

WBE: BROWSER-TECHNOLOGIEN

JAVASCRIPT IM BROWSER (TEIL 1)

ÜBERSICHT

- JavaScript im Browser
- Vordefinierte Objekte
- DOM: Document Object Model
- DOM Scripting
- CSS und das DOM

ÜBERSICHT

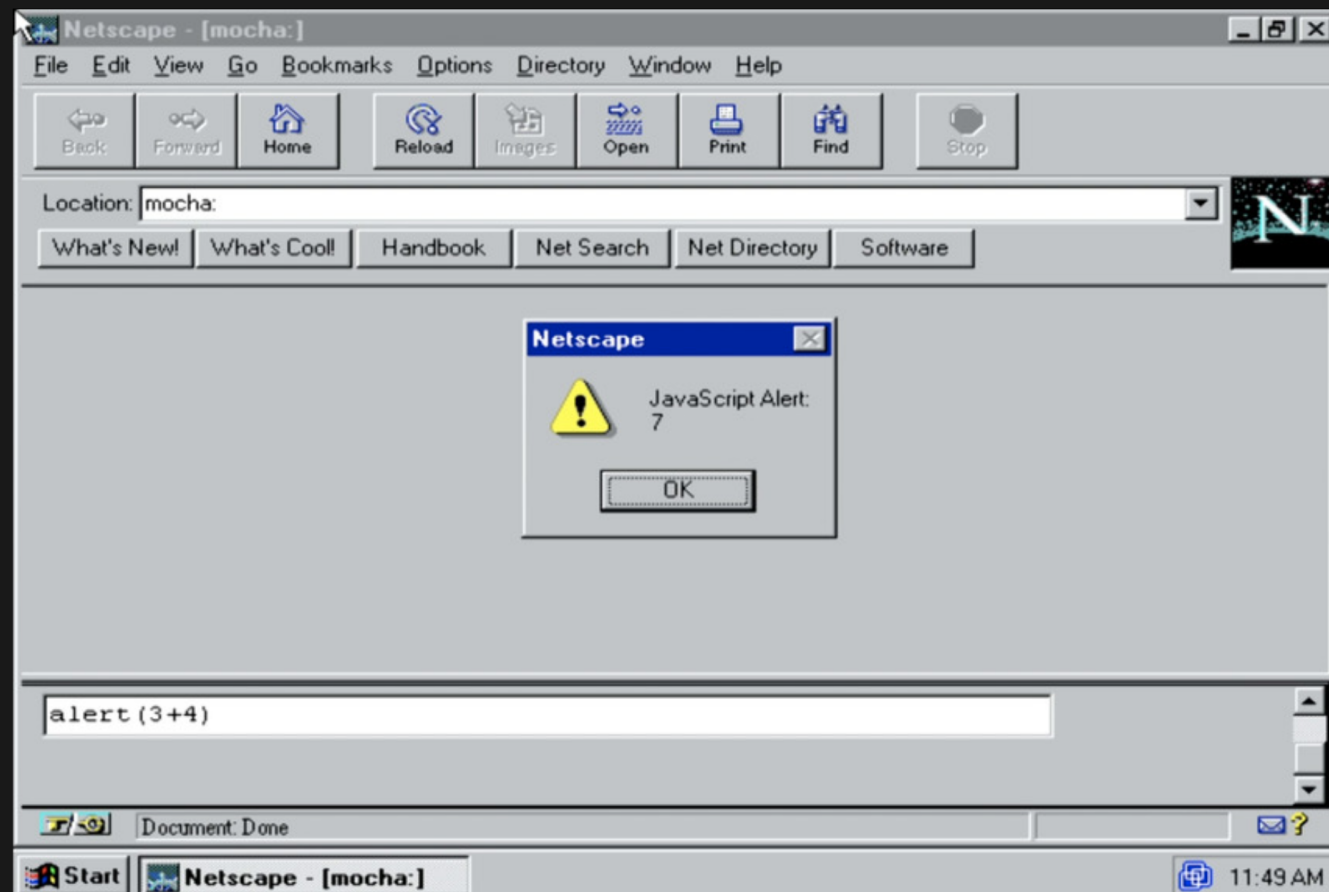
- JavaScript im Browser
- Vordefinierte Objekte
- DOM: Document Object Model
- DOM Scripting
- CSS und das DOM

JAVASCRIPT IM BROWSER

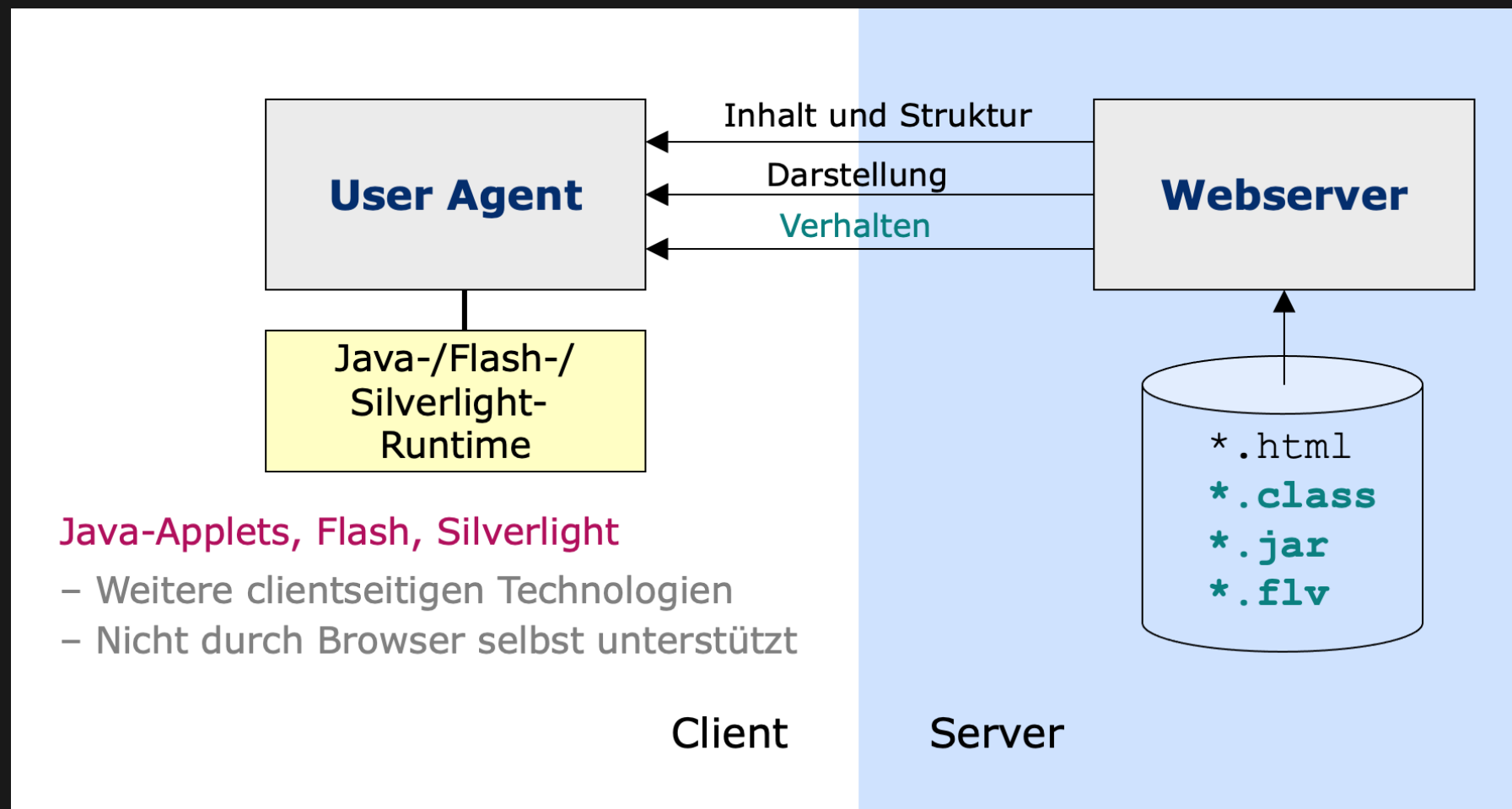
- Ohne Browser gäbe es kein JavaScript
- Für den Einsatz im Browser entwickelt
- Brendan Eich, 1995: Netscape Navigator

JAVASCRIPT IM NETSCAPE 2 PRE-ALPHA

Mutmasslich erste
JavaScript-Demo
durch Brendan Eich
1995



ANDERE CLIENTSEITIGE TECHNOLOGIEN



Alle im Laufe
der Zeit wieder
verschwunden

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

JAVASCRIPT-KONSOLE

The screenshot shows a web browser window with the address bar displaying `https://gburkert.github.io/selectors/`. The main content area displays a DOM tree structure. The tree has a root node `h1` and a `nav` node. The `nav` node has a child `ul` node, which has five children: `li`, `li`, `li` (with `class="active"`), `li`, and `li`. Each `li` node has a child `a` node. The `section` node has three children: `h2`, `p`, and `p` (with `id="ads"`). The first `p` node has a child `img` node (with `src="logo.png"` and `alt="Logo"`). The second `p` node has a child `a` node (with `href="docs.pdf"`). The `li` nodes with `class="active"` and the `img` node are highlighted in green. The `li` nodes with `class="active"` and the `img` node are also highlighted in green. The `li` nodes with `class="active"` and the `img` node are also highlighted in green.

The bottom of the browser window shows the developer tools interface. The `Konsole` tab is selected and highlighted with a red circle. The console output shows the following commands and results:

```
>> let f = n => n * n
< undefined

>> f(5)
< 25

>> document.querySelectorAll("li:nth-child(2n+1)").forEach(item=>item.style.backgroundColor="MediumAquaMarine")
< undefined

>> |
```

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
- Vordefinierte Objekte
- DOM: Document Object Model
- DOM Scripting
- CSS und das DOM

VORDEFINIIERTE OBJEKTE

Allgemeine Objekte	Browser-Objekte
Object	document
Array	window
Function	event
String	history
Date	location
Math	navigator
RegExp	...
...	...

VORDEFINIIERTE OBJEKTE

- Die **allgemeinen Objekte** sind in JavaScript vordefiniert
- Tatsächlich handelt es sich um Funktionen/Konstrukturen
- 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 Webseite
- Einstieg ins **DOM** (Document Object Model)
- Diverse Attribute und Methoden, zum Beispiel:

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

window

- Repräsentiert das Browserfenster
- Zahlreiche Attribute und Methoden, u.a.:

```
window.document      /* Zugriff auf Dokument      */  
window.history        /* History-Objekt            */  
window.innerHeight   /* Höhe des Viewports       */  
window.pageYOffset    /* vertikal gescrollte Pixel */
```

globales Objekt

- `window` ist das globale Objekt der Browser-Plattform
- Alle globalen Variablen und Methoden sind hier angehängt
- Neue globale Variablen landen ebenfalls hier

```
window.alert === alert      /* → true */  
window.setTimeout === setTimeout /* → true */  
window.parseInt === parseInt /* → true */
```


navigator

```
> navigator.userAgent
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101
Firefox/119.0"

> navigator.language
"de"

> navigator.platform
"MacIntel"

> navigator.onLine
true
```

MDN: Navigator

location

- Aktuelle Webadresse im Browser
- Zugänglich über `window.location` und `document.location`

```
> location.href  
"https://gburkert.github.io/selectors/"  
  
> location.protocol  
"https:"  
  
> document.location.protocol  
"https:"
```

MDN: Location

ÜBERSICHT

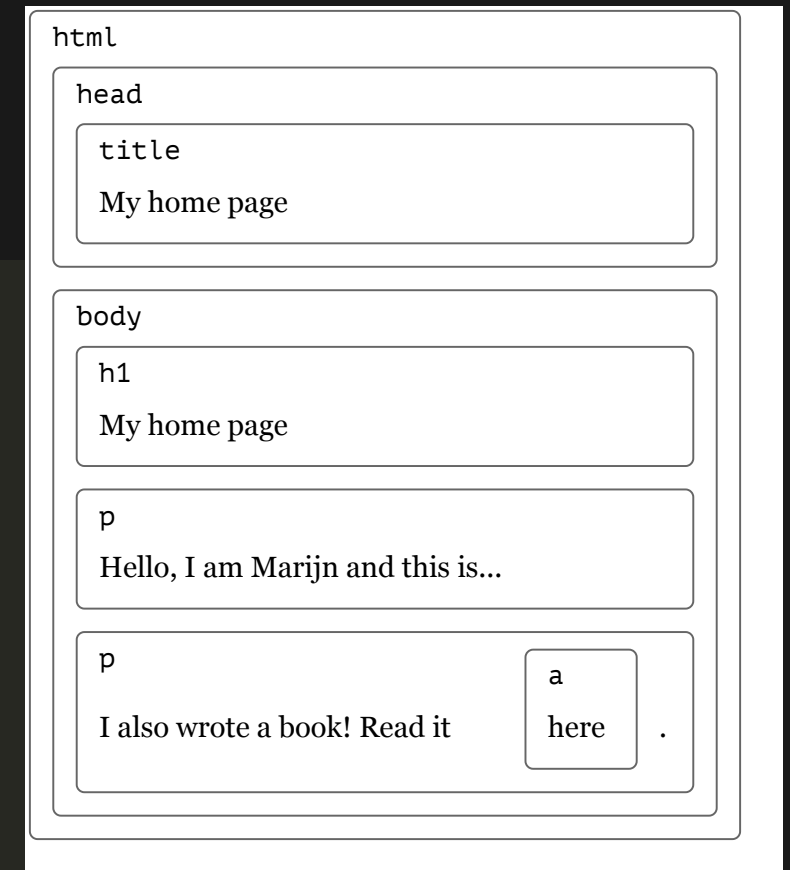
- JavaScript im Browser
- Vordefinierte Objekte
- **DOM: Document Object Model**
- DOM Scripting
- CSS und das DOM

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
- Zugriff über das globale Objekt `document`
 - Attribut `documentElement` ist Referenz auf HTML-Knoten
 - Attribut `body` ist Referenz auf das body-Element
- 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:	""

childNodes, children

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

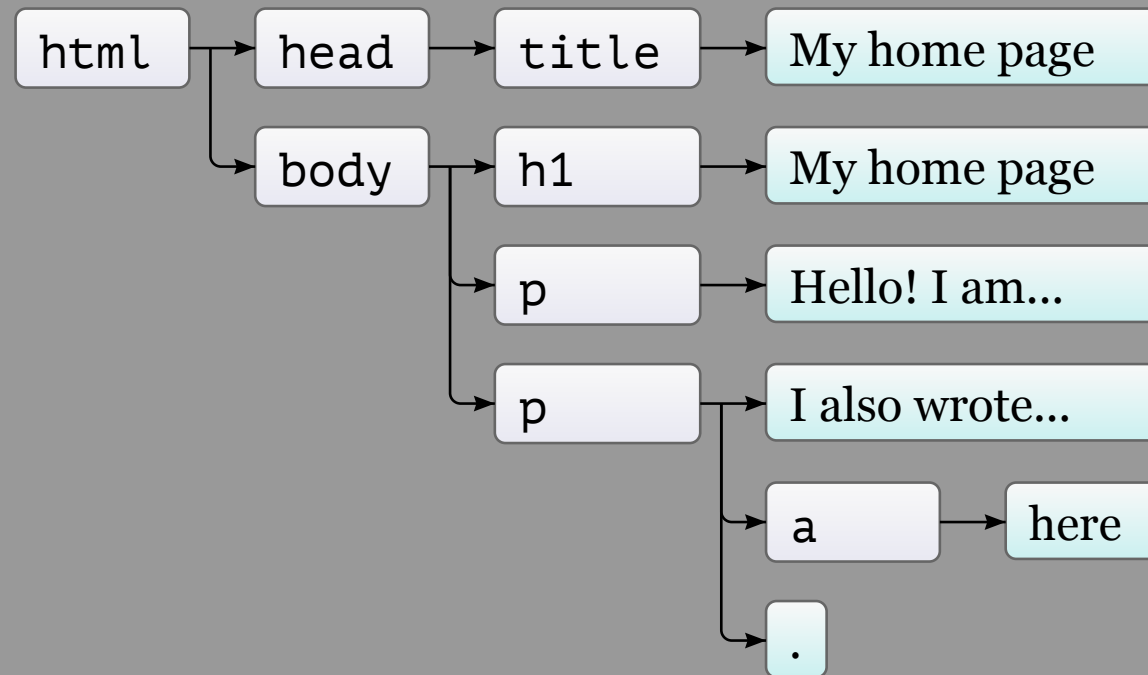
```
▶ childNodes:      NodeList(7) [ #text , h1 , #text , ... ]
▼ children:        HTMLCollection { 0: h1 , 1: p , length: 3, ... }
  ▶ 0:              <h1>
  ▶ 1:              <p>
  ▶ 2:              <p>
  length:           3
```


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



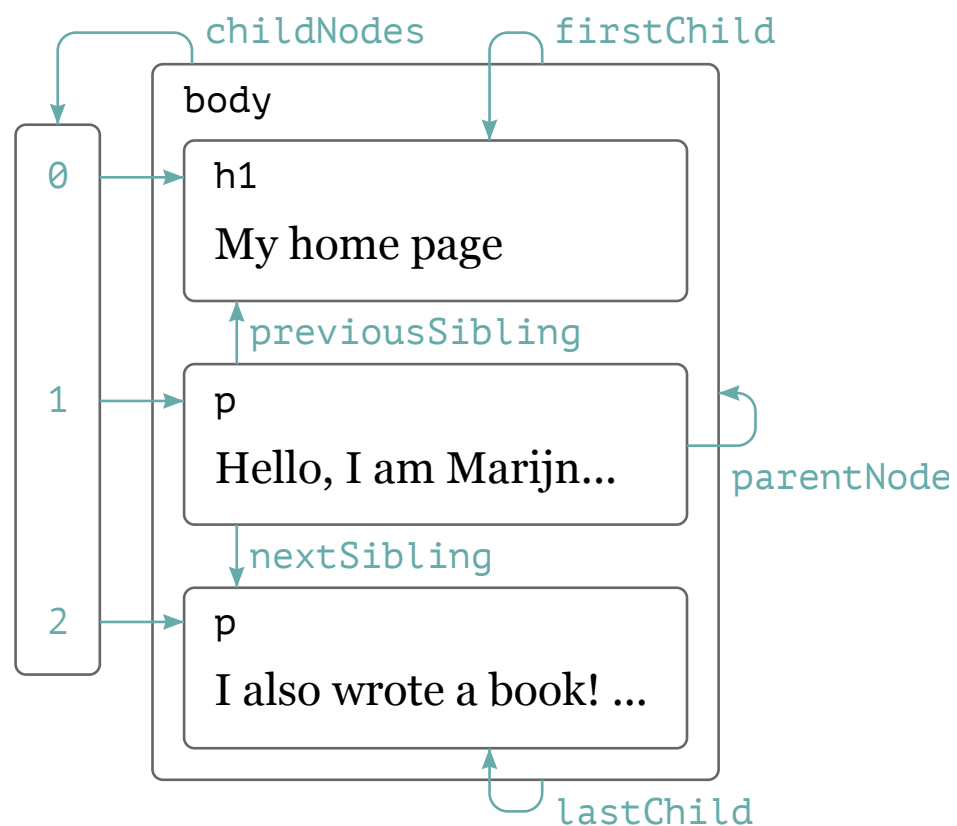
DOM ALS STANDARD

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

ÜBERSICHT

- JavaScript im Browser
- Vordefinierte Objekte
- DOM: Document Object Model
- **DOM Scripting**
- CSS und das DOM

BAUMSTRUKTUR ABARBEITEN



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

BEISPIEL

```
1  /* scans a document for text nodes containing a given string and */
2  /* returns true when it has found one */
3  function talksAbout (node, string) {
4      if (node.nodeType == Node.ELEMENT_NODE) {
5          for (let i = 0; i < node.childNodes.length; i++) {
6              if (talksAbout(node.childNodes[i], string)) {
7                  return true
8              }
9          }
10         return false
11     } else if (node.nodeType == Node.TEXT_NODE) {
12         return node.nodeValue.indexOf(string) > -1
13     }
14 }
15
16 console.log(talksAbout(document.body, "book"))
17 /* → true */
```

BEISPIEL

- Das Attribut `childNodes` liefert kein echtes Array, sondern eine „live“ `NodeList`
- Eine Iteration mit `for/of` wäre damit auch möglich

„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

ARRAY-ÄHNLICHE OBJEKTE

- 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"]
```


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

ELEMENT VERSCHIEBEN...

- 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

```
1 <p>The  in the
2   .</p>
3
4 <p><button onclick="replaceImages()">Replace</button></p>
5
6 <script>
7   function replaceImages () {
8     let images = document.body.getElementsByTagName("img")
9     for (let i = images.length - 1; i >= 0; i--) {
10      let image = images[i]
11      if (image.alt) {
12        let text = document.createTextNode(image.alt)
13        image.parentNode.replaceChild(text, image)
14      }
15    }
16  }
17 </script>
```

Quelle: Eloquent JavaScript

NEUES ELEMENT ANLEGEN

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

ELEMENT ANLEGEN: ABSTRAKTION

```
1 function elt (type, ...children) {  
2   let node = document.createElement(type)  
3   for (let child of children) {  
4     if (typeof child !== "string") node.appendChild(child)  
5     else node.appendChild(document.createTextNode(child))  
6   }  
7   return node  
8 }
```

- Hilfsfunktion zum Erzeugen von Elementknoten
- Element mit Typ (1. Argument) erzeugen
- Kindelemente (weitere Argumente) hinzufügen

ELEMENT ANLEGEN: BEISPIEL

```
1 <blockquote id="quote">
2   No book can ever be finished. While working on it we learn ...
3 </blockquote>
4
5 <script>
6   /* definition of elt ... */
7
8   document.getElementById("quote").appendChild(
9     elt("footer", "-",
10       elt("strong", "Karl Popper"),
11       ", preface to the second edition of ",
12       elt("em", "The Open Society and Its Enemies"),
13       ", 1950"))
14
15 </script>
```

HTML-ELEMENTOBJEKTE

- Bisher haben wir die Knoten der HTML-Struktur allgemein als Elementknoten behandelt
- Dieses universelle DOM funktioniert mit allen XML-Sprachen
- Speziell für HTML gibt es aber auch die **Elementobjekte**
- Je nach Elementtyp haben sie spezielle Attribute/Methoden

<https://www.w3schools.com/jsref/default.asp>

ELEMENTOBJEKT **img**

- Zugriff auf die Attribute des img-Elements
- Möglichkeit, die Eigenschaften eines Bilds zu ändern, etwa das Bild auszutauschen

Attribut	Bedeutung
src	URL oder Pfad zur Bilddatei
alt	Alternativtext
width	Breite des Bilds
height	Höhe des Bilds
...	...

ATTRIBUTE

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

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

DOM:

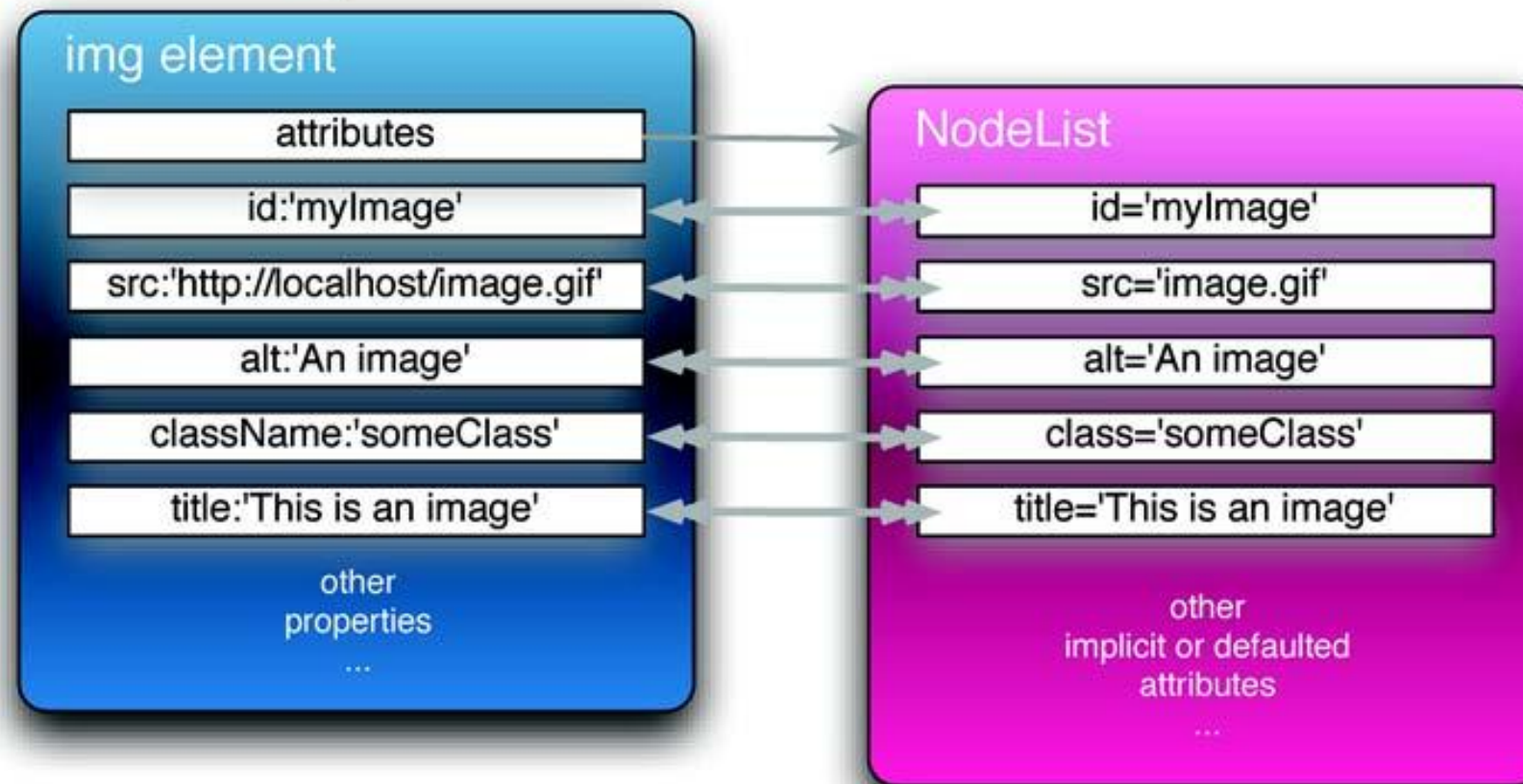
```
a-element
  accessKey: ""
  accessKeyLabel: ""
  attributes: NamedNodeMap [ href="http://eloquentjavascript.net" ]
  childNodes: NodeList [ #text ]
  children: HTMLCollection { length: 0 }
  classList: DOMTokenList []
  className: ""
  ...
  href: "http://eloquentjavascript.net/"
  ...
```

ATTRIBUTE

HTML markup

```

```



ATTRIBUTE **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"  
...
```

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>
```

ÜBERSICHT

- JavaScript im Browser
- Vordefinierte Objekte
- DOM: Document Object Model
- DOM Scripting
- CSS und das DOM

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 ANPASSEN: **class**

- DOM-Scripting kann Inhalte eines Dokuments anpassen
- Damit auch: Attribut `class` von Elementen
- Stylesheet wie gewohnt separat
- CSS-Regeln mit `class`-Selektor
- Damit ist eine dynamische Anpassung der Darstellung vom Script aus möglich

DARSTELLUNG ANPASSEN: *style*

- 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>
```

QUELLEN

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

LESESTOFF

Geeignet zur Ergänzung und Vertiefung

- Kapitel 13 und 14 von:
Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
<https://eloquentjavascript.net/>

