Zusammenfassung WEBAPP FS2018

Alex Neher

June 20, 2018

Inhalt

Abbildungsverzeichnis

1 JavaScript

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

1.1 Basics

1.1.1 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;
```

1.1.2 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);
   };
```

```
}

person.printFullName();
```

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

```
var anotherPerson = {
   firstName = "Donald",
   lastName = "Trump"
}

anotherPerson.printFullName() //error: undefined

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**.

```
var myCounter = (function(){
   var value = 0;
   return {
      increment: function(inc){
         value =+ inc;
      },
      getValue: function(){
         return value;
      }
   };
}());

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

1.1.3 Objekte

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

```
//1. über "var"
var bachelorModule = {
   title: "Webapplication Development",
   instructor: "Thomas Koller"
};

//2. über new und dem default-Konstruktor
var bachelorModule = new Object();

//3. über Object.create()
var bachelorModule = Object.create(Object.prototype); //ein leeres Objekt

//4. mit einem bereits bestehenden Objekt als Prototyp
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

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`

1.1.4 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);
console.log(sparseArray[500]); //undefined
```

1.1.5 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.

```
var json = JSON.stringify(bachelorModule));
//Output: {"title":"WebApp","instructor":"Thomas Koller"}

var otherBachelorModule = JSON.parse(json);
```

2 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.