Web-Entwicklung

Jil Zerndt, Lucien Perret
December 2024

Browser-Technologien

Vordefinierte Objekte

- Die allgemeinen Objekte sind in JavaScript vordefiniert
- Tatsächlich handelt es sich um Funktionen/Konstruktoren
- Die Browser-Objekte existieren auf der Browser-Plattform
- Sie beziehen sich auf das Browser-Fenster, das angezeigte Dokument, oder den Browser selbst
  document
- Repräsentiert die angezeigte Webseit
- Einstieg ins DOM (Document Object Model)
- Diverse Attribute und Methoden, zum Beispiel:

```
1 document.cookie /* Zugriff auf Cookies */
2 document.lastModified /* Zeit der letzten Änderung */
3 document.links /* die Verweise der Seite */
4 |document.images /* die Bilder der Seite */
```

window

- Repräsentiert das Browserfenster
- Zahlreiche Attribute und Methoden, u.a.:
- Alle globalen Variablen und Methoden sind hier angehängt
- Neue globale Variablen landen ebenfalls hier

```
1 window.document /* Zugriff auf Dokument */
2 window.history /* History-Objekt */
3 window.innerHeight /* Höhe des Viewports */
4 window.pageYOffset /* vertikal gescrollte Pixel */
5 window.alert === alert /* -> true */
6 window.setTimeout === setTimeout /* -> true */
7 window.parseInt === parseInt /* true */
```

navigator

Konsolen-eingabe auf dem folgenden Bild:

```
> navigator.userAgent
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101 Firefox/91.0"
> navigator.language
"de"
> navigator.platform
"MacIntel"
> navigator.onLine
true
location
> location.href
"https://gburkert.github.io/selectors/"
> location.protocol
"https:"
> document.location.protocol
"https:"
```

JavaScript und DOM

- Element erzeugen: document.createElement
- Attribute erzeugen: document.createAttribute
- Und hinzufügen: .setAttribute
- Element in Baum einfügen: .appendChild

Element auffinden

```
1 let aboutus = document.getElementById("aboutus")
2 let aboutlinks = aboutus.getElementsByTagName("a")
3 let aboutimportant = aboutus.getElementsByClassName("important")
4 let navlinks = document.querySelectorAll("nav a")
```

Textknoten erzeugen

```
<p>The <img src="img/cat.png" alt="Cat"> in the
<img src="img/hat.png" alt="Hat">.</p>
<p><button onclick="replaceImages()">Replace</button></p>
<script>
    function replaceImages () {
        let images = document.body.getElementsByTagName("img")
        for (let i = images.length - 1; i >= 0; i--) {
                let image = images[i]
                if (image.alt) {
                let text = document.createTextNode(image.alt)
                image.parentNode.replaceChild(text, image)
                }
            }
    }
</script>
```

Elementknoten erzeugen

```
<blockquote id="quote">
    No book can ever be finished. While working on it we learn
</blockquote>
<script>
/* definition of elt ... */
document.getElementById("quote").appendChild(
    elt("footer", "-",
    elt("strong", "Karl Popper"),
    ", preface to the second edition of ",
    elt("em", "The Open Society and Its Enemies"),
    ", 1950"))
</script>
```

Attribut setzen

1
6
8

```
3 let att = document.createAttribute("class")
4 att.value ="democlass"
5 h1.setAttributeNode(att)
7 /* oder kürzer: */
let h1 = document.querySelector("h1")
,h1.setAttribute("class","democlass")
```

Style anpassen

1 Nice text
2

Event handling

Event abonnieren/entfernen

Mit der Methode addEventListener() wird ein Event abonniert. Mit removeEventListener kann das Event entfernt werden.

```
<button>Act-once button</button>
<script>
    let button = document.querySelector("button")
    function once () {
        console.log("Done.")
        button.removeEventListener("click", once)
    }
    button.addEventListener("click", once)
</script>
```

Wenn ein Parameter zur Methode hinzugefügt wird, wird dieses als das Event-Objekt gesetzt.

```
<script>
    let button = document.querySelector("button")
    button.addEventListener("click", (e) => {
        console.log("x="+e.x+", y="+e.y)
    })
</script\
```

Das Event wird bei allen abonnierten Handlern ausgeführt bis ein Handler stopPropagation() ausführt.

```
<script>
    let button = document.querySelector("button")
    button.addEventListener("click", (e) => {
                console.log("x="+e.x+", y="+e.y)
...             e.stopPropagation()
    })
</script>
```

Viele Ereignisse haben ein Default verhalten. Eigene Handler werden vor Default-Verhalten ausgeführt. Um das Default-Verhalten zu verhindern, muss die Methode preventDefault() ausgeführt werden.

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
    let link = document.querySelector("a")
    link.addEventListener("click", event => {
            console.log("Nope.")
                event.preventDefault()
        })
,/script>
```

Tastatur-Events

- keydown
- keyup
- Achtung: bei keydown kann das event mehrfach ausgelöst werden

```
<p>Press Control-Space to continue.</p>
<script>
    window.addEventListener("keydown", event => {
            if (event.key == " " && event.ctrlKey) {
                console.log("Continuing!")
            }
    })
</script>
```

Mauszeiger-Events

- Mausklicks:
- mousedown
- mouseup
- click
- dblclick
- Mausbewegung
- mousemove
- Touch-display
- touchstart
- touchmove
- touched

Scroll-Events

Das Scrollevent hat die Attribute des Event-Objekts: pageYOffset, pageXOffset.

```
window.addEventListener("scroll", () => {
    let max = document.body.scrollHeight - innerHeight
    bar.style.width = `${(pageYOffset / max) * 100}%`
,})
```

Fokus- und Ladeereignisse
- Fokus erhalten / verlieren
- focus
- blur
- Seite wurde geladen (ausgelöst auf window und document.body)
- load
- beforeunload

Jquery
JQuery ist eine freie JavaScript-Bibliothek, die Funktionen zur DOM-Navigation und -Manipulation zur Verfügung stellt.

```
$("button.continue").html("Next Step...")
var hiddenBox = $("#banner-message")
$("#button-container button").on("click", function(event) {
        hiddenBox.show()
    .})
```

Aufruf	Bedeutung
$(Funktion)	DOM ready
$("CSS Selektor") .aktion(arg1, ...) .aktion(...)	Wrapped Set
	- Knoten, die Sel. e
	- eingepackt in jQue
$("HTML-Code")	Wrapped Set - neuer Knote
	- eingepackt in jQue - noch nicht im DO
$(DOM-Knoten)	Wrapped Set
	- dieser Knoten - eingepackt in jQue

Web-Grafiken

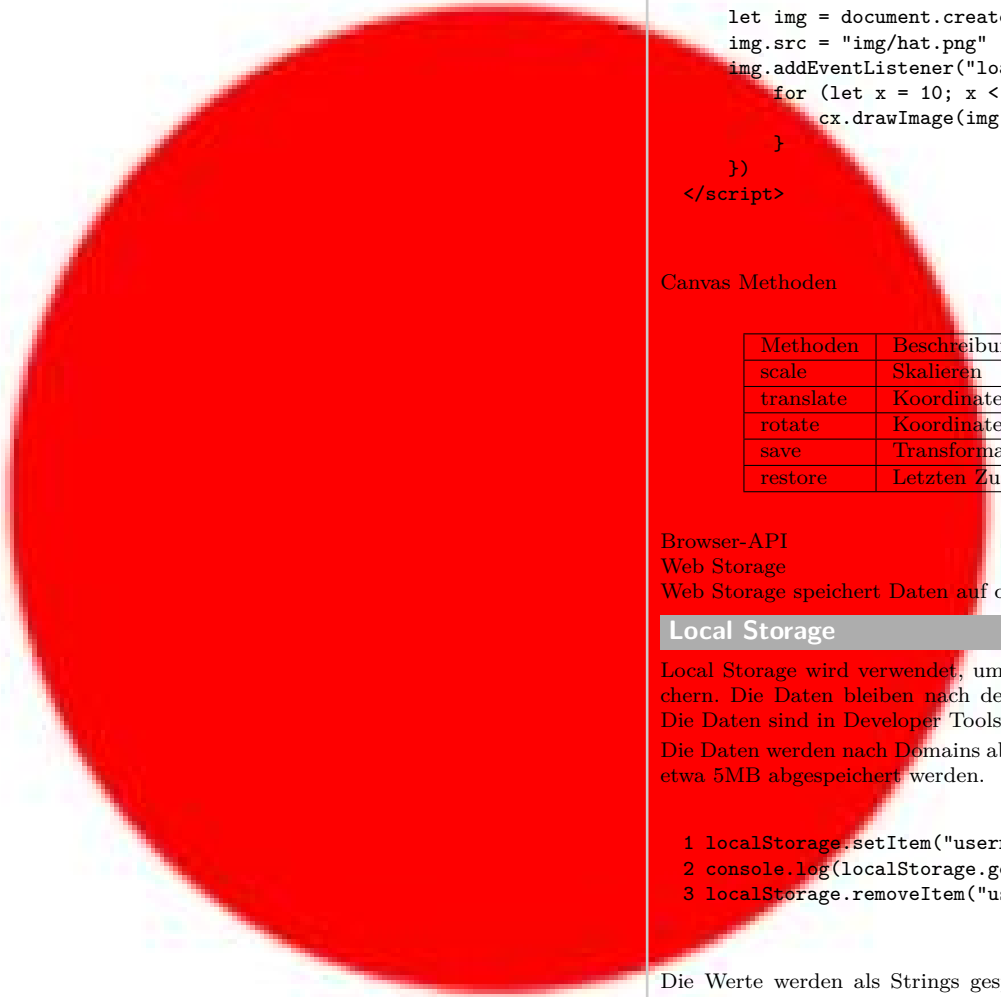
- Einfache Grafiken mit HTML und CSS möglich
- Zum Beispiel: Balkendiagramme
- Alternative für Vektorgrafiken: SVG
- Alternative für Pixelgrafiken: Canvas

SVG
- Basiert wie HTML auf XML
- Elemente repräsentieren grafische Formen
- Ins DOM integriert und durch Scripts anpassbar

Beispiel:

```
p>Normal HTML here.</p>
<svg xmlns="http://www.w3.org/2000/svg">
    <circle r="50" cx="50" cy="50" fill="red"/>
    <rect x="120" y="5" width="90" height="90" stroke="blue" fill="none"/>
</svg>
```

Ausgabe:
Normal HTML here.



JavaScript:

```
1 let circle = document.querySelector("circle")
2 circle.setAttribute("fill","cyan")
```

Canvas

- Element canvas als Zeichenbereich im Dokument
- API zum Zeichnen auf dem Canvas

```
<canvas></canvas>
<script>
    Let cx = document.querySelector("canvas").getContext("2d")
    cx.beginPath()
    cx.moveTo(50, 10)
    cx.lineTo(10, 70)
    cx.lineTo(90, 70)
    cx.fill()
    let img = document.createElement("img")
    img.src = "img/hat.png"
    img.addEventListener("load" , () => {
        for (let x = 10; x < 200; x += 30) {
            cx.drawImage(img, x, 10)
        }
    })
</script>
```

Canvas Methoden

Methoden	Beschreibung
scale	Skalieren
translate	Koordinatensystem verschieben
rotate	Koordinatensystem rotieren
save	Transformationen auf Stack speichern
restore	Letzten Zustand wiederherstellen

Browser-API
Web Storage
Web Storage speichert Daten auf der Seite des Client.

Local Storage

Local Storage wird verwendet, um Daten der Webseite lokal abzuspeichern. Die Daten bleiben nach dem Schliessen des Browsers erhalten. Die Daten sind in Developer Tools einsehbar und änderbar.

Die Daten werden nach Domains abgespeichert. Es können pro Webseite etwa 5MB abgespeichert werden.

```
1 localStorage.setItem("username","bkrt")
2 console.log(localStorage.getItem("username")) // -> bkrt
3 localStorage.removeItem("username")
```

Die Werte werden als Strings gespeichert, daher müssen Objekte mit JSON codiert werden:
1 Let user = {name: "Hans", highscore: 234}
2 localStorage.setItem(JSON.stringify(user))

History

History gibt zugriff auf den Verlauf des akutellen Fensters/Tab.

```
1 function goBack() {
2 window.history.back();
3
    ,}
```

Methoden	Beschreibung
length (Attribut)	Anzahl Einträgte inkl. aktueller Seite. Keine Methode!
back	zurück zur letzten Seite

GeoLocation
Mit der GeoLocation-API kann der Standort abgefragt werden.

```
var options = { enableHighAccuracy: true, timeout: 5000, maximu
function success(pos) {
    var crd = pos.coords
    console.log(`Latitude : ${crd.latitude}`)
    console.log(`Longitude: ${crd.longitude}`)
    console.log(`More or less ${crd.accuracy} meters.`)
}
function error(err) { ... }
navigator.geolocation.getCurrentPosition(success, error, option
```

Client-Server-Interaktion

Formulare

Formulare ermöglichen Benutzereingaben. Sie gilt als Grundlade für Interaktion mit dem Web.
Input types:

- submit, number, text, password, email, url , range , date , search , color

```
<form>
    <fieldset>
        <legend>General information</legend>
        <label>Text field <input type="text" value="hi"></label>
        <label>Password <input type="password" value="hi"></label
        <label class="area">Textarea <textarea>hi</textarea></lab
    </fieldset>
    <fieldset>
        <legend>Additional information</legend>
        <label>Checkbox <input type="checkbox"></label>
        <label>Radio button <input type="radio" name="demo" checked></label>
        <label>Another one <input type="radio" name="demo"></label>
    </fieldset>
    <form>
    <label>Button <button>Click me</button></label>
    <label>Select menu
    <select name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
    </select>
    </label>
    <input type="submit" value="Send">
</form>
|'
```

Formulare können auch POST/GET Aktionen ausführen:
Action beschreibt das Skript, welches die Daten annimmt. Method ist die Methode die ausgeführt wird.

```
<form action="/login" method="post">
2 ...
3 </form>
```

Formular Events

Events
change
input
submit

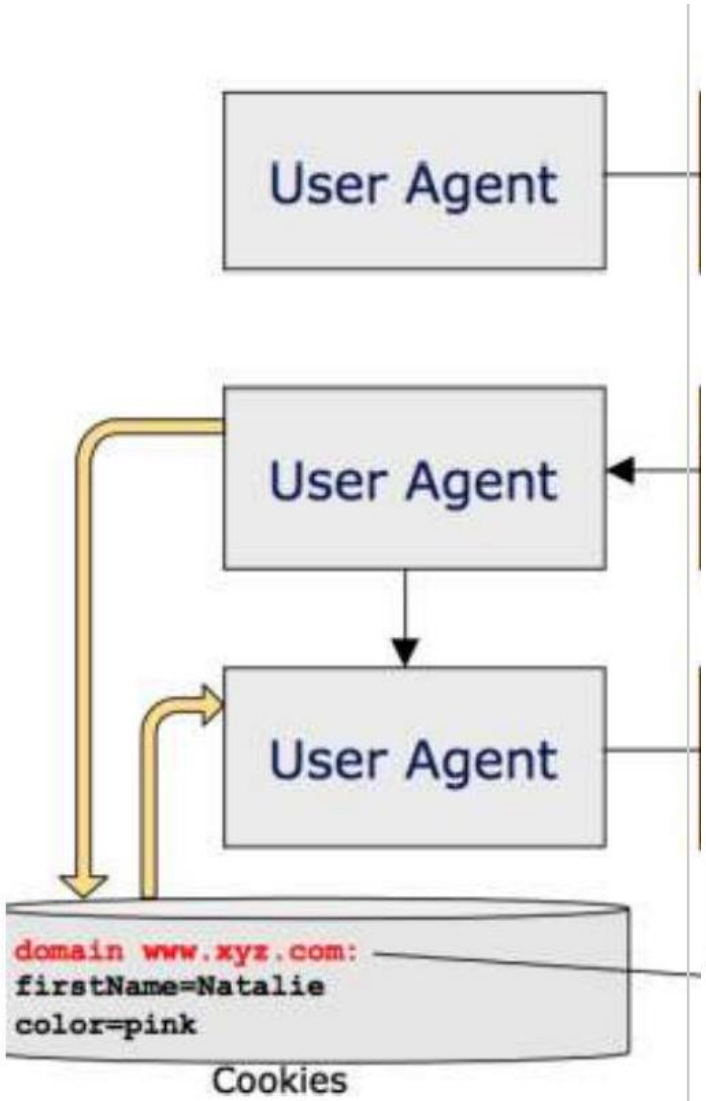
GET/POST-Methode

```
1   <form action="http://localhost/c
2       <fieldset>
3           <legend>Login mit GET</l
4           <label for="login_get">B
5           <input type="text" id="l
6           <label for="password_get
7           <input type="password" i
8           <label for="submit_get">
9           <input type="submit" id=
10      </fieldset>
11  </form>
```

Cookies und Sessions

Cookies

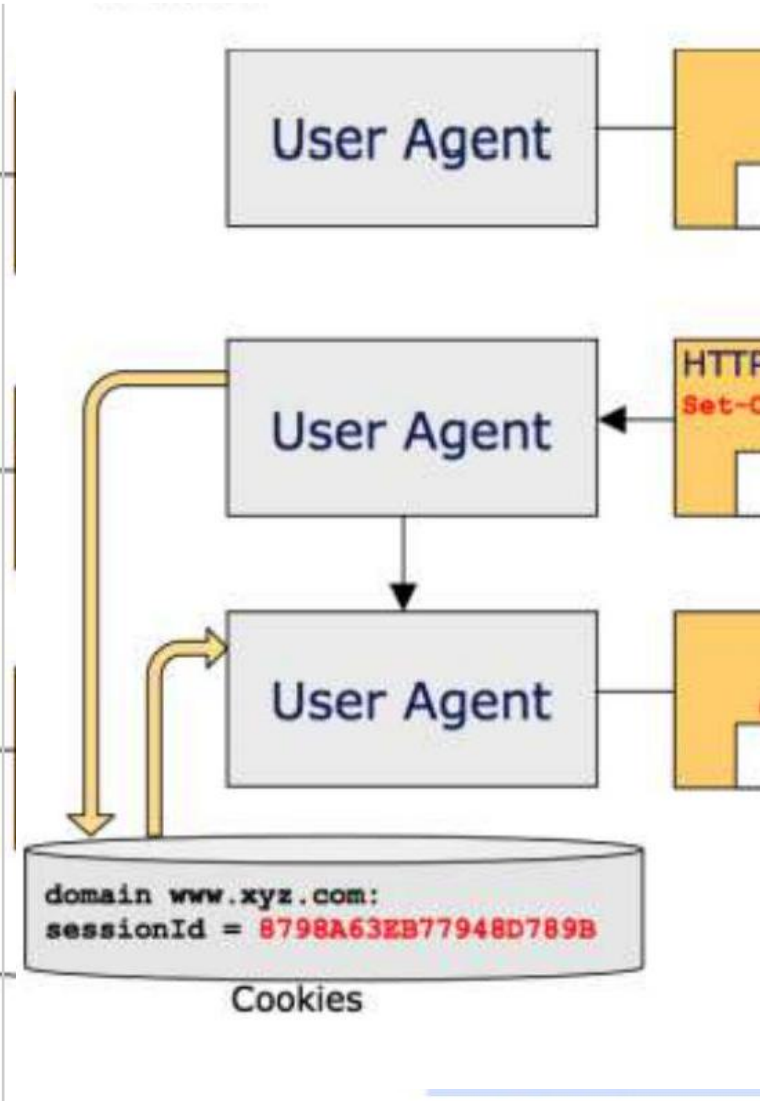
- HTTP als zustandsloses Protokoll konzipiert
- Cookies: Speichern von Informationen auf dem Client
- Response: Set-Cookie -Header, Request: Cookie -Header
- Zugriff mit JavaScript möglich (ausser HttpOnly ist gesetzt)



Sessions

- Cookies auf dem Client leicht manipulierbar
- Session: Client-spezifische Daten auf dem Server speichern
- Identifikation des Clients über Session-ID (Cookie o.a.)
- Gefahr: Session-ID gerät in falsche Hände (Session-Hijacking)

Ablauf:
http://www.xyz.com



Fetch API

- HTTP-Requests von JavaScripts
- Geben Promise zurück
- Nach Server-Antwort erfüllt mit Response-Objekt

```
fetch("example/data.txt")
.then(response => {
            console.log(response.status) // -> 200
    console.log(response.headers.get("Content-Type")) // -> text/p
})
.then(resp => resp.text())
.then(text => console.log(text))
// -> This is the content of data.txt
```

Response Objekt

- headers : Zugriff auf HTTP-Header-Daten Methoden get, keys, forEach , ...

- status: Status-Code
- json() : liefert Promise mit Resultat der JSON-Verarbeitung
- text() : liefert Promise mit Inhalt der Server-Antwort