

CAS FEE 17

ABEND 04: DOM-API, JQUERY TEMPLATING

Markus Stolze

31.6.2017



Inhaltsverzeichnis

- **Callbacks, Events, Frameworks**
- **Dev Tools Links**
- **DOM API Basics**
- **DOM Event Handling**
- **Single Page App Architektur I: MVC**
- **JQuery**
- **Templating mit Handlebars**

Callbacks, Events, Frameworks (& Libraries)

Bekannt: Funktionen als Parameter (und Werte und Return Werte)

```
function add(a, b) {  
    return a + b;  
}  
  
function minus(a, b) {  
    return a - b;  
}  
  
function calc(fn, a, b) {  
    console.log(fn(a, b));  
}  
  
calc(add, 3, 4);  
calc(minus, 3, 4);
```

```
const opsMap = {'+': add, '-': minus};  
  
function calc2 (opStr, a, b) {  
    const opFn = opsMap[opStr] || add;  
    console.log( 'calc2', opFn(a, b) );  
}  
calc2('+', 3, 4);  
calc2('-', 3, 4);  
  
function createCalculator (opsMap2, calcName='Calculator') {  
    return function (opStr, a, b) {  
        const opFn = opsMap2[opStr] || add;  
        console.log(calcName, opFn(a, b));  
    };  
}  
  
const calculator = createCalculator({'+plus': add, '-minus': minus});  
  
calculator('+plus', 3, 4);  
calculator('-minus', 3, 4);
```

1_Callbacks_Events/1_functionAsParameters.js

Neu: Callback Funktionen

```
function announceFinished () {
    console.log('we are done waiting');
}
console.log('timer started');
setTimeout(announceFinished, 1000);

function countDown (start, waitTime) {
    waitedTimes = 0;
    const waitOnce = function () {
        waitedTimes++;
        if (waitedTimes < start) {
            console.log('Counting Down: '+ (start-waitedTimes)); //why brackets?
            setTimeout(waitOnce, waitTime);
        }else{
            console.log('we are done waiting');
        }
    };
    setTimeout(waitOnce, waitTime);
}

countDown(5, 1000);
```

1_Callbacks_Events/2_timerCallback.js

Neu: UI-Event-Callback Funktionen in Node: Readline Event-Registrierung

```
const readline = require('readline');
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
  prompt: 'CAS-FEE HW CLI> '
});
rl.on('line', (line) => {
  switch(line.trim()) {
    case 'hello':
      console.log('world!');
      break;
    default:
      console.log(`Say what? I might have heard '${line.trim()}'`);
      break;
  }
  rl.prompt();
});
rl.on('close', () => {
  console.log('Have a great day!');
  process.exit(0);
});

rl.prompt();
```

1_Callbacks_Events/3_helloCLI.js

Neu: Event Callback Funktionen im Browser: Event

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>HalloWelt0</title>
</head>
<body>
  <h1>Hallo Welt 0</h1>
  <label>
    Geben Sie ihren Namen ein:
    <input id="name" autofocus>
  </label>
  <button id="greetBtn">Grüss Mich</button>
  <div id="greetDiv"></div>
  <script>
    const input = document.getElementById('name');
    const greetOutput = document.getElementById('greetDiv');
    const button = document.getElementById('greetBtn');
    button.addEventListener('click', (event) => {
      greetOutput.innerHTML = 'Hallo ' + input.value;
    });
  </script>
</body>
</html>
```

Frameworks vs. Libraries

■ Framework

- Entwickelter Code registriert sich beim Framework
- Entwickelter Code wird vom Framework aufgerufen
- Kontrolle beim Frameworks
- Beispiele:
 - Browser-Events
 - Node: Konsole-API, Networking, Files,

■ Library

- Entwickelter Code ruft Library-Funktionen auf
- Entwickelter Code hat Kontrolle
- Beispiel:
 - JS: Math
 - Underscore

Developer Tools

■ DevTool Features

- Element Tab (select & modify DOM-Element, Inspect API)
- Console Tab (\$0,

■ Dok: <https://developers.google.com/web/tools/chrome-devtools/>

- <https://developers.google.com/web/tools/chrome-devtools/console/>
- <https://developers.google.com/web/tools/chrome-devtools/console/console-reference>
- <https://developers.google.com/web/tools/chrome-devtools/console/command-line-reference>

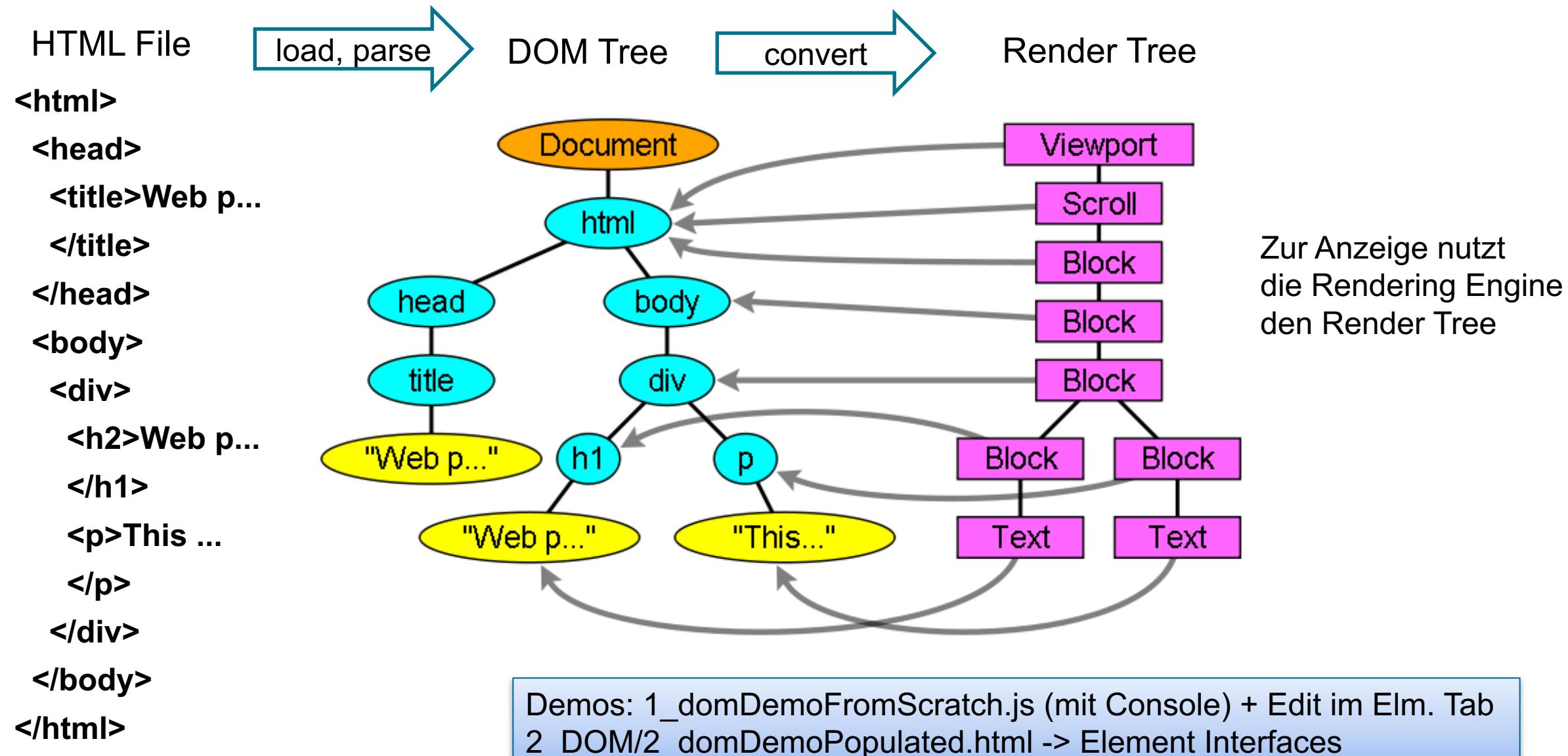
■ Tutorial 1: <https://egghead.io/courses/using-chrome-developer-tools-elements>

■ Tutorial 2: <https://www.codeschool.com/courses/discover-devtools>

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



HTML to DOM parsing: HTML wird geparsst. Resultat ist das "DOM"



DOM "from Scratch" Demo

```
// about:blank
document.body.insertAdjacentHTML('beforeend', '<h2>dom demo 1</h2>');
let btnCnt = 0;
function addBtn() {
    document.body.insertAdjacentHTML('beforeend', `<button id="btn${btnCnt++}">click me</button>`);
}
addBtn();
//document.head.insertAdjacentHTML('beforeend', '<title>dom demo 1</title>');
//$0 & $(selector) are chrome console functions
//$0.onclick = function () {console.count('click')}
//$('button').onclick = function () {console.count('clack')}
//$('#btn0').onclick = function () {console.count('clack')}
//$('#btn0').onclick = function () {console.count('clack2', addBtn())}
```

DOM Abfrage und Manipulation im Browser – Übersicht

- 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 bzw. document.DOMContentLoaded (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 bei Elementen im DOM registriert werden
z.B. e1.addEventListener("click", ...)

```
<!DOCTYPE html>           Demo: 2_DOM/2_domDemoPopulated.html
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM API Demo</title>
</head>
<body>
<div>
  <h1>DOM API Demo</h1>
  <p>This page demos basic DOM features</p>
  <p>The following DOM interfaces are most important</p>
  <ul>
    <li>Document</li>
    <li>Node</li>
    <li>Element</li>
    <li>NodeList</li>
  </ul>
  <p>The following global browser variables are most important</p>
  <ul>
    <li>document</li>
    <li>window</li>
    <li>console</li>
  </ul>
  <h2>UI Elements</h2>
  <label>Input <input id="inpt1"></label>
  <p>Button <button id="btn1">press me</button></p>
</div>
</body>
</html>
```

HTML to DOM parsing: Gutes HTML -> DOM entspricht weitgehend der HTML Vorlage

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)

3_domDemoStrangeDOMTree-SpecialChar.html

The screenshot shows a browser window with developer tools open. On the left, a "Web page" view displays the text "This page demos basic DOM features". On the right, the "Elements" tab of the developer tools shows the DOM tree. The tree structure is as follows:

- <!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> (highlighted in blue)
- ▼ <div>
- <h1>Web page</h1>
- <p>This page demos basic DOM features</p>
- </div>
- </body>
- </html>

HTML to DOM parsing: HTML-Parsing kann durch "special characters" fehlerhaft sein

(Chrome und Firefox)



3_domDemoStrangeDOMTree-SpecialChar.html

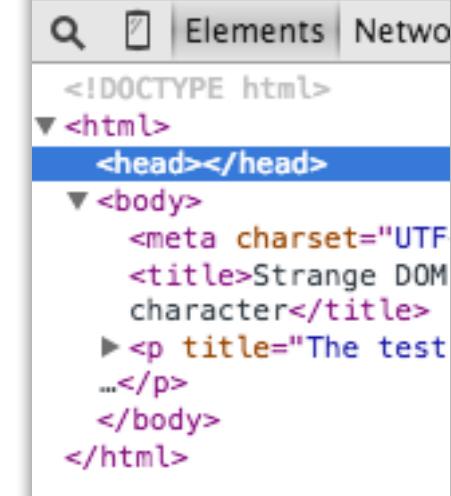
```

1  <!DOCTYPE html>
2  <html>
3  <head>· lang="en">
4  ···<meta charset="UTF-8">
5  ···<title>Strange DOM Den
6  </head>
7  <body>
8  <p>· title="The test para
9  ···This is a sample of s
10 ···<b>HTML you might<br/>
11 </p>
12 </body>
13 </html>
14

```

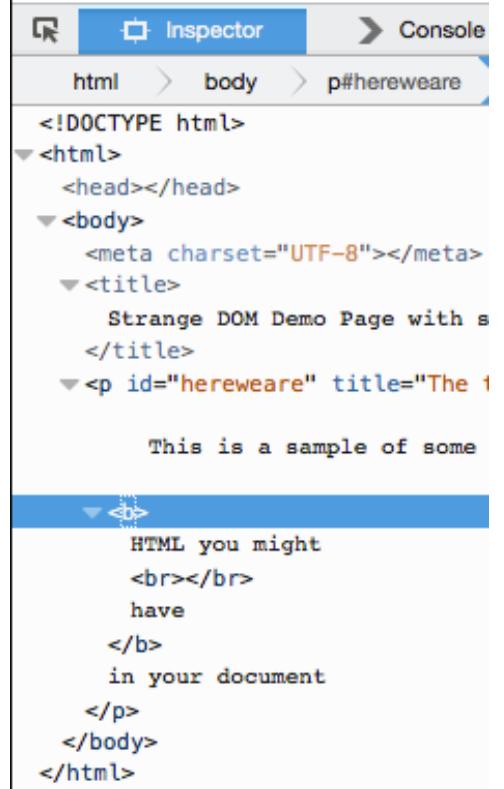
DOM Tree Chrome

This is a sample of som
havein your document



DOM Tree Firefox

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



HTML to DOM parsing: Validierung ist wichtig

W3C Validierungs-Service: <https://validator.w3.org/>

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "[Invalid] Markup Validation". Below that is the URL "validator.w3.org/check". The main area has a blue header with the W3C logo and "Markup Validation Service". A sub-header says "Check the markup (HTML, XHTML, ...) of Web documents". Below this are three buttons: "Jump To: Notes and Potential Issues", "Validation Output", and "Validation Output". A red banner at the top of the main content area says "Errors found while checking this document as HTML5!". It displays the following information:

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 <input type="button" value="detect automatically"/>
Doctype :	HTML5 <input type="button" value="detect automatically"/>
Root Element:	html

Validation Output: 9 Errors

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

<!DOCTYPE html> ...

Validierung im Build Prozess z.B. mit
<https://www.npmjs.com/package/valimate>

HTML Parsing & Whitespace

- Zeilenumbrüche und Blanks zwischen Elementen
-> DOM-Baum enthält **Whitespace Text-Nodes**
- Whitespace Text-Nodes bei **Navigation** und **Abfrage** beachten
(daher besser **Node.nextElementSibling** als **Node.nextSibling** nutzen)
- des DOM Tree Rendering
-> Whitespace Text-Nodes (auch mehrere) als einfacher Abstand dargestellt
- Mögliches Layout Problem:
Umbruch-Fehler durch zusätzlichen Whitespace.
Z.B. 3 div mit **box-sizing: border-box**; und **width: calc(100% / 3)**.
-> 2_DOM/4_whitespaceDemo.html
- Abhilfe:
 - Whitespace mittels JavaScript aus dem DOM-Baum entfernen
 - 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>
```

DOM Standard API

DOM Standard API

- Globale Browser Objekte
- DOM Schnittstellen / Interfaces
 - Document
 - Node
 - Element
 - NodeList
- DOM Manipulation
- DOM Events, Event-Registrierung, Callback-Funktionen

Wichtige globale BROWSER Objekte

■ **console**

- Browser Console (log(), ...)

■ **window**

- Top-Level Object (vergleichbar in Node.JS mit **global**)
- Enthält Properties / Methoden für das gesamte Window (z.B. scrollTo(), print())
- Löst den **window.onload** Event/Fn aus wenn DOM fertig geparsst ist

■ **document**

- Startpunkt für DOM Queries/Navigation und Erstellung neuer Elemente
- Löst den **document.onDOMContentLoaded** Event/Fn aus wenn DOM fertig geparsst ist
- Mehr später

■ **location**

- Enthält Properties basierend auf der momentanen URL

■ **history**

- Enthält Properties, welche die zuvor besuchten Seiten repräsentieren

■ **navigator**

- Enthält Properties mit Name und Version des benutzten Browsers

Wichtige DOM Schnittstellen in der Übersicht

■ **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>

document (globales Objekt) , Document (DOM Interface)
<https://developer.mozilla.org/en-US/docs/Web/API/document>

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) -> Node
- **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)

DOM API Document Anfragen & Manipulation (Properties + Methods)

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

■ Properties (Auswahl)

- Document.activeElement
- **Document.body**
- Document.documentElement
- 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.createAttribute()
- Document.createCDATASection()
- Document.createComment()
- **Document.createElement()**
- **Document.createTextNode()**
- **Document.getElementById()**
- **Document.getElementsByClassName()**
- **Document.getElementsByTagName()**
- Document.getSelection()
- Document.hasFocus()
- **Document.querySelector()**
- **Document.querySelectorAll()**

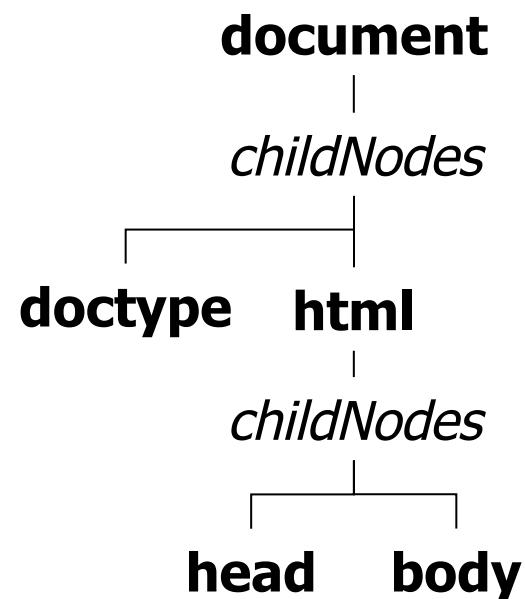
Node (DOM Interface)

- **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`

DOM API Node Anfragen & Manipulation (Properties + Methods)

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

■ Properties (Auswahl)

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

■ Methods

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

Element (DOM Interface)

■ 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**
- **Property (r/w): outerHTML**
- **Method: `getAttribute(attrName)`** -> `valStr` // value as found in file
- **Method: `getAttributeNode(attrName)`** -> `Node`
- **Method: `setAttributeNode(attrNode)`**
- **Method: `setAttribute(attrNameStr, valueStr)`**
- **Method: `Element.insertAdjacentElement(posStr, htmlStr)`**
//posStr: 'beforebegin', 'afterbegin', 'beforeend', 'afterend'
- **Method: `Element.insertAdjacentText(posStr, textStr)`**
//posStr: 'beforebegin', 'afterbegin', 'beforeend', 'afterend'
- **Method: `remove()`** (wenn `ChildNode`)
- **Method: `getElementsByTagName(tagNameStr)`** -> `NodeList`
- **Method: `getElementsByClassName(classNameStr)`** -> `NodeList`
- **Method: `getElementById(idStr)`** -> `NodeList`
- **Method: `querySelectorAll(selectorStr)`** -> `NodeList`
- **Method: `querySelector(selectorStr)`** -> `Node`

NodeList (DOM Interface)

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

- Methoden wie `document.getElementsByTagName(...)` geben eine **NodeList** zurück

- NodeList ist KEIN Array**

- Aber hat **Property:** `length`

- Iterieren über Node List

- Regulärer For Loop

```
for (let i = 0; i < my NodeList.length; ++i) {
    const item = my NodeList[i];
}
```

- Umwandlung der NodeList in einen Array

```
const divList = document.querySelectorAll('div');
const divArray = [... div_list];
//divArray.forEach(
//    divElement => divElement.innerHTML = '888');
//oder
for (divElement of divArray) {
    divElement.innerHTML = '888';
}
```

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>DOM API Demo</title>
    <script>
        const overwritePs = function () {
            const p NodeList = document.querySelectorAll('p');
            for (let i=0; i < p NodeList.length; i++) {
                const p Element = p NodeList[i];
                p Element.innerHTML='8888';
            }
        };
        const overwriteDivs = function () {
            const replaceText=theInput.value;
            for (dElement of [...document.querySelectorAll('div')]) {
                dElement.innerHTML=replaceText;
            }
        };
        window.onload = function () {
            setTimeout(overwritePs, 2000);
            theButton.onclick =overwriteDivs;
        }
    </script>
</head>
<body>
    <h1>DOM API Demo: Node List Traversal</h1>
    <p>p1</p>
    <p>p2</p>
    <p>p3</p>
    <div>d1</div>
    <div>d2</div>
    <div>d3</div>
    <label>Type the text to replace in Divs
        <input id="theInput">
    </label>
    <button id="theButton">Overwrite Divs</button>
</body>
</html>
```

DOM Manipulation mit DOM Elementen

DOM Manipulation (e: Element)

- e.insertAdjacentHTML(posStr, htmlStr)
//posStr: 'beforebegin', 'afterbegin', 'beforeend', 'afterend'
- e.insertAdjacentText(posStr, textStr)
- e.insertAdjacentElement(posStr, element)

- e.removeChild(<childNodes>)

- e.appendChild(<newDocNode>)
e.insertAfter(<childNodes>, <newDocNode>)

- e.textContent = "....."; e.innerHTML = "...." ; e.innerHTML += "...."

- **Effizientes append von mehreren Elementen:**
"document fragment" nutzen
const df = document.createDocumentFragment();
songs.forEach(function (song) {
 const liElement = document.createElement('li');
 [...]
 df.appendChild(liElement);
});
document.getElementById('songs').appendChild(df);

DOM API Anfragen

Wichtig: X-Browser Support beachten: zB. innerText

`document.body.inner`

f innerHTML (HTMLElement)

string

f innerText (HTMLElement)

string

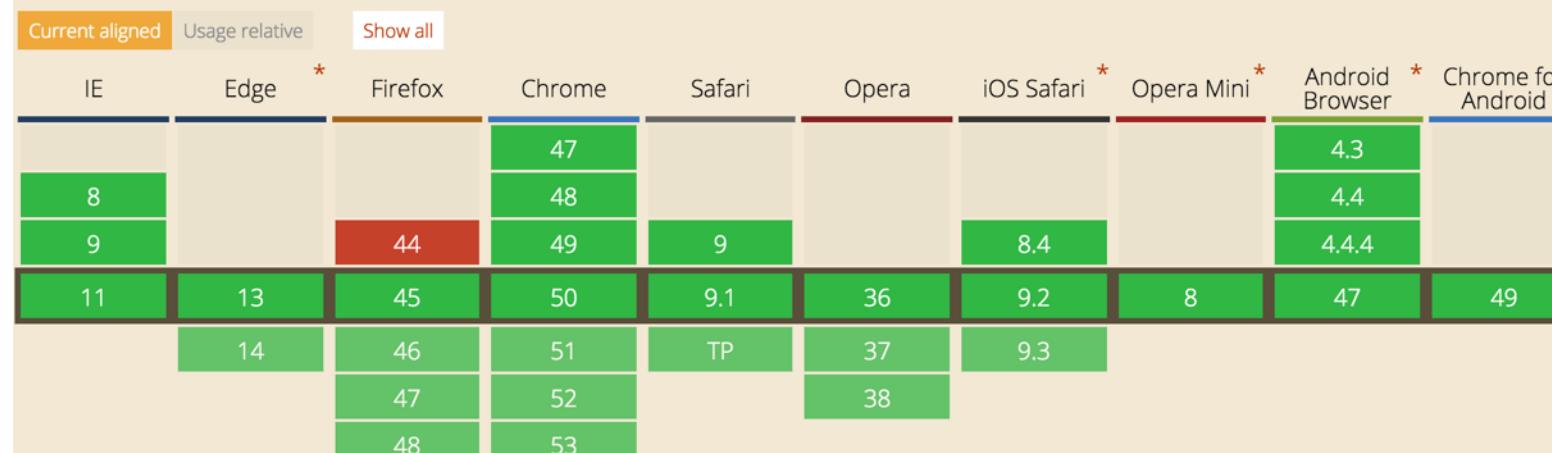
^↓ and ^↑ will move caret down and up in the editor [»](#)

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.



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: DOM API

LogOutput?

```
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM API Quiz</title>
  <script>
    let outTxt = document.getElementById('p1').nextSibling.textContent;
    console.log(outTxt); // -> _____
    //outText = document.getElementById('p1')._____._____ // -> Text2.2
    //outTxt = document.getElementById('p1').nextElementSibling.firstElementChild.textContent;
    //console.log(outTxt); // -> _____
  </script>
</head>
<body>
  <p id="p1">Text1.1<em>Text1.2</em>Text1.3</p>
  <p>Text2.1<em>Text2.2</em>Text2.3</p>
</body>
</html>
```

DOM Loaded Events: window.onload oder document Listener für DOMContentLoaded

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>domLoadedDemo</title>
  <script>
    console.log('headScript executing');
    document.onload = _ => console.log('onload executed'); // does not fire
    document.onDOMContentLoaded = _ => console.log('onDOMContentLoaded executed'); // does not exist / fire
    document.addEventListener('DOMContentLoaded',
      _ => console.log('document DOMContentLoaded executed'));
    window.onload = _ => console.log('window.onload executed');
    console.log('headScript ended');
  </script>
</head>
<body>
  <h1>Dom Loaded Demo</h1>
  <script>
    console.log('endScript executing');
  </script>
</body>
</html>
```

2_DOM/6_domLoadedDemo.html

headScript executing
headScript ended
endScript executing
document DOMContentLoaded executed
window.onload executed

Literatur DOM API

■ 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

Aufgabe: Music01InnerHTMLBasicExercise

```
<body>
<h1>Songs</h1>
<ul id="songs"></ul>
<script>
    // Model
    const songs = [
        {"title": "Thank you for the music", "artist": "ABBA"}, ...
        {"title": "Beds Are Burning", "artist": "Diesel and Dust"}
    ];

    // Controller
    function createSongsHtml (songs) {
        let htmlString = "";
        songs.forEach(function (song) {
            /*
                TODO create html string with a li element for each song with an h3 for title and p for artist
                htmlString += ...;
                htmlString += ...;
                ...
            */
        });
        return htmlString;
    }
    // TODO: wait for window load, then add the generated html to the ul (songs) element using the innerHTML property
</script>
</body>
</html>
```

Aufgabe: Music02AppendChildBasicExercise

```
<script>
    // Model
    const songs = [
        {"title": "Thank you for the music", "artist": "ABBA"},

        ...
        {"title": "Beds Are Burning", "artist": "Diesel and Dust"}
    ];

    // Controller
    function appendSongsLiElements (songs) {
        songs.forEach(function (song) {
            const liElement = document.createElement("li");
            // TODO create and add h3 with title and p with artist info
            document.getElementById("songs").appendChild(liElement); // better at the end
        });
    }
    window.onload = function(){
        appendSongsLiElements(songs);
    };
</script>
```

DOM EVENTS

Globale Browser-Objekte und DOM-Events

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

- Um vom Browser informiert zu werden ist eine Registrierung als Event-Listener nötig ('Hollywood Prinzip', HTML/JS = GUI-FRAMEWORK)

z.B.: die Seite ist fertig geladen -> Zustandsänderung

- Registrieren des Event Handlern Beispiel 1: `window.onload = myLoadEventListerner`
- Registrieren von Event Handlern Beispiel 2:
`window.addEventListener('Load', myLoadEventHandler)`

- z.B.: Nutzer hat einen Button geklickt -> Benutzerinteraktion

- Registrieren des Event Handlern Beispiel 1: `button1.onclick = myLoadEventListerner`
- Registrieren von Event Handlern Beispiel 2:
`button1.addEventListener('click', myClickEventHandler)`

Registrierung von DOM EventHandlern

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 -> JSLint Warnung. Möglich in dynamisch generiertem HTML (ohne "this")

- 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>)

Wichtige DOM Events

■ 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

Event Handler, Callbacks & Closures

- Event Handler werden in JavaScript durch Zuweisung oder Hinzufügen von Callback-Funktionen definiert.

```
h2e.addEventListener('click', myClickEvantHandler);  
h2e.onclick = myClickEvantHandler ;
```

- JavaScript Funktionen (und damit auch Callback Funktionen) können Werte aus ihrem Definitions-Kontext kapseln -> Closure

```
function attachCountingClickEventHandler(attachToElement, messageToElement) {  
    let count = 0;  
    attachToElement.onclick = function () {  
        messageToElement.innerHTML += '#' + count++;  
    };  
}  
attachCountingClickEventHandler(btn1, out);
```

2_DOM/2_domDemoPopulated.html

De-Registrierung von Event-Listenern

- Event-Listener sind nicht mehr aktiv wenn ein Element aus dem DOM-Tree entfernt wird.
Es können aber Memory-Leaks entstehen bei dieser Art der "harten" Deregistrierung

Beispiel

```
h2e = document.querySelector('h1');
h2e.onclick = _ => console.count('clicked'+e.target.outerHTML)
h2e.outerHTML = h2e.outerHTML;
```

- Besser: explizite De-Registrierung

- button1.onclick = null;
- button1.removeEventListener('click', button1EventListener)
ACHTUNG Referenz auf Event-Listener ist nötig!

Event Handler, Nutzung von Closures (1)

- Folgendes Beispiel funktioniert nicht (Webseite mit 10 p-Elementen)

2_DOM/8_domEventsClosure_V1.html

```
pElementsNodeList = document.querySelectorAll('p');
let i=0;
for (pElement of [...pElementsNodeList]) {
    pElement.onclick = function () {console.log('I am P-Element #' + i)};
    i++;
}
```

- Lösung: Closure 1

2_DOM/8_domEventsClosure_V2.html

```
function createIndexedEventListener(index) {
    return function () {console.log('I am P-Element #' + index)}
}
window.onload = function () {
    pElementsNodeList = document.querySelectorAll('p');
    let i=0;
    for (pElement of [...pElementsNodeList]) {
        pElement.onclick = createIndexedEventListener(i);
        i++;
    }
}
```

Event Handler, Nutzung von Closures (2)

■ Problem (rep.)

```
let i=0;
for (pElement of [... document.querySelectorAll('p')]) {
    pElement.onclick = function () {console.log('I am P-Element #' + i)};
    i++;
}
```

■ Alternative Lösung 2: Closure 2

```
window.onload = function () {
    pElementsArray = [...document.querySelectorAll('p')];
    pElementsArray.forEach((e, i) =>
        e.onclick = () => console.log('I am P-Element #' + i));
};
```

8_domEventsClosure_V3

■ Alternative Lösung 3: Closure 3 (jedes let/const generiert eine Closure)

```
window.onload = function () {
    pElementsNodeList = document.querySelectorAll('p');
    for (let i = 0; i < pElementsNodeList.length; i++) {
        pElementsNodeList[i].onclick =
            () => console.log('I am P-Element #' + i);
    }
};
```

8_domEventsClosure_V4

Event Handler, Nutzung von data-attributes

- Folgendes Beispiel funktioniert nicht (Webseite mit 10 p-Elementen)

8_domEventsClosure_V1

```
pElementsNodeList = document.querySelectorAll('p');
let i=0;
for (pElement of [...pElementsNodeList]) {
    pElement.onclick = function () {console.log('I am P-Element #' + i)};
    i++;
}
```

- Lösung: Nutzung von data-attributes (diese könnten auch im html z.B. direkt mit data-index spezifiziert werden)

```
window.onload = function () {
    pElementsNodeList = document.querySelectorAll('p');
    let i=0;
    for (pElement of [...pElementsNodeList]) {
        pElement.dataset.index = i;
        pElement.onclick = function (e) {console.log('I am P-Element #' + e.target.dataset.index)};
        i++;
    }
}
```

8_domEventsClosure_V5

Event Bubbling

■ User-Events in Java-Script "bubbeln":

- User-Events werden nicht nur auf dem Target-Element ausgelösst, sondern auch auf allen Parent-Elementen.
- Reihenfolge: Von innen nach aussen bis zum document
- event.target enthält das originale Ziel-Element.
- Abbruch des Event-bubbling wird durch Aufruf von e.stopPropagation() möglich

■ Typischer Anwendungsfall:

Registrierung von Event-Handlern bei Parents eines häufig ersetzen DOM-Unterbaumes

- Neu-Registrierung von Event-Handlern nach Ersatz eines DOM-Unterbaums nicht nötig
- Beispiel: Ersetzen eines DOM-Unterbaums mit einem HTML-String oder Rendering eines Templates

Beispiel

```
document.body.onclick = myClickListener;  
h2e = document.querySelector('h1');  
h2e.outerHTML = e1.outerHTML;  
document.body.onclick = function(e)  
{console.log('clicked'+count+e.target.outerHTML)};
```

"this" in Event-Listenern

- "this" in Event-Listenern
 - = DOM-Element welches den Event ausgelöst hat (d.h. das DOM-Element bei dem der EventListener registriert wurde)
- Event-Listener erhalten beim Aufruf als erstes Argument den aktuellen Event (= Objekt mit Informationen zum aktuellen Event)
- **e.target =**
DOM-Element mit dem Nutzer interagiert hat
=> Bei "bubbled" Events ist event.eventTarget != this
- **e.currentTarget ist immer = this =** DOM-Element dem der EventListener registriert wurde
- **Empfehlung:**
Lieber mit e.target oder e.currentTarget arbeiten als mit "this"

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOM Events Demo</title>
  <script>
    function buttonClickListener (e) {
      console.log('e.target.id=' + e.target.id);
      console.log('this.id=' + this.id);
      console.log('e.currentTarget.id=' + e.currentTarget.id);
    }
    window.onload = function () {
      par1.addEventListener("click", buttonClickListener);
      button1.addEventListener("click", buttonClickListener);
    }
  </script>
</head>
<body>
  <p id="par1">
    <button id="button1">button1</button>
  </p>
</body>
</html>
```

Event Queue und der JS-Thread

- 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 ACHTUNG

■ Fakten

- **JavaScript ist "single threaded"**
- **Timer-Events werden erst behandelt wenn alle User-Interactions (clicks, keys) abgearbeitet worden sind.**
- **Event-Handler können den DOM-Baum inklusive registrierte Event-Handler verändern. Der veränderte Status des DOM Baums wird schon während der Behandlung von Events wie click berücksichtigt**

■ Empfehlungen

- **Event-Handler sollten nach spätestens ca. 100ms (1/10 sec) terminieren, sonst wird das UI merklich “unresponsive”**

Achtung in den Übungen

- 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)

Aufgabe: Music03InnerHTML+InlineEventsExercise

```
<script>
  const songs = [
    {"id":"01", "title":"Thank you for the music", "artist":"ABBA", "rating":3}, ...
  ];
  function createSongsHtml (songs) {
    let htmlString = "";
    songs.forEach(function (song) {
      htmlString += "<li><h3>";
      htmlString += song.rating;
      /* TODO add a button "+" with a rateSongPlus called in the inline click handler and a corresponding "-" button */
      ....
    });
    return htmlString;
  }
  function compareSongs(s1, s2) { ... }
  function findSong(id) {...}
  function rateSongPlus (songId) {
    /*
    TODO
    - use findSong to find the song
    - increase song rating
    - call renderSongs
    */
  }
  function rateSongMinus (songId) { ... }
  function renderSongs () {
    /*
    TODO sort the songs array (use the Array.prototype sort function and the compareSongs function)
    */
  }
  document.getElementById("songs").innerHTML=createSongsHtml(songs);
```

Aufgabe: Music04InnerHTML+BubblingEventsExercise

```
<script>
    // Model
    const songs = [ ...],
    // Controller
    function createSongsHtml (songs) {
        let htmlString = "";
        songs.forEach(function (song) {
            htmlString += "<li><h3>";
            htmlString += song.rating;
            /* TODO add + and - button each with an attribute data-songId containing the songId */
            ...
        })
        ...
        function bubbledClickEventHandler(event) {
            //takes advantage of event bubbling
            // TODO
            // - get the button as the click-target from the event
            // - read button text using textContent
            // - get the value of the data-songId attribute from the button
            // - based on button text call rateSong with the right parameters
        }
        function renderSongs () {...}
        window.onload = function() {
            renderSongs();
            // TODO register bubbledClickEventHandler with an element that receives all bubbled click events from
        }
    }
</script>
```

SINGLE PAGE APP ARCHITEKTUR: MVC

Traditionelle GUI Architektur: Model View Controller

■ Model

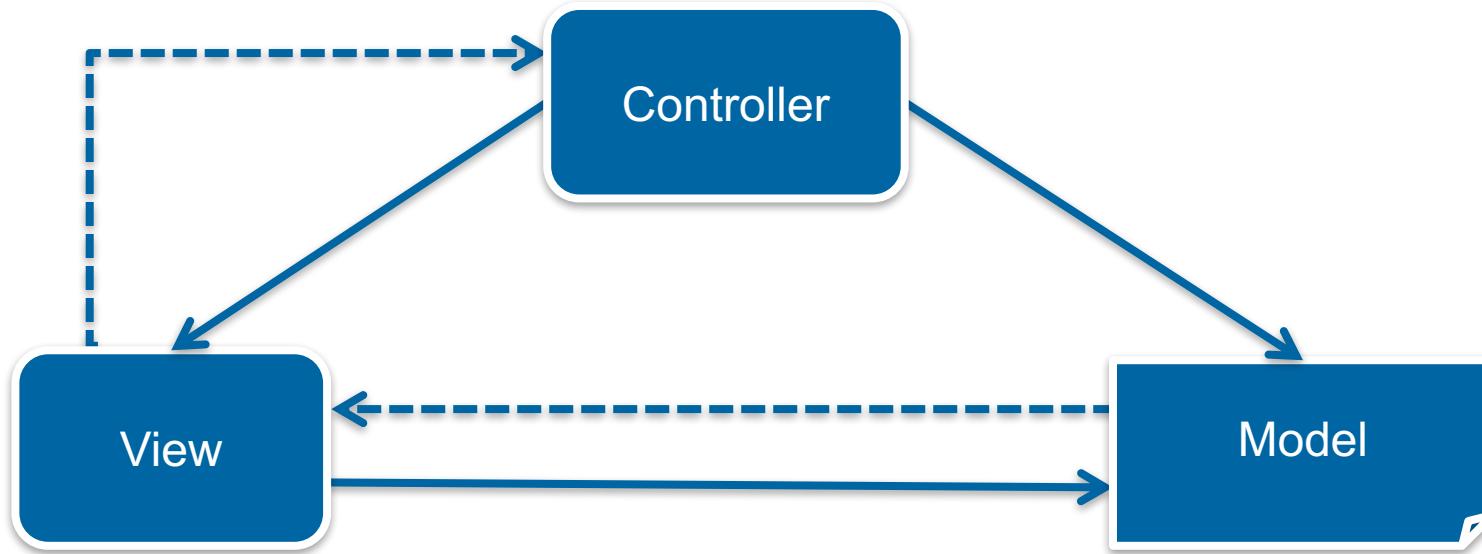
Data Model (App State)

■ View

Anzeige

■ Controller

- User Input
- Business Logic
- View-Wechsel (Routing)



Unterschiedlichste Patterns zur Verbindungen Komponenten bekannt
Zum Teil andere Benennung

- Model-View-Presenter (MVP),
- Model-View-ViewModel (MVVM)

Erweiterung nötig bei verteiltem System mit Server Access

Model View Controller im Web UI (ohne Network / AJAX)

■ Model

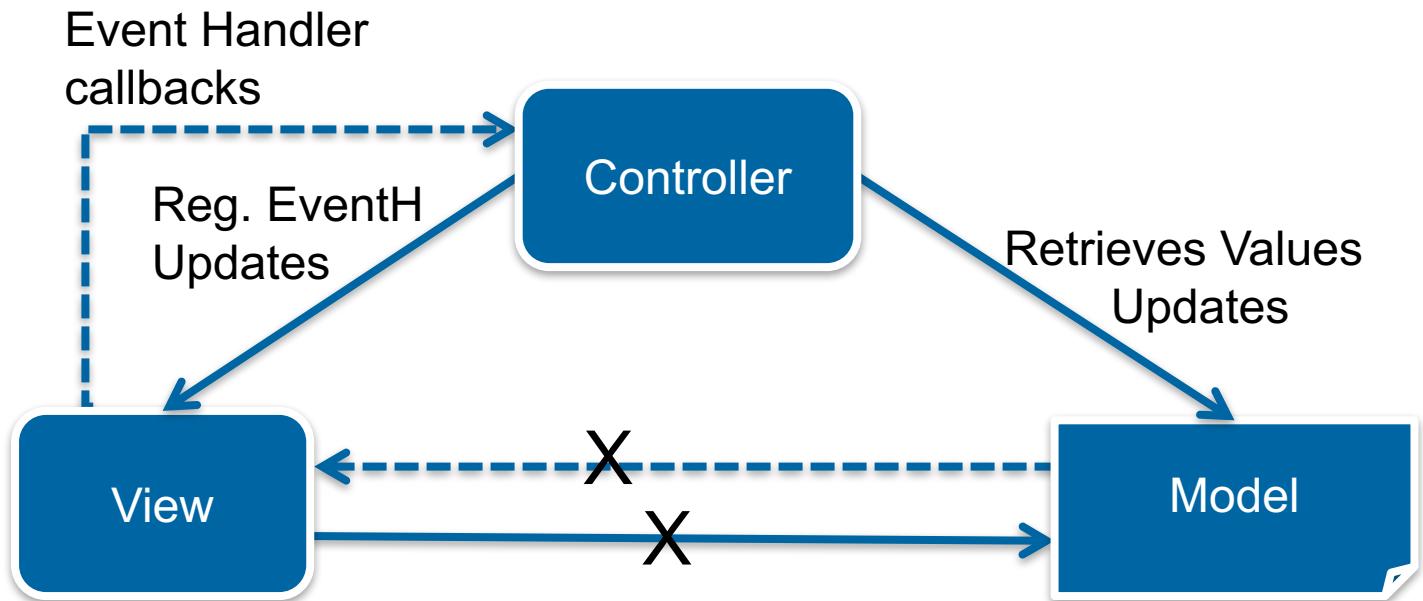
Data Model (App State)

■ View

■ DOM

■ Controller

- User Input: Event Handler
- Business Logic (?)
- View-Wechsel (Routing)



Mögliches Problem: "Massive View Controller"
Auch im Web unterschiedliche Patterns möglich
(z.T. durch Framework gegeben)
Erweiterung nötig bei Network Access / AJAX
z.B. Model wird Teil eines DataManager mit Callbacks

Beispiel Counter App – Simpel

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>LayeredCount1</title>
</head>
<body>
  <h1>Layered Count 1</h1>
  <p>
    <button id="upBtn">
      Count Up</button>
  </p>
  <p>
    <button id="downBtn">
      Count Down</button>
  </p>

  Current Count:
  <div id="countDiv">0</div>

  <script> ...</script>
</body>
</html>
```

```
window.onload = function () {
  // core // model
  let count = 0;
  function countUp () {
    count++
  }
  function countDown () {
    count--
  }
  // UI
  const countDiv = document.getElementById('countDiv');
  const upButton = document.getElementById('upBtn');
  const downButton = document.getElementById('downBtn');
  const renderUI = function () {
    countDiv.innerHTML = count;
  };
  upButton.onclick = function () {
    countUp();
    renderUI();
  };
  downButton.onclick = function () {
    countDown();
    renderUI();
  };
};
```

4_MVC/1_MvcCountDemo1.html

Beispiel Counter App – Komplexer: Objekte

```
// core // model
const countModel = {
  count: 0,
  countUp: function () {
    this.count++
  },
  countDown: function () {
    this.count--
  },
};

// UI
const makeCountController = function (model, upButton, downButton, countDiv){
  const countController = {
    renderUI: function () {
      countDiv.innerHTML = model.count;
    }
  };
  upButton.onclick = function () {
    model.countUp();
    countController.renderUI();
  };
  downButton.onclick = function () {
    model.countDown();
    countController.renderUI();
  };
  return countController;
};

let countControl = makeCountController(
  countModel,
  document.getElementById('upBtn'),
  document.getElementById('downBtn'),
  document.getElementById('countDiv') );

countControl.renderUI();
```

4_MVC/2_MvcCountDemo2.html

Beispiel Counter App – Noch Komplexer: Klassen, **Observer**, Objekte

```
class Observable {  
    constructor () {  
        this.observers = [];  
    }  
    addObserver(observer) {  
        this.observers.push(observer);  
    }  
    notifyObservers () {  
        this.observers.forEach(  
            observer=>observer.update())  
    }  
}  
// core // model  
class CountModel extends Observable {  
    constructor (countInit) {  
        super();  
        this.count = countInit;  
    }  
    countUp () {  
        this.count++;  
        this.notifyObservers();  
    }  
    countDown () {  
        this.count--;  
        this.notifyObservers();  
    }  
}  
const countModel = new CountModel(0);
```

```
class CountController {  
    constructor (model, upButton, downButton, countDiv){  
        model.addObserver(this);  
        this.model = model;  
        upButton.onclick = function () {  
            this.model.countUp();  
        }.bind(this);  
        downButton.onclick = function () {  
            this.model.countDown();  
        }.bind(this);  
    }  
    renderUI () {  
        countDiv.innerHTML = this.model.count;  
    }  
    update () {  
        this.renderUI();  
    }  
}  
let countControl = new CountController(  
    countModel,  
    document.getElementById('upBtn'),  
    document.getElementById('downBtn'),  
    document.getElementById('countDiv') );  
  
countControl.renderUI();
```

Class noch nicht behandelt

Navigation in Single Page Apps

- History API kann genutzt werden um Navigation zu simulieren
- Die aktuelle URL kann ohne Verlassen der Seite über Navigation zu einem Anchor auf der gleichen Seite geändert werden.
``
Die Navigation wird mit dem window.onhashchange Event abgefangen
- Alternativ kann über history.pushState die angezeigte URL geändert werden (hier feuert window.onhashchange nicht)

Idee für diese Demo:

Navigation for Single Page Applications Curran Kelleher
<https://www.youtube.com/watch?v=xN9QxPtK2LM>

```
<head>
  <meta charset="UTF-8">
  <title>SPA Navigation Demo</title>
  <script>
    window.onload = function() {
      // view template
      const partials = {
        home: "this is home", about: "this is about", contact: "this is contact"
      };
      // view template access
      const getCurrentPartial = function() {
        const hash = location.hash || "#home";
        return partials[hash.substr(1)];
      };
      //navigation controller
      const content = document.getElementById('content');
      const navContact = document.getElementById('navContact');
      navContact.addEventListener('click',
        function(e) {
          e.preventDefault();
          history.pushState(null, 'Contact', "#contact");
          renderCurrentPartial() });
      const renderCurrentPartial = function() {
        content.innerHTML = getCurrentPartial();
      };
      window.addEventListener("hashchange", renderCurrentPartial);
      //init
      renderCurrentPartial();
    };
  </script>
</head>
<body>
  <h1>Simple Singlepage Demo</h1>
  <a href="#home">home</a><a href="#about">about</a><a id="navContact" href="">contact</a>
  <div id="content"></div>
</body>
</html>
```

JQUERY

Inhaltsverzeichnis

- **jQuery Einleitung**
- **Ready vs. Load**
- **IIFE & noConflict**
- **jQuery**
 - **DOM Anfragen**
 - **DOM Manipulation**
 - **Style Manipulation**
 - **DOM Event Handling**
 - **Pitfalls**

JQuery Einleitung

- **JQuery** = weit verbreitete JS Library (Open Source, MIT Lizenz) Version im April17: 1.12.4 und 3.2.1
 - Wichtigste Funktion: Die JQuery Funktion = \$

```
> $  
< function ( selector, context ) {  
    // The jquery object is actually just the init constructor 'enhanced'  
    // Need init if jquery is called (just allow error to be thrown if not included)  
    return new jQuery.fn.init( selector, context );  
}  
> jQuery  
< function ( selector, context ) {  
    // The jquery object is actually just the init constructor 'enhanced'  
    // Need init if jquery is called (just allow error to be thrown if not included)  
    return new jQuery.fn.init( selector, context );  
}  
.
```

- Einfache, code-effiziente X-Browser-konsistente Abfrage und Manipulation des DOMs
Resultat einer JQuery Anfrage ist ein JQuery Result Set (Collection/Array)
=> Unterstützt JQuery Methoden
=> "function chaining" möglich (funktionaler Stil)
z.B. Disable alle Input-Felder und Buttons
\$(':input').prop('disabled', true)
- Einfache, code-effiziente Registrierung von Event Handlern
- Weitere Utility Funktionen..

- **Falls nur ein sehr kleinen Teil von jQuery genutzt wird ist reines JavaScript sinnvoller**
 - Falls nur wenige DOM-Manipulationen getätigt werden
 - Falls nur eine Animation benötigt wird.
 - Falls nur der AJAX Befehl genutzt wird. => später
- **Beispiele für jQuery vs. JavaScript**
 - <http://youmightnotneedjquery.com/>

JQuery Einbinden & Starten

■ Einbinden der JQuery Lib:

(Local): <script src="js/jquery-3.2.1.min.js"></script>

(CDN): <script src="https://code.jquery.com/jquery-3.2.1.min.js">
</script>

■ Startpunkt (meist) statt window.onload

in JQuery:

```
$(document).ready(function () {  
    ...  
});
```

oder (äquivalent)

```
jQuery(function ($) {  
    ...  
});
```

oder (noch sicherer)

```
(function($, window, document) {  
    $(function() {  
        ...  
    });  
    }(window.jQuery, window, document));
```

Startpunkt

- **Einsatz von jQuery macht in den meisten Fällen erst Sinn nachdem das DOM bereit ist...**
 - ... deshalb auf das DOM-Ready Event warten
- **ready wird ausgeführt falls DOMContentLoaded ausgelöst wurde.**
 - Falls DOMContentLoaded schon gefeuert wurde – wird diese Funktion sofort aufgerufen!
 - Vorteil da normalerweise diese Funktion ignoriert werden würde.

```
$ (document) .on ( 'ready' ,  (function () {  
  
    // code ...  
    $ ("#container") .text ("Hello world");  
}) ;  
  
// oder kürzer  
  
$ (function () {  
  
    // code ...  
    $ ("#container") .text ("Hello world");  
}) ;
```

JQUERY: DOM ELEMENTS SELECTION

DOM Anfragen

- **jQuery erlaubt es CSS-ähnliche Selektoren zu definieren**
 - jQuery nutzt hierfür möglichst die native Funktion document.querySelectorAll(...)
- **Beispiele**

```
$("p > em");
$("#container");
$(".alert");
```
- Es ist auch möglich die Suche einzuschränken
 - Beispiel: Alle Elemente mit der Klasse die unter dem angegebenen jQuery Objekt liegen:

```
var container = $("#container");
$(".alert", container);
```
- Das Resultat aller Anfragen ist eine Liste von gefunden Elementen (JQuery Result-List)
 - Jedes Element ist ein jQuery Objekt mit allen jQuery Funktionen
 - Das native DOM Objekt erhält man mit `var domElm = $(".alert").get(index)` oder `$(".alert")[index]`
 - `$(domElm)` erzeugt aus einem nativen Element ein jQuery Objekt
 - `.length` gibt die Anzahl von gefunden Elementen zurück
 - Aktionen die auf der Result-List ausgeführt werden, werden auf alle Elemente in der Liste angewendet.
 - Diese Resultat kann verwendet werden um weiter zu navigieren z.B. zum Parent / zu den Kindern.

Details: <http://api.jquery.com/category/selectors/>

```
<!DOCTYPE html>
<html lang="en">
```

jQuery/02_jQuerySelectionDemo.html

```
<head>
  <meta charset="UTF-8">
  <title>WED1 JQuery Selection Demo</title>
  <script src="jquery-1.12.4.js"></script>
  <script>
    jQuery(function($) {
      console.log(1, $("p"));
      console.log(2, $("p > span"));
      console.log(3, $("button span"));
      console.log(4, $("p > span").eq(2));
      console.log(5, $("p > span:nth-child(2)"));
      console.log(6, $("p > span:contains('Query')"));
    })
  </script>
</head>
<body>
```

JQuery DOM Selection

```
<p>
  <span>DOM</span>
  <span>Selection</span>
  <span>with</span>
  <span>JQuery</span>
</p>
<p>
  <button>press <span>me</span></button>
</p>
<p>
  <label>enter name<input />
  </label>
</p>
<div id="output">
```

Wichtig:

- Basis CSS Selektoren
- :nth-child(n); nth-last-child(n)
- :contains('searchText')
- :input
- eq(n)
- first()

1	► [p, p, p, prevObject: jQuery.fn.init(1), context: document, selector: "p"]	02_jQuerySelectionDemo
2	► [span, span, span, span, prevObject: jQuery.fn.init(1), context: document, selector: "p > span"]	02_jQuerySelectionDemo
3	► [span, prevObject: jQuery.fn.init(1), context: document, selector: "button span"]	02_jQuerySelectionDemo
4	► [span, prevObject: jQuery.fn.init(4), context: document]	02_jQuerySelectionDemo
5	► [span, prevObject: jQuery.fn.init(1), context: document, selector: "p > span:nth-child(2)"]	02_jQuerySelectionDemo
6	► [span, prevObject: jQuery.fn.init(1), context: document, selector: "p > span:contains('Query')"]	02_jQuerySelectionDemo

```
<!DOCTYPE html>          jQuery/03_jQueryTravDemo.html
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>WED1 JQuery Navigation Demo</title>
  <script src="jquery-1.12.4.js"></script>
  <script>
    jQuery(function($) {
      console.log(1, $("p"));
      console.log(2, $("p").parent().prop('tagName'));
      console.log(3, $("p").prev().prop('tagName'));
      console.log(4, $("p").next().prop('tagName'));
      console.log(5, $("p").parent().prop('tagName'));
      console.log(6, $("p").children().prop('tagName'));
      console.log(2, $("p > span").nth(2).next());
    })
  </script>
</head>
<body>
  <h1>JQuery DOM Navigation</h1>
  <p>
    <span>DOM</span>
    <span>Navigation</span>
    <span>with</span>
    <span>JQuery</span>
  </p>
</body>
</html>
```

Wichtig:

- parent()
- prev()
- next()
- children()

JQUERY: DOM MANIPULATION

- **jQuery bietet einen einfachen Syntax zur Abfrage und Manipulation von DOM –Element Attributen**

- **val()**
 - Abfragen und Setzen von Werten von Form-Elementen
 - Abfrage: Wert des ersten Elements in der Result-List; Setzen: Werte aller Elemente in der Result-List auf den Wert setzen.
 - **text()**
 - Abfragen und Setzen von Werten von nicht-Form-Elementen
 - Abfrage: Konkatenierte Texte aller Elemente in der Result-List (inkl. Sub-Elemente); Setzen: Texte aller Elemente in der Result-List auf den Wert setzen.
 - **html()**
 - Abfrage: HTML-Inhalt des ersten Elements; Setzen: HTML Inhalt jedes Elements in der Liste wird gesetzt.
 - **prop()**
 - Abfrage: Eigenschafts-Wert des ersten Elements; Setzen: Attribut-Werte jedes Elements in der Liste wird gesetzt.
 - **data()** (achtung: nicht interoperabel mit element.dataset)
 - Abfrage: data-* Attribute +JQ-Data des ersten Elements als Objekt; Setzen: JQ-Data jedes Elements in der Liste wird gesetzt.
 - Andere nützliche Funktionen
 - **hide(), remove(), append(), toggle()**
- **\$(“html“) -> neue Element Struktur**
 - Beispiel: `$(“”)` => neues ul-element
 - Funktioniert auch mit ganzen HTML-Strukturen, z.B. `$(“myItem”)`

Details: <http://api.jquery.com/category/manipulation/>

```
<!DOCTYPE html>      jQuery/04_jQueryManipulationDemo.html
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>WED1 JQuery Manipulation Demo</title>
  <script src="jquery-1.12.4.js"></script>
</head>
<body>
<h1>JQuery DOM Manipulation</h1>
<p>
  <span>DOM</span>
  <span>Selection</span>
  <span>with</span>
  <span>JQuery</span>
</p>
<p>
  <button>press <span>me</span></button>
</p>
<p>
  <label>enter name<input />
  </label>
</p>
<div id="out"></div>
<pre>
...
</pre>
</body>
</html>
```

Wichtig:

- css()
- prop()
- val()
- text()
- append()
- replaceWith()

```
$( 'span' ).css('font-style', 'italic');
$( 'span' ).css('font-style', 'normal');
$( 'button span' ).css('font-weight', 'bolder');
$( 'button span' ).css('font-weight', 'normal');
$( 'p > span:nth-last-child(2)' ).css('font-style', 'italic');
$( 'p > span:nth-last-child(2)' ).css('font-style', 'normal');
$( 'button' ).prop('disabled', true);
$( ':input' ).prop('disabled', true);
$( 'input' ).val('hello');
$( '#out' ).text($('input').val());
$( '#out' ).append($('
```

JQuery Style Manipulation

- JQuery bietet einen einfachen Syntax zur Styling Manipulation und Animation
 - Classen
 - addClass (name)
 - removeClass (name)
 - CSS Eigenschaften ändern
 - css(property-name, value)
 - Definiert diese Werte als Inline-Style
 - CSS-Animationen
 - fadeIn() / fadeOut() / fadeToggle()
 - slideUp() / slideDown
 - ...

```
<head>        4_jQuery/05_jQueryStyleManipulationDemo.html
...
<style>
    .cool {background-color: aqua;}
</style>
</head>
<body>
<h1>JQuery Style Manipulation</h1>
<p>
    <span>Style</span><span>Manipulation</span><span>with</span>
</p>
...
<script>
    function animatePs () {
        const jQPs = $('p');
        jQPs.slideUp('slow', _=>jQPs.addClass("cool")); //no th
        jQPs.slideDown('slow');
    }
</script>
</body>
</html>
```

JQUERY: EVENT HANDLING

■ **jQuery Event-Registrierung mit .on() (alternativ: <event>()**

- .click() oder besser .on("click", ...)
- .hover() oder besser .on("hover", ...)
- .focus() oder besser .on("focus", ...)

■ **.on() ermöglicht Optionen (z.B. Event-Bubbling + 'delegates')**

- `$(body).on(eventsStr [, delegate-selector-string] [, data], handlerFn)`
 - eventsStr: String mit Liste der Events (mit einem Leerzeichen getrennt)
 - **delegate-selector-string** : cssSelectrString: nur für Elemente auf die der Selektor zutrifft wird die handlerFn aufgerufen
 - data: optionals (json) Object welches zum Zeitpunkt des Event-Handlings in der handlerFn unter event.data genutzt werden kann (z.B. für index etc)
 - Handler: HandlerFn die aufgerufen wird – wenn derEvent ausgelöst wird.
- Hier wird die handlerFn direkt auf dem Body Element registriert.
- Alle Events welche auf den Kinder-Elementen ausgelösst werden gelangen zum Parent-Element (Event-Bubbling)
 - Vorteil: Falls ein neues Child hinzugefügt wird ist die handlerFn auch für dieses Element aktiv! Selektion mit selectorStr
 - Wichtig für Template Engines => Nächste Woche
- <http://api.jquery.com/on/>

■ **Die .off() Funktion kann die Registration wieder rückgängig machen (Referenz zur handlerFn ist notwendig)**

- <http://api.jquery.com/off/>

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>WED1 JQuery Manipulation Demo</title>
  <script src="jquery-1.12.4.js"></script>
</head>
<body>
<h1>JQuery DOM Manipulation</h1>
<p>
  <span>DOM</span>
  <span>Selection</span>
  <span>with</span>
  <span>JQuery</span>
  <span><button>another button</button></span>
</p>
<p>
  <button>press <span>mee</span></button>
</p>
<p>
  <label>enter name<input />
  </label>
</p>
<div id="out"></div>
<pre>
...
</pre>
</body>
</html>
```

Wichtig:

- on(handlerFn)
- on(delegateSelectStr, handlerFn)
- off(handlerFn)
- once(handlerFn)

```
$('body').on('click', 'button', (e)=>console.log(e.target));
$('body').on('click', 'p > button', (e)=>console.log(e.target));
$('body').on('click', 'p > button', ()=>$('#out').text($('input').val()));
$('span > button').on('click', (e)=>console.log(1, e.target));
$('span > button').replaceWith($('<button>another button</button>'));
$('p > button').replaceWith($('<button>press meehee</button>'));
```

addEventListener(DOMContentLoaded) vs. ready(...)

```
document.addEventListener("DOMContentLoaded", function () {
    console.log("DOMContentLoaded");
}) ;
$(document).on('ready', function () {
    console.log("ready");
}) ;

setTimeout(function () {
    document.addEventListener("DOMContentLoaded", function () {
        console.log("setTimeout - DOMContentLoaded");
    }) ;
    $(document).ready(function () {
        console.log("setTimeout - ready");
    }) ;
}, 10);
```

Output:

ready

DOMContentLoaded

setTimeout - ready



`$.ready` vs. `DOMContentLoaded` vs. `onload` vs. `load`

■ Load Events werden erst ausgelöst, wenn der gesamte Content geladen ist

- Bilder
- Flash
- Vorteil: Es ist ALLES geladen
- Nachteil: Es muss ALLES geladen sein.

■ Ready Events

- Keine Garantie für das Bilder schon vorhanden sind.
- Frühester Zeitpunkt welcher DOM Manipulationen möglich sind.
- In den meisten Fällen die bessere Wahl.

\$.ready vs. DOMContentLoaded vs. onload vs. load

```
window.onload = function () {
    console.log("load", "window.onload");
};

$(window).on('load', (function() {
    console.log("load", "$.load");
})__);

window.addEventListener('load', function () {
    console.log("load", "window.addEventListener");
    console.log("img height", $("img").first().height());
}, false);

$(function() {
    console.log("ready", "$.ready");
    console.log("img height", $("img").first().height());
});

document.addEventListener('DOMContentLoaded', function () {
    console.log("ready", "document.addEventListener");
}, false);
```

ready \$.ready
ready img height 0
ready document.addEventListener
load window.onload
load \$.load
load window.addEventListener
load img height 100

JQUERY: IIFE & NOCONFLICT

■ Immediately-invoked function expression

- https://en.wikipedia.org/wiki/Immediately-invoked_function_expression

■ Eine Funktion die sofort ausgeführt wird.

■ Vorteile:

- Es wird ein neuen Scope definiert
- Variablen und Funktionen innerhalb eines IIFE sind nicht global
- Weniger wichtig mit ES6 let -> block-scope

```
(function() {
```

```
    // eigener Scope
```

```
)();
```

noConflict

- Problem: Das \$ Zeichen wird auch von anderen Frameworks genutzt.
- jQuery kann einen «noConflict» Modus aktivieren.
 - \$ wird wieder auf die alte Referenz gelegt.
 - jQuery ist nur noch über jQuery zugreifbar.
 - Es kann ein anderer Alias vergeben werden.

```
var $j = jQuery.noConflict();
```
- Lösung mit IIFE
 - (**function** (\$) {

 // \$ ist jQuery

}) (**jQuery**);
 - Details: <https://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries>
 - Lösung über JQuery Parameter Injection

```
jQuery(function($) {  
  
    // $ ist jQuery  
    console.log($(".em"));  
  
});
```

JQUERY PITFALLS & MORE

JQuery Pitfalls & Antipatterns

Pitfalls

- <http://www.codeproject.com/Articles/346904/Common-Pitfalls-of-jQuery>
- **Wichtige Punkte / Achtung bei**
 - Unnötigen jQuery Abfragen statt Zwischenspeichern von Resultaten
 - Ineffziente Selektoren, z.B. #elMlds nicht genutzt um Suche einschränken
 - Shortcuts (wie hide, show, toggle, empty) nicht genutzt -> ineffizient & unleserlich
 - Repetitive Selektoren

Antipatterns

- <http://blog.javascripting.com/2015/01/12/real-world-javascript-anti-patterns-part-two/>
- SCHLECHT: .addClass und .removeClass statt .toggleClass
- SCHLECHT: registrieren von generischen Handlern, dann Switch/Case zu Runtime
- GUT: Nutzung von Event-Namespaces zu Gruppen-de-Registrierung von Handlern \$(window).off('.mycomponent')

Aufgabe: Music05JQuery+BubblingEventsExercise

```
<!DOCTYPE html>
]<html>
]<head lang="en">...
]<body>

<h1>Songs</h1>
<ul id="songs"></ul>

<script src="jquery-3.2.1.js"></script>

]<script>
]  const songs = [...];
]  function createSongsHtml (songs) {...}
]  function compareSongs(s1, s2) {...}
]  function findSong(id) {...}
]  function rateSong (songId, delta) {...}
]  function bubbledClickEventHandler(event) {...}
]  function renderSongs () {
]    //TODO use JQuery to do the equivalent of
]    //document.getElementById("songs").innerHTML=createSongsHtml(songs.sort(compareSongs));
]
]  }
]$(<function>{
]  renderSongs();
]  //TODO use JQuery to do the equivalent of
]  // document.getElementById("songs").addEventListener("click", bubbledClickEventHandler);
])
]</script>

]</body>
]</html>
```

JQuery Weiterführende Informationen

- <https://learn.jquery.com/>
- <https://api.jquery.com/>

TEMPLATING AUF DEM CLIENT



```
var createNoteHtml_M = function (id, content, dueDate) {
    var startHTML = '<li>\n<strong>' + content + '</strong> due:' + dueDate + '<br>\n';
    var buttonHTML = '<button class="editNote" data-id="' + id + '">Edit</button>';
    var editContainerL1 = '<br>\n<div id="editContainer' + id + '" style="display:none">';
    editContainerL1 += '<label for="content' + id + '">Content<input id="content' + id + '" type="text" ';
    editContainerL1 += 'value="' + content + '" placeholder="your note text" name="content">';
    editContainerL1 += '</label>\n<label for="dueDate' + id + '">' + dueDate;
    editContainerL1 += '<input id="dueDate' + id + '" type="date" value="' + dueDate + '" ';
    editContainerL1 += 'placeholder="' + dueDate + '" name="' + dueDate + '"> </label>';
    editContainerL1 += '<br>\n<input type="submit" value="Absenden" class="sendEdit" ';
    editContainerL1 += 'data-id="' + id + '">' + '</div>\n</li>';
    return startHTML + buttonHTML + editContainerL1;
};

var createNotesHtml_M = function (notes) {
    var notesLength = notes.length;
    var notesHtml = '<ul>\n';
    for (var i = 0; i < notesLength; i++) {
        notesHtml += createNoteHtml_M(notes[i].id, notes[i].content, notes[i].dueDate);
    }
    notesHtml += '</ul>\n';
    return notesHtml;
}
```

Templating auf dem Client: Motivation – CreateHtmlFn mit einer Templating Library (z.B. Handlebars) (1)

HTML Seite mit komplexem Template und Einbinden der Handlebars Library

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="/stylesheets/style.css">
  <script src="/javascripts/handlebars-v4.0.5.js"></script>
  [...]
<script id="notes-template" type="text/x-handlebars-template">
  <div class="entries">
    <ul>
      {{#each this}}
        <li>
          <strong>{{content}}</strong> due:{{formatDate dueDate "short"}}
          <br><button class="editNote" data-id="{{id}}>Edit</button>
          <br>
          <div id="editContainer{{id}}" style="display:none">
            <label for="content{{id}}>Content<input id="content{{id}}"
                  type="text" value="{{content}} placeholder="your
            </label>
            <label for="dueDate{{id}}>Due Date<input id="dueDate{{id}}"
                  type="date" value="{{formatDate dueDate 'short'}}>
            </label>
            <br>
            <input type="submit" value="Absenden" class="sendEdit" data-id="{{id}}>
          </div>
        </li>
      {{/each}}
    </ul>
  </div>
```

Einfaches Beispiel vom Templating mit Handlebars (-> 5_Handlebars/JQuery-HandlebarsDemo.html)

Beispiel: Rendern eines Arrays von Kunden als “Unorderd List”

- Einbinden der Handlebars Library

```
<script language="javascript"  
src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.5/handlebars.min.js">
```

- Daten

```
var customers = {customers: [{firstName:'Peter', lastName: 'Pan', age:13},  
                           {firstName:'Captain', lastName:'Hook', age:35}]};
```

- Handlebars Script (im HTML)

```
<script id='customerTemplateText' type='text/x-handlebars-template'>  
  <h2>New Customers</h2>  
  <ul>  
    {{#each customers}}  
      <li>{{ firstName }} {{ lastName }} </li>  
    {{/each}}  
  </ul>  
</script>
```

- Laden und “compilieren” des Templates

(in vielen Tutorials leider "template" genannt, nicht create...html)

```
var customerTemplateText = $('#customerTemplateText').html();  
var createCustomersHtml = Handlebars.compile (customerTemplateText);
```

- Anwenden des Template (Nutzung von JQuery)

```
$(document.body).append (createCustomersHtml (customers));
```

Client Templating mit Handlebars – Details (1)

■ Templates können pre-compiled werden

(siehe <http://handlebarsjs.com/precompilation.html>)

oder das Template HTML wird in einem Script-Tag definiert z.B.

```
<script id="entry-template" type="text/x-handlebars-template">
  <div class="entry">
    <h1>{{name}}</h1>
    <div class="body">
      {{adress}}
    </div>
  </div>
</script>
```

■ Valide Handlebars Expressions

<h1>{{name}}</h1> (lookup im Context Objekt)

<h1>{{adress.zip}}</h1> (paths sind erlaubt: adress ist eine Eigenschaft des Context Objekts; mit ../ Navigation zum Parent)

■ Standard Block Helpers -> http://handlebarsjs.com/block_helpers.html

■ {{#each this}}

 Id: {{userId}} Name: {{name}}

 {{/each}}

■ {{#if isActive}}


 {{/if}}

Client Templating mit Handlebars – Details (2)

Custom Helper Funktionen müssen registriert werden:

```
Handlebars.registerHelper ("formatDate", function(datetime, format) {  
    if (moment) {  
        format = DateFormats[format] || format;  
        return moment(datetime).format(format);  
    }  
    else {  
        return datetime;  
    }  
});
```

Aufgabe: Music06Handlebars+InlineEventsExercise

```
<!DOCTYPE html>
<html>
<head lang="en">...
<body>
<h1>Songs</h1>
<ul id="songs"></ul>

<script src="http://cdnjs.cloudflare.com/ajax/libs/handlebars.js/1.0.0/handlebars.min.js"></script>
<script id="songs-template" type="text/x-handlebars-template">

    //TODO Add Content to Handlebars Template
    //(loop over songs, for each song li, h3 (with rating, buttons, title), p (with artist)

</script>
<script>
    const songs = [...];
    function compareSongs(s1, s2) {...}
    function findSong(id) {...}
    function rateSong (songId, delta) {...}

    /*
    var createSongsHtml = ... TODO compile Handlebars template
    */
    function renderSongs () {
        /*
        document.getElementById("songs").innerHTML= ... TODO use the createSongsHtml function;
        */
    }
    window.onload = renderSongs;
</script>
</body>
</html>
```

Aufgabe: Music07Handlebars+JQuery+BubblingEventsExercise

```
<!DOCTYPE html>
<html>
<head lang="en">...
<body>
<h1>Songs</h1>
<ul id="songs"></ul>

<script src="jquery-3.2.1.js"></script>
<script src="http://cdnjs.cloudflare.com/ajax/libs/handlebars.js/1.0.0/handlebars.min.js"></script>
<script id="songs-template" type="text/x-handlebars-template">
    //TODO Add Content to Handlebars Template
    // (loop over songs, for each song li, h3 (with rating, buttons – no event handler, title), p (with artist)
</script>
<script>
    const songs = [...];
    function compareSongs(s1, s2) {...}
    function findSong(id) {...}
    function rateSong(songId, delta) {...}
    function bubbledClickEventHandler(event) {...}

    // var createSongsHtml = ... TODO Compile Template

    function renderSongs () {
        // TODO use the compiled createSongsHtml function to add html to the <ul id="songs"> element
    }
    $(function() {
        renderSongs();
        $("#songs").on("click", bubbledClickEventHandler);
    });
</script>
</body>
</html>
```