

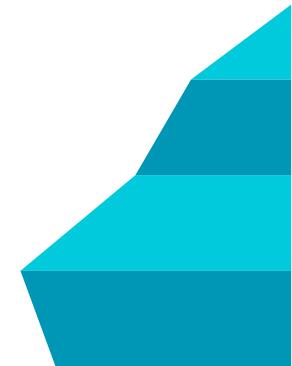


FH Bielefeld
University of
Applied Sciences

Campus Minden



http://upload.wikimedia.org/wikipedia/commons/2/25/Aphaia_pediment_Aias_W-IX_Glyptothek_Munich_80.jpg?uselang=de



Studiengang Informatik Fachbereich Technik

Webbasierte Anwendungen

SS 2015

Clientseitige Implementierungstechnologien:

Ajax

Dozentin: Grit Behrens
<mailto:grit.behrens@fh-bielefeld.de>



Studiengang Informatik Fachbereich Technik

Lehrinhaltsübersicht der Vorlesungen zu WBA

1. Einführung in WBA
2. Wiederholung: Grundlagen des WWW, HTML und HTTP
3. **Clientseitige Implementierungstechnologien:** Javascript, DOM, **Ajax, jQuery,**(Java-Applet)
4. Serverseitige Implementierungstechnologien: JSP, Java-Servlet
5. Anbindung von Datenbanken
6. WEB-Frameworks: JSF, Struts

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

Clientseitige Implementierungs-technologien (III): Ajax

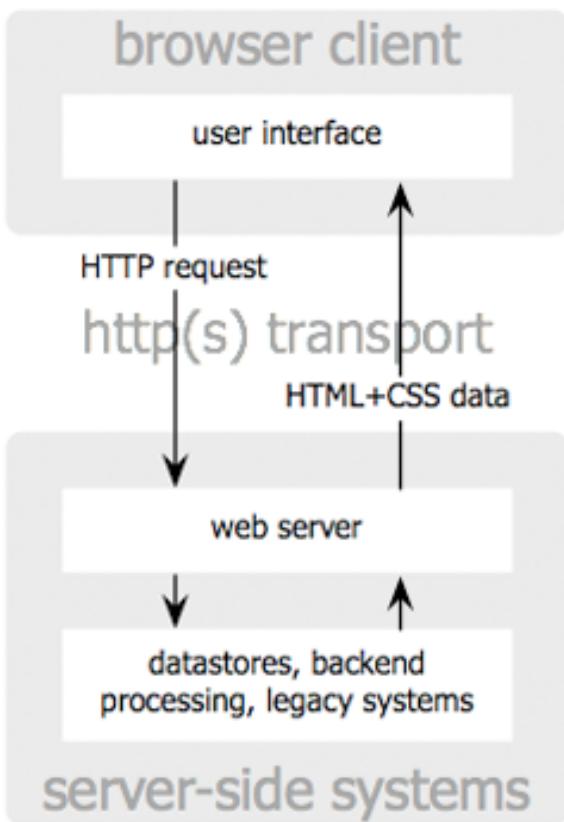
- 1. Einführung**
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

Einführung

- **AJAX** bedeutet ausgeschrieben: **A**synchron **J**avascript **A**nd **X**ml
- keine neue Technologie, sondern **sinnvolle und kreative Kombination bestehender Technologien**
- „Ajax“ funktionierte längere Zeit als **Marketing-Begriff (ca. 2005-2010)**
- Begriff wurde im Februar 2005 von Jesse James Garrett geprägt
- Begriff sollte beitragen, eine neue Generation der Webentwicklung zu beschreiben und den Begriff „WEB2.0“ zu untermauern
- **Merkmale von Ajax:**
 - Datenübertragung zwischen Client und Server asynchron im Hintergrund
 - der Anwender bemerkt nichts von der Datenübertragung
 - Erscheinungsbild einer Desktopanwendung:
 - Formulare müssen nicht mehr komplett ausgefüllt und abgeschickt werden, bevor man auf den Response warten kann
 - Aktuelle Seite wird nicht komplett neu geladen
 - nur relevante Inhalte mit den gerade neu angeforderten Dateninhalten werden aktualisiert

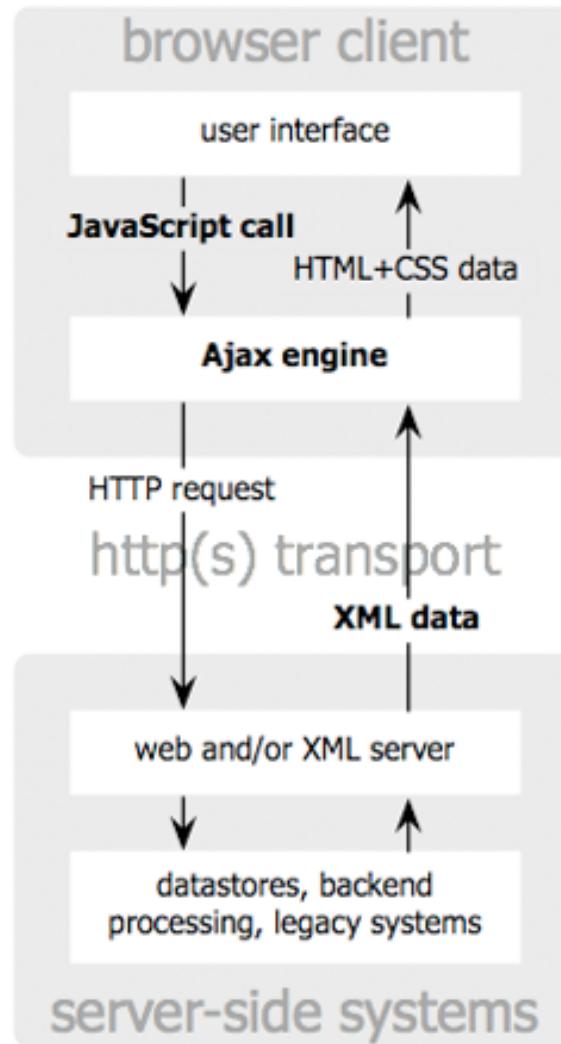
Einführung

- Komponenten von Ajax:
 - XMLHttpRequest-Objekt
 - JavaScript als Schnittstelle aller Komponenten
 - XHTML und CSS
 - DOM
 - XML
 - auch andere Formate zum Datenaustausch wie Text, Strukturen, JSON
- Toolset der modernen Webentwicklung
- schafft mehr Komfort und Flexibilität der Anwendungen



classic web application model

Jesse James Garrett / adaptivepath.com

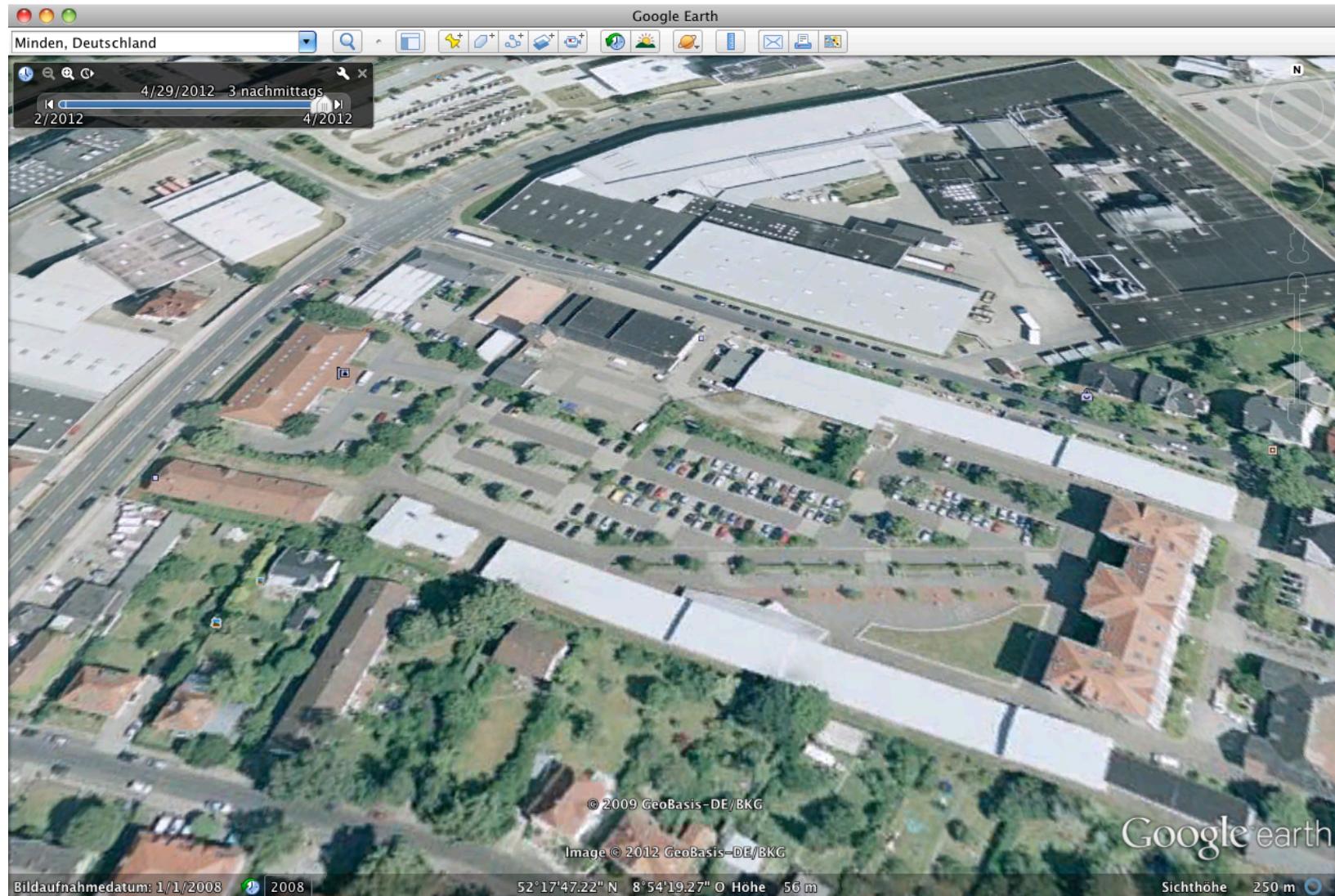


Ajax web application model

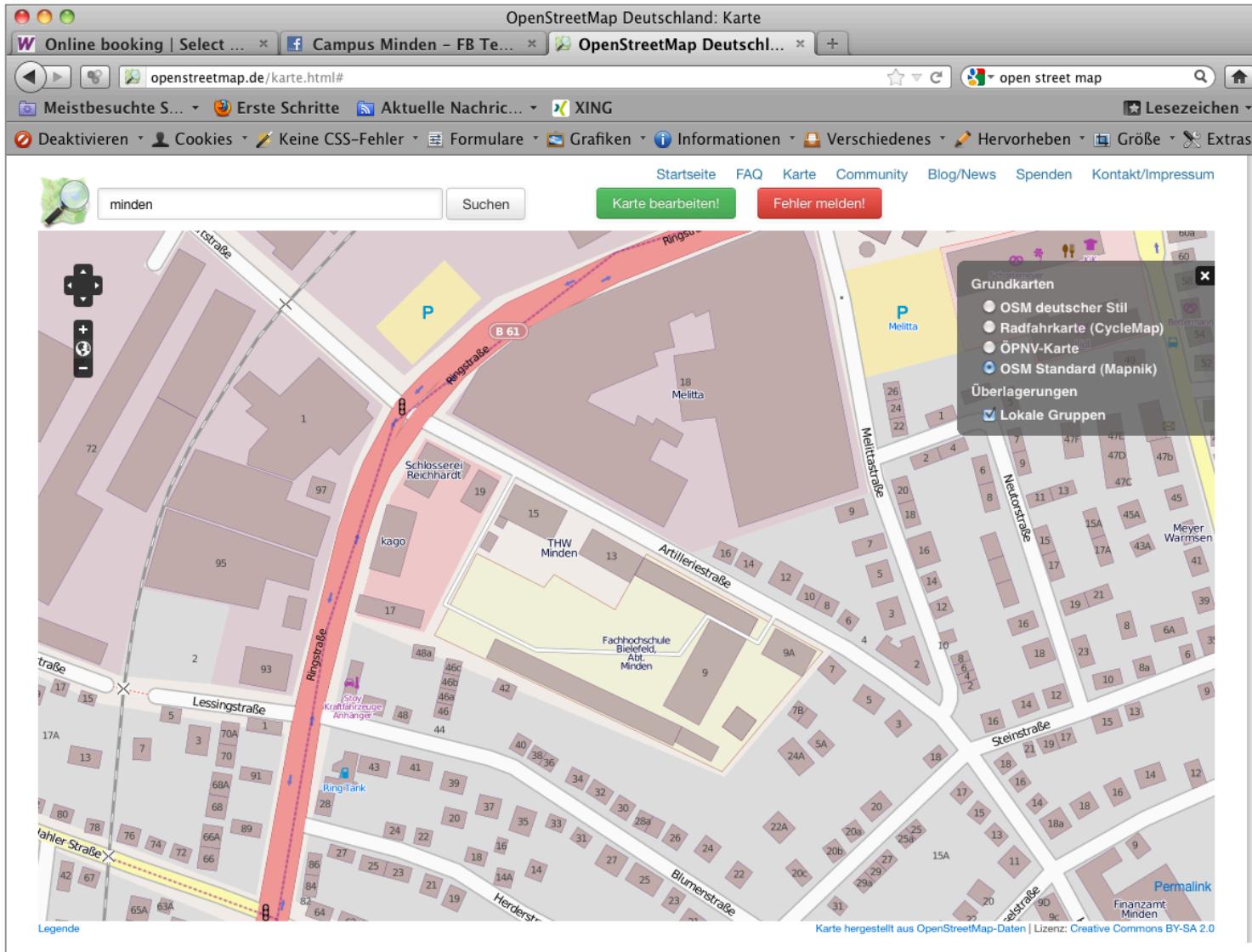
Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
- 2. Beispielanwendungen**
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

Beispielanwendungen: virtuelle Landkarten



Beispielanwendungen: virtuelle Landkarten



Beispielanwendungen: Soziale Netzwerke

The screenshot shows a Flickr tour page. On the left, there's a sidebar with links like 'Mit anderen teilen', 'Hochladen und verwalten', 'Entdecken', and 'Sie können noch mehr tun!'. Below that are buttons for 'Kostenlosen Account einrichten' and 'oder Anmelden'. Further down are links for 'Brauchen Sie Hilfe?' and 'FAQs'. The main content area features a large photo of a man holding a dog. A green annotation box with the text 'Her ears look just like Max's! - d_leung' is overlaid on the dog's ear. To the right of the photo is a box for the photo's author, Phil King, with a map of California and a small photo of him. Below the photo are sections for comments ('Kommentare und Favoriten') and tags ('Tags').

Annotations:

- 1: Kommentare und Favoriten
- 2: Annotation box on the dog's ear.
- 3: Map of California showing the location of the photo.
- 4: Author's profile box.
- 5: Tag cloud.

Beispielanwendungen: Soziale Netzwerke

The screenshot shows a Facebook page titled "Solar Computing Lab". The page header includes tabs for "Seite", "Aktivität 2", "Statistiken", and "Einstellungen". A sidebar on the right displays statistics: 5 "Gefällt mir"-Angaben, 37 Beitragserreichweite, 2 Benachrichtigungen, and 0 Nachrichten. The main content area features a large photo of five people standing behind a large geodesic dome structure. Below the photo is a cover image for "Solar Computing Lab" which is a "Computer/Internet-Webseite". The cover image includes a blue logo with binary code and the text "Solar Computing Lab". Below the cover are links for "Chronik", "Info", "Fotos", and "„Gefällt mir“-Angaben". A news feed item from "Solar Computing Lab" is visible, sharing a link posted by Grit Behrens on April 11, 2013, with the caption "Mit einem Solarflugzeug rund um die Erde fliegen!". The browser's address bar shows the URL <https://www.facebook.com/SolarComputingLab?ref=ts>.

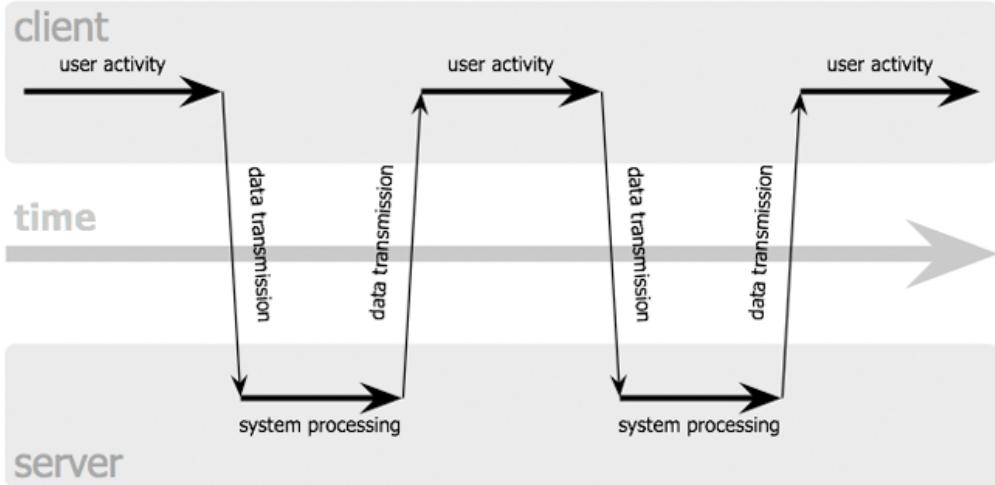
Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
- 3. Asynchron oder synchron?**
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

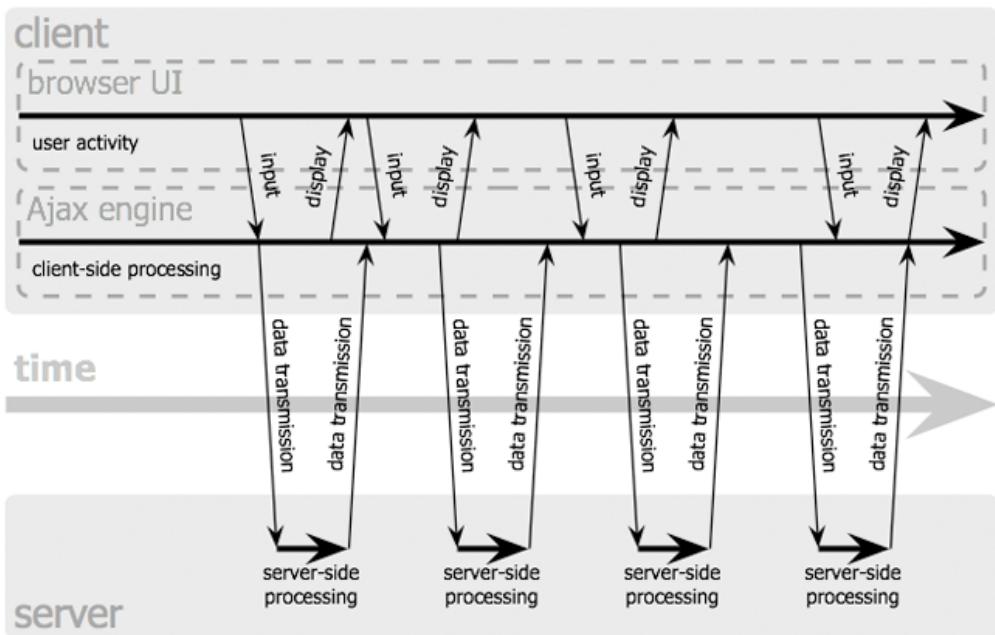
Ajax - Datenübertragung: synchron oder asynchron?

- Antwort erfolgt erst nach vollständiger Übertragung aller angeforderten Daten
- Anfrage bleibt aktiv, bis Antwort vom Server erzeugt wurde
-> Blockade des weiteren Programmablaufs
- Ergebnis nach der Übertragung sofort und komplett verfügbar
- keine Statusinformationen während Datenübertragung verfügbar
-> bei längeren Datenübertragungen wird vom Anwender aus Unwissenheit oft abgebrochen
- Antwort des Servers erfolgt sofort
-> Requests sind nicht an den Rhythmus von „Formular absenden“ oder „Seite laden“ usw. gebunden
- Nächster Request kann gestellt werden, noch bevor vorheriger beantwortet wurde
- beliebige Elemente einer Seite können isoliert aktualisiert werden.
- aktueller Status wird während der Anfrage ständig geliefert
-> optimale Kontrolle über den Ablauf der Anfrage an den Server

classic web application model (synchronous)



Ajax web application model (asynchronous)



[http://
www.adaptivepath.com/ideas/
essays/archives/000385.php](http://www.adaptivepath.com/ideas/essays/archives/000385.php)

Jesse James Garrett / adaptivepath.com

Ajax: Vorteile und Nachteile

Vorteile:

- Daten können verändert werden **ohne Neuladen** der kompletten Webseite
- **Schnellere Reaktion auf Benutzereingaben**
- statische unveränderte Daten einer Webseite müssen **nicht neu im Internet übertragen** werden

Nachteile:

- Unterschiedliche Browserimplementierungen (werden jetzt überwunden)
 - Nachteile werden zum Teil durch JavaScript-Bibliotheken beseitigt, die genutzt werden können
 - Erfolgreichstes OpenSource-Framework **jQuery**
- Verwendung der „zurück“-Schaltfläche eingeschränkt (da nur statische Seiten im Browsecache gespeichert werden können)

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
- 4. Das XMLHttpRequest-Objekt**
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks

Das XMLHttpRequest-Objekt

- Herz von Ajax: ermöglicht Client/Server-Kommunikation mittels JavaScript
- wurde von Microsoft eingeführt ab MSIE 5.0 in einer **AktiveX-Komponente**, welche Daten von einem Server anfordern kann
- Mozilla 1.0 und Netscape 7 (beide auch Gecko-Browser genannt) zogen schnell nach mit dem **XMLHttpRequest-Objekt**
- Übertragung von Text- und XML-Daten möglich
- in jQuery gibt es das **jQXHR-Objekt**
 - Datenübertragung von text, Klartext mit html, xml, **JSON**

Das JSON-Format

- **JavaScript Object Notation**
- universelle Datenbeschreibung
- beinhaltet direkt JavaScript-Datentypen und Javascript- Datenstrukturen
(Objekte, Arrays, Zeichenketten, Zahlen, boolsche Werte)
- maschinenlesbares Klartextformat zum Datenaustausch (wie XML)
- Bibliotheken zum Erstellen und Parsen in vielen Programmiersprachen
 - Javascript:
 - `eval()`
 - `JSON.parse()`
- Syntax: z.B. kommaseparierte ungeordnete Liste von Objekteigenschaften in geschweiften Klammern

```
{  
    "Attribut1": "Wert1",  
    "Attribut2": [ "Wert21", "Wert22", "Wert23"]  
}
```

*erzeugen aus einem JSON-String
ein JavaScript-Objekt*

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
- 4. Das XMLHttpRequest-Objekt**
 - 1. Erzeugung**
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

Erzeugung des XMLHttpRequest-Objekts

Mozilla Firefox:

```
var ajax = new XMLHttpRequest();
```

Microsoft Internet Explorer

- ältere Versionen:

```
var ajax = new ActiveXObject("Microsoft.XMLHTTP");
```

- neue Versionen auch zusätzlich:

```
var ajax = new XMLHttpRequest();
```

Erzeugung des XMLHttpRequest-Objekts

Beispiel für browserunabhängige Erzeugung des XMLHttpRequest-Objektes:

```
var req = (window.XMLHttpRequest)
?
new XMLHttpRequest()           //erfüllt-Anweisung für MF u. MSIE ab 7.X
:
((window.ActiveXObject)) // nicht-erfüllt-Anweisung für MSIE
?
new ActiveXObject("Microsoft.XMLHTTP") //erfüllt für MSIE
:
false      //es war keine der beide Bedingungen erfüllt, kein Objekt erzeugt
);
```

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
- 4. Das XMLHttpRequest-Objekt**
 1. Erzeugung
 - 2. Methoden**
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

XMLHttpRequest-Objekt: Methoden

Es werden Methoden betrachtet, welche für beide Objekte im MF und MSIE implementiert sind.

Methode	Beschreibung

XMLHttpRequest-Objekt: Methoden

Es werden Methoden betrachtet, welche für beide Objekte im MF und MSIE implementiert sind.

Methode	Beschreibung
<code>abort()</code>	aktuell laufender Request wird beendet
<code>getAllResponseHeaders()</code>	gibt Liste aller vorhandenen Header als key/value-Paare in einem String zurück
<code>getResponseHeader(name)</code>	gibt Wert für angegebenen Headername zurück
<code>open(method, url[, syncFlag, username, password])</code>	Starten eines Request an Zieladresse (url) und Vereinbarung der Übertragungsart
<code>send(body null)</code>	sendet Request an den Server Body ist optional
<code>setRequestHeader(key, value)</code>	Setzen eines optionalen Headers für Request als key/value- Paar

XMLHttpRequest-Objekt: Methoden

- ein gültiger Request benötigt mindestens die Methoden `open()` und `send()`.

`open(method, url[, syncFlag, username, password])`

- startet Anfrage an den Server
- erwartet Methode: meist GET
- Angabe der URL für gefordertes Dokument (z.B. XML oder Text)
- Art der Übertragung wird durch syncFlag angegeben:
 - true – asynchron (default)
 - false - synchron
- bei method = PUT oder DELETE Authentifizierung auf Server notwendig:
 - username
 - passwort
- Beispielaufruf minimal:

```
req.open("GET", "beispiel.xml");
```

`send(body | null)`

- optional kann Inhalt mit übergeben werden: String, DOM-Objekt, XML-Objekt, Formulareingabedaten, ...
- Beispielaufufe:
`req.send(null);`
`req.send("");`
`req.send(xml);`

XMLHttpRequest-Objekt: Methoden

setRequestHeader()

- dient der Übergabe von Headerinformationen an den Server
- muss vor der send-Methode stehen
- kann mehrfach hintereinander aufgerufen werden, um mehrere Request-Header zu setzen
- es können zusätzlich eigene Merkmale definiert werden
- Beispielaufrufe:

```
req.setRequestHeader("Accept", "image.gif"); //offizieller Http_header  
req.setRequestHeader("PersonalExample", "value"); //eigenes Merkmal
```

getAllResponseHeaders(); getResponseHeader()

- liefern Headerinformationen des Servers
- ResponseHeader sind zum Teil browserspezifisch implementiert

abort()

- beendet einen Response vorzeitig, z.B. bei Zeitüberschreitung einsetzbar

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
- 4. Das XMLHttpRequest-Objekt**
 1. Erzeugung
 2. Methoden
 - 3. Eigenschaften**
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

XMLHttpRequest-Objekt: Eigenschaften

Es werden Eigenschaften betrachtet, welche für beide Objekte im MF und MSIE übereinstimmend implementiert sind.

Eigenschaft	Beschreibung
<code>onreadystatechange</code>	Event-Handler, der eine zugeordnete Funktion bei jeder Änderung im Status des Requests ausführt.
<code>readyState</code>	Gibt den aktuellen Status des Requests zurück und ändert sich bis zum Ende der Anfrage
<code>responseText</code>	Serverantwort als String im Body
<code>responseXML</code>	Serverantwort als XML-Objekt
<code>status</code>	Numerischer Wert des Serverstatus; beträgt am Ende der Übertragung den Wert 4
<code>statusText</code>	String mit einer Beschreibung des Serverstatus. Sobald die Übertragung beendet ist.

XMLHttpRequest-Objekt: Eigenschaften

readystate (Status des Requests) : Werte und deren Bedeutung

Wert	Bedeutung	Beschreibung
0	uninitialized	Request wurde noch nicht durch <code>open()</code> ausgelöst
1	loading	Request wird gestartet, wurde aber bisher noch nicht abgeschickt
2	loaded	Request wurde durch <code>send()</code> ausgeführt, die Serverantwort steht noch aus
3	interactive	Übertragung der Serverantwort läuft, Teile davon sind schon im Buffer und mittels <code>responseText</code> oder <code>responseXML</code> verfügbar
4	complete	Request wurde vollständig ausgeführt und beendet

XMLHttpRequest-Objekt: Eigenschaften

• **onreadystatechange**

- ist ein EventHandler und dient dazu, den aktuellen Status der Anfrage zu erfahren
- wird in Verbindung mit der Eigenschaft readyState verwendet
- eine zugewiesene Funktion kann ausgelagert oder direkt an den Handler gebunden werden
 - zum Beispiel:

```
var req.onreadystatechange = funktionsname; //ausgelagerte Funktion
```

```
var req.onreadystatechange = function() ; // Funktion direkt an Handler gebunden
{
...
}
```

- **readyState**, **status** und **statusText** können gemeinsam genutzt werden, um für den jeweiligen Zustand eine Aktion anzubieten:

```
if (req.readyState == 4) { //request erfolgreich ausgeführt?
    if (req.status==200 || req.statusText=="OK") { //gültiges Dokument im resp.?
        ... Antwort verarbeiten ...
    }
    else {
        ... Fehler in der Anfrage...
        alert("ERROR: \n": + req.statusText) ; } } //Ausgabe der Fehlerinfo des Servers
```

XMLHttpRequest-Objekt: Eigenschaften

- **responseText** und **responseXML**

- beide Eigenschaften stehen zur Weiterverarbeitung der Serverantwort zur Verfügung
- enthalten die angeforderten Daten in der Serverantwort
- zum Beispiel:

```
var str = req.responseText; //Text - Antwort  
var xml = req.responseXML; //XML – Antwort (muss Mimetype text/xml im Header  
besitzen)
```

XML oder Text im Body der Serverantwort?

- „XML“ kann über DOM weiterbearbeitet werden
- bei Aktualisierung nur eines einzelnen Elements genügt „Text“
- für umfangreiche Datenstrukturen sollte XML verwendet werden

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
- 5. Erstes Beispiel (Hallo Ajax)**
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

Erstes Beispiel: Hallo-Ajax!

- mittels Ajax soll eine Datei auf dem Server angefragt werden
- der Inhalt dieser Datei soll in einer Dialogbox und in einem XHTML-Element ausgegeben werden
 - Serverdokument ist ein Textdokument **hallo.txt** mit dem Inhalt: “**Hello Ajax!**“

Ajax-Beispiel in einer XHTML-Datei mit dem Aufbau:

```
<!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" xml:lang="de" lang="de">
<head>
<title>Hello Ajax</title>
<script type="text/javascript">
//<![CDATA[
window.onload = function() //verursacht Ausführen der Anfrage nach Aufruf der Seite
... // Funktionsinhalt auf der kommenden Seite ...
//]]>
</script>
</head>
<body>
<div id="hallo"></div> //im Element mit CSS-ID „Hallo“ soll Inhalt der Datei hallo.txt angezeigt werden
</body>
</html>
```

Erstes Beispiel: Hallo-Ajax!

Ajax-Beispiel : Inhalt der Funktion aus `window.onload = function()`

```
window.onload = function()
{
    var req = (window.XMLHttpRequest) // Erzeugung des XMLHttpRequest
        ?
        new XMLHttpRequest()
        :
        (window.ActiveXObject)
        ?
        new ActiveXObject("Microsoft.XMLHTTP")
        :
        false
    );

    req.open("GET","http://localhost:8080/LearnAjax/hallo.txt",true);
    req.onreadystatechange = function()
    {
        if (req.readyState==4)      // XMLHttpRequest beendet
        {
            if (req.status == 200)//http status 200 : ok
            {
                var d = document.getElementById("hallo"); //Auswahl der Einfügestelle in die XHTML-
                                                // Seite mit DOM - Methoden
                d.innerHTML = req.responseText;           // Einfügen der Serverantwort in XHTML
                alert(req.responseText);                  // Ausgabe der Serverantwort als Dialogbox
            }
        }
    }
    req.send(null)
}
```

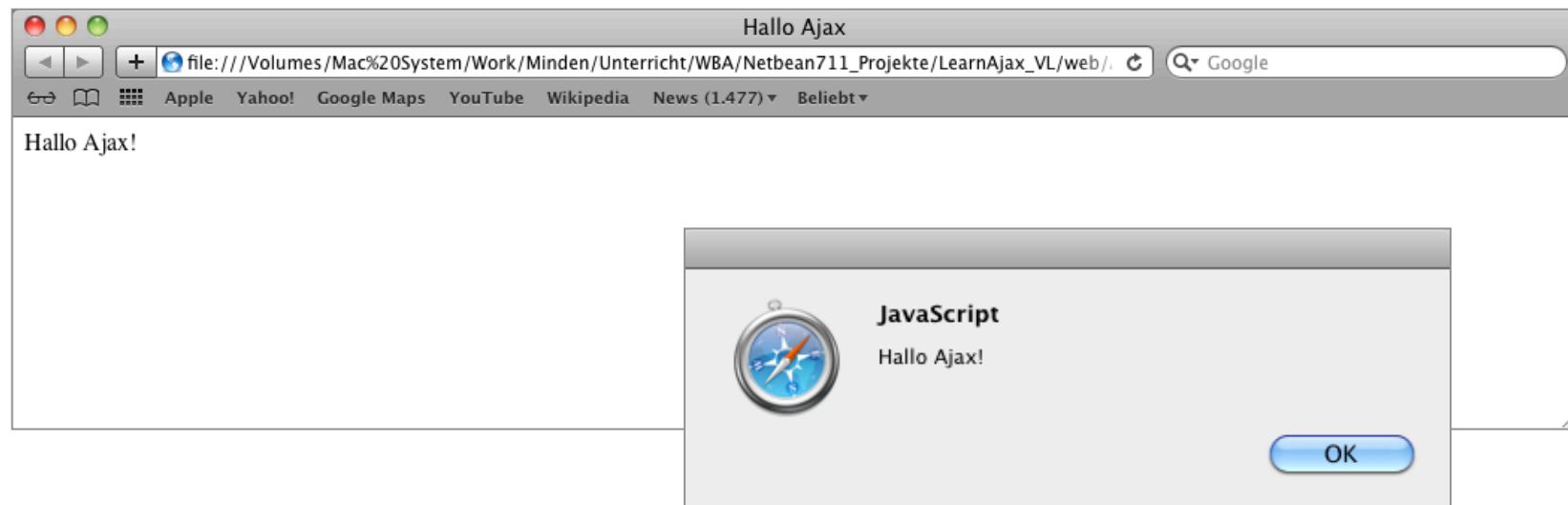
Erstes Beispiel Hallo-Ajax! : Listing

```
<!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" xml:lang="de" lang="de">
<head>
    <title>Hallo Ajax</title>
<script type="text/javascript">
//<![CDATA[
window.onload = function()
{
    var req = (window.XMLHttpRequest)
    ?
    new XMLHttpRequest()
    :
    ((window.ActiveXObject)
    ?
    new ActiveXObject("Microsoft.XMLHTTP")
    :
    false
    );

    req.open("GET", "http://login1.informatik.fh-wiesbaden.de/~behrrens/hallo.txt", true);
    req.onreadystatechange = function()
    {
        if (req.readyState==4)
        {
            if (req.status == 200)
            {
                var d = document.getElementById("hallo");
                d.innerHTML = req.responseText;
                alert(req.responseText);
            }
        }
        req.send(null)
    }
//]]>
</script>
</head>
<body>
<div id="hallo"></div>
</body>
</html>
```

Erstes Beispiel: Hallo-Ajax! : Browseransicht Safari

Ausführung im Safari



Im Browserfenster wurde in das div-element mit id=„hallo“ vom DOM mit „Hallo Ajax!“ aus der Serverantwort überschrieben + Alertfenster mit „Hallo Ajax“-Ausgabe (inhalt des Text-Responses)

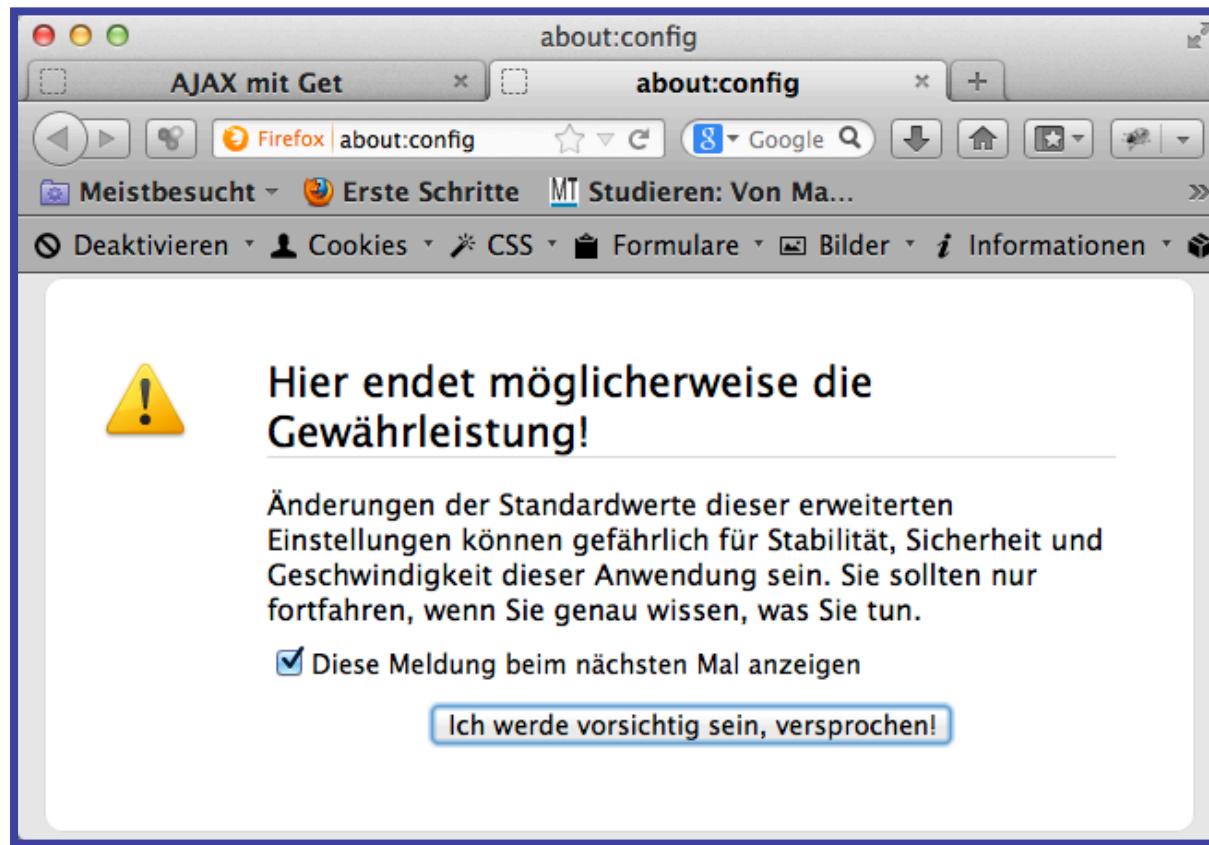
Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
- 6. Nutzung externer Quellen**
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
9. JQuery

Nutzung externer Quellen

- Einbeziehung fremder Inhalte beinhaltet Sicherheitslücken
- bösartige Programmierer können diese zum Aufruf kritischer Aktionen nutzen
- **MSIE Browser** warnen vor Zugriff auf externe Quellen je nach Sicherheitseinstellung mit Dialogbox, die mit ja bestätigt werden muß
- **Mozilla Firefox** hat restiktivere Sicherheitsbestimmungen:
 - System von Privilegien
(<http://www.mozilla.org/projects/security/components/signed-scripts.html>)
 - Privilegien können im Browser manuell eingestellt werden mit:
about:config oder
 - Nutzung des Objektes **netscape.security.PrivilegeManager** verbunden mit Dialogbox

Nutzung externer Quellen: z.B. manuelles Einstellen der Privilegien im MF(2014)



Eingabe von `about:config` in die Eingabezeile des Browsers

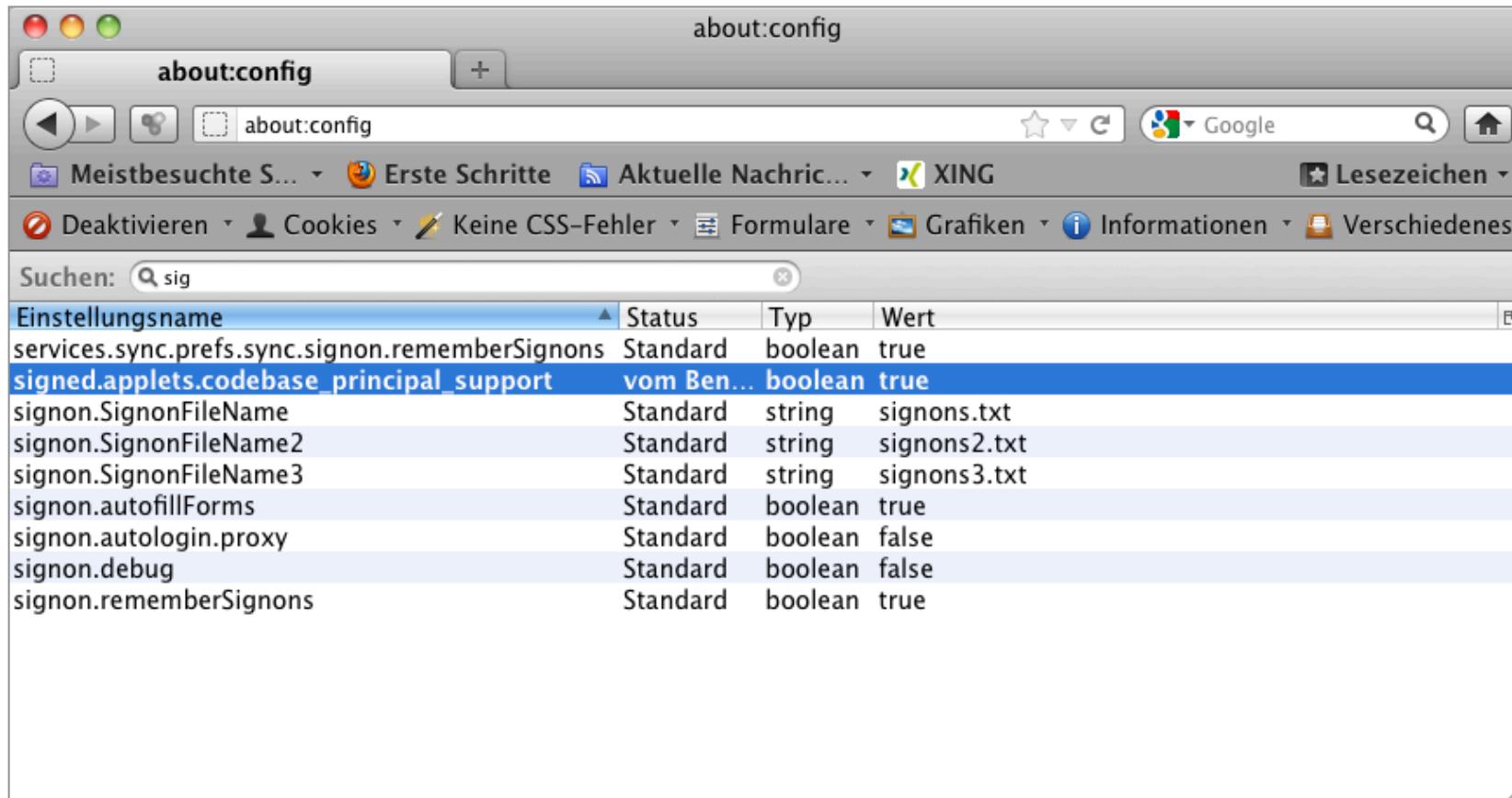
Nutzung externer Quellen: z.B. manuelles Einstellen der Privilegien im MF (2014)

The screenshot shows the Firefox browser window with the 'about:config' page open. The search bar at the top contains the text 'sig'. Below the search bar is a table with four columns: 'Einstellungsname', 'Status', 'Typ', and 'Wert'. The table lists several configuration parameters related to signatures:

Einstellungsname	Status	Typ	Wert
dom.mozApps.signed_apps_installation	Standard	string	https://marketplace.firefox.com
dwhelper.yt-sig-perm-81	Standard	string	[[56,57],[79,80],[78,79],[77,78],...]
dwhelper.yt-sig-perm-83	Standard	string	[[6,7],[3,6],[33,34],[7,24],[0,1],[2...
dwhelper.yt-sig-perm-85	Standard	string	[[2,8],[0,1],[9,21],[65,66],[22,65]...
dwhelper.yt-sig-perm-86	Standard	string	[[80,81],[81,82],[26,27],[79,80],...]
dwhelper.yt-sig-perm-87	Standard	string	[[4,23],[86,87],[24,85]]
dwhelper.yt-sig-perm-89	Standard	string	[[78,79],[83,84],[82,83],[81,82],...]
dwhelper.yt-sig-perm-92	Standard	string	[[25,26],[3,25],[2,3],[26,42],[79,...]
security.enable_md5_signatures	Standard	bool...	false
services.sync.prefs.sync.signon.remem...	Standard	bool...	true
signed.applets.codebase_principal...	vom B...	bool...	true
signon.SignonFileName	Standard	string	signons.txt
signon.SignonFileName2	Standard	string	signons2.txt
signon.SignonFileName3	Standard	string	signons3.txt
signon_autoGIFForms	Standard	bool	true

Bei Eingabe der ersten Zeichen „sig“ werden zugehörige Konfigurationsparameter herausgefiltert.

Nutzung externer Quellen: manuelles Einstellen der Privilegien im MF (2014)



Durch Klick oder Doppelklick auf den gewünschten Wert, kann die Einstellung auf „True“ geändert werden.

Prinzipiell: Nutzung externer Quellen

- Ajax-Anfragen dürfen in der Regel nur auf dem gleichen Server Daten anfordern, von dem die anfordernde Webseite stammt.
 - Sandkastenprinzip „**Sandbox“ mit „Same-Origin-Policy“**
- Missbrauch bei sog. **Cross-Domain-Acess** soll verhindert werden.
- Lösung in jQuery mit **JSONP-Format**
 - Nachladen eines Scriptes **\$.getscript()**
 - Script laden Cross Domain (ist erlaubt)
 - Die Scripte enthalten **JSON-Daten**
 - **JSON-Daten** können so von beliebigen Servern geladen werden

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
- 7. XML-Dateien auswerten**
8. Bibliotheken und Frameworks
9. JQuery

XML - Dateien auswerten

- Eigenschaft des XMLHttpRequest-Objekts : `responseXML` wird gesetzt, um den Mimetype der Serverantwort als XML-Objekt festzulegen.
- Das Serverscript mussMimeType XML liefern

z.B. als JSP:

```
<?xml version="1.0" encoding="UTF-8"?>
<% response.setContentType("text/xml") ;%>
```

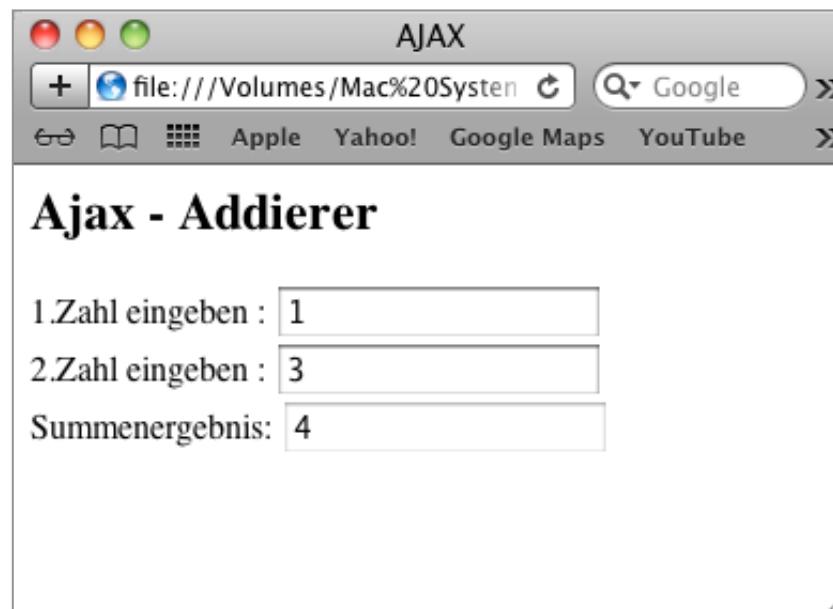
oder als PHP

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?php header("Content-type: text/xml");
```

- Auswerten der Rückgabedatei mit den Methoden des DOM

Ajax mit XML: Beispiel „Addierer“

1. Anlegen eines Serverdienstes, der zwei Integer addiert
JSP **AjaxWithXmlSumme.jsp** erreichbar unter
(z.B.) `http://localhost:8080/LearnAjax/AjaxWithXmlSumme.jsp`
 2. Schreiben der AjaxAnwendung **Addierer.html**
Sobald die 2. Zahl eingegeben wurde, soll das Summenergebnis (vom Server berechnet) erscheinen.



Safari (2015)

1. Summe.jsp

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<%                               // Deklaration eines Scriptlets
    response.setContentType("text/xml"); //Setzen des MIME-Typs XML
    response.setHeader("Cache-Control", "no-cache"); //Ausschalten des Ladens aus dem Browser -Cache
    String zahl1 = request.getParameter( "zahl1" ); //Auslesen von „zahl1“ aus Parameterstring
    String zahl2 = request.getParameter( "zahl2" ); //Auslesen von „zahl2“ aus Parameterstring
    String summe = "-";
    long longsumme;

    if( null == zahl2 || 0 >= zahl2.trim().length() ) { // Entfernen von Leer- und Kontrollzeichen mit trim()
        zahl2 = "-";
    } else {
        try {
            long i1 = Long.parseLong( zahl1 );
            long i2 = Long.parseLong( zahl2 );
            longsumme = i1 + i2;
            summe = "" +longsumme;
        } catch( Exception ex ) {}
    }
%>
<MeinErgebnis>
    <summe><%= summe %></summe>      // Schreiben des XML-Ergebnisdokumentes
</MeinErgebnis>
```

2. Addierer.html

'< body ...>' -HTML-Tag mit `bodyonload 'meinAjaxInit()'`
-> wird vom Webbrowser zum Beginn der Webseite aufgerufen.

'meinAjaxInit()'-Funktion:

- legt 'XMLHttpRequest'-Objekt an
- Aufruf im Formular mit `'onKeyUp="meinAjaxAufruf ()"`
-> nach jedem eingegebenen Zeichen erfolgt eine Ajax-Kommunikation

URL zum Ajax-Dienst:

- ins Formular eingegebene Zahlen werden als Parameter übergeben

Callback-Funktion 'meineCallbackFkt ()'

- Ergebniswerte als XML-Struktur per `'req.responseXML'`

Auswertung des XML-Dokuments mit DOM

- Extrahieren der Summe mit `'getElementsByTagName ()'`,
- Einsetzen der Summe in die HTML-Webseite mit
`'getElementById () . value'`

2. Addierer.html : Grundgerüst

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>AJAX</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<script language="JavaScript" type="text/javascript">
    var url0 = "http://localhost:8080/LearnAjax_Pr/AjaxExamples/Summe.jsp";
    var req;
    function meinAjaxInit() {...}
    function meinAjaxAufruf()
    {
        ...
        req.onreadystatechange = meineCallbackFkt;
        ...
    }
    function meineCallbackFkt() {....}
</script>
</head>
<body onload="meinAjaxInit()">
<h2>Ajax - Addierer</h2>
    ...
    [FormulareingabeEvent]="meinAjaxAufruf();"><br>
    ...
</body>
</html>
```

Addierer.html: Function meinAjaxInit()

```
function meinAjaxInit() {  
    try {  
        if( window.XMLHttpRequest ) {  
            req = new XMLHttpRequest();  
        } else if( window.ActiveXObject ) {  
            req = new ActiveXObject( "Microsoft.XMLHTTP" );  
        } else {  
            alert( "Ihr Webbrowser unterstuetzt leider kein Ajax!" );  
        }  
        } catch( e ) {  
            alert( "Fehler: " + e );  
        }  
    }  
}
```

Anlegen des XMLHttpRequest- Objektes

Addierer.html: Function meinAjaxAufruf

```
function meinAjaxAufruf() {  
    if( req ) {  
        var zahl1Eingabe = document.getElementById( "zahl1Eingabe" );  
        var zahl2Eingabe = document.getElementById( "zahl2Eingabe" );  
        var url = url0 + "?zahl1=" + escape( zahl1Eingabe.value )  
            + "&zahl2=" + escape( zahl2Eingabe.value );  
        req.open( "GET", url, true );  
        req.onreadystatechange = meineCallbackFkt;  
        req.send( null );  
    }  
}
```

- Auswerten des Formulars aus dem html –body
- Zahlen, welche eingegeben wurden, werden an die URL als Übergabeparameter des HTTP-GET angehängt
- Aufruf der CallBackFunktion zur Auswertung des Responses

Addierer.html: Function meineCallbackFkt()

```
function meineCallbackFkt() {
    if( 4 == req.readyState ) {
        if( 200 != req.status ) {
            alert( "Fehler " + req.status + ": " + req.statusText );
        } else {
            ergebnis = req.responseXML.documentElement;
            var summeAusgabe = ergebnis.getElementsByTagName('summe')
[0].firstChild.data;
            document.getElementById("summeAusgabe").value = summeAusgabe;
        }
    }
}
</script>
</head>
```

- Extrahieren der Summe aus der XML-Response des Servers
- Einsetzen der Summe in die HTML-Webseite

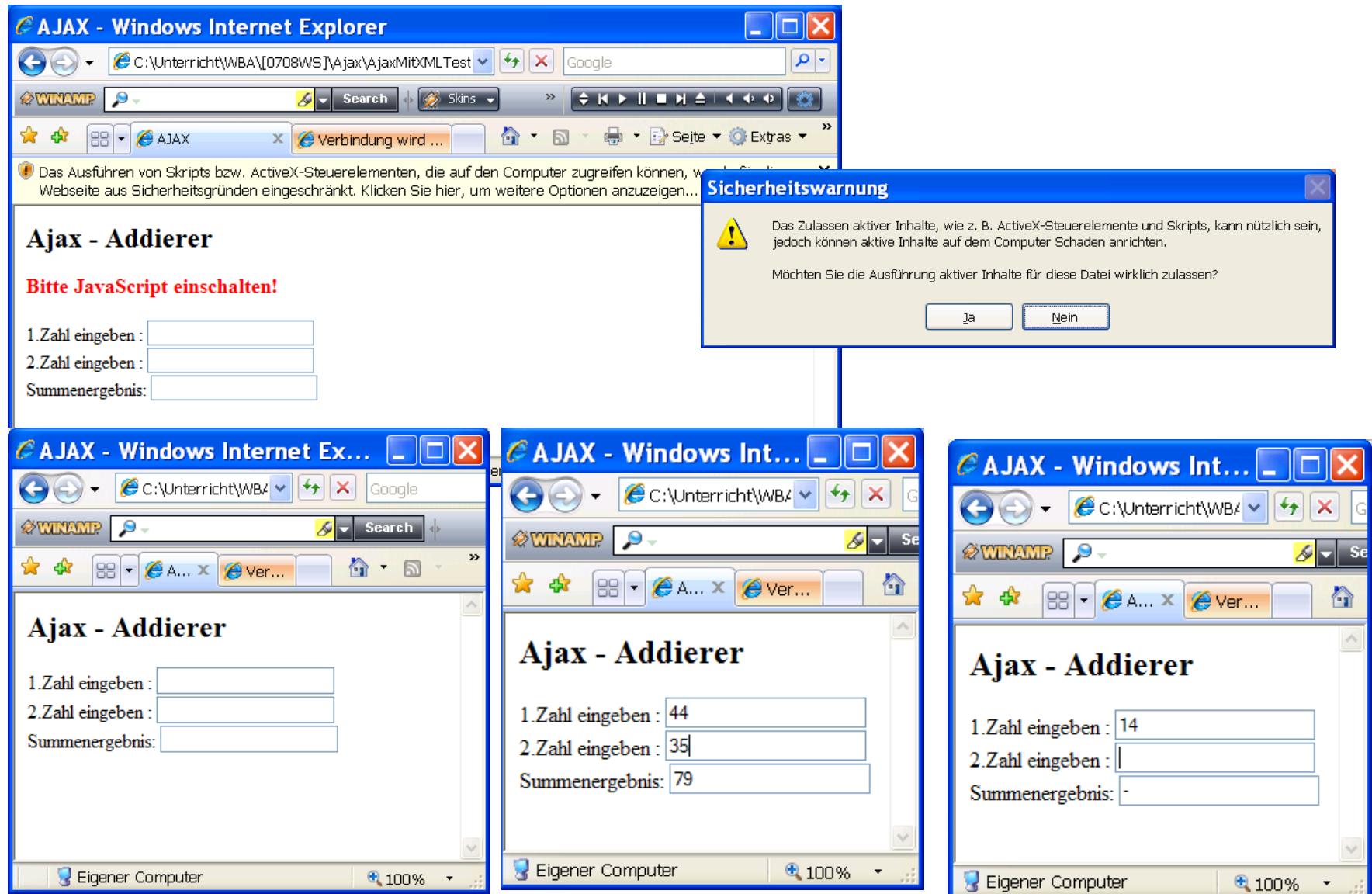
Addierer.html: Das Formular

```
<body onload="meinAjaxInit()">
  <h2>Ajax - Addierer</h2>
  <noscript>
    <font color="red"><big><b>Bitte JavaScript einschalten!</b></big></font><br>
  </noscript>
  <form action="#" method="GET" name="AddiererFormular">
    1.Zahl eingeben :
    <input type="text" name="zahl1Eingabe" id="zahl1Eingabe"><br>
    2.Zahl eingeben :
    <input type="text" name="zahl2Eingabe" id="zahl2Eingabe" onKeyUp="meinAjaxAufruf();"><br>
    Summenergebnis:
    <input type="text" name="summeAusgabe" id="summeAusgabe" readonly="true">
  </form>
</body>
</html>
```

- 'meinAjaxInit()' wird vom Webbrowser zum Beginn der Webseite aufgerufen.
- nach jedem eingegebenen Zeichen erfolgt eine Ajax-Kommunikation

Formular AddiererFormular wird nie versendet: **action="#"**

XML Dateien auswerten



Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
- 8. Bibliotheken und Frameworks**
9. JQuery

Ajax – Bibliotheken und Frameworks

- Vorteil: JavaScript – Bibliotheken und –Frameworks sind größtenteils kostenlos
- Typische Aufgaben von JavaScript – Bibliotheken sind:
 - komfortable Schnittstellen zur Ajax-Technik mit vorgefertigten Browserweichen und browserspezifischen Sicherheitskonzepten
 - Manipulation der Dateien mittels DOM
 - visuelle Effekte
 - > Arbeit mit Ajax kann mit Bibliotheken stark vereinfacht werden
- zahlreiche Bibliotheken und Frameworks unter <http://www.ajaxian.com/resources>
- Einarbeitungszeit in z.T. komplexe Frameworks darf nicht unterschätzt werden
- Aufwand bei der Einarbeitung sollte im Verhältnis zur gestellten Aufgabe stehen

• Framework Prototype:

- frei verfügbar unter <http://prototype.conio.net>
- Schnittstelle zur Ajax-Technologie (ajax.js)
- OOP-Features (u.a. base.js)
- Methoden für die Arbeit mit dem DOM (dom.js)
- Methoden für die Arbeit mit Formularen (form.js)
- nützliche Funktionen (z.B. string.js)
- Einbinden wie normale Bibliotheksfunktion in JavaScript:
`<script src="prototype.js" type="text/javascript"></script>`

Ajax – Bibliotheken und Frameworks

• Framework script.aculo.us :

- verfügbar unter <http://script.aculo.us>
- verwendet prototype als basis
- definiert spektakuläre visuelle Effekte wie z.B.:
 - Drag & Drop (Bewegen einzelner Elemente der Webseite mithilfe der Maus)
 - Beeinflussung der Transparenz von Elementen
 - Bewegung der Elemente entlang von x/y-Koordinaten
 - gleichzeitige Kombination von verschiedenen Effekten
 - „Highlighting“ - Blinken des Hintergrundes eines Elements
 - automatisches Skalieren von Elementen
 - unsichtbar und sichtbar Machen von Elementen
 - und viele andere...
- Einbinden der Reihe nach, erst Prototype, dann script.aculo.us:

```
<script src="prototype.js" type="text/javascript"></script>
<script src="src/script.aculo.us.js" type="text/javascript"></script>
```

Clientseitige Implementierungs-technologien (III): Ajax

1. Einführung
2. Beispielanwendungen
3. Asynchron oder synchron?
4. Das XMLHttpRequest-Objekt
 1. Erzeugung
 2. Methoden
 3. Eigenschaften
5. Erstes Beispiel (Hallo Ajax)
6. Nutzung externer Quellen
7. XML-Dateien auswerten
8. Bibliotheken und Frameworks
- 9. JQuery**

Ajax – Bibliotheken und Frameworks

- JQuery:

- verfügbar unter <http://jquery.com>
- (Download + Tutorials in englisch)
- Herunterladen und in Webapplikationsverzeichnis kopieren
z.B. aktuell (April 2015) die Datei: [jquery-2.1.0.js](#)
- Einbinden in die HTML-Seite mit:

```
<script src="jquery-2.1.0.js" type="text/javascript"></script>
```

- Testen:

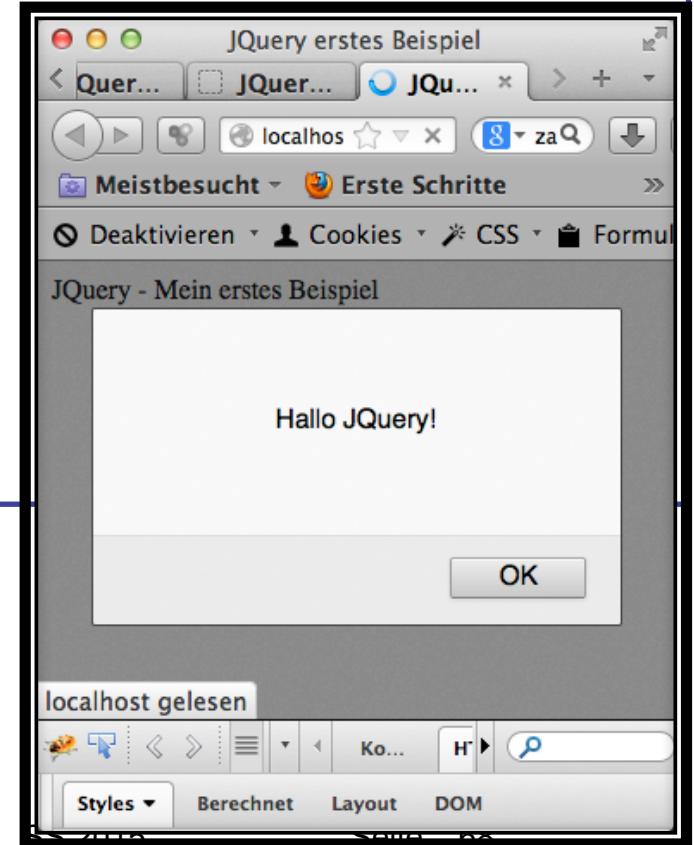
```
<script>
window.onload = function()
{
  alert( "welcome" );
}
</script>
```



JQuery: erstes Beispiel



```
<!DOCTYPE html>
<html>
  <head>
    <title>JQuery erstes Beispiel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <script type="text/javascript" src="jquery-2.1.0.js"></script>
  </head>
  <body>
    <div>JQuery - Mein erstes Beispiel</div>
    <script>
      window.onload = function()
      {
        alert( "Hallo JQuery!" );
      };
    </script>
  </body>
</html>
```



JQuery: ein paar Vorteile



- Verkürzte Schreibweise und vereinheitlichter Zugriff auf DOM-Elemente
- Kompensation browserabhängiger Besonderheiten
 - In der Regel auf allen offiziell unterstützten Browsern getestet
 - <http://jquery.com/browser-support/> (April 2015)
 - MSIE 9+, Google-Chrome (Current-1), Firefox (Current-1), Safari 5.1+
 - Opera 12.2 (Current-1), iOS 6.0+, Android 4.0+

The screenshot shows a web browser window displaying the jQuery website's 'Browser Support' page. The page features the jQuery logo and tagline 'write less, do more.' at the top. Below the logo, there is a navigation bar with links for 'Download', 'API Documentation', 'Blog', 'Plugins', 'Browser Support' (which is highlighted in blue), and a search bar. The main content area is titled 'Browser Support' and contains a table showing current active support for various browsers and platforms. The table includes columns for Internet Explorer, Chrome, Firefox, Safari, Opera, iOS, and Android. It lists specific versions for each browser, such as '6+' for Internet Explorer and '12.1x,' for iOS.

	Internet Explorer	Chrome	Firefox	Safari	Opera	iOS	Android
jQuery 1.x	6+	(Current - 1) or Current	(Current - 1) or Current	5.1+	12.1x, (Current - 1) or Current	6.0+	4.0+
jQuery 2.x	9+						

JQuery: DOM-Zugriff und Schutz



- Schutz des DOM-Baumes
 - mit Methode `ready()`
 - Stellt sicher, dass Webseite vollständig geladen ist, bevor Manipulationen per javascript/ jQuery beginnen.
 - Ereignisbehandlung ähnlich zu Event `onload()`
 - `ready(function () { . . . });`
- DOM-Zugriff in Kurzschreibweise
 - mit der Methode `$()`
 - referenziert ein Element der Webseite über eine id oder einen Selektor
 - z.B. `$(document)`

Angabe bei jeder jQuery-DOM-Manipulation empfohlen:

```
$(document).ready(function () { . . . });
```

JQuery: DOM-Zugriff und Schutz



```
<title>Schutz des DOM</title>
<link href="DOM-Schutz.css" rel="stylesheet" type="text/css" />
<script src="http://code.jquery.com/jquery-1.10.1.min.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function () {
  $("#a").click(function () {
    $("#ausgabe").html("Du hast Button 1 geklickt"));
  });
  $("#b").click(function () {
    $("#ausgabe").html("Du hast Button 2 geklickt"));
  });
  $("#c").click(function () {
    $("#ausgabe").html("Du hast Button 3 geklickt"));
  });
}
</script>
</head>
<body>


## JQuery schützt das Document Object Model



### Klicke die Buttons 1, 2 oder 3!


<button id="a">Button 1</button>
<button id="b">Button 2</button>
<button id="c">Button 3</button>
<div id="ausgabe"></div>
</body>
</html>
```

Referenz auf
jQuery-Bibliothek

Ansprechen der Webseite mit `$(document)`
Ganze Webseite geladen? -> Methode `ready()`

Ereignisbehandlungs Routinen
für Button 1 bis 3 jeweils mit
`click(function()
{ ... });`
einzigartige id kennzeichnet
Buttons

Methode `html()`
Ähnlich zu Javascript -
Methode `innerHTML()`

JQuery: DOM-Zugriff und Schutz



Standardmethoden zum Umgang mit Ajax in jQuery



- **Nachladen von Serverdaten mit den Methoden `$.get()` und `$.post()`**

```
jQuery.get(Url, [Daten], [sucess(Daten, textStatus, jqXHR)], [Datentyp])
jQuery.post(Url, [Daten], [sucess(Daten, textStatus, jqXHR)], [Datentyp])
```

Url – ist einzig obligatorischer Parameter

Daten – Objekt vom Typ Map oder String (zum Server senden)

[sucess(Daten, textStatus, jqXHR)]

- Callback-Funktion : wird im Erfolgsfall aufgerufen
- Daten vom Server
- textStatus – Serverstatus
- jqXHR
 - **jQuery XMLHttpRequest** – Objekt ist Rückgabewert der Ajax-Aktivitäten
 - Enthält Erweiterungen und alle klassischen Methoden und Eigenschaften des XMLHttpRequest außer onreadystatechange

Datentyp – `xml`, `text`, `json`, `jsonp`, `script` oder `html`

Beispiel : Ajax mit jQuery I/II



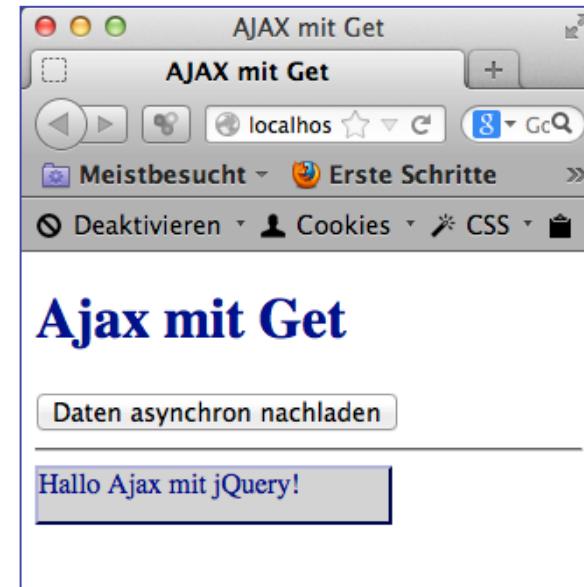
- Klartext aus Datei vom Server anfordern mit den Methode `$.get()`

Die Datei `helloAjax.txt` enthält:

`Hallo Ajax mit jQuery!`

Anfordern über normalen Buttons (kein submit, sonst kein Ajax!)

- per get mit `text()` - Methode des jqXHR-Objektes



Nach Klick auf den Button, wird der Text aus der Datei `helloAjax` von `localhost` geladen und in das Textausgabefeld eingegeben (DOM-Manipulation).

Beispiel : Ajax mit jQuery II/II



```
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <title>AJAX mit Get</title>
    <link rel="stylesheet" type="text/css" href="AjaxMitGet.css"/>
    <script src="http://code.jquery.com/jquery-1.10.1.min.js"
        type="text/javascript"></script>
    <script type="text/javascript" src="AjaxMitGet.js"></script>
</head>
<body>
    <h1 class="headline">Ajax mit Get</h1>
    <button>
        Daten asynchron nachladen
    </button>
    <hr/>
    <div id="antwort" class="klasse"></div>
</body>
</html>
```

AjaxMitGet.html

AjaxMitGet.js

```
$(function() {
    $("button").click(function(){
        $.get("helloAjax.txt", function(data) {
            $("#antwort").text(data);
        });
    });
});
```

Referenz auf
javascript-Datei **nach**
Einbinden von jQuery

Einfacher
Button

Button mit
Eventmethode
click()
gebunden

get-methode
von jQuery

Daten vom Server (**data**)
werden in Texfeld eingefügt
(partielle DOM-Manipulation).

Beispiel: Ajax per get und post I/II



- Klartext aus Datei vom Server anfordern mit den Methoden `$.get()` und `$.post()`

Die Datei `ajax.txt` enthält:

Mit `<u>Get oder Post</u>`wurden diese Daten vom Server nachgefordert.

Anfordern über normale Buttons

- per get mit `html()`-Methode
- per post mit `text()`- Methode

? Was passiert?

Der Inhalt der Datei wird vom Browser einmal als Text dargestellt und einmal als html interpretiert.

A screenshot of a Mac OS X desktop environment showing a web browser window titled "AJAX". The browser has three tabs open, all labeled "AJAX". The main content area displays two identical text boxes side-by-side. Both text boxes contain the same text: "Mit Get oder Post wurden diese Daten vom Server nachgefordert." The top text box is enclosed in a light gray box, while the bottom one is enclosed in a darker gray box. Below the text boxes are two buttons: "Ajax mit 'jQuery.GET()'" and "Ajax mit 'jQuery.POST()'".

Daten asynchron nachladen

Ajax mit "jQuery.GET()" Ajax mit "jQuery.POST()"

Mit Get oder Post wurden diese Daten vom Server nachgefordert.

Mit Get oder Post wurden diese Daten vom Server nachgefordert.

Beispiel Ajax per get und post II/II



- Quellcode der html-Datei:

```
<!DOCTYPE ...>  
...  
<script type="text/javascript" src="DatenMitGetUndPost.js"></script>  
</head>  
<body>  
    <h1 class="headline">Daten asynchron nachladen </h1>  
    <button> Ajax mit "jQuery.GET()" </button>  
    <button> Ajax mit "jQuery.POST()" </button>  
    <hr/>  
    <div id="antwort1" class="k1"></div>  
    <div id="antwort2" class="k1"></div>  
</body>  
</html>
```

Buttons in Array
haben z.B.
Eigenschaften **first**,
last, **eq(number)**

Nutzen der
Methode
html() von
jQHXR

- Quellcode der javaScript-Datei DatenMitGetUndPost.js:

```
$(function () {  
    $("button:first").click(function () {  
        $.get("ajax.txt", function (data) {$("#antwort1").html(data);  
    });  
});  
    $("button:eq(1)").click(function () {  
        $.post("ajax.txt", function (data) {$("#antwort2").text(data);  
    });  
});  
});
```

Nutzen der Methode
text() von jQHXR

Zusammenfassung

Wir haben die Ajax-Technologie von „Grund auf“ und ohne doppelten Boden kennen gelernt.

- JavaScript und DOM als Grundlage
- Anfordern einfacher Textdaten
- Anfordern und Weiterverarbeiten von XML-Daten

Es gibt JavaScript - Frameworks, die bei der Anwendung der Ajax-Technologie unterstützen.

Literatur für Ajax und JQuery

1. Stefan Mintert, Christoph Leisegang: „Ajax: Grundlagen, Frameworks und Praxislösungen.“, dpunkt.verlag GmbH Heidelberg 2007, ISBN-10:3-89864-404-9
2. Johannes Gamperl: „Ajax Grundlagen“, Herdt- Verlag Bildungsmedien GmbH, Bodenheim 2006,
<http://www.herd4you.de>
3. Ralph Steyer: „**Jquery – Das universelle Javascript- Framework für das interaktive WEB und mobile Anwendungen**“, Hanser-Verlag, München 2014 , ISBN: 978-3-446-43941-2



Links und Quellen

- Jesse James Garrett „Ajax:A new Approach to Web Applications“
<http://www.adaptivepath.com/ideas/essays/archives/000385.php>
- www.netwipes.com
- www.flickr.com
- <http://ajaxpatterns.org>
- http://www.galileocomputing.de/openbook/javascript_ajax/
- <http://www.torsten-horn.de/techdocs/javascript-ajax.htm>
- <http://www.ajaxian.com/resources>
- <http://prototype.conio.net>
- <http://script.aculo.us>

Ausblick

- **Clientseitige Implementierungstechnologien:**
Javascript, DOM, Ajax, Jquery, **(Java-Applet kurz)**
- **Zusammenfassung Clienttechnologien**
- **Einführung Servertechnologien**