# Netzbasierte Anwendungen

# Dynamisches Nachladen von Javascript im Browser

Johannes Hamfler & Philipp Dockhorn

Hochschule für Telekommunikation Leipzig

29. Januar 2015

- Anforderungen an Webanwendungen $\Uparrow$ (Traffic $\Uparrow$)
- Personalisierte Nutzerschnittstellen
- Dynamische Nutzerschnittstellen
- Ausführungsgeschwindigkeit $\Downarrow$
- Modularisierung

- Möglichkeiten aufzuzeigen
  - Nachlademethoden
  - Datenformate
- REST-Schnittstelle implementiert
- Praktisch Beispiele demonstrieren
- Praxistauglichkeit
- Sicherheitsaspekte

- Einfachste Möglichkeit
- Im HTML-head oder -body
- Nur Dateien (zunächst)
- Same-Origin-Policy greift nicht
  ↷ laden aus beliebigen, auch Domain-fremden Quellen
- Laden des Script-Code bei Seitenaufruf
  ↷ Traffic ⇑ Aufrufzeit ⇑

```
1 <head>
2 <script src="js/test.js"></script>
3 ...
4 </head>
```

- Script-Tag erst zur Laufzeit einbinden

```
1 function loadScript(scriptname) {
2     var s = document.createElement('script');
3     s.setAttribute('type', 'text/javascript');
4     s.setAttribute('src', scriptname);
5     document.getElementsByTagName('head')[0].
          appendChild(s);
6 }
```

- **A***synchronous* **J***avaScript* **a***nd* **X***ML*
- Asynchronen Datenübertragung (S⇔C)
- Dateiformat nicht zwangsläufig XML
- XMLHttpRequest-Objekt (ActiveXObject)

```
1 var xhr=new XMLHttpRequest();
2 xhr.open("GET","foo/bar.js",true);
3 xhr.send();
```

- Wichtige Funktion
  - ⇒ xhr.open(...)
  - ⇒ xhr.send(...)
  - ⇒ xhr.onreadystatechange = function () {...}
  - ⇒ xhr.status
  - ⇒ xhr.responseText
- successHandler-Callback-Funktion
  - ⇒ Verarbeitung erhaltener Daten
- errorHandler-Callback-Funktion
  - ⇒ Fehlerbehandlung

```
1 getsth (url,successHandler,errorHandler);
```

- Was können sie?
  - ⇒ Grundstruktur / Entwicklungsrahmen
  - ⇒ Bibliotheken
  - ⇒ Basisbausteine (abstrakte und konkreten Klassen)
  - ⇒ Vereinfacht Nutzung
  - ⇒ zusätzliche Funktionalitäten
- Nutzen AJAX zum Nachladen
- Vereinfachtes Nachladen

```
function getJSON(url, successHandler, errorHandler) {
    var xhr = typeof XMLHttpRequest != 'undefined' ? new XMLHttpRequest() :
          new ActiveXObject('Microsoft.XMLHTTP');
    xhr.open('get', url, true);
    xhr.onreadystatechange = function () {
        var status;
        var data;
        if (xhr.readyState == 4) {
            if (status == 200) {
                successHandler(xhr.responseText);
            } else {
                errorHandler(xhr.status);
            }
        }
    };
    xhr.send();
};
```

```
$http.get(url).
  success(function(data, status, headers, config) {
    dosth();
  }).
  error(function(data, status, headers, config) {
   dosth();
  });
```

- Auszeichnungssprache; lesbar für Mensch und Maschine
- Selbst definierte Tags
- Escaping
  - Ersetzung reservierter Zeichen
  - XML entities
  - Overhead
- CDATA
  - `<![CDATA[` *beliebige ASCII-Zeichen* `]]>`
  - Weniger Overhead
  - Verbotene Zeichen erlaubt

- Kein komplexer Parser nötig
- JSON-Parser ist in Javascript vorhanden
- Sehr wenig Overhead
- Von JavaScript einfach zu interpretieren
- Einfach zu verstehen
- Kein Escaping notwendig, da JSON Javascript-kompatibel ist

```
1 {"Arbeiter":[
2 {"Vorname":"Reiner","Nachname":"Zufall"},
3 {"Vorname":"Anna","Nachname":"Nass"},
4 ]}
```

- Aufruf einer Funktion mit Parametern
- Parameter enthalten Payload
- Aufgerufene Funktion muss vorher definiert sein
- Funktionsname wird der Anfrage übergeben
- Umgeht SOP mit `script`-Tags

```
1 doparseResponse ({"Name":"Foo",
2                   "Nachname":"Bar","ID":42});
```

```
1 http://server2.example.com/Users/1234?jsonp=
    doparseResponse
```

- Zwei Server auf unterschiedlichen Domains
- MySQL-Datenbank
- REST-Schnittstelle

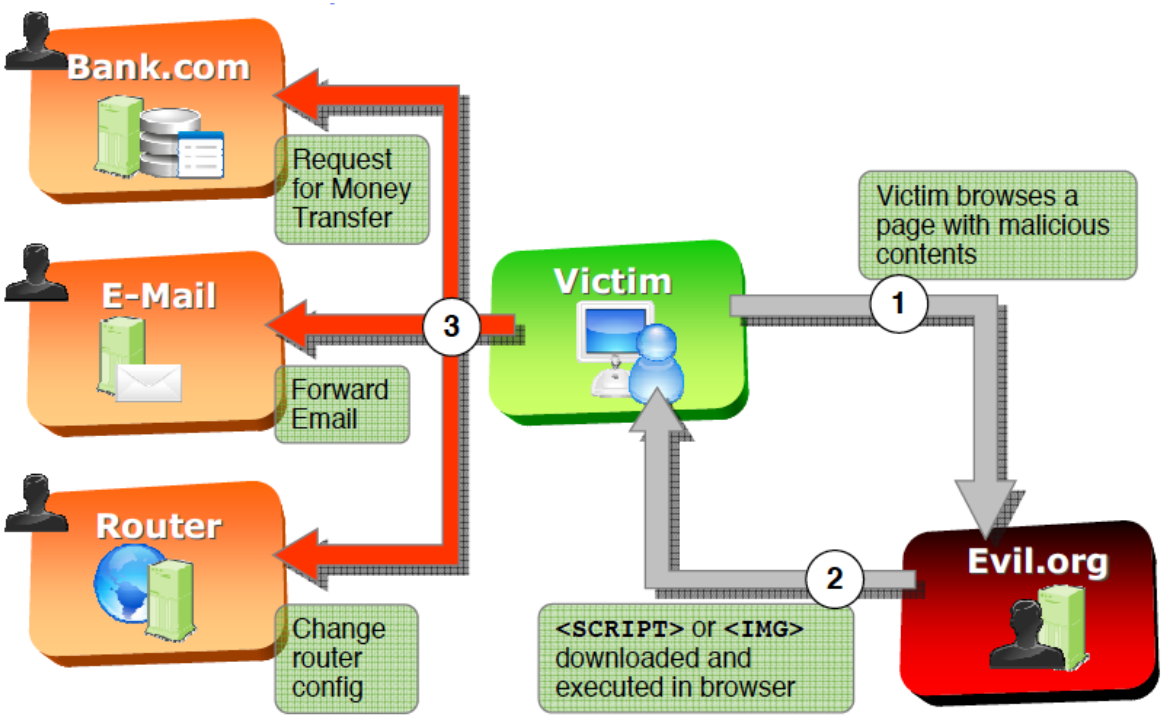
  $\Rightarrow$ JAVA
  - Java Persistence API und Jersey-Server

  $\Rightarrow$ NodeJS
  - Node, Module express, mysql und body-parser

  $\Rightarrow$ PHP
  - Datenbankframework Medoo

- Absicherung gegen Zugriff auf Inhalte aus fremder Domain
- Browser verwehrt Zugriff, nicht der Server
- Verhindert Nutzung von fremden Ressourcen
- Gleiche Quelle ist erlaubt
- Verwendet Port (z.B. 80), Protokoll (z.B. HTTPS) und IP-Adresse für Policy

- CSRF (Cross-Site Request Forgery)




1

[1] Quelle: http://www.devbattles.com/images/upload/csrf.png

- Proxy
  - Originwebsite durch den Proxy
  - Abfrage an fremden Server durch den Proxy
- JSONP
  - Einbindung in `script`-Tag
  - Browser wendet SOP nicht an
- CORS (Cross-Origin Resource Sharing)
  - HTTP-Header
  - Browser erlaubt Zugriff auf Response

```
1 res.header("Access-Control-Allow-Origin", "*");
2 res.header("Access-Control-Allow-Headers", "X-Requested-With");
3 res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
```

- `Document.write` ist nicht zu verwenden
  - Zerstört / überschreibt aktuelle Seite
  - Fehlerquellen treten auf
  - Sicherheitslücken entstehen
  - Nutzungszeitraum von `Document.write` ist auf Ladezeitraum beschränkt
- Bessere Lösung ist das Einbinden in Tags

```
1 document.getElementsByTagName("head")[0].appendChild(script);
2
3 document.getElementById(id).innerHTML = script;
4
5 document.getElementById(id).setAttribute('onclick',script +' '+aufruf);
```

- Dynamisches Nachladen von Javascript ist notwendig
- Gut realisierbar / Praxistauglich
- Nutzerfreundlichkeit kann erhöht werden

# Referenzen



http://www.w3schools.com/ajax/ , 22.01.2015


http://www.w3schools.com/xml/ , 22.01.2015


http://www.w3schools.com/json/ , 22.01.2015


http://www.json.org/ , 22.01.2015


http://www.jsonp.eu/ , 22.01.2015


http://www.hdfgroup.org/HDF5/XML/xml_escape_chars.htm , 22.01.2015


http://jsonp.eu/sop.html , 26.01.2015


http://www.freeformatter.com/xml-escape.html , 24.01.2015


http://nodejs.org/ , 19.01.2015


http://bsonspec.org/ , 27.01.2015


https://github.com/mongodb/js-bson , 27.01.2015

# Vielen Dank
# für die Aufmerksamkeit.