Zusammenfassung WEBAPP FS2018

Alex Neher

June 25, 2018

Inhalt

1 Einführung

1.1 Technologien LAMP/MEAN Stack

L inux

A pache

M ySQL

P HP

M ongoDB

E xpress

A ngularJS

N odeJS

2 JavaScript

2.1 Grundlegende Sprachkonzepte

Javascript, auch ECMAScript genannt ist eine clientseitige web-Development Sprache für DOM- und CSS-Manipulation, AJAX oder EventHandling. Javascript kann in HTML-Code eingebunden werden, entweder inline über `<script></script>`-Tags, es kann mittels `<script src=path/to/file.js></script>` geholt werden oder direkt über EventHandler `<input type="checkbox" name="options" onchange="order.options.giftwrap = this.checked;">`.

2.1.1 Variablen definieren

Javascript hat keine Typisierung. Das heisst, Variablen können einfach mit dem Keyword `var` definiert werden, ohne dass ein Datentyp spezifiziert werden muss.

```
var a = 2; // a ist nun eine Nummer
a = 'Jetzt bin ich ein String';
a = false;
```

2.1.2 Arrays

```
//Instanziierung von Arrays über var
var emptyArray = [];
var numberArray = [1, 3, 6];

//Instanziierung von Arrays über new
var anotherEmptyArray = new Array();
var arrayOfFive = new Array(5);

//Da JavaScript keine Typisierung kennt, sind auch gemsiche Arrays möglich
var mixedArray = ["String", 4, true];

//Es können auch Objekte in Arrays verpackt werden
var modulesArray = [
   {title:"WEBAPP", instructor:"Koller"},
   {title:"WEBTEC", instructor:"Infanger"}
];

//Zugriff und hinzufügen von Array-Elemente erfolgt gleich wie bei bekannten Sprachen
console.log(numberArray[0]) //1
numberArray[3] = "new Element"

//Arrays müssen nicht zwingend ganz gefüllt sein
var sparseArray = new Array(1000);
```

```
24 console.log(sparseArray[500]); //undefined
```

2.1.3 Objekte

JavaScript kann auch objektbasiert programmiert werden. Es gibt grundsätzlich vier Möglichkeiten, Objekte zu instanziieren:

```
1  //1. über "var"
2  var bachelorModule = {
3     title: "Webapplication Development",
4     instructor: "Thomas Koller"
5  };
6
7  //2. über new und dem default-Konstruktor
8  var bachelorModule = new Object();
9
10 //3. über Object.create()
11 var bachelorModule = Object.create(Object.prototype); //ein leeres Objekt
12
13 //4. mit einem bereits bestehenden Objekt als Prototyp
14 var masterModule = Object.create(bachelorModule) //ist jetzt ein "Klon" von
       bachelorModule
```

Der Zugriff auf Properties funktioniert wie bei anderen objektorientierten Sprachen

```
1 console.log(bachelorModule.title); //Output: Webapplication Development
2 console.log(bachelorModule["instructor"]); //Output: Thomas Koller
```

Objekte sind dynamisch. Das heisst, es können zur Laufzeit noch Properties hinzugefügt oder entfernt werden:

```
1 //hinzufügen von Properties
2 bachelorModule.credits = 3;
3
4 //entfernen von Properties
5 delete bachelorModule.credits;
6
7 //Ebenfalls kann gecheckt werden, ob ein Property existiert
8 bachelorModule.hasOwnProperty("title"); //true
9 bachelorModule.hasOwnProperty("credits"); //false
```

2.2 Funktoinsdeklarationen und -aufruf

In Javascript werden Funktionen als Objekte behandelt. Sie können ebenfalls mit dem Keyword `var` erstellt werden. Jede Funktion hat ihren eigenen Kontext/Scope. Jede Funktion hat per Default zwei Parameter: `this` und `arguments`.

Der `this` Parameter gibt den Kontext der Funktion zurück. Dieser hängt davon ab, wie und wo die Funktion aufgerufen wird. `arguments` ist ein Array, in welchem alle mitgegebenen Argumente speichert.

```
1 //1. Über anonymes Function Literal
2 var add = function(a,b){
3    console.log(arguments[0]) //gibt den Wert von a zurück
4    return a+b;
5 }
```

```
6
7  //2. Über Function Literal mit Name
8  var sub = function sub(a,b){
9     return a-b;
10 }
11
12 //3. Über Function Declaration
13 function mult(a,b){
14    return a*b;
15 }
16
17 //4. Über Immediate Function Invocation
18 var TenDividedByTwo = function(a,b){return a/b;}(10,5);
19 console.log(TenDividedByTwo) //Output: 5
```

Funktionen können auch direkt in Objekten definiert werden

```
1  var person = {
2     firstName = "Thomas",
3     lastName = "Koller",
4
5     printFullName: function(){
6        console.log(this.firstName + " " + this.lastName);
7     };
8  }
9
10 person.printFullName();
```

Mit dem 'apply'-Pattern können auch Funktionen von anderen Objekten aufgerufen werden

```
1  var anotherPerson = {
2     firstName = "Donald",
3     lastName = "Trump"
4  }
5
6  anotherPerson.printFullName() //error: undefined
7
8  aPerson.printFullName.apply(anotherPerson); //Output: Donald Trump
```

2.3 Closures

Wie bereits erwähnt, werden Funktionen als Objekte behandelt und haben ihren eigenen Kontext/Scope. Das heisst, von ausserhalb kann nicht direkt auf Variablen innerhalb einer Funktion zugegriffen werden, sondern nur über verschachtelte Funktionen. Dieses Konstrukt nennt man **Closure**.

```
1  var myCounter = (function(){
2     var value = 0;
3     return {
4        increment: function(inc){
5           value =+ inc;
6        },
7        getValue: function(){
8           return value;
9        }
10    };
11 }());
```

```
12
13 myCounter.increment(10) //value = 10
14 console.log(myCounter.value); //error: undefined
15 console.log(myCounter.getValue()); //10
```

2.4 Prototypen und Konstruktoren

2.4.1 Konstruktoren

JavaScript hat, wie auch andere objektorientierten Sprachen, Konstruktoren (die immer gross geschrieben werden)

```
1 function Name(vorname, nachname){
2    this.vorname = vorname;
3    this.nachname = nachname;
4    this.birthDate = {
5       year: 0,
6       month: 0,
7       day: 0
8    }
9 }
```

2.4.2 Prototypen

Wie bereits im Kapitel 'Funktionen' beschrieben, können Funktionen direkt in Objekten definiert werden. Dann existieren sie jedoch nur für diese eine Objekt (z.B. dem Objekt aPerson). Wenn ich nun eine Funktion definieren will, die für alle `Name`-Objekte existiert, muss ich sie mithilfe dem `prototype`-Keyword definieren.

```
1 Name.prototype.hello = function(){
2    console.log("Hello " + this.vorname);
3 }
4
5 var aPerson = new Name("Thomas", "Koller");
6 aPerson.hello();
```

Diese Methode funktioniert, da jedes Objekt ein Prototype hat. Wenn ein gesuchtes Property nicht im Objekt-eigenen Prototype gefunden, so wird rekursiv im Prototype des Prototype-Objekts gesucht, bis man ganz oben bei `Object` angekommen ist. Falls dort immer noch nichts gefunden wurde, wird `null` zurückgegeben.

Wenn ein Objekt mithilfe der `Object.create()`-Methode instanziiert wird, so hat das neu erstellte Objekt den Prototypen des mitgegebenen Objekts.

```
1 var obj1 = {
2    a: 1
3 }
4
5 var obj2 = Object.create(obj1); //Der Prototyp von obj2 ist jetzt obj1
6
7 console.log(obj2.a) //Output: 1
```

`obj2` selbst hat kein 'a'-Property, es wird also eine Stufe höher gesucht, im Property von `obj2`, also `obj1`

2.5 Asynchrone Programmierung

Mittels den Methoden `long setInterval(function f, unsigned long interval, any args)` und `long setTimeOut(function f, unsigned long timeout, any args)` kann die Ausführung der übergebenen Methode verzögert (setTimeOut) oder in einem definierten Intervall wiederholt (setInterval) werden. Die Ausführung der Methode `f` wird um `timeout` Millisekunden verzögert bzw. nach `interval` Millisekunden wiederholt.

2.6 Promises

3 Client Side JavaScript

3.1 JavaScript Verarbeitung / Event Handling

JavaScript kann normal, asynchron oder deferred ausgeführt werden. Bei der normalen Verarbeitung wird das HTML-Parsing pausiert, das JS-Skript heruntergeladen, kompiliert und ausgeführt und erst anschliessend mit dem HTML-Parsing weitergemacht. Wenn man die asynchrone Verarbeitung wählt, wird das JS-Skript im Hintergrund heruntergeladen. Erst wenn das Skript heruntergeladen wurde, wird das HTML-Parsing pausiert und das Skript wird ausgeführt. Bei der deferred-Methode wird das Skript ebenfalls im Hintergrund heruntergeladen. Jedoch wird hier gewartet, bis das gesamte HTML-Parsing abgeschlossen ist, bevor das Skript ausgeführt wird.

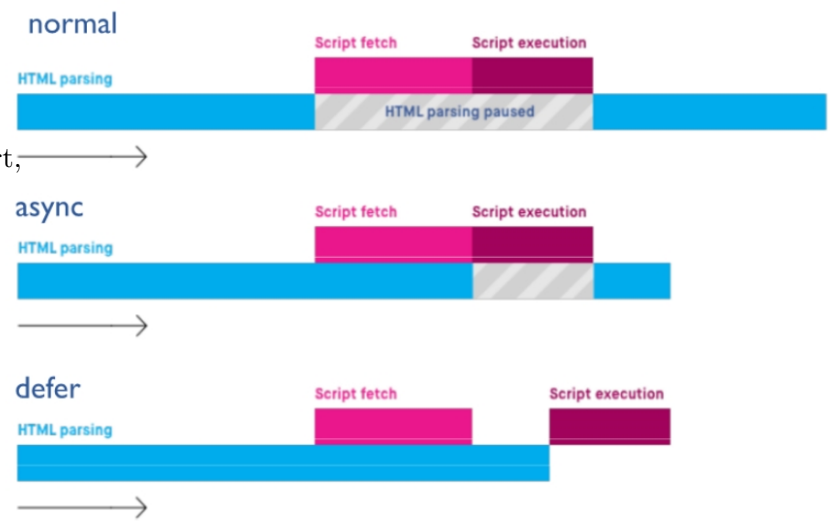

Abb. 3.1: Verschiedene Arten, wie JS verarbeitet werden kann

3.2 Konzept DOM / CSS Manipulation

3.2.1 DOM-Manipulation

DOM steht für *Document Object Model* und bezeichnet die Struktur einer HTML-Website. Alle Objekte der Website werden in der Baumstruktur des DOMs als Node abgespeichert. JavaScript kann direkt auf diese Nodes zugreifen z.B. wäre `"simple"` in Abb. 3.2 mittels `document.childNodes[0].childNodes[1].lastChild.firstChild.nextSibling.childNodes[0]` erreichbar. Da dies jedoch ein bisschen umständlich ist, werden im nächsten Kapitel einige einfachere Selektions-Methoden vorgestellt.

Wie bereits zu Beginn des Kapitels erwähnt, wird JavaScript unter anderem zur DOM-Manipulation verwendet. Um Elemente des DOM manipulieren zu können, muss dem Skript zuerst mitgeteilt werden, welches Element man manipulieren möchte. Dabei gibt es verschiedene Methoden:

Über ID (empfohlen)

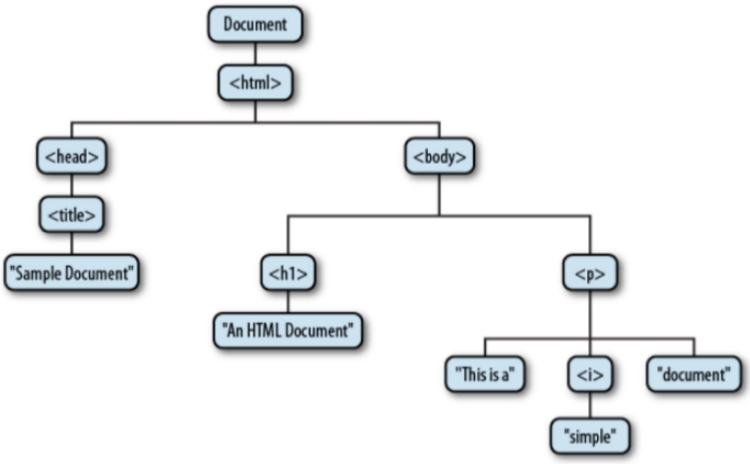

Abb. 3.2: Beispiel eines DOM-Baumes

```
var button = document.getElementById("button1")
```

```
<button id="button1">Click me!</button>
```

Über Namen

Diese Methode gibt eine NodeList zurück mit allen gefundenen Elementen

```
var buttons = document.getElementsByName("option_buttons")
console.log(buttons[0].tagName) //Output: button
```

```
<button name="option_button">Pizza Margharita</button>
<checkbox name="option_button">Pizza Diavola</checkbox>
```

Über Tags

Funktioniert gleich wie `getElementsByName` aber filtert nach HTML-Tags

```
var buttons = document.getElementsByTagName("button")
console.log(buttons[0].name) //Output: option_button
```

```
<button name="option_button">Pizza Margharita</button>
```

Über Klasse(n)

Gibt eine HTML-Collection zurück. Falls mehrere Klassen spezifiziert werden, muss das Element Mitglied *aller* Klassen sein

```
var buttons = document.getElementsByClassName("options")
console.log(buttons[0].name) //Output: option_button
```

```
<button class="options">Pizza Margharita</button>
```

Über CSS-Selektoren

Nur kompatibel mit HTML5. Gibt eine NodeList zurück.

```
var logs = document.querySelectorAll("#log>span)
```

```
<div id=span>
<span>I'm selected</span>
</div>
```

3.3 JQuery

3.4 Tools und Libraries

3.5 Modules

4 Server Side JavaScript

4.1 NodeJS

4.2 Express

5 TypeScript

6 Single Page Application

6.1 Konzept

6.2 Begriffe

7 Angular

7.1 Eigenschaften

7.2 Organisation einer Angular Applikation

8 REST

8.1 Eigenschaften

8.2 Resourcen und Methoden

9 Mobile App

9.1 Ionic

9.2 Cordova

10 Authentifizierung

10.1 Standard-Mechanismen

10.2 OAuth2.0