

Webbasierte Anwendungen

AJAX und jQuery

Prof. Dr. Ludger Martin

Gliederung

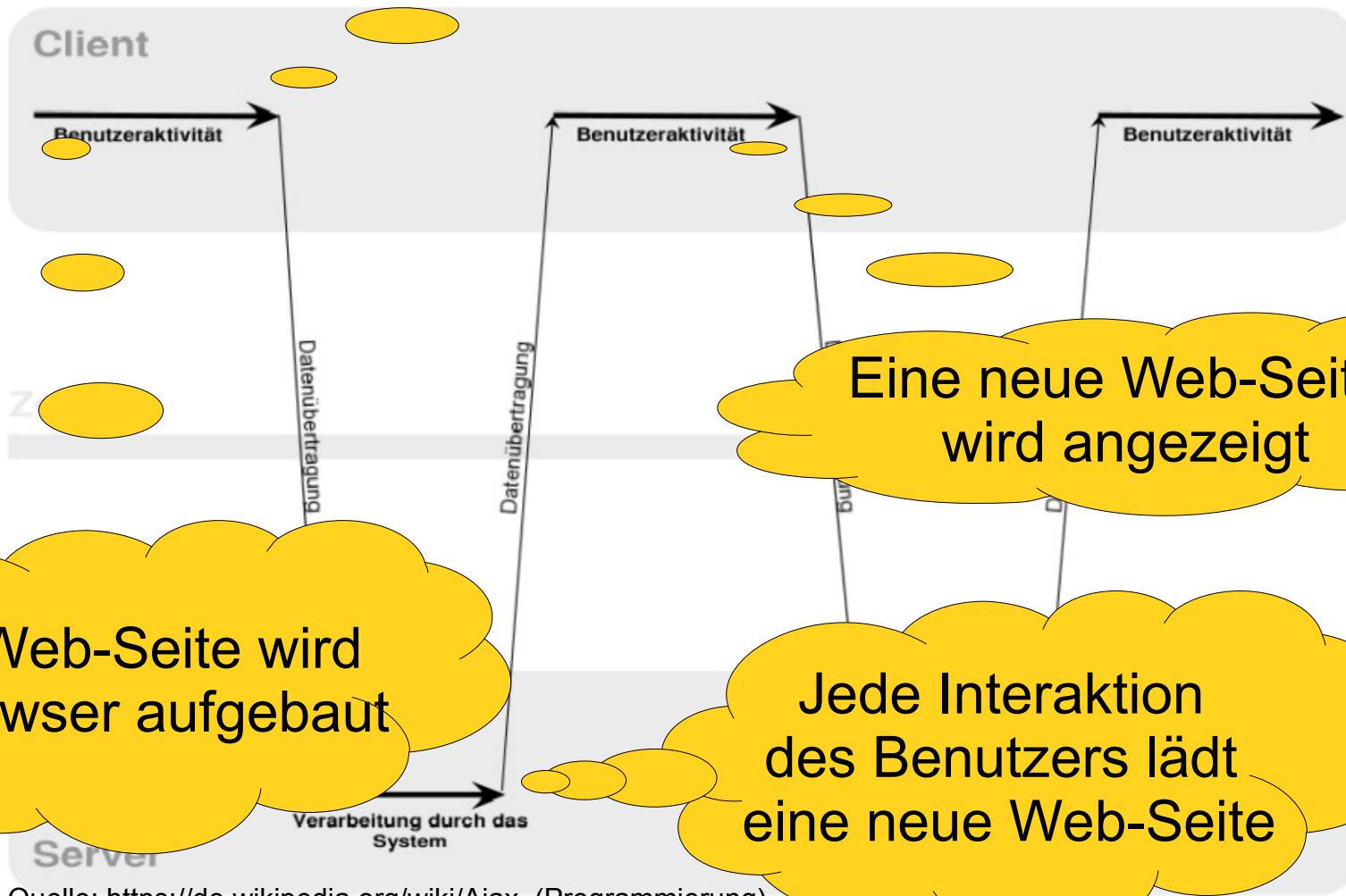
- ★ Einführung
- ★ AJAX-Programmierung
- ★ Bibliothek jQuery
- ★ Bibliothek jQuery (AJAX)

Einführung

- ★ AJAX: **A**ynchronous **J**avaScript **a**nd **X**ML
- ★ Web Dokumente in HTML und CSS
- ★ DOM für Anzeige und Interaktion
- ★ Asynchroner Datenaustausch per XMLHttpRequest (Dies ist das Neue an der ganzen Technik)
- ★ Ausgiebige Nutzung von JavaScript

Sie wird angeschaut und
es werden ggf.
Formularfelder ausgefüllt

Der klassische Web Seiten Zyklus:



Eine neue Web-Seite
wird angezeigt

Eine Web-Seite wird
im Browser aufgebaut

Jede Interaktion
des Benutzers lädt
eine neue Web-Seite

Quelle: [https://de.wikipedia.org/wiki/Ajax_\(Programmierung\)](https://de.wikipedia.org/wiki/Ajax_(Programmierung))

Einführung

Der klassische Web-Seiten Zyklus:

- ★ Eine Web-Seite wird im Browser aufgebaut.
- ★ Sie wird angeschaut und es werden ggf. Formularfelder ausgefüllt.
- ★ Jede Interaktion des Benutzers lädt eine neue Web-Seite.
- ★ Eine geänderte Web-Seite wird angezeigt.
- ★ usw.

Einführung

Von normalen Desktop-Anwendungen sind wir gewohnt:

- ★ Eine Benutzungsoberfläche wird geladen.
- ★ Ein Benutzer tätigt seine Eingaben.
- ★ Gleichzeitig reagiert die Anwendung auf die Eingaben, ohne dass eine Blockade stattfindet.
- ★ usw.

Einführung

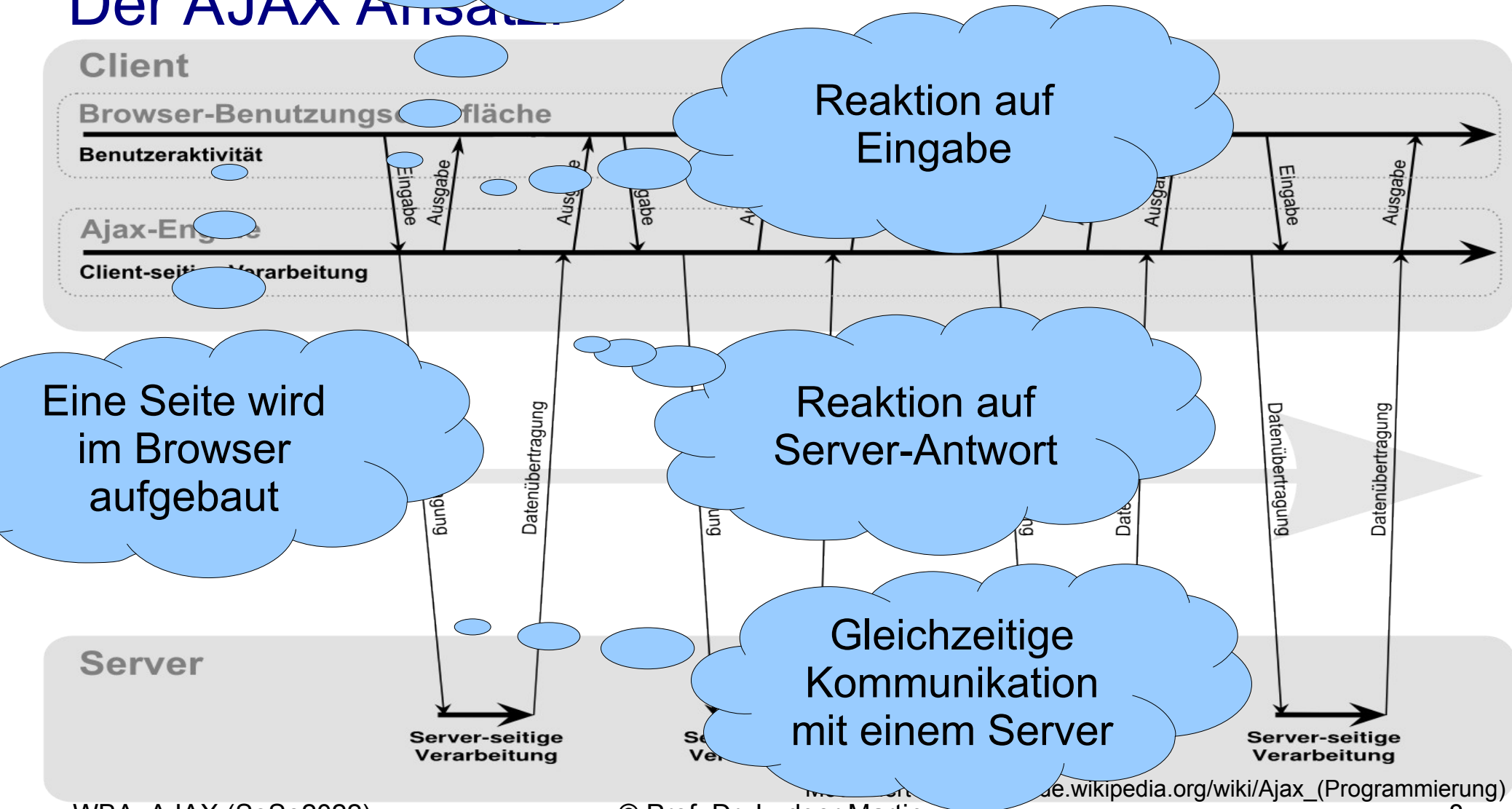
Der AJAX Ansatz:

- ★ Eine Seite wird im Browser aufgebaut.
- ★ Sie wird angeschaut und es werden ggf. Formularfelder ausgefüllt.
- ★ Gleichzeitige Kommunikation mit einem Server und dem darauf installierten Service.
- ★ Die Benutzungsoberfläche wird dynamisch angepasst.
- ★ usw.

ung

z.B.
Formularfelder
ausgefüllt

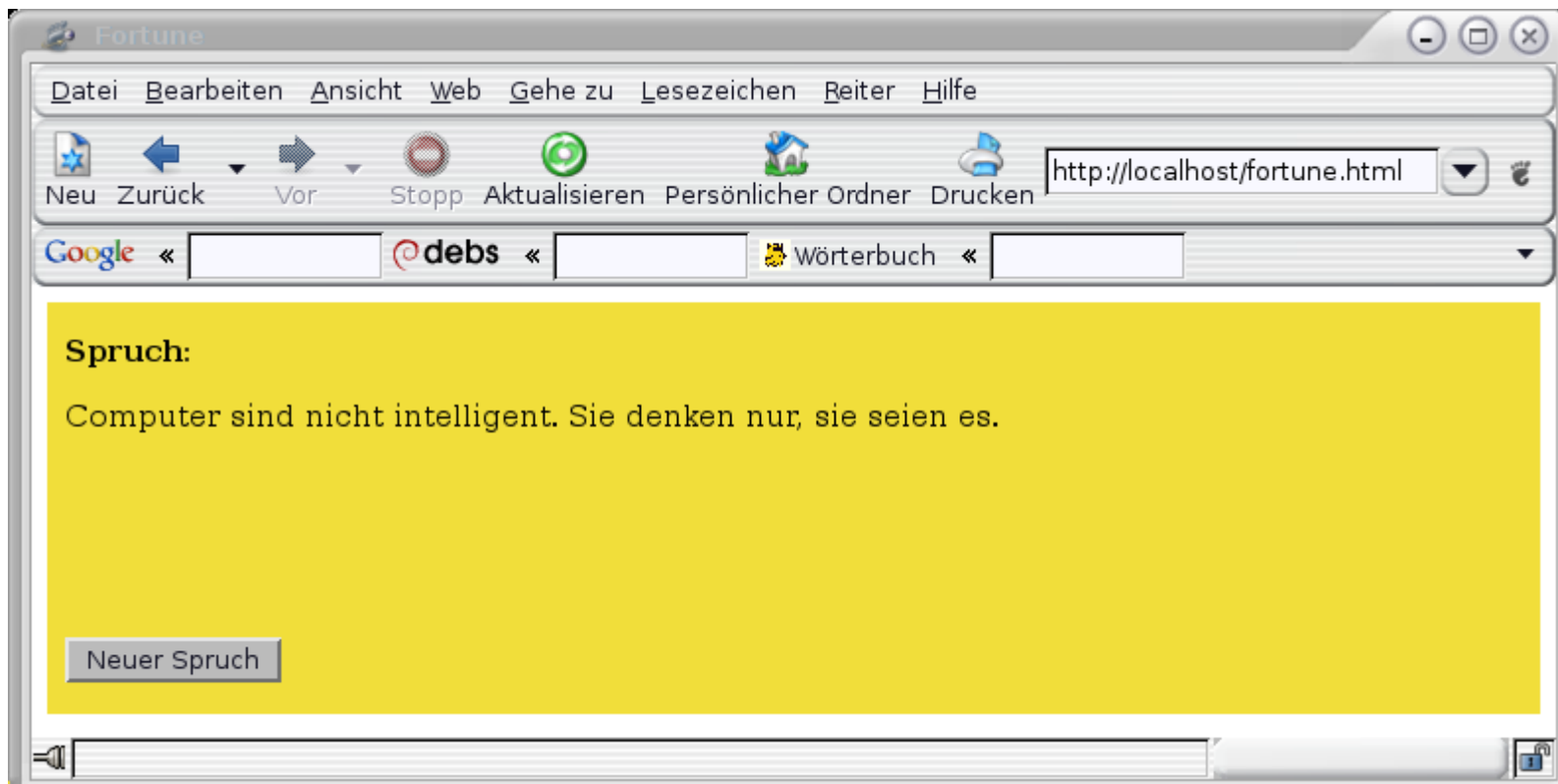
Der AJAX Ansatz



Einführung

★ AJAX Beispiel:

- ★ Laden eines Spruchs auf einer Web-Seite, ohne dass sie neu geladen werden muss.



Einführung

- ★ Lässt sich ohne viel Aufwand mit einem reinen JavaScript Programm lösen:
 - ★ Array mit allen Sprüchen
 - ★ Auf Knopfdruck wird eine JavaScript Funktion aufgerufen, die einen neuen Spruch aus dem Array sucht und darstellt.
- ➡ **Nachteil:** Alle Sprüche müssen in der JavaScript Datei übertragen werden, was u.U. lange Ladezeiten mit sich bringt!

Einführung

- ★ Lässt sich ohne viel Aufwand mit einem reinen JavaScript Programm lösen:
 - ★ Array mit allen Sprüchen
 - ★ Auf Knopfdruck
sucht man im Array
nach dem Spruch, den man
haben möchte.
Nachdem man den Spruch
gefunden hat, wird er in
JavaScript in HTML umgewandelt, was u.U.
lange Laufzeiten mit sich bringt!

Ein AJAX-Programm lädt auf
Anfrage nur einen Spruch
von einem Service!

Einführung

★ Vorteile:

- ★ Es wird keine neue Seite geladen, sondern die bestehende Seite wird verändert.
- ★ Auslöser für das Ändern oder Neuladen von Seiten sind nicht nur Mausklicks, sondern beliebige Ereignisse.
- ★ Die Kommunikation zwischen Client und Server läuft asynchron ab.
- ★ Die Steuerung wird mittels JavaScript auf dem Client realisiert.
- ★ Die Nachrichten zwischen Client und Server sind im XML-Format.

Einführung

Neues Paradigma:

- ★ Eine AJAX-Anwendung hat nichts mit dem herkömmlichen Web mit verknüpften Seiten zu tun.
- ★ Eine AJAX-Anwendung läuft auf einem Server und das Web ist die grafische Benutzungsoberfläche auf dem Client.
- ★ AJAX stellt die Verbindung zwischen Benutzungsoberfläche und Kern-Anwendung dar.
- ★ Für AJAX existieren viele Bibliotheken, die dem Programmierer die Arbeit leichter machten.

Einführung

Was benötigt AJAX?

AJAX ist eine Zusammenstellung von verschiedenen Techniken:

- ★ JavaScript
- ★ DOM (Document Object Model)
- ★ XML (oder JSON)

AJAX-Programmierung

- ★ Das native AJAX besteht nur aus einem einzigen Objekt, das mit JavaScript angesprochen wird.
- ★ Probleme bilden dabei wieder unterschiedliche Varianten von JavaScript in den Browsern.
- ★ Das XMLHttpRequest Objekt erlaubt:
 - ★ HTTP/HTTPS Anfrage
 - ★ Abfrage der Antwort
 - ★ Abbrechen der Anfrage

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

1. Request-Objekt ausprägen

- ★ Für alle Browser außer IE < Version 7

```
new XMLHttpRequest ( )
```

- ★ Für IE Version 5 und 6

```
new ActiveXObject ("Msxml2.XMLHTTP")
```

- ★ IE mit alten System Bibliotheken

```
new ActiveXObject ("Microsoft.XMLHTTP")
```

Beispiel:

```
let request = new XMLHttpRequest ( ) ;
```


AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

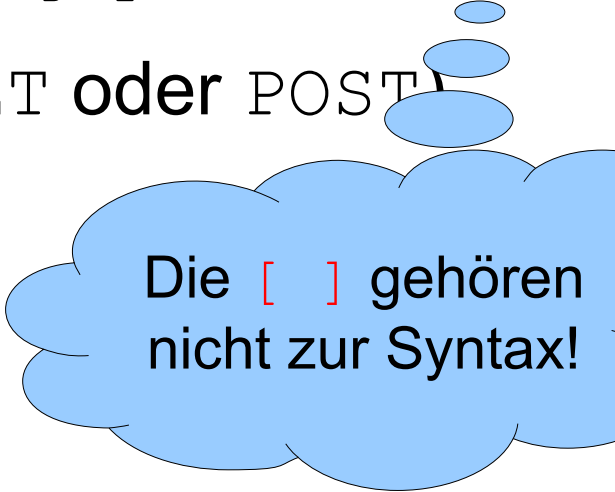
2. Request initialisieren

```
void XMLHttpRequest::open(String method,  
                          String url, boolean async [,  
                          String username, String password])
```

★ method: **Zu benutzende Methode (GET oder POST)**

★ async: **asynchron oder synchron**

★ username, password: **optional, falls Authentifizierung notwendig**



Die [] gehören nicht zur Syntax!

Beispiel:

```
request.open("get", "fortune_x.php", true);
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

3. Für asynchrone Kommunikation eine Event-Funktion dem Event-Handler

`onreadystatechange` zuweisen.

Beispiel:

```
request.onreadystatechange = zeigeSpruch;
```

4. Anfrage stellen

```
void XMLHttpRequest::send(Object body)
```

★ **body:** Wenn die Methode `POST` spezifiziert ist, enthält der `body` eine Zeichenkette bzw. ein XML-Dokument, andernfalls `null`.

Beispiel:

```
request.send(null);
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

5. Event-Handler-Funktion implementieren:

- a) Die Eigenschaft `readyState` auf erfolgreiche Ausführung abfragen

`readonly short XMLHttpRequest::readyState`

Zustand	Name	Beschreibung
0	UNSENT	Dies ist der Anfangszustand. Das <code>XMLHttpRequest</code> Objekt wurde ausgeprägt oder mit <code>abort()</code> zurück gesetzt.
1	OPENED	Die Verbindung ist mit <code>open()</code> geöffnet, aber die Methode <code>send()</code> ist noch nicht aufgerufen worden.
2	HEADERS RECEIVED	Die Methode <code>send()</code> wurde aufgerufen und die HTTP(S) Anfrage wurde gesendet. Eine Antwort ist aber noch nicht eingetroffen.
3	LOADING	Die HTTP(S) Antwort wird gerade empfangen. Der <code>body</code> ist aber noch nicht vollständig.
4	DONE	Die HTTP(S) Antwort ist vollständig empfangen.

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

5. Event-Handler-Funktion implementieren:

b) Die Eigenschaft `status` abfragen

```
readonly short XMLHttpRequest::status
```

Der HTTP-Statuscode des Servers wird in dieser Variable zurückgegeben. 200 steht z.B. für erfolgreich und 404 für Dokument nicht gefunden. Diese Eigenschaft darf nur gelesen werden, wenn `readyState >= 2` ist.

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

5. Event-Handler-Funktion implementieren:

c) Die Antwort kann mit der Eigenschaft

```
readonly String  
XMLHttpRequest::responseText
```

abgefragt werden, solange `readyState >= 3` **ist**
und mit

```
readonly Document  
XMLHttpRequest::responseXML,
```

wenn es sich um ein XML-Dokument handelt und
`readyState === 4` **ist.**

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (HTML-Teil):

```
<div id="fortune" >
  <p><b>Spruch:</b></p>
  <!-- Spruch wird im div ausgetauscht -->
  <div id="fortuneMessage">&nbsp;</div>
  <form action="">
    <!-- Event zum Auslösen der Aktion -->
    <p><input type="button"
      value="Neuer Spruch"
      onclick="ladeSpruch();" />
    </p>
  </form>
</div>
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (JS-Teil 1):

```
// HTTP-Request Objekt ausprägen
let request = new XMLHttpRequest();

// Event handler
function zeigeSpruch() {
    // Prüfe, dass Kommunikation beendet
    if (request.readyState === 4) {
        // Prüfe Server Status-Code: Erfolgreich
        if (request.status === 200) {
            // XML Antwort abfragen
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (JS-Teil 2):

```
// XML Antwort abfragen
let xml = request.responseXML;
let spruch = xml.getElementsByTagName
    ('spruch')[0].textContent;
// Text in HTML setzen
document.getElementById
    ('fortuneMessage').firstChild.
        nodeValue = spruch;

    }
}
}
```


AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (JS-Teil 3):

```
// Eventhandler zuweisen
request.onreadystatechange = zeigeSpruch;

// Event-Funktion
function ladeSpruch() {
    // URL für Anfrage setzen
    request.open("get", "fortune_x.php", true);
    // Anfrage ausführen
    request.send(null);
}
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (PHP-Service Teil 1):

```
<?php
header("Content-Type: text/xml");
// XML Kopf setzen
$xw = new xmlWriter();
$xw->openMemory();
$xw->startDocument('1.0', 'UTF-8');

// Rufe das Programm fortune auf
$fp=popen('/usr/games/fortune -u
          /usr/share/games/fortunes/de 2>&1', "r");
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (PHP-Service Teil 2):

```
// Lese die Ausgaben von fortune
$results = '';
while (!feof($fp)) {
    $results .= fgets($fp, 4096);
} fclose($fp);
// XML ausgeben
$xml->startElement('spruch');
// Konvertiere nach HTML und gib dies aus
$xml->text(htmlspecialchars($results));
$xml->endElement();
// Ausgabe
print $xml->outputMemory(true);
```

AJAX-Programmierung

Benutzen von XMLHttpRequest Objekt

★ Beispiel (PHP-Service Teil 2):

Aufruf fortune_x.php:

```
/ $
W <?xml version="1.0" encoding="UTF-8"?>
  <spruch>
    Drei Mitarbeiter eines Softwarehauses sind mit dem Auto
  } unterwegs. Auf einer Gebirgsstraße versagen die Bremsen,
  / der Wagen stürzt einen Abhang runter und landet in einem
  / Bach. Was tun?
$ Der Marketing-Manager: &quot;Wir benennen das Problem,
  / formulieren eine Lösung und nähern uns so der
  / Problemlösung.&quot;;
$ Der Leiter der Hotline: &quot;Wir rufen einen Techniker,
$ der die Bremse ersetzt.&quot;;
  / Der Software-Entwickler: &quot;Unsinn, wir schieben den
  / Wagen auf die Straße zurück fahren weiter und schauen erst
  / mal, ob sich der Vorfall wiederholt.&quot;;
  </spruch>
```

AJAX-Programmierung

Asynchronität

- ★ Mehrere Anfragen können mittels JavaScript parallel abgesetzt werden.
- ★ Es ist nicht vorhersehbar, in welcher Reihenfolge diese antworten.
- ★ Sind Abhängigkeiten im Programm notwendig, so müssen diese per Hand geprüft werden:
 - ★ Setzen von Flags
 - ★ Abfragen von `readyState` einer vorhergehenden Anfrage
 - ★ Verwenden von synchronen Anfragen

AJAX-Programmierung

Asynchronität

- ★ Eventuell ist es notwendig eine frühere Anfrage abubrechen, die noch nicht geantwortet hat:

```
void XMLHttpRequest::abort ( )
```

- ★ Beispiel:

```
if (request &&  
    (request.readyState === 2 ||  
     request.readyState === 3)) {  
    request.abort();  
}
```

AJAX-Programmierung

Timeout

★ Unterschiedliche Timeouts:

★ Timeout durch Benutzerinaktivität

Das Problem wird durch das Verwerfen einer Session verursacht, wenn keine weiteren Anfragen getätigt werden. Vermieden werden kann dies durch Senden von unbemerkten Anfragen mittels `window.`

`setInterval()` oder `window.setTimeout()`

★ Der Server bricht die Anfrage ab, da sie zu lange dauert.

Je nach Server wird in der `status` Eigenschaft angezeigt, ob das Skript erfolgreich ausgeführt wurde.

AJAX-Programmierung

Timeout

- ★ Unterschiedliche Timeouts (Fortsetzung):
 - ★ Auslösen eines Timeouts, wenn der Server nicht auf eine Anfrage antwortet.
Dazu kann ein Timeout definiert werden, der Ablaufen ein Event auslöst.
`request.timeout`
`request.ontimeout`

AJAX-Programmierung

Timeout

★ Beispiel:

```
// Timeout von 5 Sekunden setzen  
request.timeout = 5000;
```

```
// Eventhandler für Timeout  
request.ontimeout = function (e) {  
    window.alert("Timeout");  
}
```

AJAX-Programmierung

POST

- ★ Sollen Daten per POST übertragen werden, muss ein POST-Body bei Methode `request.send(body)` angegeben werden
- ★ Zusätzlich muss **zuvor** der Header für die Anfrage gesetzt werden

```
request.setRequestHeader(  
    'Content-Type',  
    'application/x-www-form-urlencoded');
```

AJAX-Programmierung

AJAX elegant

- ★ Im vorangegangenen Beispiel ist die Programmierung recht aufwändig und benötigt unschöne globale Variablen.
- ★ Eleganter ist es, wenn alles in einer Klasse gekapselt wird:
 - ★ Wahlweise synchrone oder asynchrone Aufrufe
 - ★ Timeout-Zeiten
 - ★ HTTP-GET oder HTTP-POST
 - ★ Abbrechen einer Anfrage
 - ★ Und vieles mehr

AJAX-Programmierung

AJAX abgewandelt

- ★ AJAX muss nicht in seiner ursprünglichen Idee benutzt werden.
 - ★ **AJAX ohne A:** Es kann durchaus viele Gründe geben, um eine Kommunikation synchron zu gestalten. Wie dies gemacht werden kann, ist schon kurz angesprochen worden.
 - ★ **AJAX ohne X:** AJAX muss nicht ausschließlich nur mit XML-Nachrichten arbeiten. Es kann z.B. auch Text, HTML, JavaScript oder JSON per AJAX übertragen werden. Je nach Wahl muss `responseText` oder `responseXML` benutzt werden. In Verbindung mit JavaScript und JSON mit der Methode `JSON.parse()`.

Bibliotheken

- ☆ Es gibt sehr viele verschiedene Bibliotheken, die den Einsatz von JavaScript erleichtern sollen.
- ☆ Eine Einordnung lässt sich grob nach Bibliotheken für die Client- oder Serverseite machen.
 - ☆ **Clientseite:** Vereinfachung von JavaScript und XMLHttpRequest Programmierung
 - ☆ **Serverseite:** automatische Generierung von JavaScript zur DOM Programmierung auf Serverseite (Java, PHP, Perl, C#, ...)

Bibliotheken jQuery

★ jQuery

- ★ Objektorientierte Bibliotheken
- ★ Zentrales ist das `jQuery` Objekt
- ★ Vereinfacht die JavaScript Programmierung
- ★ Einfach nutzbare AJAX Klassen
- ★ Kapselt die Unterschiede in den einzelnen Browsern
- ★ <http://jquery.com/>

Bibliotheken jQuery

- ★ Einbinden (komprimierte Variante):
`<script src="jquery.js"></script>`
- ★ Factory-Funktion/Objekt:
- ★ `$ ()` oder `jQuery ()`
 - ★ vergleichbar mit `getElementBy...`
 - ★ aber liefert ein `jQuery`-Objekt zurück
 - ★ kann auch eine Knotenliste enthalten, wenn Selektor auf mehrere Elemente zutrifft

Bibliotheken jQuery

★ Beispiel:

```
<div id="div1"></div>
```

```
<div id="div2"></div>
```

★ `let div1 = $("#div1");`

`div1` ist ein jQuery-Objekt, mit dem das `div`-Element mit ID `div1` modifiziert werden kann

★ `let div = $("div");`

`div` ist ein jQuery-Objekt, mit dem alle `div`-Elemente gleichzeitig modifiziert werden können (es ist kein Array wie bei `getElementsByName`)

Bibliotheken jQuery

★ Mögliche Selektoren für `$ ()`

- ★ `$ (element)` : Selektiert übergebenes DOM-Element oder ein Array von DOM-Elementen
- ★ `$ (jquery)` : Selektiert jquery-Element
- ★ `$ (htmlString)` : Erzeugt ein DOM-Element aus String
- ★ `$ (css)` : Selektiert anhand von beliebigen CSS-Selektoren (ID, Klassen, Pseudo-Klassen)

Bibliotheken jQuery

★ Mögliche Selektoren für `$ ()` (Fortsetzung)

★ Basisselektoren

Selektor	Typ	Erklärung
<code>*</code>	Alle	Selektiert typunabhängig alle Elemente
<code>E</code>	Typ	Selektiert alle Elemente mit dem Bezeichner E
<code>.class</code>	Klasse	Selektiert alle Elemente mit genannter CSS-Klasse
<code>#id</code>	ID-Sel.	Selektiert alle Elemente mit der übergebenen ID

★ Mehrfachselektoren

Selektor	Typ	Erklärung
<code>.class1.class2</code>	Klasse	Selektiert Elemente, die alle genannten CSS-Klassen enthalten

Bibliotheken jQuery

★ Mögliche Selektoren für `$ ()` (Fortsetzung)

★ Gruppen- und Kontextselektoren

Selektor	Erklärung
<code>sel1, sel2, ...</code>	Selektiert die Kombination aus allen durch die Selektionen ausgewählten Knoten.
<code>E F</code>	Selektiert alle Elemente F, die Nachfahren eines Elements E sind
<code>E > F</code>	Selektiert alle Elemente F, die Kindknoten eines Elements E sind
<code>E + F</code>	Selektiert alle unmittelbar auf ein Element vom Typ E folgenden Elemente vom Typ F
<code>E ~ F</code>	Selektiert alle folgenden Geschwisterknoten eines Elements E, die den Typ F besitzen

Bibliotheken jQuery

★ Beispiel:

```
<div id="div1">
  <p id="p1">Text 1</p>
  <p id="p2">Text 2
    
  </p>
</div>
<div id="div2"></div>
```

```
$("#div1 *")      → #p1, #p2, #img1
$("#div1 > *")    → #p1, #p2
$("#div1 ~ *")    → #div2
```

Bibliotheken jQuery

★ Mögliche Selektoren für `$ ()` (Fortsetzung)

★ Filter

Selektor	Erklärung
<code>:first/:last</code>	Selektiert das erste/letzte Element
<code>:even/:odd</code>	Selektiert alle Elemente mit geraden/ungeraden Index innerhalb der aktuellen Kollektion
<code>:not(sel)</code>	Selektiert alle Elemente, auf die der Selektor <code>sel</code> nicht zutrifft
<code>:contains(t)</code>	Selektiert alle Elemente, die den übergebenen Text <code>t</code> enthalten
<code>:empty</code>	Selektiert alle Elemente ohne Inhalt
<code>:hidden</code>	Selektiert alle Elemente mit <code>display="none"</code> , <code>Höhe/Breite = 0</code> oder <code><input type="hidden"/></code>
<code>:visible</code>	Gegenstück zu <code>:hidden</code>

Bibliotheken jQuery

★ Beispiel:

```
<div id="div1">
  <p id="p1">Text 1</p>
  <p id="p2">Text 2
    
  </p>
</div>
<div id="div2"></div>
```

<code>\$ ("p:first")</code>	→ <code>#p1</code>
<code>\$ ("p:contains(Text 1)")</code>	→ <code>#p1</code>
<code>\$ ("p:not(:contains(Text 1))")</code>	→ <code>#p2</code>
<code>\$ (":empty")</code>	→ <code>#img1, #div2</code>

Bibliotheken jQuery

★ Mögliche Selektoren für `$ ()` (Fortsetzung)

★ Attributfilter

Selektor	Erklärung
<code>[name]</code>	Selektiert die Elemente beliebigen Type, die über ein Attribut <code>name</code> verfügen
<code>E[name]</code>	Selektiert alle Elemente vom Type <code>E</code> , die über ein Attribut <code>name</code> verfügen
<code>[name=wert]</code>	Selektiert alle Elemente, wenn der Wert des Attributs dem übergebenen String entspricht
<code>[name*=wert]</code>	Selektiert alle Elemente, wenn der Wert des Attributs dem übergebenen String als Substring enthält



Nur eine Auswahl!

Bibliotheken jQuery

★ Beispiel:

```
<div id="div1">  
  <input id="i1" type="number" />  
  <input id="i2" type="checkbox" />  
</div>  
<div></div>
```

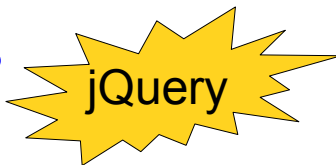
```
$("[id]")           → #div1, #i1, #i2  
$("input[id]")      → #i1, #i2  
$("[type=checkbox]")   → #i2
```


Bibliotheken jQuery

★ Eventhandling

★ Event-Objekt

Eigenschaft	Erklärung
<code>e.target</code>	Das DOM-Element, an dem das Ereignis stattfand
<code>e.currentTarget</code>	Das DOM-Element, das während der Capturing-/ Bubbling-Phase Ziel des Ereignisses ist
<code>e.clientX/Y</code>	Mausposition relativ zum Browserfenster
<code>e.timeStamp</code>	Zeitstempel des Ereignisses (s. <code>Date()</code>)
<code>e.type</code>	Die Art des Ereignisses
<code>e.keyCode</code>	Die Keyboard-Taste, die betätigt wurde
<code>e.data</code>	Die Daten, die für das aktuelle Ereignis übergeben werden, sofern vorhanden
...	...



Bibliotheken jQuery

★ Eventhandling – Methoden von `$()`



vor Version 1.7
bind/unbind

★ `.on(typ, fn)`

★ Bindet einen Handler `fn` an den Event `typ`

★ `fn` bekommt ein Event-Objekt übergeben

★ `typ` können auch mehrere Events sein

★ Ist `$()` eine Kollektion, wird allen der Handler zugewiesen

★ `.on(typ, [data], fn)`

★ Optional Daten, die in Event-Objekt gespeichert werden

★ `.off(typ)`

★ Entfernt alle Eventbindungen vom Typ `typ`

Bibliotheken jQuery

★ Beispiel:

```
<p id="p1">Text 1</p>
```

```
<p id="p2">Text 2</p>
```

```
$("#p1").on("click", function (e) {  
    window.alert("angeklickt");  
});
```

```
$("p").on("click", function (e) {  
    window.alert("angeklickt");  
});
```

Bibliotheken jQuery

★ Eventhandling – Abbruch

- ★ `e.stopPropagation()` – verhindert Weitergabe des Ereignisses an folgende Observer
- ★ `e.preventDefault()` – verhindert die Defaultaktion, die mit dem Ereignis verbunden ist

★ Eventhandling – Shortcuts – Methoden von `$()`

★ `.ready(fn(e))` Sobald HTML-Struktur bereitsteht (vor load)

★ `.click(fn(e))` Bindet ein Handler `fn` an das `click`-Event

★ ...

Bibliotheken jQuery

★ Event feuern – Methoden von `$()`

★ `.trigger(typ, arrPm)` – Triggert ein Event `typ` und übergibt Array als weiteren Parameter

★ Beispiel:

```
<input type="checkbox" id="i2"/>
```

```
$("#i2").trigger('click');
```

Bibliotheken jQuery

★ Inhalt setzen – Methoden von \$ ()

★ `.html ()`

`.html (htmlString)`

`.append (inhalt)`

Holt bzw. ersetzt (hängt an) den HTML-Inhalt durch den übergebenen HTML-String

★ `.text ()`

`.text (txt)`

Holt bzw. setzt den Textinhalt auf den übergebenen Textstring txt

★ `.empty ()`

Entfernt den Inhalt (alle Kindknoten) aus den Elementen

Bibliotheken jQuery

★ Beispiel

```
<div id="div1"></div>
```

```
<div id="div2"></div>
```

```
$("#div1")
```

```
    .html("<p>Dies ist HTML-Inhalt.</p>");
```

```
$("#div2")
```

```
    .text("Dies ist Text ohne <tags>.");
```

Bibliotheken jQuery

★ Attribute setzen – Methoden von \$ ()

- ★ `.attr(name)`
`.attr(name, val)`
`.removeAttr(name)`

Holt bzw. setzt den Wert eines Attributs `name`

- ★ `.val()`
`.val(wert)`

Holt bzw. setzt den Wert eines Input-Elements

Bibliotheken jQuery

★ Beispiel

```
<input type="checkbox" id="i2"/>
```

```
★ let i2 = $("#i2");  
i2.attr("type", "text");  
i2.val("Text");
```

oder

```
★ $("#i2").attr("type", "text")  
    .val("Text");
```

Bibliotheken jQuery (AJAX)

★ Inhaltsmethoden von `$ ()`

- ★ Methode `.ajax (config)` stellt eine AJAX-Anfrage
- ★ Die Konfiguration geschieht mittels JSON
 - ★ `type`: `get` oder `post`
(andere nicht von allen Browsern unterstützt)
 - ★ `url`: Adresse und get-Parameter der Anfrage
 - ★ `dataType`: vom Server übermittelter Type (`xml`, `json`)
 - ★ `data`: Daten der Anfrage als Querystring oder JSON
 - ★ `success`: `function (data) { }`: Funktion, die ausgeführt werden soll, wenn Anfrage erfolgreich
 - ★ `error`: `function (...`) { }: Funktion, die ausgeführt werden soll, wenn Fehler aufgetreten

Bibliotheken jQuery (AJAX)

★ Fortune-Beispiel:

```
<body> <script type="text/javascript"
        src="jquery.min.js"></script>
<div id="fortune" >
  <p><b>Spruch:</b></p>
  <div id="fortuneMessage"></div>
  <form action="">
    <p><input type="button" value="Neuer Spruch"
              onclick="
                $.ajax( { url: 'fortune.php',
                          success: function (data) {
                            $('#fortuneMessage').html(data);
                          } } );"/>
    </p> </form>
  </div> </body>
```

Bibliotheken jQuery (AJAX)

★ AJAX Convenience-Methoden

- ★ `.post()` Setzt einen post-Request ab
- ★ `.get()` Setzt einen get-Request ab
- ★ `.getJSON()` Setzt einen get-Request ab und erwartet JSON-encodierte Daten
- ★ `.getScript()` Setzt einen get-Request ab und führt das zurückgegebene JavaScript aus
- ★ `.load(url)` Lädt die mit `url` bezeichnete Ressource und fügt diese als HTML in das durch die Collection bestimmte Zielelement ein (*Achtung:* bis V1.7 kann `.load()` auch für Events genutzt werden)

Bibliotheken jQuery (AJAX)

★ Fortune-Beispiel:

```
<body> <script type="text/javascript"
        src="jquery.min.js"></script>
<div id="fortune" >
  <p><b>Spruch:</b></p>
  <div id="fortuneMessage"></div>
  <form action="">
    <p><input type="button" value="Neuer Spruch"
              onclick="
                $('#fortuneMessage').load('fortune.php');
              ">
    </p> </form>
</div> </body>
```

Bibliotheken jQuery (AJAX)

★ Serialisierung von Formulardaten

- ★ `.serialize()` - Serialisiert die Werte aus Formularelementen als URL-encodierten Textstring
- ★ `.serializeArray()` - Serialisiert die Werte aus Formularelementen als JavaScriptArray (für JSON)
- ★ Als Parameter für AJAX nutzbar

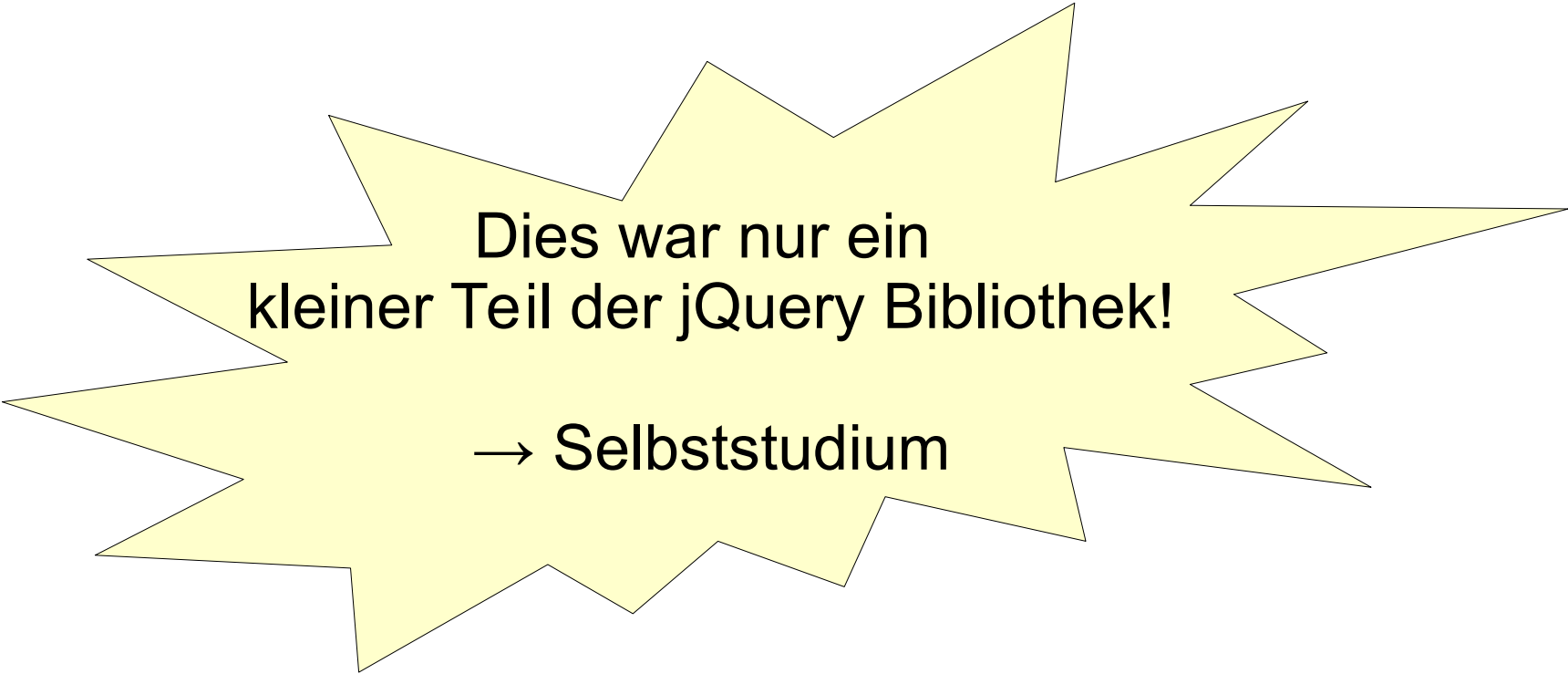
Bibliotheken jQuery (AJAX)

★ Beispiel:

```
<div id="div2">
  <input type="checkbox" id="i1" name="i1"
        value="true" checked/>
  <label for="i1">I1</label>
  <input type="checkbox" id="i2" name="i2"
        value="true"/>
  <label for="i2">I1</label>
  <input type="text" id="i3" name="i3" value="Text"/>
</div>
```

```
console.log($('#div2 > *').serialize());
console.log($('#div2 > *').serializeArray());
```

Bibliotheken jQuery (AJAX)



Dies war nur ein
kleiner Teil der jQuery Bibliothek!

→ Selbststudium

Literatur

- ★ Vallendorf, M. und Bongers, F.: jQuery – Das Praxisbuch, Galileo Computing, 2010
- ★ jQuery Project: jQuery, <http://jquery.com>
- ★ Flanagan, D.: JavaScript - The Definitive Guide, Auflage 5, O'Reilly, 2006
- ★ Mintert, Stefan und Christoph Leisegang: Ajax – Grundlagen, Frameworks und Praxislösungen, dpunkt Verlag, 2007
- ★ Wösten, André: jQuery – Das Training für interaktive und animierte Websites! Galileo Computing, 2011