

JavaScript-basierte Entwicklung und SAP HANA

Ausgewählte Datenbankkonzepte/-techniken

Prof. Dr. Ingo Claßen

Hochschule für Technik und Wirtschaft Berlin

Asynchrone Funktionen

Promises

JavaScript und Hana

Web Services / HTTP Server

Quellen

Asynchrone Funktionen

- ▶ Aufruf von Funktionen ohne Blockierung des Aufrufers
- ▶ Kein Warten auf die Antwort der aufgerufenen Funktion
- ▶ Zeitliche Versetzung der Ergebnisbereitstellung
- ▶ Typische Beispiele: File IO, DB-Aufrufe, Aufruf Web Services
- ▶ f1: Platzhalter für diese Arten von Funktionen

```
function f1(cb) {  
  setTimeout(() => {  
    console.log("f1 erledigt (nach 5 sek)");  
    cb(null, 1);  
  }, 5000 + utils.randomIntInclusive(0, 100));  
}
```

- ▶ Callback-Funktion cb: (err, result) => body

Asynchrone Ausführungsreihenfolge

```
f1((err1, result1) => console.log(result1));  
f2((err2, result2) => console.log(result2));  
f3((err3, result3) => console.log(result3));  
f4((err4, result4) => console.log(result4));  
  
console.log("main läuft weiter");
```

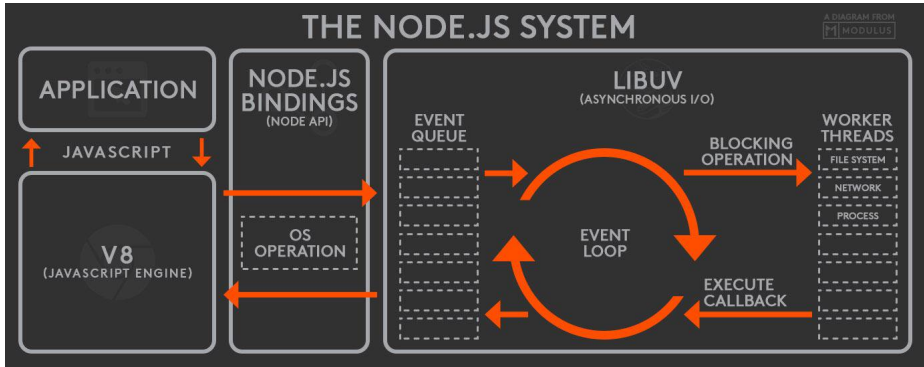
Erster Durchlauf

```
main läuft weiter  
f2 erledigt (nach 5 sek)  
2  
f1 erledigt (nach 5 sek)  
1  
f4 erledigt (nach 5 sek)  
4  
f3 erledigt (nach 5 sek)  
3  
  
[Done] exited with code=0 in 5.223 seconds
```

Erneuter Durchlauf

```
main läuft weiter  
f2 erledigt (nach 5 sek)  
2  
f3 erledigt (nach 5 sek)  
3  
f4 erledigt (nach 5 sek)  
4  
f1 erledigt (nach 5 sek)  
1  
  
[Done] exited with code=0 in 5.197 seconds
```

Node.js-System



Synchrone Ausführungsreihenfolge – Callback Hell

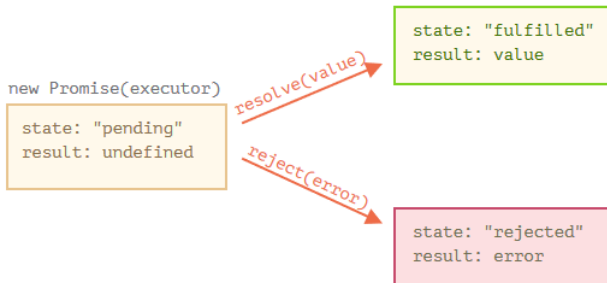
```
f1((err1, result1) =>
  f2((err2, result2) =>
    f3((err3, result3) =>
      f4((err4, result4) =>
        console.log(result1 + result2 + result3 + result4)
      )
    )
  )
);
console.log("main läuft weiter");
```

```
main läuft weiter
f1 erledigt (nach 5 sek)
f2 erledigt (nach 5 sek)
f3 erledigt (nach 5 sek)
f4 erledigt (nach 5 sek)
10
```

```
[Done] exited with code=0 in 20.17 seconds
```

Promises

```
let promise = new Promise(function (resolve, reject) {  
  // Anwendungscode  
  // Aufruf von resolve(value)  
  // oder  
  // Aufruf von reject(error)  
});
```



Promises Ausführung

```
let promise1 = new Promise((resolve, reject) => {
  setTimeout(() => resolve("done1"), 1000 + utils.randomIntInclusive(0, 100))
});
let promise2 = new Promise((resolve, reject) => {
  setTimeout(() => reject(new Error("Error")), 1000 + utils.randomIntInclusive(0, 100))
});

console.log(promise1);
console.log(promise2);

promise1.then(
  result => console.log('ok: promise1: ' + result),
  error => console.log('error: promise1')
);
promise2.then(
  result => console.log('ok: promise2: ' + result),
  error => console.log('error: promise2')
);
console.log("main läuft weiter");
```

```
Promise { <pending> }
Promise { <pending> }
main läuft weiter
error: promise2
ok: promise1: done1

[Done] exited with code=0
in 1.197 seconds
```

Asynchrone Ausführung

► Entkopplung von Callback-Funktion

```
let promise1 = new Promise((resolve, reject) => {
  setTimeout(() => resolve("done1"), 1000 + utils.randomIntInclusive(0, 100))
});

let promise2 = new Promise((resolve, reject) => {
  setTimeout(() => resolve("done2"), 1000 + utils.randomIntInclusive(0, 100))
});

promise1.then(console.log);
promise2.then(console.log);
console.log("main läuft weiter");
```

Erster Durchlauf

```
main läuft weiter
done2
done1
```

[Done] exited with code=0 in 1.219 seconds

Erneuter Durchlauf

```
main läuft weiter
done1
done2
```

[Done] exited with code=0 in 1.17 seconds

Synchronisierung 1 – async/await

- ▶ await geht nur in async-Funktionen

```
async function deterministic1() {  
  let promise1 = new Promise((resolve, reject) => {  
    setTimeout(() => resolve("done1"), 1000 + utils.randomIntInclusive(0, 100))  
  });  
  let promise2 = new Promise((resolve, reject) => {  
    setTimeout(() => resolve("done2"), 1000 + utils.randomIntInclusive(0, 100))  
  });  
  let msg1 = await promise1;  
  console.log(msg1);  
  let msg2 = await promise2;  
  console.log(msg2);  
}  
deterministic1()
```

```
main läuft weiter  
done1  
done2
```

```
[Done] exited with code=0 in 1.192 seconds
```

- ▶ Promise 1 und 2 laufen asynchron
- ▶ Ausgabe der Ergebnisse synchronisiert

Synchronisierung 2 – async/await

```
async function deterministic2() {  
  let promise1 = new Promise((resolve, reject) => {  
    setTimeout(() => resolve("done1"), 1000 + utils.randomIntInclusive(0, 100))  
  });  
  let msg1 = await promise1;  
  console.log(msg1);  
  let promise2 = new Promise((resolve, reject) => {  
    setTimeout(() => resolve("done2"), 1000 + utils.randomIntInclusive(0, 100))  
  });  
  let msg2 = await promise2;  
  console.log(msg2);  
}  
deterministic2()  
console.log("main läuft weiter");
```

```
main läuft weiter  
done1  
done2
```

```
[Done] exited with code=0 in 2.283 seconds
```

- Erst läuft Promise 1 dann Promise 2
- Doppelte Ausführungszeit!

Promisification

```
const f1p = util.promisify(f1);  
const f2p = util.promisify(f2);  
const f3p = util.promisify(f3);  
const f4p = util.promisify(f4);
```

```
async function doit() {  
  const result1 = await f1p();  
  const result2 = await f2p();  
  const result3 = await f3p();  
  const result4 = await f4p();  
  console.log(result1 + result2 + result3 + result4);  
}  
doit().catch(err => console.log(err));  
console.log("main läuft weiter");
```

```
main läuft weiter  
f1 erledigt (nach 5 sek)  
f2 erledigt (nach 5 sek)  
f3 erledigt (nach 5 sek)  
f4 erledigt (nach 5 sek)  
10
```

```
[Done] exited with code=0 in 20.436 seconds
```

Datenbanktabelle

```
create table kv (  
    key integer not null primary key,  
    value varchar(100) not null  
);
```

key		value
1		'value1'
2		'value2'
3		'value3'
4		'value4'

Callback-Variante

```

const config = require('../a_config/config');
const hdbext = require('@sap/hdbext');

hdbext.createConnection(config.hdb, (error, connection) => {
  if (error) {
    return console.error("Connection error: ", error);
  }
  if (connection) {
    const sql = "select * from kv where key<?";
    const stmt = connection.prepare(sql);
    stmt.exec([4], (error, rows) => {
      console.log(rows);
      connection.disconnect(() => console.log("disconnected"));
      if (error) {
        return console.error("SQL execute error: ", error);
      }
    });
  }
});

console.log("main läuft weiter");

```

```

main läuft weiter
[ { KEY: 1, VALUE: 'value1' },
  { KEY: 2, VALUE: 'value2' },
  { KEY: 3, VALUE: 'value3' } ]
disconnected

```

Promisified Variante

```
const config = require('../a_config/config');
const Hdb = require('../a_mod/hdb');

async function doit() {
  const connection = await Hdb.createConnection(config.hdb);
  const hdb = new Hdb(connection);
  const sql = "select * from kv where key<?";
  const pstmt = await hdb.preparePromisified(sql);
  const rows = await hdb.statementExecPromisified(pstmt, [4]);
  await connection.disconnect(() => console.log("disconnected"));
  console.log(rows);
}

doit().catch(console.log);
console.log("main läuft weiter");
```

Hdb Klasse (1)

```

module.exports = class {

    static createConnection(options) {
        return new Promise((resolve, reject) => {
            const hdbext = require("@sap/hdbext");
            hdbext.createConnection(options, (error, connection) => {
                if (error) {
                    reject(error);
                } else {
                    resolve(connection);
                }
            });
        });
    }

    constructor(connection) {
        this.connection = connection;
        this.util = require("util");
        this.connection.promisePrepare = this.util.promisify(this.connection.prepare);
    }

    ...
}

```

Hdb Klasse (2)

```
module.exports = class {  
  ...  
  
  statementExecPromisified(statement, parameters) {  
    statement.promiseExec = this.util.promisify(statement.exec);  
    return statement.promiseExec(parameters);  
  }  
  
  statementExecBatchPromisified(statement, parameters) {  
    statement.promiseExec = this.util.promisify(statement.execBatch);  
    return statement.promiseExec(parameters);  
  }  
  
  loadProcedurePromisified(hdbext, schema, procedure) {  
    hdbext.promiseLoadProcedure = this.util.promisify(hdbext.loadProcedure);  
    return hdbext.promiseLoadProcedure(this.connection, schema, procedure);  
  }  
  
  ...  
}
```


DB Op

```
// Executes a database operation  
// Returns an array of resulting rows in case of select  
// Example: [ { KEY: 1, VALUE: 'value1' }, { KEY: 2, VALUE: 'value2' } ]  
// Returns number of effected rows in case of insert, update, delete  
dbOp: async function (connectionParams, sql, sqlParams = []) {  
  const Hdb = require('../a_mod/hdb');  
  const connection = await Hdb.createConnection(connectionParams);  
  const hdb = new Hdb(connection);  
  const pstmt = await hdb.preparePromisified(sql);  
  const result = await hdb.statementExecPromisified(pstmt, sqlParams);  
  await connection.disconnect();  
  return result;  
}
```

DB Op Select

```
const config = require('../a_config/config');
const utils = require('../a_mod/utils');

async function doit() {
  const sqlSelectAll = "select * from kv";
  const result = await utils.dbOp(config.hdb, sqlSelectAll);
  console.log(result);
}
```

Textanalytics

```
var values = {
  DOCUMENT_TEXT: 'Ich will in die Treskowallee 8 fahren.',
  LANGUAGE_CODE: 'DE',
  CONFIGURATION: 'EXTRACTION_CORE',
  RETURN_PLAINTEXT: 0
};

hdbext.createConnection(config.hdb, (err, connection) => {
  if (err) {
    return console.error("Connection error", err);
  }

  ta.analyze(values, connection, function done(err, parameters, rows) {
    if (err) { return console.error('error', err); }
    console.log(rows);
  });
});
```

```
[ { RULE: 'Entity Extraction',
  COUNTER: 1,
  TOKEN: 'Treskowallee 8',
  TYPE: 'ADDRESS1',
  NORMALIZED: null,
  STEM: null,
  PARAGRAPH: 1,
  SENTENCE: 1,
  OFFSET: 16,
  PARENT: null,
  EXPANDED_TYPE: null } ]
```

Web Framework Express

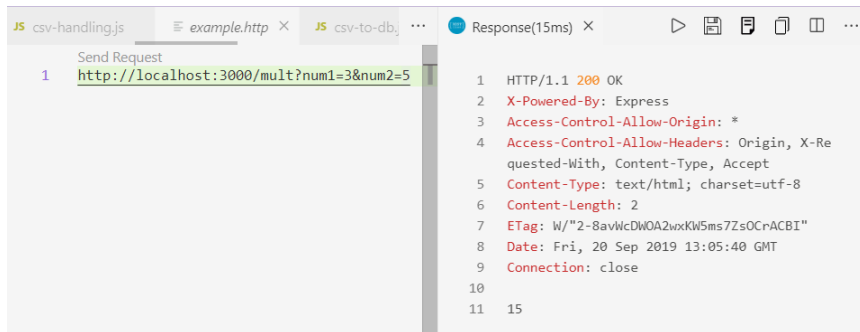
```
const express = require('express')
const app = express()
const port = 3000

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept");
  next();
});

app.get('/mult', (req, res, next) => {
  res.send(`${req.query.num1 * req.query.num2}`);
});

app.listen(port, () =>
  console.log(`Example app listening on port ${port}!`))
```

VS Code – Rest Client



The screenshot shows the VS Code interface with a REST client extension. The left pane shows a file named `example.http` with a single line of a REST client request. The right pane shows the response of the request, which is an HTTP 200 OK status with various headers and a body containing the number 15.

```
JS csv-handling.js  example.http X  JS csv-to-db.js ...  Response(15ms) X
```

Send Request

```
1 http://localhost:3000/mult?num1=3&num2=5
```

```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
5 Content-Type: text/html; charset=utf-8
6 Content-Length: 2
7 ETag: W/"2-8avWcDWOA2wxKW5ms7ZsOCrACBI"
8 Date: Fri, 20 Sep 2019 13:05:40 GMT
9 Connection: close
10
11 15
```

Quellen

- ▶ The Modern JavaScript Tutorial
<http://javascript.info/>
- ▶ SAP HANA Platform - Help Portal
https://help.sap.com/viewer/product/SAP_HANA_PLATFORM/2.0.04/en-US
- ▶ The Node.JS Sytem
<https://mobile.twitter.com/TotesChaoticMeh/status/494959181871316992>