FH Bielefeld
University of
Applied Sciences

Campus Minden

Webbasierte Anwendungen

SS 2018

Document Object Model (DOM)

Dozent: B. Sc. Florian Fehring

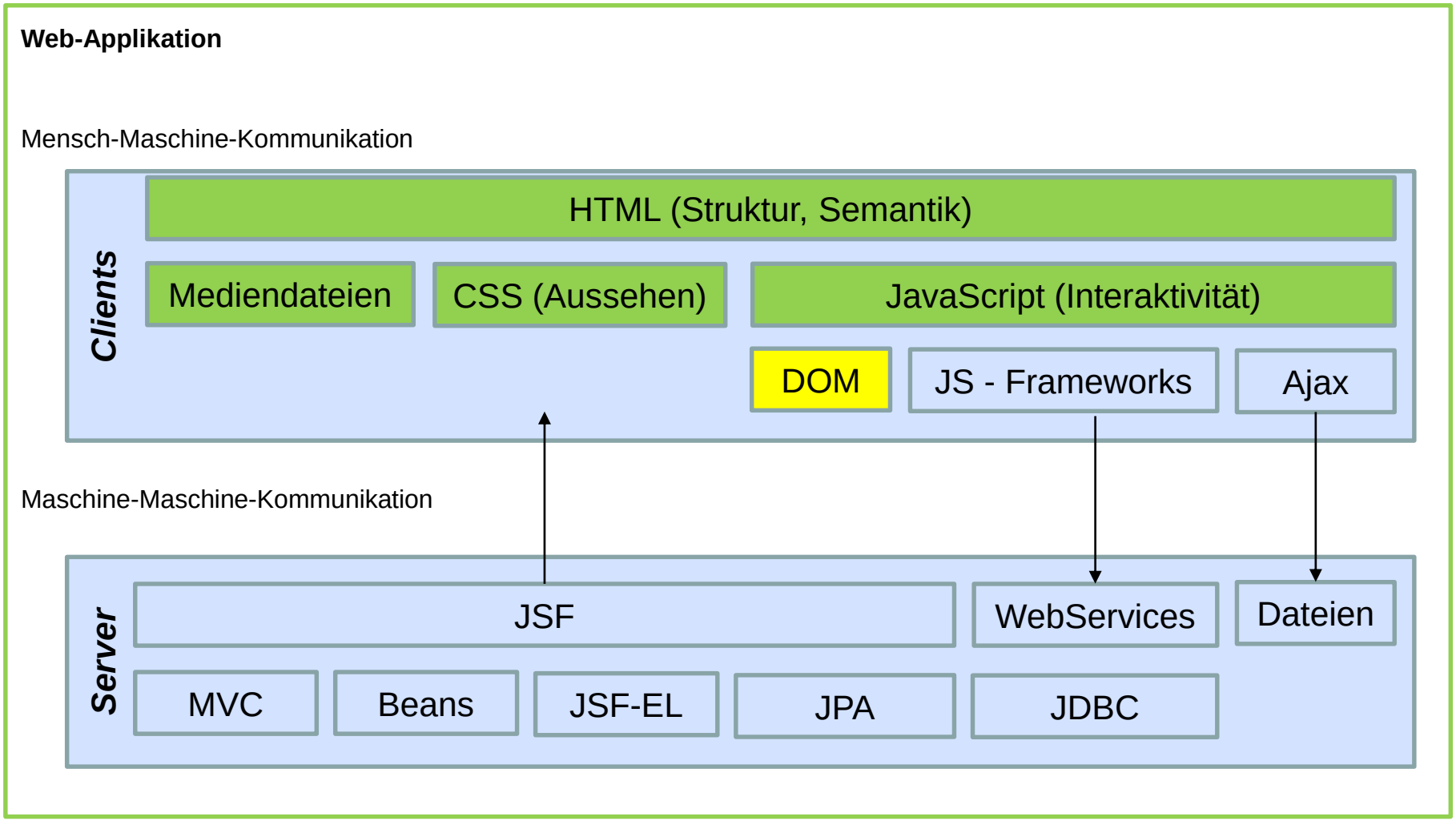
mailto: florian.fehring@fh-bielefeld.de

Studiengang Informatik

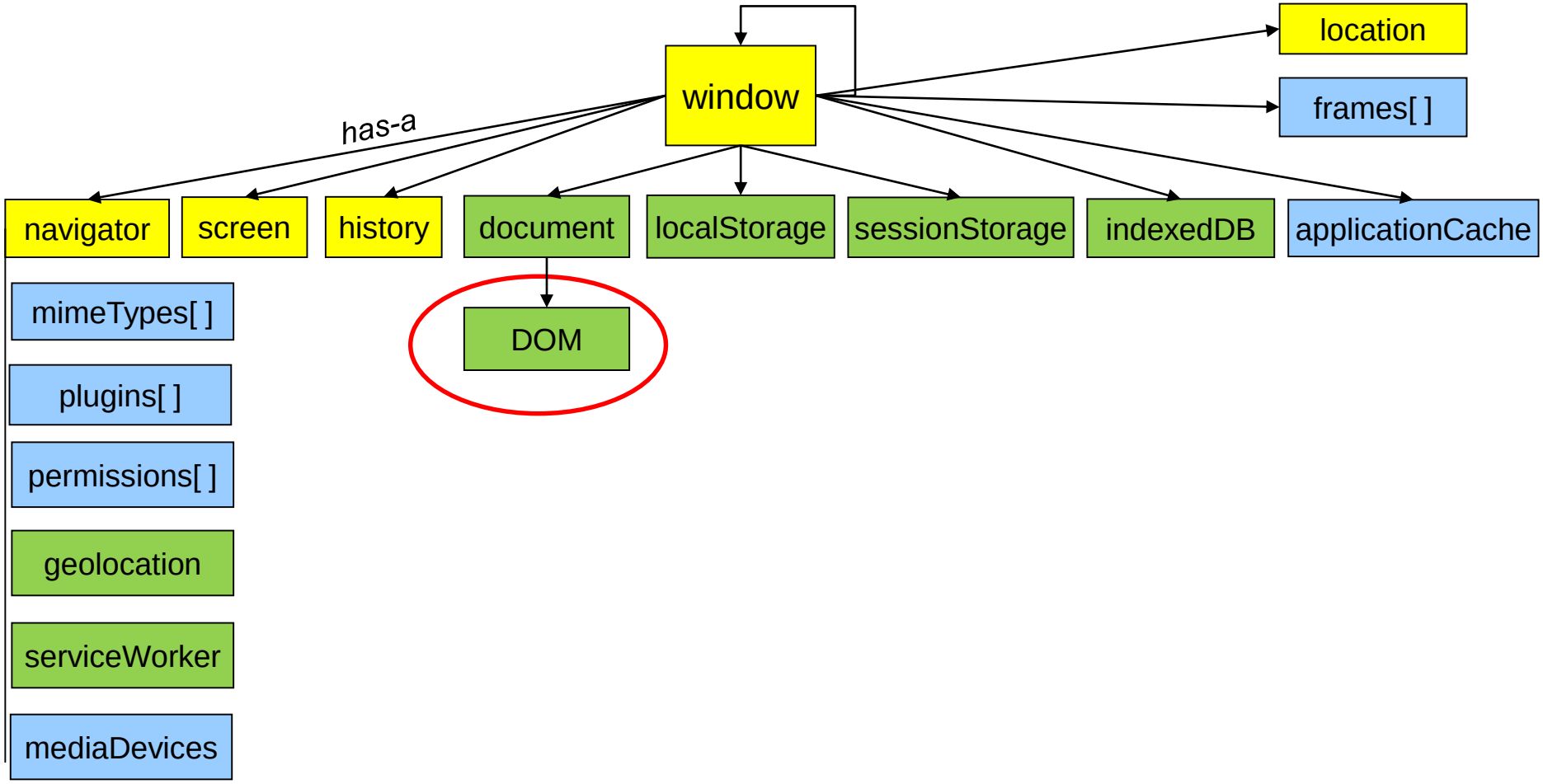
DOM

1. **Kontext und Motivation**
2. Allgemeines
3. Baumstruktur
4. DOM Zugriffe
5. DOM Manipulation
6. SpecialObjects
7. Event-Handling
8. Darüber hinaus
9. Projekt

Problemfelder



Browser-Objekte I - Objektmodell



Weitere Informationen screen: https://wiki.selfhtml.org/wiki/JavaScript/Screen

Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel
- Externe Inhalte einbinden
- Medien hochladen / runterladen
- Kommentare hochladen / runterladen
- Kommentare speichern
- Kommunikation untereinander

DONE

- Technologische Grundlagen erarbeiten
- Was ist eine Web-Anwendung?
- News darstellen
- Projekte vorstellen
- Aufgaben darstellen
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendatein einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen

DOM

1. Kontext und Motivation
2. **Allgemeines**
3. Baumstruktur
4. DOM Zugriffe
5. DOM Manipulation
6. SpecialObjects
7. Event-Handling
8. Darüber hinaus
9. Projekt

Allgemeines I – DOM

Definition: **Das Document Object Model (DOM) ist standardisiertes Fundament moderner Webanwendungen**

Eigenschaften:

- zum großen Teil in aktuellen Browsern gleich interpretiert
- plattform- und sprachunabhängig
- Schnittstelle für den Zugriff auf XML- und HTML-Dokumente
- dynamischer Zugriff: Bearbeitung und Löschen von einzelnen Dokumentelementen (Inhalt, Struktur, Layout)
- Bearbeitung des DOM mit JavaScript
  (nach ECMA262 Standard siehe http://www.ecma-international.org/publications/standards/Ecma-262.htm

Es gibt weitere Auszeichnungssprachen, auf die das DOM anwendbar ist:

- MathML – Beschreibung mathematischer Ausdrücke
- SVG – Scaleable Vector Graphics

Allgemeines II - Geschichte

Während der Browserkriege in erster Version entstanden und
Meilenstein für Beginn einer neuen Generation des WWW

Geschichte vom DOM:

DOM L0 Implementiert in Netscape 2.0

- nicht formal spezifiziert.
- bezeichnet mittels JavaScript nutzbare Techniken zum ***Zugriff auf HTML-Dokumente***

DOM L1 Standardisierung durch da W3C

- ***Bewegen im DOM-Baum*** und ***Manipulation der Knoten*** inklusive des ***Einfügens neuer Elemente*** und des Setzens von Attributen.

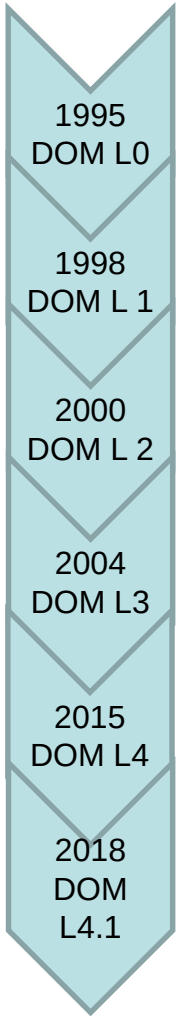
DOM L2 Ergänzungen für XHMTL, XML Namensräume und CSS

- dynamisches Auslesen, Hinzufügen und Ändern des Layouts
- Verarbeitung von Ereignissen im Dokument
- Durchlaufen des Knotenbaums anhand von bestimmten Auswahlkriterien

DOM L3 verbesserte Ausnahmebehandlung, Serialisierung

DOM L4 Verbesserte Selections-Möglichkeiten, CustomEvents

- Vereinheitlichungen für Zusammenarbeit mit JavaScript und HTML



Allgemeines III – DOM und JavaScript

Definition: **Das Document Object Model (DOM) wird in JavaScript Objekte abgebildet.**

Zu beachten:

- Das DOM wird schrittweise geladen, Skripte im Head werden beim Laden ausgeführt, noch bevor der Rest vom DOM geladen ist
- Nur Skripte im selben Scope wie das Dokument haben Zugriff auf das DOM. (Kein Zugriff auf DOM aus WebWorkern / ServiceWorkern)

```
// Funktion die nach dem vollständigen Laden des DOM ausgeführt wird
window.onload = function() {
   // Hier ist der Zugriff auf DOM Elemente sicher möglich
}
```

DOM

1. Kontext und Motivation
2. Allgemeines
3. **Baumstruktur**
4. DOM Zugriffe
5. DOM Manipulation
6. SpecialObjects
7. Event-Handling
8. Darüber hinaus
9. Projekt

Baumstruktur I - Struktur

Definition: **Eine vollständig übertragene Webseite repräsentiert einen hierarchisch geordneten Dokumentenbaum, in dem jedes HTML-Element einen Knoten bildet.**

Eigenschaften:

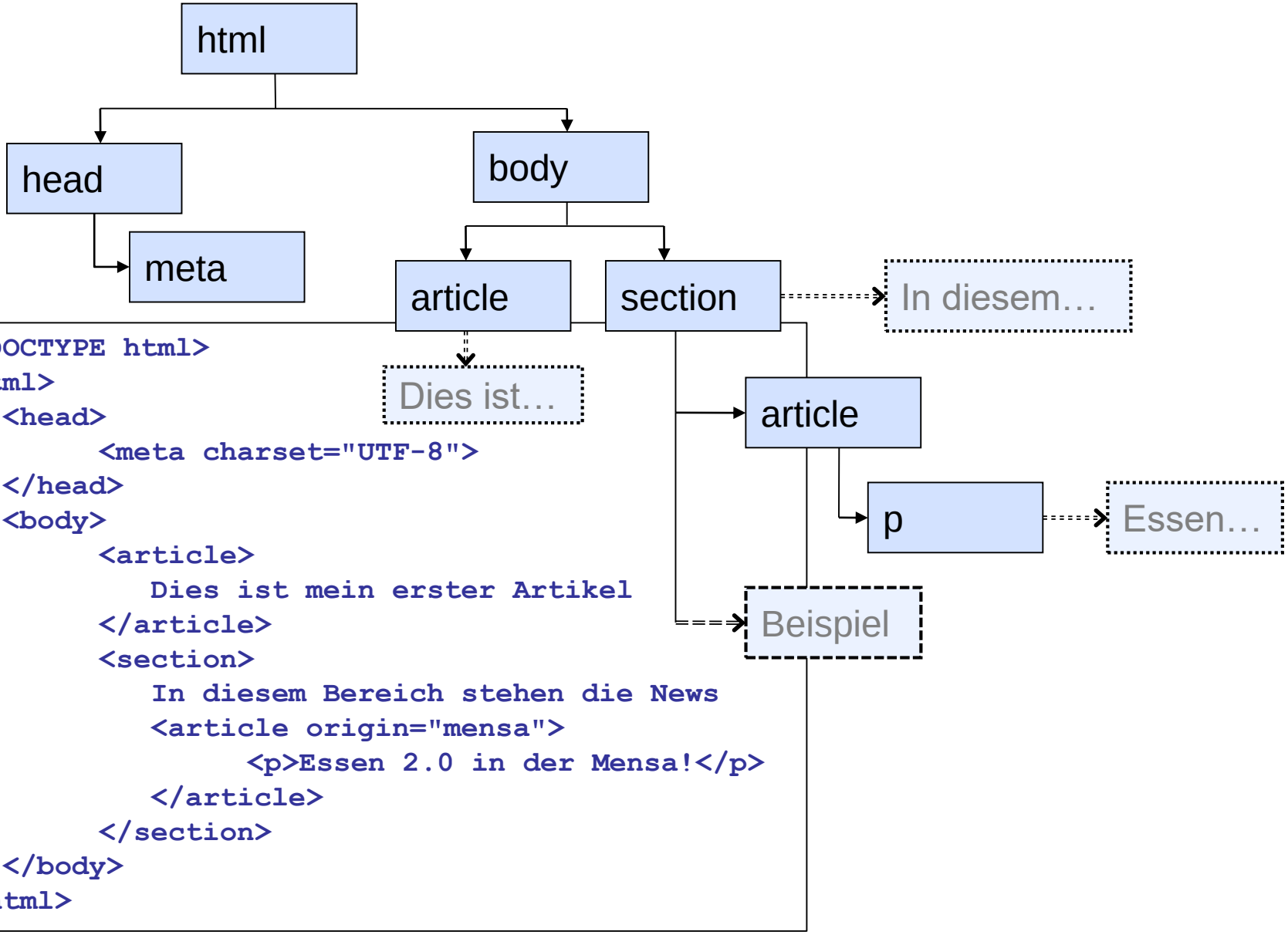
- hierarchisch: Elemente stehen in einer Eltern-Kind Beziehung
- Die Wurzel ist das HTMLDocument-Objekt (in HTML Dokumenten)
- Die Wurzel ist über die Eigenschaft document des Browsers zugreifbar

```
<!DOCTYPE html>
<html>
   <head>
         <meta charset="UTF-8">
   </head>
   <body>
         <article>
            Dies ist mein erster Artikel
         </article>
         <section>
            In diesem Bereich stehen die News
            <article origin="mensa">
                  <p>Essen 2.0 in der Mensa!</p>
            </article>
         </section>
   </body>
</html>
```

Knoten im Dokument:

<!DOCTYPE html>
<html>
<head>
<meta>
<body>
<article> (2x)
<section>
<article>
<p>

Baumstruktur I – Der Dokumentenbaum




```
<!DOCTYPE html>
<html>
   <head>
         <meta charset="UTF-8">
   </head>
   <body>
         <article>
            Dies ist mein erster Artikel
         </article>
         <section>
            In diesem Bereich stehen die News
            <article origin="mensa">
                  <p>Essen 2.0 in der Mensa!</p>
            </article>
         </section>
   </body>
</html>
```

Baumstruktur II - Knoten

Definition: **DOM Knoten besitzen Eigenschaften und Methoden, abhängig von ihrem Knotentyp. Sie sind Objekte und über JavaScript zugreifbar.**

Eigenschaften:

- können Verweise auf (Kind)-Knoten sein (auch Arrays davon)
- können Eigenschaftswerte halten (z.B. Namen eines Knoten)
- können Skript-Definiert sein

```
Knoten.eigenschaft
```

Methoden:

- sind mächtige Werkzeuge bei der Bearbeitung einer Seite
- Erlauben das dynamische löschen, verändern oder einstellen von Knoten
- können nützliche Informationen für die Webanwendung liefern

```
Knoten.methode()
```

Bei Knoten die laut HTML-Standard bestimmte Kindknoten haben, sind diese oft als Eigenschaft zugänglich

```
document.body   // Zugriff auf das body-Element
```

Baumstruktur II – Knotentypen

DOM definiert 7 Typen von Knoten.

`nodeType`	**Konstante**	**Beschreibung**
1	ELEMENT_NODE	Element
2	ATTRIBUTE_NODE	Attribut
3	TEXT_NODE	Text
8	COMMENT_NODE	Kommentar
9	DOCUMENT_NODE	Dokument

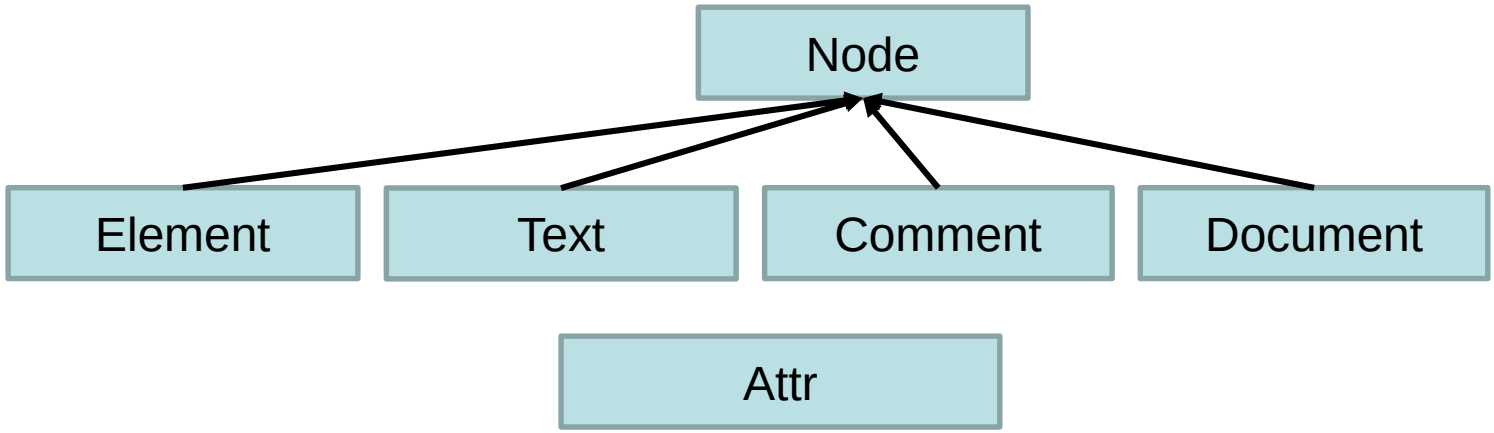
Der Knotentyp kann über die Eigenschaft **`nodeType`** abgefragt werden.

```
if (Knoten.nodeType == 8) alert(„Ich bin ein Kommentar“);
```

Baumstruktur II – Knotentypen

`nodeType`	**Konstante**	**Beschreibung**
1	ELEMENT_NODE	Element
2	ATTRIBUTE_NODE	Attribut
3	TEXT_NODE	Text
8	COMMENT_NODE	Kommentar
9	DOCUMENT_NODE	Dokument

DOM Interface-Hierarchy (entsprechende Hierarchie in JavaScript)



Baumstruktur III – ELEMENT_NODE

Definition: **ELEMENT_NODES sind alle HTML Tag-Elemente.**

ELEMENT_NODES haben Eigenschaften und Methoden mit Bezug auf ***Elemente***.

Wichtigste Eigenschaften:

`firstElementChild`	zeigt auf den ersten Element-Knoten der Struktur
`lastElementChild`	zeigt auf den letzten Element-Knoten der Struktur
`children[]`	Array ermöglicht Zugriff auf alle Kind-Element-Nodes
`parentElement`	zeigt auf den Eltern-Element-Knoten
`nodeName`	Namen des HTML-Elements
`nodeType`	hat den Wert 1 bzw. ELEMENT_NODE

Wichtigste Funktionen:

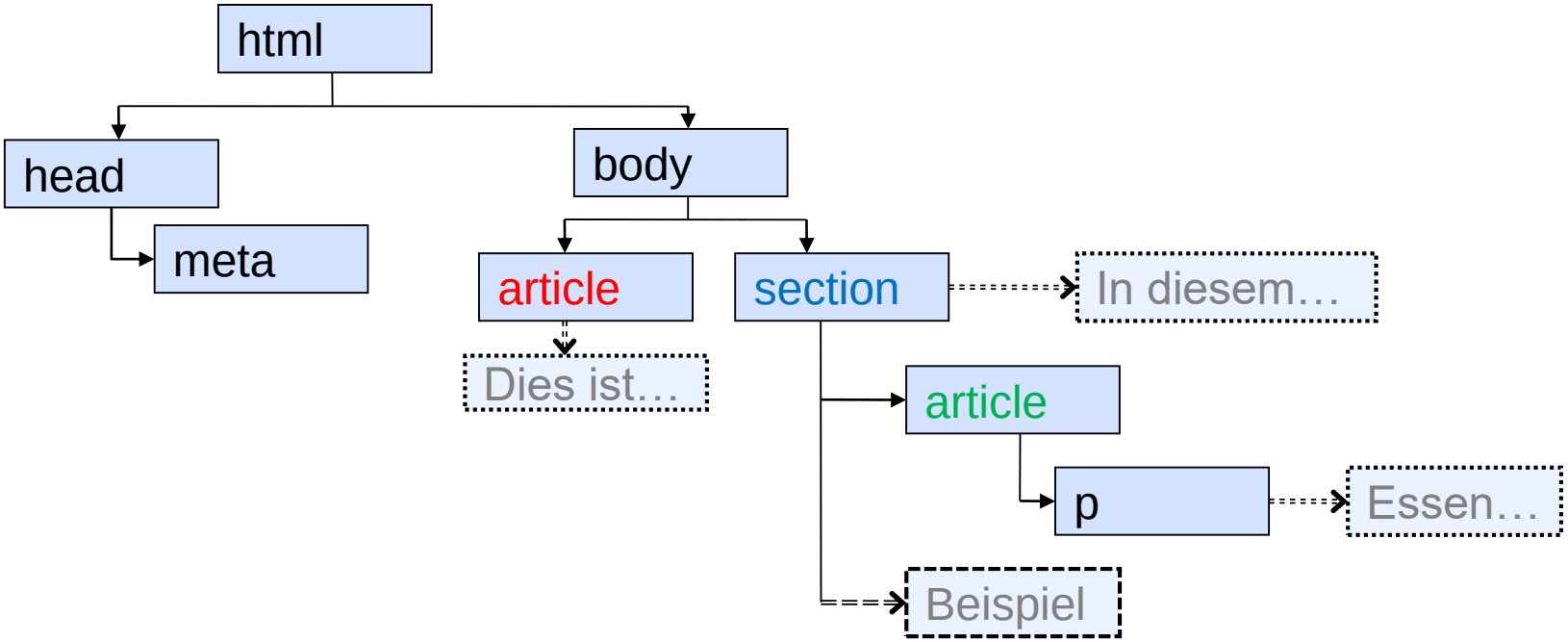
`click()`	Simuliert einen Klick auf das Element
`scrollIntoView()`	Scrollt die Seite so, dass das Element sichtbar ist

Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_all.asp

Baumstruktur III – Navigation (Elemente)

- Bei tiefer verschachtelten Knoten muss ggf. rekursiv durch alle Ebenen gegangen werden.

```
window.onload = function() {
    console.log(document.body.children[0].nodeName);
    console.log(document.body.children[1].nodeName);
    console.log(document.body.children[1].children[0].nodeName);
}
```



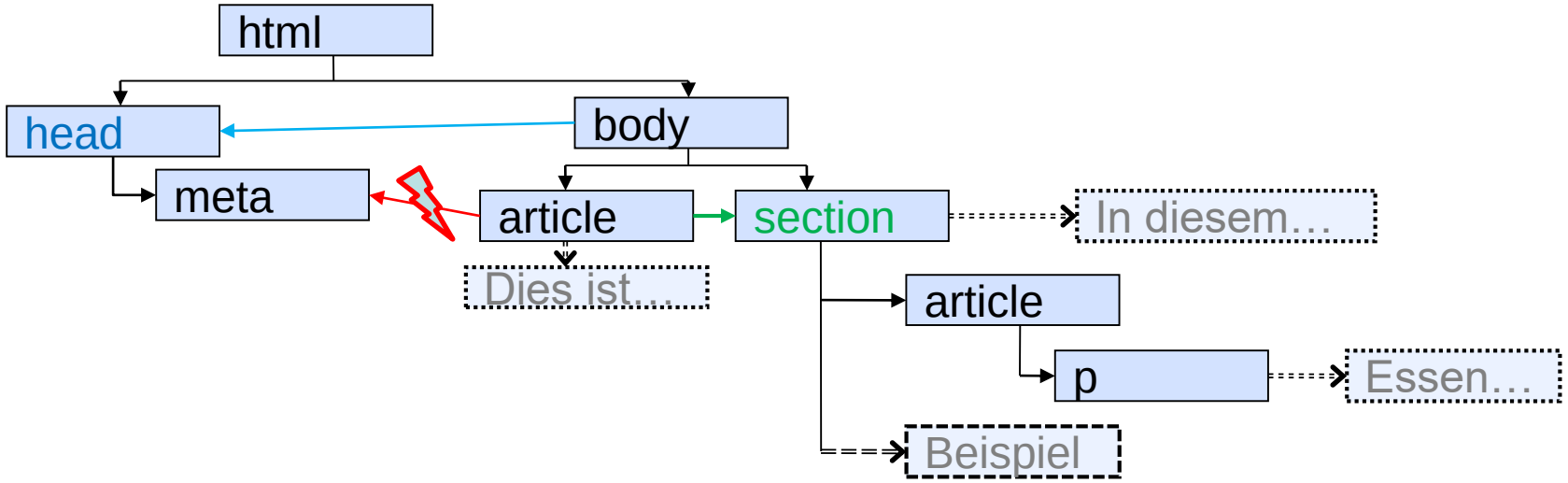
Baumstruktur III – Geschwister (Elemente)

Zugriff auf Geschwisterelemente:

`previousElementSibling`	Element links vom aufrufenden Element
`nextElementSibling`	Element rechts vom aufrufenden Element

```
window.onload = function() {

console.log(document.body.children[0].nextElementSibling.nodeName);
   console.log(document.body.previousElementSibling.nodeName);
   console.log(document.body.children[0].previousElementSibling);
}
```



Baumstruktur III – innerHTML (Elemente)

Das Attribut innerHTML ist ein spezielles Attribut von Elementen, das Zugriff auf das HTML innerhalb des Elements erlaubt.

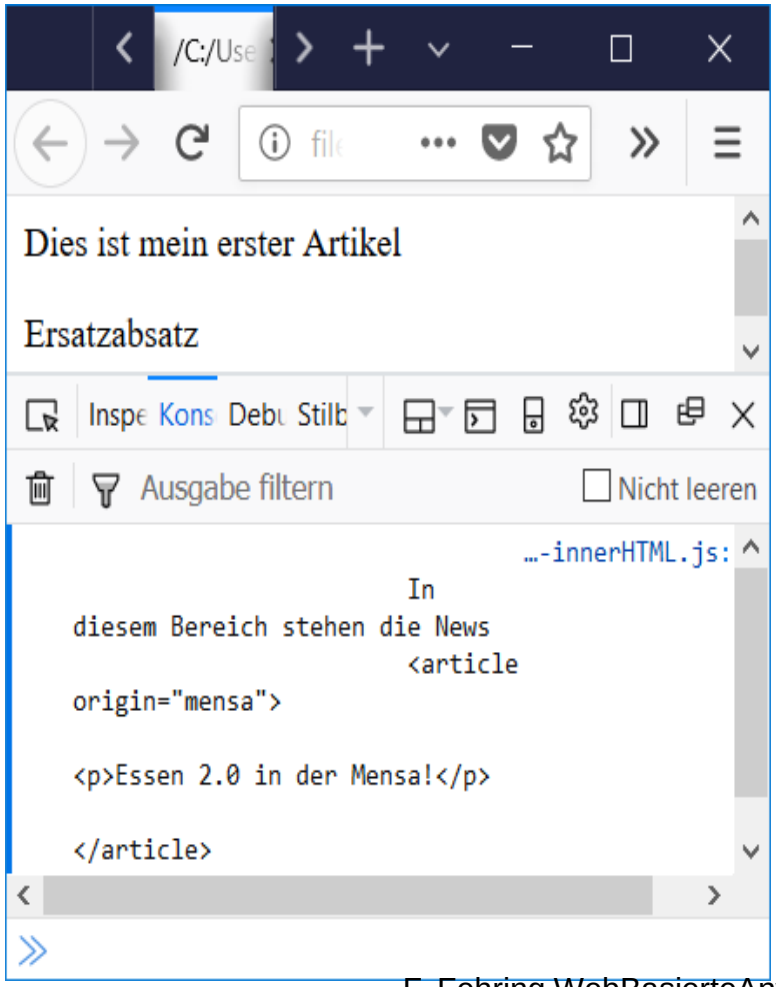
Eigenschaften:

- erlaubt Auslesen des Inhalts eines Element-Knotens als HTML (String)
- erlaubt das Setzen des Inhalts mit einem HTML-String
- parst das HTML beim Setzen und fügt es in die Baumstruktur als Nodes ein
- sofortige Darstellung der veränderten oder neuen Elemente
- ist aktuell nicht Bestandteil der DOM-Spezifikation
    - wurde von Microsoft mit MSIE 5.0 eingeführt, später vom Netscape Browser 6.0 übernommen und hat sich stark verbreitet

Baumstruktur III – innerHTML (Elemente)

```
console.log(document.body.children[1].innerHTML);            // Auslesen

document.body.children[1].innerHTML = "<p>Ersatzabsatz</p>";// Ersetzen
```




```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <script src="dom-innerHTML.js"></script>
   </head>
   <body>
      <article>
         Dies ist mein erster Artikel
      </article>
      <section>
         In diesem Bereich stehen die News
         <article origin="mensa">
            <p>Essen 2.0 in der Mensa!</p>
         </article>
      </section>
   </body>
</html>
```

Baumstruktur IV – NODE

Alle Arten von NODEs (ausgenommen ATTRIBUTE_NODES) haben Eigenschaften und Methoden mit Bezug auf andere ***Nodes***.

Wichtigste Eigenschaften:

`firstChild`	zeigt auf den ersten Knoten der Struktur
`lastChild`	zeigt auf den letzten Knoten der Struktur
`childNodes[]`	Array ermöglicht Zugriff auf alle Kind-Knoten
`parentNode`	zeigt auf den Eltern-Knoten
`nodeName`	Name des Knotens

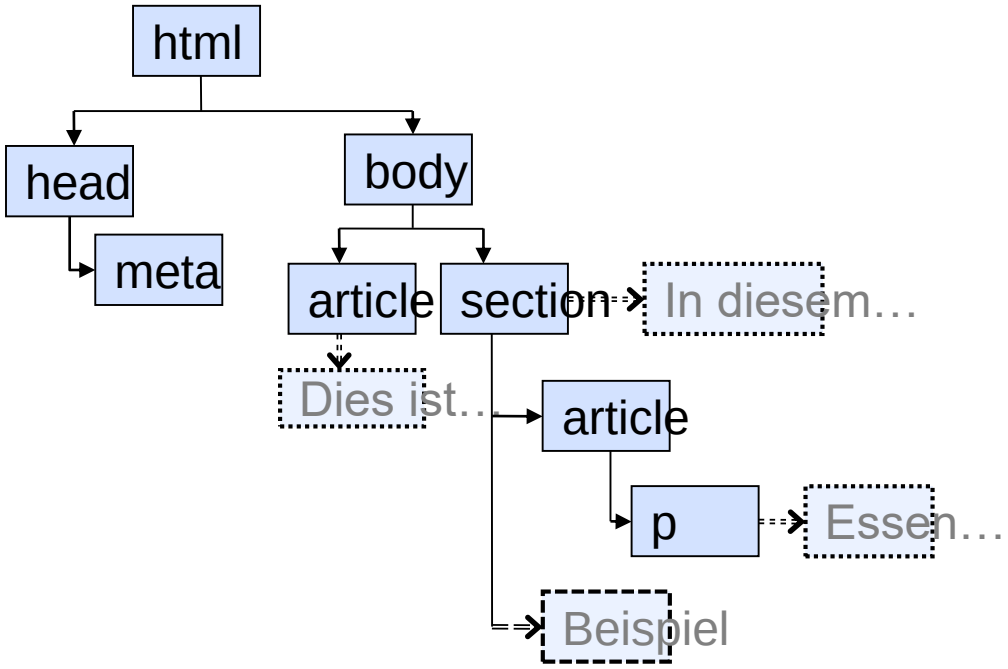
Wichtigste Methoden (für die Abfrage):

`contains()`	prüft, ob ein Knoten in diesem Konten vorkommt
`hasChildNodes()`	prüft, ob dieser Knoten Kinder hat
`isEqualNode(node)`	prüft, ob zwei Knoten gleich sind

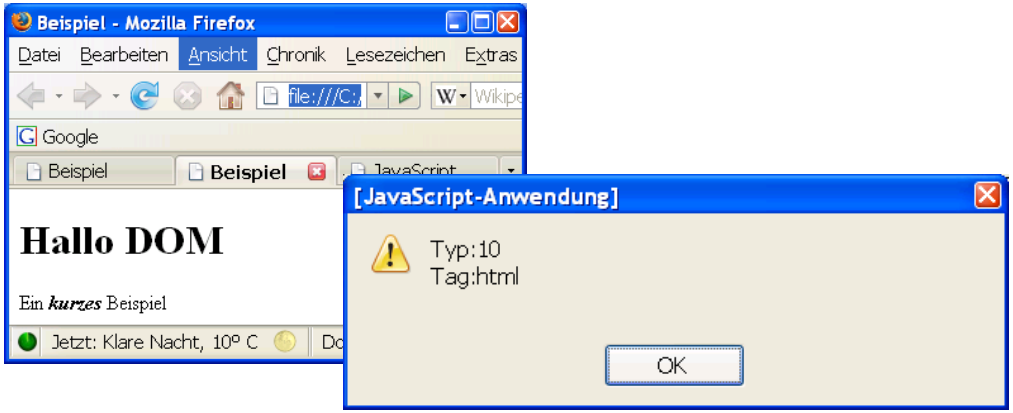
ACHTUNG: Bis DOM L3 hatten auch alle anderen Knotentypen alle Funktionen und Attribute von dieser Seite ab DOM L4 nicht mehr!

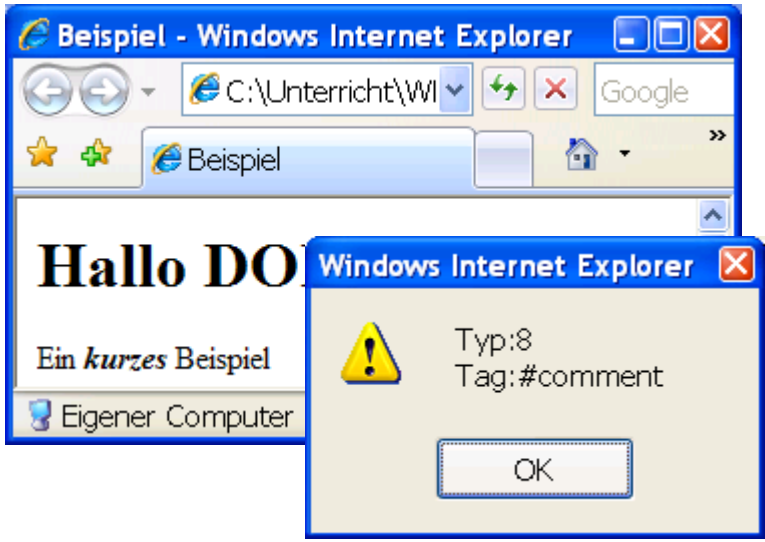
Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_all.asp

Baumstruktur IV – firstChild



- `document.firstChild` zeigt in einem korrekten HTML5 Dokument auf den Doctype
- In älteren HTML Varianten und Browsern oder bei weglassen des Doctypes kann es abweichende Interpretationen der Browser geben!

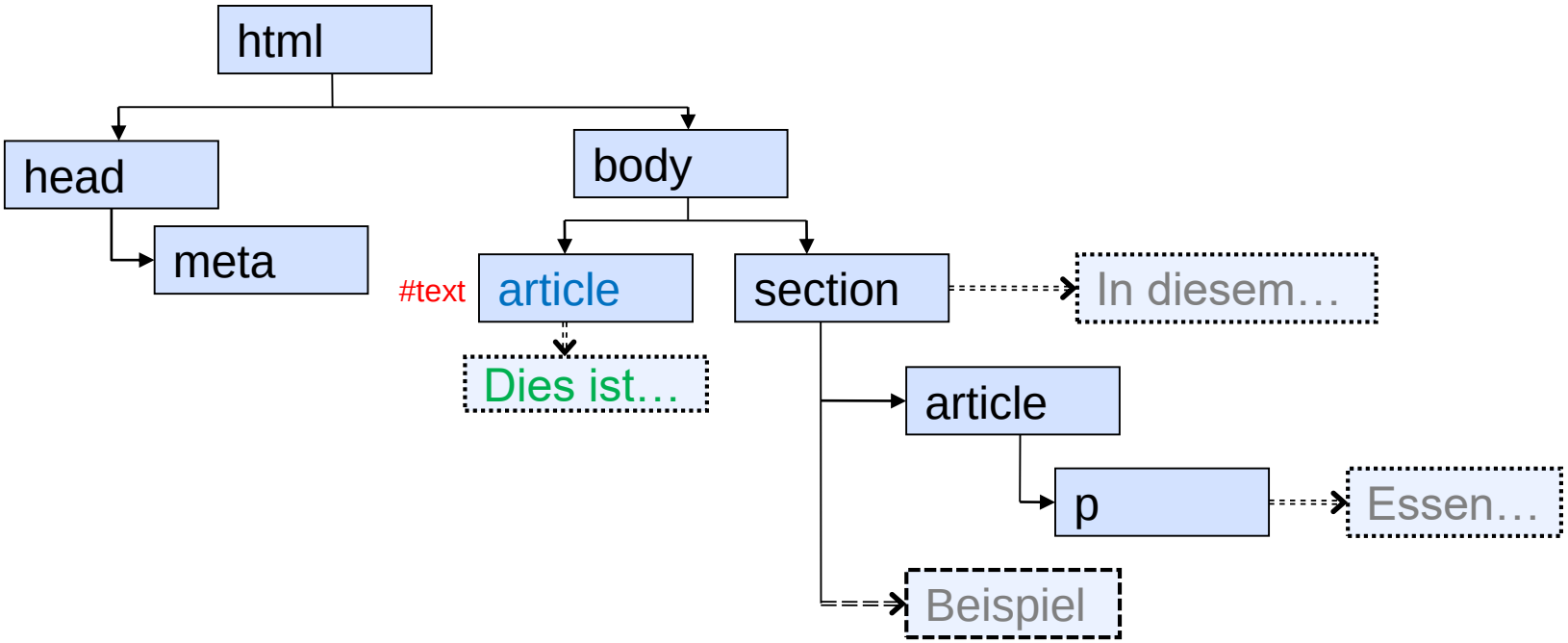




Baumstruktur IV – Navigation (Knoten)

Unterschied zwischen Navigation mit Elementen und Knoten: Navigation mit Knoten schließt andere Knotentypen mit ein. Auch implizit vorhandene Knoten, die im Code nicht sichtbar sind

```
window.onload = function() {
   console.log(document.body.childNodes[0].nodeName);
   console.log(document.body.childNodes[1].nodeName);
console.log(document.body.childNodes[1].childNodes[0].nodeName);
}
```

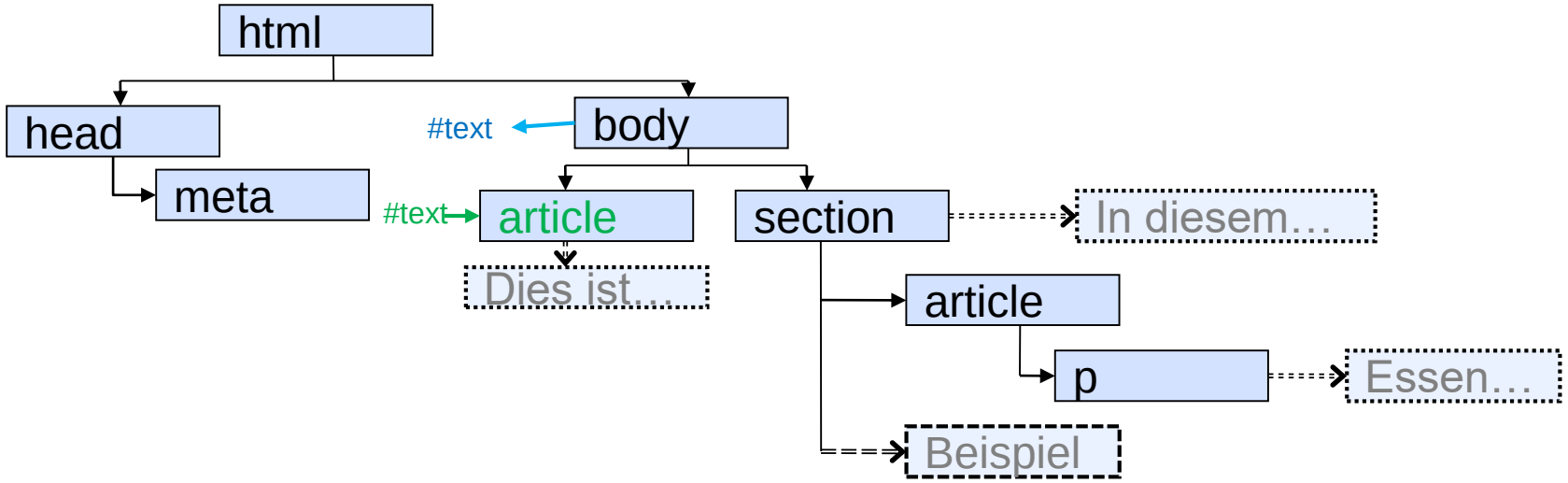


Baumstruktur IV – Geschwister (Knoten)

Zugriff auf Geschwister:

`previousSibling`	Knoten links vom aufrufenden Element
`nextSibling`	Knoten rechts vom aufrufenden Element

```
window.onload = function() {
    console.log(document.body.childNodes[0].nextSibling.nodeName);
    console.log(document.body.previousSibling.nodeName);
    console.log(document.body.childNodes[0].previousSibling);
}
```



Baumstruktur V – ATTRIBUTE_NODE

Definition: **ATTRIBUTE_NODES sind alle Attribute bei HTML-Tags. Sie sind NICHT Teil des Dokumentenbaums.**

Attribut-Knoten haben einen Namen und eine Wert. Sie besitzen keine Kinder.

Wichtigste Eigenschaften:

`name`	Name des Attributs (alternativ: `nodeName`)
`value`	Wert des Attributs (alternativ: `nodeValue`)
`isId`	Angabe, ob es sich um ein id-Attribut handelt

Die Attribute eines Elements sind über die Eigenschaft attributes zugreifbar. Dieses Attribut ist ein NamedNodeMap-Objekt.

Methoden des NamedNodeMap-Objekts:

`getNamedItem(name)`	Holt den Attribut-Knoten mit dem Namen aus der Map
`setNamedItem()`	Setz den Wert eines Attribut-Knotens, fügt ggf. hinzu
`removeNamedItem()`	Löscht einen Attribut-Knoten

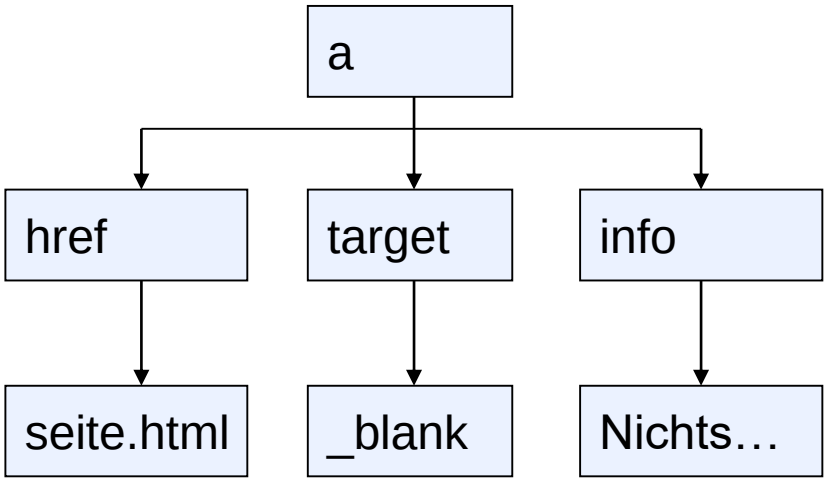
Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_attributes.asp

Baumstruktur V – ATTRIBUTE_NODE

HTML5 erlaubt das Hinzufügen beliebiger weiterer Attribute zu einem Dokument. Man muss jedoch darauf achten, keine von HTML5 definierten Attributnamen neu zu definieren.

```
<a href="seite.html" target="_blank" info="Nichts besonderes.">Eine Seite</a>
```

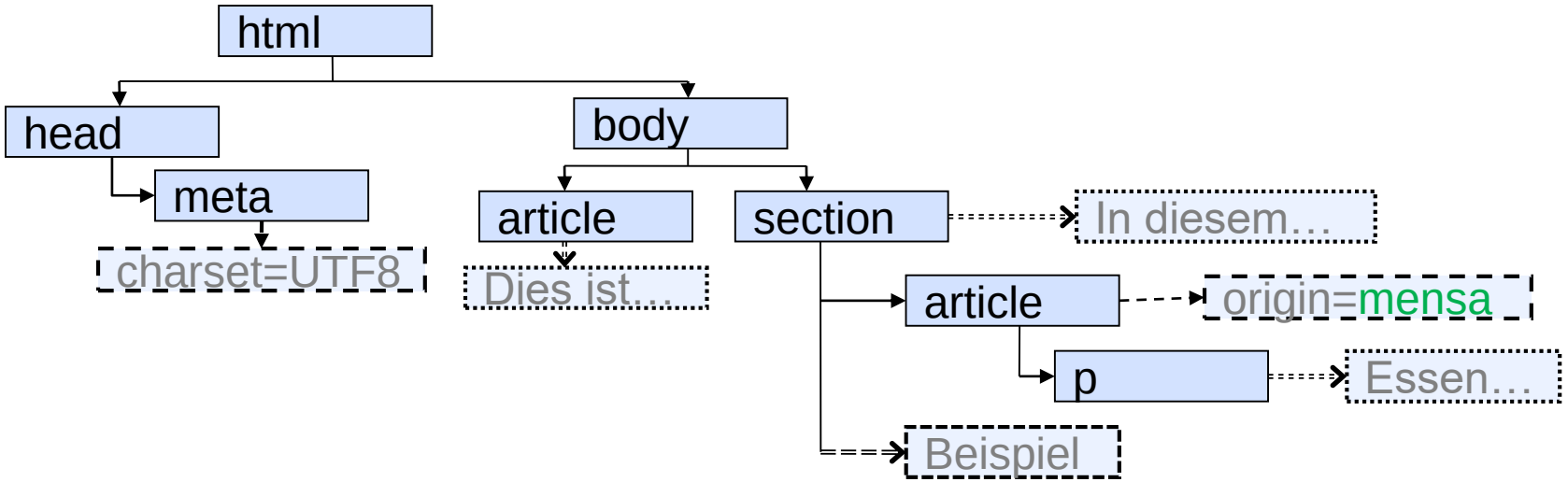
- `href` und `target` sind Attributelemente des übergeordneten Elementknotens `a`
- `info` ist ein zusätzliches Attribut



Baumstruktur V – Attribute

```
console.log(document.body.children[1].children[0]
        .attributes.getNamedItem("origin").value);
```

mensa



DOM

1. Kontext und Motivation
2. Allgemeines
3. Baumstruktur
4. **DOM Zugriffe**
5. DOM Manipulation
6. SpecialObjects
7. Event-Handling
8. Darüber hinaus
9. Projekt

DOM Zugriffe I – Übersicht

Das DOM definiert zahlreiche Methoden für den Zugriff auf Nodes. Außerdem ist der Zugriff über Attribute möglich.

Die Methoden können von jedem Node aus genutzt werden.

Wichtigste Methoden:

`querySelector()`	ein Element anhand eines CSS-Selectors
`querySelectorAll()`	alle Elemente anhand eines CSS-Selektors
`getElementById()`	ein Element anhand einer CSS-ID
`getElementsByName()`	alle Elemente, die als name-Attribute das Argument der Methode enthalten
`getElementsByTagName()`	alle Elemente als Array, die als Tagname das Argument der Methode enthalten

DOM Zugriffe I – querySelector()

```
node = querySelector(cssSelectorString)
Node[] = querySelectorAll(cssSelectorString)
```

`cssSelectorString` Ein gültiger CSS-Selektor

```
let allarts = document.querySelectorAll("article");
for(var art of allarts) {
        console.log(art.innerHTML);
}

console.log(document.querySelector("article[origin=mensa]").innerHTML);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

Frage: Was gibt das Programm aus?

Dies ist mein erster Artikel
Essen 2.0 in der Mensa!
Essen 2.0 in der Mensa!

DOM Zugriffe II – getElementById()

```
node = getElementById(cssId)
```

cssId Eine eindeutige CSS-ID

```
console.log(document.getElementById("art1").innerHTML);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

Frage: Was gibt das Programm aus?

Dies ist mein erster Artikel

DOM Zugriffe II – Einzelne Zugriffe

Es gibt eine Reihe von Eigenschaften und Methoden für den Zugriff und die Anwendung auf einzelne Elemente des Dokumentenbaumes.

Eigenschaften des document-Objekts:

`defaultView`	Standardansicht des aktuellen Dokuments, welche in Webbrowsern durch das **`window`**–Objekt repräsentiert wird
`Doctype`	zeigt den Inhalt einer **`<!DOCTYPE …>`** - Deklaration an, falls vorhanden ist, sonst **`null`**
`styleSheets`	liefert ein Array mit Referenzen auf StyleSheets, die mittels des **`link-`** oder **`style-`**Tags eingebunden sind.

DOM

1. Kontext und Motivation
2. Allgemeines
3. Baumstruktur
4. DOM Zugriffe
5. **DOM Manipulation**
6. SpecialObjects
7. Event-Handling
8. Darüber hinaus
9. Projekt

DOM Manipulation I – NODE

Alle Arten von NODEs (ausgenommen ATTRIBUTE_NODES) haben Methoden zu ihrer Manipulation ***Nodes***.

Wichtigste Funktionen:

`appendChild(node)`	fügt Knoten an das Ende des Knotens ein
`cloneNode()`	kopiert den Knoten
`insertBefore(node)`	fügt einen Knoten vor dem Knoten ein
`removeChild()`	entfernt das Element
`replaceChild(node)`	Ersetzt das Element durch ein anderes

ACHTUNG: Bis DOM L3 hatten auch alle anderen Knotentypen alle Methoden von dieser Seite ab DOM L4 nicht mehr!

Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_all.asp

DOM Manipulation II – appendChild()

```
node1.appendChild(node2)
```

node2 Ein Node-kompatibles Objekt (Element, Text,…)

```
let firstArt = document.querySelector("article");
document.getElementById("art2").appendChild(firstArt);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

Frage: Was gibt das Programm aus?

In diesem Bereich stehen die News

Essen 2.0 in der Mensa!

Dies ist mein erster Artikel

DOM Manipulation III – cloneNode()

```
node.cloneNode(boolean copychildren)
```

copychildren gibt an, ob die Kindknoten mit kopiert werden sollen

```
let cloneArt = document.getElementById("art1").cloneNode(true);
document.querySelector("section").appendChild(cloneArt);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

Frage: Was gibt das Programm aus?

Dies ist mein erster Artikel
In diesem Bereich stehen die News

Essen 2.0 in der Mensa!

Dies ist mein erster Artikel

DOM Manipulation IV – removeChild()

```
node1.removeChild(node2)
```

node2 Knoten der entfernt werden soll

```
let toDelete = document.getElementById("art1");
toDelete.parentNode.removeChild(toDelete);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

In diesem Bereich stehen die News

Essen 2.0 in der Mensa!

DOM Manipulation V – replaceChild()

```
node1.replaceChild(replacementNode,originalNode)
```

replacementNode	Knoten der den anderen ersetzen soll
originalNode	zu ersetzender Knoten

```
let original = document.getElementById("art1");
let forReplace = original.cloneNode();
forReplace.innerHTML = "Ich bin der Ersatz";
original.parentNode.replaceChild(forReplace,original);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

In diesem Bereich stehen die News

Essen 2.0 in der Mensa!

DOM Manipulation VI – Attribute

ATTRIBUTE_NODES werden über Methoden der Elemente, denen sie zugeordnet sind, bearbeitet.

Attribute können über die Eigenschaft attributes eines Elements abgerufen werden. Es gibt aber auch Funktionen am Element selbst.

Wichtigste Funktionen:

`getAttribute()`	liefert Wert bzw. Inhalt eines Attributes
`setAttribute()`	fügt neues Attribute mit gewünschtem Wert ein (od. ändert)
`removeAttribute()`	löscht einen Attributknoten aus einem Element
`hasAttribute()`	prüft, ob ein Attribut angelegt ist

Weitere Informationen: https://www.w3.org/TR/dom/#attr

DOM Manipulation VI – Attribute

```
let artNodes = document.querySelectorAll("article");
console.log(artNodes[0].getAttributeNode("id"));
artNodes[0].setAttribute("origin","general");
console.log(artNodes[1].hasAttribute("origin"));
artNodes[1].removeAttribute("origin");
console.log(artNodes[1].hasAttribute("origin"));
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

```
<body>
   <article id="art1" origin="general">
         Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

```
id="art1"
true
false
```

DOM Manipulation VII – Knoten erzeugen

Ein wesentliches Merkmal des DOM ist es, neue Elemente dynamisch zu erzeugen und an eine beliebige Position im Dokumentenbaum einhängen zu können.

Methoden:

`createElement()`	erzeugt einen neuen html-Elementknoten
`createTextNode()`	erzeugt einen neuen Textknoten

Hinweis:

~~`createAttribute()`~~	~~erzeugt einen neuen Attributknoten~~

Attribute werden in DOM L4 nur noch über die setAttribute()-Methode des Element-Objekts erzeugt.

DOM Manipulation VII – Knoten erzeugen

```
let myNewArticle = document.createElement("article");
myNewArticle.setAttribute("origin","newscenter");
let myNewText = document.createTextNode("Dies ist ein neuer
Artikel");
myNewArticle.appendChild(myNewText);

let firstArticle = document.querySelector("section article");
firstArticle.parentNode.insertBefore(myNewArticle,firstArticle);
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

```
<body>
   <article id="art1">
      Dies ist mein erster Artikel
   </article>
   <section>
      In diesem Bereich stehen die News
      <article origin="newscenter">
         Dies ist ein neuer Artikel
      </article>
      <article id="art2" origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

Dies ist mein erster Artikel
In diesem Bereich stehen die News
Dies ist ein neuer Artikel

Essen 2.0 in der Mensa!

DOM Manipulation VIII – Klassen

```
DOMTokenList = elementNode.classList
```

Die Eigenschaft classList eines Elements enthält ein spezielles Listen-Objekt, welches alle Klassen enthält, die einem Element zugeordnet wurden.

Eigenschaften und Methoden:

`length`	Anzahl der Klassen des Elements
`item(index)`	holt die i-te Klassenangabe
`contains(name)`	prüft, ob eine Klasse mit dem Namen gesetzt ist
`add(name)`	fügt die Klasse mit dem Namen hinzu
`remove(name)`	entfernt die Klasse mit den Namen
`toggle(name)`	entfernt oder fügt ein

DOM Manipulation VIII – Klassen

```
<head>
   <meta charset="UTF-8">
   <link rel="stylesheet" href="dom-classList.css">
   <script src="dom-classList.js"></script>
</head>
<body>
   <article id="b" class="x y z">
      Dies ist mein erster Artikel
   </article>
   <section class="a">
      In diesem Bereich stehen die News
      <article class="a">
         <p>Jetzt Essen 2.0 in der Mensa!</p>
      </article>
   </section>
</body>
```

```
let art1 = document.querySelector("#b");
for(let i of art1.classList) {
        console.log(i);
}

art1.classList.remove("x");
art1.classList.add("zz");
art1.classList.toggle("y");

console.log("Nach Manipulation:");
for(let i of art1.classList) {
        console.log(i);
}
```

```
x
y
z
Nach Manipulation:
z
zz
```

DOM Manipulation VIII – CSS

```
elementNode.style
window.getComputedStyle(element)
```

elementNode.style
Obwohl es möglich ist, CSS Angaben über das spezielle Attribut style eines Elements zu setzen, sollte von dieser Möglichkeit kein Gebrauch mehr gemacht werden. Besser ist es, das Layout komplett in einer css-Datei zu definieren und entsprechend Klassen an den Elementen zu aktivieren oder zu deaktivieren. Auch zum Auslesen von CSS Werten eignet sich das style-Attribut nicht. Es beinhaltet nur die Werte, die aus dem Dokument selbst kommen.

getComputedStyle
Gibt für das angegebene Element den aktuellen, berechneten Stil aus. Somit auch die angewendeten Stile aus css-Dateien. Die Eigenschaften dieses Objekts sind identisch mit denen es style-Attributs.

Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_style.asp

DOM Manipulation VIII – CSS

```
#b {
   color: blue;
}
```

```
window.onload = function() {
   let art1 = document.querySelector("#b");
   console.log("Stil: " + art1.style.color);
   console.log("Computed stile: " +
window.getComputedStyle(art1).color);
}
```

Stil:
Computed stile: rgb(0, 0, 255)

DOM

1. Kontext und Motivation
2. Allgemeines
3. Baumstruktur
4. DOM Zugriffe
5. DOM Manipulation
6. **SpecialObjects**
7. Event-Handling
8. Darüber hinaus
9. Projekt

SpecialObjects I

Neben den Knoten-bildenden Objekten definiert das DOM als API spezielle Objekte, die einen Zugriff erleichtern sollen. Außerdem können Elemente aus HTML Elementknoten sein, aber auch eigene Methoden mitbringen (API-Elemente).

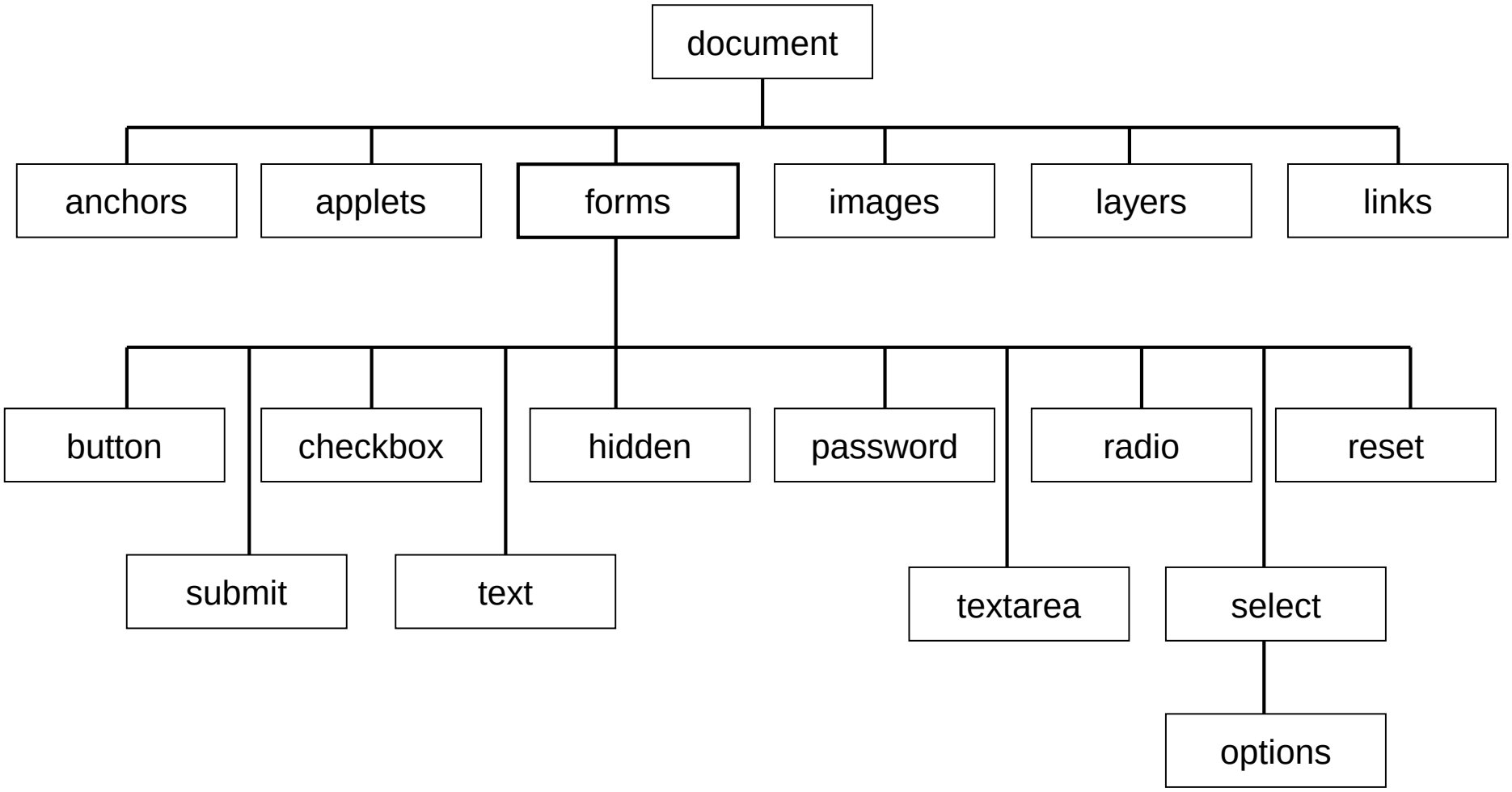
Elementsammlungen:

- sind durch das DOM spezifiziert
- bieten einfacheren Zugriff, geordnet nach Element-Art
- sind in Form von Attributen am document-Objekt realisiert
- liefern ihre Elemente als Arrays

API-Elemente:

- sind nicht durch das DOM spezifiziert

SpecialObjects I - Elementsammlungen



SpecialObjects II - Canvas

Definition: **Das Canvas-Element ermöglicht die Darstellung einer graphischen Zeichenfläche. Auf diese kann mit JavaScript-Methoden gezeichnet werden.**

`<canvas>:`

`width`	Breite der Zeichenfläche
`Height`	Höhe der Zeichenfläche

Methoden und Eigenschaften:

`getContext('2d')`	den 2D-Zeichenkontext vom Element holen
`createTextNode()`	erzeugt einen neuen Textknoten
`fillStyle`	Style für alle nachfolgend gezeichneten Füllungen
`fillRect()`	Ein gefülltes Rechteck zeichnen
`font`	Eigenschaften für eine zu zeichnenden Text
`fillText()`	Eine Text zeichnen
`drawImage()`	Ein Bild auf die Zeichenfläche malen

Weitere Informationen:
https://developer.mozilla.org/de/docs/Web/Guide/HTML/Canvas_Tutorial

SpecialObjects II - Canvas

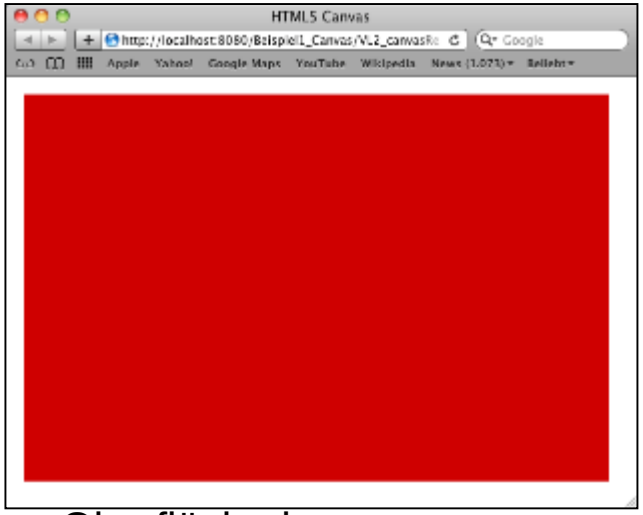
```
<!DOCTYPE html>
<html>
   <head>
         <meta charset="UTF-8">
         <script src="canvas.js"></script>
   </head>
   <body>
         <canvas id="stage" width="600" height="400"></canvas>
   </body>
</html>
```

- erzeugt eine leere graphische Oberfläche im Browser mit Breite 600 Pixel und Höhe 400 Pixel
- Zugriff über id (z.B. „stage“)

SpecialObjects II – Canvas - Rechteck

```
<!DOCTYPE html>
<html>
   <head>
         <meta charset="UTF-8">
         <script src="canvas.js"></script>
   </head>
   <body>
         <canvas id="stage" width="600" height="400"></canvas>
   </body>
</html>
```

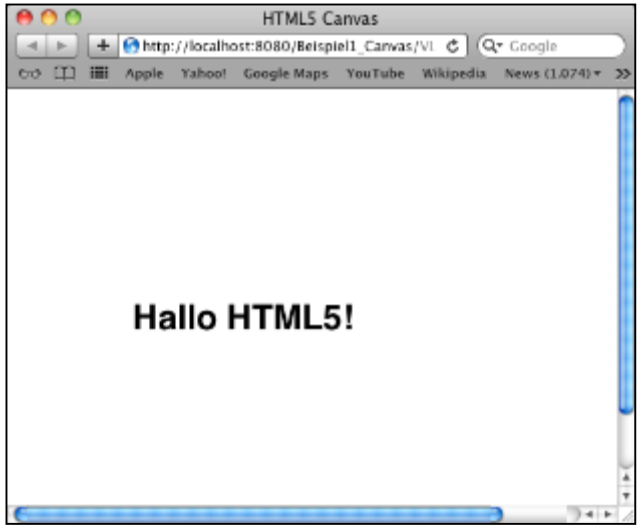
```
function zeichne(){
  var canvas = document.getElementById("stage");
  if (canvas.getContext){
      var stageContext = canvas.getContext('2d');
      stageContext.fillStyle = "rgb(200,0,0)";
      stageContext.fillRect (10, 10, 590, 390);
  }
}
```



– erzeugt ein rotes ausgefülltes Rechteck in der graphischen Oberfläche im Browser mit Breite 580 Pixel und Höhe 380 Pixel (Rand zur „stage“ je 10 Pixel)

SpecialObjects II – Canvas - Text

```
function schreibe(){
   var canvas = document.getElementById("stage");
   if (canvas.getContext){
      var stageContext = canvas.getContext('2d');
      stageContext.font="bold 30 ox sans-serif";
      stageContext.fillText("Hallo HTML5!",100, 200);
   }
}
```



SpecialObjects II – Canvas - Bildausgabe

```
function male(){
   var canvas = document.getElementById("stage");
   if (canvas.getContext){
      var stageContext = canvas.getContext('2d');
         var imgBuffer = new Image();
         imgBuffer.src = "canvas.jpg";
         imgBuffer.onload = function(){
         stageContext.drawImage(imgBuffer,50,50,180,120);
      }
   }
}
```

DOM

1. Kontext und Motivation
2. Allgemeines
3. Baumstruktur
4. DOM Zugriffe
5. DOM Manipulation
6. SpecialObjects
7. **Event-Handling**
8. Darüber hinaus
9. Projekt

Event-Handling

Definition: **Das Event-Handling im DOM erlaubt das registrieren von Listenern auf bestimmte Ereignisse. Diese Listener sind JavaScript-Funktionen.**

Eigenschaften:

- ermöglichen das Abfangen von Ereignissen (Klick, mouseover,…)
- ermöglichen das Auswerten von Mausbewegungen
- ermöglichen somit client-seitige Validierung von Eingaben
- es gibt zahlreiche Events und dazugehörige Event-Objekte
- es können eigene Events definiert werden

Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_event.asp

Event-Handling II – Event-Phasen

Definition: **Das Event-Handling im DOM läuft in zwei Phasen ab, der capture-Phase und der Bubble-Phase.**

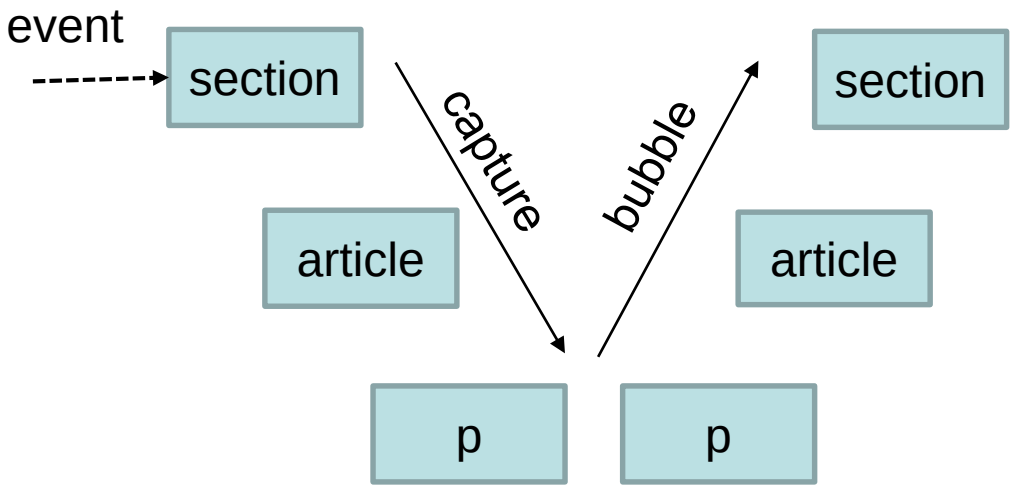
Ablauf:

1. Ein Event wird ausgelöst
2. Capture-Phase beginnt
    1. Jedes Element in der Hierarchie hat die Möglichkeit das Event abzufangen
3. Deepest-Point
    1. Das unterste Element in der Hierarchie kann auf das Ereignis reagieren
4. Bubble-Phase
    1. Jedes Element in der Hierarchie hat die Möglichkeit zusätzliche Aktionen zu starten

Welches Element in welcher Phase reagiert (wenn überhaupt) wird durch den Event-Handler festgelegt.

Event-Handling II – Event-Phasen

```
<section>
   In diesem Bereich stehen die News
   <article origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
   </article>
</section>
```



Event-Handling III - Implementierung

```
node.addEventListener(type,listener,capture)
node.removeEventListener(type,listener,capture)
```

addEventHandler:

`type`	Name des Event-Typs, der behandelt werden soll
`listener`	Funktion, die beim Eintritt ausgeführt werden soll
`capture`	boolean, ob der Handler in der Capture-Phase (true) oder in der Bubble-Phase (false) ausgeführt werden soll

removeEventHandler: Entfernt einen EventHandler wieder

Weitere Informationen: https://www.w3schools.com/jsref/dom_obj_event.asp

Event-Handling III - Implementierung

```
function myEventHandler(evt) { … }
```

Die aufgerufene Event-Handler-Funktion bekommt automatisch ein Event-Objekt. Je nach aufgetretenem Event hat dieses Event-Objekt unterschiedliche Attribute.

Allgemeine Eigenschaften:

`type`	Name des Event-Typs, der aufgetreten ist
`target`	Referenz auf das (unterste) Objekt, welches geklickt wurde
`currentTarget`	Referenz auf das Objekt, das gerade das Ereignis fing
`eventPhase`	Gibt an, in welcher Phase sich das Event befindet
`timestamp`	Genauer Zeitpunkt zu dem das Ereignis auftrat

Allgemeine Methoden:

`preventDefault()`	Die Standardaktion (des Browsers) für das Ereignis wird nicht ausgeführt
`stopPropagation()`	Der Durchlauf durch die Phasen wird abgebrochen

Weitere Informationen: https://www.w3.org/TR/dom/#concept-event

Event-Handling III - Implementierung

```
<section>
   In diesem Bereich stehen die News
   <article origin="mensa">
         <p>Essen 2.0 in der Mensa!</p>
   </article>
</section>
```

```
let capture = function(evt) {
   console.log('captured on ' + evt.currentTarget.nodeName);
}

let bubble = function(evt) {
   console.log("bubble on " + evt.currentTarget.nodeName);
}

window.onload = function() {
   document.querySelector("section").addEventListener("click",capture,true);
   document.querySelector("section article").addEventListener("click",capture,true);
   document.querySelector("section article p").addEventListener("click",capture,true
   document.querySelector("section").addEventListener("click„,bubble,false);
   document.querySelector("section article").addEventListener("click",bubble,false);
   document.querySelector("section article p").addEventListener("click",bubble,false
}
```

captured on SECTION
captured on ARTICLE
captured on P (2)
captured on ARTICLE
captured on SECTION

Event-Handling IV - Ereignisse I

Ereignis	Auslösegrund	HTML-Tags mit dem Ereignis (Beispiele)
click	Klicken eines Elements mit der Maus	*<a> <address> <area> <b> <body> <button> <col> <form> <h1> <h2> … <i> <img> <input> <label> <link> <option> <p> <select> <strike> <table> <textarea> …*
abort	Abbruch des Ladens einer Webseite	*<img>*
blur	Verlassen eines Elements	*<a> <area> <button> <input> <label> <select> <textarea>*
change	Änderungen von Angaben	*<input> <select> <textarea>*
dblclick	Doppeltes Anklicken	fast alle HTML- Tags (wie *onClick*)
error	Fehlerfall (z.B. falsche Bildangabe)	*<img>*
focus	Aktivieren eines selektierbaren Elements	*<a> <area> <button> <input> <label> <select> <textarea>*

Event-Handling IV - Ereignisse II

Ereignis	Auslösegrund	HTML-Tags mit dem Ereignis (Beispiele)
keysdown	Betätigen einer Taste	fast alle HTML- Tags
keypress	Betätigen einer Taste	fast alle HTML- Tags
keyup	Loslassen einer Taste	fast alle HTML- Tags
load	Laden einer Webseite	*<frameset> <body>*
mousedown	Betätigen der Maustaste	fast alle HTML-Tags
mouseout	Verlassen eines Elements mit der Maustaste	fast alle HTML-Tags
mouseover	Überfahren eines Elements mit der Maus	fast alle HTML-Tags
mouseup	Loslassen der Mausetaste	fast alle HTML-Tags
reset	Zurücksetzten eines Formulars	*<form>*
select	Selektieren von Text in Eingabefeldern	*<input> <textarea>*
submit	Absenden von Formulardaten	*<form>*
unload	Verlassen einer Webseite	<frameset> <body>

Event-Handling IV - *submit*

```
Dein Kommentar:
<form id="f" action="">
   <input type="Text" name="Feld"> <br> <br>
   <input type="submit" name="Testsubmit" value="Abgeben">
</form>
```

```
let checkForm = function(evt) {
   let feldvalue =
document.getElementsByName("Feld")[0].value;
   if(feldvalue == "") {
         alert("Bitte eine Eingabe!");
   } else {
         alert("Danke für Ihre Eingabe!");
   }
}

window.onload = function() {
   let form = document.getElementById("f");
   form.addEventListener("submit",checkForm);
}
```

Event-Handling IV - *keypress*

```
let shortCutHandler = function(evt) {
   console.log("Sie haben " + evt.key + " gedrückt.");
   if(evt.ctrlKey) {
         console.log("Sie haben auch den ctrl-Key gedrückt!");
   }
}

window.onload = function() {
   let form = document.addEventListener("keypress",shortCutHandler);
}
```

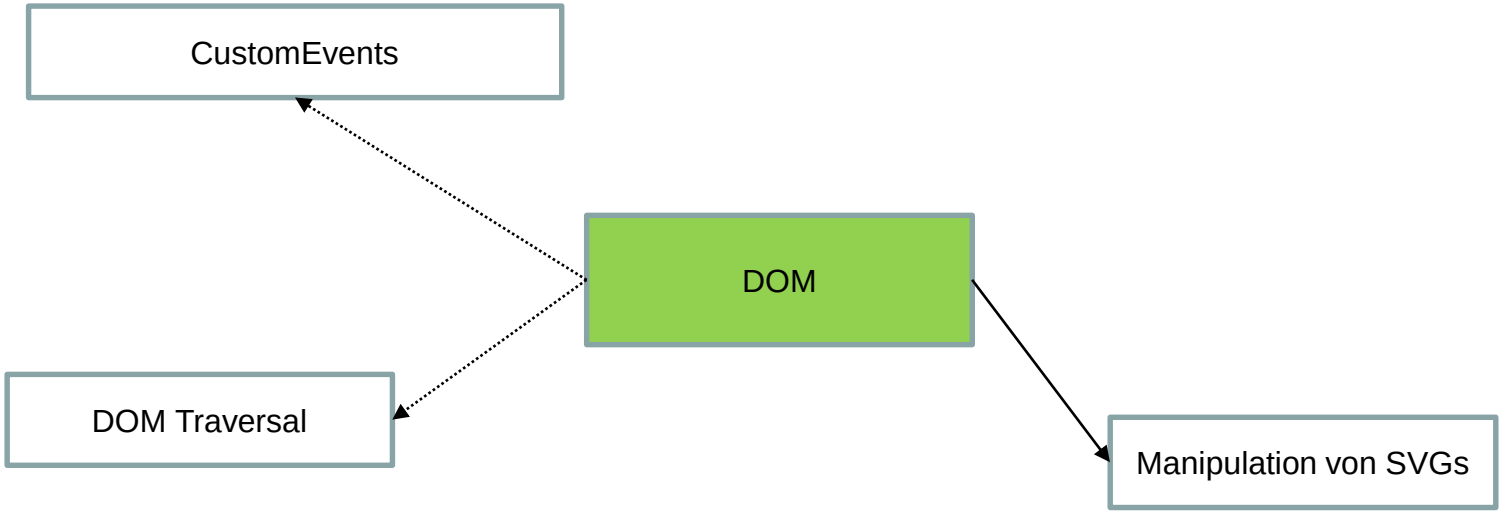
Beim drücken von ctrl + k:

Sie haben k gedrückt.
Sie haben auch den ctrl-Key gedrückt!

DOM

1. Kontext und Motivation
2. Allgemeines
3. Baumstruktur
4. DOM Zugriffe
5. DOM Manipulation
6. SpecialObjects
7. Event-Handling
8. **Darüber hinaus**
9. Projekt

Darüber hinaus



Links:

CustomEvents - https://weblogs.asp.net/mikaelsoderstrom/customevent-in-dom-level-3-and-dom-level-4

DOM Traversal - https://www.w3.org/TR/dom/#traversal

DOM

1. Kontext und Motivation
2. Allgemeines
3. Baumstruktur
4. DOM Zugriffe
5. DOM Manipulation
6. SpecialObjects
7. Event-Handling
8. Darüber hinaus
9. **Projekt**

Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel
- Externe Inhalte einbinden
- Medien hochladen / runterladen
- Kommentare hochladen / runterladen
- Kommentare speichern
- Kommunikation untereinander

DONE

- Technologische Grundlagen erarbeiten
- Was ist eine Web-Anwendung?
- News darstellen
- Projekte vorstellen
- Aufgaben darstellen
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendatein einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen