Version 2024b

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Grundlagen der Web-Programmierung

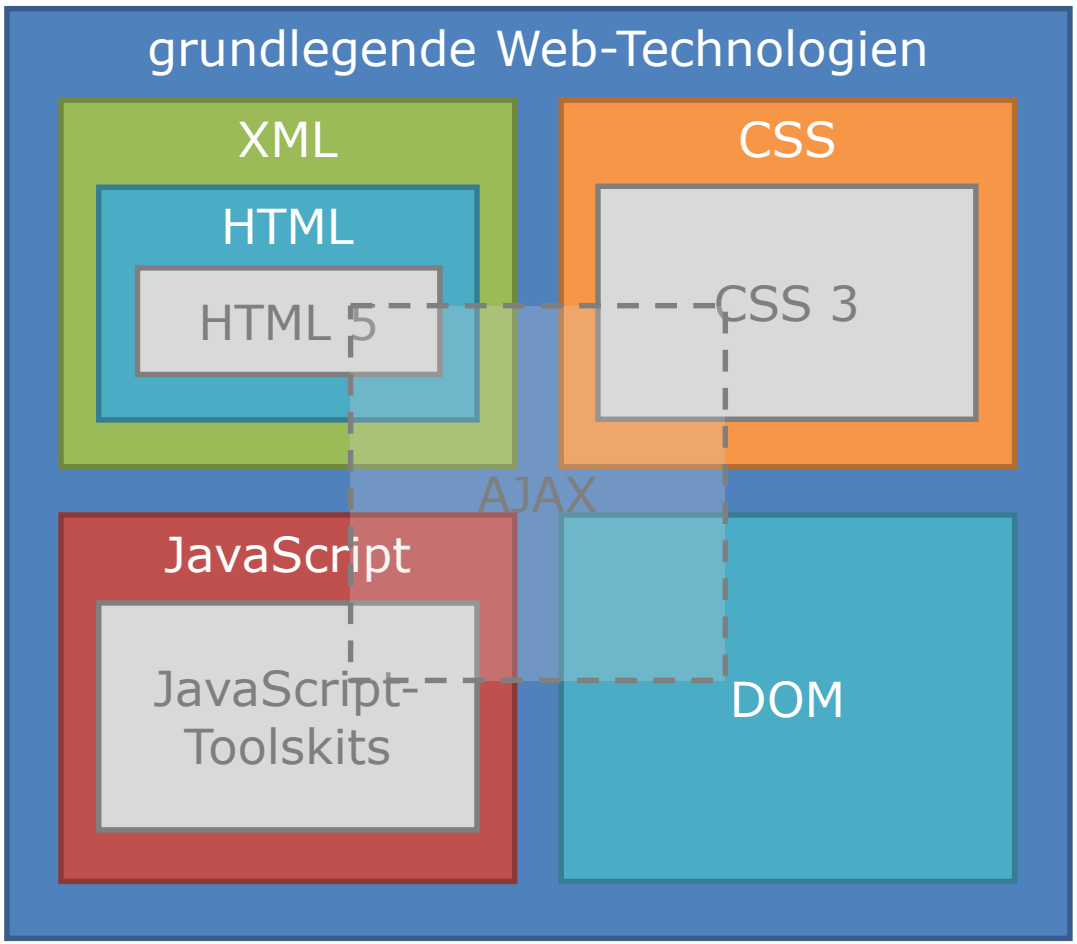
JavaScript & Cookies

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung

Progressive Web Apps

Search Engine Optimization



Inhalte von Web-Seiten

- Bilder
- SVG
- Audio/Video

Proprietäre Standards

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

JavaScript

JavaScript

- eigentlich Programmiersprache zur Erweiterung von HTML und CSS um
  - Interaktivität auf Basis von Nutzeraktionen
  - Dynamik ohne Nutzeraktionen
- heute auch serverseitig und in Micro-Controllern verbreitet
- hat trotz des Namens nichts mit Java zu tun!
  (ist in ECMA 262 als ECMA Script spezifiziert)
  - dynamisch spezifiziert
  - objektorientiert
  - *klassenlos*
- erlaubt prozedurales, funktionales und objektorientiertes Programmieren
  (auch vermischt!)

Anwendungsgebiete

- dynamische Manipulation des DOM
- Datenvalidierung bei Formularen
  (vor/während/nach der Kommunikation mit dem Server)
- Anzeige von Dialogfenstern
- AJAX (⇢Vorlesungskapitel 6)
- Animieren von Werbebannern oder Laufschriften
- Obfuskation (bspw. Verschleierung der E-Mail-Adresse)
- Frame-Manipulation
  - mehrere Frames auf einmal wechseln
  - Seite aus dem Frameset lösen
  - Seite in Frameset einbetten
- Schreib-/Lesezugriff auf Cookies und Web Storage

Sandbox

- Browser führen JavaScript isoliert als Single-Thread in einer Sandbox aus
  - beschränkt Zugriff auf Objekte im Browser
  - beschränkt Zugriff auf DOM von Seiten des selben Web-Servers (Same Origin Policy; SOP) zur Erschwerung von seitenübergreifenden skriptbasierten Angriffen (Cross Site Scripting; XSS)
- einzige definierte Interaktionsmöglichkeit aus Sandbox heraus: Dateidialoge (`<input type="file" />`)
- sonstige Erweiterungen von JavaScript machen Sandbox i.d.R. unsicherer (bspw. Microsoft JScript)
- Sandboxes können unterschiedlich restriktiv sein (bis hin zur Deaktivierung von JavaScript)
  → keine Garantie, dass JavaScript in allen Umgebungen gleich agiert!

Datentypen

- dynamische Typisierung: Typ ist nicht an Variable gebunden, sondern an Wert zur Laufzeit (Ermittlung mittels `typeof`)

- mögliche Datentypen

`String`	primitiv; Zeichenketten
`Number`	primitiv; Zahlen (Ganzzahlen, Gleitkomma)
`Boolean`	primitiv; Wahrheitswerte
`Symbol`	primitiv; Flag/Identifier
`Undefined`	primitiv; nicht definiert (bei Deklaration ohne Instanziierung automatisch zugewiesen)
`Function`	Funktionen und zusammengesetzte Typen
`Object`	RegEx, Array, `null`, zusammengesetzte Typen, mittels String(), Number() oder Boolean() erzeugte Werte

- dynamische Konversion (ähnlich Casting) zur Laufzeit

Deklaration von Variablen

- Variablen werden deklariert mit

  `var` Variable ist
  - nur innerhalb der Funktion in der sie deklariert wurde sichtbar
  - global, wenn außerhalb von Funktionen deklariert

  `let` Variable ist nur im Block in dem sie deklariert wurde sichtbar

  `const` wie mit `var` oder `let`, aber muss mit Wert deklariert werden und kann nach der Deklaration nicht mehr geändert werden

- bei `var` und `let` *können* Variablen Werte oder Funktionen zugewiesen werden

```
<VariableDeclaration> ::=
   "var" | "let" <VariableName> [
      "=" <Value> | <AnonymousFunctionDeclaration>
                  | <NamedFunctionDeclaration>
   ] {
      "," <VariableName> [
        "=" <Value> | <AnonymousFunctionDeclaration>
                    | <NamedFunctionDeclaration>
      ]
   } ";" ;
```

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

JavaScript: Kontrollstrukturen

Kontrollstrukturen – bedingte Verzweigung (1/2)

```
<IfStatement> ::=
   "if" (
      ( "(" <LogicExpression> ")" )
      | <LogicExpression>
   )
      <Statement> | <Block>
   [
      "else"
         <Statement> | <Block>
   ] ;
```

Nur bei Wertzuweisungen:

```
   <ConditionalAssignment> ::=
      <Variable> "=" <LogicExpression>
                 "?" <Value>
                 ":" <Value>
                 ";" ;
```

KONTROLLSTRUKTUREN – BEDINGTE VERZWEIGUNG (2/2)

```
<SwitchStatement> ::=
   "switch" ( ( "(" <Variable> ")" ) | <Variable> ) "{"
      "case" <Value> ":"
         <Statement> | <Block>
         [ "break" ";" ]
      {
      "case" <Value> ":"
         <Statement> | <Block>
         [ "break" ";" ]
      } [
      "default" ":"
         <Statement> | <Block>
      ]
   "}" ;
```

Kontrollstrukturen – Schleifen (1/2)

```
<ForLoop> ::=
   "for" "("
      <Statement> ";" <LogicCondition> ";" <Statement>
   ")"
      <Statement> | <Block> ;

<WhileLoop> ::
   "while" "(" <LogicExpression> ")"
      <Statement> | <Block> ;

<DoWhileLoop> ::=
   "do"
      <Statement> | <Block>
   "while" "(" <LogicExpression> ")" ";" ;
```

KONTROLLSTRUKTUREN – SCHLEIFEN (2/2)

```
<ForInLoop> ::=
   "for" "(" "var" <Property> "in" <Objekt> ")"
     <Statement> | <Block> ;
```

- durchläuft alle eigenen und ererbten Eigenschaften eines Objektes (wenn internes Attribut Enumerable ist)
- angegebene Variable erhält aktuellen Eigenschafts***namen*** des Objekts

```
<ForOfLoop> ::=
   "for" "(" "var" <Value> "of" <Objekt> ")"
     <Statement> | <Block> ;
```

- wie for…in-Schleife, aber angegebene Variable erhält Eigenschafts***wert***
- Objekt muss Eigenschaften haben

Deklaration von Funktionen (1/5)

```
<Parameter> ::=
(* brauchen wir zur Verkürzung der EBNF der
 * unterschiedlichen Funktionsdeklarationen *)
  <ParameterName> [ "=" <Value> ] ;


<NamedFunctionDeclaration> ::=
   "function" <FunctionName> "("
      [ <Parameter> { "," <Parameter> } ]
   ")"
      ( "return" <Statement> ) | <Statement>
      | (
         "{"
            <Statement> { <Statement> }
            [ "return" <Statement> ]
         "}"
      ) ;
```

Deklaration von Funktionen (2/5)

```
<AnonymousFunctionDeclaration> =
(* anonyme Funktionen haben keinen Namen und können
 * (wenn sie nicht einer Variablen zugewiesen werden)
 * nicht aufgerufen werden! *)
  "function" "(" [ <Parameter> { "," <Parameter> } ] ")"
  ( "return" <Statement> ) | <Statement>
  | (
     "{"
       <Statement> { <Statement> }
       [ "return" <Statement> ]
     "}"
  ) ;
```

Deklaration von Funktionen (3/5)

```
<SelfInvokingAnonymousFunctionDeclaration> ::=
(* ausgeführt sobald DOM geladen und „ready“ gemeldet *)
   "("
      "function" "(" [ <Parameter> { "," <Parameter> } ] ")"
      ( "return" <Statement> )
      | <Statement>
      | (
         "{"
            <Statement> { <Statement> }
            "return" <Statement>
         "}"
      )
   ")"
   "("
      [ <Expression> ]
   ")"
   ";" ;
```

Deklaration von Funktionen (4/5)

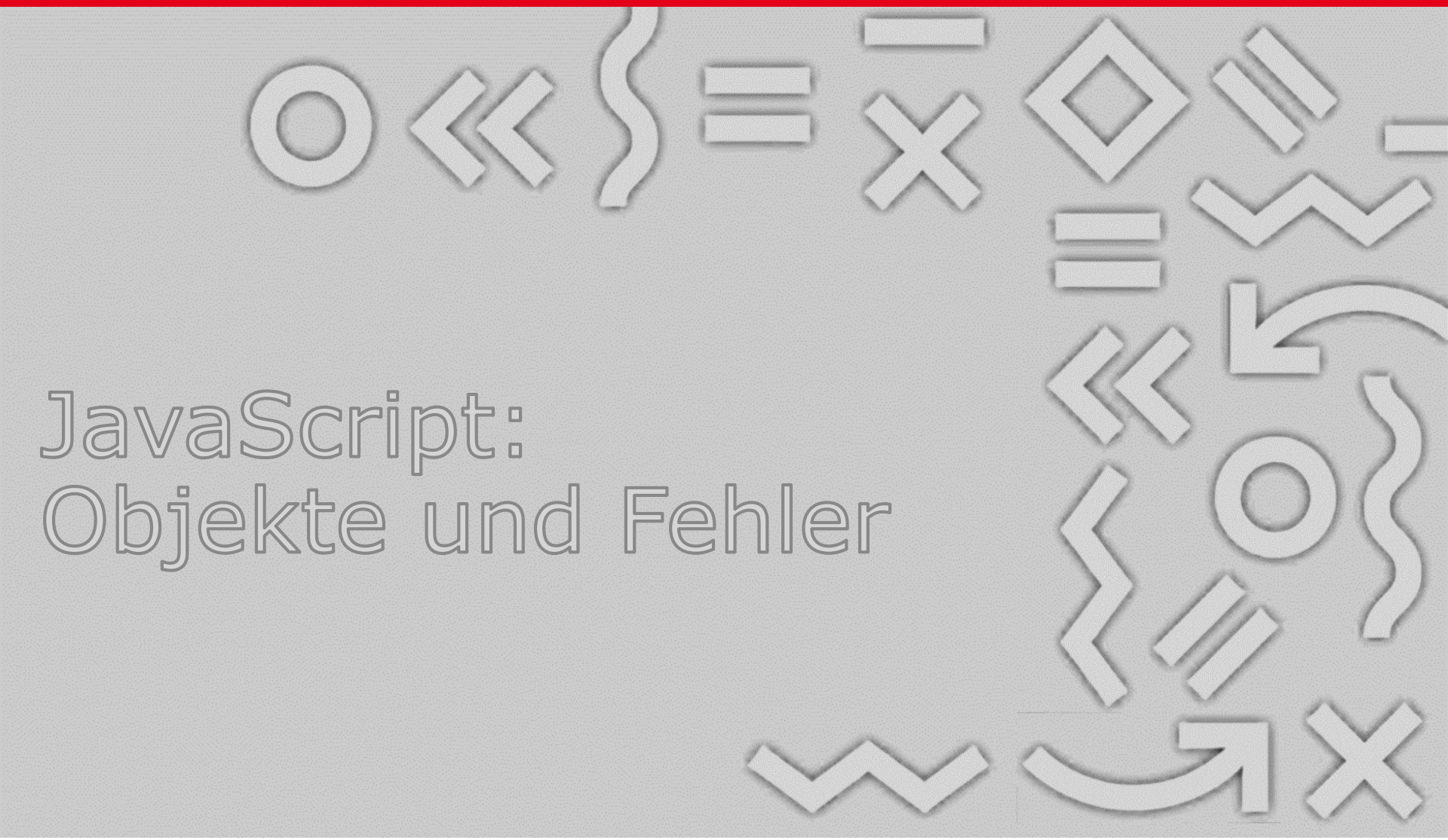
```
<ConstructorFunctionDeclaration> ::=
   "var" | "let" | "const" <VariableName> "="
   "new" "Function" "("
      [ "'" <Parameter> "'" { "," "'" <Parameter> "'" } "," ]
      "'"
         ( "return" <Statement> )
         | <Statement>
         | (
            "{"
               <Statement> { <Statement> }
               "return" <Statement>
            "}"
         )
      "'"
   ")" ";" ;
```

Deklaration von Funktionen (5/5)

```
<LambdaFunctionDeclaration> ::=
(* Lambda-Ausdruck ohne geschweifte Klammern
 * und return-Anweisung
 * → gibt Ergebnis des <Expression> zurück *)
  "function" <FunctionName> "("
     [ <Parameter> { "," <Parameter> } ]
  ")" <Expression> ";" ;


<ArrowFunctionDeclaration> ::=
(* Arrow Functions ab ECMAScript 2015 *)
  <VariableName> | <Parameter>
  | ( "(" <Parameter> { "," <Parameter> } ")" )
  "=>"
  <Expression>
  | ( "{"
     <Statement> { <Statement> }
     "return" <Statement>
  "}" ) ;
```

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION



JavaScript: Objekte und Fehler

OBJEKTE (1/2)

- Konstruktorfunktionen zur Erzeugung von Objektinstanzen
- vordefinierte Objekttypen

`Object`	Grundtyp von dem alle Objekte abgeleitet werden
`Function`	Funktionen
`Array`	Felder
`String`	Zeichenketten
`Boolean`	Wahrheitswerte
`Number`	64-Bit-Gleitkommazahlen gemäß IEEE 754
`Date`	Datumsformate (Daten und Zeitpunkte)
`RegExp`	reguläre Ausdrücke
`Error`	Laufzeitfehler (`throw`-Mechanismus mit `try…catch`)
`Math`	(Singleton) mathematische Konstanten und Methoden
`JSON`	(Singleton) (De)Serialisierung von Objekten von/zu JSON
`Reflect`	(Singleton) Ermittlung/Änderung von Objekt-Metadaten

OBJEKTE (2/2)

- unterschiedliche Notationen möglich
  - Punktnotation

```
objekt.eigenschaft
objekt.methode(parameterliste)
```

  - Klammernotation

```
objekt["eigenschaft"]
objekt["methode"](paramterliste)
```

- Traversion mittels for…in-Schleife

```
for (var eigenschaft in objekt) {
   console.log(eigenschaft, ' = ', objekt[eigenschaft]);
}
```

- Löschen von Eigenschaften/Funktionen jederzeit möglich

```
delete objekt.eigenschaft;
delete objekt["eigenschaft"];
```

Eigene Objekttypen und Sicherbarkeit

- eigene Objekttypen werden über Konstruktorfunktionen definiert
- es gibt in JavaScript keine privaten Eigenschaften
  → stattdessen Funktionsobfuskation

```
var Cat = function () {
    var lifes = 7;
    var mew = function () {
        return (lifes > 0) ? "mew" : "argh";
    };
    return {
        kill: function () {
            lifes -= 1;
            alert(mew());
        }
    };
};
var kitty = Cat();
kitty.kill();
```

namenlose Funktion obfuskiert alle innen deklarierten Variablen und Funktionen

vom Effekt her unsichtbare Variablen und Funktionen (entspricht Sichtbarkeit „private“)

wegen return vom Effekt her öffentliche Methode (entspricht Sichtbarkeit „public“)

Prototyp-basierte Vererbung (1/2)

- kein explizite Vererbung (kein `extends`-Mechanismus)
- stattdessen über nachträglich deklarierte Objekteigenschaft `prototype`

```
var human = {
   motion : ["crawl","creep","roll","run","swim","walk"]
};
var Xmen = function () {
   this.motion = ["crawl","creep","fly","roll","run","swim",
                  "teleport","time-travel","walk"];
};
Xmen.prototype = human;
var DavidHaller = new Xmen();
alert(DavidHaller.motion.length);
// David Haller hat als X-Men „Legion“ 9 Bewegungsarten
delete DavidHaller.motion;
// David Haller erhält den X-Gen-Suppressor
alert(DavidHaller.motion);
// als normaler Mensch hat David Haller 6 Bewegungsarten
```

Prototyp-basierte Vererbung (2/2)

- Prüfung auf Prototyp-Vererbung mittels `hasOwnProperty`

```
var human = {
   motion : ["crawl","creep","roll","run","swim","walk"]
};

// …

alert(DavidHaller.hasOwnProperty('motion'));
// TRUE → David Haller ist ein Mutant. Verhaftet ihn!

delete DavidHaller.motion;

alert(DavidHaller.hasOwnProperty('motion'));
// FALSE → das DoMA interessiert sich nicht für David
Haller
```

- Traits und Mixins via `this` in Funktionen
- Delegation von Objekteigenschaften mittels `call` oder `apply`

Fehlerbehandlung (1/2)

- Fehler können aus der Laufzeit selbst eskalieren
- alternativ kann man Fehler selbst eskalieren („werfen“)

```
throw new Error("Fehlernachricht");
```

- Fehler werden mit `try…catch`-Mechanismus behandelt
- unbehandelte Fehler führen zum (sofortigen) Execution Stop

Fehlerbehandlung (2/2)

```
<ErrorHandling> ::=
   "try"
      <statement> | <block>
   "catch" "(" <ExceptionType> ")"
      <statement> | <block>
   {
      "catch" "(" <ExceptionType> ")"
         <statement> | <block>
   } [
      (* falls angegeben, wird der „finally“-Block immer
       * ausgeführt, unabhängig vom Fehlerfall *)
      "finally"
         <statement> | <block>
   ] ;
```

JavaScript und HTML

- JavaScript-Kode kann überall im HTML-Dokument eingebunden werden
  - inline im Header oder Body

```
<script type="text/javascript">
   alert("Hallo Welt!");
</script>
```

  - extern im Header oder Body

```
<script type="text/javascript" src="Kode.js"></script>
```

- JavaScript-Kode wird ausgeführt
  - im Header: sofort nachdem der HTML-Quellkode vollständig ausgewertet wurde
  - im Body: sofort nachdem die entsprechende HTML-Quellkode-Stelle ausgewertet wurde
  - nacheinander in der Reihenfolge wie der JavaScript-Kode im HTML-Quellkode steht

JavaScript-Tutorial

ein Tutorial zum eigenverantwortlichen Lernen finden Sie unter:

https://www.jshero.net/home.html

Cookies

PERSISTIEREN

- ohne Persistieren der Daten gehen diese nach Schließen des HTML-Dokumentes verloren
- sitzungsübergreifende Informationen
  - Login-Status
  - Nutzerrechte
  - Voreinstellungen
  - Aktivitätshistorie
  - …

Cookies

- Erzeugung auf Server oder Client
  - Server: Übertragung im HTTP-Header an den Client
  - Client: durch JavaScript oder weitere Skriptsprachen
    (das Skript ist i.d.R. in der vom Server übermittelten Webseite)
- werden lokal auf dem Client gespeichert
- vom Browser verwaltet
- besonders zu schützen (XSS, CSRF, …)
- können mit Requests als Datagramm an den Server gesendet werden
  → ermöglicht unterbrechbare, fortsetzbare Sitzungen

AUFBAU

- als zusätzliche Zeile im HTTP-Header
- Setzen:

```
"Set-Cookie:" <Name> "=" <Wert> { ";" <Attribut>}
```

- Verwenden:

```
"Cookie:" <Name> "=" <Wert> { ";" <Name> "=" <Wert>}
```

- erlaubter Zeichenvorrat entsprechend RFC-2616

Beispiel: ACL-Cookie

Server sendet signiertes Cookie mit ACL-Rechten:

```
HTTP/1.1 200 OK↵
Set-Cookie: sessionID=P3X112llIGF1ZmJhdQ==;
  expires=Sun, 2-Jun-2019 21:01:17 GMT;
  acl=/gallery/hara/|rwx,/gallery/|r-x,/shared/|rwx,*|---;
  signature=79dbeb10d9314e83853e54f2ccebafceabb6edfa1f836a4676
  02d171978cda5868e496ab469618637e3d872b562a2ced08cb66db2b5f3e
  a8356ec2112dddf30d↵
```

Zugriff auf Cookies

- Cookies lesen:

```
allCookies = document.cookie;
```

Achtung! Liefert Semikolon-separierten String mit allen Cookies!

Zugriff auf ein bestimmtes Cookie bspw. so:

```
function getCookie(name) {
  var v = document.cookie.match(
            '(^|;) ?' + name + '=([^;]*)(;|$)');
  return v ? v[2] : null;
}
```

- Cookies schreiben:

```
document.cookie = "name=wert";
```

- Cookies löschen:

```
document.cookie = "name=";
```

Browser-Sicherheit

Cookies

Cookies stellen ein nicht unerhebliches Sicherheitsrisiko dar

- Verfolgung über Grenzen der Webseiten hinweg
- Profilbildung (Profiling)
- Session-Hijacking

→ Problem sind insbesondere Drittanbieter-Cookies

Google forciert in Chrome Push-out von Drittanbieter-Cookies

→ Herausforderung für Dienstanbieter (insb. Werbung)

JavaScript

JavaScript stellt ein nicht unerhebliches Sicherheitsrisiko dar

- Berechnungen im Hintergrund
  (z.B. nicht autorisiertes Bitcoin-Mining)
- fortgesetzte Berechnungen im Hintergrund
  (z.B. laufen WebWorker weiter, wenn Browser „geschlossen“ ist)
- Profiling von Nutzern möglich
  (Browser Profiling; ohne Cookies möglich wegen der Art, wie Berechnungen durchgeführt werden und auf bestimmte Fähigkeitsanfragen reagiert wird)
  https://www.zenrows.com/blog/browser-fingerprinting#bypass-browser-fingerprinting

→ Inzwischen auch bei deaktiviertem JavaScript möglich!

Was ist ein sicherer Browser?

- verschiedenste Definitionen von Browser-Sicherheit
- hängt von Perspektive ab
- in Deutschland für den KRITIS-Bereich beim BSI definiert https://www.bsi.bund.de/DE/Themen/Oeffentliche-Verwaltung/Mindeststandards/Webbrowser/Webbrowser_node.html