, ...

Mashups

- Daten und Dienste können intern oder extern sein

Daten	Dienst	
intern	intern	Film-Quiz
intern	extern	GPS Tour Book
extern	intern	RSS Feed Reader
extern	extern	USGS Live Earthquake Mashup

- Problem bei externen Daten: „Same Origin Policy“ in JavaScript
- auf andere Sprachen/Methoden ausweichen (PHP, application proxy, `...`)
- `mod_proxy`/`mod_rewrite` auf dem lokalen Webserver

USGS Live Earthquake Mashup

- RSS Feed vom USGS (United States Geological Survey)

```
<item>
  <pubDate>Sun, 22 Oct 2006 17:12:28 GMT</pubDate>
  <title>M 2.9, Island of Hawaii, Hawaii</title>
  <description>October 22, 2006 17:12:28 GMT</description>
  <link>http://earthquake.usgs.gov/eqcenter/recenteqsww/Quakes/hv00021312.php</link>
  <geo:lat>19.9292</geo:lat>
  <geo:long>-155.9570</geo:long>
</item>
```

- lokalen URL für Feed einrichten

```
RewriteRule ^/ajxp/usgseq-feed.xml
  http://earthquake.usgs.gov/eqcenter/catalogs/feed.php?feed=eqs7day-M2.5.xml
  [L,P]
```

- XHR in Google Maps API

```
GDownloadUrl('/ajxp/usgseq-feed.xml', showFeed);
```

Google Maps API

- gute Dokumentation
- einfach in eigene Web-Seiten einzubinden (nach vorheriger Registrierung)
- Overlays aus Punkten und Polygonzügen
- Popup-Fenster mit Text oder HTML
- XML-Parser, XSLT-Prozessor
- Geocoder