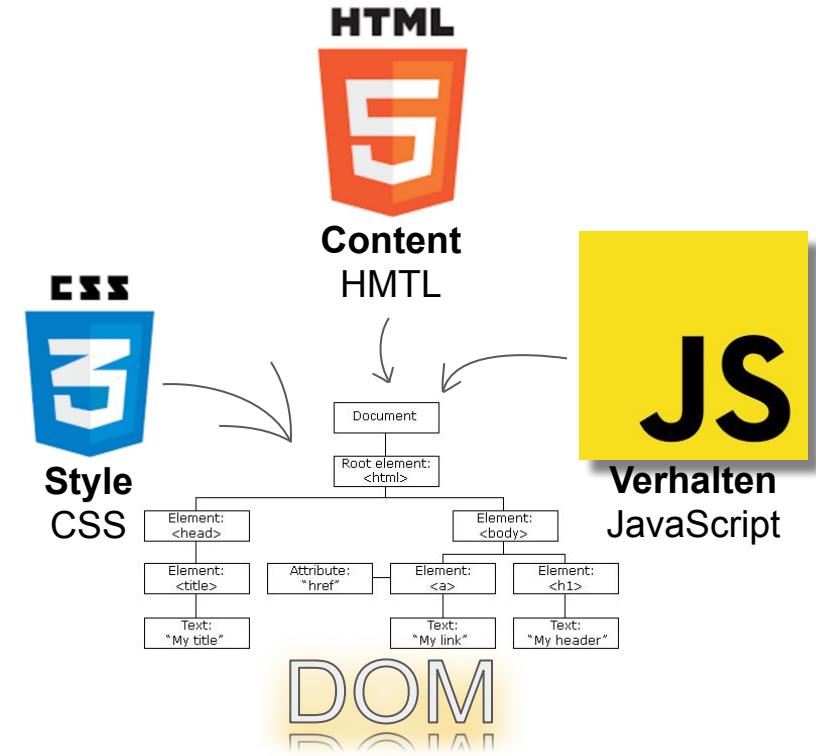


DOM-API

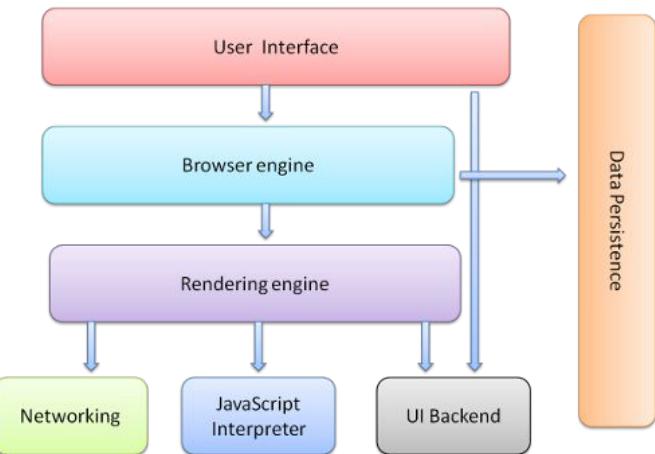
Das Zusammenspiel aller drei Technologien



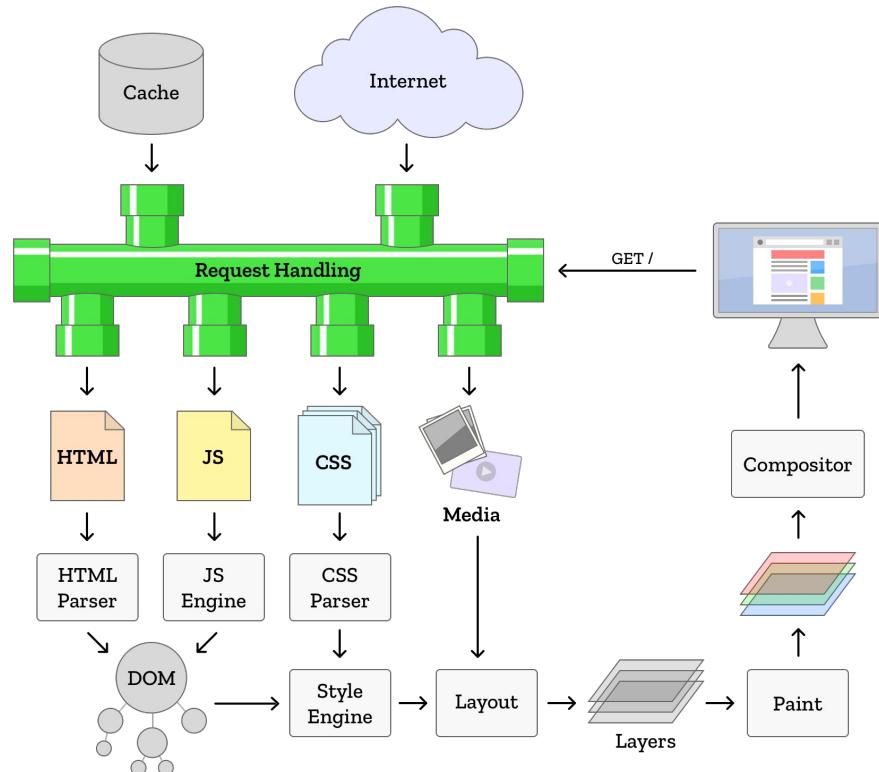
Grundbegriffe Web APIs



Gemeinsame Software-Architektur aller Browser

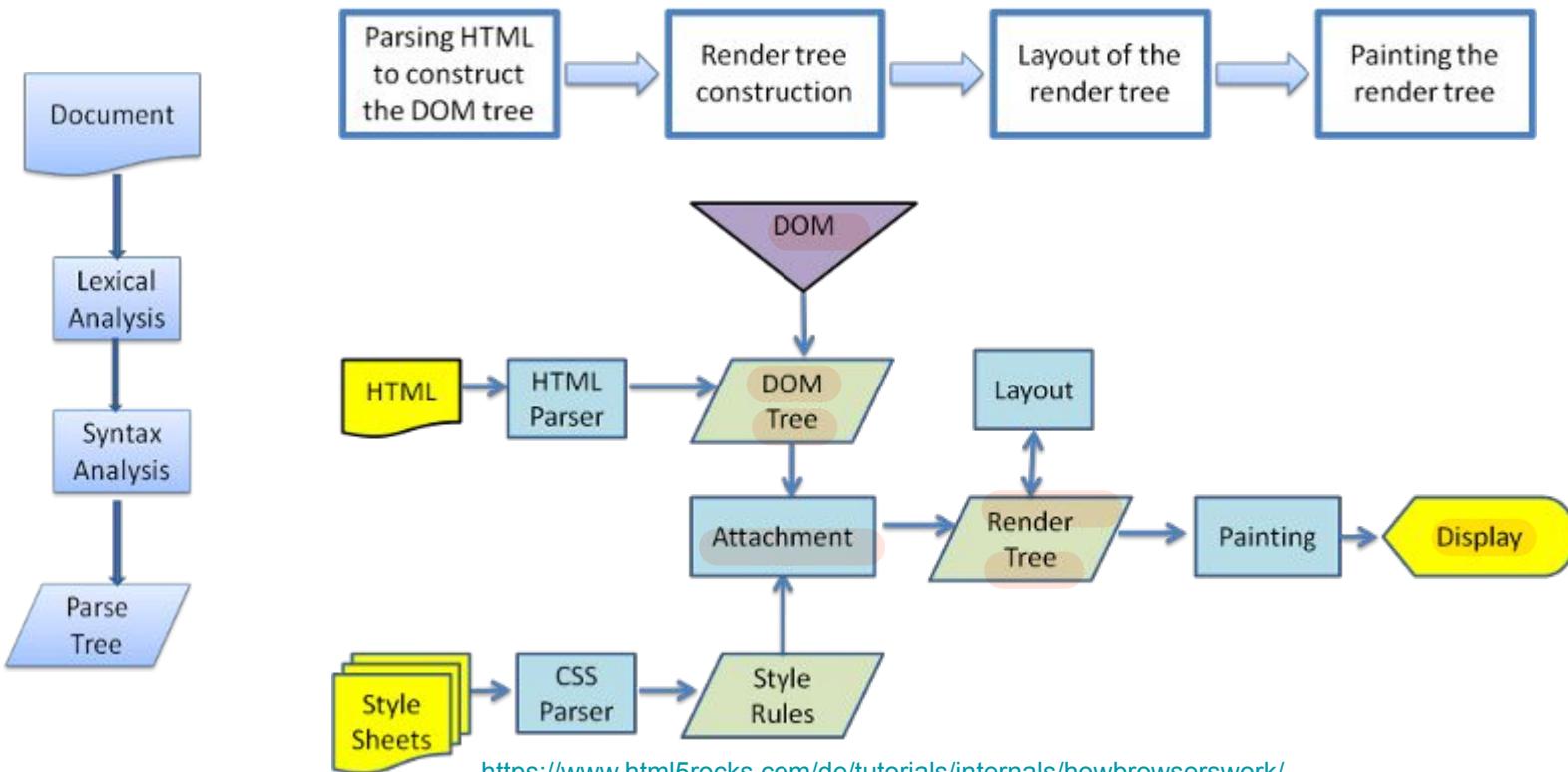


<https://www.html5rocks.com/de/tutorials/internals/howbrowserswork/>



<https://hacks.mozilla.org/2017/05/quantum-up-close-what-is-a-browser-engine/>

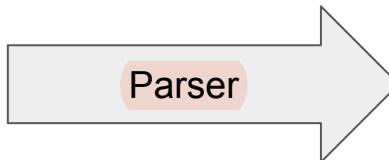
Grundlegender Ablauf des Renderings



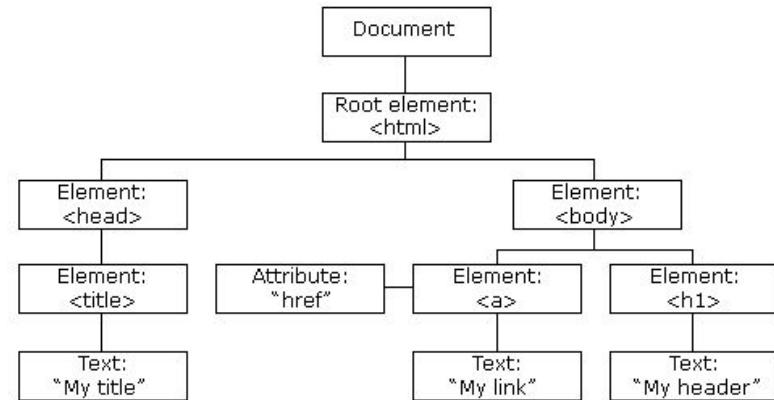
Compilerbau: Parser ⇒ AST

kontextfreie Sprache
(Chomsky-2)

```
<!DOCTYPE html>
<html lang="de">
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="link.html">My link</a>
</body>
</html>
```



Abstract Syntax Tree (AST)



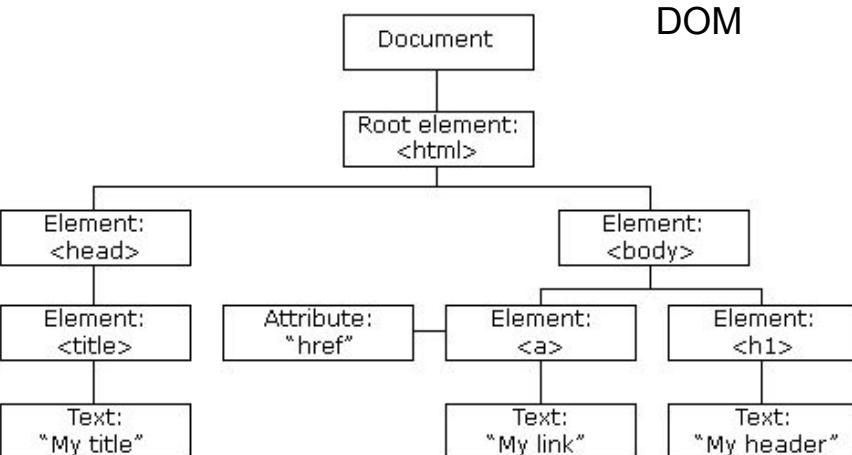
Besonderheit: Der AST ist **standardisiert** für HTML, XML, ... und heißt Document Object Model (DOM).

```

<!DOCTYPE html>
<html lang="de">
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="link.html">My link</a>
</body>
</html>

```

DOM und JavaScript



<script>

```

const header = document.querySelector('h1');
const link1 = document.querySelector('#link1');

header.innerText += "... added plain text from JS";
header.innerHTML += "... with <i>Markup</i>";

link1.href = "link2.html";
link1.style.color = 'red';

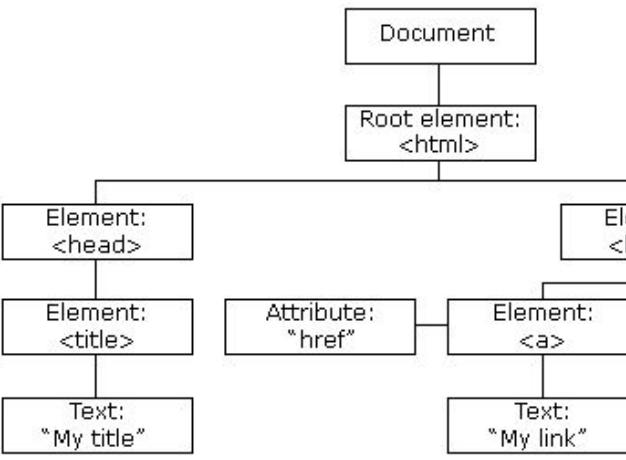
```

</script>

```

<!DOCTYPE html>
<html lang="de">
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="link.html">My link</a>
</body>
</html>

```



D **HTM**

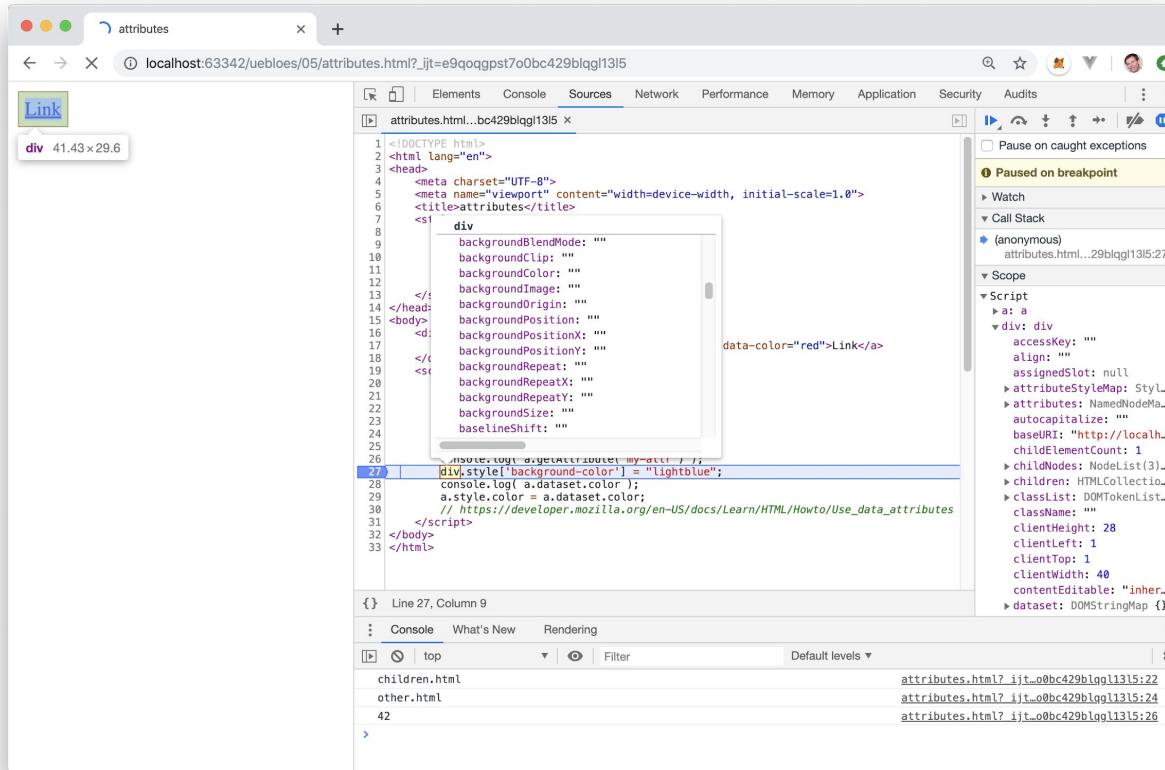
```

<script>
  const header = document.querySelector('h1');
  const link1 = document.querySelector('#link1');
header.
  hea m addEventListener(type: K, listener: (this:Eleme... void
  hea p classList Element (lib.dom.d.ts) DOMTokenList
  li p innerHTML Element (lib.dom.d.ts) string
  li m scrollIntoView(arg?: boolean | ScrollIntoViewOp... void
  m setAttribute(qualifiedName: string, value: stri... void
  p className Element (lib.dom.d.ts) string
  m getAttribute(qualifiedName: string) string | null
  m setAttributeNS(namespace: string | null, qualif... void
  m removeEventListener(type: K, listener: (this:El... void
  m getAttributeNames() string[]
  p outerHTML Element (lib.dom.d.ts) string
  m attachShadow(shadowRootInitDict: ShadowRo... ShadowRoot
  p assignedSlot Element (lib.dom... HTMLSlotElement | null
  p attributes Element (lib.dom.d.ts) NamedNodeMap
  p clientHeight Element (lib.dom.d.ts) number
  p clientLeft Element (lib.dom.d.ts) number
  p clientTop Element (lib.dom.d.ts) number
  p clientWidth Element (lib.dom.d.ts) number
  p closest(selector: K)

```

Press ^ to choose the selected (or first) suggestion and insert a dot afterwards >>

Exploring the Element Interface in DevTools



2 Welten: HTML versus JavaScript-Objekte

HTML Attributes

```
<div>
  <a href="children.html" data-color="red">Link</a>
</div>
```

getAttribute('name')

Object Properties

```
<script>
  const div = document.querySelector('div');
  const a = document.querySelector('a');
  console.log( a.getAttribute('href') );
  a.setAttribute( 'href', 'other.html' );
  console.log( a.getAttribute('href') );
  a.setAttribute( 'my-attr', 42 );
  console.log( a.getAttribute('my-attr') );
  div.style['background-color'] = "lightblue";
  console.log( a.dataset.color );
  a.style.color = a.dataset.color;
</script>
```

setAttribute('name')

a.my-attr // Syntaxfehler, da Minus
arithmetische Operation

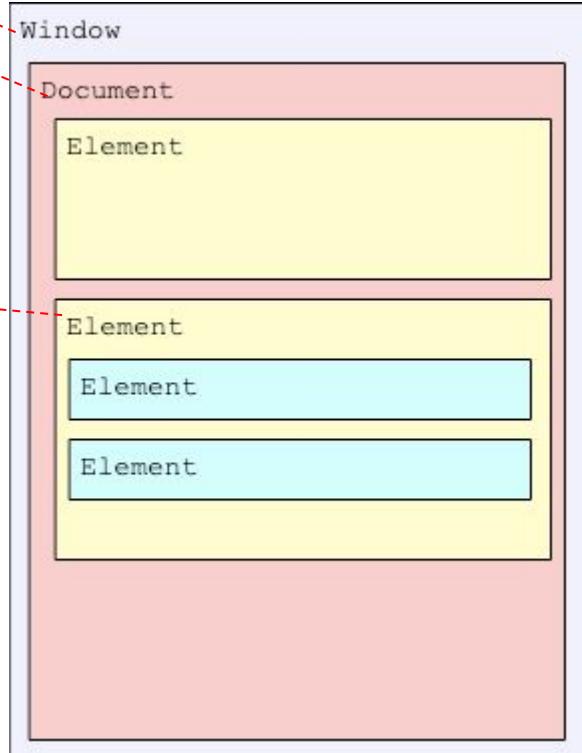
Console	Who
children.html	
other.html	
42	
red	

window ist der globale Namensraum document ist das Wurzelement des DOM

Namensraum pro Tabulator in Browser

Einstieg in DOM
über document

hierarchisch
geschachtelte
HTMLElemente



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

Object.keys(window)

```
["InstallTrigger", "alert", "applicationCache", "atob", "blur", "btoa", "caches",  
"cancelAnimationFrame", "captureEvents", "clearInterval", "clearTimeout",  
"close", "closed", "confirm", "content", "createImageBitmap", "crypto",  
"customElements", "devicePixelRatio", "document", "dump", "external",  
"fetch", "find", "focus", "frameElement", "frames", "fullScreen",  
"getComputedStyle", "getDefaultComputedStyle", "getSelection", "history",  
"indexedDB", "innerHeight", "innerWidth", "isSecureContext", "length",  
"localStorage", "location", "locationbar", "matchMedia", "menubar", "moveBy",  
"moveTo", "mozInnerScreenX", "mozInnerScreenY", "mozPaintCount", "name",  
"navigator", "onabort", "onabsoluteDeviceOrientation", "onafterprint",  
"onanimationend", "onanimationiteration", "onanimationstart", "onauxclick",  
"onbeforeprint", "onbeforeunload", "onblur", "oncanplay", "oncanplaythrough",  
"onchange", "onclick", "onclose", "oncontextmenu", "ondblclick", "ondeviceLight",  
"ondevicemotion", "ondeviceOrientation", "ondeviceProximity", "ondrag",  
"ondragend", "ondragenter", "ondragexit", "ondragLeave", "ondragover",  
"ondragstart", "ondrop", "ondurationchange", "onemptied", "onended", "onerror",  
"onfocus", "onhashchange", "oninput", "oninvalid", "onkeydown", "onkeypress",  
"onkeyup", "onlanguagechange", "onload", "onloadeddata", "onloadedmetadata",  
"onloadend", "onloadstart", "onmessage", "onmousedown", "onmouseEnter",  
"onmouseleave", "onmousemove", "onmouseout", "onmouseover", "onmouseup",  
"onmozfullscreenchange", "onmozfullscreenerror", "onoffline", "ononline",  
"onpagehide", "onpageshow", "onpause", "onplay", "onplaying", "onpopstate",  
"onprogress", "onratechange", "onreset", "onresize", "onscroll", "onseeked",  
"onseeking", "onselect", "onselectstart", "onshow", "onstalled", "onstorage",  
"onsubmit", "onsuspend", "ontimeupdate", "ontoggle", "ontransitioncancel",  
"ontransitionend", "ontransitionrun", "ontransitionstart", "onunload",  
"onuserproximity", "onvolumechange", "onwaiting", "onwebkitAnimationEnd",  
"onwebkitAnimationIteration", "onwebkitAnimationStart", "onwebkitTransitionEnd",  
"onwheel", "open", "opener", "outerHeight", "outerWidth", "pageXOffset",  
"pageYOffset", "parent", "performance", "personalBar", "postMessage", "print",  
"prompt", "releaseEvents", "requestAnimationFrame", "resizeBy", "resizeTo",  
"screen", "screenX", "screenY", "scroll", "scrollBy", "scrollByLines",  
"scrollByPages", "scrollMaxX", "scrollMaxY", "scrollTo", "scrollX", "scrollY",  
"scrollbars", "self", "sessionStorage", "setInterval", "setResizable",  
"setTimeout", "showModalDialog", "sidebar", "sizeToContent", "speechSynthesis",  
"status", "statusbar", "stop", "toolbar", "top", "updateCommands", "window"]
```

window is the top-level this.

document === window.document 10

Vorsicht: automatische Globalisierung !

var, let, const
vergessen

```
<script>
  function sum(max){
    for (i = 0; i<max; i++){
      x += i;
    }
    return x;
  }
  x = 1;
  sum(10);
  console.log( x ); // 46
</script>
```



- Nicht-deklarierte Variablen werden automatisch zu Properties des window-Objekts

document

Document

Properties

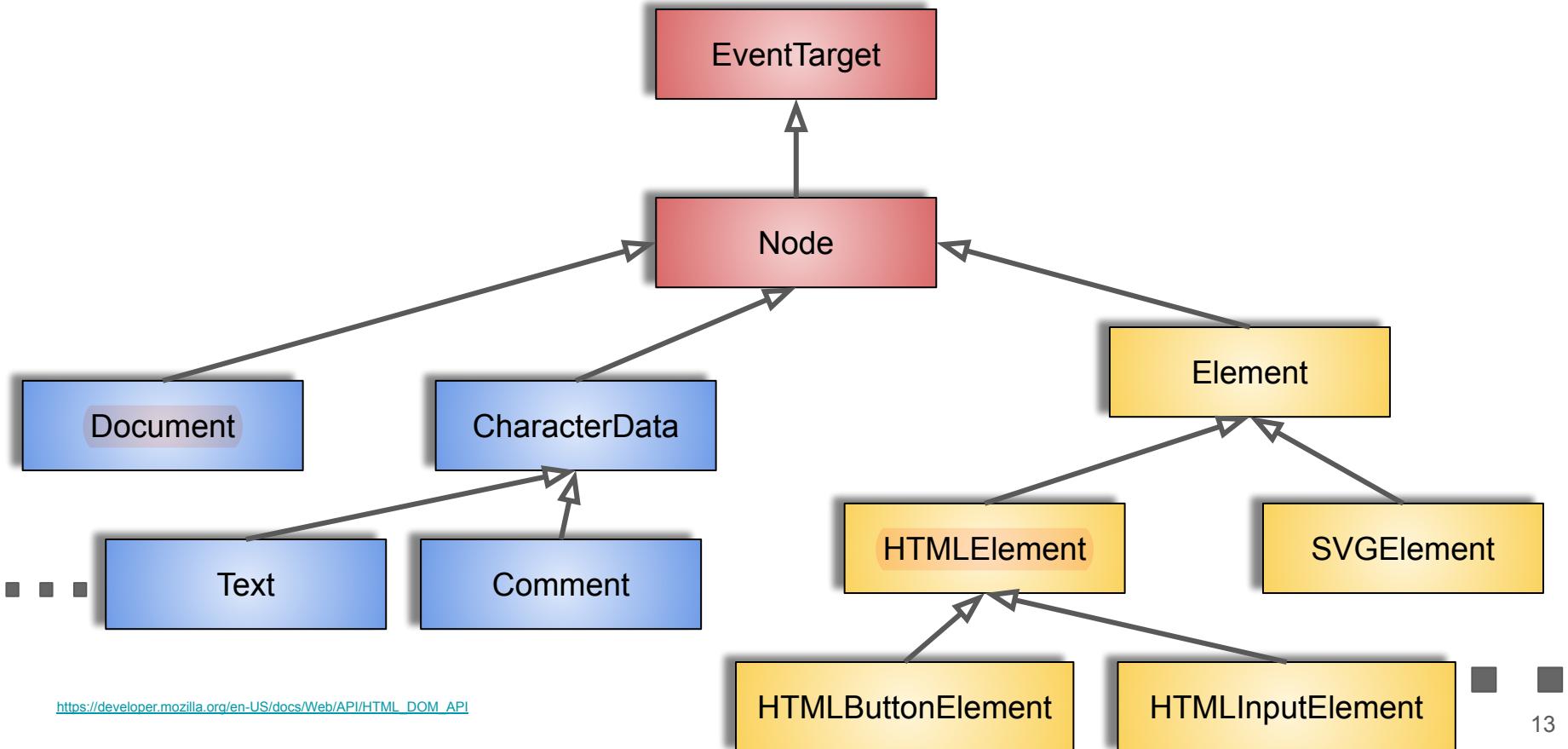
- head
- body
- characterSet
- documentURI
- ■ ■ ■

Methods

- createElement()
- createTextNode()
- getElementsByClassName()
- getElementsByTagName()
- getElementById(String id)
- querySelector()
- querySelectorAll()
- ■ ■ ■

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

DOM-Objekt-Interface-Vererbungshierarchie



HTML element interfaces

HTML element interfaces

These interfaces represent specific HTML elements (or sets of related elements which have the same properties and methods associated with them).

HTMLAnchorElement	HTMLHeadingElement	HTMLPictureElement
HTMLAreaElement	HTMLHtmlElement	HTMLPreElement
HTMLAudioElement	HTMLIFrameElement	HTMLProgressElement
HTMLBRElement	HTMLImageElement	HTMLQuoteElement
HTMLBaseElement	HTMLInputElement	HTMLScriptElement
HTMLBaseFontElement 	HTMLIndexElement 	HTMLSelectElement
HTMLBodyElement	HTMLLIElement	HTMLSlotElement
HTMLButtonElement	HTMLLabelElement	HTMLSourceElement
HTMLCanvasElement	HTMLLegendElement	HTMLSpanElement
HTMLDLListElement	HTMLLinkElement	HTMLStyleElement
HTMLDataElement	HTMLMapElement	HTMLTableCaptionElement
HTMLDataListElement	HTMLMarqueeElement 	HTMLTableCellElement
HTMLDetailsElement	HTMLMediaElement	HTMLTableColElement
HTMLDialogElement	HTMLMenuElement	HTMLTableElement
HTMLDirectoryElement	HTMLMenuItemElement 	HTMLTableRowElement
HTMLDivElement	HTMLMetaElement	HTMLTableSectionElement
HTMLElement	HTMLMeterElement	HTMLTemplateElement
HTMLEmbedElement	HTMLModElement	HTMLTextAreaElement
HTMLFieldSetElement	HTMLOLListElement	HTMLTimeElement
HTMLFontElement 	HTMLObjectElement	HTMLTitleElement
HTMLFormElement	HTMLOptGroupElement	HTMLTrackElement
HTMLFrameElement 	HTMLOptionElement	HTMLULListElement
HTMLFrameSetElement 	HTMLOutputElement	HTMLUnknownElement
HTMLHRElement	HTMLParagraphElement	HTMLVideoElement
HTMLHeadElement	HTMLParamElement	

interface Node

Node

Node properties:

- childNodes
- firstChild
- lastChild
- nextSibling
- nodeName
- nodeType
- nodeValue
- parentNode
- previousSibling

Node methods:

- appendChild()
- cloneNode()
- contains()
- hasChildNodes()
- insertBefore()
- isEqualNode()
- removeChild()
- replaceChild()



interface HTMLElement

HTMLElement

HTMLElement properties:

- innerHTML
- outerHTML
- textContent
- innerText
- outerText
- firstElementChild
- lastElementChild
- nextElementChild
- previousElementChild
- children

HTMLElement methods:

- blur()
- click()
- focus()



Unterschied zwischen Nodes und Elements

```
<body>
  <h1>Überschrift</h1>

  Leerzeilen

  <p>
    Paragraph

  </p>
  <script>
  </script>
</body>
```

```
▼ childNodes: NodeList(7) [ #text "", h1, #text "", ... ]
  ▶ 0: #text ""
  ▶ 1: <h1>
  ▶ 2: #text ""

  Leerzeilen

  ▶ 3: <p>
  ▶ 4: #text ""
  ▶ 5: <script>
  ▶ 6: #text ""

  ▶ length: 7
```

Node

```
▼ children: HTMLCollection(3) [ h1, p, script ]
  ▶ 0: <h1>
  ▶ 1: <p>
  ▶ 2: <script>

  ▶ length: 3
```

HTMLElement

DOM-Inspektor

versus Element-Inspektor

The screenshot shows the "Live DOM Viewer" interface. At the top, there's a toolbar with icons for back, forward, search, and refresh. Below it, the title bar says "Live DOM Viewer". Underneath, there's a section for "Markup to test (permalink, save, upload, download, hide)". It contains the following HTML code:

```
<!DOCTYPE html>
<h1>Überschrift</h1>
Leerzeilen
<p>
Paragraph
</p>
```

Below this is a "DOM view (hide, refresh)" section showing the DOM tree:

```
DOMTYPE: html
└─ HTML
   └─ HEAD
      └─ BODY
         └─ H1
            └─ #text: Überschrift
         └─ P
            └─ #text: Leerzeilen
            └─ #text: Paragraph
```

At the bottom, there's a "Rendered view (hide)" section showing the rendered HTML:

Überschrift

Leerzeilen

Paragraph

A yellow callout box points from the text "inkl. Text Nodes" to the "#text" entries in the DOM tree.

The screenshot shows the "children" tab in the Chrome DevTools Elements panel. The title bar says "Überschrift". The DOM tree shows:

```
<!DOCTYPE html>
...html lang="en"> == $0
  ><head>...</head>
  ><body>
    ><h1>Überschrift</h1>
    " "
    Leerzeilen
    " "
    <p>
      Paragraph
      </p>
    ><script>...</script>
  </body>
</html>
```

Below the tree is a "Styles" panel with CSS rules:

```
element.style {
}
html[Attributes Style] {
  -webkit-locale: "en";
}
html {
  display: block;
  color: -internal-root-color;
}
```

On the right, there's a visual representation of the element hierarchy with bounding boxes and styles applied. A yellow callout box points from the text "exkl. Text Nodes" to the absence of text node entries in the DOM tree.

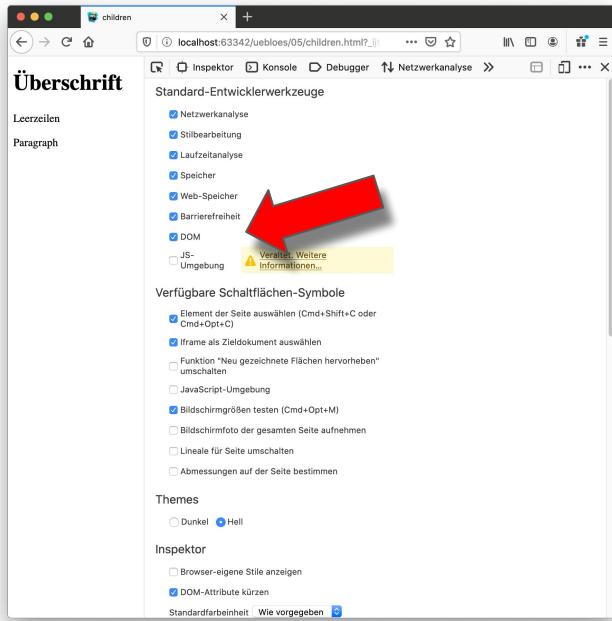
Chrome Elements

- nur Elemente
- keine Text Nodes

<https://software.hixie.ch/utilities/js/live-dom-viewer/>

Firefox DOM-Inspektor

Einstellungen:



The screenshot shows the Firefox DOM Inspector interface with the URL 'localhost:63342/uebloes/05/children.html?_j' in the address bar. The main view displays the DOM structure of an 'Überschrift' (Section header) element. The DOM tree is as follows:

- body:
 - aLink: ...
 - accessKey: ...
 - accessKeyLabel: ...
 - assignedSlot: null
 - attributes: NamedNodeMap []
 - background: ...
 - baseURI: "http://localhost:63342/...5v18sukfhg1kc3s6hcd9rrb"
 - bgColor: 3
 - childElementCount: 3
 - childNodes:
 - 0: #text "
 - 1: <h1>
 - 2: #text "
 - 3: Leerzeilen
 - 4: #text "
 - 5: <script>
 - 6: #text "
 - length: 7
 - children:
 - 0: <h1>
 - 1: <p>
 - 2: <script>
 - length: 3
 - classList: DOMTokenList []

Beispiel HTMLInputElement

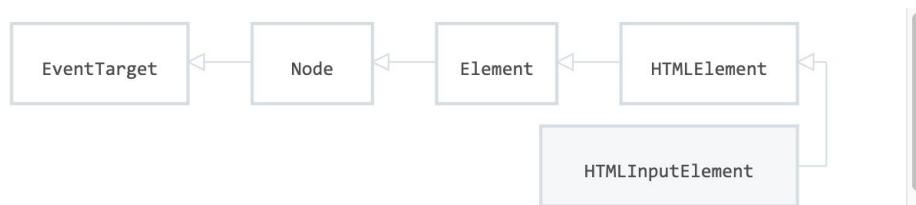
HTMLInputElement

Web technology for developers › Web APIs › HTMLInputElement

On this Page

- [Properties](#)
- [Methods](#)
- [Events](#)
- [Specifications](#)
- [Browser compatibility](#)
- [See also](#)

The `HTMLInputElement` interface provides special properties and methods for manipulating the options, layout, and presentation of `<input>` elements.



Referenz

HTMLElement

Web technology for developers > Web APIs > HTMLElement

Jump to: Properties Methods Events Specifications Browser compatibility See also

The `HTMLElement` interface represents any `HTML` element. Some elements directly implement this interface, while others implement it via an interface that inherits it.



Properties

Inherits properties from its parent, `Element`, and implements those from `GlobalEventHandlers` and `TouchEventHandlers`.

`HTMLElement.accessKey`

Is a `DOMString` representing the access key assigned to the element.

`HTMLElement.accessKeyLabel` Read only

Returns a `DOMString` containing the element's assigned access key.

`HTMLElement.contentEditable`

Is a `DOMString`, where a value of "true" means the element is editable and a value of "false" means it isn't.

`HTMLElement.isContentEditable` Read only

Returns a `Boolean` that indicates whether or not the content of the element can be edited.

Methods

Inherits methods from its parent, `Element`.

`HTMLElement.attachInternals()` △

Attaches an `ElementInternals` instance to the custom element.

`HTMLElement.blur()`

Removes keyboard focus from the currently focused element.

`HTMLElement.click()`

Sends a mouse click event to the element.

`HTMLElement.focus()`

Makes the element the current keyboard focus.

`HTMLElement.forceSpellCheck()` △

Runs the spell checker on the element's contents.

Demo Web-Entwicklung

Einkaufsliste

Enter a new item: Add item

- Kaffee Delete
- Äpfel Delete
- Bananen Delete

Introduction to DOM-API

w3schools.com



HTML

CSS

JAVASCRIPT

https://www.w3schools.com/js/js_htmldom.asp

JS HTML DOM

DOM Intro

DOM Methods

DOM Document

DOM Elements

DOM HTML

DOM CSS

DOM Animations

DOM Events

DOM Event Listener

DOM Navigation

DOM Nodes

DOM Collections

DOM Node Lists

DOM-API: Erzeugungsmethoden

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>||Working with elements||</title>
</head>
<body>
  <div id="div1">The text above has been created dynamically.</div>
</body>
</html>
```

JavaScript

```
document.body.onload = addElement;

function addElement () {
  // create a new div element
  // and give it some content
  var newDiv = document.createElement("div");
  var newContent = document.createTextNode("Hi there and greetings!");
  newDiv.appendChild(newContent); //add the text node to the newly created div.

  // add the newly created element and its content into the DOM
  var currentDiv = document.getElementById("div1");
  document.body.insertBefore(newDiv, currentDiv);
}
```

d = document

d.createElement	creates the HTML element
d.createTextNode	Creates a new Text node.
<pre>var last_p = d.createElement('p'); d.body.appendChild(last_p);</pre>	

d = document
e = element

DOM-API: Selektionsmethoden

d.getElementById	Returns a reference to the element by its ID
d.getElementsByClassName	Returns an array-like object of all child elements which have all of the given class names.
d.getElementsByTagName	Returns an HTMLCollection of elements with the given tag name.
d.getElementsByName	Returns a nodelist collection with a given name.
e.querySelector	Returns the first Element that matches the specified group of CSS selectors
e.querySelectorAll	Returns a list of the elements that matches the specified group of CSS selectors

Note: The matching is done using **depth-first** pre-order traversal of the document's nodes starting with the first element in the document's markup and iterating through sequential nodes by order of the number of child nodes.

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

Damit kann man auch **Unterbäume** durchsuchen!

Problem: HTMLCollection ist kein JS-Array

Lösung ES6-Spread-Operator ... zur Iteration über HTMLCollection

```
2 |<html lang="en">
3 |<head>
4 |  <meta charset="UTF-8">
5 |  <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 |  <title>Spread</title>
7 |</head>
8 |<body>
9 |  <h1>Spread Operator</h1>
10 |<ul>
11 |  <li>1</li>
12 |  <li>2</li>
13 |</ul>
14 |<script>
15 |  document.getElementsByTagName('li').forEach( li => li.innerHTML += ". Fall" );
16 |</script>
17 |> HTMLCollection(2)
18 |  ▶ 0: li
    |  ▶ 1: li
    |  length: 2
    |  __proto__: HTMLCollection
```

Spread Operator

- 1. Fall
- 2. Fall

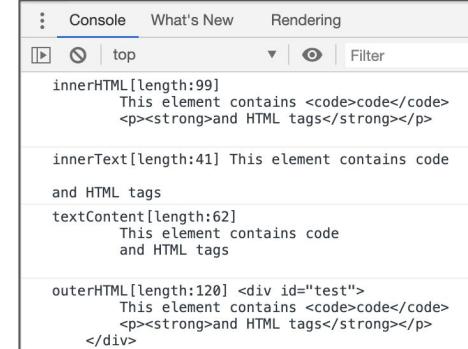
✖ ▶ Uncaught TypeError:
document.getElementsByTagName(...).forEach is not a function
at Spread.html? ijt=1p9...omsgh56bp18i07f4:15

```
<!doctype html>
<h1>Spread Operator</h1>
<ul>
  <li>1</li>
  <li>2</li>
</ul>
<script>
  [...document.getElementsByTagName('li')].map(li=>li.innerHTML += ". Fall");
</script>
```

Lesen mit innerHTML, innerText, textContent und outerHTML

- **innerHTML**
mit inneren HTML-Tags
- **innerText**
ohne Tags, style-aware
- **textContent**
ohne Tags, not aware of style
- **outerHTML**
mit äußereren Tags

```
<div id="test">  
  This element contains <code>code</code>  
  <p><strong>and HTML tags</strong></p>  
</div>  
<script>  
  const x = document.getElementById('test');  
  ['innerHTML', 'innerText', 'textContent', 'outerHTML'].forEach(  
    prop => {  
      console.log(`#${prop}[length:${x[prop].length}] ${x[prop]}`);  
    }  
  );  
</script>
```



Schreiben mit innerHTML, innerText, textContent und outerHTML

- **innerHTML**
ruft HTML-Parser, Layouter,
Painter auf
- **innerText**
nur Text, **Reflow**, style aware
- **textContent**
nur Text, not aware of style
- **outerHTML**
überschreibt auch äußere Tags,
ruft HTML-Parser, Layouter,
Painter auf

```
<div id="test">  
  This element contains <code>code</code>  
  <p><strong>and HTML tags</strong></p>  
</div>
```

```
<script>  
const testDiv = document.getElementById('test');  
const content = `This element contains <code>code</code>  
  <p><strong>and HTML tags</strong></p>`;  
testDiv.innerHTML = content;  
// testDiv.innerText = content;  
// testDiv.textContent = content;  
// testDiv.outerHTML = content;  
</script>
```

testDiv.innerHTML = content;

This element contains code
and HTML tags

```
▼<body>
  ▼<div id="test">
    "This element contains "
    <code>code</code>
    ▼<p>
      <strong>and HTML tags</strong>
    </p>
  </div>
```

testDiv.outerHTML = content;

This element contains code
and HTML tags

```
▼<body>
  "
    This element contains "
    <code>code</code>
    ▼<p>
      <strong>and HTML tags</strong>
    </p>
```

`testDiv.textContent = content;`

This element contains

`<code>code</code> <p>and
HTML tags</p>`

```
▼<body>
  ▼<div id="test">
    "This element contains <code>code</code>
      <p><strong>and HTML tags</strong></p>"
  </div>
```

`testDiv.innerText = content;`

This element contains `<code>code</code>`
`<p>and HTML tags</p>`

```
▼<body>
  ▼<div id="test">
    "This element contains <code>code</code>
      <br>
      "
      <p><strong>and HTML tags</strong></p>"
  </div>
```

ohne `
`

<https://stackoverflow.com/questions/19030742/difference-between-innertext-and-innerhtml>

Differences `textContent` versus `innerText`

Don't get confused by the differences between `Node.textContent` and `HTMLElement.innerText`.

Although the names seem similar, there are important differences:

- `textContent` gets the content of all elements, including `<script>` and `<style>` elements. In contrast, `innerText` only shows “human-readable” elements.
- `textContent` returns every element in the node. In contrast, `innerText` is aware of styling and won’t return the text of “hidden” elements. Moreover, since `innerText` takes CSS styles into account, reading the value of `innerText` triggers a `reflow` to ensure up-to-date computed styles. (Reflows can be computationally expensive, and thus should be avoided when possible.)
- Unlike `textContent`, altering `innerText` in Internet Explorer (version 11 and below) removes child nodes from the element and permanently destroys all descendant text nodes. It is impossible to insert the nodes again into any other element or into the same element anymore.

4 Wege, JavaScript in HTML einzubetten

4 Wege der Einbettung:

```
<!doctype html>
<html>
<head>
  <title>My title</title>
  <script> alert('Hallo Welt'); //... JavaScript direkt
    document.write('<script src="http://URL.de"></script>');
  </script>
  <script src="/Pfad/zur/externen_Datei/javascript.js"></script>
</head>
<body>
  <h1>My header</h1>
  <div onclick="alert('Hallo Welt')"> ... </div>
```

1. JavaScript direkt hinein schreiben

2. JavaScript-Programm ausführen, um ein
anderes JavaScript-Programm aus einer
externen Datei zu laden

3. JavaScript aus einer externen Datei laden

4. Inline JavaScript

JavaScript disabled:

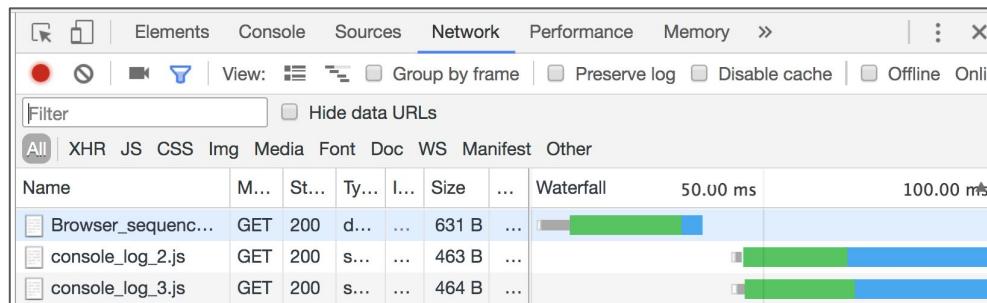
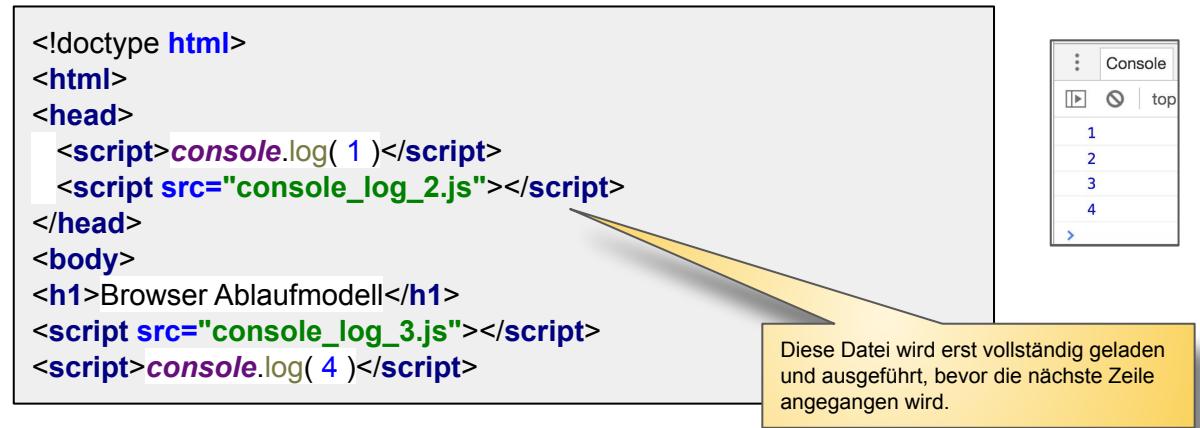
```
<noscript>
```

Sorry but you need to have scripting
enabled to use this site.

```
</noscript>
```

Reihenfolge: Ablaufmodell, wie der Browser eine Webseite abarbeitet

Zeile für Zeile
sequentiell
synchron



console_log_2.js

console.log(2);

<script>-Tag

```
<!doctype html>
<meta charset="utf-8">
<script>
  const myDiv = document.getElementById("myDiv");
  myDiv.innerHTML = "JavaScript wurde jetzt ausgeführt!"
</script>
<div id="myDiv">JavaScript wurde noch nicht ausgeführt!</div>
```

JavaScript wurde noch nicht ausgeführt!

✖ Uncaught TypeError: Cannot set property 'innerHTML' of null

Reihenfolge wichtig: DOM-Elemente müssen vorher existieren:

```
<!doctype html>
<meta charset="utf-8">
<div id="myDiv">JavaScript wurde noch nicht ausgeführt!</div>
<script>
  const myDiv = document.getElementById("myDiv");
  myDiv.innerHTML = "JavaScript wurde jetzt ausgeführt!"
</script>
```

JavaScript wurde jetzt ausgeführt!

<script>-Tag für Event-Handler

```
<!doctype html>
<meta charset="utf-8">
<div onclick="changeText(this)">JavaScript wurde noch nicht ausgeführt!</div>
<script>
function changeText( element ){
    element.innerHTML = "JavaScript wurde jetzt ausgeführt!";
}
</script>
```

JavaScript wurde noch nicht ausgeführt!

JavaScript wurde jetzt ausgeführt!



Unobstrusive JavaScript

```
<!DOCTYPE html>
<h1 onclick="this.innerText += 'Ooops!';">Click on this text!</h1>
```

```
<!DOCTYPE html>
<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText( elem ) {
  elem.innerText += "Ooops!";
}
</script>
```

*Saubere Trennung von
HTML und JavaScript*

Unobstrusive JavaScript



```
<!DOCTYPE html>
<h1>Click on this text!</h1>

<script>

const h1 = document.querySelector("h1");
h1.addEventListener("click", function( event ){
  this.innerText += "Ooops!";
});

</script>
```

Timer-API

1. `setTimeout(callback [, delay]);`
2. `repeater = setInterval(callback, delay);`
3. `clearInterval(repeater);`

```
<h1></h1>
<script>
  const h1 = document.querySelector('h1');
  const repeater = setInterval( () => h1.innerText = new Date().toLocaleTimeString('de-DE'), 1000);
  setTimeout( () => clearInterval( repeater ), 10_000 );
</script>
```

[https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope setTimeout](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope	setTimeout)
<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval>
<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/clearInterval>

Add or Change CSS rules at runtime

```
<!doctype html>
<body>
<button>Es werde farbig!</button>
<script>
const button = document.querySelector('button');
button.onclick =
    event => document.body.style.backgroundColor = getRandomColor();

function getRandomColor() {
    var letters = '0123456789ABCDEF';
    var color = '#';
    for (var i = 0; i < 6; i++) {
        color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
}
</script>
</body>
```

Es werde farbig!

Form values ⇒ **value**-Property in HTMLElement

HTML:

```
<input type="text" name="name" id="uniqueID" value="value" />
```

JS:

```
var nameValue = document.getElementById("uniqueID").value;
```



<https://stackoverflow.com/questions/3547035/javascript-getting-html-form-values>

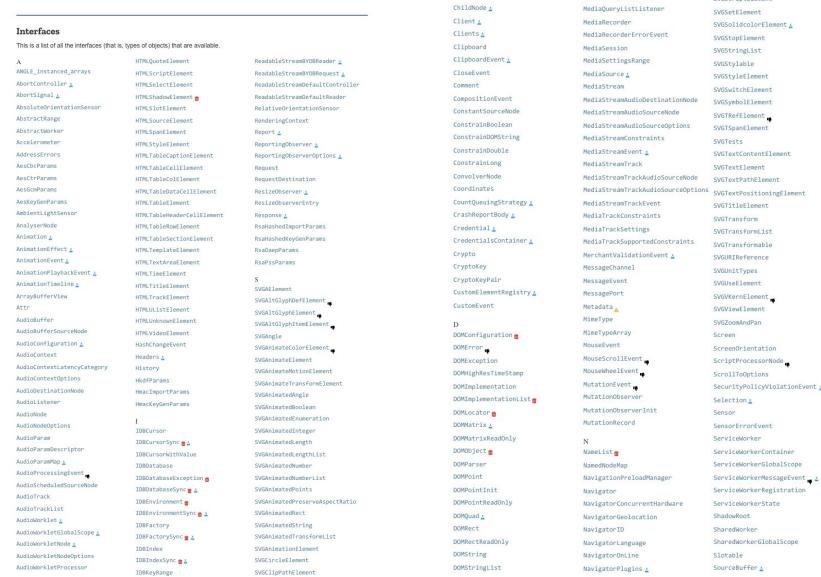
Web APIs auf dem Stand von 2019

- API (HTML or XML page) = DOM + JS (scripting language)
- 66 Specifications
- 922 Interfaces

Specifications

This is a list of all the APIs that are available.

A	B	C	D	E	F	G
Ambient Light Events	Background Tasks	HTML Drag and Drop API	DOM	Navigation Timing	Fetch API	Gamepad API
Battery API	Beacon	High Resolution Time	Background	Network Information API	Performance API	File System API
Bluetooth API	IndexedDB	History API	Canvas API	Web Notifications	Performance Timeline API	Frame Timing API
Broadcast Channel API	Intersection Observer API	Image Capture API	Channel Messaging API	Web Storage API	Permissions API	Pointer Events
CSS Counter Styles	Media Capabilities API	Media Capture and Streams	Media Session API	Web Workers API	Pointer Lock API	Proximity Events
CSS Font Loading API	M	Vibration API	Media Source Extensions	Web Audio API	Push API	Push API
CSSTM	T	Touch Events	MEDIASTREAM RECORDING	Web Animations		
Canvas API	I	WebVibration API	MediaSource	Web Animation API		
Channel Messaging API	S	Vibration API	AUDIO	Web Worker API		
Console API	U	Web Worker API	AUDIO	Web Authentication API		
Credential Management API	V	Web Workers API	AUDIOVIDEO	Web Crypto API		
DOM	W	Web Workers API	AUDIOVIDEO	WebRTC		
DOM	X	Web Workers API	AUDIOVIDEO	WebXR Device API		
DOM	Y	Web Workers API	AUDIOVIDEO	WebSockets API		
DOM	Z	Web Workers API	AUDIOVIDEO			



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

HTML DOM API interfaces ⊂ Web APIs

1. HTML Element interfaces
2. Web app and browser integration interfaces
3. Form support interfaces
4. Canvas and image interfaces
5. Media interfaces
6. Drag and drop interfaces
7. Page history interfaces
8. Web Components interfaces
9. Miscellaneous and supporting interfaces
10. Interfaces belonging to other APIs
 - a. Web storage interfaces
 - b. Web Workers interfaces
 - c. WebSocket interfaces
 - d. Server-sent events interfaces