

Softwareprojekt 2

Einstieg in Node.js Teil 2

Prof. Dr. Darius Schippritt

Büro L4.2-E02-140

darius.schippritt@hshl.de

Überblick

- **Das letzte Mal... und Lernziele**
- Einstieg in Node.js Teil 2
- ECMAScript 6
- Zum Schluss...

Das letzte Mal...



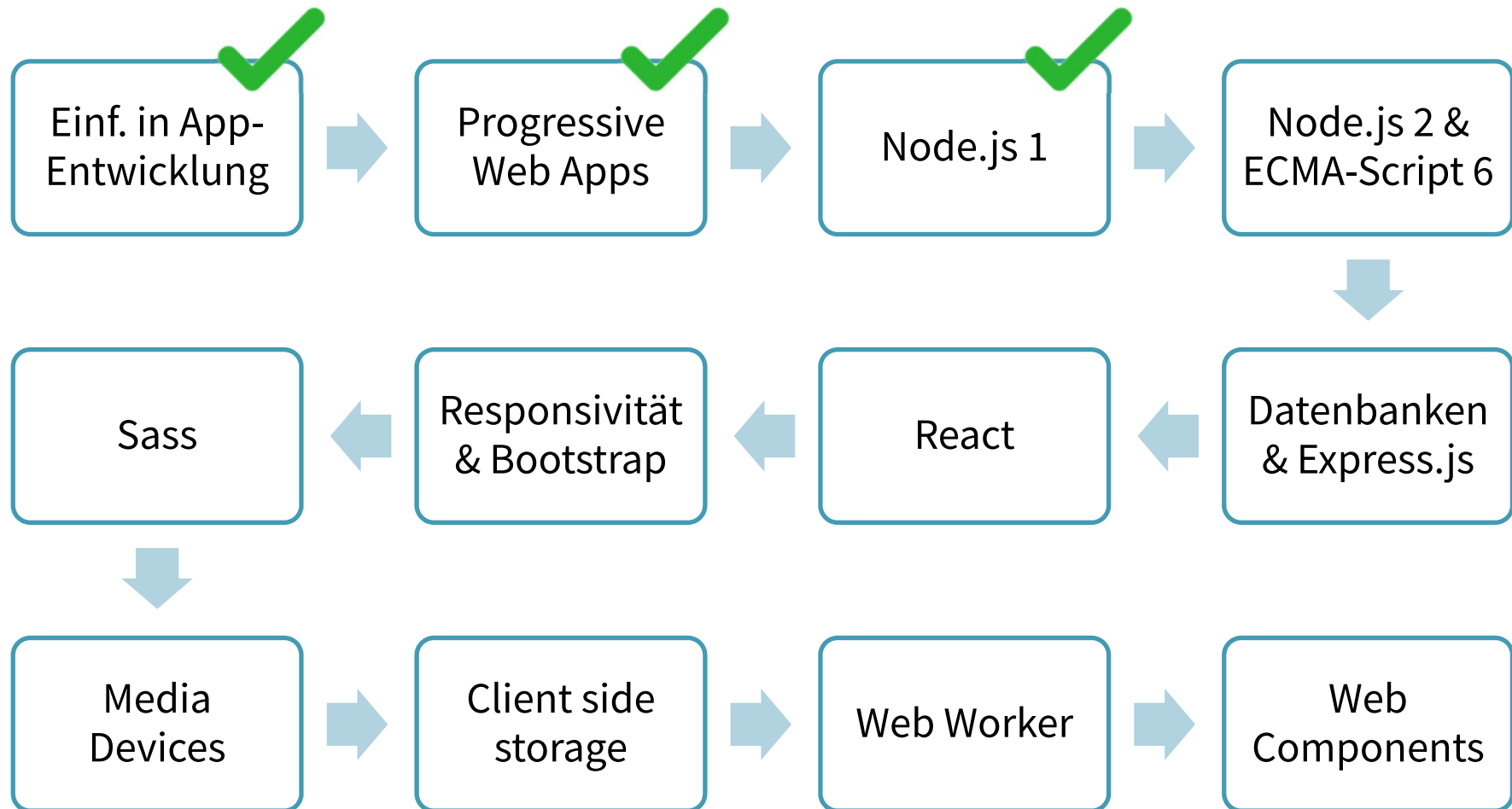
Entscheidungs-
kriterien

SPA-Frameworks

Backend-Stack



Ziele und Inhalte



Lernziele



- Sie können hochgeladene Dateien verarbeiten.
- Sie können Dateien lesen und schreiben.
- Sie können zusätzliche Module in Node.js installieren oder eigene Module erstellen.
- Sie können asynchrone Prozesse steuern und gezielt mit Event-Handlern und Emittern arbeiten.

Überblick

- Das letzte Mal... und Lernziele
- **Einstieg in Node.js Teil 2**
- ECMAScript 6
- Zum Schluss...

Einstieg in Node.js Teil 2

Node Package Manager (NPM)

- Node Package Manager erleichtert die Paketverwaltung für Node.js
- Kommandozeilenaufruf: **npm [cmd] [package name]**
- NPM unterstützt z.B. Installation, Update, Deinstallation und Suche von Packages
- Packages können als Modul über **require()** eingebunden werden

```
c:\Progs\nodejs>npm install express
npm WARN saveError ENOENT: no such file or directory, open 'c:\Progs\nodejs\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'c:\Progs\nodejs\package.json'
npm WARN nodejs No description
npm WARN nodejs No repository field.
npm WARN nodejs No README data
npm WARN nodejs No license field.

+ express@4.16.2
added 44 packages, removed 407 packages and moved 4 packages in 7.203s
```

Einstieg in Node.js Teil 2

Upload von Dateien

- 1. Step
 - Webserver, der HTML-Formular samt input-Elementen erstellt

```
var http = require("http");
http.createServer(function(request, response) {
    response.writeHead(200, {"content-type": "text/html; charset=utf-8"});

    var body =
        "<form action='fileupload' method='post' enctype='multipart/form-data'>" +
        "<input type='file' name='file'><br>" +
        "<input type='submit'>" +
        "</form>";
    var htmlResponse = getHTMLStruct("Webtechnologien", body);

    response.end(htmlResponse);
}).listen(8080, "127.0.0.1");
console.log("Webserver wird ausgeführt.");
```


Einstieg in Node.js Teil 2

Upload von Dateien

- 2. Step
 - Ggf. Installation des Moduls formidable mit Befehl **npm install formidable**
 - Parsen des abgesendeten Formulars (File wird dabei automatisch im tmp-Ordner abgelegt)

```
var http = require("http");  
var formidable = require("formidable");  
http.createServer(function(request, response) {  
  if (request.url == "/fileupload") {  
    var form = new formidable.IncomingForm();  
    form.uploadDir = "/tmp/";  
    form.parse(request, function(error, fields, files) {  
      if (error)  
        throw error;  
      response.write("File uploaded");  
      response.end();  
    });  
  }  
});
```

Einstieg in Node.js Teil 2

Upload von Dateien

- 3. Step
 - Verschieben der hochgeladenen Datei vom tmp-Ordner in Ziel-Ordner

```
var http = require("http");
var formidable = require("formidable");
var fs = require("fs");
http.createServer(function(request, response) {
  if (request.url == "/fileupload") {
    var form = new formidable.IncomingForm();
    form.parse(request, function(error, fields, files) {
      var oldpath = files.file.path;
      var newpath = __dirname+"/files/"+files.file.name;
      fs.rename(oldpath, newpath, function(error) {
        if (error)
          throw error;
        response.write("File uploaded and moved");
        response.end();
      });
    });
  }
});
```

Einstieg in Node.js Teil 2

Upload von Dateien (vollständiges Skript)

```
var http = require("http");
var formidable = require("formidable");
var fs = require("fs");
http.createServer(function(request, response) {
  if (request.url == "/fileupload") {
    var form = new formidable.IncomingForm();
    form.parse(request, function(error, fields, files) {
      var oldpath = files.file.path;
      var newpath = __dirname+"/files/"+files.file.name;
      fs.rename(oldpath, newpath, function(error) {
        if (error)
          throw error;
        response.write("File uploaded and moved");
        response.end();
      });
    });
  }
  else {
    response.writeHead(200, {"content-type": "text/html; charset=utf-8"});

    var body =
      "<form action='fileupload' method='post' enctype='multipart/form-data'>"+
      "<input type='file' name='file'><br>"+
      "<input type='submit'>"+
      "</form>";
    var htmlResponse = getHTMLStruct("Webtechnologien", body);

    response.end(htmlResponse);
  }
}).listen(8080, "127.0.0.1");
console.log("Webserver wird ausgeführt.");
```

Einstieg in Node.js Teil 2

Versenden von Emails

- Verwendung des Moduls nodemailer (ggf. Installation notwendig, **npm install nodemailer**)
- **transporter** legt Mailedienstleister und Authentifizierungsoptionen fest
- **mailOptions** enthalten Sender, Empfänger, Subject und Body
- Absenden selbst über Funktion **.sendMail()**

```
var nodemailer = require("nodemailer");
var transporter = nodemailer.createTransport({
  service: "gmx",
  auth: {
    user: "sender@gmx.de",
    pass: "password"
  }
});

var mailOptions = {
  from: "sender@gmx.de",
  to: "recipient@gmx.de",
  //to: "recipient@gmx.de, recipient2@gmx.de",
  subject: "Testmail via Node.js",
  text: "Mailbody"
  //html: "<p>Mailbody</p>"
};

transporter.sendMail(mailOptions, function(error, info) {
  if (error)
    console.log(error);
  else
    console.log("Success -> "+info.response);
});
```

Einstieg in Node.js Teil 2

Eigene Module

- Zusätzliche eigene Module
- Speicherung als js-Datei
- Einbindung per **require()**
- **exports** macht Variablen, Objekte und Funktionen außerhalb des Moduls sichtbar

```
var date = require("./dateTools");  
console.log(date.germanDate());
```

1.12.2017

dateTools.js

```
date = function() {  
    var date = new Date();  
    return date.getFullYear()+"-"+  
           +(date.getMonth()+1)+"-"+  
           +date.getDate();  
}  
  
exports.germanDate = function() {  
    var date = new Date();  
    return date.getDate()+"."  
           +(date.getMonth()+1)+"."  
           +date.getFullYear();  
};
```

Einstieg in Node.js Teil 2

Buffer

- Buffer erlauben Transport von Binärdaten
- HEX-Wert von Daten wird encodiert (Standard UTF-8)
- Zahlreiche Methoden, z.B.
- **.write()** → schreibt in einen Buffer
- **.toString()** → gibt Buffer als Text aus
- **.toJSON()** → gibt Buffer im JSON-Format aus

```
var myBuffer = new Buffer(100);  
  
var len = myBuffer.write("Guten Tag");  
  
console.log("Länge="+len+"\n");  
console.log(myBuffer);
```

Länge=9

```
<Buffer 47 75 74 65 6e 20 54 61 67 00 00 00 00 00 00 00 00  
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ... >
```

Einstieg in Node.js Teil 2

Zugriff auf Dateien / Filesystem

- Zugriff über buffer-Objekt, das im Modul „fs“ enthalten ist
- **readFile()** liest Datei aus und speichert Inhalt in lokale Variable **data**
- „magische“ Variablen:
 - __filename → vollen Pfad und Namen der js.Datei
 - __dirname → vollen Pfad der js.Datei

```
var fs = require("fs");

fs.readFile("test.txt", function (error, data) {
  console.log(data);

  console.log(data.toString());
});
```

Dies ist eine
Testdatei.

Einstieg in Node.js Teil 2

Zugriff auf Dateien / Filesystem

- **writeFile()** erstellt eine neue Datei und schreibt Inhalt
- **appendFile()** fügt an und erstellt ggf. Datei
- **rename()** benennt eine Datei um
- **unlink()** löscht eine Datei

```
fs.writeFile("newFile.txt", "Inhalt der\nDatei", (error) => {  
  if (error)  
    throw error;  
  console.log("file was saved");  
});
```



Inhalt der
Datei

Einstieg in Node.js Teil 2

Streams

- Streams sind Objekte, die kontinuierlichen Zugriff auf einen „Datenstrom“ erlauben
- Vier Typen: readable, writable, duplex (rw) und transform
- Streams sind gleichzeitig EventEmitter mit folgenden Events
 - data: Daten sind zum Lesen verfügbar
 - end: keine Daten mehr zum Lesen verfügbar
 - error: Fehler beim Lesen oder Schreiben der Daten
 - finish: alle Daten wurden ausgegeben



Einstieg in Node.js Teil 2

Streams

- Streams können gelesen, geschrieben, aneinandergehängt (chaining) oder der Output des einen als Input des anderen genutzt werden

```
var fs = require("fs");
var data = "";

var streamReader = fs.createReadStream("test.txt");

streamReader.setEncoding("UTF8");

streamReader.on("data", function(chunk) {
    data += chunk;
});

streamReader.on("end", function() {
    console.log(data);
});

streamReader.on("error", function(error) {
    console.log(error.stack);
});
```

Dies ist eine
Testdatei.

Error: ENOENT: no such file or directory, open 'c:\Progs\nodejs\hsh1\test_.txt'

Einstieg in Node.js Teil 2

Timeout und Interval

- **setTimeout(cb, ms)** erlaubt nach ms eine Funktion cb auszuführen
- **clearTimeout(timeout)** löscht eine definierte Timeout-Funktion
- **setInterval(cb, ms)** erlaubt alle ms eine Funktion cb auszuführen; ms maximal $2^{31} \sim 2,15$ Mrd
- **clearInterval(interval)** löscht eine definierte Interval-Funktion

```
function cb() {  
    console.log("CB called");  
}  
setTimeout(cb, 1000);  
  
var myTimeout = setTimeout(cb, 1000);  
clearTimeout(myTimeout);  
  
var myInterval = setInterval(cb, 1000);  
clearInterval(myInterval);
```

Einstieg in Node.js Teil 2

Process

- Globales Objekt **process** gewährt Zugriff auf aktuellen Node.js Prozess
- Ohne **require()** verfügbar
- **exit()** beendet Prozess kontrolliert und übergibt optionalen Statuscode, z.B. **process.exit(3)**;
- **kill(process.pid, "SIGKILL")** beendet Prozess mit ID sofort unkontrolliert
- **process.on()** erlaubt auf Prozess-Events zu reagieren

```
process.on("exit", (code) => {  
    console.log("process closed");  
});
```

Einstieg in Node.js Teil 2

Utility-Modul

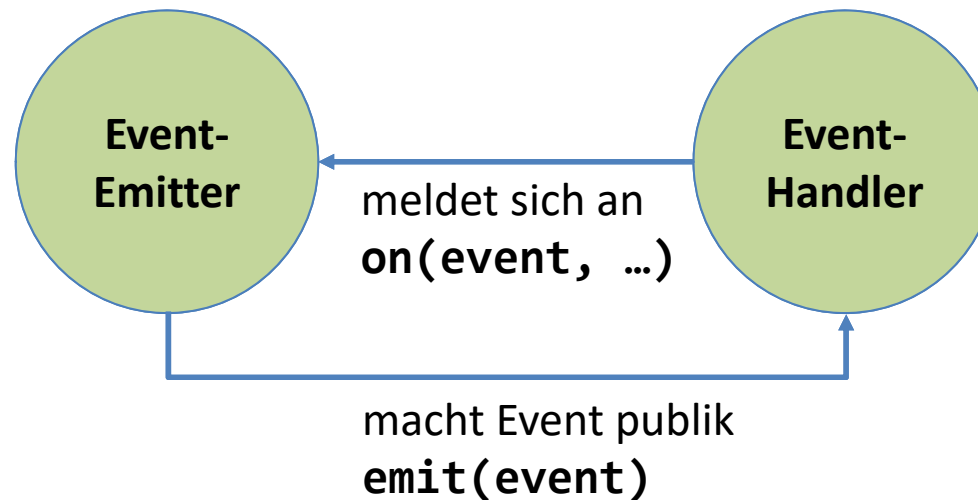
- Bietet viele nützliche Funktionen; sind aber teilweise deprecated
- **.format()** funktioniert wie **printf()** → formatiert Werte für die Ausgabe als Text
 - %s → String
 - %d → Zahl
 - %j → JSON
 - %% → Maskierung für %-Zeichen

```
var util = require("util");  
  
var s = util.format("Guten %s", "Tag");  
util.log(s);  
console.log(s);
```

```
1 Dec 16:40:27 - Guten Tag  
Guten Tag
```

Einstieg in Node.js Teil 2

Event-Loop / Event-Modul



```
var events = require("events");
var EventEmitter = new events.EventEmitter();

var eventHandler = function() {
  console.log("Event caught");
}

eventEmitter.on("myEvent", eventHandler);

eventEmitter.emit("myEvent");
```

Einstieg in Node.js Teil 2

Event-Loop / Event-Modul

- Einige weitere Event-Methoden
 - **.once("myEvent", eventHandler);** → eventHandler wird nur einmalig aufgerufen
 - **.removeListener("myEvent", eventHandler);** → entfernt den eventHandler für myEvent
 - **.removeAllListeners();** → entfernt alle eventHandler
 - **.removeAllListeners("myEvent");** → entfernt alle eventHandler für myEvent
 - **.listeners("myEvent");** → gibt eine Liste aller eventHandler für myEvent zurück



Überblick

- Das letzte Mal... und Lernziele
- Einstieg in Node.js Teil 2
- **ECMAScript 6**
- Zum Schluss...

ECMAScript 6

JavaScript-Sprachstandard ECMAScript 6

- Node.js basiert auf Googles JS V8-Engine
- Unterstützt ECMAScript 6 (2015)
- Gegenüber ECMAScript 5 viele neue Funktionen, z.B.
 - Zusätzliche String- und Array-Funktionen
 - Strict Mode
 - Accessor-Methoden
 - Object-Erweiterungen
 - JSON



ECMAScript 6

Einige neue Arrayfunktionen

Methode	Beschreibung
Array.from	Wandelt ein iterierbares Objekt in ein Array-Objekt um
Array.of	Fügt übergebene Argumente in ein neues Array ein
Array.prototype.fill	Füllt Array mit statischem Wert
Array.prototype.find	Sucht Element innerhalb eines Arrays und gibt es zurück
Array.prototype.findIndex	Sucht Element innerhalb eines Arrays und gibt seinen Index zurück
Array.prototype.entries	Liefert einen Iterator, der pro Durchlauf ein Key-Value-Paar zurückgibt (ähnlich foreach in PHP)
Array.prototype.keys	Liefert einen Iterator über die Keys eines Arrays
Array.prototype.copyWithin	Kopiert einen Teil eines Arrays an einer andere Stelle innerhalb des Arrays

ECMAScript 6

Einige neue Stringfunktionen

Methode	Beschreibung
<code>String.fromCharCode</code>	Wandelt Codepoints in reguläre Unicode-Zeichen um
<code>String.prototype.codePointAt</code>	Liefert den Codepoint des Zeichens an einer bestimmten Stelle einer Zeichenkette
<code>String.prototype.startsWith</code>	Prüft, ob eine Zeichenkette mit einer bestimmten Zeichenkette beginnt
<code>String.prototype.endsWith</code>	Prüft, ob eine Zeichenkette mit einer bestimmten Zeichenkette endet
<code>String.prototype.includes</code>	Prüft, ob eine bestimmte Zeichenkette in einer anderen Zeichenkette enthalten ist
<code>String.prototype.repeat</code>	Wiederholt eine bestimmte Zeichenkette n Mal
<code>String.prototype.normalize</code>	Normalisiert eine Zeichenkette

ECMAScript 6

Scope – Gültigkeitsbereiche von Variablen



- Standard-Scopes in JavaScript
 - global scope → Variable ist überall gültig
 - block scope (let) → Variable ist nur innerhalb des aktuellen Blocks gültig (z.B. if-Block)
block scoping funktioniert in Node.js nur, wenn dies im aktuellen Skript mit **‘use strict’**; aktiviert wurde
 - function scope → Variable ist nur innerhalb der Funktion gültig
- Node.js führt einen weiteren Gültigkeitsbereich, den module scope ein
 - innerhalb eines Moduls definierte Variablen sind dann nur innerhalb dieses Moduls gültig

ECMAScript 6

Klassen

- Um Klassen in JavaScript verwenden zu können, sollte der strict-Modus aktiviert werden
- Anders, als bei den bereits vorgestellten Konstruktoren, wird bei einer Klasse eine constructor-Methode innerhalb der Klasse definiert
- Zugriff auf Eigenschaften und Methoden erfolgt per Punktnotation

Deklaration einer Klasse

```
'use strict';  
  
class Circle {  
  let radius = 0;  
  let pi = 3.14159;  
  
  constructor (radius) {  
    this.radius = radius;  
  }  
  
  getArea() {  
    return this.pi * this.radius * this.radius;  
  }  
  
  getExtent() {  
    return this.pi * 2.0 * this.radius;  
  }  
}
```

Instanziierung Klasse und Zugriff auf Eigenschaften

```
var circle = new Circle(1.1);  
var circleArea = circle.getArea();
```

ECMAScript 6

Vererbung

- **extends**-Schlüsselwort gibt an, von welcher Klasse geerbt wird
- **MsgBus** erbt alle Methoden von **EventEmitter**
- Es kann immer nur von einer Klasse geerbt werden (Einfachvererbung)
- Allerdings kann über mehrere Kaskaden geerbt werden
- **super()** erlaubt Zugriff auf den Konstruktor der Elternklasse
- **super.** erlaubt Zugriff auf Elternfunktionen, z.B. **super.on()**

```
'use strict';
var EventEmitter = require('events').EventEmitter;

class MsgBus extends EventEmitter {

  constructor() {
    super();
    this.events = ['create', 'read', 'update', 'delete'];
  }

  on(event, listener) {
    if (this.events.indexOf(event) === -1) {
      throw new Error('Invalid event');
    }
    super.on(event, listener);
  }
}

var msgBus = new MsgBus();
msgBus.on('create', console.log);
msgBus.emit('create', 'Guten Tag');
```

ECMAScript 6

(Tagged) Template Strings

- Template Strings sind eine dritte String-Variante, die mit Backticks ` , anstatt doppelter oder einfacher Anführungszeichen arbeitet
- Template Strings werden anders, als „normale“ Strings geparsed
- Der Stringteil innerhalb von `\${...}` wird als JavaScript-Code interpretiert und das Ergebnis als Text in den String eingefügt
- In Kombination mit einer Tag-Funktion werden sie zu sog. Tagged Template Strings
- String-Array (**string**) enthält alle Stringteile zwischen den Ausdrücken
- Die weiteren Argumente (**hours**) enthalten die Ausdrücke des Template Strings

```
function getDayTime(string, hours) {  
  
    var dayTime = "";  
    if (hours < 12) {  
        dayTime = "Morgen";  
    }  
    else if (hours < 18) {  
        dayTime = "Tag";  
    }  
    else {  
        dayTime = "Abend";  
    }  
  
    return string[0] + dayTime;  
}  
  
console.log(getDayTime `Guten ${ (new Date).getHours() }`);  
console.log(getDayTime("Guten", (new Date).getHours()));
```


Aufruf ohne Template String

ECMAScript 6

Collections

- Neben den primitiven Typen und Objekttypen führt ECMAScript 6 neue zusammengesetzte Datentypen ein
 - Set ist eine Menge von beliebigen Value-Typen
 - Map ist ein einfacher Key-Value-Store

```
var myFunc = function() {};  
var myObj = {};  
var myStr = "Hallo";  
  
var map = new Map();  
map.set(myFunc, "Function");  
map.set(myObj, "Object");  
map.set(myStr, "String");  
  
console.log(`Size of map: ${map.size}`);  
for (let entry of map) {  
  console.log(`Key: ${entry[0]} Value: ${entry[1]}`);  
}
```



```
Size of map: 3  
Key: function() {} Value: Function  
Key: [object Object] Value: Object  
Key: Hallo Value: String
```


ECMAScript 6

Generators

- Ein Generator ist quasi eine Funktion, die mittendrin angehalten und zu einem späteren Zeitpunkt fortgesetzt wird
- Werden z.B. für asynchrone Ausführungen verwendet
- **function*** leitet den Generator ein
- Anstatt **return** wird **yield** verwendet; **yield** stoppt gleichzeitig die Ausführung
- **next()** setzt die Ausführung des Generators fort
- Rückgabewert des **yield**-Ausdrucks kann zugleich als Argument per **next()** übergeben werden

```
function* myGenerator() {
  var n = 4;
  n = yield "Ergebnis=1*["+n+"]=" + (1*n);
  n = yield "Ergebnis=1+["+n+"]=" + (1+n);
  n = yield "Ergebnis=["+n+"]*["+n+"]=" + (n*n);
}

var myGen = myGenerator();
console.log(myGen.next());
console.log(myGen.next(7));
console.log(myGen.next(9));
```



```
{ value: 'Ergebnis=1*[4]=4', done: false }
{ value: 'Ergebnis=1+[7]=8', done: false }
{ value: 'Ergebnis=[9]*[9]=81', done: false }
```

ECMAScript 6

Promises

- Promises bieten Handler, die den Umgang mit asynchronen Operationen vereinfachen
- Promise-Konstruktor enthält eine Callback-Funktion, die zwischen den Status **resolve** und **reject** unterscheidet

```
var fs = require("fs");

function readFile(filename) {
  return new Promise(function(resolve, reject) {
    fs.readFile(filename, 'utf-8', function(err, data) {
      if (err) {
        reject(err);
      }
      else {
        resolve(data);
      }
    });
  });
}

readFile("test.txt").then(function success(data) {
  console.log("Dateiinhalte: ", data);
}, function failure(error) {
  console.log("Fehler: ", error.message);
});
```

- Insgesamt gibt es vier Status
 - **pending**: Operation läuft noch
 - **settled**: Operation wurde beendet
 - **resolved**: Operation wurde erfolgreich beendet
 - **rejected**: Operation ist fehlgeschlagen
- **.then()** definiert, wie auf die Rückmeldung des Promise reagiert werden soll

Überblick

- Das letzte Mal... und Lernziele
- Einstieg in Node.js Teil 2
- ECMAScript 6
- **Zum Schluss...**

Folgendes sollten Sie nun (beantworten) können:



- Mit welcher Funktion werden externe Module in ein Node.js-Skript eingebunden?
- Wie kann ein Timeout gesetzt und gelöscht werden?
- Erläutern Sie den Event-Loop. Welche Aufgaben haben Event-Handler und Event-Emitter?
- Welche Bedeutung hat der module scope?
- Mit welchem Schlüsselwort ist eine Vererbung möglich?
- Was machen die Anweisungen **super()** bzw. **super.?**

Zum Schluss...

Weiterführende Links und Literatur

[Node.js] <https://nodejs.org/en/>

[Node.js API-Beschreibung] <https://nodejs.org/api/>

[Tutorial für Node.js] <http://www.w3ii.com/nodejs/default.html>

[npm] <https://www.npmjs.com/>

[ECMAScript 6] <http://es6-features.org/>

Golo Roden: „Node.js & Co: Skalierbare, hochperformante und echtzeitfähige Webanwendungen professionell in JavaScript entwickeln“, dpunkt.Verlag GmbH, 1. Auflage, 2012.

Sebastian Springer: „Node.js: Das Praxisbuch“, Rheinwerk Computing, 2. Auflage, 2016.

Zum Schluss...

Quellen

[sonstige Abbildungen] <https://pixabay.com> und <https://icon-icons.com>

Vielen Dank für Ihr Interesse!

