

Web-Anwendungen: Rest-Implementierung

Web-Engineering II

Prof. Dr. Sebastian von Klinski

Einführung Node.js

Umsetzung von REST-Services

- Gängige Stacks
 - Node.js + express.js
 - Spring Boot
 - Django
 - ASP.NET
 - Ruby on Rails
 - Etc.
- Derzeit am weitesten Verbreitet
 - Node.js + express.js
 - Spring Boot (eBusiness, Java)

Node.js

- Plattform basierend auf Google Chrome's JavaScript Runtime
- Plattformunabhängig (Windows, Linux, etc.)
- Anwendungen werden in JavaScript geschrieben
- Single-Threaded-Server!
- Wo sollte Node.js nicht verwendet werden?
 - Es können keine “Threads” erzeugt werden, um rechenintensive Operationen auszulagern
 - Es können ausschließlich IO-Operationen in asynchrone OS-Aufrufe (Events) verschoben werden
 - **Aufwändige Berechnungen blockieren alle Anfragen!**
 - Nicht für CPU-intensive Anwendungen geeignet
- Populär, weil Server-Umsetzung schnell und einfach

Node.js: Module

- Installation über Installer
- Aufruf über „node“
- npm – Package Manager für node.js (Download von benötigten Modulen)
- Starten von Anwendungen (JavaScript-Datei) per Terminal
- Anwendungen sind JavaScript-Dateien
- Beispiel: starten von app.js

```
node app.js
```

Weitere Infos: https://www.w3schools.com/nodejs/nodejs_npm.asp

Node.js: Module

- Durch Nutzung von vorhandenen Modulen können eigene Programme häufig sehr klein bleiben
- Insbesondere Prototypen lassen sich sehr schnell umsetzen
- Beispiel: Web-Server, der auf alle Anfragen „Hello World“ zurück gibt:

```
var http = require('http');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);
```

Weitere Infos: https://www.w3schools.com/nodejs/nodejs_npm.asp

Node.js: Module

- Module werden durch Schlüsselwort „require“ geladen
- Require führt die folgenden Schritte aus
 - Laden der JavaScript-Datei
 - Ausführen der Datei
 - Zurückgeben des Export-Objekts
- Beispiel: Modul „fs“ für das Nutzen von lokalen Dateien

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

Node.js: File System

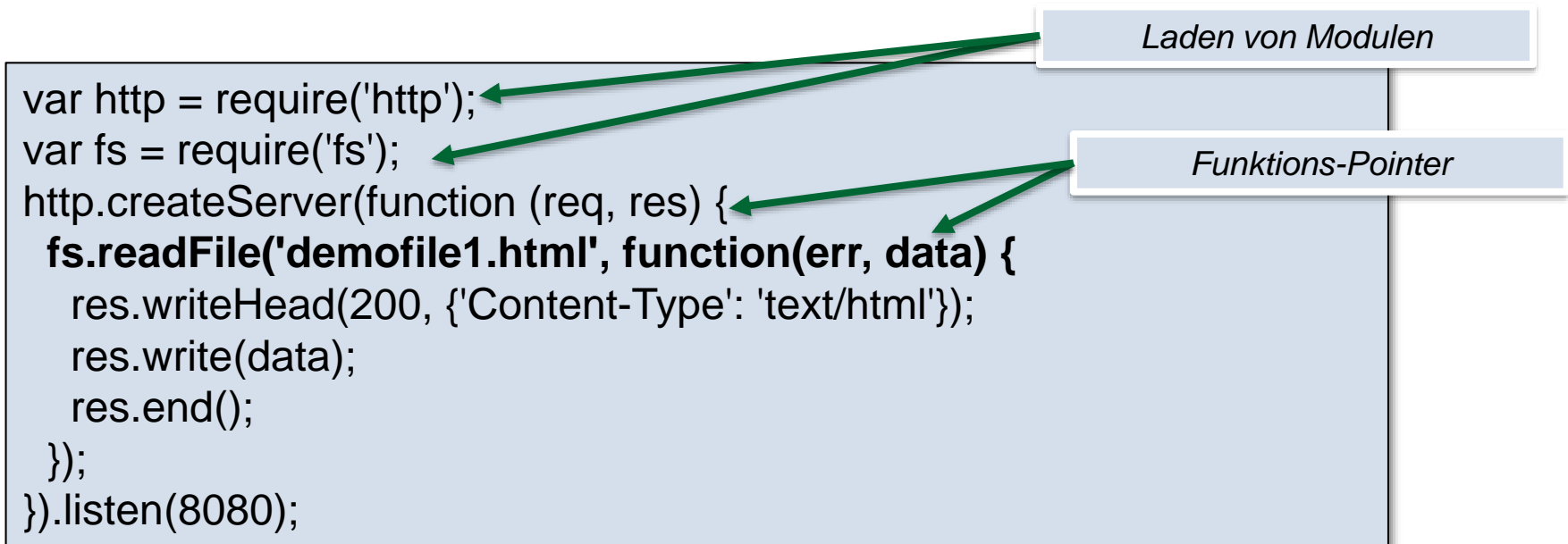
- Modul „fs“ exportiert beispielsweise unter anderem die folgenden Funktionen
 - `appendFile()`
 - `open()`
 - `writeFile()`
 - `unlink()` (Löschen)
 - `rename()`

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

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err)
    throw err;
  console.log('Saved!');
});
```


Node.js: HTTP-Server

- **Beispiel 2:** Web-Server, der eine HTML-Datei zurückgibt
- Module http (HTTP-Server) und fs (File System)
- Viele Module versuchen Dead-Locks durch asynchrone Callback-Methoden zu vermeiden
- Auch in diesem Beispiel: Schachtelung von Funktions-Pointern



Node.js: asynchrone Aufrufe

- Da Node.js single-threaded ist, blockieren synchrone Aufrufe den Thread
- Zur Performance-Optimierung sollten lang laufende Aufrufe asynchron ausgeführt werden
- „fs“ bietet Funktion für synchrones und asynchrones Laden per Callback an

```
var fs = require("fs");  
var data = fs.readFileSync('input.txt');
```

synchron

```
console.log(data.toString());  
console.log("Read file 1");
```

```
fs.readFile('input.txt', function (err, data) {  
  if (err) return console.error(err);  
  console.log(data.toString());  
});
```

asynchron

```
console.log("Final End");
```

Node.js: asynchrone Aufrufe

Verwendung von Callback-Methode

```
var fs = require("fs");  
var data = fs.readFileSync('input.txt'); ← synchron  
  
console.log(data.toString());  
  
fs.readFile('input.txt', function (err, data) {  
  if (err) return console.error(err);  
  console.log(data.toString()); ← asynchron  
  });  
  
console.log("Final End");
```

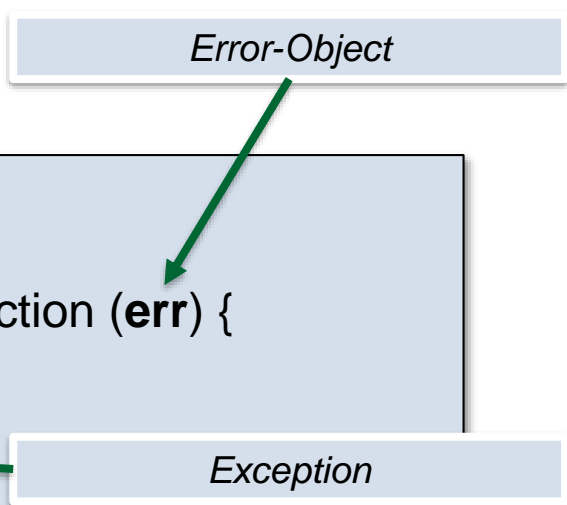
```
S C:\Beuth-Temp\vorlesungen-test\test01> node .\loadFile.js  
Das ist der Dateiinhalt  
Final End  
Das ist der Dateiinhalt  
PS C:\Beuth-Temp\vorlesungen-test\test01>
```

Node.js: Errors und Exceptions

- Häufig wird den Callback-Funktionen ein Error-Object übergeben, falls ein Fehler aufgetreten ist.
- Falls es keine Fehler gab, ist es null/ undefined.
- Beachte: Es gibt einen Unterschied zwischen Errors und Exceptions.

```
var fs = require('fs');  
  
fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {  
  if (err)  
    throw err;  
  console.log('Saved!');  
});
```

Error-Object



Exception

Node.js: Errors und Exceptions

- Error ist Instanz von Error-Klasse
- Wird instanziiert

```
var error = new Error('something bad happened')
```

- Exception wirft Error-Objekt
- Eine Error-Objekt, das geworfen wird, ist eine Exception
- Diese muss gefangen werden oder führt zum Abbruch des Prozesses

```
throw new Error('something bad happened');
```

- In Node.js werden selten Exceptions geworfen
- In der Regel werden Error-Objekte an die Callback-Funktion gegeben

```
callback(new Error('something bad happened'));
```

Quelle: <https://www.joyent.com/node-js/production/design/errors>

Grundlagen: Möglichkeiten des Errors-Handling

- Individuelle Lösung des Problems
 - Z.B. Verbindung zu einem Server kann nicht aufgebaut werden → Lokale oder gecachte Daten verwenden
- Weiterleiten der Fehlernachricht an den Client
 - Beispielsweise eine Nachricht an den Nutzer
 - Diese sollte jedoch verständlich für den Endnutzer sein
- Aktion nochmal versuchen
 - Wird nur in wenigen Fällen erfolgreich sein
 - Es darf nicht zur Endlosschleife werden
- Exception werfen
 - Das kann zum Absturz des Prozesses führen, wenn die Exception nicht gefangen wird
- Fehler loggen
 - Ist gegebenenfalls in jedem Fall sinnvoll
- Nicht tun → Absturz

Grundlagen: Möglichkeiten des Errors-Handling

- Vorgehensweise bei Fehlern muss individuell festgelegt werden
- Wichtig sind folgende Aspekte
 - Auf Client-Seite muss ein definierter Umgang mit Fehlern möglich sein
 - Der Server als Ganzes darf nicht abstürzen oder in der Bereitstellung seines Dienstes beeinträchtigt werden.
- Einzelne Anfragen können durchaus abstürzen, wenn der Client damit umgehen kann.
 - Das sollte aber nicht unbemerkt bleiben.
 - Das geht nur, wenn dadurch die Client-Session nicht beeinträchtigt wird (zustandslose Services)

Node.js: Events

- In Node.js-Anwendungen können asynchrone Events versandt und empfangen werden
- Setzt Observer-Pattern um

```
var events = require('events');

var EventEmitter = new events.EventEmitter();

//Create an event handler:
var myEventHandler = function () {
  console.log('I received the test!');
}

//Assign the event handler to an event:
eventEmitter.on('testEvent', myEventHandler);

//Fire the 'testEvent' event:
eventEmitter.emit('testEvent');
```

Module in Node.js

Node.js: Module im Detail

- Wesentlich Grundlage für die Popularität von Node.js sind die zahllosen fertigen Module, die in der Regel kostenlos genutzt werden können
- Module kapseln zusammengehörige Funktionskomplexe und/ oder Datenstrukturen
- Beispiel: http ist ein Kernmodul von Node.js um einen einfachen Web-Server umzusetzen

```
var http = require('http');
```

- Starten des Servers

```
const server = http.createServer((req, res) => {  
  //handle every single request with this callback  
})
```

Node.js: Module

- Zum Erstellen eines eigenen Moduls kann eine normal JavaScript-Datei erstellt werden
- In der Datei können Funktionen oder Objekte erstellt und zur Verfügung gestellt werden
- Damit Objekte genutzt werden können, müssen sie exportiert werden

myModule.js

```
exports.myDateTime = function () {  
  return Date();  
};
```

- module.exports oder exports ist ein spezielles Objekt in allen JS-Dateien in Node.js
- Es wird von der require-Funktion zurückgegeben

Node.js: Module

- Laden des Moduls über „require“
- Das zurückgegebene Objekt ermöglicht Zugriff auf exportierte Objekte

```
var http = require('http');  
var dt = require('./myModule');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write("The date and time are currently: " + dt.myDateTime());  
  res.end();  
}).listen(8080);
```

Node.js: Module

- Auslagern von Objekten

Person.js

```
module.exports = function (firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  this.fullName = function () {  
    return this.firstName + ' ' + this.lastName;  
  }  
}
```

- Nutzen des Objekts

```
var person = require('./Person');  
var person1 = new person('James', 'Bond');  
console.log(person1.fullName());
```

Node.js: Installation von Modulen

- Beispiel: formidable ist ein Modul zum Parsen von Formulardaten, insbesondere zur Umsetzung von File-Uploads

```
var formidable = require('formidable');
```

- Sollte ein Modul fehlen, kann es über npm install [modul] geladen werden
- NPM ist ein Paketmanager für Node.js und umfasst derzeit 350.000 Module

```
PS C:\Beuth-Temp\vorlesungen-test\test01> npm install formidable  
npm notice created a lockfile as package-lock.json. You should commit this file.
```

```
+ formidable@1.2.2  
added 1 package and audited 1 package in 1.699s
```

```
1 package is looking for funding  
run `npm fund` for details
```

```
found 0 vulnerabilities
```

Node.js: File Uploads

- File Upload über das Modul Formidable

```
var http = require('http');
var formidable = require('formidable');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.on('fileBegin', function(name, file) {
      file.path = 'c:/temp/uploads/' + file.name;
    });
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="fileupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Parsen von Datei

*Formular zum Hochladen von
Datei*

Node.js: File Uploads, Datei umbenennen

- Wenn Datei nicht umbenannt wird, wird sie in einem temporären Verzeichnis abgelegt
- Pfad ist in „files“ abrufbar

```
...
if (req.url == '/fileupload') {
  var form = new formidable.IncomingForm();
  form.parse(req, function (err, fields, files) {
    var oldpath = files.fileupload.path;
    var newpath = 'C:/Users/Your Name/' + files.fileupload.name;
    fs.rename(oldpath, newpath, function (err) {
      if (err) throw err;
      res.write('File uploaded and moved!');
      res.end();
    });
  });
}
...
```

Gängige Module

Node.js: URL Module

- Zur Umsetzung von Web-Seiten kann beispielsweise das Modul „url“ verwendet werden
- Es erlaubt die Auswertung von URLs

```
var url = require('url');  
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';  
var q = url.parse(adr, true);  
  
console.log(q.host); //returns 'localhost:8080'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?year=2017&month=february'  
  
var qdata = q.query; //returns an object: { year: 2017, month: 'february' }  
console.log(qdata.month); //returns 'february'
```

Node.js: URL Module

- `http://localhost:8080/summer.html` → Lädt `summer.html` und gibt das zurück

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Node.js: Email

```
var nodemailer = require('nodemailer');

var transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'youremail@gmail.com',
    pass: 'yourpassword'
  }
});

var mailOptions = {
  from: 'youremail@gmail.com',
  to: 'myfriend@yahoo.com',
  subject: 'Sending Email using Node.js',
  text: 'That was easy!'
};

transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});
```

Node.js: REST-Service mit HappyJS

- Hapi: Modul zur Umsetzung von Web-Anwendungen, REST-APIs und Services

```
const Hapi = require('hapi');
const host = 'localhost';
const port = 3000;
const server = Hapi.Server({
  host: host,
  port: port
});
const init = async () => {
  await server.start();
  console.log("Server up and running at port: " + port);
}

//Setup the routes
require('./routes/routes')(server);

init();
```

Node.js: REST-Service mit HappyJS

- Z.B. Umsetzung mit Happy JS (hapi.js)
- Endpoint

```
module.exports = function(server) {  
  ...  
  server.route({  
    method: 'GET',  
    path: '/calculator/add/{num1}/{num2}',  
    handler: function (request, h) {  
  
      const num1 = parseInt(request.params.num1);  
      const num2 = parseInt(request.params.num2);  
  
      var data = {  
        answer: num1 + num2  
      };  
  
      return data;  
    }  
  });  
  ...  
}
```