WBE: BROWSER-TECHNOLOGIEN

JAVASCRIPT IM BROWSER

ÜBERSICHT

- JavaScript im Browser
- Document Object Model
- Event Handling im Browser
- Kleiner Exkurs: jQuery
- SVG, Canvas und Web-Storage

ÜBERSICHT

- JavaScript im Browser
- Document Object Model
- Event Handling im Browser
- Kleiner Exkurs: jQuery
- SVG, Canvas und Web-Storage

JAVASCRIPT IM BROWSER

- Ohne Browser gäbe es kein JavaScript
- Für den Einsatz im Browser entwickelt
- Brendan Eich, 1995: Netscape Navigator

HTML UND JAVASCRIPT

- Element `script` (End-Tag notwendig)
- Vom Browser beim Lesen des HTML-Codes ausgeführt
- Oder Code als Reaktion auf Ereignis ausführen

```
<!-- Code ausführen -->
<script>alert("hello!")</script>

<!-- Code aus JavaScript-Datei ausführen -->
<script src="code/hello.js"></script>

<!-- Code als Reaktion auf Ereignis ausführen -->
<button onclick="alert('Boom!')">DO NOT PRESS</button>
```

HTML UND JAVASCRIPT

- Laden von ES-Modulen möglich
- Angabe von `type="module"`

```
<script type="module" src="code/date.js"></script>
```

https://eloquentjavascript.net/10_modules.html#h_hF2FmOVxw7

SANDBOX

- Ausführen von Code aus dem Internet ist potentiell gefährlich
- Möglichkeiten im Browser stark eingeschränkt
- Zum Beispiel kein Zugriff auf Filesystem, Zwischenablage etc.
- Trotzdem häufig Quelle von Sicherheitslücken
- Abwägen: Nützlichkeit vs. Sicherheit

Sicherheitslücken werden meist schnell geschlossen.
Immer die neuesten Browser-Versionen verwenden.

ÜBERSICHT

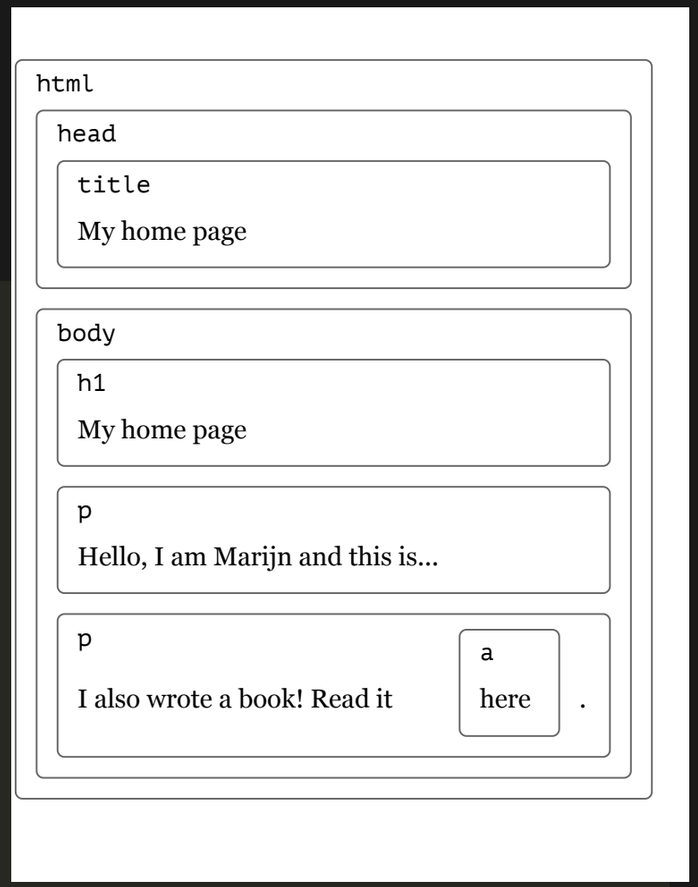
- JavaScript im Browser
- Document Object Model
- Event Handling im Browser
- Kleiner Exkurs: jQuery
- SVG, Canvas und Web-Storage

WEBSITE IM BROWSER-SPEICHER

- Browser parst HTML-Code
- Baut ein Modell der Dokumentstruktur auf
- Basierend auf dem Modell wird die Seite angezeigt
- Auf diese Datenstruktur haben Scripts Zugriff
- Anpassungen daran wirken sich live auf die Anzeige aus

BEISPIEL

```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home
      page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```



DOCUMENT OBJECT MODEL (DOM)

- Jeder Knoten im Baum durch ein Objekt repräsentiert
- Bezeichnung: Document Object Model (DOM)
- Zugriff über das globale Objekt `document`
- Attribut `documentElement` ist Referenz auf HTML-Knoten
- Zahlreiche Attribute und Methoden

ELEMENTKNOTEN BODY

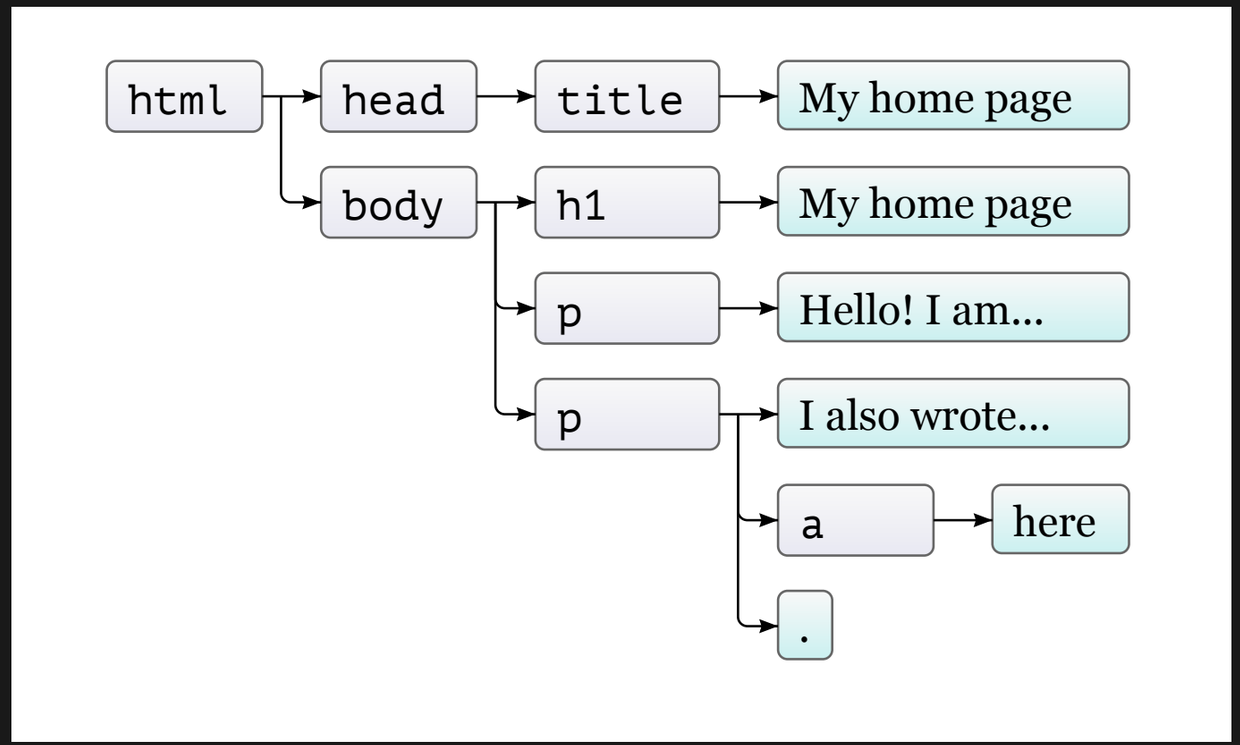
```
▼ 1:                          <body>
    aLink:                    ""
    accessKey:                ""
    accessKeyLabel:           ""
    assignedSlot:             null
    attributes:               NamedNodeMap []
    background:               ""
  ▶ baseURI:                  "file:///Users/Shared/Dis…/08-client-js/demo.html"
    bgColor:                  ""
    childElementCount:        3
  ▶ childNodes:               NodeList(7) [ #text , h1 , #text , … ]
  ▼ children:                 HTMLCollection { 0: h1 , 1: p , length: 3, … }
    ▶ 0:                      <h1>
    ▶ 1:                      <p>
    ▶ 2:                      <p>
      length:                 3
    classList:                DOMTokenList []
    className:                ""
```

BAUMSTRUKTUR

- Jeder Knoten hat ein `nodeType`-Attribut
- HTML-Elemente haben den `nodeType` 1

NodeType	Konstante	Bedeutung
1	Node.ELEMENT_NODE	Elementknoten
3	Node.TEXT_NODE	Textknoten
8	Node.COMMENT_NODE	Kommentarknoten

BAUMSTRUKTUR



DOM ALS STANDARD

- Im Laufe der Jahre gewachsen
- Sprachunabhängig konzipiert
- Zahlreiche Redundanzen
- Kein klares und verständliches Design
- Ehrlich gesagt: ziemlich unübersichtlich

ATTRIBUT CHILDNODES

- Instanz von `NodeList`
- Array-ähnliches Objekt (aber kein Array)
- Numerischer Index und `length`-Attribut
- Alternative: `children`-Attribut
    - Instanz von `HTMLCollection`
    - enthält nur die untergeordneten Elementknoten

```
childNodes:                NodeList(7) [ #text , h1 , #text , … ]
children:                  HTMLCollection { 0: h1 , 1: p , length: 3, … }
  0:                       <h1>
  1:                       <p>
  2:                       <p>
  length:                  3
```

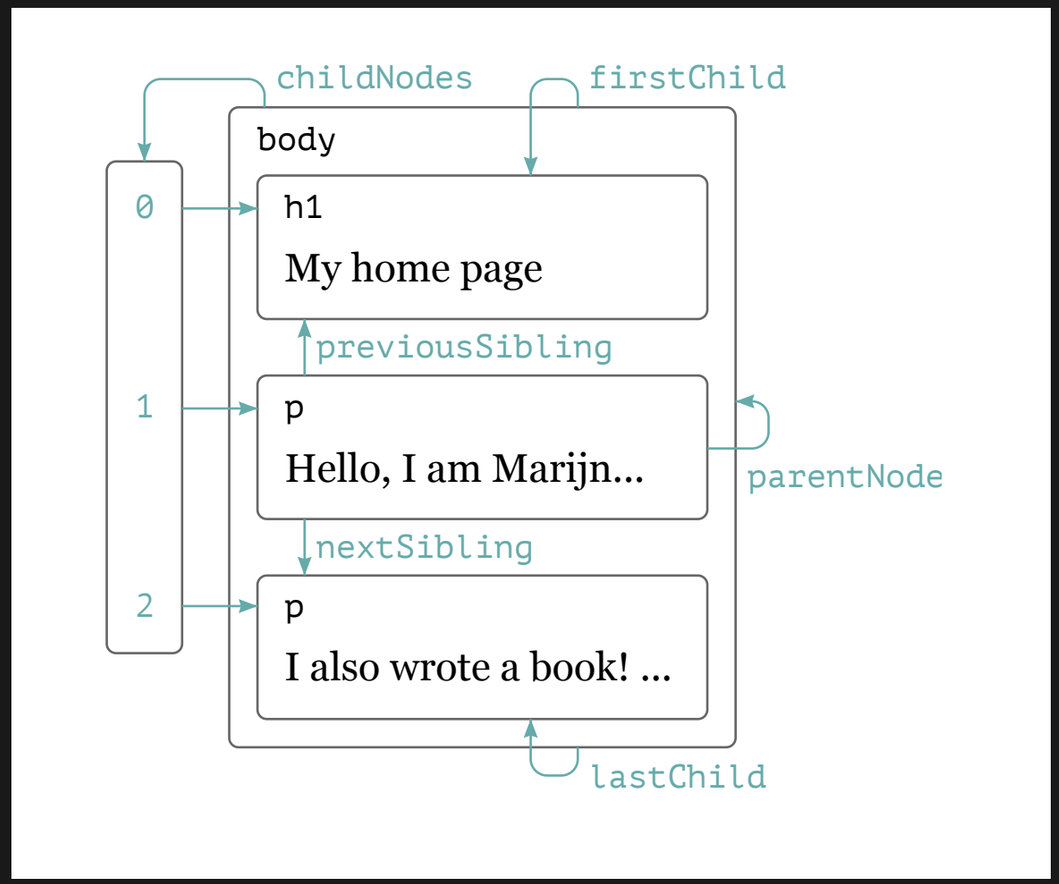
ELEMENT HINZUFÜGEN

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

„Code that interacts heavily with the DOM tends to get long, repetitive, and ugly.” (Eloquent JavaScript)

Gut dagegen: JavaScript erlaubt es, problemlos eigene Abstraktionen zu definieren

BAUMSTRUKTUR ABARBEITEN



- Diverse Attribute und Methoden zur Navigation im DOM-Baum
- Häufig: Array-ähnliche Objekte

BEISPIEL

```
// scans a document for text nodes containing a given string and
// returns true when it has found one
function talksAbout (node, string) {
  if (node.nodeType == Node.ELEMENT_NODE) {
    for (let i = 0; i < node.childNodes.length; i++) {
      if (talksAbout(node.childNodes[i], string)) {
        return true
      }
    }
    return false
  } else if (node.nodeType == Node.TEXT_NODE) {
    return node.nodeValue.indexOf(string) > -1
  }
}

console.log(talksAbout(document.body, "book"))
// → true
```

ELEMENTE AUFFINDEN

```
let aboutus = document.getElementById("aboutus")
let aboutlinks = aboutus.getElementsByTagName("a")
let aboutimportant = aboutus.getElementsByClassName("important")

let navlinks = document.querySelectorAll("nav a")
```

- Gezielte Suche im ganze Dokument oder Teilbaum
- Zum Beispiel alle Elemente mit bestimmtem Tagnamen
- Oder nach bestimmtem Wert des `id`- oder `class`-Attributs
- Alternativ mit Hilfe eines CSS-Selektors

DOKUMENT ANPASSEN

- Diverse Methoden zum Knoten entfernen, einfügen, löschen oder verschieben
- Zum Beispiel: `appendChild`, `remove`, `insertBefore`

```
<p>One</p>
<p>Two</p>
<p>Three</p>

<script>
  let paragraphs = document.body.getElementsByTagName("p")
  document.body.insertBefore(paragraphs[2], paragraphs[0])
</script>
```

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>
```

ARRAYS

- Datenstrukturen im DOM sind häufig Array-ähnlich
- Sie haben Zahlen sowie `length` als Attribute
- Mit `Array.from` können sie in echte Arrays konvertiert werden

```
let arrayish = {0: "one", 1: "two", length: 2}
let array = Array.from(arrayish)
console.log(array.map(s => s.toUpperCase()))
// → ["ONE", "TWO"]
```

ELEMENTKNOTEN ERZEUGEN

```
function elt (type, ...children) {
  let node = document.createElement(type)
  for (let child of children) {
    if (typeof child != "string") node.appendChild(child)
    else node.appendChild(document.createTextNode(child))
  }
  return node
}
```

- Element mit Typ (1. Argument) erzeugen
- Kindelemente (weitere Argumente) hinzufügen

ATTRIBUTE

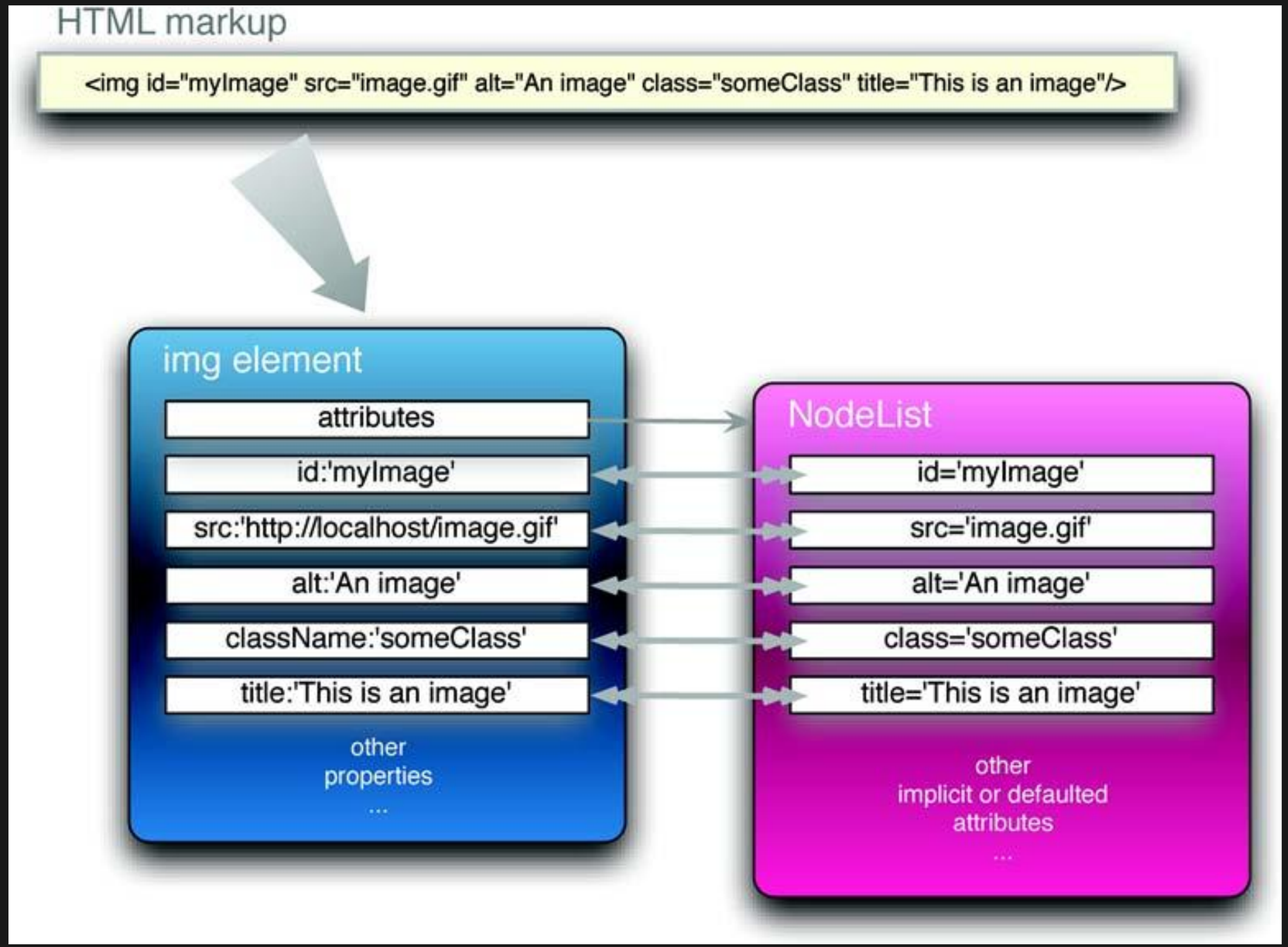
- Viele HTML-Attribute entsprechen Attributen im DOM
- Beispiel: `href`-Attribut des `a`-Elements

```
<a href="http://eloquentjavascript.net">here</a>
```

DOM:

```
a-element
  accessKey: ""
  accessKeyLabel: ""
  attributes: NamedNodeMap [ href="http://eloquentjavascript.net" ]
  childNodes: NodeList [ #text ]
  children: HTMLCollection { length: 0 }
  classList: DOMTokenList []
  className: ""
  ...
  href: "http://eloquentjavascript.net/"
  ...
```

ATTRIBUTE



EIGENE ATTRIBUTE

- Beginnen mit "data-"
- DOM-Attribut `dataset` liefert DOMStringMap mit allen `data`-Attributen

```
<p data-classified="secret">The launch code is 00000000.</p>
<p data-classified="unclassified">I have two feet.</p>

<script>
  let paras = document.body.getElementsByTagName("p")
  for (let para of Array.from(paras)) {
    if (para.dataset.classified == "secret") {
      para.remove()
    }
  }
</script>
```

ATTRIBUT CLASS

- Mehrere Klassen durch Leerzeichen getrennt möglich
- Im DOM zugreifbar über `className` oder `classList`
- Achtung: `className` statt `class` (reservierter Name)

```
<p class="hint info">I also wrote a book!</p>
```

DOM:

```
...
classList     DOMTokenList [ "hint", "info" ]
className     "hint info"
...
```

LAYOUT

- Browser positioniert Elemente im Viewport
- Grösse und Position ebenfalls in DOM-Struktur eingetragen
- `clientWidth`: Breite von Blockelementen inkl. Padding
- `offsetWidth`: Breite inkl. Border
- Einheit: Pixel (`px`)
- Beispiel:

```
clientHeight   19
clientLeft     0
clientTop      0
clientWidth    338
```

```
offsetHeight   19
offsetLeft     8
offsetParent   <body>
offsetTop      116
offsetWidth    338
```

PERFORMANZ

- Layout einer Seite aufbauen ist zeitaufwendig
- Konsequenz: Seitenänderungen via DOM möglichst zusammenfassen
- Beispiel: Warum ist folgende Sequenz ungünstig?

```
let target = document.getElementById("one")
while (target.offsetWidth < 2000) {
  target.appendChild(document.createTextNode("X"))
}
```

DARSTELLUNG

- Attribut `style` (HTML und DOM)
- Wert ist ein String (HTML) bzw. ein Objekt (DOM)
- HTML: CSS-Eigenschaften mit Bindestrich: `font-family`
- DOM: CSS-Eigenschaften in „Camel Case”: `fontFamily`

```
<p id="para" style="color: purple">Nice text</p>

<script>
  let para = document.getElementById("para")
  console.log(para.style.color)
  para.style.color = "magenta"
</script>
```

ANIMATION

- Anpassen zum Beispiel der `position`-Eigenschaft
- Synchronisieren mit der Browser-Anzeige: `requestAnimationFrame`

```
function animate(time, lastTime) {
  // calculate new position
  // ...
  requestAnimationFrame(newTime => animate(newTime, time));
}
requestAnimationFrame(animate);
```

ÜBERSICHT

- JavaScript im Browser
- Document Object Model
- Event Handling im Browser
- Kleiner Exkurs: jQuery
- SVG, Canvas und Web-Storage

EVENT HANDLING IM BROWSER

- Im Browser können Event Handler registriert werden
- Methode: `addEventListener`
- Erstes Argument: Ereignistyp
- Zweites Argument: Funktion, die beim Eintreten des Events aufgerufen werden soll

```
<p>Click this document to activate the handler.</p>
<script>
  window.addEventListener("click", () => {
    console.log("You knocked?")
  })
</script>
```

EVENTS UND DOM-KNOTEN

- Events können auch an DOM-Elementen registriert werden
- Nur Ereignisse, die im Kontext dieses Elements auftreten, werden dann berücksichtigt

```
<button>Click me</button>
<p>No handler here.</p>
<script>
  let button = document.querySelector("button")
  button.addEventListener("click", () => {
    console.log("Button clicked.")
  })
</script>
```

EVENT-OBJEKT

- Nähere Informationen über das eingetretene Ereignis
- Wird dem Event Handler automatisch übergeben
- Je nach Ereignistyp verschiedene Attribute
- Bei Mouse Events z.B. x und y (Koordinaten)

```
<script>
  let button = document.querySelector("button")
  button.addEventListener("click", (e) => {
    console.log("x="+e.x+", y="+e.y)
  })
  // z.B.: x=57, y=14
</script>
```

https://developer.mozilla.org/en-US/docs/Web/API/Event

EVENT-WEITERLEITUNG

- Ereignisse werden für Knoten im DOM-Baum registriert
- Reagieren auch, wenn Ereignis an untergeordnetem Knoten auftritt
- Alle Handler nach oben bis zur Wurzel des Dokuments ausgeführt
- Bis ein Handler `stopPropagation()` auf dem Event-Objekt aufruft

```
<p>A paragraph with a <button>button</button>.</p>
<script>
  let para = document.querySelector("p")
  let button = document.querySelector("button")
  para.addEventListener("mousedown", () => {
    console.log("Handler for paragraph.")
  })
  button.addEventListener("mousedown", event => {
    console.log("Handler for button.")
    if (event.button == 2) event.stopPropagation()
  })
</script>
```

EVENT-WEITERLEITUNG

- Element, bei welchem das Ereignis ausgelöst wurde: `event.target`
- Element, bei welchem das Ereignis registriert wurde: `event.currentTarget`

```
<button>A</button>
<button>B</button>
<button>C</button>
<script>
  document.body.addEventListener("click", event => {
    if (event.target.nodeName == "BUTTON") {
      console.log("Clicked", event.target.textContent)
    }
  })
</script>
```

DEFAULT-VERHALTEN

- Viele Ereignisse haben ein Default-Verhalten
- Beispiel: auf einen Link klicken
- Eigene Handler werden vor Default-Verhalten ausgeführt
- Aufruf von `preventDefault()` auf Event-Objekt verhindert Default-Verhalten

```
<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-EREIGNISSE

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

- Ereignisse `keydown` und `keyup`
- Modifier-Tasten als Attribute des Event-Objekts
- Achtung: `keydown` kann bei längerem Drücken mehrfach auslösen

ZEIGER-EREIGNISSE

- Mausklicks:
  `mousedown`, `mouseup`, `click`, `dblclick`
- Mausbewegung:
  `mousemove`
- Berührung (Touch-Display):
  `touchstart`, `touchmove`, `touchend`

SCROLL-EREIGNISSE

- Ereignis-Typ: `scroll`
- Attribute des Event-Objekts: `pageYOffset`, `pageXOffset`

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

https://eloquentjavascript.net/15_event.html#h_xGSp7W5DAZ

FOKUS- UND LADE-EREIGNISSE

- Fokus erhalten/verlieren: `focus`, `blur`
- Seite wurde geladen: `load`
  - Ausgelöst auf `window` und `document.body`
  - Elemente mit externen Ressourcen (`img`) unterstützen ebenfalls `load`-Events
  - Bevor Seite verlassen wird: `beforeunload`
- Diese Ereignisse werden nicht propagiert

WEB WORKERS

- Laufen parallel zum Haupt-Script
- Ziel: aufwändige Berechnungen blockieren nicht die Event Loop

```
// squareworker.js
addEventListener("message", event => {
  postMessage(event.data * event.data)
})

// main script
let squareWorker = new Worker("code/squareworker.js")
squareWorker.addEventListener("message", event => {
  console.log("The worker responded:", event.data)
})
squareWorker.postMessage(10)
squareWorker.postMessage(24)
```

VERZÖGERTES BEARBEITEN

- Bestimmte Ereignisse in schneller Folge ausgelöst
- Zum Beispiel: `mousemove`, `scroll`
- Ereignisbearbeitung auf Wesentliches reduzieren
- Oder jeweils mehrere Ereignisse zusammenfassen

```
<textarea>Type something here...</textarea>
<script>
  let textarea = document.querySelector("textarea")
  let timeout
  textarea.addEventListener("input", () => {
    clearTimeout(timeout)
    timeout = setTimeout(() => console.log("Typed: " + textarea.value), 500)
  })
</script>
```

ÜBERSICHT

- JavaScript im Browser
- Document Object Model
- Event Handling im Browser
- Kleiner Exkurs: jQuery
- SVG, Canvas und Web-Storage

JQUERY

- DOM-Scripting ist oft mühsam
- Grund: unübersichtliche, inkonsistente API
- Abhilfe für lange Zeit: jQuery
  - DOM-Element mit CSS-Selektor auswählen
  - Einfache Anpassungen am DOM
  - Asynchrone Serverzugriffe (Ajax)
- Bedeutung von jQuery heute abnehmend

JQUERY: DOM UND EVENTS

```
$("button.continue").html("Next Step...")

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

- `$(<selector>)` erzeugt jQuery Objekt, das eine Sammlung von DOM-Elementen enthält
- Darauf sind zahlreiche Methoden anwendbar
- DOM-Traversal und -Manipulation sehr einfach

https://api.jquery.com
http://jqapi.com

JQUERY: ÜBERBLICK

Aufruf	Bedeutung	Beispiel
$(*Funktion*)	DOM ready	$(function() { .... });
$("*CSS Selektor*") .aktion(arg1, .,..) .aktion(...)	Wrapped Set - Knoten, die Sel. erfüllen - eingepackt in jQuery Obj.	$(".toggleButton").attr("title") $(".toggleButton").attr("title", "click here") $(".toggleButton").attr({title : "click here", ...}) $(".toggleButton").attr("title", function(){...}) .css(...) .text(...) .on("click", function(event) { ...})
$("*HTML-Code*")	Wrapped Set - neuer Knoten - eingepackt in jQuery Obj. - noch nicht im DOM	$(" ... ").addClass(...) .appendTo("*Selektor*") $(" ... ").length $(" ... ")[0]
$(DOM-Knoten)	Wrapped Set - dieser Knoten - eingepackt in jQuery Obj.	$(document.body) $(this)

ÜBERSICHT

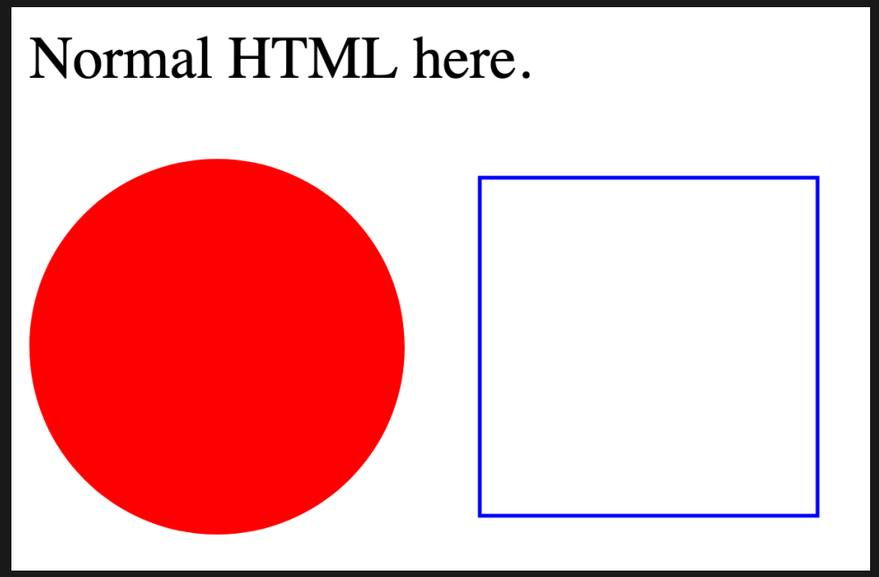
- JavaScript im Browser
- Document Object Model
- Event Handling im Browser
- Kleiner Exkurs: jQuery
- SVG, Canvas und Web-Storage

WEB-GRAFIKEN

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

SVG

```
<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>
```



SVG

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

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

```
▼ children:                    HTMLCollection { 0: circle , 1: rect , length: 2 }
  ▼ 0:                         <circle r="50" cx="50" cy="50" fill="cyan">
      assignedSlot:            null
    ▼ attributes:              NamedNodeMap(4) [ r="50", cx="50", cy="50", … ]
      ▶ 0:                     r="50"
      ▶ 1:                     cx="50"
      ▶ 2:                     cy="50"
      ▶ 3:                     fill="cyan"
        length:                4
```

CANVAS

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

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
  let canvas = document.querySelector("canvas")
  let context = canvas.getContext("2d")
  context.fillStyle = "red"
  context.fillRect(10, 10, 100, 50)
</script>
```

CANVAS: PFADE

```
<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()
</script>
```

CANVAS: WEITERE MÖGLICHKEITEN (1)

- Quadratische Kurven: `quadraticCurveTo`
- Bezier-Kurven: `bezierCurveTo`
- Kreisabschnitte: `arc`
- Text: `fillText`, `strokeText` (und: `font`-Attribut)
- Bild: `drawImage`

CANVAS: WEITERE MÖGLICHKEITEN (2)

- Skalieren: `scale`
- Koordinatensystem verschieben: `translate`
- Koordinatensystem rotieren: `rotate`
- Transformationen auf Stack speichern: `save`
- Letzten Zustand wiederherstellen: `restore`

HTML, SVG, CANVAS

HTML

- einfach
- Textblöcke mit Umbruch, Ausrichtung etc.

SVG

- beliebig skalierbar
- Struktur im DOM abgebildet
- Events auf einzelnen Elementen

Canvas

- einfache Datenstruktur: Ebene mit Pixeln
- Pixelbilder verarbeiten

WEB STORAGE

- Speichern von Daten clientseitig
- Einfache Variante: Cookies (s. spätere Lektion)
- Einfache Alternative: LocalStorage
- Mehr Features: IndexedDB (nicht Stoff von WBE)

LOCALSTORAGE

```
localStorage.setItem("username", "bkrt")
console.log(localStorage.getItem("username"))    // → bkrt
localStorage.removeItem("username")
```

- Bleibt nach Schliessen des Browsers erhalten
- In Developer Tools einsehbar und änderbar
- Alternative solange Browser/Tab geöffnet: `sessionStorage`

LOCALSTORAGE

- Gespeichert nach Domains
- Limit für verfügbaren Speicherplatz pro Website (~5MB)
- Attributwerte als Strings gespeichert
- Konsequenz: Objekte mit JSON codieren

```
let user = {name: "Hans", highscore: 234}
localStorage.setItem(JSON.stringify(user))
```

QUELLEN

- Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
  https://eloquentjavascript.net/
- Ältere Slides aus WEB2 und WEB3

LESESTOFF

Geeignet zur Ergänzung und Vertiefung

- Kapitel 14, 15 und 17 von:
  Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
  https://eloquentjavascript.net/