

WED1 2016

VL WOCHE 8: DOM, EVENTS



M. Stolze

Übersicht

■ Theorie 1: DOM

- HTML Parsing
- DOM Abfrage & Manipulation (Übersicht)
- DOM APIs im Details
 - Wichtige globale DOM Objekte
 - Wichtige DOM Interfaces

■ Theorie 2: Events

- Registrierung von Events
- Wichtige DOM Events
- Event-Bubbling
- Event-Verarbeitung mit der Event-Queue

- **Sie können die in Vorlesung und Übungen eingeführten Konzepte und APIs für DOM Abfrage und Manipulation erklären und anwenden.**
- **Sie können die in Vorlesung und Übungen eingeführten Konzepte und Methoden des Event-Handling von JavaScript erklären und anwenden.**

DOM (Level 4) Standard API
<http://www.w3.org/TR/dom/>

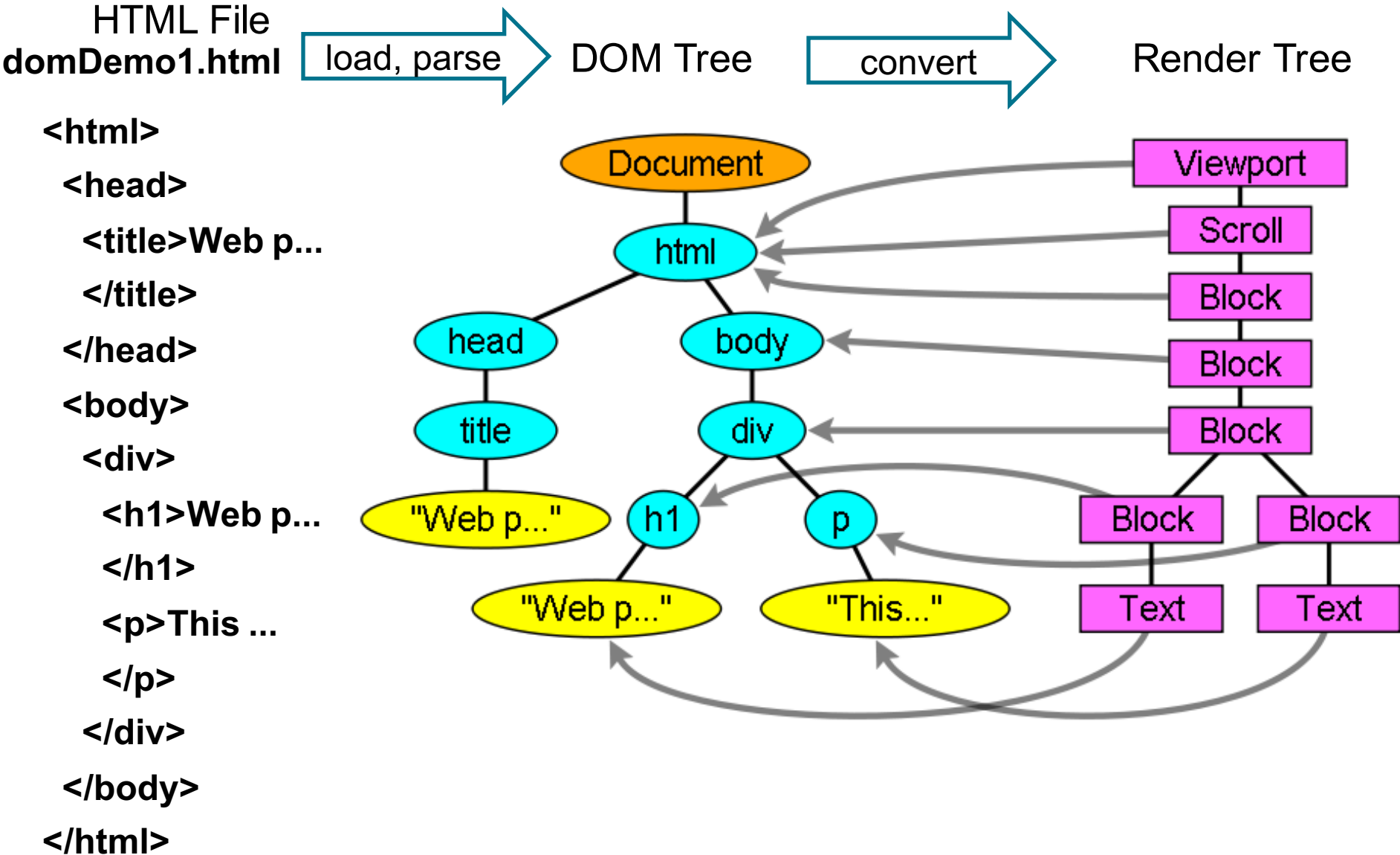


W3C DOM4

DOM Theorie: HTML wird geparst. Resultat ist das "DOM" (1)

5

Zur Anzeige nutzt die Rendering Engine den Render Tree



DOM Theorie: HTML wird geparst. Resultat ist das "DOM" (2)

6

DOM Elemente (und Render Blöcke) können analysiert werden
Änderungen am DOM führen zum Neuaufbau des Render Trees (aufwändig)



```
<html>
<head>
  <title>Web p...
</title>
</head>
<body>
  <div>
    <h1>Web p...
  </h1>
  <p>This ...
</p>
</div>
</body>
</html>
```

localhost:63342/Abend4-dom/domDemo1.html

Web page

h1 1055px x 37px basic DOM features

Elements Network Sources Timeline Profiles Resources Audits Console PageSpeed AngularJS

Styles Computed Event Listeners DOM Breakpoints Properties \$scope

▼ <!DOCTYPE html>

▼ <html>

▶ <head lang="en">...</head>

▼ <body>

▼ <div>

▼ <h1>Web page</h1>

▶ <p>This page demos basic DOM features</p>

▶ </div>

▶ </body>

▶ </html>

▼ h1

accessKey: ""

align: ""

attributes: NamedNodeMap

baseURI: "http://localhost:63342/Abend4-dom/domDemo1.html"

childElementCount: 0

childNodes: NodeList[1]

children: HTMLCollection[0]

classList: DOMTokenList[0]

className: ""

clientHeight: 37

clientLeft: 0

clientTop: 0

clientWidth: 1055

contentEditable: "inherit"

dataset: DOMStringMap

dir: ""

draggable: false

firstChild: text

firstElementChild: null

hidden: false

id: ""

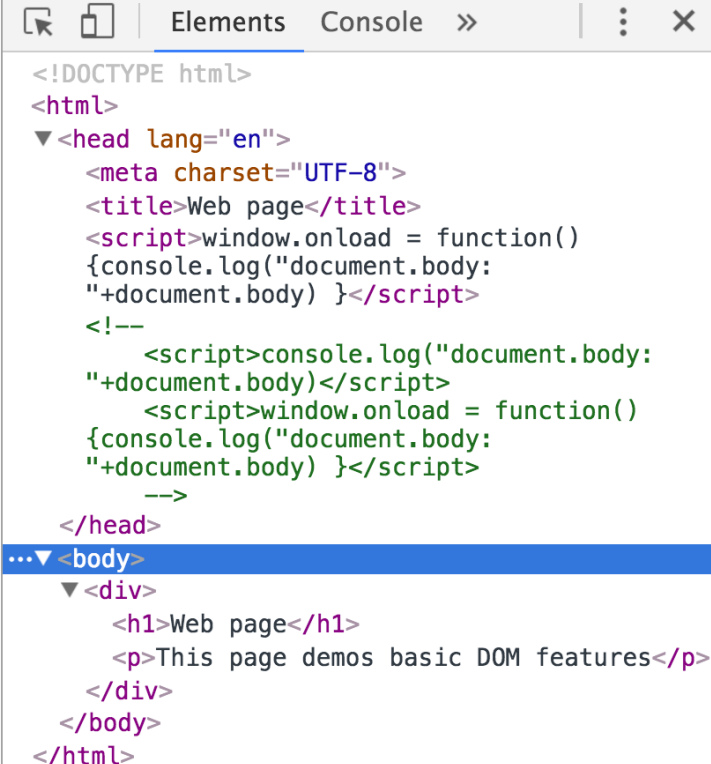
Bei korrekt geschriebenem HTML stimmt der geladene HTML Text mit der sichtbaren Repräsentation des DOM Baums überein (inkl. Kommentare etc.)

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Web page</title>
  <script>window.onload = function() {console.log("document.body: "+document.body) }</script>
  <!--
  <script>console.log("document.body: "+document.body)</script>
  <script>window.onload = function() {console.log("document.body: "+document.body) }</script>
  -->
</head>
<body>
<div>
  <h1>Web page</h1>
  <p>This page demos basic DOM features</p>
</div>
</body>
</html>
```

Wenn das HTML im File nicht dem HTML Standard entspricht kann der DOM Baum von Text im File abweichen (z.B. head & body Elemente vertauscht)

Web page

This page demos basic DOM features



```
<!DOCTYPE html>
<html>
  <head lang="en">
    <meta charset="UTF-8">
    <title>Web page</title>
    <script>window.onload = function()
    {console.log("document.body:
    "+document.body) }</script>
    <!--
      <script>console.log("document.body:
      "+document.body)</script>
      <script>window.onload = function()
      {console.log("document.body:
      "+document.body) }</script>
    -->
  </head>
  <body>
    <div>
      <h1>Web page</h1>
      <p>This page demos basic DOM features</p>
    </div>
  </body>
</html>
```

DOM Theorie: HTML wird geparkt ("DOM")

HTML-Parsing kann durch "special characters" fehlerhaft sein

8

(Chrome und Firefox)

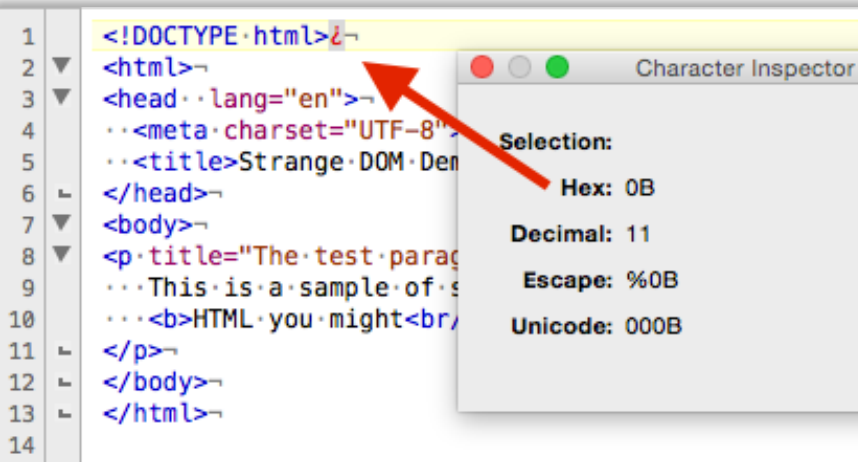
HTML File

load, parse

DOM Tree Chrome

DOM Tree Firefox

domDemo2strangeDOMTree-SpecialChar.html



```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4   <meta charset="UTF-8">
5   <title>Strange DOM Demo
6 </head>
7 <body>
8   <p title="The test paragraph"
9     This is a sample of some
10    <b>HTML you might
11  </p>
12 </body>
13 </html>
```

Character Inspector

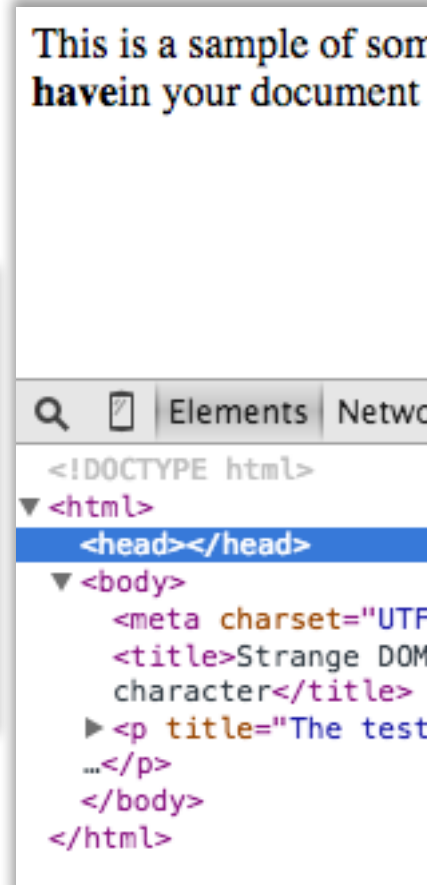
Selection:

Hex: 0B

Decimal: 11

Escape: %0B


Unicode: 000B



This is a sample of some
havein your document

Elements

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <meta charset="UTF-8"></meta>
    <title>Strange DOM Demo Page with s
    </title>
    <p id="hereweare" title="The 1
      This is a sample of some
    <b>
      HTML you might
    <br></br>
    have
    </b>
    in your document
  </p>
</body>
</html>
```



This is a sample of some **HTML y**
havein your document

Inspector

html > body > p#hereweare

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <meta charset="UTF-8"></meta>
    <title>
      Strange DOM Demo Page with s
    </title>
    <p id="hereweare" title="The 1
      This is a sample of some
    <b>
      HTML you might
    <br></br>
    have
    </b>
    in your document
  </p>
</body>
</html>
```

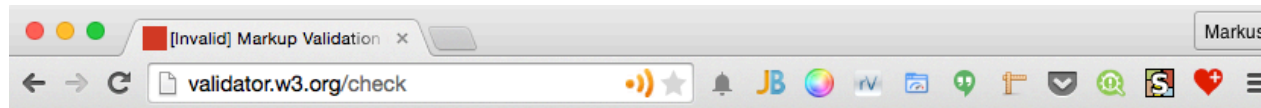

DOM Theorie: HTML wird geparkt ("DOM")


HTML-Parsing kann durch "special characters" fehlerhaft sein

9

Daher: HTML Files validieren!

<https://validator.w3.org/>





Markup Validation Service

Check the markup (HTML, XHTML, ...) of Web documents

Jump To: [Notes and Potential Issues](#) [Validation Output](#)

Errors found while checking this document as HTML5!

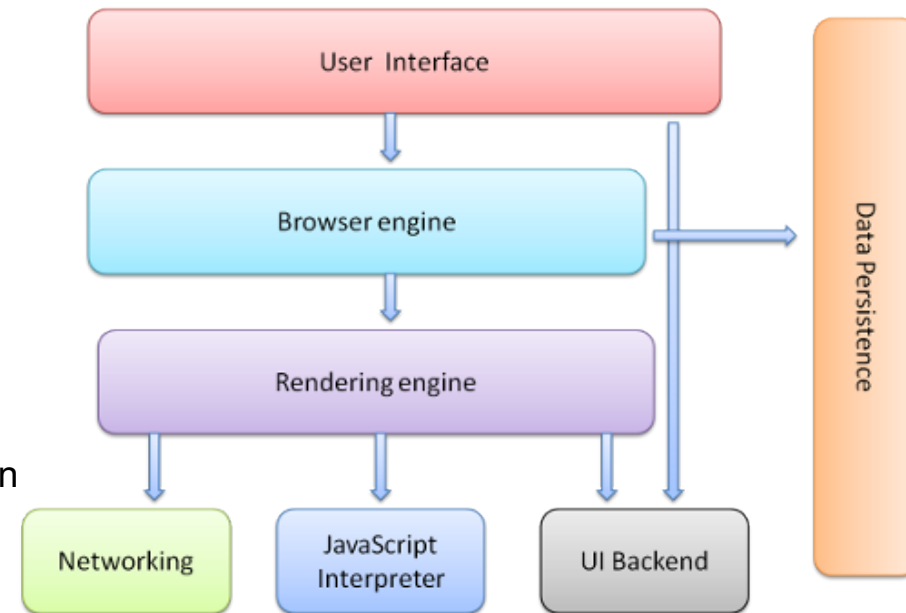
Result:	9 Errors, 3 warning(s)	
Source:	<pre><!DOCTYPE html> <html> <head lang="en"> <meta charset="UTF-8"> <title>Strange DOM Demo Page with special character</title> </head> <body> <p title="The test paragraph" id="hereweare"> This is a sample of some HTML you might
havein your document </p> </body></pre>	
Encoding:	utf-8	(detect automatically)
Doctype:	HTML5	(detect automatically)
Root Element:	html	

Validation Output: 9 Errors

✖ Line 1, Column 16: Forbidden code point U+000b.

<!DOCTYPE html>...

- Der Text im html File wird in das DOM übersetzt ("Parsing"). Dies geht Schritt für Schritt (auch Scripts)
ACHTUNG: DOM erst gesamthaft verfügbar bei/nach dem window.onload Event (später)
- Das DOM kann abgefragt werden (API)
Das globale Objekt document ist Startpunkt.
z.B. `e1 = document.querySelector('h1')`
- DOM kann mittels JavaScript manipuliert werden
z.B. `e1.innerHTML = "newText"`
- Events können an Elemente im DOM gehängt werden
z.B. `e1.addEventListener("click", ...)`



Demo: domDemo1.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Web page</title>
  <script>console.log("document.body: "+document.body)</script>
</head>
<body>
<div>
  <h1>Web page</h1>
  <p>This page demos basic DOM features</p>
</div>
</body>
</html>
```

DOM Standard API

- Grundlage: JS Einbinden
- Globale Browser Objekte
- DOM Schnittstellen
 - Document
 - Node
 - Element
 - NodeList
- DOM Manipulation
- DOM Events, Event-Registrierung, Callback-Funktionen

Grundlagen - JavaScript in HTML Seiten einbinden (1)

Script Tag: <http://www.w3.org/TR/html5/scripting-1.html#script>
Alt: <http://www.w3.org/TR/html401/interact/scripts.html#h-18.2.1>

Variante 1 (alt)

```
<script type="text/javascript" language="javascript">  
    alert("TEST");  
</script>
```

Variante 1b: HTML5 (problem in WebStorm mit

```
<script>  
    alert("TEST");  
</script>
```

Variante 2: (Beste)

```
<script src="alert.js"></script>
```

Variante 3: (Schlechteste

```
<button onclick="alert('hi hsr')">
```

```
<!DOCTYPE html>  
<html>  
<head lang="en">  
    <meta charset="UTF-8">  
    <title></title>  
    <!--1-->  
</head>  
<body>  
    <!--2-->  
    <h1>Content</h1>  
    <!--3-->  
  
</body>  
</html>
```

- JavaScript sollte idealerweise in einem separaten File unabhängig vom HTML verwaltet werden. Nur in “Schulbeispielen” und Demos ist es manchmal sinnvoll JavaScript direkt mit dem HTML zu kombinieren.
- Das .js file wird über ein script Element ins HTML File eingebunden.
- Das Script Element kann im Header stehen, es kann aber auch am Ende des Files stehen. Header: Gut zu finden. Ende: Performantes Rendering.
- Sollen DOM Manipulation im JS File gemacht werden, dann darf damit erst begonnen werden wenn das DOM fertig geladen wurde. Dafür wird am besten der window.onload Event genutzt

```
<!DOCTYPE html>
<html lang="en">
<head>
  ... <meta charset="UTF-8">
  ... <title>Title</title>
  ... <script src="hello_world.js"></script>
</head>
<body>
```

- Alle Zeilenumbrüche und Blanks zwischen Elementen werden im DOM-Baum als **Whitespace Text-Node** eingefügt
- Die Existenz dieser Whitespaces ist bei der **Navigation** und **Abfrage** im Baum zu beachten (z.B. **Node.nextSibling**)
- Beim Rendering des DOM Trees werden Whitespace Text-Nodes von Blanks und Zeilenumbrüchen gleich behandelt. Sie werden als Anstand dargestellt und benötigen etwas Platz.
- Das Resultat wirkt sich besonders beim Layout aus. Durch den zusätzlichen Whitespace kann es dazu kommen, dass 3 Elemente die je 100/3 % in der Breite zugeteilt bekommen nicht auf in eine 100% Breite passen. Dies trotz box-sizing: border-box -> Demo whitespaceDemo.html
- Abhilfe:
 - Whitespace kann mittels JavaScript aus dem DOM-Baum entfernt werden
 - Während der Minification kann Whitespace eliminiert werden
 - Die Returns mit Kommentaren umschliessen

```
<div class="box">
  <div>width: calc( 100% / 3 );
    box-sizing: border-box;</div>
</div><!--
--><div class="box">
  <div>width: calc( 100% / 3 );
    box-sizing: border-box;</div>
```

■ `console`

- Browser Console

■ `window`

- Top-Level Object (vergleichbar in Node.JS mit `global`)
- Enthält Properties für das gesamte Browser-Fenster
- Ein `window` Objekt für jedes Browser „child window“ / Tab
- `window.onload` wird aufgerufen wenn DOM fertig geparst ist

■ `document`

- Enthält Properties für die einzelnen Inhalte, wie color, links, and forms

■ `history`

- Enthält Properties, welche die zuvor besuchten Seiten repräsentieren

■ `navigator`

- Enthält Properties mit Name und Version des benutzten Browsers
- Zugriff auf Geo-Location

■ **Node**

- Basis aller Schnittstellen des DOM-Trees
- Repräsentiert einen Knoten im Baum
- Zugriff auf und Manipulation von Kindknoten

■ **Document**

- Repräsentiert das ganze HTML-Dokument (Wurzelknoten)
- Erzeugen und suchen von Knoten

■ **Element, Attr, Text**

- Knotentypen für Elemente, Attribute und Text

■ **NodeList, NamedNodeMap**

- Reordnete Liste von Knoten, Map von Knoten

■ ***Spezifikation der Schnittstellen***

- <https://www.w3.org/DOM/DOMTR>
- <https://developer.mozilla.org/en-US/docs/Web/API>

Der Wurzelknoten des geladenen Dokuments ist über das vordefinierte globale Objekt `document` verfügbar. `document` implementiert `Document`

Wichtige Eigenschaften und Methoden:

- Property: `head`
- Property: `body`
- Method: `createElement(tagName) -> Node`
- Method: `createAttribute(attributeName) -> Node`
- Method: `getElementsByTagName(tagNameStr) -> NodeList ("life")`
- Method: `getElementsByClassName(classNameStr) -> NodeList ("life")`
- Method: `getElementById(idStr) -> NodeList`
- Method: `querySelectorAll(selectorStr) -> NodeList`
(Suche im DOM Tree mit CSS Selektor Syntax; Antwort nicht "life")
- Method: `querySelector(selectorStr) -> Node`
(Suche im DOM Tree mit CSS Selektor Syntax, erster Match)

DEMO: (1) (life) getElementsByTagName vs. (nonLife) querySelectorAll

(2) Abfrage und Modifikation von Elementen (Button) nicht “einfach”¹⁸

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM API Demo</title>
  <script>
    function replaceTheButton2Text() {
      document.getElementById('button1').childNodes[0].nodeValue = text1.value;
    }
    function addNewPElement() {
      var p1 = document.createElement('p');
      p1.innerHTML = document.getElementById("text1").value;
      var emptyDiv = document.getElementById('emptyDiv');
      emptyDiv.appendChild(p1);
    }
    function setPNodeLists () {
      lifeInputsNodeList = document.getElementsByTagName('p');
      notLifeListOfInputs = document.querySelectorAll("p");
    }
  </script>
</head>
```

domDemo3domAPI.html

```
<body>
<h1>DOM API Demo</h1>
text in body <em>with emphasized text</em> and more
<p id="p1">paragraph <em>with emphasized text</em> and more</p>

<input id="text1" type="text" value="some Text"/>
<button id="button1" value="valueText">Click Me</button>
<div id="emptyDiv"></div>

</body>
</html>
```

DOM API Demo

text in body *with emphasized text* and more

paragraph *with emphasized text* and more

some Text some Text

Elements Console Sources Network Timeline Profiles Resources >>

top ▼ ☐ Preserve log

```
> addNewPElement()
< undefined

> addNewPElement()
< undefined

> setPNodeLists()
< undefined

> lifeInputsNodeList
< [ <p id="p1">...</p>, <p>some Text</p>, ● <p>some Text</p> ]

> notLifeListOfInputs
< [ <p id="p1">...</p>, <p>some Text</p>, ● <p>some Text</p> ]

> lifeInputsNodeList
< [ <p id="p1">...</p>, ● <p class="__web-inspector-hide-shortcut__">some Text</p> ]

> notLifeListOfInputs
< [ <p id="p1">...</p>, <p>some Text</p>,
  ● <p class="__web-inspector-hide-shortcut__">some Text</p> ]

> replaceTheButton2Text()
< undefined

> document.getElementById('button2').childNodes[0].nodeValue
✖ ▶ Uncaught TypeError: Cannot read property 'nodeValue' of undefined(...)
```

■ Properties (Auswahl)

- Document.activeElement
- Document.body
- Document.currentScript
- Document.documentURI
- Document.forms
- Document.head
- Document.height
- Document.lastModified
- Document.linkColor
- Document.location
- Document.onafterscriptexecute
- Document.onoffline
- Document.ononline
- Document.readyState
- Document.referrer
- Document.title
- Document.tooltipNode
- Document.URL
- Document.width

■ Methods (Auswahl)

- Document.caretPositionFromPoint()
- Document.createAttribute()
- Document.createCDATASection()
- Document.createComment()
- Document.createElement()
- Document.createTextNode()
- Document.elementFromPoint()
- Document.evaluate()
- Document.getElementById()
- Document.getElementsByTagName()
- Document.getElementsByTagName()
- Document.getSelection()
- Document.hasFocus()
- Document.queryCommandSupported()
- Document.querySelector()
- Document.querySelectorAll()

- **document** implementiert auch das **Node** Interface

- **Node Interface**

<https://developer.mozilla.org/en-US/docs/Web/API/Node>

- Property: `nodeType` (z.B. `ELEMENT_NODE`, `TEXT_NODE`)
- Property: `nodeValue` (nur bei `Text_Nodes` gefüllt)
- Property: `childNodes` (gibt `NodeList` zurück)
- Property: `firstChild`
- Property: `firstElementChild` kein Text etc.
- Property: `nextSibling` / `nextElementSibling`
- Method: `appendChild()`
- Method: `removeChild()`
- Method: `addEventListener()`

- **Beispiel: Das body-Element ist erreichbar über**

- `document.childNodes[1].childNodes[1]`
(Annahme kein **White-Space** zwischen head & body)
- `document.body`
- `document.firstElementChild.firstElementChild.nextElementSibling`

■ Properties (Auswahl)

- Node.childNodes
- parentNode.childElementCount
- Node.firstChild
- parentNode.firstElementChild
- Node.lastChild
- parentNode.lastElementChild
- Node.localName
- Node.nextSibling
- Node.nodeName
- Node.nodeType
- Node.nodeValue
- Node.ownerDocument
- Node.parentElement
- parentNode
- Node.previousSibling
- Node.textContent

■ Methods

- Node.appendChild()
- Node.cloneNode()
- Node.compareDocumentPosition()
- Node.contains()
- Node.hasChildNodes()
- Node.insertBefore()
- Node.isEqualNode()
- Node.removeChild()
- Node.replaceChild()

■ Elemente wie `document.body` implementieren das `Element` Interface

■ `Element` Interface

<https://developer.mozilla.org/en-US/docs/Web/API/Element>

- Property (r/w): `id`
- Property: `tagName`
- Property: `attributes` -> `NamedNodeMap`
- Property (r/w): `className`
- Property (r/w): `innerHTML`
- Method: `getAttribute(attrName) -> valStr`
- Method: `getAttributeNode(attrName) -> Node`
- Method: `setAttributeNode(attrNode)`
- Method: `setAttribute(attrNameStr, valueStr)`
- Method: `remove()` (wenn `ChildNode`)
- Method: `getElementsByTagName(tagNameStr) -> NodeList (live)`
- Method: `getElementsByClassName(classNameStr) -> NodeList (live)`
- Method: `getElementById(idStr) -> NodeList`
- Method: `querySelectorAll(selectorStr) -> NodeList (not live)`
- Method: `querySelector(selectorStr) -> Node`

- Elemente wie `document.getElementsByTagName(...)` gibt eine `NodeList` zurück

- `NodeList` ist KEIN Array

- Property: `length`

Siehe Demo
[domDemo4NodeListIteration.html](#)

- Iterieren über Node List

- Regulärer For Loop (Problem: Deklaration der Zählvariablen)

```
var childList = document.getElementById('pDiv').childNodes;
for (var i = 0; i < childList.length; ++i) {
    var child = childList[i];
    console.log(child.innerHTML);
}
```
- Umwandlung der `NodeList` in einen Array

```
var childList = document.getElementById('pDiv').childNodes;
var div_array = Array.prototype.slice.call(childList);
div_array.forEach((child) => {console.log(child.innerHTML)})
```
- Weitere Alternativen:
<https://developer.mozilla.org/en-US/docs/Web/API/NodeList>

DOM Manipulation (e: Element)

- `e.removeChild(<childNodes>)`
- `e.appendChild(<newDocNode>)`
`e.insertAfter(<childNodes>, <newDocNode>)`
- `e.textContent = "....."; e.innerHTML = "...."`
- Effizientes append von mehreren Elementen ohne Parent-Element


"document fragment" nutzen

```
var df = document.createDocumentFragment();
songs.forEach(function (song) {
  var liElement = document.createElement("li");
  [...]
  df.appendChild(liElement);
});
document.getElementById("songs").appendChild(df);
```



```
document.body.inner
```

```
f innerHTML (HTML<element>) string
f innerText (HTML<element>) string
^↓ and ^↑ will move caret down and up in the editor >>
```

Node.innerText  - UNOFF

Global

89.68%

A currently-nonstandard property representing the text within a DOM element and its descendants. As a getter, it approximates the text the user would get if they highlighted the contents of the element with the cursor and then copied to the clipboard.

Current aligned

Usage relative

Show all

IE	Edge [*]	Firefox	Chrome	Safari	Opera	iOS Safari [*]	Opera Mini [*]	Android Browser [*]	Chrome for Android
			47					4.3	
8			48					4.4	
9		44	49	9		8.4		4.4.4	
11	13	45	50	9.1	36	9.2	8	47	49
	14	46	51	TP	37	9.3			
		47	52		38				
		48	53						

Notes

Known issues (0)

Resources (6)

Feedback

This test only checks that the property exists and works correctly in a very simple case.

[This blog post by kangax](#) explains the history of this property, gives much more detailed cross-browser compatibility information, and gives a detailed strawman specification for the property.

`Node.innerText` is similar to, but has some important differences from, the standard `Node.textContent` property.

Mini-Quiz: HTML Parsing & Script Interpretation

Logout?

26

```
1  <!DOCTYPE html>
2  <html>
3  <head lang="en">
4      <meta charset="UTF-8">
5      <title>HTML Parsing Demo</title>
6      <script>
7          var outTxt;
8          outTxt = document.getElementById("p1").nextSibling.textContent; // -> _____
9          console.log(outTxt);
10         //outText = document.getElementById("p1")._____ // -> Text2.2
11
12         //...
15         console.log(outTxt);
16     </script>
17 </head>
18 <body>
19     <p id="p1">Text1.1<em>Text1.2</em>Text1.3</p>
20     <p>Text2.1<em>Text2.2</em>Text2.3</p>
21 </body>
22 </html>
```

domDemo5HTMLParsing.html

■ Basics

- SelfHTML

<http://wiki.selfhtml.org/wiki/JavaScript/Objekte/DOM>

- MDN DOM Introduction

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

■ Weiterführend

- MDN DOM Developer Guide

<https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM>

- MDN DOM Reference

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

DOM EVENTS

- Viele **Benutzerinteraktionen** und Zustandsänderungen im Browser werden vom Browser als **EVENT** an das `window` oder direkt an Elemente im DOM Baum geliefert.

Das heisst der Browser ruft die vom Web-Seiten Autor gesetzten oder registrierten Event Handler auf wenn der entsprechende Zustand erreicht wurde (z.B. die Seite ist fertig geladen)

- Setzen von Event Handlern Beispiel:
`window.onload = myLoadEventHandler`
- Registrieren von Event Handlern Beispiel:
`window.addEventListener("load", function (event){...})`

- Genauso wie Änderungen im Browserzustand werden auch **Benutzerinteraktionen** an die entsprechenden Elemente im DOM geliefert. Clickt ein Benutzer auf einen Button im HTML, so stellt der Browser sicher, dass alle gesetzten oder registrierten click-EventHandlerler aufgerufen werden.
- GUI Programmierung im Browser unterscheidet sich damit nicht dramatisch von anderen **GUI Frameworks**. Bei allen modernen GUI Frameworks definieren User Interface Entwickler das Verhalten des GUIs dadurch, dass Sie für GUI-Komponenten Event Handler definieren und bei diesen Komponenten registrieren. Im Gegensatz zu einem Java main Programm, ist damit die Hauptkontrolle nicht beim Code des Entwicklers, sondern beim GUI Framework. Diese Abgabe der Kontrolle wird auch das "**Hollywood Prinzip**" genannt ("don't call us, we call you")

Für die Registrierung von DOM Events gibt es folgende Möglichkeiten

- Direktes Einfügen von JS code in der **on.... Eigenschaft** des Elements z.B.

```
<button name="button1"  
  onclick="console.log('clicked '+this.name+'e: '+event)">B1</button>
```

Hierbei kann im Code sowohl auf `this` (jeweils das 'TargetElement') als auch den `event` zugegriffen werden (mehr später).
Dies Methode ist nicht empfohlen. Ausnahme: dynamisch generiertes HTML
- Zuweisung des Events Handlers z.B.

```
buttonClickHandler = function (event) { ... }  
button2.onclick = buttonClickListener;
```

Wenn in der Funktion `event` als parameter definiert wurde, dann kann im Code sowohl auf `this` (jeweils das 'TargetElement') als auch den `event` zugegriffen werden.
Diese Methode eignet sich wenn sicher nur ein Event Handler zugewiesen wird.
- Registrierung des Events Handlers z.B.

```
button3.addEventListener("click", buttonClickListener);
```

Wenn in der Funktion `event` als parameter definiert wurde, dann kann im Code sowohl auf `this` (jeweils das 'TargetElement') als auch den `event` zugegriffen werden
Diese Methode eignet sich für alle aktuellen Browser.
(nicht behandelt: Zusatzparameter `capture`, `passive` ->
Details: <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>)

■ Element MouseEvents

- click
- dblclick
- mousedown
- mouseup
- mouseenter
- mouseleave
- mousemove

■ Element (Form) KeyboardEvents

- keyup
- keydown
- keypress

■ Element ClipboardEvent

- copy
- cut
- paste

■ Element FocusEvents

- focus
- blur
- invalid
- select

■ Document Events

- DOMContentLoaded
(z.T. besser als window.onload)

■ Window Events

- load
- resize
- scroll
- error
- hashchange
- beforeprint

■ Mehr Details

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

Demo 6 Event Handler und Event Bubbling

32

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM Events Demo</title>
  <script>
    document.id="document";
    function button2ClickHandler (event) {
      console.log('(button2ClickHandler) called for target '+event.target.id);
      console.log('(button2ClickHandler) this value: '+this.id);
    }
    function button3ClickHandler (event) {
      console.log('(button3ClickHandler) called for target '+event.target.id);
      console.log('(button3ClickHandler) this value: '+this.id);
      event.stopPropagation();
      console.log('stopped propagation');
    }
    function button3ClickHandler2 (event) {
      console.log('(button3ClickHandler2) called for target '+event.target.id);
      console.log('(button3ClickHandler2) this value: '+this.id);
      event.stopPropagation();
      console.log('stopped propagation');
    }
    function documentClickHandler (event) {
      console.log('(documentClickHandler) called for target '+event.target.id);
      console.log('(documentClickHandler) this value: '+this.id);
    }

    window.onload = function () {
      var button2 = document.querySelector("#button2");
      button2.onclick = button2ClickHandler; //only a single listener possible

      var button3 = document.querySelector("#button3");
      button3.addEventListener("click", button3ClickHandler);
      button3.addEventListener("click", button3ClickHandler2);

      document.addEventListener("click", documentClickHandler);
    }
  </script>
</head>
<body>
  <button id="button1" onclick="console.log('clicked button '+this.id + ' event='+ event)">Button1
  </button>
  <button id="button2">Button2 assignend click handler</button>
  <button id="button3">button3 2 added handlers stopping propagation</button>
</body>
</html>
```

Button1 InlineCode

Button2 assignend
click handlerbutton3 2 added
handlers stopping
propagation

Elements	Console	Sources	Network	Timeline
top				
Clicked button button1 event=[object MouseEvent]				
(documentClickHandler) called for target button1				
(documentClickHandler) this value: document				
(button2ClickHandler) called for target button2				
(button2ClickHandler) this value: button2				
(documentClickHandler) called for target button2				
(documentClickHandler) this value: document				
(button3ClickHandler) called for target button3				
(button3ClickHandler) this value: button3				
stopped propagation				
(button3ClickHandler2) called for target button3				
(button3ClickHandler2) this value: button3				
stopped propagation				

- Events in Java-Script "**bubbeln**". Das heisst, als erstes werden die Event-Handler auf dem Element aufgerufen welches direkt mit der Benutzer-Interaktion verbunden sind ("Ziel-Element" z.B. Button)
- Das Ziel-Element wird am besten aus dem Event ausgelesen mit `event.target`. Dieser Event wird allen Event-Handlern beim Aufruf als Argument übergeben. Weitere nützliche Eigenschaften wie modifier Keys und Mouse-Position können aus dem Event ausgelesen werden
(siehe z.B. <https://developer.mozilla.org/en/docs/Web/Events/click>)
- Das Element auf dem der Event-Handler angehängt ist kann unter "this" oder über `event.currentTarget` zugegriffen werden.
- Solange ein Element nicht explizit im Event-Handler verhindert das ein Event "bubbled", werden nach dem Aufruf von Event-Handlern auf dem Ziel-Element der entsprechende Event auf dem Parent Element aufgerufen, und danach wieder auf dem Parent Element, bis alle Events beim document aufgerufen werden.
(**Reihenfolge von innen nach aussen**)
- Event-Handler müssen nicht auf den eigentlichen Ziel-Elementen definiert sein. Im Prinzip könnten alle Event-Handler auf dem document definiert sein.
- Abbruch des Event-bubbling wird durch Aufruf von `event.stopPropagation()` erreicht

- JavaScript ist im Normalfall **Single-Treaded** (Alternative: WebWorkers)
- **Verarbeitung eines Events** sollte **nicht länger als 100ms** benötigen, da sonst die User-Interaktion merklich blockiert wird (z.B. Buttons reagieren nicht)
- User-Events (z.B. click, mousemove, hover) und System-Events (z.B. Timer) werden alle in die gleiche **Event-Queue** eingetragen. Die Events aus der Queue werden in der Reihenfolge des Eintrags (**first-come-first-served**) bearbeitet. Erst wenn ein Event fertig bearbeitet ist wird die Bearbeitung des nächsten Events gestartet. Ein Umordnen der Event-Queue ist nicht möglich: Events können andere Events nicht überholen. Eine parallele Bearbeitung von Events ist (im Normalfall) nicht möglich.

Event Handling Demo 7: Event Queue (1)

Double Click wird (ohne Gegenmassnahme) zu zwei Klicks

35

domDemo7MoreEvents.html

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM Events Demo</title>
  <script>
    document.id="document";
    function documentClickHandler ( event ) {
      console.log('(documentClickHandler) called for target '+event.target.id);
    }
    function blockJSThreadFor( sleepDurationInMilliseconds ){
      var now = new Date().getTime();
      while(new Date().getTime() < now + sleepDurationInMilliseconds){ /* do nothing */ }
    }
    function button1ClickHandler (event) {
      // Bad practice: Event handler should not block the event thread for too long
      console.log('(button1ClickHandler) called for target '+event.target.id);
      blockJSThreadFor(2000);
      console.log('(button1ClickHandler) done ');
    }
    window.onload = function () {
      document.addEventListener("click", documentClickHandler);
      var button1 = document.querySelector("#button1");
      button1.onclick = button1ClickHandler;
      var button2 = document.querySelector("#button2");
      function button2ClickHandler (event) {
        button2.removeEventListener('click', button2ClickHandler);
        console.log('(button2ClickHandler) called for target '+event.target.id);
        event.stopPropagation();
        console.log('stopped propagation');
        blockJSThreadFor(2000);
        console.log('(button2ClickHandler) done ');
        //button2.addEventListener('click', button2ClickHandler); //does not work
        setTimeout(function() {
          button2.addEventListener('click', button2ClickHandler);
        }, 0)
      }
      button2.addEventListener("click", button2ClickHandler);
    }
  </script>
</head>
<body>
  <button id="button1">LongBlockingButton</button>
  <button id="button2">DoubleClickResistentButton</button>
</body>
</html>
```

Event Handling Demo 7: Event Queue (2)

Long Running Tasks mit dem Timer Schritt-für-Schritt bearbeiten

36

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM Events Demo</title>
  <script>
    document.id="document";
    function documentClickHandler ( event ) {
      console.log('(documentClickHandler) called for target '+event.target.id);
    }
    function blockJSThreadFor( sleepDurationInMilliseconds ){
      var now = new Date().getTime();
      while(new Date().getTime() < now + sleepDurationInMilliseconds){ /* do nothing */ }
    }
    function startLongRunning () {
      var theCounter=0;
      var keepRunning = true;
      var stopLongRunning = function () {console.log("stop"); keepRunning=false};
      var button4 = document.querySelector("#button4");
      button4.addEventListener("click", stopLongRunning);
      var longRunning = function () {
        blockJSThreadFor(500);
        theCounter++;
        console.log("did it "+theCounter+" times");
        if (keepRunning) {
          setTimeout(longRunning, 0);
        }else{
          button4.removeEventListener("click", stopLongRunning);
        }
      };
      longRunning();
    }
    window.onload = function () {
      document.addEventListener("click", documentClickHandler);
      var button3 = document.querySelector("#button3");
      button3.addEventListener("click", startLongRunning);
    }
  </script>
</head>
<body>
  <button id="button3">Start LongRunning</button>
  <button id="button4">StopLongRunning</button>
</body>
</html>
```

domDemo8EvenMoreEvents.html

- Events auf korrekte Schreibung überprüfen:
Testen ob Event Handler überhaupt aufgerufen wird
(console.log oder Break-Point)
 - Events mit on (oder ohne) an der falschen Stelle
 - Events beim falschen Objekt (es gibt kein document.onload)