Verteilte Systeme 1

Technologien des World Wide Web

christian.zirpins@hs-karlsruhe.de

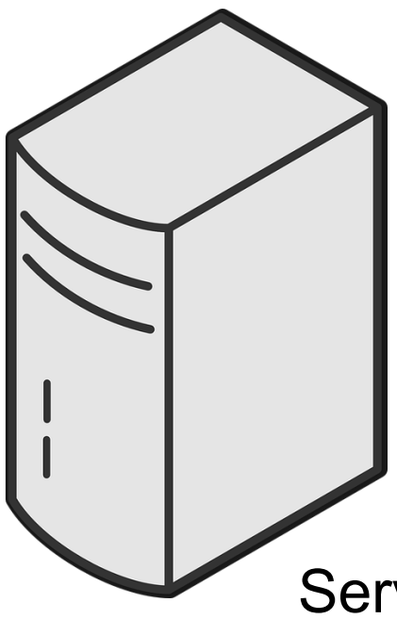
Browser Interaktion mit JavaScript

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

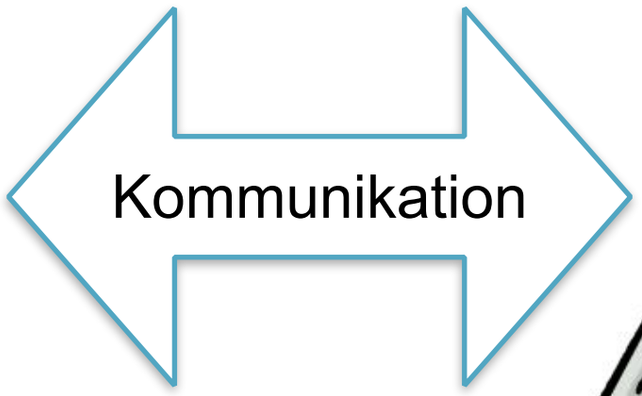
Lernziele

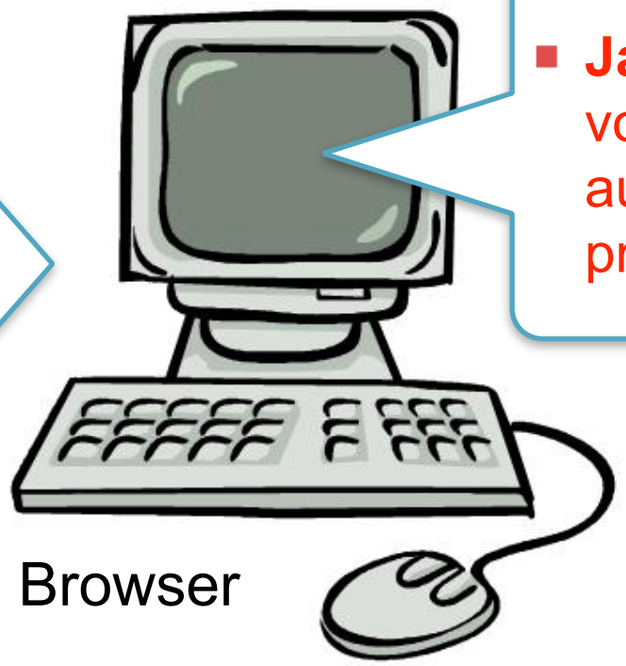
Nach dieser Vorlesung können Sie…

- … interaktive Web-Anwendungen mit **JavaScript** schreiben
- … jQuery-basierten Code in **jQuery-losen Code** übersetzen
- … das Prinzip der **Callbacks** erklären



Server





Browser

- **HTML**: zeichnet Inhalte aus
- **CSS**: steuert Erscheinungsbild
- **JavaScript**: manipuliert Inhalte von HTML Dokumenten, reagiert auf Benutzerinteraktion, programmiert Plattform APIs

JavaScript im Browser

Sandboxing

- JavaScript wird in HTML Dokumenten eingebettet oder referenziert.
  - Code läuft auf dem Client Rechner, der das Dokument geladen hat.
  - Damit dieser Code nichts Schädliches tut, gibt es Einschränkungen.
- JavaScript im Browser wird in einer **Sandbox** ausgeführt, d.h.:
  - kein Zugriff auf Betriebssystemfunktionen (z.B. Dateizugriff),
  - Beschränkung auf die Browser APIs,
  - Zugriff nur auf die Elemente der eigenen Seite (Same Origin Policy).

Client-Side JavaScript

- Core JavaScript + Browser APIs

Erinnerung: Einbetten vs. Referenzieren

- JavaScript lässt sich in ein HTML-Dokument entweder direkt einbetten oder referenzieren. Dazu wird das **Script-Tag** verwendet.
- JavaScript-Dokumente werden in der Reihenfolge abgearbeitet, in der sie verwendet werden.
- Fehler in einem Script Tag bleiben lokal und betreffen keine anderen Script Tags.
  - Mit separaten Tags können Skripte isoliert werden.
- `<Script>` erzeugt Instanz der JavaScript Engine und parst das Skript. Das Page Rendering unterbricht.
  - Durch viele Script Tags werden Seiten träge.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello</title>
</head>
<body>
  <script src="/javascripts/hello.js"></script>
  <script>
    alert("Ich bin ein eingebettetes Skript.");
  </script>
</body>
</html>
```

Datei "hello.html"

```
alert("Ich bin ein referenziertes Skript.");
```

Datei "hello.js"

Das `Script` Tag

*"**Wir setzen `<script>` Tags in das Body-Element**: der Browser zeigt die Seite in einer Top-down-Weise an und erzeugt DOM-Elemente, wenn er darüber läuft.*

*Durch **Platzieren der <script> -Tags am Ende**, sind die JavaScript-Dateien eines der letzten Dinge, die die Seite lädt.*

*Da JavaScript-Dateien oft Zeit zum Laden brauchen, tun wir dies als letztes, so dass Benutzer von anderen Elementen **so schnell wie möglich visuelles Feedback** erhalten." (Semmy Purewal)*

Das Window Objekt

- Jedes Browserfenster hat ein **`window`-Objekt** (*Tabs* haben unterschiedliche `window`-Objekte).
- Das `window`-Objekt ist die Wurzel des globalen Namensraums.
  - Eine neue globale Variable wird Teil des **`window`**-Objekts

```
<script>
  myString = "Hello!";
  alert(window.myString); // "Hello!"
</script>
```

- Eigenschaften und Methoden des `window`-Objekts können ohne explizite Referenz zugegriffen werden.
  - Man kann auf **`window.myString`** oder auf **`myString`** zugreifen.

Das Document Objekt

- Der wichtigste Member des `window`-Objekts ist `document`.
  - Hierüber ist das Dokument zugreifbar, das gerade angezeigt wird.

Eigenschaften und Methoden von Dokument Objekten

- Die Eigenschaft `location.href` enthält die URL des Dokuments.

```
alert(document.location.href);
```

  - Durch Ändern der URL-Eigenschaft lädt der Browser die Seite neu.
- Mit der Methode `write` lässt sich das Dokument beschreiben.

```
document.write("<h1>Hello!</h1>");
```

  - Schreiben in den HTML-Strom kann mit Page Rendering kollidieren
    ➔ Skripte sollten besser die **Document Object Model (DOM) API** verwenden.

Nochmal LWAD Kapitel 4

```
var main = function () {
  "use strict";
  $(".comment-input button").on("click", function (event) {
    var $new_comment = $("<p>"),
    comment_text = $(".comment-input input").val();
    $new_comment.text(comment_text);
    $(".comments").append($new_comment);
  });
};
$(document).ready(main);
```

- Verwendet **jQuery** extensiv (große Zeitersparnis)
- **Aber**: es ist wichtig zu verstehen, was jQuery "versteckt"

Schritt für Schritt: interaktive Benutzeroberfläche erstellen

```
/* jQuery's Art, um DOM Elemente zuzugreifen */
$(".comment-input button").on("click", function (event) {
  // ...
});
```

1 2 3,4

1. Eine Steuerung wählen (z. B. eine Schaltfläche)
2. Ein Ereignis wählen (z.B. einen Klick auf die Schaltfläche)
3. Eine JavaScript-Funktion schreiben: was soll passieren, wenn das Ereignis auftritt? (z.B. ein Fenster erscheint)
4. Die Funktion an das Ereignis der Steuerung als Callback anhängen

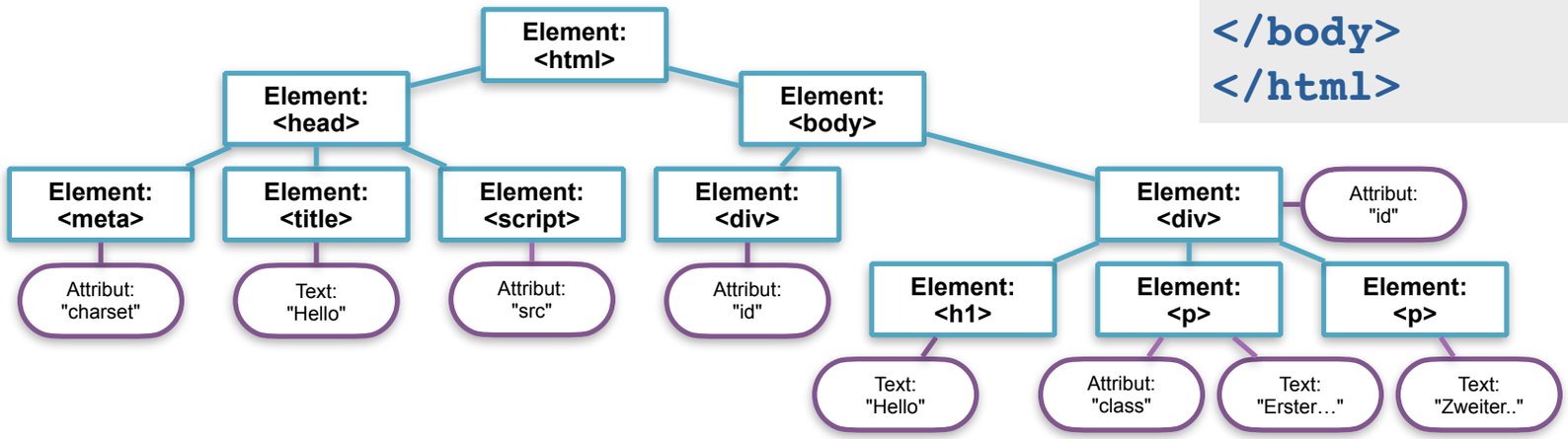
Wir wollen nun die Grundlagen dieser Schritte im **DOM** kennenlernen.

Die DOM API

DOM API

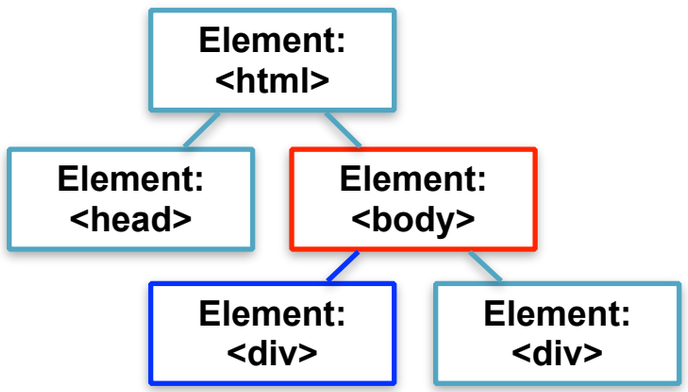
- Das **DOM** (Document Object Model) basiert auf dem Modell einer baumartigen Struktur der HTML-Elemente. Über die DOM-API lässt sich das DOM eines HTML-Dokuments auslesen und manipulieren.
- Jedes Element im Baum geht mit einem Knoten Objekt einher.
- Der Zugriff auf Knoten Objekte erfolgt allgemein über das `document` Objekt, (für den Body auch kurz per `document.body`).

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello</title>
<script src="/javascripts/helloDoc.js"/>
</head>
<body>
    <div id="prelude"></div>
    <div id="content">
        <h1>Hello</h1>
        <p class="first">Erster Absatz.</p>
        <p>Zweiter Absatz.</p>
    </div>
</body>
</html>
```



DOM API - Navigation

- Die Navigation im DOM erfolgt über Knoteneigenschaften:
- Auf Kinderknoten greift man per `childNodes` bzw. `firstChild` für das erste und `lastChild` für das letzte Kindelement zu, auf das Vaterelement über `parentNode`, auf Knoten der gleichen Ebene über `nextSibling` bzw. `previousSibling`.
  - Z.B. `document.body.firstChild`



- Bei fehlenden Knoten gibt die DOM-API null zurück.

DOM API - Finder Methoden

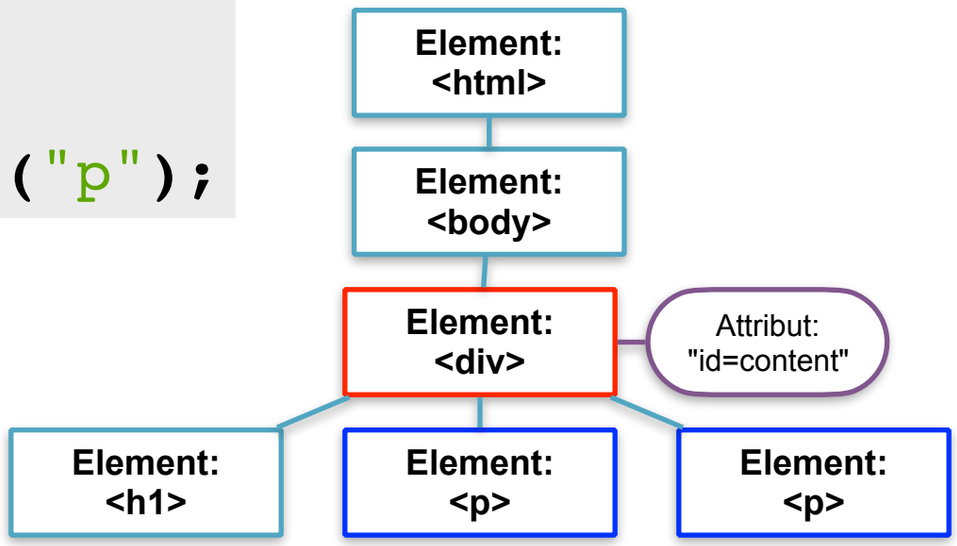
`document` (das Webseiten-Objekt) enthält einige Finder Methoden:

- `document.getElementById`
- `document.getElementsByTagName`
- `document.getElementsByClassName`
- `document.getElementsByName`
- `document.querySelector`
- `document.querySelectorAll`

DOM API - Finder Methoden

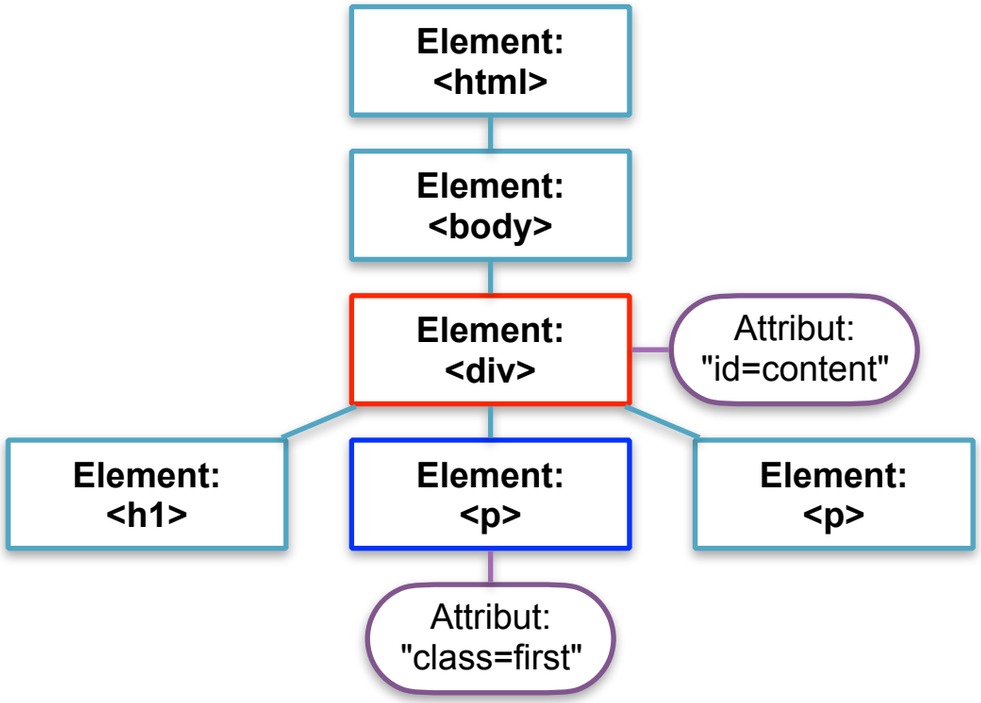
- Mit Finder Methoden können Knoten direkt adressiert werden.
- Mit `getElementById` lässt sich ein Knoten über sein ID-Attribut im HTML-Code finden.
  - Z.B. `document.getElementById("content");`
- `getElementsByTagName` liefert ein Array aller Nachfahren, die einen bestimmten Typ von HTML-Tag haben.

```
var contentElement =
   document.getElementById("content");
var paragraphs =
   contentElement.getElementsByTagName("p");
```



DOM API - Finder Methoden

- Es gibt noch weitere Finder Methoden.
- Über `getElementsByClassName` lassen sich Elemente mit einer bestimmten **CSS-Klasse** finden.
  - ```
    var firstParagraphs = contentElement.
      getElementsByClassName("first");
    ```



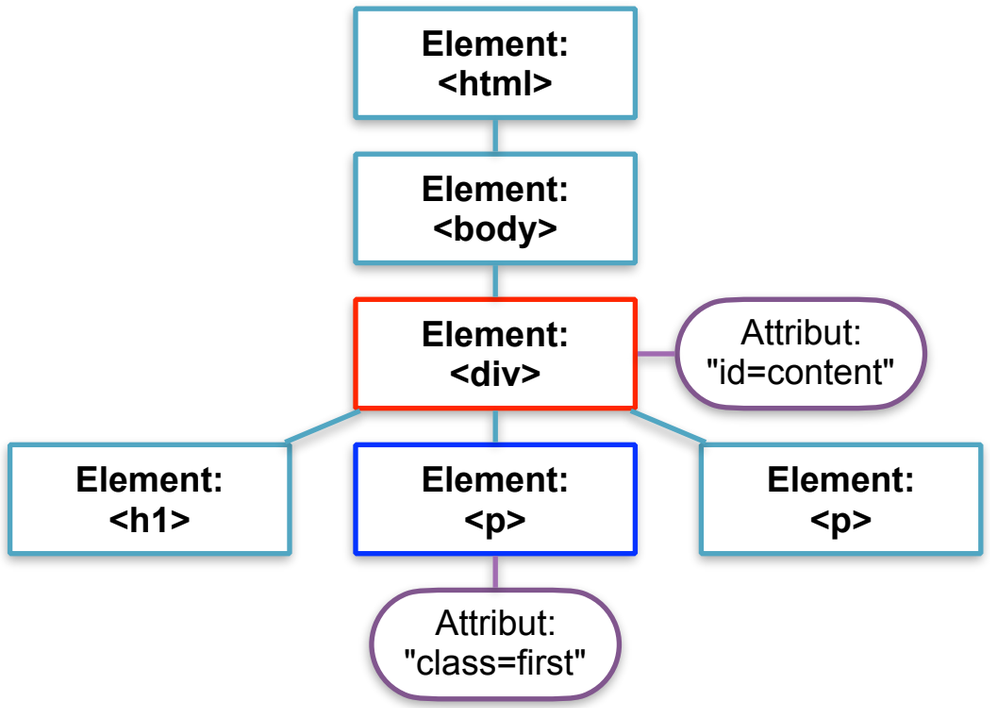
- Über CSS-Klassen lassen sich HTML-Elemente per **CSS** gestalten und auch semantisch annotieren.
    ```

DOM API - Finder Methoden

- Finder Methoden sind relativ **aufwändig**. Gefundene Knoten-Objekte sollten in Variablen zwischengespeichert werden.
  - (Zwischengespeicherte) Knoten-Objekte der DOM API passen sich dynamisch an Veränderungen des HTML-Dokumentes an.
- Finder werden nicht von allen Browsern nativ unterstützt.
  - Bibliotheken wie **jQuery** bieten einheitliche Methoden, die auf den meisten Browsern zu gleichen Ergebnissen führen.

DOM API - Knoten-Typen

- Knoten-Objekte repräsentieren HTML-Elemente, Attribute, Text oder Kommentare. Der Typ liegt als Eigenschaft `nodeType` vor.
  - `Node.ELEMENT_NODE // Integer Wert`
  - `Node.TEXT_NODE`
  - `Node.COMMENT_NODE`
  - `Node.DOCUMENT_NODE`
  - `Node.DOCUMENT_TYPE_NODE`
  - `Node.DOCUMENT_FRAGMENT_NODE`



- Die Eigenschaft `nodeName` entspricht dem Typ des Tags, aus dem ein ELEMENT_NODE gebildet wurde.
  -
```
contentElement.
  getElementsByClassName("first")[0].
    nodeName; // "P" Knotenname großgeschrieben
```

DOM API - Properties (ELEMENT_NODE)

```
<!-- HTML -->
<div id="main" class="main_class">
    <p>Hello <em>World!</em></p>
    <img id="worldImage" src="images/1.jpg" />
</div>
```

```
//JavaScript
var m = document.getElementById("main");
var w = document.getElementById("worldImage");
```

Property des Objekts	Beispiel
tagName	**`m.tagName`** ist **`div`**
className	**`m.className`** ist **`main_class`**
innerHTML	**`m.innerHTML`** ist **`<p>Hello....`**
src	**`w.src`** ist **`images/1.jpg`**

DOM API - Properties (Formulare)

```
<!-- HTML -->
<input id="firstName" type="text" value="Christian"/>
<input id="agreed" type="checkbox checked="checked" />
Did you read the terms and conditions?
```

```
//JavaScript
var f = document.getElementById("firstName");
var a = document.getElementById("agreed");
```

Property des DOM Objekts	Beispiel
value (Text in input Steuerung)	**`f.value`** ist vielleicht **`"Tom"`**
checked (checkbox)	**`a.checked`** ist **`true`**
disabled (ob Steuerung deaktiviert ist)	**`a.disabled`** ist **`false`**
readOnly (read-only text box)	**`f.readOnly`** ist **`false`**

DOM API - Manipulation: `setAttribute`

- Die Methoden `getAttribute` und `setAttribute` erlauben Zugriff auf die (HTML) Attribute eines Knotens. Eigene Properties lassen sich beliebig erweitern, wirken sich jedoch nicht visuell aus.

```
headline2.setAttribute("class", "important" );
headline2.class = "Some value…";
headline2.getAttribute("class"); // important
headline2.class; // some value
content.getElementsByClassName("important")[0].class;
// some value
```

- Die `style` Eigenschaft von HTML-Knoten repräsentiert den Style, der im HTML Dokument selbst (‚inline‘) definiert wurde.
  - Für die Manipulation des Styles existiert eine eigene API.
    - Z.B. `headline2.style.backgroundColor = "red";`
    - CSS-Bindestriche werden zu CamelCase.

DOM API - Manipulation mit `innerHtml`

- Die Eigenschaft `innerHtml` von Element-Knoten-Objekten enthält den untergeordneten HTML-Code

```
document.getElementById("content").innerHTML;
// <h1>Hello</h1>
// <p class="first">Erster Absatz.</p>
// <p>Zweiter Absatz.</p>
```

- `innerHtml` erlaubt auch die Manipulation des Dokuments.

```
document.getElementById("content").innerHTML =
   "<h1>Catchphrase</h1>" +
   "<p>And now for something different.</p>";
```

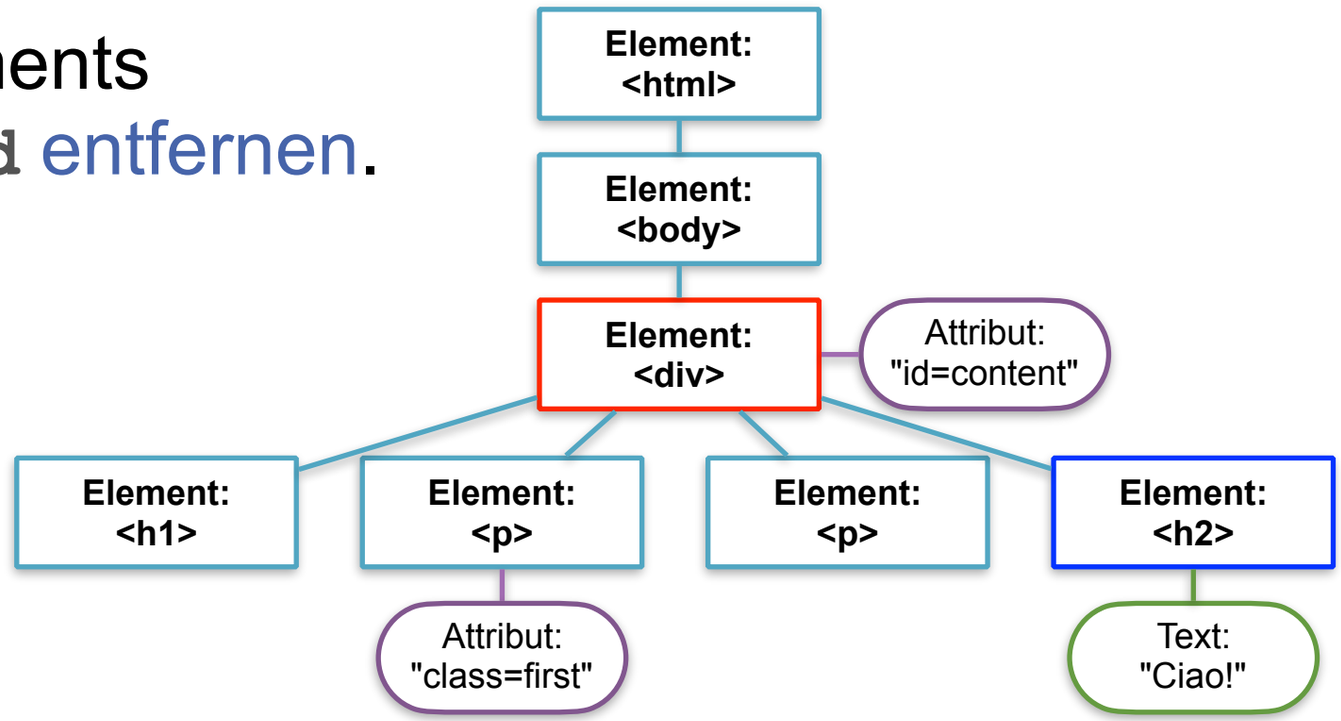
- Gleiches gilt für die Eigenschaft `nodeValue` eines Text-Knotens.

DOM API - neue Knoten erzeugen

- Elemente lassen sich auch per DOM API erzeugen und platzieren.

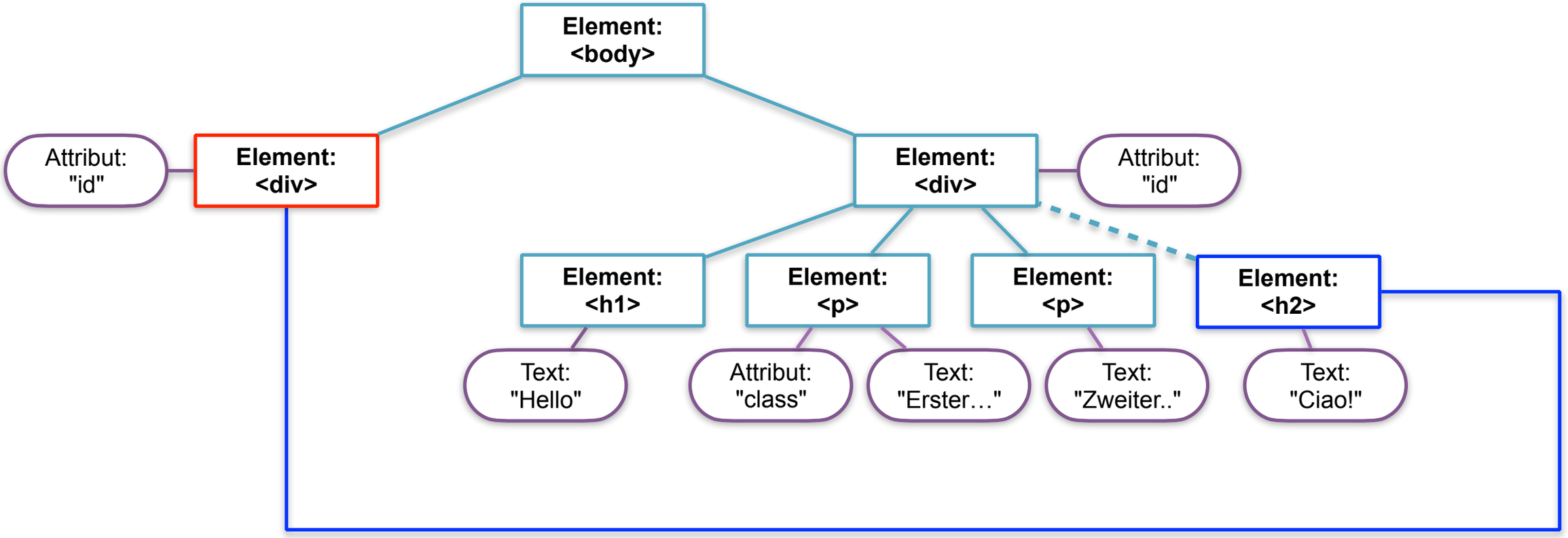
```
var headline2 = document.createElement("h2");
var headline2Value = document.createTextNode("Ciao!");
headline2.appendChild(headline2Value);
contentElement.appendChild(headline2);
```

- Knoten können vor bzw. nach einem Element mit `insertBefore` und `insertAfter` eingefügt werden.
- Knoten unterhalb eines Elements lassen sich mit `removeChild` entfernen.



Verschieben von Knoten im DOM-Baum

- Knoten Objekte können nur an einer Stelle des HTML-Baums existieren. Das Verschieben eines Knotens erfolgt daher einfach über Selektieren und Einfügen - der Knoten wird dabei von seiner ursprünglichen Position entfernt.
  - ```
    document.getElementById("prelude")
      .appendChild(headline2);
    ```


    ```

Timer und Events

Timer Ereignis und ein asynchroner Callback

- Über das `window`-Objekt lässt sich ein Timer starten, der nach einem Timeout eine Funktion startet:
  - Das Entfernen eines Timers geschieht per `clearTimeout`.

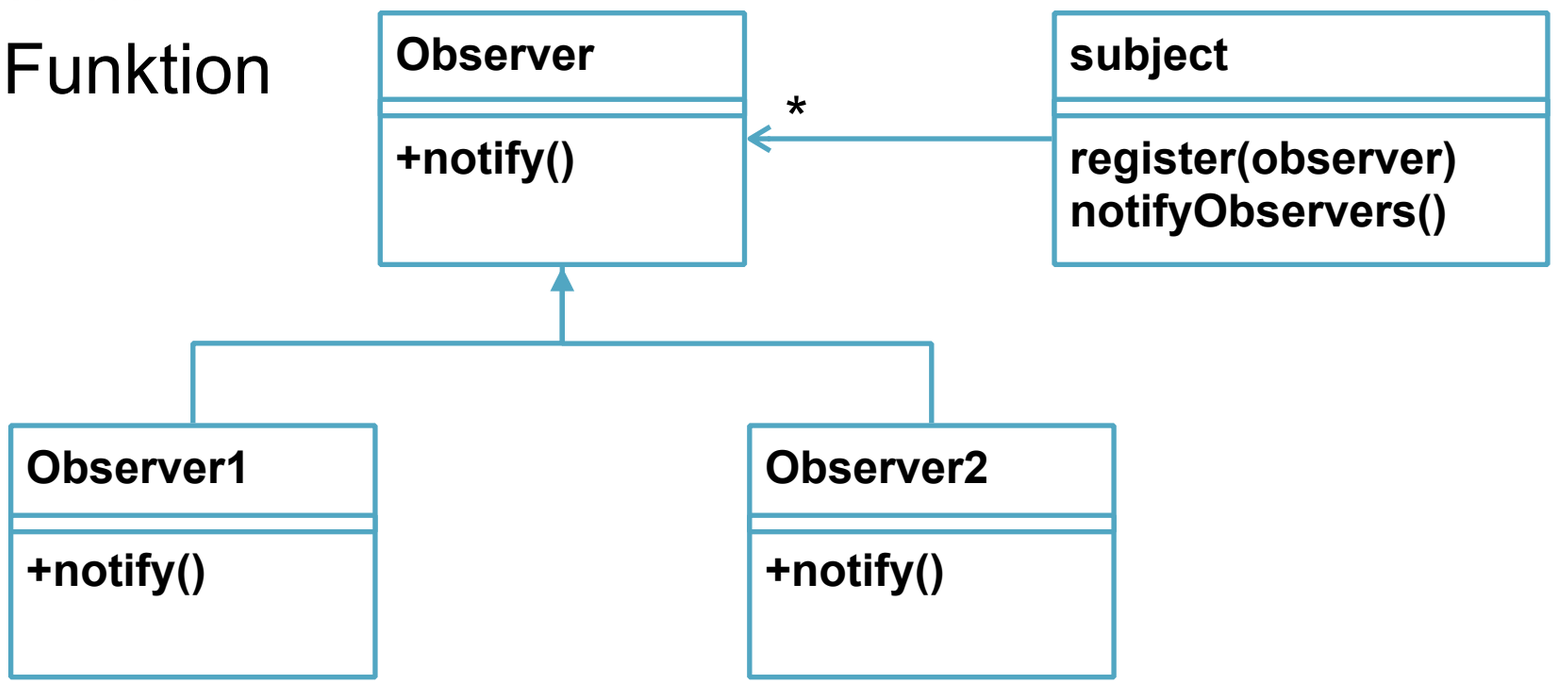
```
var timer = window.setTimeout(function() {
  alert("...how about a short break?");
}, 5000);
window.clearTimeout(timer); // changed my mind
```

- Der Aufruf von `setTimeout` erfolgt **asynchron**
  - Das Skript wird sofort weiter ausgeführt!
- Erst beim Timeout **Ereignis** wird die Funktion aufgerufen
  - Man spricht von einem **Callback**

Was passiert, wenn das Skript ausgeführt wird?

Event Handling

- Zur Realisierung dynamischer Webseiten, die auf Benutzeraktionen reagieren, können entsprechende **Ereignisse** verwendet werden.
- Das **Event-Handling** im Browser erfolgt nach folgendem Schema:
  1. Ein **Handler** (Funktion) für ein bestimmtes Event (z.B. `click`, `keyPressed`) wird auf einem DOM Element registriert.
  2. Die Handler Funktion wir aufgerufen, wenn das Event feuert.
- Entspricht dem **Observer Entwurfsmuster**
  - **Subjekt** → DOM Element
  - **Observer** → Handler Funktion



Events - Registrieren

- Die Registrierung eines Event-Handlers an einem DOM-Element erfolgt per **`addEventListener`** Methode mit drei Parametern:
  - das Event, auf das reagiert werden soll,
  - eine Funktion, die das Event Verarbeitet, und
  - einen booleschen Wert, der das Event-Bubbling kontrolliert.

```
function eventHandler(event) {
  alert("Event " + event.type
    + " fired on element " + this.tagName);
}
headline2.addEventListener("click", eventHandler, true);
```

- **`removeEventListener`** entfernt Event-Listener wieder.
- Es können auch mehrere Listener pro Event registriert werden.

Events - Event-Objekt

- Standardbrowser übergeben dem Event-Handler ein **Event Objekt** als Parameter. Hierüber kann der Event Kontext gelesen werden.
  - `mouseClick` Events haben Eigenschaften `clientX` und `clientY`
  - `keyPress` Events haben die Eigenschaft `keyCode`

```
document.addEventListener("keypress", function(event){
  var character = event.key;
  alert("Die Taste " + character + "wurde gedrueckt." );
}, true);
```

- `event.preventDefault()` verhindert, dass der Standard-Event Handler aufgerufen wird.

Events - Maus-Ereignisse

- Drücken der Maustaste beinhaltet mehrere Ereignisse ("Klick"):
  1. `mousedown`
  2. `mouseup`
  3. `click`
- Ein Mausklick bei gedrückter Maustaste ("Ziehen"):
  4. `mousedown`
  5. `mousemove`
  6. `...`
  7. `mousemove`
  8. `mouseup`
- Andere Maus-Effekte: `dblclick`, `mouseover`, `mouseout`

Events - in HTML Formularen

- Das Formular feuert ein **submit-Event** beim Absenden.
  - Eine Handler-Funktion kann das Formular individuell validieren.
  - `event.preventDefault()` verhindert das Absenden.
- Eingabeelemente feuern bei Auswahl ein **focus-Event** und beim Verlassen ein **blur-Event** und bei Änderung ein **change-Event**.
  - Die **Auswahl** eines Eingabeelements lässt sich auch erzwingen:

```
document.getElementsByTagName("INPUT")[0].focus();
```

Events - Keyboard- und Text-Ereignisse

Ereignis	Beschreibung
`blur`	Element verliert Tastaturfokus
`focus`	Element bekommt Tastaturfokus
`change`	Element hat sich geändert
`keydown`	User drückt Taste während Element Tastaturfokus hat
`keypress`	Benutzer drückt Taste und lässt los, während Element Tastaturfokus hat (problematisch)
`keyup`	Benutzer lässt Taste los, während das Element den Tastaturfokus hat
`select`	Benutzer selektiert Text in einem Element

Events - Keyboard- und Text-Ereignisse

- Benutzer hält eine Taste und löst **Autorepeat** aus: mehrere `keydown` und `keypress` Ereignisse
- `keydown` vs. `keypress`: verwende `keydown` wenn möglich; `keypress` ist bei Browsern weniger konsistent
- Nur ein Element zur Zeit hat den Fokus

Events - Keyboard- und Text-Ereignisse

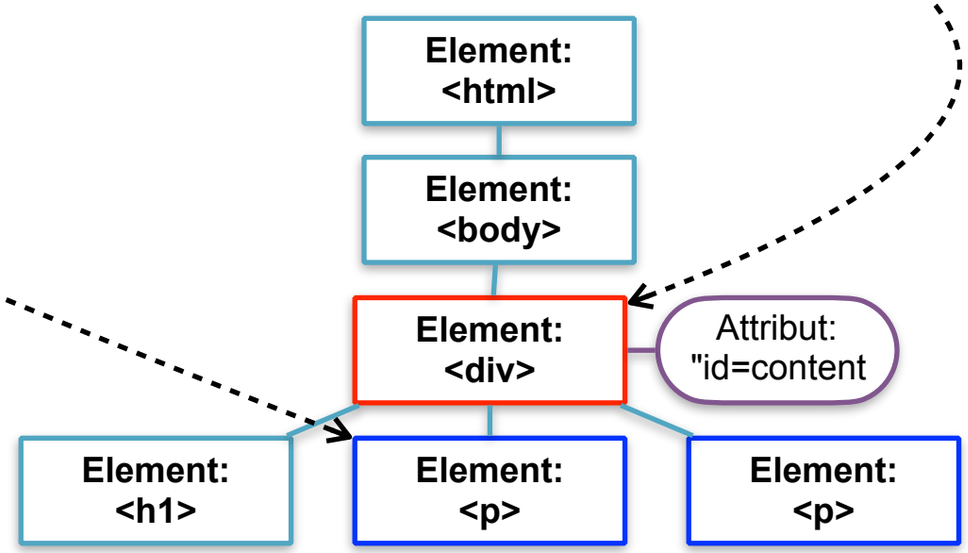
Welche Taste wurde gedrückt?

- Nicht mehr nutzen (deprecated):
  - `keyCode`: **ASCII-Wert** des gedrückten Zeichens (Umwandlung in ein Zeichen über `String.fromCharCode`)
  - `charCode`: das eingegebene Zeichen (nicht standardisiert!)
- Stattdessen
  - `key`: falls die Taste ein Zeichen ist: **Unicode Character String**, sonst: **standard Tastenwert** (z.B. "`Alt`"), oder "`Dead`".
  - Auch möglich: `altKey`, `ctrlKey`, `shiftKey`: `true`, wenn Alt, Ctrl, Shift gedrückt gehalten wird

Events - Event-Bubbling

- Ein Event wird von einem DOM-Element gefeuert und zuerst von dort registrierten Handlern verarbeitet. Dann wandert es den DOM-Baum hoch bis zur Wurzel. An jedem Element wird das Event erneut gefeuert und von lokalen Event-Handlern verarbeitet. Dieses Verhalten nennt man **Event-Bubbling**.

```
document.getElementsByTagName('P')[0].
  addEventListener('click', eventHandler1);
document.getElementById('content').
  addEventListener('click', eventHandler2);
```



Events - Event-Bubbling

- Man kann das Bubbling abbrechen, indem man im Event-Handler auf dem Event die Methode **stopPropagation** aufruft.

```
function eventHandler(event) {
  alert("Event " + event.type + " fired on element " +
    + this.tagName);
  event.stopPropagation;}
```

Events - Event Delegation

- Bei **Event Delegation** wird ein Handler auf einem übergeordneten Element registriert und per Event Bubbling erreicht. Dadurch wird bei vielen Nachfolgern (z.B. eine lange Liste) verhindert, dass an jedes Element ein Handler registriert werden muss.
- Im Event-Handler zeigt `event.target` auf die ursprüngliche Event-Quelle, `this` zeigt hingegen auf die aktuelle Event-Quelle.

```
document.getElementById("content").
  addEventListener("click", function(event) {
    if (event.type === "click" && event.target.tagName === "P"){
      window.alert("Paragraph '" +
        event.target.firstChild.nodeValue + "' clicked!");
    }
});
```

- Bei Event Delegation ist die Reaktion auf Benutzerereignisse ggf. etwas verzögert. Dies kann in bestimmten Fällen (z.B. `mouseOver` Event) störend sein.

Browser Interaktion mit jQuery

jQuery

- Die DOM-API ist nicht bei allen Browsern einheitlich implementiert.
  - Die API ist auch z.T. umständlich z.B. mit langen Funktionsnamen.
- Weitere Inkompatibilitäten bestehen bei der JavaScript Semantik, wie z.B. beim Iterieren einer Liste.
- **jQuery** ist eine JavaScript Bibliothek, die als einheitliche und gut verwendbare Abstraktionsschicht über den DOM dient.
  - Sie löst daneben auch die anderen Kompatibilitätsprobleme z.B. bei Iteratoren und der Ajax-API (dazu später mehr).
- jQuery wurde 2006 von John Resig vorgestellt und ist mittlerweile De-facto Standard bei der Web Client Programmierung.
- jQuery ist kein Framework, das ein spezifisches Paradigma implizieren würde. Die Bibliothek verhält sich sehr neutral.

jQuery verwenden

- jQuery ist u.a. von Content Delivery Networks (CDN) zu beziehen.
  - Google bietet mit den Ajax-APIs ein CDN für JavaScript-Bibliotheken.

```
<script
   src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
```

- `jQuery` (kurz: `$`) ist die Hauptfunktion und dient zum Finden von DOM-Elementen, die durch jQuery angereichert werden.
  - Auf angereicherten DOM-Elementen lassen sich weitere jQuery-Funktionen aufrufen.

jQuery Selektoren und Event-Handling

- Selektoren in jQuery nutzen eine domänenspezifische Sprache, ähnlich CSS-Selektoren.
  - `$("INPUT")` sucht Elemente über ihren Elementtyp.
  - `$(".messageInput")` sucht Elemente über CSS-Class-Attribute.
  - `$("#messageForm")` sucht ein Element über sein ID-Attribut.
- jQuery definiert ein eigenes Event-Handling.
  - Mit der on-Funktion bindet man einen Event-Handler an ein Event.
  - Mit der off-Funktion entfernt man einen Event-Handler wieder.

```
$(function() {
  $("#messageForm").on("submit", validate);
  $("INPUT")[0].focus();
});
```

Nach dem Laden des DOM Funktion ausführen

Literatur

- **Learning Web App Development, Kapitel 4**
- Empfehlung 1: Marijn Haverbeke, *"Eloquent JavaScrip"*, No Starch Press, 2014 (Online: http://eloquentjavascript.net)
- Empfehlung 2: Oliver Ochs, *"JavaScript für Enterprise-Entwickler"*, dpunkt.verlag, 2012