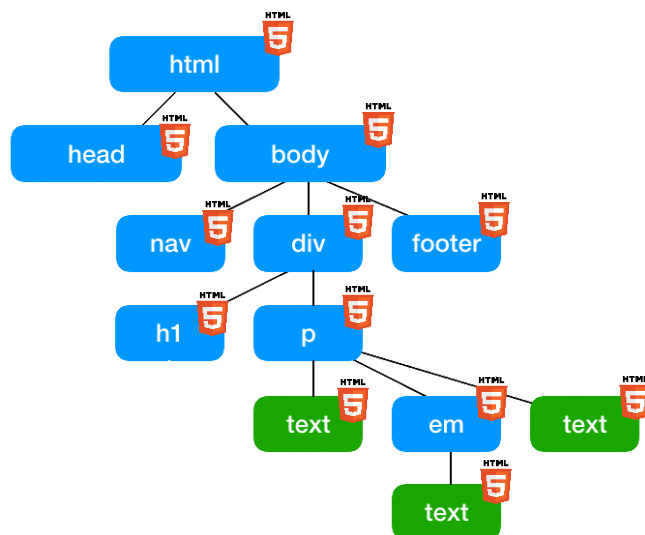


L2_1 Einführung – Javascript-Objekte – Teil 1

HTML-Dokumente lassen sich immer auch als Baumstruktur darstellen. Die Grundidee ist, dass jedes HTML-Element ein Knotenpunkt in einem Baum ist. Die Knoten gehen dabei eine Eltern-Kindbeziehung ein. Knoten ohne Kinder bezeichnen wir als Blätter. Die Struktur eines typischen HTML-Dokuments könnte wie folgt aussehen:



```
<html>
  <head></head>
  <body>
    <nav></nav>
    <div>
      <h1></h1>
      <p>
        abc<em>def</em>ghi
      </p>
    </div>
    <footer></footer>
  </body>
</html>
```

In der Abbildung sehen wir die folgenden Arten von Knoten:

- **Elementknoten** (blau): entspricht einem HTML-Element
- **Textknoten** (grün): entspricht dem Text innerhalb eines HTML-Elements

Ein HTML-Seite ist zunächst eine statische Seite. Bei jedem Aufruf der Seite wird diese sich gleich präsentieren. Möchten wir Inhalte verwalten oder die Darstellung modifizieren, müssen wir den HTML-Code im Editor anpassen.

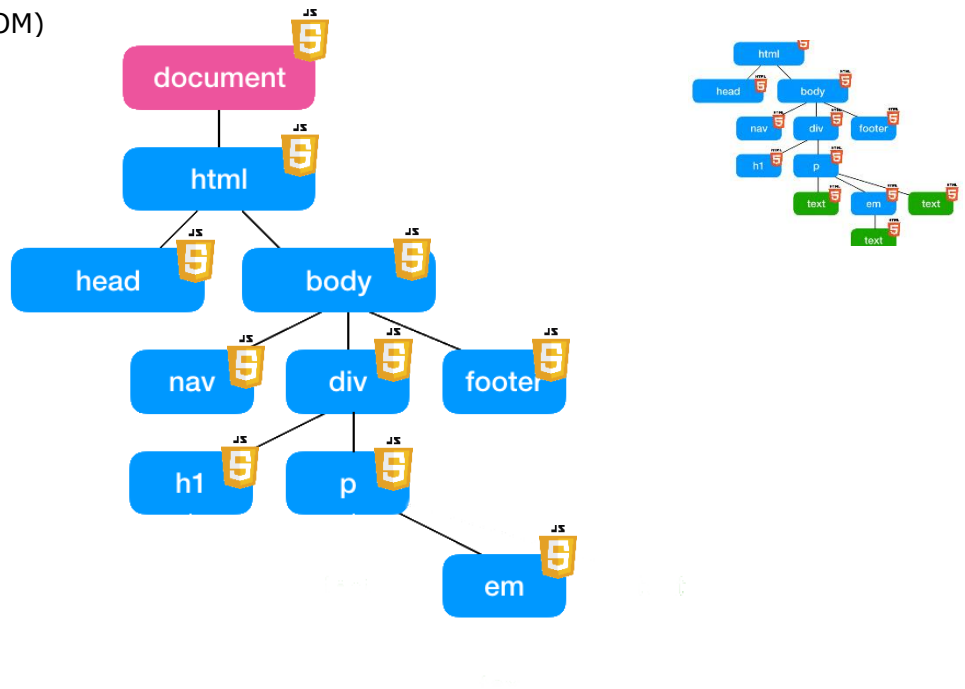
Möchten wir mehr Dynamik haben, dann ist dies nur unter Einsatz einer weiteren Webtechnologie möglich. Wir wollen im Folgenden betrachten, wie wir mit der Programmiersprache Javascript die Struktur und das Aussehen einer HTML-Seite programmieretechnisch beeinflussen können.

Zentrale Idee:

Zu jedem HTML-Element gibt es ein zugeordnetes Javascript-Objekt, so dass auch ein Baum von Javascript-Objekten darstellbar ist. Die Struktur ist identisch zum „HTML-Baum“, allerdings sind die Knoten nun als Javascript-Objekt zu interpretieren. Eine Manipulation des Javascript-Objektes hat direkt Einfluss auf das HTML-Element!

!

Document Object Model (DOM)



In der Abbildung sehen wir die folgenden Arten von Knoten:



- **Dokumentknoten** (pink): repräsentiert das gesamte Dokument und stellt viele Hilfsfunktionen bereit
- **Elementknoten** (blau): entspricht einem dem HTML-Element zugeordneten Javascript-Objekt

Man spricht auch vom DOM (Document Object Model). Das DOM verbindet Javascript mit den HTML-Elementen und erzeugt eine Baumstruktur, in der jedes HTML-Element eindeutig erreicht wird.

Vorgehensweise bei einer clientseitigen, dynamischen Modifikation des dargestellten Dokuments

1. In der Regel werden nur einzelne Bereich innerhalb der Seite manipuliert. Identifizieren Sie im ersten Schritt alle HTML-Elemente, die zur Lösung des Problems beitragen können. Dabei kann es sich um Eingabefelder (<input>), rein technische Bereiche (<div>) oder beliebige andere HTML-Elemente handeln.
2. Wir wissen, dass jedem der in Schritt 1 identifizierten HTML-Elementen ein Javascript-Objekt zugeordnet ist. Möchten wir eines der in Schritt 1 gefundenen HTML-Elemente modifizieren, dann müssen wir das dem HTML-Element zugeordnete Javascript-Objekt ermitteln. Diese Elemente können beispielsweise über deren ID bestimmt werden. Das `document`-Objekt stellt hierfür folgende Methoden bereit:

a. `document.getElementById(...)`  

b. `document.querySelector(...)`  

Die Varianten setzen voraus, dass das betroffene HTML-Element ein `id`-Attribut besitzt. Die Varianten liefern genau ein Javascript-Objekt.

Beispiel:

HTML:

```
<input id="zahl1" type="text">
```

Javascript:

```
jsobj = document.getElementById('zahl1');
```

```
jsobj = document.querySelector('#zahl1');
```

Die zweite Methode erlaubt die Auswahl des Objektes anhand eines CSS-Selektor-ähnlichen Ausdrucks.

3. Nachdem das Javascript-Objekt nun im Zugriff ist, muss ermittelt werden, welche Eigenschaften und Methoden das Objekt zur Verfügung stellt. Dabei können wir das Objekt als HTML-Elementobjekt (<https://www.w3schools.com/jsref/default.asp> → HTML-Objects) oder als DOM-Element Object (https://www.w3schools.com/jsref/dom_obj_all.asp) interpretieren. Praktisch bedeutet dies, dass alle Eigenschaften und Methoden der beiden Typen Verwendung finden können! Wir müssen uns also nicht explizit entscheiden, ob wir das Objekt als HTML-Elementobjekt oder als DOM-Element Object interpretieren!

Beispiel (Fortsetzung des obigen Beispiels):

```
obj.value = "Hallo Welt";
```

Interpretation als HTML-Elementobjekt

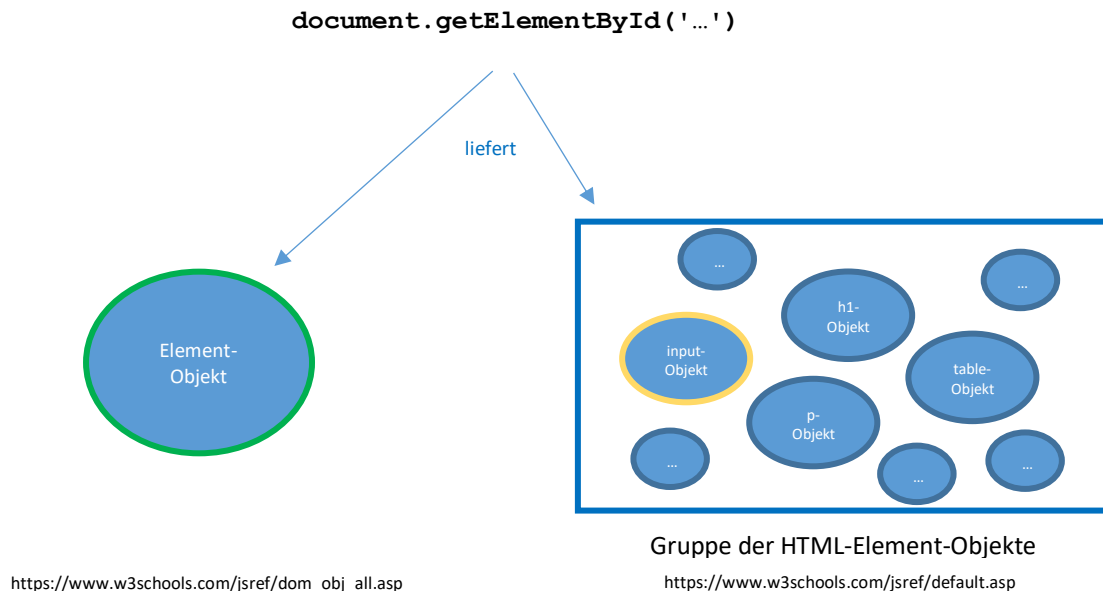
(Die Eigenschaft **value** gehört zum *HTML-Elementobjekt input*)

```
obj.className = "error";
```

Interpretation als element-Objekt

(Die Eigenschaft **className** gehört zum *DOM Element-Objekt*)

Folgende Grafik soll den Sachverhalt noch einmal verdeutlichen:



	Dynamische Webseiten mit Javascript Informationsmaterial	
--	--	--

Häufig genutzte Manipulationsmöglichkeiten

A. Dynamisches Ändern von bestehenden Inhalten (Texten)

Hier helfen die Attribute „**innerText**“ und „**innerHTML**“ des DOM Element-Objektes weiter. Beide Attribute können ausgelesen oder mit einem neuen Wert beschrieben werden. Bei Verwendung von „**innerText**“ werden kein HTML-Tags interpretiert. Diese Vorgehensweise eignet sich auch, um komplett neue Inhalte dynamisch zu generieren. Beachten Sie die unterschiedliche Groß-/Kleinschreibung.

Beispiel: Den Inhalt einer Überschrift von „Lottosimulation“ auf „Meine Lottosimulation“ ändern

HTML:

```
<h1 id="u1">Lottosimulation</h1>
```

Javascript:

```
document.getElementById('u1').innerText = "Meine Lottosimulation";  
oder alternativ  
document.getElementById('u1').innerHTML = "Meine Lottosimulation";
```

Achtung: Bei inhaltslosen Standalone-Elementen, also HTML-Elemente ohne Inhalt, haben diese beiden Eigenschaften keine Bedeutung (
, , ...)

B. Dynamisches Hinzufügen oder Ändern von Attributen eines HTML-Elements

Hier hilft die Methode „**setAttribute(...)**“ des DOM Element-Objektes weiter. Diese Methode erwartet zwei Parameter, nämlich den Namen des Attributs und den neuen Wert des Attributs (https://www.w3schools.com/jsref/met_element_setattribute.asp).

Beispiel 1: Das Verweisziel eines Links mit Hilfe des DOM Element-Objektes nachträglich ändern

HTML:

```
<a href="http://www.google.de" id="i"> Zur Suchmaschine </a>
```

Javascript (DOM Element-Objekt):

```
document.getElementById('i').setAttribute('href', 'http://www.bing.de');
```

	Dynamische Webseiten mit Javascript Informationsmaterial	
--	--	--

Alternativ kann aber auch die Eigenschaft „href“ des Anchor-Objektes aus der Gruppe der HTML-Element-Objekte genutzt werden. (https://www.w3schools.com/jsref/dom_obj_anchor.asp).

Beispiel: Das Verweisziel eines Links mit Hilfe des Anchor-Objekts aus der Gruppe der HTML-Element-Objekte nachträglich ändern

HTML:

```
<a href="http://www.google.de" id="i"> Zur Suchmaschine </a>
```

Javascript (A-Objekt aus der Gruppe der HTML-Element-Objekte):

```
document.getElementById("i").href = "http://www.bing.de"
```

Dynamisches Ändern der Gestaltung von Elementen (style-Attribut)

Hier hilft das Attribut „**style**“ des DOM Element-Objektes weiter. Dabei handelt es sich wieder um ein Objekt, welches weitere Eigenschaften besitzt. Die Namen dieser Eigenschaften ähneln sehr den Eigenschaftsnamen, die wir aus CSS kennen. Anstelle der in CSS zu findenden Snake-Schreibweise werden die Eigenschaften im „style-Objekt“ in Camel-Case-Schreibweise geschrieben (https://www.w3schools.com/jsref/dom_obj_style.asp).

Beispiel: Die Farbe einer Überschrift von blau auf rot ändern

HTML:

```
<h1 id="u1" style="color: blue">Lottosimulation</h1>
```

Javascript:

```
document.getElementById('u1').style.color = "red";
```

Dynamisches Ändern der Gestaltung von Elementen (classList)

Über das Attribut „**classList**“ des DOM Element-Objektes können Sie einem HTML-Element neue Klassen hinzufügen (Methode *add*) oder bestehende Klassen entfernen (Methode *remove*) (https://www.w3schools.com/jsref/prop_element_classlist.asp).

Dies ist oft dann nützlich, wenn man mehrere Gestaltungseigenschaften eines Elementes gleichzeitig ändern möchte. In der Regel erstellt man dann im ersten Schritt eine CSS-Regel, welche die gewünschten Eigenschaften definiert. In einem zweiten Schritt vergibt man allen HTML-Elementen, die sich nach diesen Eigenschaften richten sollen, eine Klasse mit dem passenden Wert (Name des CSS-Selektors). Häufig findet man dieses Vorgehen bei der Evaluierung von Formulardaten. Wird erkannt, dass ein Eingabefeld durch den Anwender mit ungültigen Daten beschrieben wurde, dann wird diesem Feld eine Klasse hinzugefügt, welche dazu führt, dass das Formularfeld mit den fehlerhaften Eingaben speziell markiert wird – beispielsweise durch einen dunkelroten Rahmen und einem hellroten Hintergrund.

Beispiel: Die Hintergrundfarbe und den Rahmen eines Eingabefeldes ändern

CSS:

```
.error {  
    border: 1px solid red;  
    background-color: #ffaana;  
}
```

HTML:

```
<input type="text" id="i">
```

Javascript:

```
document.getElementById('i').classList.add("error")
```

```
//Klasse wieder entfernen:  
document.getElementById('i').classList.remove("error")
```