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

Speaker notes

Zur Erinnerung:

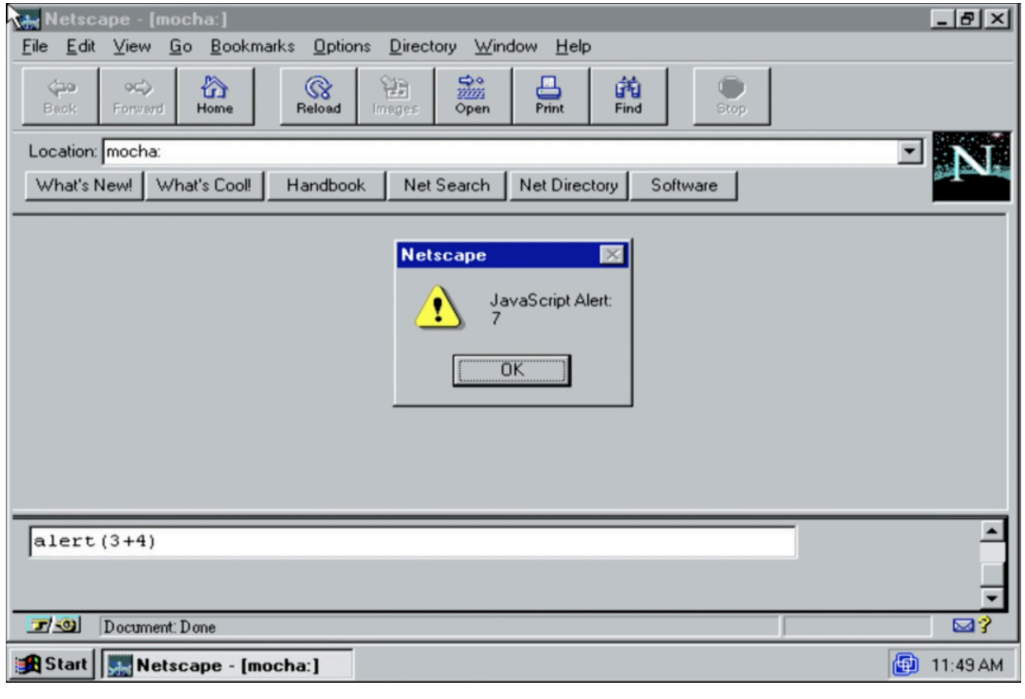
Mitte der 1990er Jahre war der Netscape Navigator der vorherrschende Browser, bevor ihm der Internet Explorer den Rang abgelaufen hat.

Die Entwicklung von JavaScript sehr schön zusammengefasst finden Sie im Vortrag „JavaScript: The First 20 Years” von Allen Wirfs-Brock und Brendan Eich auf der Konferenz PLDI 2021:

https://www.pldi21.org/track_hopl.html

Eine kurze Zusammenfassung gibt es in den Lecture Notes zum Thema *JavaScript Grundlagen* (Semesterwoche 2)

JAVASCRIPT IM NETSCAPE 2 PRE-ALPHA

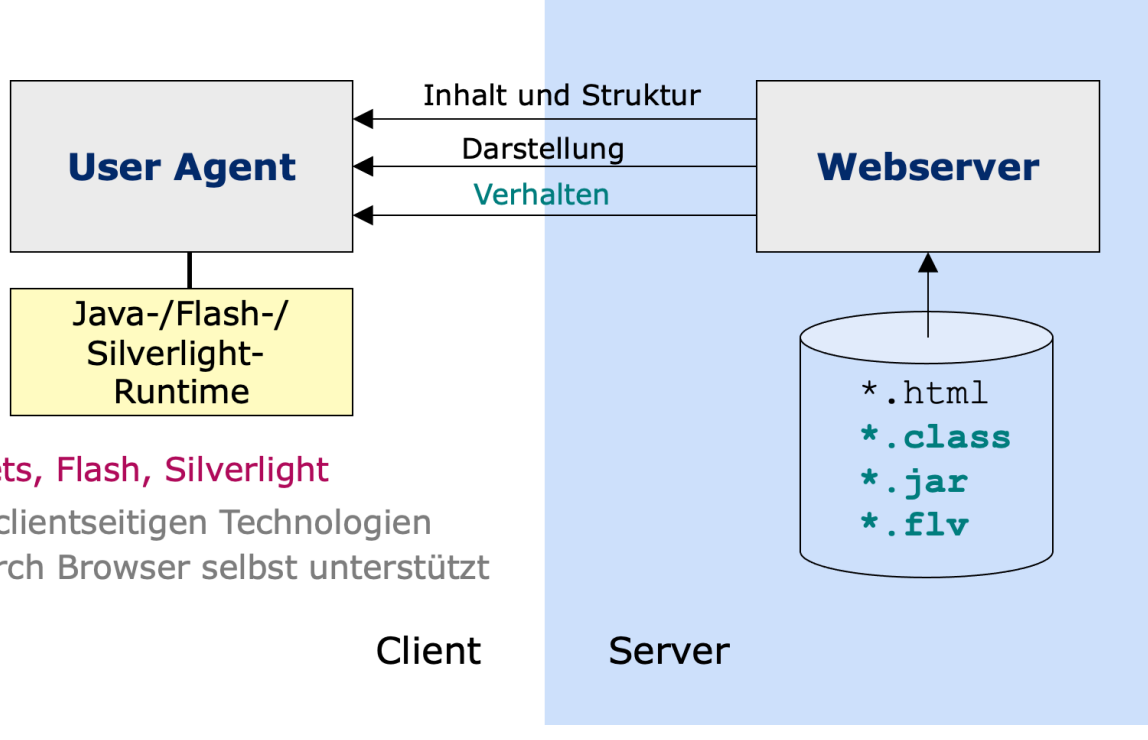


Mutmasslich erste JavaScript-Demo durch Brendan Eich 1995

Speaker notes

Noch unter der Bezeichnung *Mocha*.

ANDERE CLIENTSEITIGE TECHNOLOGIEN



Java-Applets, Flash, Silverlight

- Weitere clientseitigen Technologien
- Nicht durch Browser selbst unterstützt

Alle im Laufe der Zeit wieder verschwunden

Speaker notes

In den 1990er Jahren war zunächst nicht ganz klar, ob JavaScript gegen Java im Browser überhaupt eine Chance hat. Immerhin konnten mit Java komplette Programme mit komplexen grafischen Oberflächen geschrieben werden – etwas, was mit HTML, CSS und JavaScript noch in weiter Ferne lag.

Die Technologie, Java im Browser auszuführen, hier *Java Applets*. Entsprechende Programme wurden mit einem Java-Compiler übersetzt und als class- oder jar-Dateien dem Browser zur Verfügung gestellt. Während die Idee, auf diese Weise im Browser Programme mit grafischer Oberfläche umzusetzen, an sich nicht schlecht war, war doch die Umsetzung nicht befriedigend. Unterschiedliche Unterstützung in verschiedenen Browsern, Abstürze und andere Instabilitäten führten dazu, dass Java-Applets sich nie richtig durchsetzen konnten.

Der erfolgreichere Ansatz war Flash von Macromedia (später Adobe). Viele Spiele und andere Applikationen wurden damit umgesetzt. Es funktionierte tatsächlich browser- und plattformübergreifend und stellte Technologien zur Verfügung, welche in den Browsern ansonsten nicht verfügbar war, etwa das clientseitige Speichern von Daten (Browser unterstützten nur Cookies). Das Ende von Flash wurde durch die Entscheidung von Apple eingeläutet, Flash auf ihrer Mobilplattform nicht zu unterstützen.

Silverlight, Microsofts Alternative zu Applets und Flash, war nie ein grosser Durchbruch beschieden.

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>
```

Speaker notes

Normalerweise sollte das `script`-Element nur verwendet werden, um eine JavaScript-Datei zu laden, also die mittlere der drei angegebenen Varianten.

Gelegentlich sieht man das `script`-Element auch mit `type`-Attribut und JavaScript-MIME-Type:

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

Das ist nicht mehr nötig. Aus der HTML-Spezifikation:

> *`type`*
> *This attribute indicates the type of script represented. The value of this attribute will be in one of the following categories:*
> *Omitted or a JavaScript MIME type: This indicates the script is JavaScript. The HTML5 specification urges authors to omit the attribute rather than provide a redundant MIME type. In earlier browsers, this identified the scripting language of the embedded or imported (via the src attribute) code. JavaScript MIME types are listed in the specification.*
> *`module` Causes the code to be treated as a JavaScript module. [...]*

Hinweis: Im Quellcode der Slides dürfen keine *script*-Tags vorkommen, daher wird in den Beispielen in *script*-Tags der Buchstabe s nicht als ein ASCII-Zeichen angegeben. Konsequenz: Der Code wird nach Copy/Paste nicht unmittelbar funktionieren. Sie müssen erst korrekte *script*-Tags einsetzen.

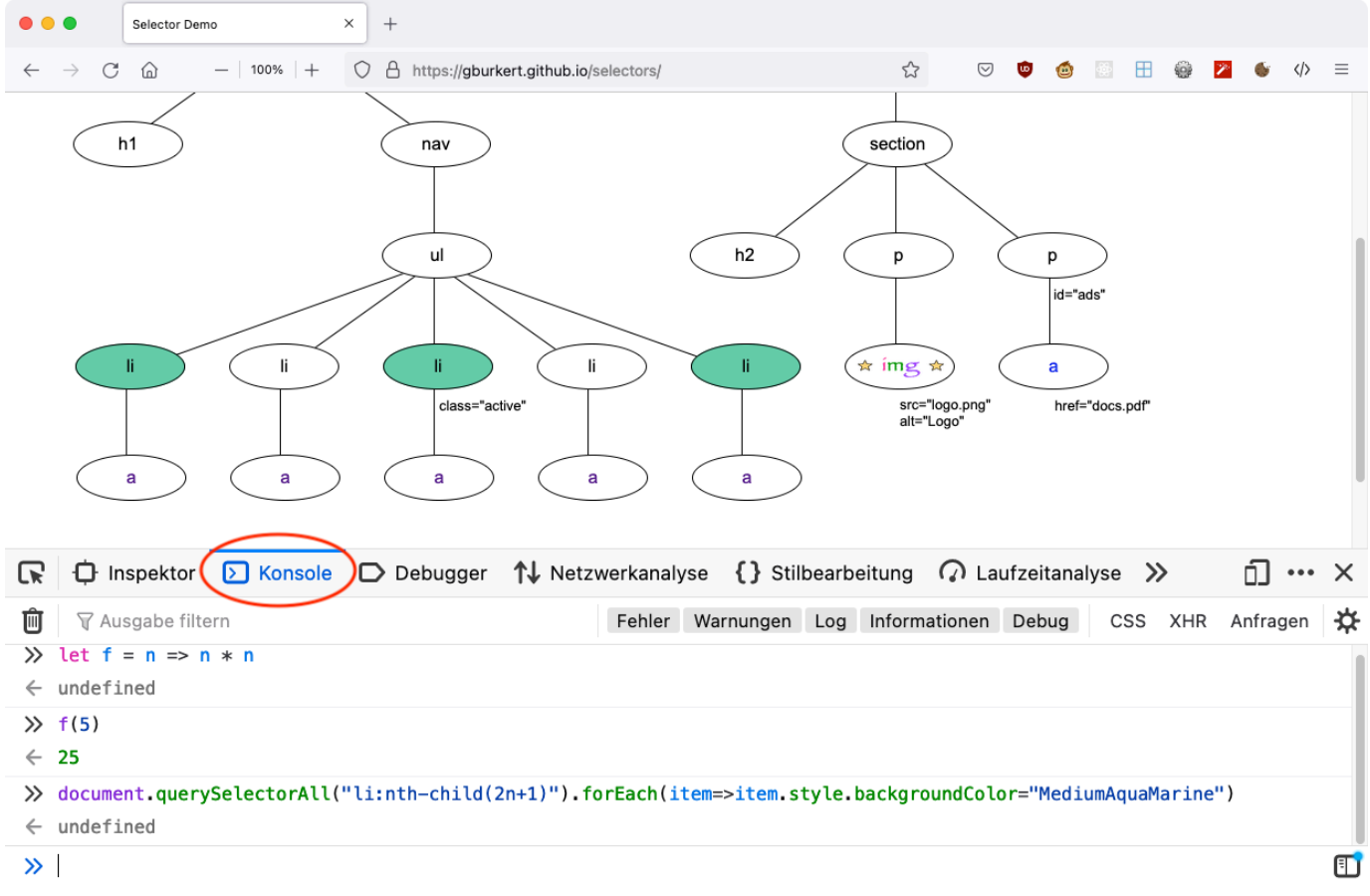
HTML UND JAVASCRIPT

- 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



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

VORDEFINIERTE OBJEKTE

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

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

Speaker notes

Genau genommen handelt es sich bei den allgemeinen Objekten *normalerweise* um Funktionen bzw. Konstruktoren. Die Grossschreibung zeigt an, dass es sich um Funktionen handelt, welche *normalerweise* mit `new` aufgerufen werden.

Eine Ausnahme ist `Math`. Es ist kein Konstruktor und kann nicht mit `new` aufgerufen werden.

Von den Browser-Objekten ist `document` in verschiedenen Standards und Versionen rund um das DOM (Document Objekt Model) beschrieben. Für die anderen (`window`, `navigator`, etc.) existieren keine öffentlichen Standards. Sie werden aber von den JavaScript-Implementierungen der Browser unterstützt.

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       */
```

Speaker notes

Falls beim Setzen eines Cookies (in einer späteren Lektion behandelt) via HTTP das Attribut `HttpOnly` eingefügt wird, kann auf dieses Cookie nicht von JavaScript aus zugegriffen werden.

Zum Zugriff auf Verweise und Bilder werden heute meist die (im nächsten Abschnitt behandelten) allgemeinen DOM-Methoden verwendet. Für manche Elemente, etwa für den Zugriff auf Formularelemente, gibt es aber spezifische Attribute und Methoden:

```
document.forms                // Zugriff auf Formulare
document.forms[0].elements    // Zugriff auf Elemente des ersten Formulars
```

Das Öffnen des Dokuments und Schreiben von Text mit Hilfe von Methoden des `document`-Objekts war in den Anfangszeiten von JavaScript üblich, gilt heute aber als veraltet:

```
document.open();              // Dokument zum Schreiben öffnen
document.write("Hallo");      // Einfügen von Text
document.close();             // Dokument schliessen
```

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 */
```

Speaker notes

Am globalen Objekt sind zahlreiche Attribute und Methoden angehängt. Zum Beispiel im Firefox 119 mit ein paar installierten Erweiterungen:

```
> Object.getOwnPropertyNames(window).length
871
```

Unter Node.js ist das globale Objekt `global`:

```
> Object.getOwnPropertyNames(global).length
155
```

Über `globalThis` kann auf allen Plattformen auf das globale Objekt zugegriffen werden. Der Name kommt daher, dass der Wert dem von `this` im globalen Gültigkeitbereich entspricht.

```
> Object.getOwnPropertyNames(globalThis).length
/* → 871 im Browser, 155 unter Node.js 18.7 */
```

Tatsächlich ist die ganze Geschichte rund um das globale Objekt komplizierter, als es auf den erstem Blick scheint (wie vieles in JavaScript, aber zum Glück kommt man in der Regel ohne diese Details gut zurecht...). Mehr zum Thema:

https://exploringjs.com/deep-js/ch_global-scope.html

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

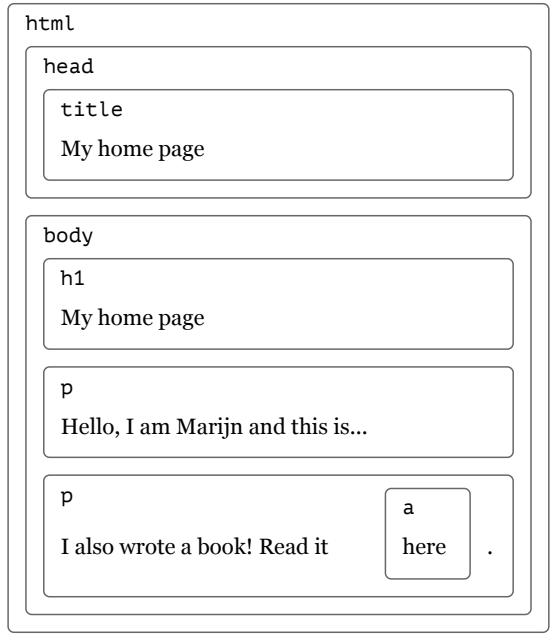
- JavaScript im Browser
- Vordefinierte Objekte
- DOM: Document Object Model
- DOM Scripting
- CSS und das DOM

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

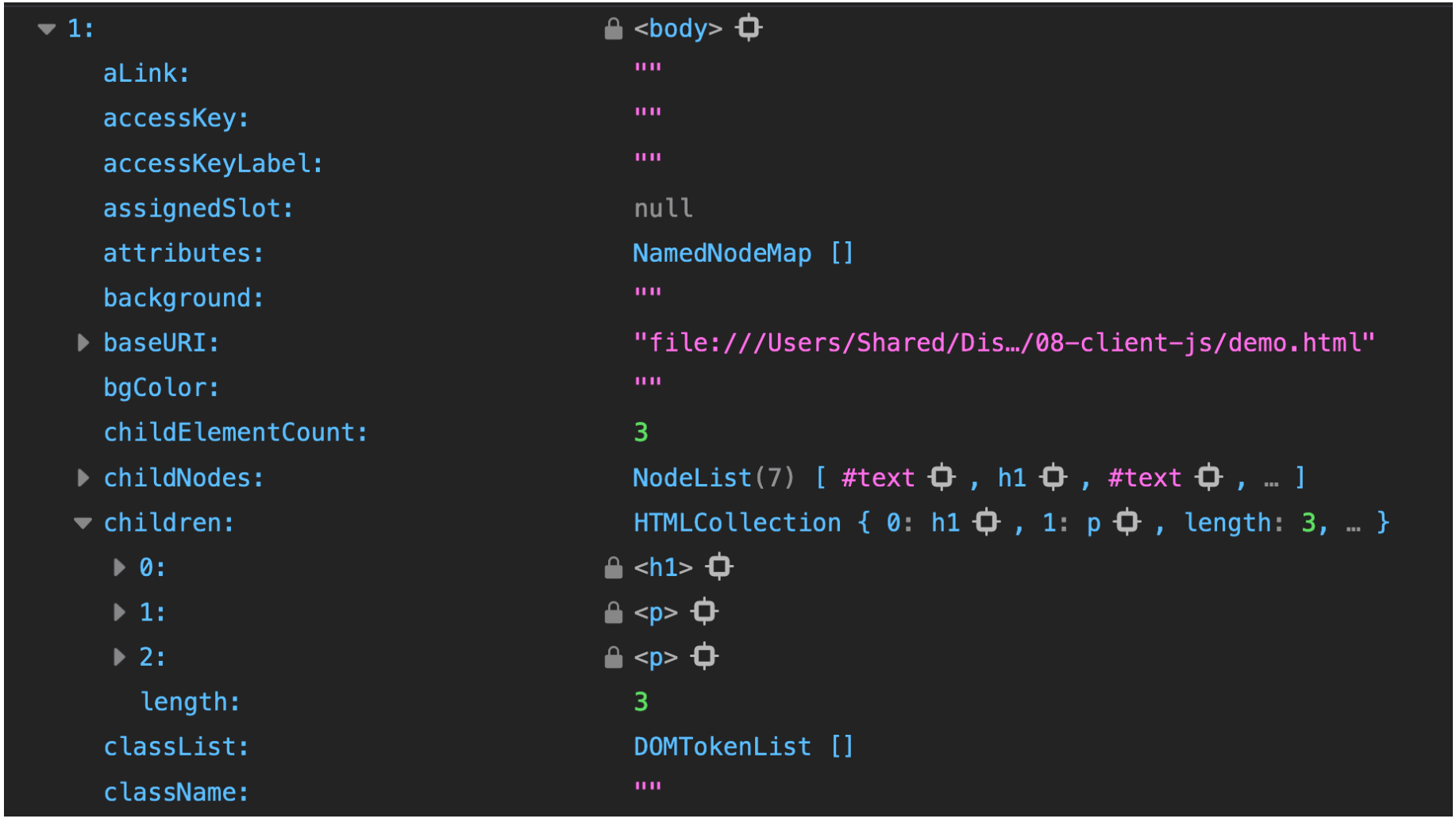
```
<!doctype html>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <h1>My home page</h1>
    <p>Hello, I am Marijn and this is my home
      page.</p>
    <p>I also wrote a book! Read it
      <a href="http://eloquentjavascript.net">here</a>.</p>
  </body>
</html>
```



DOCUMENT OBJECT MODEL (DOM)

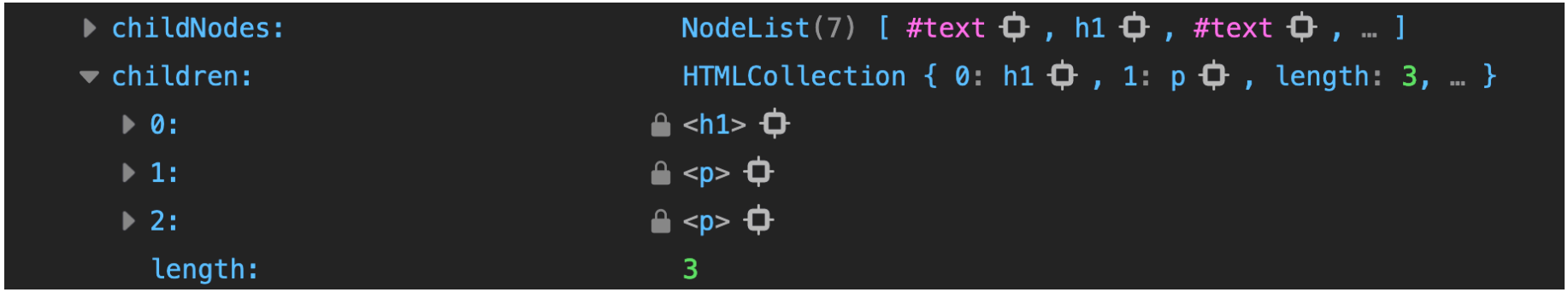
- Jeder Knoten im Baum durch ein Objekt repräsentiert
- Zugriff über das globale Objekt `document`
  - Attribut `documentElement` ist Referenz auf HTML-Knoten
  - Attribut `body` ist Referenz auf das body-Element
- Zahlreiche Attribute und Methoden

ELEMENTKNOTEN body



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



Speaker notes

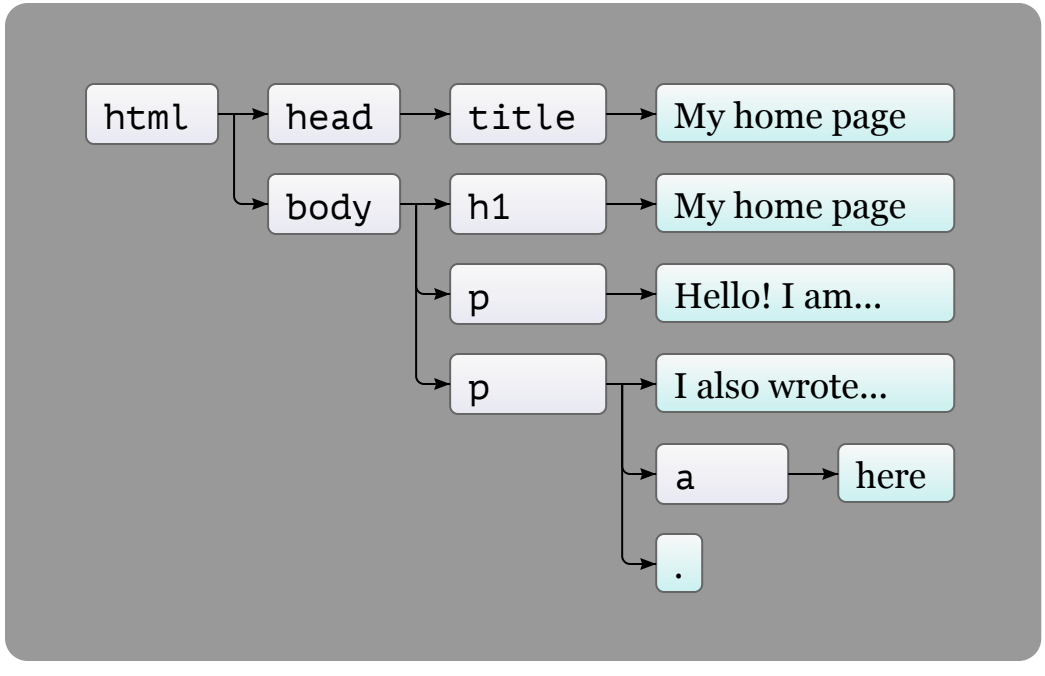
Da es kein Array ist, funktionieren Array-Methoden wie `slice` und `map` nicht, wohl aber `cnodes[0]` oder `cnodes.length`. Man kann aber mit `Array.from` ein Array erstellen.

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



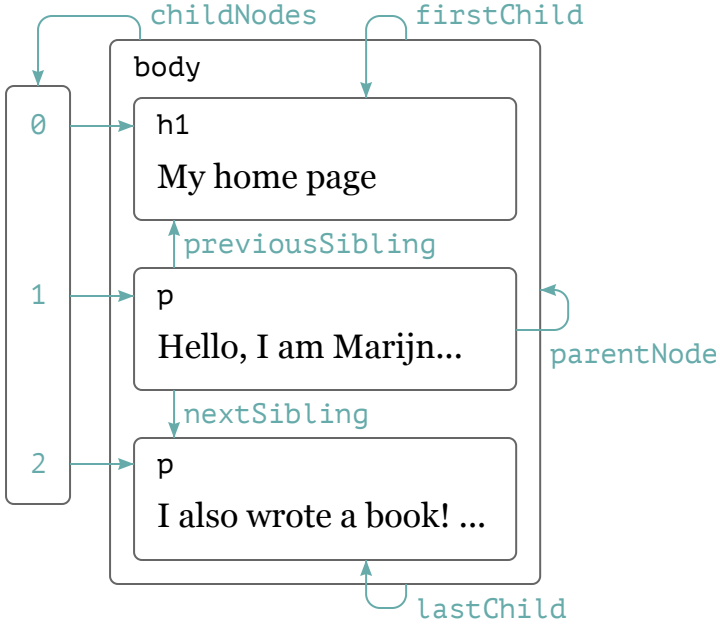
DOM ALS STANDARD

- 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

Speaker notes

„The NodeList being live means that its content is changed each time new children are added or removed.” (https://devdocs.io/dom/node/childnodes)

Mittlerweile unterstützt `NodeList` auch Iterationen mit `forEach` oder `for...of`.

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

Speaker notes

Auch `querySelectorAll` liefert wie viele andere DOM-Methoden ein Array-ähnliches Objekt als Ergebnis (konkret: es ist eine Instanz von `NodeList`).

Im Gegensatz zu `querySelectorAll` liefert `querySelector` nur das erste passende Element oder null, wenn es kein solches gibt.

https://eloquentjavascript.net/14_dom.html#h_5ooQzToxht

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>
```

Speaker notes

Das Beispiel wählt alle Absätze aus und verschiebt den letzten Absatz vor den ersten. Durch das Einfügen des Absatzes wird er automatisch an der ursprünglichen Stelle entfernt, da ein Knoten nur an einer Stelle im Dokument vorkommen kann.

Wenn statt `insertBefore` die Methode `replaceChild` verwendet wird, wird der erste Absatz durch den dritten ersetzt und es resultiert:

Three
Two

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>
```

Quelle: Eloquent JavaScript

Speaker notes

Das Beispiel ersetzt alle Bilder durch ihren im `alt`-Attribut abgelegten Alternativtext. Der Textknoten wird mit `document.createTextNode` erzeugt und das Bild mit `replaceChild` ersetzt.

Die Liste der Bilder wird hier von hinten her bearbeitet, da die `images`-Referenz auf eine *live* aktualisierte Datenstruktur verweist. Das bedeutet, dass die Anzahl der Bilder (images.length) bereits um eins reduziert ist, nachdem das erste Bild ersetzt wurde.

Das Ergebnis von `querySelectorAll` ist im Gegensatz zu `getElementsByTagName` übrigens keine "Live"-Datenstruktur, sondern eine statische `NodeList`. So geht es also auch:

```
function replaceImages () {
    let images = document.querySelectorAll("img")
    for (let image of images) {
      if (image.alt) {
        let text = document.createTextNode(image.alt)
        image.parentNode.replaceChild(text, image)
      }
    }
  }
```

NEUES ELEMENT ANLEGEN

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

Speaker notes

Die beiden Methoden `document.createAttribute` und `<element>.setAttributeNode` können auch zusammengefasst werden mit der Methode `<element>.setAttribute`.

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

Speaker notes

Die Kindelemente werden als Textknoten angelegt, wenn es Strings sind. Ansonsten wird angenommen, dass es selbst bereits Elemente sind, welche hinzugefügt werden können.

Was die Funktion `elt` noch nicht enthält ist das Hinzufügen von Attributen. Eine entsprechende Ergänzung wird im Praktikum angesehen.

ELEMENT ANLEGEN: BEISPIEL

```
<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>
```

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

Speaker notes

Es ist schon etwas unübersichtlich. Da gibt es also ein universelles DOM, das allgemein für XML-Sprachen verwendet werden kann, auch HTML. Dann gibt es noch ein HTML-spezifisches DOM.

Die Attribute und Methoden eines Elementknotens des universellen DOM sind in der Klasse `Element` beschrieben. Die zusätzlichen Attribute und Methoden eines Elementknotens des HTML-DOM sind in der Klasse `HTMLElement` beschrieben.

Lassen Sie sich von den vielen Möglichkeiten, Standards und Dokumentationen nicht verwirren. In diesen Folien wird nur ein kleiner Teil der Möglichkeiten beschrieben. Diese reichen aber im Normalfall gut aus, um Webanwendungen zu entwickeln. Wenn mehr benötigt wird, konsultieren Sie die Online-Dokus und Spezifikationen.

Weitere Informationen:

- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- https://devdocs.io/dom/

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/"
  ...
```

Speaker notes

In englischen Texten wird meist unterschieden zwischen `attribute`, wenn zum Beispiel HTML-Attribute gemeint sind, und `property`, wenn die "Properties" von DOM-Knoten (Objekten) gemeint sind.

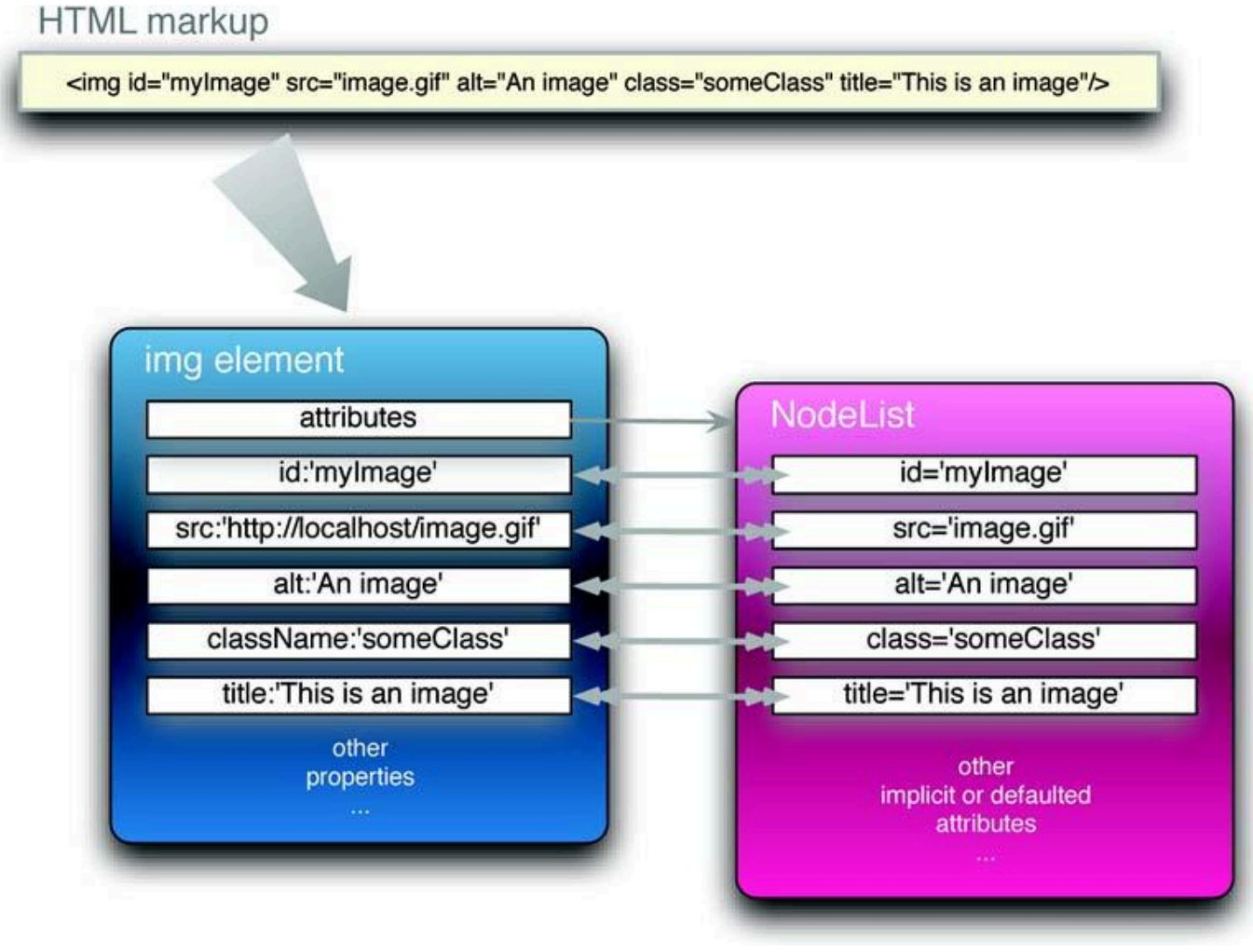
In der deutschen Sprache ist diese Unterscheidung nicht so gut möglich, da wir nicht so gern von *Objekt-Eigenschaften* sprechen. Also verwenden wir auch in diesem Fall die Bezeichnung *Attribute* und ergänzen diese in Fällen, in denen sonst Verwechslungen möglich sind. Also:

- HTML-Attribut (engl. attribute)
- Element-Attribut (engl. attribute)

sowie

- DOM-Attribut (engl. property)
- Objekt-Attribut (engl. property)

ATTRIBUTE


HTML markup

img element
attributes
id:'myImage'
src:'http://localhost/image.gif'
alt:'An image'
className:'someClass'
title:'This is an image'
other
properties
...
NodeList
id='myImage'
src='image.gif'
alt='An image'
class='someClass'
title='This is an image'
other
implicit or defaulted
attributes
...

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"
...
```

Speaker notes

Statt `class` wird im DOM `className` verwendet, weil `class` ein reservierter Name in JavaScript ist. Mit `getAttribute` und `setAttribute` kann aber auch `class` verwendet werden.

Das Bearbeiten von Klassen über `className` ist umständlich, da mit String-Verarbeitung verbunden. Aus diesem Grund wurde später `classList` hinzugefügt, das eine komfortablere Bearbeitung erlaubt.

Tipp: Damit weiss man immer noch nicht, wie die Bearbeitung der `classList` funktioniert. Als Typ wird `DOMTokenList` angegeben. Relativ schnell findet man mehr Informationen dazu in einer Referenz wie https://devdocs.io.

Auf der Seite https://devdocs.io/dom/domtokenlist erfährt man dann, dass Methoden wie `add`, `remove` und einige andere zur Verfügung stehen.

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>
```

Speaker notes

Eigene HTML-Attribute werden also nicht direkt in die DOM-Knoten übernommen. Wenn sie mit `data-` beginnen, kann man via `dataset` auf sie zugreifen. Das ist die empfohlene Variante, anwendungsspezifische Attribute einzuführen.

Alternativ kann mit `getAttribute` und `setAttribute` auf HTML-Attribute zugegriffen werden. Im Beispiel daher auch möglich:

```
if (para.getAttribute("data-classified") == "secret") {...}
```

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
```

Speaker notes

Früher entsprach ein Pixel einem Leuchtpunkt des Displays. Bei heutigen hochauflösenden Displays ist das nicht mehr der Fall, das heisst ein Pixel entspricht meist mehreren Leuchtpunkten des Displays.

Am besten findet man die Position eines Elements im Viewport mit der Methode `getBoundingClientRect` heraus, die ein Objekt mit Attributen `top`, `bottom`, `left` und `right` zurückgibt. Um die Position relativ zum gesamten Dokument zu bestimmen, muss man noch die Scroll-Positionen `pageXOffset` und `pageYOffset` addieren.

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"))
}
```

Speaker notes

Es sollen offenbar 'X'-Zeichen geschrieben werden, bis eine Breite von 2000px erreicht ist. Ganz unabhängig davon, ob das ein sinnvolles Beispiel ist: Nach jedem Schreiben von 'X' wird das Layout neu berechnet und die neue Breite des Absatzes abgefragt.

Besser: Breite eines 'X' bestimmen, einen String mit der passenden Anzahl von Zeichen erstellen und diesen aufs Mal ins DOM übernehmen:

```
let target = document.getElementById("two")
target.appendChild(document.createTextNode("XXXXX"))
let total = Math.ceil(2000 / (target.offsetWidth / 5))
target.firstChild.nodeValue = "X".repeat(total)
```

Diese Variante hat im Test 1ms benötigt gegenüber 19ms beim Schreiben der Zeichen in der while-Schleife.

Demo:
https://eloquentjavascript.net/14_dom.html#p_TzaMQEZthV

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

Speaker notes

Klar, dass das nicht nur über `class` geht. Auch das `id`-Attribut kann man anpassen. Das wird man aber seltener machen. Auch eigene Attribute mit `data–` kann man in Kombination mit Attribut-Selektoren verwenden. Oder auch Elemente wie `div` oder `span` einfügen und die Darstellung danach richten.

Die einfachste und gängigste Methode ist aber, `class` hierfür zu verwenden.

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/