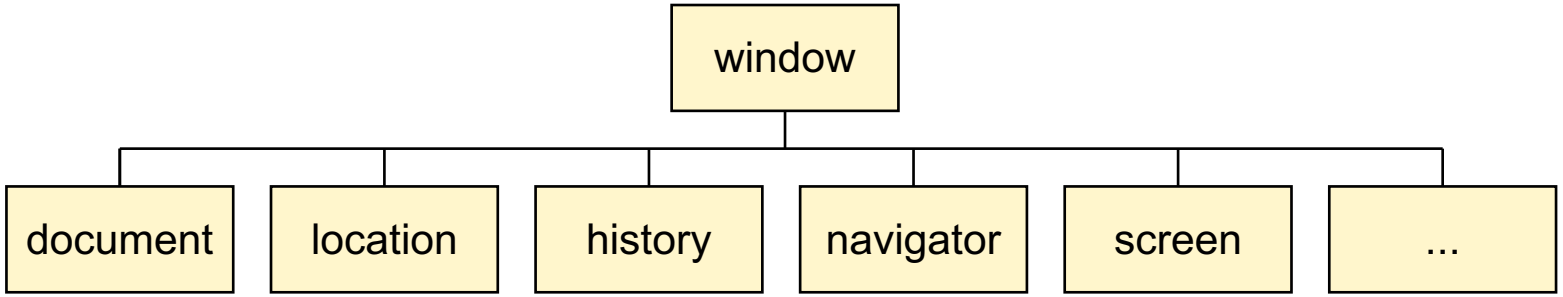
Inhaltsübersicht

Überblick

- Im Browser ist `window` das globale Objekt des aktuellen Fensters / Reiters
- Alle Standardobjekte stehen als Eigenschaften des `window`-Objekts zur Verfügung



Window-Objekt

- Neues Fenster öffnen
  - Mit Standardwerten (wird dann i.d.R. in einem Reiter angezeigt)

```
let newWindow = window.open('seite.html');
```

  - Mit expliziten Angaben (Auswahl)

```
window.open('seite.html', '_blank', 'width=640, height=400');
```

- Aktuelles Fenster schließen

```
window.close();
```

- Zuvor geöffnetes Fenster schließen

```
newWindow.close();
```

Window-Objekt

- Meldung im Dialog anzeigen

```
alert('Hello World!');
```

- Bestätigungsdialog anzeigen

```
let confirmed = confirm('Änderungen verwerfen?');
```

  - Rückgabewert ist `true`, falls der OK-Button gedrückt wurde, sonst `false`

- Eingabedialog anzeigen

```
let name = prompt('Bitte Namen eingeben', 'Max Mustermann');
```

  - Bei Abbruch des Dialogs ist der Rückgabewert `null`

Window-Objekt

- Eine Funktion einmalig zeitverzögert ausführen

```
let timerId = setTimeout(function(param) { console.log(param); },
                         1000, 'Argument für die Funktion');
```

  - Funktion wird nach der angegebenen Zeit (in Millisekunden) aufgerufen
  - Dieser Funktion können beliebig viele Argumente mitgegeben werden

- Eine Funktion in bestimmten Zeitintervallen aufrufen

```
let timerId = setInterval(function(param) { console.log(param); },
                          1000, 'Argument für die Funktion');
```

  - Parameter wie bei `setTimeout`

- Timer abbrechen

```
clearTimeout(timerId);
```

```
clearInterval(timerId);
```

Location-Objekt

- Über die `location`-Eigenschaft eines `Window`-Objekts erreichbar
- Bietet Informationen zur URL des angezeigten Dokuments (Auswahl)

Eigenschaft	Erläuterung	Beispiel
`href`	Vollständige URL	`"http://example.com/path/to/resouce?lang=de#chapter1"`
`protocol`	Protokoll	`"http:"`
`hostname`	Hostname	`"example.com"`
`port`	Port (Leerstring beim Default-Port)	`""`
`origin`	Kombination aus `protocol`, `hostname` und `port`	`"http://example.com"`
`pathname`	Pfad	`"/path/to/resouce"`
`search`	Anfragestring	`"?lang=de"`
`hash`	Position im Dokument	`"#chapter1"`

Location-Objekt

- Aktuelles Dokument erneut laden

```
window.location.reload();
```

- Zur angegebenen URL navigieren

```
window.location.assign('http://example.com/other/path');
```

  - Wenn die neue URL eine andere Origin als die aktuelle URL hat, dann wird ein `SecurityError` geworfen

- Zur angegebenen URL navigieren (ersetzt aktuellen History-Eintrag)

```
window.location.replace('http://www.google.com');
```

  - Kein `SecurityError`, wenn die Origin sich ändert

Navigator-Objekt

- Über die `navigator`-Eigenschaft eines `Window`-Objekts erreichbar
- Bietet Informationen zum Browser (Auswahl)

Eigenschaft	Erläuterung	Beispiel
`platform`	Name der Pattform	`"MacIntel"`
`userAgent`	Inhalt des User-Agent HTTP-Headers	`"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/601.5.17 (KHTML, like Gecko) Version/9.1 Safari/601.5.17"`
`language`	Vom Benutzer präferierte Sprache	`"de-de"`
`onLine`	Online-Status des Browsers	`false`
`cookieEnabled`	`true`, falls Cookies gesetzt werden können	`true`

History-Objekt

- Über die `history`-Eigenschaft eines `Window`-Objekts erreichbar
- Ermöglicht Zugriff auf die Navigationshistorie (Auswahl)
  - Anzahl Einträge

```
window.history.length;
```

  - Zur vorherigen / nächsten URL navigieren

```
window.history.back();
```

```
window.history.forward();
```

  - Um *n* Schritte nach vorwärts (positiv) bzw. zurück (negativ) zu navigieren

```
window.history.go(2); // 2 Schritte vorwärts navigieren
```

    – Bei 0 wird die aktuelle Seite neu geladen

Screen-Objekt

- Über die `screen`-Eigenschaft eines `Window`-Objekts erreichbar
- Bietet Informationen zum Bildschirm (Auswahl)

Eigenschaft	Erläuterung	Beispiel
`width`	Gesamtbreite	`1920`
`height`	Gesamthöhe	`1200`
`availableWidth`	Verfügbare Breite	`1200`
`availableHeight`	Verfügbare Höhe	`1109`

  - Alle Angaben in CSS Pixeln
    (bei hochauflösenden Displays ungleich den physischen Pixeln)

Mikroübung

Richtig oder falsch?

- Ein Browser-Standardobjekt kann sowohl direkt als auch über das globale Objekt angesprochen werden (z. B. `screen` und `window.screen`)
- Es ist möglich, programmatisch zur vorherigen Seite zu navigieren
- Eine Seite kann sich nicht selbständig (ohne Benutzerinteraktion) erneut laden

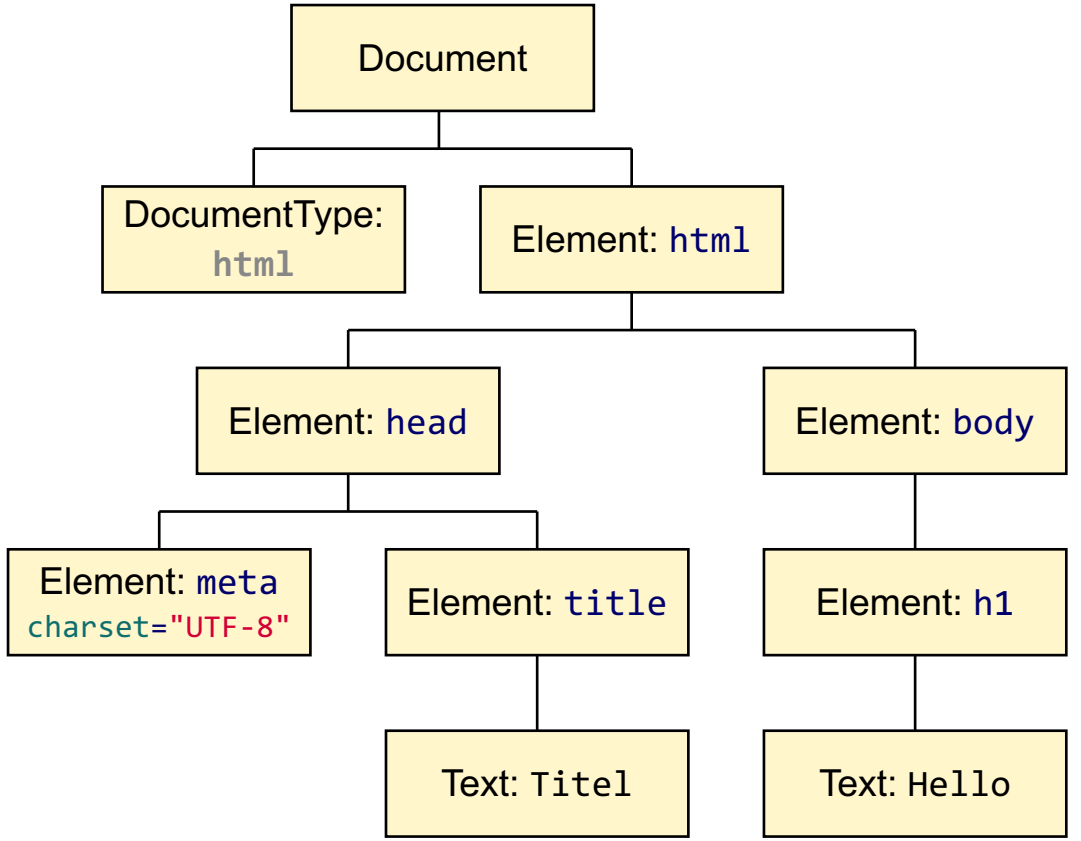
Inhaltsübersicht

Document Object Model

- DOM
  - Das Document Object Model (DOM) ist ein plattformneutrales Model für Ereignisse (*events*) und **Knotenbäume** (*node trees*) zur Repräsentation von und Interaktion mit HTML-/XML-Dokumenten im Hauptspeicher
- DOM-APIs
  - Ermöglichen den programmatischen Zugriff/Manipulation eines DOMs
  - Sind im Standard plattform- und sprachunabhängig definiert
    - DOM4 verwendet hierzu die WebIDL (Web Interface Definition Language)
- Spezifikationen
  - http://www.w3.org/DOM/DOMTR

Knotenbaum (DOM-Baum)


```
<!doctype html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Titel</title>
</head>

<body>
  <h1>Hello</h1>
</body>

</html>
```



Vereinfachte Darstellung (Text-Knoten zwischen den Elementen fehlen hier)

Knotenbaum (DOM-Baum)

Knoten-Typen und Interfaces

- Ein Knotenbaum besteht aus typisierten Knoten

Knotentyp (Interface)	Erläuterung
`Document`	Knoten repräsentiert die Wurzel des Knotenbaums
`DocumentType`	Knoten repräsentiert die Dokumenttyp-Deklaration
`Element`	Knoten repräsentiert ein HTML-/XML-Element inkl. seiner Attribute
`Text`	Knoten repräsentiert einen Text
`Comment`	Knoten repräsentiert einen Kommentar
`DocumentFragment`	Knoten repräsentiert ein Fragment siehe auch `template`-Element
`ProcessingInstruction`	Knoten repräsentiert eine Verarbeitungsanweisung

Knotenbaum (DOM-Baum)

Knoten-Typen und Interfaces

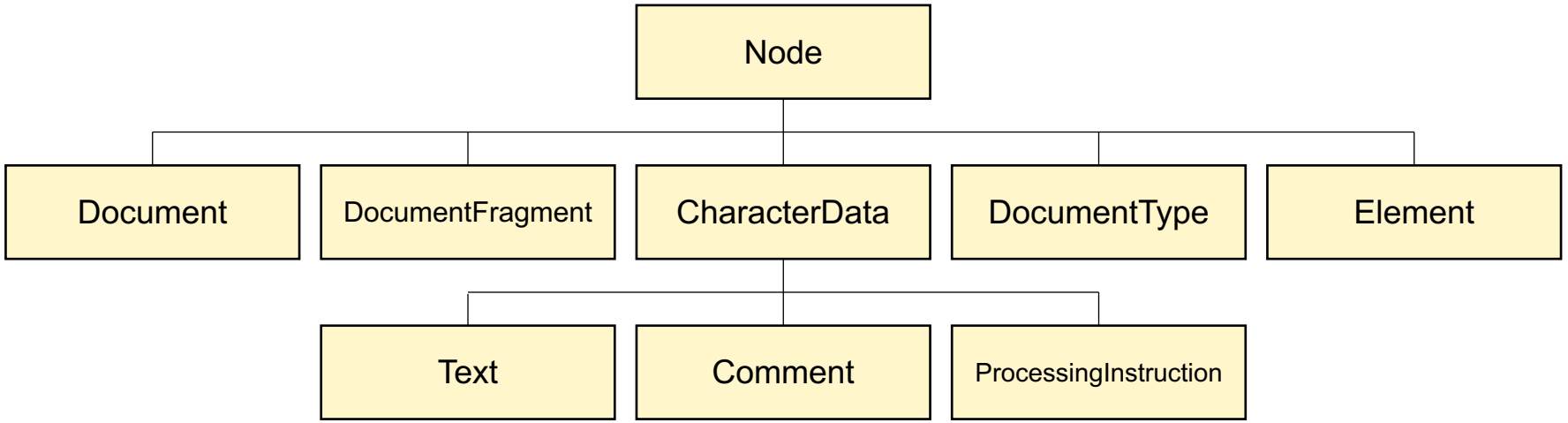
- Ein Interface beschreibt die Attribute und Methoden eines Knotens des zugehörigen Typs
  - Ein Knoten **implementiert** das zugehörige Interface

- Im DOM-Kontext werden auch folgende Bezeichnungen verwendet, um Missverständnisse zu vermeiden
  - **Inhaltsattribute** (*content attributs*) für HTML-Attribute
  - **IDL-Attribute** (*IDL attributs*) für Interface-Attribute

Knotenbaum (DOM-Baum)

Interface-Hierarchie (Ausschnitt)

- Die Interfaces stehen in einer hierarchischen Beziehung zueinander



- Weitere Interfaces für HTML (http://www.w3.org/TR/DOM-Level-2-HTML/)
  - `HTMLElement, HTMLImageElement, HTMLTableElement, ...`

Knotenbaum (DOM-Baum)

DOM-Zugriff in JavaScript (Browser)

- Jeder Knoten wird durch ein JavaScript-Objekt abgebildet
  - Die Attribute und Methoden des zugehörigen Interfaces stehen als Eigenschaften des JavaScript-Objekts zur Verfügung
  - Zu jedem Interface existiert ein gleichnamiges Standardobjekt (Eigenschaft des `window`-Objekts)
    - `window.Node`, `window.Element` etc. bzw. einfach `Node`, `Element` etc.
  - Dabei gilt z. B. für einen Elementknoten `e`

```
e instanceof Element; // true
```

- Der Document-Knoten zum aktuellen HTML-Dokument ist über das Standardobjekt `document` erreichbar

Knotenbaum (DOM-Baum)

Wesentliche Node-Eigenschaften

Eigenschaften	Erläuterung

`nodeType`	Typ des Knotens als Zahl
	`Element: 1, Text: 3, ProcessingInstruction: 7, Comment: 8,`
	`Document: 9, DocumentType: 10, DocumentFragment: 11`

`nodeName`	Knotenname (bei Elementknoten: Elementname großgeschrieben)
`parentNode`	Elternknoten

`childNodes`	Kindknoten
	`node.childNodes.length; // Anzahl Kindknoten`
	`node.childNodes[0]; // erster Kindknoten`

`firstChild`	Erster Kindknoten
`lastChild`	Letzter Kindknoten
`hasChildNodes`	`true`, falls Kindknoten existieren
`nextSibling`	Nächster Geschwisterknoten
`previousSibling`	Vorheriger Geschwisterknoten

Elemente ermitteln

Methoden

- Element mit bestimmter ID

```
let absatz1 = document.getElementById("absatz1");
```

  - Ergebnis ist ein `Element`-Objekt

- Elemente mit bestimmten Namen

```
let absaetze = document.getElementsByTagName("p");
```

  - Ergebnis ist ein `HTMLCollection`-Objekt
  - Funktion kann auch auf Element-Objekten aufgerufen werden

- Elemente mit bestimmter CSS-Klasse

```
let icons = document.getElementsByClassName("glyphicon");
```

  - Ergebnis und Anwendbarkeit wie bei `getElementsByTagName`

Elemente ermitteln

Methoden

- Elemente mit bestimmten Wert für das name-Attribut

```
let vornameEingabefelder = document.getElementsByName("vorname");
```

  - Ergebnis und Anwendbarkeit wie bei `getElementsByTagName`

- Element anhand eines CSS-Selektors ermitteln (erster Treffer)

```
let ersterArtikelAbsatz = document.querySelector("article p");
```

  - Ergebnis ist ein `Element`-Objekt
  - Funktion kann auch auf Element-Objekten aufgerufen werden

- Elemente anhand eines CSS-Selektors ermitteln (alle Treffer)

```
let alleArtikelAbsaetze = document.querySelectorAll("article p");
```

  - Ergebnis und Anwendbarkeit wie bei `getElementsByTagName`

Elemente ermitteln

HTMLCollection

- Repräsentiert eine Sammlung von Elementen (in Dokumentreihenfolge)
  - HTML/XML-Elemente (vor DOM4 nur HTML-Elemente)
  - Die Sammlung ist live, d. h. wird aktualisiert, wenn Elemente hinzukommen bzw. entfernt werden
- Iterieren

```
let collection = document.querySelectorAll('p');

for (let i = 0; i < collection.length; i++) {
  let element = collection[i];
  // ...
}
```

Elementinhalt auslesen

Eigenschaft textContent

- Die Node-Eigenschaft **`textContent`** enthält die Konkatenation aller Textknoten, die Nachfahren des Elementknotens sind (String)
  - Inkl. Leerraum zwischen den Elementen
- Beispiel

```
let artikel = document.getElementById('artikel');
alert('"' + artikel.textContent + '"');
```

```
<body>
  <article id="artikel">Text
    <h1>Überschrift</h1>
    <p>Absatz</p>
  </article>
</body>
```

"Text
Überschrift
Absatz
"

Elementinhalt auslesen

Eigenschaft innerHTML

- Die Element-Eigenschaft **`innerHTML`** enthält den Inhalt des Elements als HTML/XML-Fragment (String, inkl. Leerraum zwischen den Elementen)
  - Spezifikation: http://www.w3.org/TR/DOM-Parsing
- Beispiel

```
let artikel = document.getElementById('artikel');
alert('"' + artikel.innerHTML + '"');
```

```
<body>
  <article id="artikel">Text
    <h1>Überschrift</h1>
    <p>Absatz</p>
  </article>
</body>
```

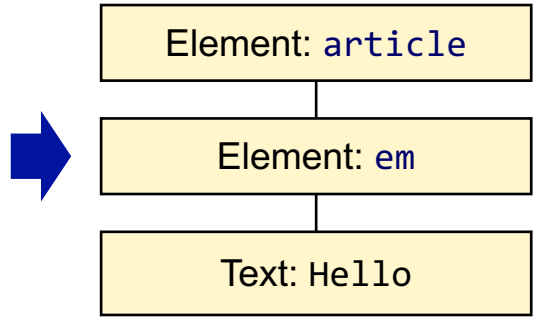
```
"Text
   <h1>Überschrift</h1>
   <p>Absatz</p>
 "
```

Elementinhalt setzen

Eigenschaften innerHTML und textContent

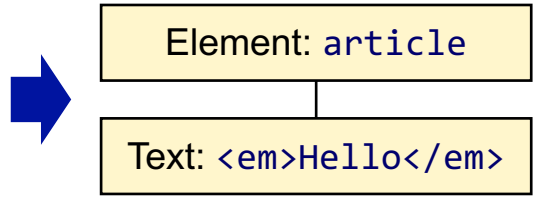
- Über die Element-Eigenschaft **`innerHTML`** können die Kindknoten des Elementknotens gesetzt bzw. ersetzt werden

```
let artikel = document.getElementById('artikel');
artikel.innerHTML = '<em>Hello</em>';
```



- Die Element-Eigenschaft **`textContent`** hingegen erlaubt nur das Setzen textueller Inhalte (Textknoten)

```
let artikel = document.getElementById('artikel');
artikel.textContent = '<em>Hello</em>';
```



Auf Attribute zugreifen

Inhaltsattribute (HTML-Attribute)

- Abfragen, ob ein bestimmtes Attribut existiert

```
if (element.hasAttribute('width')) { }
```

- Attributwert auslesen (liefert einen String)

```
let width = element.getAttribute('width');
```

- Attributwert setzen

```
element.setAttribute('width', '50');
```

- Attribut entfernen

```
element.removeAttribute('width');
```

Auf Attribute zugreifen

IDL-Attribute

- Zu den meisten Inhaltsattributen existiert ein IDL-Attribut

```
element.getAttribute('width');
```

Inhaltsattribut

```
element.width; // Number
```

IDL-Attribut

- Vergleich mit Inhaltsattributen
  - Anders als IDL-Attribute sind Inhaltsattribute stets vom Typ `String`
  - Zu selbstdefinierten Inhaltsattributen (z. B. data-foldername) existiert kein IDL-Attribut
  - Nicht jedes Inhaltsattribut hat stets den gleichen Wert wie das zugehörige IDL-Attribut (z. B. `value` bei `input`-Elementen)
  - Namen sind nicht immer gleich (z. B. class vs. className)

Auf Attribute zugreifen

Benutzerdefinierte Attribute

- Benutzerdefinierte Attribute sind über die Eigenschaft `dataset` abrufbar, deren Wert ein **`DOMStringMap`**-Objekt ist

```
let taskId = element.dataset.taskId;
```

- Es enthält für jedes benutzerdefinierte Attribut eine Eigenschaft, wobei der Attributname wie folgt konvertiert wird
  - Präfix „data-“ wird entfernt
  - Rest wird in Camel-Case-Schreibweise umgewandelt

Attributname	Eigenschaftsname
`data-length`	`length`
`data-task-id`	`taskId`

CSS-Eigenschaft setzen

Eigenschaft style

- Die Eigenschaft `style` eines Element-Objekts ermöglicht es, CSS-Eigenschaften für das Element zu setzen
  - Inhalt der `style`-Eigenschaft ist ein `CSSStyleDeclaration`-Objekt
    - Spezifikation: http://www.w3.org/TR/cssom-1/
  - Eine CSS-Eigenschaft wird als Eigenschaft des `CSSStyleDeclaration`-Objekts gesetzt
    - Eigenschafts-Name = Name der CSS-Eigenschaft in Camel Case-Schreibweise
  - Spezifität ist wie bei einer Deklaration in einem `style`-Attribut
- Beispiel

```
let absatz = document.getElementById('absatz');
absatz.style.backgroundColor = 'yellow';
```

CSS-Eigenschaft auslesen

- Die Methode `getComputedStyle` des `Window`-Objekts liefert zu einem Element alle berechneten CSS-Eigenschaften
  - Rückgabewert ist ein `CSSStyleDeclaration`-Objekt
- Beispiel

```
let absatz = document.getElementById('absatz');
let fontSize = window.getComputedStyle(absatz).fontSize;
```

Weitere Baum-Operationen

- Element erzeugen

```
let element = document.createElement('p');
element.innerHTML = 'Inhalt';
```

  - Ergebnis ist ein `Element`-Objekt (hier: `HTMLParagraphElement`)

- Knoten als letzten Kindknoten hinzufügen

```
parentNode.appendChild(node);
```

  - Rückgabewert ist der hinzugefügte Knoten

- Knoten vor einem anderen Kindknoten hinzufügen

```
parentNode.insertBefore(node, otherChildNode);
```

  - Rückgabewert ist der hinzugefügte Knoten

Weitere Baum-Operationen

- Knoten entfernen

```
parentNode.removeNode(node);
```

  - Rückgabewert ist der entfernte Knoten

- Knoten ersetzen

```
parentNode.replaceChild(newChildNode, oldChildNode);
```

  - Rückgabewert ist der ersetzte (alte) Knoten

- Knoten klonen

```
let clone = node.cloneNode(true);
```

  - Wird `true` übergeben, so wird der gesamte Teilbaum geklont, sonst nur der Knoten

Mikroübung
Quellcode-Analyse

- Was passiert beim Laden des folgenden HTML-Dokuments?

```
<!DOCTYPE html>
<html>
<head><title>Clock</title></head>
<body>
  <h1>Zeit: <span id="time"></span></h1>
  <script>
    let elem = document.getElementById('time');
    let counter = 0;
    setInterval(() => { elem.innerHTML = ++counter }, 1000);
  </script>
</body>
</html>
```

Auf Ereignisse reagieren

Ereignisse

- Bei Benutzereingaben, Netzwerkaktivitäten etc. werden **Ereignisse** *(events)* erzeugt
- Ereignisse sind Objekte, die das Interface `Event` oder ein davon abgeleitetes Interface implementieren
- Ein Ereignis wird an ein sogenanntes **Event Target** geschickt (*dispatched*). Interface: `EventTarget`
  - Wird implementiert von Window-Objekten, Document-Objekten und Elementen
- Auf Ereignisse kann reagiert werden, indem am Event Target sogenannte **Event Handler** bzw. **Event Listener** registriert werden

Auf Ereignisse reagieren

Event Handler

- Event Handler über ein Inhaltsattribut setzen*

```
<p onclick="alert('event.type: ' + event.type);">Click me</p>
```

  - Kann mit `removeAttribute('onclick')` deaktiviert werden

- Event Handler über ein IDL Attribut setzen*

```
<p id="absatz">Click me</p>
<script>
  document.getElementById('absatz').onclick = function(event) {
    alert('event.type: ' + event.type);
  }
</script>
```

  - Funktion bekommt `Event`-Objekt als Parameter übergeben
  - Kann deaktiviert werden, indem das Attribut auf `null` gesetzt wird

*Attributname: on + Event-Name (`event.type`)

Auf Ereignisse reagieren

Event Listener

- Event Listener hinzufügen

```
let element = document.getElementById('absatz');
element.addEventListener('click', function(event) {
  alert('event.type: ' + event.type);
});
```

Parameter	Erläuterung
`type`	Event-Name (siehe `event.type`)
`callback`	Callback-Funktion
`capture`	Wenn `true`/`false`, dann wird der Callback in der Bubbling-Phase/Capturing-Phase nicht aufgerufen (Default: `false`)

- Event Listener wird nur dann hinzugefügt, wenn zuvor noch kein Event Listener mit gleichen Werten für die drei Parameter hinzugefügt wurde

Auf Ereignisse reagieren

Event Listener

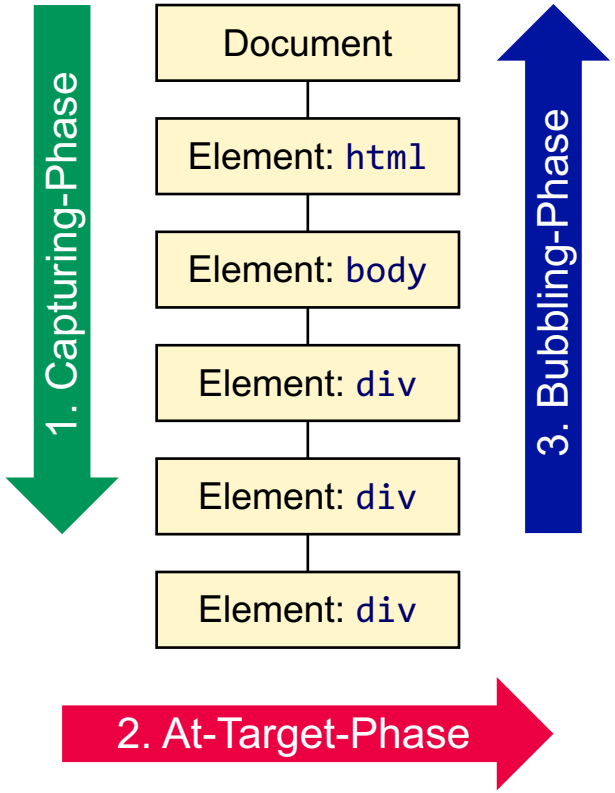
- Event Listener entfernen
  - Ein zuvor registrierter Event Listener kann über die Methode `removeEventListener` wieder entfernt werden
    - gleiche Parameter wie bei `addEventListener`
  - Beispiel

```
<p id="absatz">Click me</p>
<script>
  function clickListener(event) {
    alert('event.type: ' + event.type);
    event.target.removeEventListener('click', clickListener);
  }
  let element = document.getElementById('absatz');
  element.addEventListener('click', clickListener);
</script>
```

Auf Ereignisse reagieren

Event-Verarbeitung: Phasen

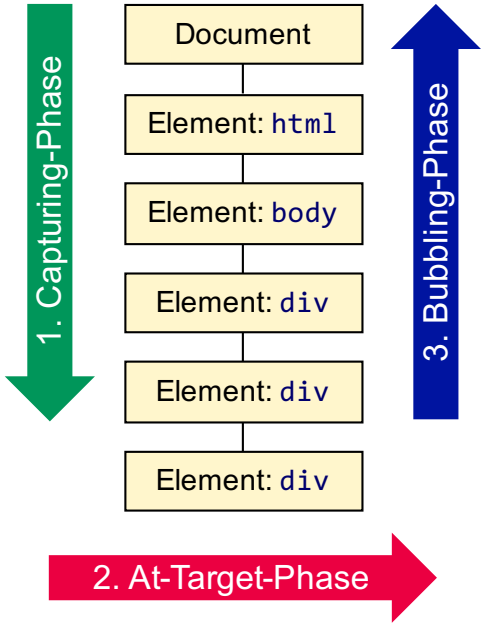

```
<!doctype html>
<html>
<head>
  <!-- ... -->
</head>

<body>
  <div id="box1">box1
    <div id="box2">box2
      <div id="box3">box3</div>
    </div>
  </div>
</body>
</html>
```



Auf Ereignisse reagieren

Event-Verarbeitung: Phasen

- Capturing-Phase
  - Das Ereignis steigt im DOM-Baum vom Dokument-Knoten bis zum Vorgänger des Zielelement-Knotens ab. Dabei werden alle **Event Listener** aufgerufen, die am **jeweiligen Elementknoten** (mit `true` für den 3. Parameter) registriert wurden.
- At Target-Phase
  - Es werden der **Event Handler und alle Event Listener** aufgerufen, die am Zielelementknoten registriert wurden
- Bubbling-Phase (wird nicht für alle Ereignisse ausgeführt)
  - Das Ereignis steigt vom Vorgänger des Zielelement-Knotens zum Dokument-Knoten auf. Dabei werden der **Event Handler und alle Event Listener** aufgerufen, die am **jeweiligen Elementknoten** (mit `false` für den 3. Parameter) registriert wurden.



Auf Ereignisse reagieren

Event-Verarbeitung: Phasen - Beispiel

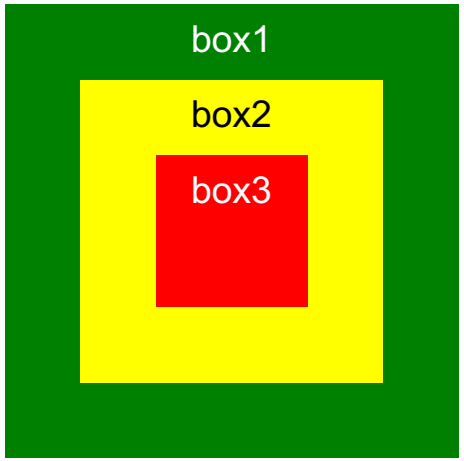
FH MÜNSTER
University of Applied Sciences

HTML-Code

```
<div id="box1">box1
  <div id="box2">box2
    <div id="box3">box3</div>
  </div>
</div>
```

Für **jede** Box

```
box.addEventListener('click',
  clickListener, false);
box.addEventListener('click',
  clickListener, true);
```



Beim Klick auf die innerste Box (box3)

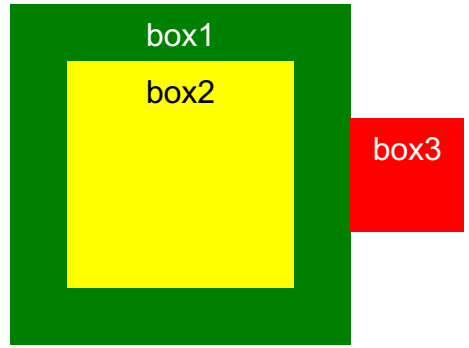
Phase	Event Target	Aktuelles Event Target
Capturing-Phase	box3	box1
Capturing-Phase	box3	box2
At-Target-Phase	box3	box3
At-Target-Phase	box3	box3
Bubbling-Phase	box3	box2
Bubbling-Phase	box3	box1

Auf Ereignisse reagieren

Event-Verarbeitung: Phasen

- **Achtung:** Es ist nur die Element-Hierarchie und nicht die Darstellung der Elemente für das Capturing/Bubbling relevant

```
<div id="box1">box1
  <div id="box2">box2
    <div id="box3">box3</div>
  </div>
</div>
```



  - Beim Klick auf die box3 werden auch die Listener, die für box1 und box2 registriert wurden, aufgerufen (wie beim vorherigen Beispiel)

Auf Ereignisse reagieren

Event-Verarbeitung

- Es gibt Events, für die die Bubbling-Phase nicht ausgeführt wird
  (z. B. `blur`, `focus`, `mouseenter`, `mouseleave`)
- Beenden der Capturing- bzw. Bubbling-Phase

```
event.stopPropagation();
```

- Abbruch der Verarbeitung durch weitere Listener / Handler

```
event.stopImmediatePropagation();
```

  - schließt das Beenden der Capturing- bzw. Bubbling-Phase ein

- Das Browser-Standard-Verhalten unterbinden

```
event.preventDefault();
```

  - z. B. beim Submit-Button eines Formulars sinnvoll einsetzbar

Auf Ereignisse reagieren

Typische Ereignisse

- Tastatur-Ereignisse

Eventname	Erläuterung
`keydown, keypress, keyup`	Wenn eine Taste der Tastatur gedrückt wird (Events in der angegebenen Reihenfolge)

- Maus-Ereignisse

Eventname	Erläuterung
`mousedown, mouseup, click`	Wenn die linke Maustaste gedrückt wird
`mousedown, mouseup, contextmenu`	Wenn die rechte Maustaste gedrückt wird
`dblclick`	Wenn ein Doppelklick erfolgt
`mouseenter, mouseleave`	Wenn der Mauszeiger das Element betritt/verlässt
`mousemove`	Wenn der Mauszeiger innerhalb des Elements bewegt wird

Auf Ereignisse reagieren

Typische Ereignisse

- Touch-Ereignisse

Eventname	Erläuterung
`touchstart, touchend`	Wenn der Finger aufgesetzt bzw. wieder entfernt wird
`touchmove`	Wenn der Finger auf dem Screen bewegt wird

- Formular-Ereignisse

Eventname	Erläuterung
`blur`	Wenn das Element den Fokus verliert
`focus`	Wenn das Element den Fokus erhält
`change`	Wenn der Wert von `value` bzw. `checked` sich ändert
`submit`	Wenn ein Formular abgeschickt werden soll

Auf Ereignisse reagieren

Wesentliche Ereignis-Eigenschaften

- Event

Eigenschaft	Erläuterung
`type`	Event-Name
`eventPhase`	Aktuelle Phase der Event-Verarbeitung (`0: NONE, 1: CAPTURING_PHASE, 2: AT_TARGET, 3: BUBBLING_PHASE`)
`target`	Event Target
`currentTarget`	Aktuelles Event Target (auch via `this` im Event Handler bzw. Callback eines Event Listeners verfügbar)

Auf Ereignisse reagieren

Wesentliche Ereignis-Eigenschaften

- `MouseEvent`

Eigenschaft	Erläuterung
`clientX, clientY`	X- bzw. Y-Koordinate in Pixeln bezogen auf das Fenster
`screenX, screenY`	X- bzw. Y-Koordinate in Pixeln bezogen auf den Bildschirm
`shiftKey, altKey, ctrlKey`	`true`, falls beim Klick die Shift- bzw. Alt- bzw. Strg-Taste gedrückt war

- `KeyboardEvent`

Eigenschaft	Erläuterung
`charCode`	Unicode Zeichencode der gedrückten Taste (z. B. 97 für "a") Umwandlung in einen String: `String.fromCharCode(97)`
`shiftKey, altKey, ctrlKey`	`true`, falls zudem die Shift- bzw. Alt- bzw. Strg-Taste gedrückt war

Mikroübung
Quellcode-Analyse

- Was passiert beim Klick auf den Button?

```
<!DOCTYPE html>
<html>
<head><title>Capturing</title>
  <script>
    document.addEventListener('click', (event) => {
      event.stopPropagation();
    }, true);
  </script>
</head>
<body>
  <button onclick="alert('clicked');">Click me</button>
</body>
</html>
```

Inhaltsübersicht

Canvas API

- Ermöglicht es, pixelbasierte Bilder, Grafiken und Animationen mit JavaScript zu erzeugen
- IDL-Interface: `CanvasRenderingContext2D`
  - Ein `CanvasRenderingContext2D`-Objekt ist über die `getContext()`-Methode eines `canvas`-Elements erreichbar
- Spezifikation
  - http://www.w3.org/TR/2dcontext

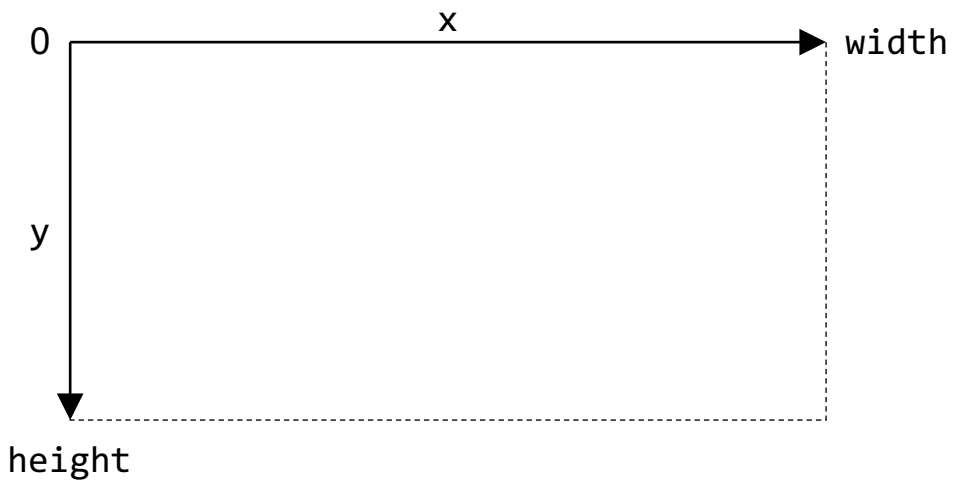
Canvas API

HTML-Element canvas

- `canvas` ist ein HTML-Element, das eine Leinwand bereitstellt, die über die Canvas API mit Inhalten gefüllt bzw. ausgelesen werden kann

```
<canvas id="myCanvas" width="600" height="400"></canvas>
```

- Koordinatensystem der Leinwand



Canvas API

- Ermitteln des Context-Objekts (`CanvasRenderingContext2D`)

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
```

  - Spezifikation: http://www.w3.org/TR/2dcontext/

- Konfiguration des Context-Objekts bei hochauflösenden Displays

```
let oldWidth, oldHeight, scaleFactor = window.devicePixelRatio;
if (scaleFactor > 1) {
  oldWidth = canvas.width;
  oldHeight = canvas.height;
  canvas.width = oldWidth * scaleFactor;
  canvas.height = oldHeight * scaleFactor;
  canvas.style.width = oldWidth + 'px';
  canvas.style.height = oldHeight + 'px';
  ctx.scale(scaleFactor, scaleFactor);
}
```

Canvas API

- Setzen der Linien- bzw. Füllfarbe

```
ctx.strokeStyle = '#000';
```

```
ctx.fillStyle = 'green';
```

- Zeichnen eines Rechtecks bzw. ausgefüllten Rechtecks

```
ctx.strokeRect(0, 0, 100, 100);
```

```
ctx.fillRect(0, 0, 100, 100);
```

  - Parameter: x-Koordinate, y-Koordinate, Breite, Höhe
- Das Context-Objekt bietet zahlreiche weitere Operationen an
  - Zeichnen von Kreisen, Bézierkurven, beliebigen Polygonen, Texten etc.
  - Speichern und Zurücksetzen des Canvas-Zustands
  - Skalierung, Rotation, Translation
  - Extraktion der binären Bilddaten
  - ...

Geolocation API

- Ermöglicht die einmalige oder regelmäßige Abfrage von Positionsdaten des Benutzers
  - Erfordert dessen Zustimmung
- IDL-Interface: `Geolocation`
  - `Geolocation`-Objekt ist über die Eigenschaft `geolocation` eines Navigator-Objekts erreichbar
- Spezifikation
  - http://www.w3.org/TR/geolocation-API

Geolocation API

- Einmaliges Auslesen der Positionsdaten

```
navigator.geolocation.getCurrentPosition(
  function(position) {
    console.log(position.latitude + ', ' + position.longitude);
  },
  function(error) {
    console.log('Error code: ' + error.code);
  }
);
```

  - Die erste Funktion wird aufgerufen, wenn die Position ermittelt werden konnte. Argument ist ein `Position`-Objekt.
  - Die zweite Funktion wird im Fehlerfall aufgerufen. Argument ist ein `PositionError`-Objekt mit u. a. folgenden Eigenschaften für Error-Codes
    - `PERMISSION_DENIED (1), POSITION_UNAVAILABLE (2), TIMEOUT (3)`

Geolocation API

- Eigenschaften eines `Position`-Objekts

Eigenschaft	Erläuterung
`latitude`	geografische Breite
`longitude`	geografische Länge
`altitude`	Höhe (Meter)
`heading`	Richtung (Grad, 0 bis 360)
`speed`	Geschwindigkeit (Meter pro Sekunde)
`timestamp`	Zeitstempel der Anfrage
`accuracy`	Genauigkeit der Breite/Länge (Meter)
`altitudeAccuracy`	Genauigkeit der Höhe (Meter)

Geolocation API

- Position dauerhaft überwachen

```
let watchId = navigator.geolocation.watchPosition(successCallback,
                                                  errorCallback);
```

  - Parameter wie bei `getCurrentPosition`
  - Rückgabewert ist eine eindeutige ID (Zahl)
  - Die erste Callback-Funktion wird bei jeder Positionsänderung aufgerufen
- Konfiguration
  - Beiden Funktionen kann als dritter Parameter ein Konfigurationsobjekt übergeben werden (u. a. zur Angabe des maximales Alter der Positionsdaten)
- Beenden der Positionsüberwachung

```
navigator.geolocation.clearWatch(watchId);
```

History API

- Mit HTML5 hat ein `History`-Objekt neue Methoden, die ein Ändern der Navigationshistorie (*session history*) ohne Absenden von HTTP-Anfragen erlauben
  - Ermöglicht ein clientseitiges Routing bei Single Page Applications
- IDL-Interface: `History`
  - `History`-Objekt ist über die Eigenschaft `history` eines Window-Objekts erreichbar
- Spezifikation
  - http://www.w3.org/TR/html/browsers.html#history-1

History API

- Neuen History-Eintrag hinzufügen

```
let stateObj = { name: "Max Mustermann" };
history.pushState(stateObj, "contact", "contact/4711");
```

  - Eintrag ist eine Kombination aus
    - einem Zustandsobjekt (Kopie des 1. Arguments)
    - einem Titel (2. Argument)
    - einer relativen URL (3. Argument, optional)

- Aktuellen History-Eintrag ersetzen

```
history.replaceState(stateObj, "home");
```

  - Insbesondere für den initialen Zustand einer Single Page App
  - Gleiche Parameter wie bei `pushState`

History API

- Zustandsobjekt des aktuellen History-Eintrags ermitteln

```
let currentStateObj = history.state;
```

- Auf Navigationsereignisse reagieren

```
window.onpopstate = function(event) { };
```

  - Parameter ist ein `PopStateEvent`-Objekt, über dessen Eigenschaft `state` das Zustandsobjekt ermittelt werden kann

XMLHttpRequest API

- Erlaubt die Übertragung von Daten per HTTP und HTTPS
  - Nicht auf XML beschränkt (auch reiner Text, HTML, JSON und Binärdaten)
  - Alle HTTP-Methoden möglich (GET, PUT, POST, DELETE etc.)
- IDL-Interface: `XMLHttpRequest`
  - Die Konstruktorfunktion `XMLHttpRequest` ist über die gleichnamige Eigenschaft eines Window-Objekts erreichbar
- Spezifikation
  - http://www.w3.org/TR/XMLHttpRequest/

XMLHttpRequest API

- `XMLHttpRequest`-Objekt erzeugen

```
let request = new XMLHttpRequest();
```

- Methode und URL setzen

```
request.open('GET', 'https://api.github.com');
```

  - Über den optionalen dritten Parameter kann definiert werden, dass die Anfrage synchron (`false`) oder asynchron (`true` = Default) erfolgen soll

- Erwarteten Antworttyp setzen (optional)

```
request.responseType = "json";
```

  - Mögliche Werte: `""`, `"text"`, `"arraybuffer"`, `"blob"`, `"document"`, `"json"`
  - Der Leerstring ist der Standardwert und wird wie `"text"` behandelt

XMLHttpRequest API

- Anfrage-Header setzen (optional)

```
request.setRequestHeader('Accept', 'application/json');
```

- Event-Handler für die Antwort-Verarbeitung setzen

```
request.onload = function (event) {
  let messageBody = request.response;
  let contentType = request.getResponseHeader('Content-Type');
  let statusCode = request.status;
  let reasonPhrase = request.statusText;
}
```

  - Parameter ist ein `ProgressEvent`-Objekt
  - Variante mit Event-Listener

```
request.addEventListener('load', function(event) { });
```

XMLHttpRequest API

- Event-Handler für die Fehlerbehandlung setzen

```
request.onerror = function (event) {
  console.error('Es ist ein Fehler aufgetreten');
}
```

  - Parameter ist ein `ProgressEvent`-Objekt
  - Variante mit Event-Listener

```
request.addEventListener('error', function(event) { });
```

- Anfrage absenden

```
request.send();
```

  - Optionaler Parameter: Daten, die zum Server geschickt werden sollen (`String`, `ArrayBufferView`, `Blob`, `Document` oder `FormData`)

XMLHttpRequest API

Beispiel: GET-Anfrage


```
let request = new XMLHttpRequest();
request.open('GET', 'https://api.github.com', true);
request.setRequestHeader('Accept', 'application/json');
request.responseType = 'json';
request.onload = function (event) {
  if (request.status === 200) {
    console.log(JSON.stringify(request.response));
  } else {
    console.log('Status: ' + request.status + ' ' + request.statusText);
  }
}
request.onerror = function (event) {
  console.error('Es ist ein Fehler aufgetreten');
}
request.send();
```

XMLHttpRequest API

Beispiel: POST-Anfrage


```
let data = { user: 'Max', content: 'Hello World!'};

let request = new XMLHttpRequest();
request.open('POST', 'http://localhost:3000/comments', true);
request.setRequestHeader('Content-Type', 'application/json');
request.onload = function (event) {
  if (request.status === 201) {
    console.log('Success');
  } else {
    console.log('Status: ' + request.status + ' ' + request.statusText);
  }
}
request.onerror = function (event) {
  console.error('Es ist ein Fehler aufgetreten');
}
request.send(JSON.stringify(data));
```

Weitere APIs

Auswahl

API	Erläuterung
Drag and Drop	Spezifikation: https://html.spec.whatwg.org/multipage/interaction.html#dnd
HTML Media Capture	Zugriff auf Kamera und Mikrofon innerhalb eines Dateiupload-Controls Spezifikation: http://www.w3.org/TR/html-media-capture/
Web Messaging	Nachrichtenaustausch zwischen Dokumenten (auch bei verschiedenen Origins) Spezifikation: http://www.w3.org/TR/webmessaging/
Web Storage	Persistente Speicherung von Schlüssel-Wert-Paaren in Web Clients Spezifikation: http://www.w3.org/TR/webstorage/
Web Workers	Ausführung von Aufgaben im Hintergrund mit Message Passing als Koordinationsmechanismus Spezifikation: www.w3.org/TR/workers/