Fortgeschrittene Programmierung

Vorlesungsfolien
Thomas Richter nach Unterlagen v. Dietmar Jannach

Serverseitiges JavaScript 1

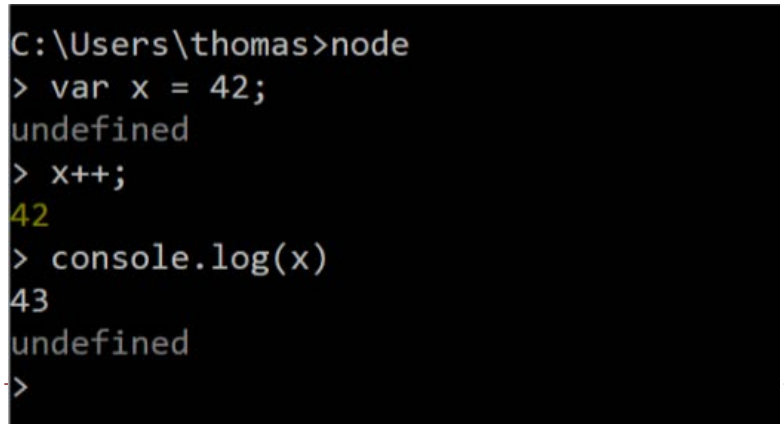
Node.js

Einleitung

- Warum JS auf dem Server?
  - Geringere technologische Breite des Systems (mindestens eine Sprache weniger, da JS auf dem Client sowieso gesetzt ist)
  - Geringere Anforderungen an Know-How der Entwickler: im Optimalfall nur eine Sprache
  - Geschäftslogik lässt sich einfacher vom Client zum Server verschieben und umgekehrt
  - Weil es geht ;-)

Node.js: Historie und Einordnung

- Serverseitige JavaScript Plattform zur Entwicklung von Webservern
- Basiert auf der JavaScript Engine V8 (Google), die auch in Google Chrome integriert ist.
- Eventbasiertes Handling von Requests
- Erweiterbar mit Modulen
- Bringt eine interaktive Console mit



Grundprinzipien

- Module werden mit require eingebunden:

```
// HTTP Modul einbinden
var http = require("http");

// FileSystem Modul einbinden
var fs = require("fs");
```

- Es können normale JavaScript-Programme ausgeführt werden:

Datei hello.js:

```
console.log("Hallo Welt");
```

```
> node hello.js
Hallo Welt
```

Webserver Hello World

```
var http = require("http");

// HTTP Server erzeugen und Serverfunktion registrieren
var server = http.createServer(function(req, res) {
    res.writeHead(200, {
        "content-type": "text/plain"
    });

    res.write("Hallo Welt, das kommt vom Server");

    res.end();
})
.listen(3000);
```

Request / Response

- Die Callback-Funktion des Servers erhält beim Eintreffen des Requests zwei Parameter:
  - req ist vom Typ http.IncomingMessage und enthält alle Daten des Requests
    - req.headers
    - req.url
    - …
    - https://nodejs.org/api/http.html#http_class_http_incomingmessage
  - res ist vom Typ http.ServerResponse und kann mit der Antwort (Response) gefüllt werden
    - Senden an den Client mit res.write();
    - Abschließen mit res.end();
    - https://nodejs.org/api/http.html#http_class_http_serverresponse

Fehlende Module installieren

```
C:\Users\...\arbeitsverzeichnis> npm install modulname
```

Demo

Dynamische Webseiten mit JavaScript

AJAX

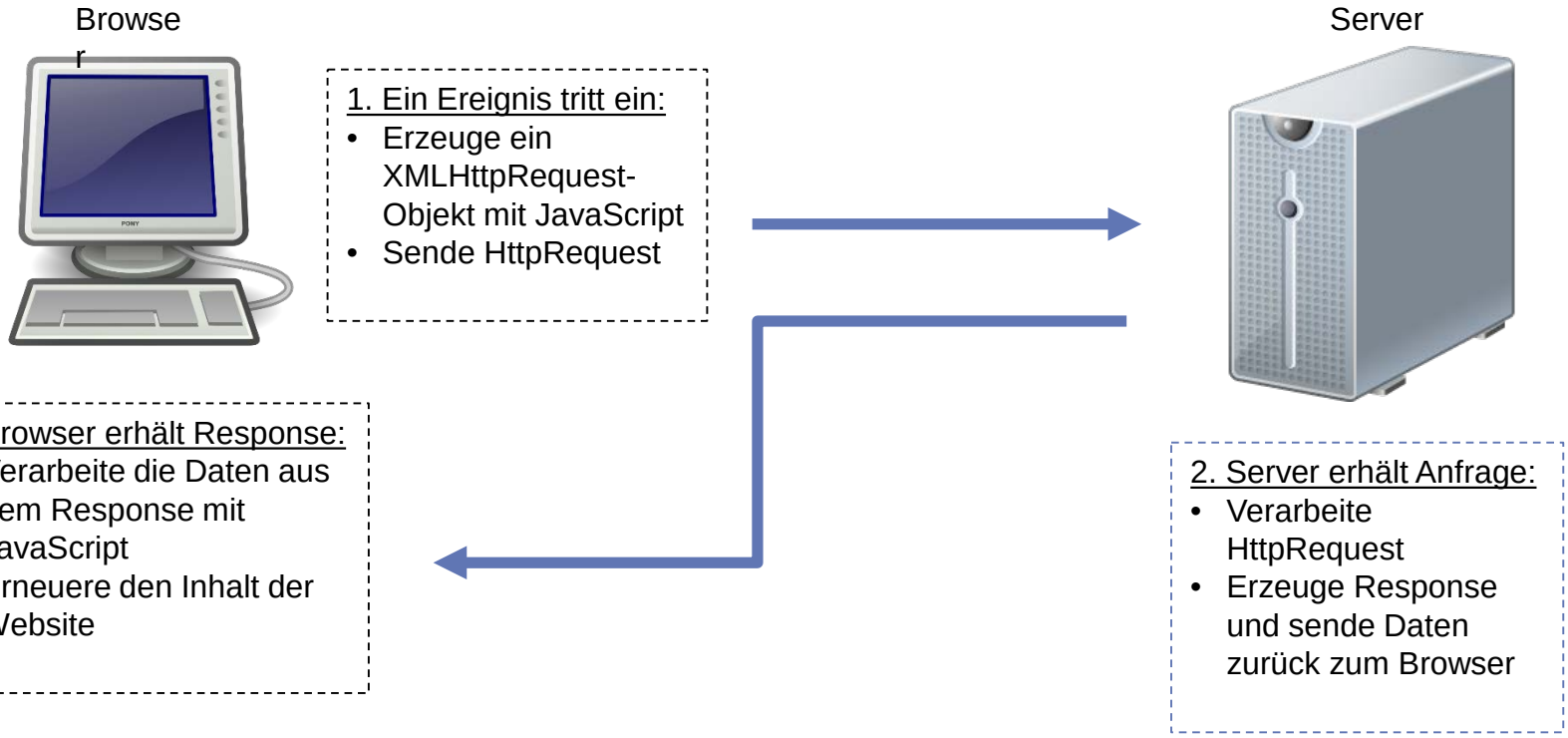
AJAX – Einführung

- Das Problem
  - Wir möchten gerne dynamische Webseiten bauen, die in der Lage sind, auf Veränderungen oder Benutzereingaben zu reagieren und "spontan" neue Informationen vom Server einzuholen. Dies soll dynamisch und im Hintergrund passieren, der Benutzer soll nicht erst auf Buttons klicken, oder die Seite manuell neu laden müssen
  - Die bisherigen Mittel ermöglichen dies durch JavaScript nur lokal beim Client

- Die Lösung
  - AJAX = Asynchronous JavaScript and XML
  - Mit Hilfe von JavaScript wird eine HTTP-Anfrage erzeugt
  - Über das sog. XMLHttpRequest-Object wird die HTTP-Anfrage an den Server übermittelt
  - Das Warten auf die Antwort blockiert die Webseite nicht. Der Response vom Server wird erst verarbeitet, wenn er da ist → asynchrone Kommunikation
  - Der Response vom Server wird mit JavaScript verarbeitet
  - Da alles über JavaScript läuft, können diese Anfragen jederzeit gesendet werden und somit auf beliebige Ereignisse reagieren

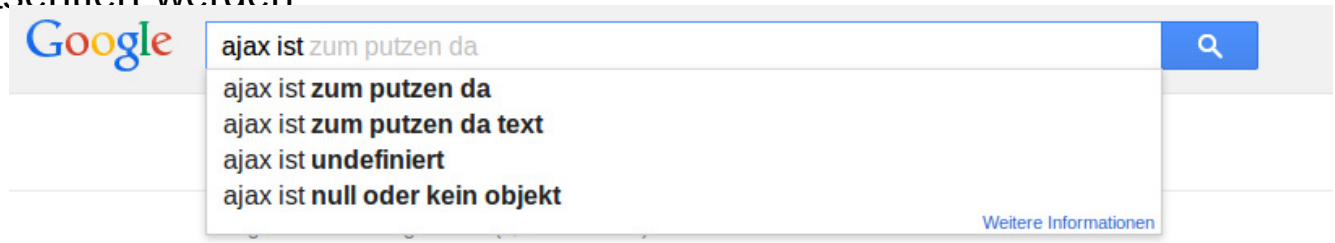
AJAX – Einführung

- Veranschaulichung



AJAX – Einführung

- AJAX wird von vielen modernen Webseiten verwendet
- Beispiel: Google-Suche
  - Sobald der Benutzer einen neuen Buchstaben ins Suchfeld eintippt, wird der aktuelle Suchbegriff an den Server geschickt
  - Dieser Antwortet mit einer Liste von Vorschlägen, die dem Benutzer sofort präsentiert werden



AJAX – Einführung

- AJAX Übersicht
  - Eine Technik für schnelle und dynamische Webseiten
  - Basiert auf bekannten Standards und ist keine neue Programmiersprache
  - Lösung für ein Problem, das erst nach der Standardisierung von HTTP entstanden ist. Würde HTTP heute entwickelt werden, würde man dieses Problem bereits dabei berücksichtigen
  - Erlaubt es, Teile einer Webseite asynchron zu aktualisieren, ohne diese vollständig neu laden zu müssen
  - Browser- und plattformunabhängig

- Verwendete Standards
  - XMLHttpRequest-Objekt zum asynchronen Datenaustausch mit dem Server
  - JavaScript/DOM zur Anzeige und Interaktion mit Daten
  - CSS für eine ansprechende Visualisierung (keine Voraussetzung)
  - XML als Format für die Datenübertragung

AJAX – Verarbeitung einer Anfrage

- Asynchrone Kommunikation
  - Empfangen und Senden von Daten erfolgt zeitlich versetzt
  - Beteiligte Prozesse werden nicht blockiert
  - Eine Webseite kann partiell aktualisiert werden
  - Der Benutzer muss nicht warten, bis die Anfrage vollständig verarbeitet wurde

- Synchrone Kommunikation
  - Wird eine synchrone Anfrage gestellt, ist die Weiterarbeit auf der Clientseite blockiert
  - Interaktion mit der Webseite nicht mehr möglich bis die Antwort vom Server da ist
  - Dieses Verhalten kann in seltenen Fällen erwünscht sein z.B. beim Ereignis `window.onbeforeunload,` bei dem eine asynchrone Anfrage nicht bis zum Ende ausgeführt werden könnte, weil die Seite dann bereits geschlossen wäre

AJAX – Rich Internet Applikationen mit Ajax

- Vielfältige Interaktionsmöglichkeiten für Benutzer
- Beispiel: Validierung von Formularen
  - Jedes Feld hat Regeln, die eingehalten werden müssen (Name, Email, Telefonnummer …)
  - Mit AJAX
    - Benutzer gibt z.B. eine E-Mail Adresse in ein Formularfeld ein
    - Er wechselt zum nächsten Feld
    - Die vorherige Eingabe wird automatisch per asynchronem Request an Server gesendet
    - Server prüft die Eingabe
    - Wenn Eingabe fehlerhaft, wird eine Fehlermeldung an entsprechender Stelle ausgegeben
    - Benutzer erkennt Fehler und kann sofort korrigieren
  - Ohne AJAX
    - Eine serverseitige Überprüfung kann erst stattfinden, wenn das Formular abgeschickt wurde

AJAX – XMLHttpRequest-Objekt

- HTTP-Anfragen per JavaScript
  - Das XMLHttpRequest-Objekt ermöglicht es, HTTP-Anfragen per JavaScript zu stellen
  - Für die Client-Server Kommunikation muss eine XMLHttpRequest-Instanz erstellt werden

AJAX – XMLHttpRequest-Objekt

Methoden

- `open()`
  - Zur Initialisierung einer Verbindung. Erwartet drei Parameter
    - Methode der Übertragung
    - Name der angefragten Datei
    - Ob die Übertragung asynchron (`true`) oder synchron (`false`) erfolgen soll
  - Wird der dritte Parameter weggelassen, erfolgt die Kommunikation asynchron

- `onreadystatechange`
  - Ist ein Ereignis, das auf Zustandsänderungen innerhalb der Anfrage reagiert
  - Bekommt als Wert den Namen der aufzurufenden Funktion (Callback-Funktion)
    - Z.B.: `http.onreadystatechange = myCallbackFunction();`
  - Bei asynchroner Kommunikation informiert der Server ständig über Zustandsänderungen

AJAX – XMLHttpRequest-Objekt

Methoden

- `send()`
  - Zum Senden der Anfrage
  - Erhält als Parameter die über POST zu sendenden Daten
  - Bei Übertragungsmethode GET wird als Parameter `null` übergeben, da die Daten bereits an die URL gehängt werden

```
anfrage.open( "GET", url, true );
anfrage.onreadystatechange = meineCallbackFkt;
anfrage.send( null );
```

Callback-Funktion
Dazu später mehr

AJAX – readyState-Eigenschaft

- Gibt den aktuellen Verbindungsstatus einer zuvor abgesendeten HTTP-Anfrage an
  - 1 (LOADING)
    HTTP-Verbindung zum Webserver erfolgreich zustande gekommen
    `open()` konnte Verbindung mit dem Server aufnehmen, aber noch keine Anfrage mit `send()` starten
  - 2 (LOADED)
    HTTP-Anfrage wurde vollständig übertragen und Antwort-Header-Zeilen vom Server liegen bereits vor
  - 3 (INTERACTIVE)
    Die eigentlichen Nutzdaten werden vom Server empfangen
    `responseText` bzw. `responseXML` werden nach und nach mit den empfangenen Daten gefüllt
  - 4 (COMPLETED)
    Server-Antwort inklusive aller Header- und Nutzdaten wurden empfangen
  - Implementierung der Zustände in den Browsern ist nicht einheitlich
  - Deshalb wird meistens `readyState==4` abgewartet, bis mit der Verarbeitung der Daten begonnen wird

Mehr unter: http://webkompetenz.wikidot.com/ajax:2-2

AJAX – Callback-Funktion

- Nach jeder Zustandsänderung der readyState-Eigenschaft wird die Callback-Funktion aufgerufen
- Ist für die Verarbeitung der Daten zuständig

```
function meineCallbackFkt() {
   if(anfrage.readyState == 4) {
      if(anfrage.status != 200) {
         alert( "Fehler " + anfrage.status + ": " + anfrage.statusText ); }
      else {
         document.getElementById("contentArea").innerHTML = anfrage.responseText;
      }
   }
}
```

- Prüft, ob Daten angekommen sind (`readyState == 4`) und ob die Server-Antwort OK ist (`status == 200`)
- Wenn nicht OK, wird Fehlermeldung unter Angabe von Status und Status-Text ausgegeben
- Wenn OK, dann wird das Element mit der ID `contentArea` ermittelt und die Server-Antwort wird in das Element geschrieben (`innerHTML = anfrage.responseText`)

Demo

AJAX – Verwendung von XML und DOM

- XML zur Übertragung von strukturierten Daten
  - Das XMLHttpRequest-Objekt erhält die Antwort vom Server als XML,
  - Die Antwort wird geparst und die Daten als XML DOM-Objekt in responseXML gespeichert
  - DOM-Funktionen können genutzt werden, um Daten aus XML zu extrahieren

```
function processResponse() {
   if(anfrage.readyState == 4) {
      if(anfrage.status != 200) {
         alert( "Fehler " + anfrage.status + ": " + anfrage.statusText ); }
      else {
         var elements = anfrage.responseXML.getElementsByTagName("elementsName")
         document.getElementById("contentArea").innerHTML =
                  elements.item(0).firstChild.nodeValue;
      }
   }
}
```

  - `elements` enthält alle Elemente die den Namen `elementsName` haben
  - `elements.item(0)` wählt das erste Element, das den Namen `elementsName` hat
  - Mit `firstChild` wird dessen erstes Kind-Element angesprochen
  - Der Inhalt des Kind-Elements wird in `contentArea` geschrieben

AJAX – JSON

- Was ist JSON?
  - JSON: **J**ava**S**cript **O**bject **N**otation
  - Eine Syntax zum Speichern und Austauschen von Daten
  - Eine einfachere Alternative zu XML
  - Sehr stark an JavaScript-Syntax angelehnt, dennoch programmiersprachenunabhängig
  - Lässt sich in AJAX verwenden (anstatt XML)
- Wieso JSON?
  - JSON-Notation ist kürzer und einfacher zu lesen und zu schreiben
  - Kann mit Arrays umgehen
  - Kann von JavaScript Standardfunktionen ausgelesen werden
  - Intuitiv und selbsterklärend
  - Bei AJAX-Anwendungen schneller und einfacher als XML

AJAX – JSON

JSON Syntax

- Syntaxregeln
  - Objekte sind Schlüssel/Werte-Paare
  - Attribute sind kommasepariert
  - Geschweifte Klammern repräsentieren Objekte
  - Eckige Klammern repräsentieren Arrays

- Syntaxbeispiel
  - `{"firstName":"John", "lastName":"Doe" }`
  - Definiert ein JSON-Objekt mit den Attributen "firstName" und "lastName"
  - Mit dieser Syntax lassen sich nicht nur JSON-Objekte, sondern auch JavaScript-Objekte erzeugen

AJAX – JSON
JSON Syntax

- JSON Arrays
  - Arrays stehen in eckigen Klammern und können mehrere Objekte beinhalten
  - In JavaScript kann JSON-Notation genutzt werden, um ein Array zu erstellen
  - Folgendes ist valider JavaScript-Code. Alles was rechts des Gleichzeichens steht, ist auch korrekte JSON-Syntax

```
var employees = [ { "firstName" : "John" , "lastName" : "Doe"},
                  { "firstName" : "Anna" , "lastName" : "Smith"},
                  { "firstName" : "Peter" , "lastName" : "Jones"} ];
```

AJAX – JSON
JSON vs XML

XML-Notation

```
<employees>
    <employee>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName>
        <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName>
        <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON-Notation

```
{"employees":[
    {"firstName":"John",
     "lastName":"Doe"},
    {"firstName":"Anna",
     "lastName":"Smith"},
    {"firstName":"Peter",
     "lastName":"Jones"}
]}
```

- Beide Beispiele sind semantisch identisch

AJAX – JSON

JSON – Zugriff auf Objekte

```
<!DOCTYPE html> <html> <body>

<h2>JSON Objekte in JavaScript erzeugen</h2>

<p id="demo"></p>

<script>
    var text = '{"name":"John Johnson","street":"Oslo West  16","phone":"555
   1234567"}';

    var obj = JSON.parse(text);

    document.getElementById("demo").innerHTML =
    obj.name + "<br>" +
    obj.street + "<br>" +
    obj.phone;
</script> </body> </html>
```

AJAX – Probleme und Kritik

- Lösung an der falschen Stelle
  - Als HTTP entwickelt wurde, hatte man noch nicht daran gedacht, dass asynchrone Kommunikation gebraucht werden könnte. Eine saubere Lösung für dieses Problem gehört eigentlich in eine tiefere Ebene
  - Daraus ergibt sich eine Reihe von Problemen
- Belastende Anfragen – Polling
  - Da der Server nicht in der Lage ist, Events asynchron an den Client zu pushen, muss der Client permanent nachfragen (polling), was zu mehr Traffic und höherer CPU-Auslastung am Server führt
- Browser-History
  - Nach einer partiellen Aktualisierung ist der vorherige Zustand mit dem Zurück-Button nicht wiederherstellbar
  - Auch Browser-Lesezeichen funktionieren nicht

AJAX – Probleme und Kritik

- Suchmaschinen
  - Suchmaschinen können den Inhalt partiell aktualisierter Webseiten nicht ohne Weiteres zur Suche indizieren
- JavaScript deaktiviert
  - AJAX funktioniert nur mit JavaScript
  - JavaScript kann deaktiviert oder nicht verfügbar sein. Dann muss eine alternative Version ohne AJAX erstellt werden, oder der Nutzer muss aufgefordert werden, JavaScript einzuschalten
- Probleme mit Barrierefreiheit
  - Partiell aktualisierte Seiten werden von Screenreadern für sehbehinderte Menschen oft nicht erkannt

Beispiel: AJAX mit jQuery

```
$.ajax({
    type: 'GET',
    url: '/myshop/products',
    contentType: 'application/json',
    dataType: 'text/plain'
})
.done(function(data){
    alert(data);
})
.fail(function() {
    alert('Error');
});
```