Kostenloses eBook

LERNEN

Node.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#node.js

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: node-js

It is an unofficial and free Node.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Node.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Node.js

Bemerkungen

Node.js ist ein ereignisbasiertes, nicht blockierendes, asynchrones E / A-Framework, das die V8-JavaScript-Engine von Google verwendet. Es wird für die Entwicklung von Anwendungen verwendet, die die Möglichkeit verwenden, JavaScript sowohl auf dem Client als auch auf dem Server auszuführen, und daher von der Wiederverwendbarkeit von Code und der fehlenden Kontextumschaltung profitieren. Es ist Open Source und plattformübergreifend. Node.js-Anwendungen werden in reinem JavaScript geschrieben und können in der Node.js-Umgebung unter Windows, Linux usw. ausgeführt werden.

Versionen

Ausführung	Veröffentlichungsdatum
v8.2.1	2017-07-20
v8.2.0	2017-07-19
v8.1.4	2017-07-11
v8.1.3	2017-06-29
v8.1.2	2017-06-15
v8.1.1	2017-06-13
v8.1.0	2017-06-08
v8.0.0	2017-05-30
v7.10.0	2017-05-02
v7.9.0	2017-04-11
v7.8.0	2017-03-29
v7.7.4	2017-03-21
v7.7.3	2017-03-14
v7.7.2	2017-03-08
v7.7.1	2017-03-02
v7.7.0	2017-02-28

Ausführung	Veröffentlichungsdatum
v7.6.0	2017-02-21
v7.5.0	2017-01-31
v7.4.0	2017-01-04
v7.3.0	2016-12-20
v7.2.1	2016-12-06
v7.2.0	2016-11-22
v7.1.0	2016-11-08
v7.0.0	2016-10-25
v6.11.0	2017-06-06
v6.10.3	2017-05-02
v6.10.2	2017-04-04
v6.10.1	2017-03-21
v6.10.0	2017-02-21
v6.9.5	2017-01-31
v6.9.4	2017-01-05
v6.9.3	2017-01-05
v6.9.2	2016-12-06
v6.9.1	2016-10-19
v6.9.0	2016-10-18
v6.8.1	2016-10-14
v6.8.0	2016-10-12
v6.7.0	2016-09-27
v6.6.0	2016-09-14
v6.5.0	2016-08-26
v6.4.0	2016-08-12

Ausführung	Veröffentlichungsdatum
v6.3.1	2016-07-21
v6.3.0	2016-07-06
v6.2.2	2016-06-16
v6.2.1	2016-06-02
v6.2.0	2016-05-17
v6.1.0	2016-05-05
v6.0.0	2016-04-26
v5.12.0	2016-06-23
v5.11.1	2016-05-05
v5.11.0	2016-04-21
v5.10.1	2016-04-05
v5.10	2016-04-01
v5.9	2016-03-16
v5.8	2016-03-09
v5.7	2016-02-23
v5.6	2016-02-09
v5.5	2016-01-21
v5.4	2016-01-06
v5.3	2015-12-15
v5.2	2015-12-09
v5.1	2015-11-17
v5.0	2015-10-29
v4.4	2016-03-08
v4.3	2016-02-09
v4.2	2015-10-12

Ausführung	Veröffentlichungsdatum
v4.1	2015-09-17
v4.0	2015-09-08
io.js v3.3	2015-09-02
io.js v3.2	2015-08-25
io.js v3.1	2015-08-19
io.js v3.0	2015-08-04
io.js v2.5	2015-07-28
io.js v2.4	2015-07-17
io.js v2.3	13.06.2015
io.js v2.2	2015-06-01
io.js v2.1	2015-05-24
io.js v2.0	2015-05-04
io.js v1.8	2015-04-21
io.js v1.7	2015-04-17
io.js v1.6	2015-03-20
io.js v1.5	2015-03-06
io.js v1.4	2015-02-27
io.js v1.3	2015-02-20
io.js v1.2	2015-02-11
io.js v1.1	2015-02-03
io.js v1.0	2015-01-14
v0.12	2016-02-09
v0.11	2013-03-28
v0,10	2013-03-11
v0,9	2012-07-20

Ausführung	Veröffentlichungsdatum
v0,8	2012-06-22
v0,7	2012-01-17
v0.6	2011-11-04
v0,5	2011-08-26
v0.4	2011-08-26
v0.3	2011-08-26
v0,2	2011-08-26
v0.1	2011-08-26

Examples

Hallo Welt HTTP-Server

Installieren Sie zunächst Node.js für Ihre Plattform.

In diesem Beispiel erstellen wir einen HTTP-Server, der auf Port 1337 überwacht wird und der `Hello, World!` an den Browser. Beachten Sie, dass Sie anstelle von Port 1337 eine beliebige Portnummer Ihrer Wahl verwenden können, die derzeit von keinem anderen Dienst verwendet wird.

Der `http` - Modul ist ein Node.js ***Core*** - ***Modul*** (ein Modul in Node.js der Quelle enthält, die keine zusätzliche Ressourcen Installation erfordert). Das `http` Modul bietet die Funktionalität zum Erstellen eines HTTP-Servers mit der `http.createServer()` Methode. Erstellen Sie zum Erstellen der Anwendung eine Datei mit dem folgenden JavaScript-Code.

```
const http = require('http'); // Loads the http module

http.createServer((request, response) => {

    // 1. Tell the browser everything is OK (Status code 200), and the data is in plain text
    response.writeHead(200, {
        'Content-Type': 'text/plain'
    });

    // 2. Write the announced text to the body of the page
    response.write('Hello, World!\n');

    // 3. Tell the server that all of the response headers and body have been sent
    response.end();

}).listen(1337); // 4. Tells the server what port to be on
```

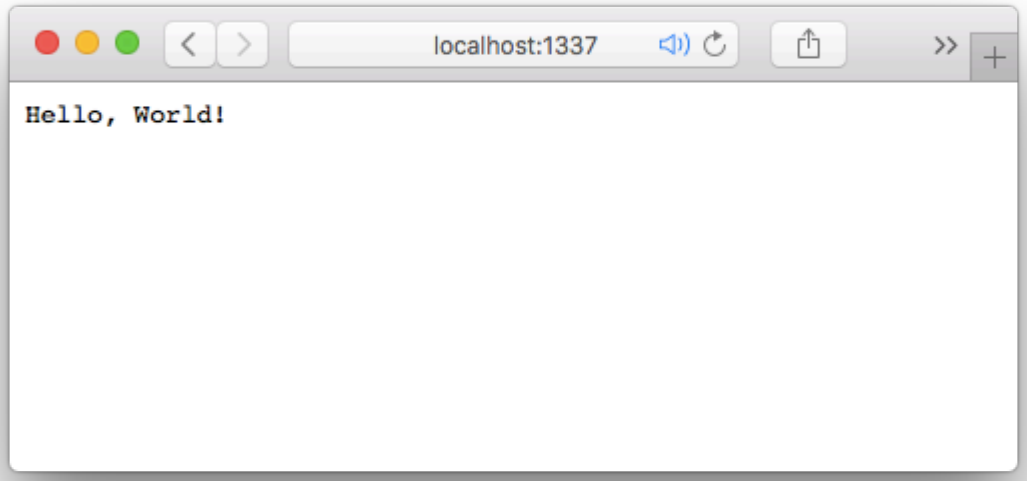
Speichern Sie die Datei mit einem beliebigen Dateinamen. Wenn wir es `hello.js` , können Sie die

Anwendung ausführen, indem Sie in das Verzeichnis `hello.js` , in dem sich die Datei befindet, und den folgenden Befehl verwenden:

```
node hello.js
```

Auf den erstellten Server kann dann mit der URL http: // localhost: 1337 oder http://127.0.0.1:1337 im Browser zugegriffen werden.

Eine einfache Webseite wird mit dem Text „Hallo, Welt!“ Oben angezeigt, wie in der Abbildung unten gezeigt.



Bearbeitbares Online-Beispiel.

Hallo Weltbefehlszeile

Node.js kann auch zum Erstellen von Befehlszeilen-Dienstprogrammen verwendet werden. Das folgende Beispiel liest das erste Argument aus der Befehlszeile und gibt eine Hello-Nachricht aus.

So führen Sie diesen Code auf einem Unix-System aus:

1. Erstellen Sie eine neue Datei und fügen Sie den folgenden Code ein. Der Dateiname ist irrelevant.
2. Machen Sie diese Datei mit `chmod 700 FILE_NAME` ausführbar
3. Führen Sie die App mit `./APP_NAME David`

Unter Windows führen Sie Schritt 1 aus und führen es mit dem `node APP_NAME David`

```
#!/usr/bin/env node

'use strict';

/*
```

```
    The command line arguments are stored in the `process.argv` array,
    which has the following structure:
    [0] The path of the executable that started the Node.js process
    [1] The path to this application
    [2-n] the command line arguments

    Example: [ '/bin/node', '/path/to/yourscript', 'arg1', 'arg2', ... ]
    src: https://nodejs.org/api/process.html#process_process_argv
 */

// Store the first argument as username.
var username = process.argv[2];

// Check if the username hasn't been provided.
if (!username) {

    // Extract the filename
    var appName = process.argv[1].split(require('path').sep).pop();

    //  Give the user an example on how to use the app.
    console.error('Missing argument! Example: %s YOUR_NAME', appName);

    // Exit the app (success: 0, error: 1).
    // An error will stop the execution chain. For example:
    //   ./app.js && ls       -> won't execute ls
    //   ./app.js David && ls -> will execute ls
    process.exit(1);
}

// Print the message to the console.
console.log('Hello %s!', username);
```

Node.js installieren und ausführen

Installieren Sie zunächst Node.js auf Ihrem Entwicklungscomputer.

Windows: Navigieren Sie zur Downloadseite und laden Sie das Installationsprogramm herunter bzw. führen Sie es aus.

Mac: Navigieren Sie zur Downloadseite und laden Sie das Installationsprogramm herunter bzw. führen Sie es aus. Alternativ können Sie den Knoten über den Homebrew-Knoten mithilfe des `brew install node` . Homebrew ist ein Befehlszeilen-Paketmanager für Macintosh. Weitere Informationen hierzu finden Sie auf der Homebrew-Website .

Linux: Befolgen Sie die Anweisungen für Ihre Distribution auf der Befehlszeilen-Installationsseite .

Ausführen eines Knotenprogramms

Um ein Node.js-Programm auszuführen, führen Sie einfach den `node app.js` oder `nodejs app.js` , wobei `app.js` der Dateiname des Quellcodes Ihrer Knoten-App ist. Sie müssen das `.js` Suffix für Node nicht `.js` , um das Skript zu finden, das Sie `.js` .

Alternativ kann unter UNIX-basierten Betriebssystemen ein Node-Programm als Terminalskript

ausgeführt werden. Um dies zu tun, muss es mit einem Shebang beginnen, der auf den Node-Interpreter zeigt, z. B. `#!/usr/bin/env node` . Die Datei muss auch als ausführbar festgelegt werden. Dies kann mit `chmod` . Jetzt kann das Skript direkt von der Befehlszeile aus ausgeführt werden.

Online-Bereitstellung Ihrer Anwendung

Wenn Sie Ihre App in einer (Node.js-spezifischen) gehosteten Umgebung bereitstellen, bietet diese Umgebung normalerweise eine `PORT` -Umgebungsvariable an, auf der Sie Ihren Server ausführen können. Wenn Sie die Portnummer in `process.env.PORT` können Sie auf die Anwendung zugreifen.

Zum Beispiel,

```
http.createServer(function(request, response) {
   // your server code
}).listen(process.env.PORT);
```

Wenn Sie während des Debugging auf diesen Offlinezugriff zugreifen möchten, können Sie Folgendes verwenden:

```
http.createServer(function(request, response) {
  // your server code
}).listen(process.env.PORT || 3000);
```

Dabei ist `3000` die Offline-Portnummer.

Debuggen Ihrer NodeJS-Anwendung

Sie können den Node-Inspector verwenden. Führen Sie diesen Befehl aus, um ihn über npm zu installieren:

```
npm install -g node-inspector
```

Dann können Sie Ihre Anwendung mit debuggen

```
node-debug app.js
```

Das Github-Repository finden Sie hier: https://github.com/node-inspector/node-inspector

---

Debugging nativ

Sie können node.js auch nativ debuggen, indem Sie es wie folgt starten:

```
node debug your-script.js
```

Verwenden Sie den folgenden Befehl, um Ihren Debugger genau in einer von Ihnen gewünschten

Codezeile abzubrechen:

```
debugger;
```

Weitere Informationen finden Sie hier .

Verwenden Sie in node.js 8 den folgenden Befehl:

```
node --inspect-brk your-script.js
```

Öffnen Sie dann `about://inspect` in einer aktuellen Version von Google Chrome und wählen Sie Ihr Knotenskript aus, um die Debugging-Funktionen der DevTools von Chrome zu erhalten.

Hallo Welt mit Express

Im folgenden Beispiel wird mit Express ein HTTP-Server erstellt, der auf Port 3000 überwacht wird und der mit "Hallo, Welt!" Antwortet. Express ist ein häufig verwendetes Web-Framework, das zum Erstellen von HTTP-APIs nützlich ist.

`myApp` einen neuen Ordner an, z. B. `myApp` . Gehen Sie zu `myApp` und `myApp` eine neue JavaScript-Datei, die den folgenden Code enthält (nennen wir es beispielsweise `hello.js`). Installieren Sie dann das Express-Modul mit `npm install --save express` von der Befehlszeile aus. *Weitere Informationen zum Installieren von Paketen finden Sie in dieser Dokumentation* .

```
// Import the top-level function of express
const express = require('express');

// Creates an Express application using the top-level function
const app = express();

// Define port number as 3000
const port = 3000;

// Routes HTTP GET requests to the specified path "/" with the specified callback function
app.get('/', function(request, response) {
  response.send('Hello, World!');
});

// Make the app listen on port 3000
app.listen(port, function() {
  console.log('Server listening on http://localhost:' + port);
});
```

Führen Sie in der Befehlszeile den folgenden Befehl aus:

```
node hello.js
```

Öffnen Sie Ihren Browser und navigieren Sie zu `http://localhost:3000` oder `http://127.0.0.1:3000` , um die Antwort `http://127.0.0.1:3000` .

Weitere Informationen zum Express-Framework finden Sie im Abschnitt Web Apps With Express

Hallo Weltgrundrouting

Wenn Sie wissen, wie ein HTTP-Server mit einem Knoten erstellt wird, ist es wichtig zu verstehen, wie er etwas erledigen kann, je nach dem Pfad, zu dem ein Benutzer navigiert hat. Dieses Phänomen wird "Routing" genannt.

Das einfachste Beispiel dafür wäre zu prüfen, `if (request.url === 'some/path/here')` und dann eine Funktion aufgerufen wird, die mit einer neuen Datei antwortet.

Ein Beispiel dafür ist hier zu sehen:

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

function index (request, response) {
    response.writeHead(200);
    response.end('Hello, World!');
}

http.createServer(function (request, response) {

    if (request.url === '/') {
        return index(request, response);
    }

    response.writeHead(404);
    response.end(http.STATUS_CODES[404]);

}).listen(1337);
```

Wenn Sie Ihre "Routen" jedoch weiterhin so definieren, erhalten Sie eine massive Rückruffunktion, und wir möchten kein riesiges Durcheinander wie dieses, also lassen Sie uns sehen, ob wir das aufräumen können.

Lassen Sie uns zunächst alle Routen in einem Objekt speichern:

```
var routes = {
    '/': function index (request, response) {
        response.writeHead(200);
        response.end('Hello, World!');
    },
    '/foo': function foo (request, response) {
        response.writeHead(200);
        response.end('You are now viewing "foo"');
    }
}
```

Nachdem wir zwei Routen in einem Objekt gespeichert haben, können wir sie jetzt in unserem Hauptcallback nach ihnen suchen:

```
http.createServer(function (request, response) {

    if (request.url in routes) {
        return routes[request.url](request, response);
    }
```

```
    response.writeHead(404);
    response.end(http.STATUS_CODES[404]);

}).listen(1337);
```

Jedes Mal, wenn Sie versuchen, auf Ihrer Website zu navigieren, wird die Existenz dieses Pfads in Ihren Routen überprüft und die entsprechende Funktion aufgerufen. Wird keine Route gefunden, antwortet der Server mit einem 404 (Not Found).

Und da haben Sie es - das Routing mit der HTTP-Server-API ist sehr einfach.

TLS Socket: Server und Client

Die einzigen Hauptunterschiede zwischen dieser und einer normalen TCP-Verbindung sind der private Schlüssel und das öffentliche Zertifikat, das Sie in ein Optionsobjekt setzen müssen.

So erstellen Sie einen Schlüssel und ein Zertifikat

Der erste Schritt in diesem Sicherheitsprozess ist die Erstellung eines privaten Schlüssels. Und was ist dieser private Schlüssel? Im Grunde handelt es sich dabei um ein zufälliges Rauschen, das zum Verschlüsseln von Informationen verwendet wird. Theoretisch könnten Sie einen Schlüssel erstellen und damit verschlüsseln, was Sie möchten. Es ist jedoch am besten, für bestimmte Dinge unterschiedliche Schlüssel zu verwenden. Wenn jemand Ihren privaten Schlüssel stiehlt, ist es ähnlich, wenn jemand Ihre Hausschlüssel stiehlt. Stellen Sie sich vor, Sie hätten mit dem gleichen Schlüssel Ihr Auto, Ihre Garage, Ihr Büro usw. gesperrt.

```
openssl genrsa -out private-key.pem 1024
```

Sobald wir unseren privaten Schlüssel haben, können wir eine CSR (Certificate Signing Request) erstellen. Dies ist unser Wunsch, den privaten Schlüssel von einer fancy Authority signieren zu lassen. Deshalb müssen Sie Informationen eingeben, die sich auf Ihr Unternehmen beziehen. Diese Informationen werden von der unterzeichnenden Behörde eingesehen und zur Bestätigung verwendet. In unserem Fall ist es egal, was Sie eingeben, da wir unser Zertifikat im nächsten Schritt selbst unterschreiben werden.

```
openssl req -new -key private-key.pem -out csr.pem
```

Jetzt, da unsere Papiere ausgefüllt sind, ist es an der Zeit, so zu tun, als wären wir eine coole Zeichnungsbehörde.

```
openssl x509 -req -in csr.pem -signkey private-key.pem -out public-cert.pem
```

Nachdem Sie nun den privaten Schlüssel und das öffentliche Zertifikat besitzen, können Sie eine sichere Verbindung zwischen zwei NodeJS-Apps herstellen. Und wie Sie im Beispielcode sehen können, ist dies ein sehr einfacher Prozess.

Wichtig!

Da wir das öffentliche Zertifikat selbst erstellt haben, ist unser Zertifikat in aller Ehrlichkeit wertlos, weil wir Nobodies sind. Der NodeJS-Server vertraut einem solchen Zertifikat standardmäßig nicht, und deshalb müssen wir ihm sagen, dass er unserem Zertifikat tatsächlich mit der folgenden Option rejectUnauthorized vertraut: false. **Sehr wichtig** : Setzen Sie diese Variable niemals in einer Produktionsumgebung auf true.

TLS-Socket-Server

```
'use strict';

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

const PORT = 1337;
const HOST = '127.0.0.1'

var options = {
    key: fs.readFileSync('private-key.pem'),
    cert: fs.readFileSync('public-cert.pem')
};

var server = tls.createServer(options, function(socket) {

    // Send a friendly message
    socket.write("I am the server sending you a message.");

    // Print the data that we received
    socket.on('data', function(data) {

        console.log('Received: %s [it is %d bytes long]',
            data.toString().replace(/(\n)/gm,""),
            data.length);

    });

    // Let us know when the transmission is over
    socket.on('end', function() {

        console.log('EOT (End Of Transmission)');

    });

});

// Start listening on a specific port and address
server.listen(PORT, HOST, function() {

    console.log("I'm listening at %s, on port %s", HOST, PORT);

});

// When an error occurs, show it.
server.on('error', function(error) {
```

```
    console.error(error);

    // Close the connection after the error occurred.
    server.destroy();

});
```

TLS-Socket-Client

```
'use strict';

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

const PORT = 1337;
const HOST = '127.0.0.1'

// Pass the certs to the server and let it know to process even unauthorized certs.
var options = {
    key: fs.readFileSync('private-key.pem'),
    cert: fs.readFileSync('public-cert.pem'),
    rejectUnauthorized: false
};

var client = tls.connect(PORT, HOST, options, function() {

    // Check if the authorization worked
    if (client.authorized) {
        console.log("Connection authorized by a Certificate Authority.");
    } else {
        console.log("Connection not authorized: " + client.authorizationError)
    }

    // Send a friendly message
    client.write("I am the client sending you a message.");

});

client.on("data", function(data) {

    console.log('Received: %s [it is %d bytes long]',
        data.toString().replace(/(\n)/gm,""),
        data.length);

    // Close the connection after receiving the message
    client.end();

});

client.on('close', function() {

    console.log("Connection closed");

});

// When an error ocoures, show it.
client.on('error', function(error) {
```

```
    console.error(error);

    // Close the connection after the error occurred.
    client.destroy();

});
```

Hallo Welt in der REPL

Bei Aufruf ohne Argumente startet Node.js eine REPL (Read-Eval-Print-Loop), die auch als " *Node-Shell* " bezeichnet wird.

Geben Sie an einer Eingabeaufforderung `node` .

```
$ node
>
```

Am Knoten Shell - Prompt `>` Typ „Hallo Welt!“

```
$ node
> "Hello World!"
'Hello World!'
```

Kern Module

Node.js ist eine Javascript-Engine (V8-Engine von Google für Chrome, die in C ++ geschrieben ist), mit der Javascript außerhalb des Browsers ausgeführt werden kann. Während zahlreiche Bibliotheken für die Erweiterung der Node-Funktionen verfügbar sind, enthält die Engine eine Reihe von *Kernmodulen,* die grundlegende Funktionen implementieren.

Derzeit sind 34 Kernmodule in Node enthalten:

```
[ 'assert',
  'buffer',
  'c/c++_addons',
  'child_process',
  'cluster',
  'console',
  'crypto',
  'deprecated_apis',
  'dns',
  'domain',
  'Events',
  'fs',
  'http',
  'https',
  'module',
  'net',
  'os',
  'path',
  'punycode',
  'querystring',
```

```
  'readline',
  'repl',
  'stream',
  'string_decoder',
  'timers',
  'tls_(ssl)',
  'tracing',
  'tty',
  'dgram',
  'url',
  'util',
  'v8',
  'vm',
  'zlib' ]
```

Diese Liste wurde von der Node-Dokumentations-API https://nodejs.org/api/all.html (JSON-Datei: https://nodejs.org/api/all.json) abgerufen.

Alle Kernmodule auf einen Blick

behaupten

Das `assert` Modul bietet einen einfachen Satz von Assertionstests, mit denen Invarianten getestet werden können.

Puffer

Vor der Einführung von `TypedArray` in ECMAScript 2015 (ES6) verfügte die JavaScript-Sprache nicht über einen Mechanismus zum Lesen oder Bearbeiten von Binärdatenströmen. Die `Buffer` Klasse wurde als Teil der Node.js-API eingeführt, um die Interaktion mit Octet-Streams im Zusammenhang mit TCP-Streams und Dateisystemvorgängen zu ermöglichen.

`TypedArray` nun in ES6 `TypedArray` hinzugefügt wurde, implementiert die `Buffer` Klasse die `Uint8Array` API auf eine Art und Weise, die für Node.js-Anwendungsfälle optimiert und geeignet ist.

c / c ++ _ Addons

Node.js Addons sind dynamisch verknüpfte, gemeinsam genutzte Objekte, die in C oder C ++ geschrieben wurden. Sie können mit der Funktion `require()` in Node.js geladen werden und werden so verwendet, als wären sie ein gewöhnliches Node.js-Modul. Sie dienen in erster Linie dazu, eine Schnittstelle zwischen JavaScript in Node.js und C / C ++ - Bibliotheken bereitzustellen.

child_process

Das `child_process` Modul bietet die Möglichkeit, `child_process` Prozesse auf ähnliche Weise wie popen (3) zu erzeugen.

Cluster

Eine einzelne Instanz von Node.js wird in einem einzelnen Thread ausgeführt. Um Multi-Core-

Systeme zu nutzen, möchte der Benutzer manchmal ein Cluster von Node.js-Prozessen starten, um die Last zu handhaben. Mit dem Cluster-Modul können Sie auf einfache Weise untergeordnete Prozesse erstellen, die alle Serverports gemeinsam nutzen.

Konsole

Das `console` bietet eine einfache Debugging-Konsole, die dem von Webbrowsern bereitgestellten JavaScript-Konsolenmechanismus ähnelt.

Krypto

Das `crypto` bietet eine kryptografische Funktionalität, die eine Reihe von Wrappern für die Hash-, HMAC-, Verschlüsselungs-, Entschlüsselungs-, Signatur- und Überprüfungsfunktionen von OpenSSL umfasst.

deprecated_apis

Node.js kann APIs ablehnen, wenn entweder (a) die Verwendung der API als unsicher angesehen wird, (b) eine verbesserte alternative API zur Verfügung gestellt wurde oder (c) in einer zukünftigen Hauptversion Änderungen der API erwartet werden .

dns

Das `dns` Modul enthält Funktionen, die zwei verschiedenen Kategorien angehören:

1. Funktionen, die die zugrunde liegenden Betriebssystemeinrichtungen zur Namensauflösung verwenden und nicht notwendigerweise Netzwerkkommunikation durchführen. Diese Kategorie enthält nur eine Funktion: `dns.lookup()` .
2. Funktionen, die eine Verbindung zu einem tatsächlichen DNS-Server herstellen, um die Namensauflösung durchzuführen, und die *immer* das Netzwerk verwenden, um DNS-Abfragen auszuführen. Diese Kategorie enthält alle Funktionen im `dns` Modul mit *Ausnahme von* `dns.lookup()` .

Domain

Dieses Modul steht noch aus . Sobald eine Ersatz-API abgeschlossen ist, wird dieses Modul vollständig abgelehnt. Die meisten Endbenutzer sollten **keinen** Grund haben, dieses Modul zu verwenden. Benutzer, die unbedingt über die Funktionalität von Domains verfügen müssen, können sich vorerst darauf verlassen, sollten aber in der Zukunft auf eine andere Lösung umsteigen müssen.

Veranstaltungen

Ein Großteil der Kern-API von Node.j basiert auf einer idiomatischen, asynchronen, ereignisgesteuerten Architektur, in der bestimmte Arten von Objekten (als "Emitter" bezeichnet) in regelmäßigen Abständen benannte Ereignisse ausgeben, die den Aufruf von Function-Objekten ("Listeners") bewirken.

fs

File I / O wird von einfachen Wrappern für POSIX-Standardfunktionen bereitgestellt. Um dieses Modul zu verwenden, `require('fs')` . Alle Methoden haben asynchrone und synchrone Formen.

http

Die HTTP-Schnittstellen in Node.js unterstützen viele Funktionen des Protokolls, die traditionell schwer zu verwenden waren. Insbesondere große, möglicherweise chunkcodierte Nachrichten. Die Benutzeroberfläche achtet nicht darauf, ganze Anfragen oder Antworten zu puffern - der Benutzer kann Daten streamen.

https

HTTPS ist das HTTP-Protokoll über TLS / SSL. In Node.js ist dies als separates Modul implementiert.

Modul

Node.js verfügt über ein einfaches Modulladesystem. In Node.js befinden sich Dateien und Module in einer Eins-zu-Eins-Entsprechung (jede Datei wird als separates Modul behandelt).

Netz

Das `net` Modul ermöglicht Ihnen mit einem asynchronen Netzwerk - Wrapper. Es enthält Funktionen zum Erstellen von Servern und Clients (Streams genannt). Sie können dieses Modul mit `require('net');` einschließen `require('net');` .

os

Das `os` Modul stellt eine Reihe von Betriebssystemmethoden zur Verfügung.

Pfad

Das `path` - Modul bietet Dienstprogramme für die mit Datei- und Verzeichnispfaden arbeiten.

Punycode

Die Version des in Node.js gebündelten Punycode-Moduls wird nicht mehr unterstützt .

Querzeichenfolge

Das Modul `querystring` bietet Dienstprogramme zum Analysieren und Formatieren von URL-`querystring` .

Zeile lesen

Das `readline` Modul bietet eine Schnittstelle zum Lesen von Daten aus einem lesbaren Stream (z. B. `process.stdin`) Zeile für Zeile.

repl

Das `repl` Modul bietet eine REPL-Implementierung (Read-Eval-Print-Loop), die sowohl als

eigenständiges Programm als auch in andere Anwendungen verfügbar ist.

Strom

Ein Stream ist eine abstrakte Schnittstelle zum Arbeiten mit Streaming-Daten in Node.js. Das `stream` Modul bietet eine Basis-API, mit der sich Objekte, die die Stream-Schnittstelle implementieren, leicht erstellen lassen.

Es gibt viele Stream-Objekte, die von Node.js bereitgestellt werden. Beispielsweise sind eine Anforderung an einen HTTP-Server und `process.stdout` beide Stream-Instanzen.

string_decoder

Das `string_decoder` Modul stellt eine API zum Dekodieren `Buffer` Objekte in Zeichenfolgen in einer Weise , die Multi-Byte - UTF-8 und UTF-16 - Zeichen codiert bewahrt.

Timer

Das `timer` stellt eine globale API zum Planen von Funktionen bereit, die zu einem späteren Zeitpunkt aufgerufen werden sollen. Da es sich bei den Timer-Funktionen um globale Werte handelt, müssen für den Einsatz der API keine „timers“ `require('timers')` aufgerufen werden.

Die Zeitgeberfunktionen in Node.js implementieren eine ähnliche API wie die von Web-Browsern bereitgestellte Timer-API, verwenden jedoch eine andere interne Implementierung, die auf der Node.js-Ereignisschleife basiert.

tls_ (ssl)

Das `tls` Modul bietet eine Implementierung der Protokolle Transport Layer Security (TLS) und SSL (Secure Socket Layer), die auf OpenSSL basieren.

Rückverfolgung

Das Ablaufereignis bietet einen Mechanismus zum Zentralisieren der Ablaufverfolgungsinformationen, die von V8, dem Knotenkern und dem Benutzerraumcode generiert werden.

Die `--trace-events-enabled` kann aktiviert werden, indem beim Starten einer Node.js-Anwendung `--trace-events-enabled` Flag `--trace-events-enabled` .

tty

Das `tty` Modul stellt die Klassen `tty.ReadStream` und `tty.WriteStream` . In den meisten Fällen ist es nicht notwendig oder möglich, dieses Modul direkt zu verwenden.

dgram

Das `dgram` Modul bietet eine Implementierung von UDP-Datagram-Sockets.

URL

Das `url` Modul stellt Dienstprogramme für die URL-Auflösung und -Parsing bereit.

util

Das `util` Modul ist hauptsächlich für die Unterstützung der internen APIs von Node.js konzipiert. Viele Dienstprogramme sind jedoch auch für Anwendungs- und Modulentwickler nützlich.

v8

Das `v8` Modul macht APIs verfügbar, die für die in der Binärdatei Node.js integrierte Version von V8 spezifisch sind.

Hinweis : Die APIs und die Implementierung können jederzeit geändert werden.

vm

Das `vm` Modul stellt APIs zum Kompilieren und Ausführen von Code in virtuellen V8-VM-Kontexten bereit. JavaScript-Code kann kompiliert und sofort ausgeführt oder kompiliert, gespeichert und später ausgeführt werden.

Hinweis : Das VM-Modul ist kein Sicherheitsmechanismus. ***Verwenden Sie es nicht, um nicht vertrauenswürdigen Code auszuführen*** .

zlib

Das `zlib` Modul bietet Kompressionsfunktionen, die mit Gzip und Deflate / Inflate implementiert werden.

So installieren Sie einen einfachen HTTPS-Webserver!

Sobald Sie node.js auf Ihrem System installiert haben, können Sie einfach die folgenden Schritte ausführen, um einen grundlegenden Webserver mit Unterstützung für HTTP und HTTPS zu erhalten!

Schritt 1: Erstellen Sie eine Zertifizierungsstelle

1. Erstellen Sie den Ordner, in dem Sie Ihren Schlüssel und Ihr Zertifikat speichern möchten:

   ```
   mkdir conf
   ```

---

2. Gehen Sie zu diesem Verzeichnis:

   ```
   cd conf
   ```

---

3. `ca.cnf` diese `ca.cnf` Datei, um sie als Konfigurationsverknüpfung zu verwenden:

   ```
   wget https://raw.githubusercontent.com/anders94/https-authorized-clients/master/keys/ca.cnf
   ```

---

4. Erstellen Sie eine neue Zertifizierungsstelle mit dieser Konfiguration:

   ```
   openssl req -new -x509 -days 9999 -config ca.cnf -keyout ca-key.pem -out ca-cert.pem
   ```

---

5. Jetzt, da wir unsere Zertifizierungsstelle in `ca-key.pem` und `ca-cert.pem` , generieren wir einen privaten Schlüssel für den Server:

   ```
   openssl genrsa -out key.pem 4096
   ```

---

6. `server.cnf` diese `server.cnf` Datei, um sie als Konfigurationsverknüpfung zu verwenden:

   ```
   wget https://raw.githubusercontent.com/anders94/https-authorized-
   clients/master/keys/server.cnf
   ```

---

7. Generieren Sie die Zertifikatsignierungsanforderung mit dieser Konfiguration:

   ```
   openssl req -new -config server.cnf -key key.pem -out csr.pem
   ```

---

8. unterschreibe die Anfrage:

   ```
   openssl x509 -req -extfile server.cnf -days 999 -passin "pass:password" -in csr.pem -CA ca-
   cert.pem -CAkey ca-key.pem -CAcreateserial -out cert.pem
   ```

Schritt 2: Installieren Sie Ihr Zertifikat als Stammzertifikat

1. Kopieren Sie Ihr Zertifikat in den Ordner Ihrer Stammzertifikate:

   ```
   sudo cp ca-crt.pem /usr/local/share/ca-certificates/ca-crt.pem
   ```

---

2. CA Store aktualisieren:

   ```
   sudo update-ca-certificates
   ```

Schritt 3: Starten Sie Ihren Knotenserver

Zunächst möchten Sie eine `server.js` Datei erstellen, die Ihren tatsächlichen `server.js` enthält.

Das minimale Setup für einen HTTPS-Server in Node.js wäre ungefähr so:

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

var httpsOptions = {
    key: fs.readFileSync('path/to/server-key.pem'),
    cert: fs.readFileSync('path/to/server-crt.pem')
};
```

```
var app = function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}

https.createServer(httpsOptions, app).listen(4433);
```

Wenn Sie auch HTTP-Anfragen unterstützen möchten, müssen Sie nur diese kleine Änderung vornehmen:

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

var httpsOptions = {
    key: fs.readFileSync('path/to/server-key.pem'),
    cert: fs.readFileSync('path/to/server-crt.pem')
};

var app = function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}

http.createServer(app).listen(8888);
https.createServer(httpsOptions, app).listen(4433);
```

1. Gehen Sie in das Verzeichnis, in dem sich Ihre `server.js` befindet:

   ```
   cd /path/to
   ```

---

2. Führen Sie `server.js` :

   ```
   node server.js
   ```

Erste Schritte mit Node.js online lesen: https://riptutorial.com/de/node-js/topic/340/erste-schritte-mit-node-js

Kapitel 2: Abhängigkeitsspritze

Examples

Warum Abhängigkeitsinjektion verwenden?

1. **Schneller Entwicklungsprozess**
2. **Entkopplung**
3. **Unit-Test schreiben**

Schneller Entwicklungsprozess
Bei der Verwendung von Abhängigkeitsinjektionsknoten können Entwickler ihren Entwicklungsprozess beschleunigen, da nach DI weniger Codekonflikte auftreten und das Modul leicht verwaltet werden kann.

Entkopplung
Module werden weniger paaren, dann ist es einfach zu warten.

Unit-Test schreiben
Durch hartcodierte Abhängigkeiten können sie an das Modul übergeben werden, und der Einheitentest kann für jedes Modul leicht geschrieben werden.

Abhängigkeitsspritze online lesen: https://riptutorial.com/de/node-js/topic/7681/abhangigkeitsspritze

Kapitel 3: Anmutiges Herunterfahren

Examples

Graceful Shutdown - SIGTERM

Mit **server.close ()** und **process.exit ()** können wir die Serverausnahme abfangen und ein ordnungsgemäßes Herunterfahren durchführen.

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

var server = http.createServer(function (req, res) {
  setTimeout(function () { //simulate a long request
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
  }, 4000);
}).listen(9090, function (err) {
  console.log('listening http://localhost:9090/');
  console.log('pid is ' + process.pid);
});

process.on('SIGTERM', function () {
  server.close(function () {
    process.exit(0);
  });
});
```

Anmutiges Herunterfahren online lesen: https://riptutorial.com/de/node-js/topic/5996/anmutiges-herunterfahren

Kapitel 4: Anwendungsfälle von Node.js

Examples

HTTP-Server

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

console.log('Starting server...');
var config = {
    port: 80,
    contentType: 'application/json; charset=utf-8'
};
// JSON-API server on port 80

var server = http.createServer();
server.listen(config.port);
server.on('error', (err) => {
    if (err.code == 'EADDRINUSE') console.error('Port '+ config.port +' is already in use');
    else console.error(err.message);
});
server.on('request', (request, res) => {
    var remoteAddress = request.headers['x-forwarded-for'] ||
request.connection.remoteAddress; // Client address
    console.log(remoteAddress +' '+ request.method +' '+ request.url);

    var out = {};
    // Here you can change output according to `request.url`
    out.test = request.url;
    res.writeHead(200, {
        'Content-Type': config.contentType
    });
    res.end(JSON.stringify(out));
});
server.on('listening', () => {
    c.info('Server is available: http://localhost:'+ config.port);
});
```

Konsole mit Eingabeaufforderung

```
const process = require('process');
const rl = require('readline').createInterface(process.stdin, process.stdout);

rl.pause();
console.log('Something long is happening here...');

var cliConfig = {
    promptPrefix: ' > '
}

/*
    Commands recognition
    BEGIN
*/
var commands = {
```

```
    eval: function(arg) { // Try typing in console: eval 2 * 10 ^ 3 + 2 ^ 4
        arg = arg.join(' ');
        try { console.log(eval(arg)); }
        catch (e) { console.log(e); }
    },
    exit: function(arg) {
        process.exit();
    }
};
rl.on('line', (str) => {
    rl.pause();
    var arg = str.trim().match(/([^"]+)|("(?:[^"\\]|\\.)+")/g); // Applying regular expression
for removing all spaces except for what between double quotes:
http://stackoverflow.com/a/14540319/2396907
    if (arg) {
        for (let n in arg) {
            arg[n] = arg[n].replace(/^\"|\"$/g, '');
        }
        var commandName = arg[0];
        var command = commands[commandName];
        if (command) {
            arg.shift();
            command(arg);
        }
        else console.log('Command "'+ commandName +'" doesn\'t exist');
    }
    rl.prompt();
});
/*
    END OF
    Commands recognition
*/

rl.setPrompt(cliConfig.promptPrefix);
rl.prompt();
```

Anwendungsfälle von Node.js online lesen: https://riptutorial.com/de/node-js/topic/7703/anwendungsfalle-von-node-js

Kapitel 5: APIs mit Node.js erstellen

Examples

GET api mit Express

`Node.js` Apis können einfach in `Express` Web-Framework erstellt werden.

Das folgende Beispiel erstellt eine einfache `GET` API, um alle Benutzer aufzulisten.

Beispiel

```
var express = require('express');
var app = express();

var users =[{
        id: 1,
        name: "John Doe",
        age : 23,
        email: "john@doe.com"
    }];

// GET /api/users
app.get('/api/users', function(req, res){
    return res.json(users);    //return response as JSON
});

app.listen('3000', function(){
    console.log('Server listening on port 3000');
});
```

POST-API mit Express

Im folgenden Beispiel erstellen Sie eine `POST` API mit `Express` . Dieses Beispiel ist dem `GET` Beispiel ähnlich, mit Ausnahme der Verwendung von `body-parser` , der die `req.body` analysiert und zu `req.body` hinzufügt.

Beispiel

```
var express = require('express');
var app = express();
// for parsing the body in POST request
var bodyParser = require('body-parser');

var users =[{
    id: 1,
    name: "John Doe",
    age : 23,
    email: "john@doe.com"
}];

app.use(bodyParser.urlencoded({ extended: false }));
```

```
app.use(bodyParser.json());

// GET /api/users
app.get('/api/users', function(req, res){
    return res.json(users);
});


/* POST /api/users
    {
        "user": {
           "id": 3,
            "name": "Test User",
            "age" : 20,
            "email": "test@test.com"
        }
    }
*/
app.post('/api/users', function (req, res) {
    var user = req.body.user;
    users.push(user);

    return res.send('User has been added successfully');
});

app.listen('3000', function(){
    console.log('Server listening on port 3000');
});
```

APIs mit Node.js erstellen online lesen: https://riptutorial.com/de/node-js/topic/5991/apis-mit-node-js-erstellen

Kapitel 6: Arduino-Kommunikation mit nodeJs

Einführung

So zeigen Sie, wie Node.Js mit Arduino Uno kommunizieren können.

Examples

Node Js Kommunikation mit Arduino über den seriellen Port

Knoten-Js-Code

Ein Beispiel zum Starten dieses Themas ist der Node.js-Server, der über seriellen Port mit Arduino kommuniziert.

```
npm install express --save
npm install serialport --save
```

Beispiel app.js:

```
const express = require('express');
const app = express();
var SerialPort = require("serialport");

var port = 3000;

var arduinoCOMPort = "COM3";

var arduinoSerialPort = new SerialPort(arduinoCOMPort, {
 baudrate: 9600
});

arduinoSerialPort.on('open',function() {
  console.log('Serial Port ' + arduinoCOMPort + ' is opened.');
});

app.get('/', function (req, res) {

    return res.send('Working');

})

app.get('/:action', function (req, res) {

   var action = req.params.action || req.param('action');

    if(action == 'led'){
        arduinoSerialPort.write("w");
        return res.send('Led light is on!');
```

```
    }
    if(action == 'off') {
        arduinoSerialPort.write("t");
        return res.send("Led light is off!");
    }

    return res.send('Action: ' + action);

});

app.listen(port, function () {
  console.log('Example app listening on port http://0.0.0.0:' + port + '!');
});
```

Beispielexpress-Server starten:

```
node app.js
```

Arduino-Code

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.

  Serial.begin(9600); // Begen listening on port 9600 for serial

  pinMode(LED_BUILTIN, OUTPUT);

  digitalWrite(LED_BUILTIN, LOW);
}

// the loop function runs over and over again forever
void loop() {

   if(Serial.available() > 0) // Read from serial port
    {
      char ReaderFromNode; // Store current character
      ReaderFromNode = (char) Serial.read();
      convertToState(ReaderFromNode); // Convert character to state
    }
  delay(1000);
}

void convertToState(char chr) {
  if(chr=='o'){
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
  }
  if(chr=='f'){
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
  }
}
```

Inbetriebnahme

1. Verbinden Sie den Arduino mit Ihrem Computer.
2. Starten Sie den Server

Steuern Sie das eingebaute LED über den Knoten-Express-Server js.

So schalten Sie die LED ein:

```
http://0.0.0.0:3000/led
```

So schalten Sie die LED aus:

```
http://0.0.0.0:3000/off
```

Arduino-Kommunikation mit nodeJs online lesen: https://riptutorial.com/de/node-js/topic/10509/arduino-kommunikation-mit-nodejs

Kapitel 7: Async / Warten

Einführung

Async / await ist ein Satz von Schlüsselwörtern, mit dem asynchroner Code prozedural geschrieben werden kann, ohne sich auf Callbacks (*Callback Hell*) oder Promise-Chaining (`.then().then().then()`) zu verlassen.

Dies funktioniert, indem Sie das `await` Schlüsselwort verwenden, um den Status einer asynchronen Funktion bis zur Auflösung eines Versprechens zu unterbrechen, und das `async` Schlüsselwort verwenden, um solche async-Funktionen zu deklarieren, die ein Versprechen zurückgeben.

Async / await ist standardmäßig von node.js 8 oder mit dem Flag `--harmony-async-await` verfügbar.

Examples

Async-Funktionen mit Try-Catch-Fehlerbehandlung

Eine der besten Funktionen der async / await-Syntax besteht darin, dass standardmäßige Try-Catch-Codierungsmethoden möglich sind, genau wie Sie synchronen Code schreiben.

```
const myFunc = async (req, res) => {
  try {
    const result = await somePromise();
  } catch (err) {
    // handle errors here
  }
});
```

Hier ein Beispiel mit Express und Promise-mysql:

```
router.get('/flags/:id', async (req, res) => {

  try {

    const connection = await pool.createConnection();

    try {
      const sql = `SELECT f.id, f.width, f.height, f.code, f.filename
                   FROM flags f
                   WHERE f.id = ?
                   LIMIT 1`;
      const flags = await connection.query(sql, req.params.id);
      if (flags.length === 0)
        return res.status(404).send({ message: 'flag not found' });

      return res.send({ flags[0] });

    } finally {
```

```
      pool.releaseConnection(connection);
    }

  } catch (err) {
    // handle errors here
  }
});
```

Vergleich zwischen Versprechen und Async / Await

Funktion mit Versprechen:

```
function myAsyncFunction() {
    return aFunctionThatReturnsAPromise()
           // doSomething is a sync function
           .then(result => doSomething(result))
           .catch(handleError);
}
```

Hier also, wenn Async / Await in Aktion tritt, um unsere Funktion zu reinigen:

```
async function myAsyncFunction() {
  let result;

  try {
      result = await aFunctionThatReturnsAPromise();
  } catch (error) {
      handleError(error);
  }

  // doSomething is a sync function
  return doSomething(result);
}
```

Das Schlüsselwort `async` wäre also ähnlich wie write `return new Promise((resolve, reject) => {...}` .

Und `await` ähnlich ab, um Ihr Ergebnis `then` Rückruf zu bekommen.

Hier hinterlasse ich ein recht kurzes Gif, das keinen Zweifel hinterlassen wird, nachdem ich es gesehen habe:

GIF

Fortschritt aus Rückrufen

Am Anfang gab es Rückrufe und Rückrufe waren in Ordnung:

```
const getTemperature = (callback) => {
  http.get('www.temperature.com/current', (res) => {
    callback(res.data.temperature)
  })
}
```

```
const getAirPollution = (callback) => {
  http.get('www.pollution.com/current', (res) => {
    callback(res.data.pollution)
  });
}

getTemperature(function(temp) {
  getAirPollution(function(pollution) {
    console.log(`the temp is ${temp} and the pollution is ${pollution}.`)
    // The temp is 27 and the pollution is 0.5.
  })
})
```

Es gab jedoch ein paar wirklich frustrierende Probleme mit Rückrufen, sodass wir alle anfingen, Versprechen zu verwenden.

```
const getTemperature = () => {
  return new Promise((resolve, reject) => {
    http.get('www.temperature.com/current', (res) => {
      resolve(res.data.temperature)
    })
  })
}

const getAirPollution = () => {
  return new Promise((resolve, reject) => {
    http.get('www.pollution.com/current', (res) => {
      resolve(res.data.pollution)
    })
  })
}

getTemperature()
.then(temp => console.log(`the temp is ${temp}`))
.then(() => getAirPollution())
.then(pollution => console.log(`and the pollution is ${pollution}`))
// the temp is 32
// and the pollution is 0.5
```

Das war ein bisschen besser. Schließlich fanden wir async / await. Was noch Versprechen unter der Haube einsetzt.

```
const temp = await getTemperature()
const pollution = await getAirPollution()
```

Stoppt die Ausführung bei Erwarten

Wenn das Versprechen etwas nicht zurückkehrt, kann die Asynchron - Aufgabe mit abgeschlossen werden `await` .

```
try{
    await User.findByIdAndUpdate(user._id, {
        $push: {
            tokens: token
```

```
        }
    }).exec()
}catch(e){
    handleError(e)
}
```

Async / Warten online lesen: https://riptutorial.com/de/node-js/topic/6729/async---warten

Kapitel 8: async.js

Syntax

- **Jeder Rückruf muss mit folgender Syntax geschrieben werden:**
- Funktionsrückruf (err, result [, arg1 [, ...]])
- **Auf diese Weise müssen Sie den Fehler zuerst zurückgeben und können die spätere Verwendung nicht ignorieren. `null` ist die Konvention ohne Fehler**
- Rückruf (null, myResult);
- **Ihre Rückrufe können mehr Argumente als *err* und *result enthalten* , sind jedoch nur für bestimmte Funktionen (Wasserfall, Seq, ...) nützlich.**
- Rückruf (null, myResult, myCustomArgument);
- **Und natürlich Fehler senden. Sie müssen es tun und Fehler behandeln (oder zumindest protokollieren).**
- Rückruf (err);

Examples

Parallel: Multitasking

async.parallel (Aufgaben, afterTasksCallback) führt eine Reihe von Aufgaben parallel aus und **wartet auf das Ende aller Aufgaben** (gemeldet vom Aufruf der **Callback-** Funktion).

Wenn Aufgaben abgeschlossen sind, ruft *async* den Hauptcallback mit allen Fehlern und Ergebnissen der Aufgaben auf.

```
function shortTimeFunction(callback) {
  setTimeout(function() {
    callback(null, 'resultOfShortTime');
  }, 200);
}

function mediumTimeFunction(callback) {
  setTimeout(function() {
    callback(null, 'resultOfMediumTime');
  }, 500);
}

function longTimeFunction(callback) {
  setTimeout(function() {
    callback(null, 'resultOfLongTime');
  }, 1000);
}
```

```
async.parallel([
    shortTimeFunction,
    mediumTimeFunction,
    longTimeFunction
  ],
  function(err, results) {
    if (err) {
      return console.error(err);
    }

    console.log(results);
  });
```

Ergebnis: `["resultOfShortTime", "resultOfMediumTime", "resultOfLongTime"]` .

Rufen Sie `async.parallel()` mit einem Objekt auf

Sie können den *Aufgaben*- Array-Parameter durch ein Objekt ersetzen. In diesem Fall sind die Ergebnisse auch ein Objekt **mit den gleichen Schlüsseln wie Aufgaben** .

Es ist sehr nützlich, einige Aufgaben zu berechnen und jedes Ergebnis leicht zu finden.

```
async.parallel({
    short: shortTimeFunction,
    medium: mediumTimeFunction,
    long: longTimeFunction
  },
  function(err, results) {
    if (err) {
      return console.error(err);
    }

    console.log(results);
  });
```

Ergebnis: `{short: "resultOfShortTime", medium: "resultOfMediumTime", long: "resultOfLongTime"}` .

Mehrere Werte auflösen

Jeder parallelen Funktion wird ein Rückruf übergeben. Dieser Rückruf kann entweder einen Fehler als erstes Argument oder Erfolgswerte zurückgeben. Wenn für einen Rückruf mehrere Erfolgswerte übergeben werden, werden diese Ergebnisse als Array zurückgegeben.

```
async.parallel({
    short: function shortTimeFunction(callback) {
      setTimeout(function() {
        callback(null, 'resultOfShortTime1', 'resultOfShortTime2');
      }, 200);
    },
    medium: function mediumTimeFunction(callback) {
      setTimeout(function() {
```

```
        callback(null, 'resultOfMediumTime1', 'resultOfMeiumTime2');
      }, 500);
    }
  },
  function(err, results) {
    if (err) {
      return console.error(err);
    }

    console.log(results);
  });
```

Ergebnis:

```
{
    short: ["resultOfShortTime1", "resultOfShortTime2"],
    medium: ["resultOfMediumTime1", "resultOfMediumTime2"]
}
```

.

Serie: unabhängiges Mono-Tasking

async.series (Aufgaben, afterTasksCallback) führt eine Reihe von Aufgaben aus. Jede Aufgabe wird **nach der anderen ausgeführt** . **Wenn eine Aufgabe fehlschlägt, stoppt *async* sofort die Ausführung und springt in den Hauptcallback** .

Wenn Aufgaben erfolgreich abgeschlossen wurden, rufen Sie *asynchron* den "Master" -Rückruf mit allen Fehlern und Ergebnissen der Aufgaben auf.

```
function shortTimeFunction(callback) {
  setTimeout(function() {
    callback(null, 'resultOfShortTime');
  }, 200);
}

function mediumTimeFunction(callback) {
  setTimeout(function() {
    callback(null, 'resultOfMediumTime');
  }, 500);
}

function longTimeFunction(callback) {
  setTimeout(function() {
    callback(null, 'resultOfLongTime');
  }, 1000);
}

async.series([
    mediumTimeFunction,
    shortTimeFunction,
    longTimeFunction
  ],
  function(err, results) {
    if (err) {
      return console.error(err);
```

```
    }

    console.log(results);
  });
```

Ergebnis: `["resultOfMediumTime", "resultOfShortTime", "resultOfLongTime"]` .

Rufen Sie `async.series()` mit einem Objekt auf

Sie können den *Aufgaben*- Array-Parameter durch ein Objekt ersetzen. In diesem Fall sind die Ergebnisse auch ein Objekt **mit den gleichen Schlüsseln wie Aufgaben** .

Es ist sehr nützlich, einige Aufgaben zu berechnen und jedes Ergebnis leicht zu finden.

```
async.series({
    short: shortTimeFunction,
    medium: mediumTimeFunction,
    long: longTimeFunction
  },
  function(err, results) {
    if (err) {
      return console.error(err);
    }

    console.log(results);
  });
```

Ergebnis: `{short: "resultOfShortTime", medium: "resultOfMediumTime", long: "resultOfLongTime"}` .

Wasserfall: abhängiges Mono-Tasking

async.waterfall (Aufgaben, afterTasksCallback) führt eine Reihe von Aufgaben aus. Jede Aufgabe wird **nach der anderen ausgeführt, und das Ergebnis einer Aufgabe wird an die nächste Aufgabe übergeben** . Wenn *async.series ()* fehlschlägt, stoppt *async* die Ausführung und ruft sofort den Hauptrückruf auf.

Wenn Aufgaben erfolgreich abgeschlossen wurden, rufen Sie *asynchron* den "Master" -Rückruf mit allen Fehlern und Ergebnissen der Aufgaben auf.

```
function getUserRequest(callback) {
  // We simulate the request with a timeout
  setTimeout(function() {
    var userResult = {
      name : 'Aamu'
    };

    callback(null, userResult);
  }, 500);
}

function getUserFriendsRequest(user, callback) {
  // Another request simulate with a timeout
```

```
  setTimeout(function() {
    var friendsResult = [];

    if (user.name === "Aamu"){
        friendsResult = [{
          name : 'Alice'
        }, {
          name: 'Bob'
        }];
    }

    callback(null, friendsResult);
  }, 500);
}

async.waterfall([
    getUserRequest,
    getUserFriendsRequest
  ],
  function(err, results) {
    if (err) {
      return console.error(err);
    }

    console.log(JSON.stringify(results));
  });
```

Ergebnis: `results` enthält den zweiten Callback-Parameter der letzten Funktion des Wasserfalls, in diesem Fall `friendsResult` .

async.times (Um Schleife besser zu handhaben)

Um eine Funktion innerhalb einer Schleife in node.js auszuführen, können Sie eine `for` Schleife für kurze Schleifen verwenden. Die Schleife ist jedoch lang, die Verwendung der `for` Schleife erhöht die Verarbeitungszeit, wodurch der Knotenprozess hängen bleiben kann. In solchen Szenarien können Sie **Folgendes** verwenden: **asycn.times**

```
function recursiveAction(n, callback)
{
    //do whatever want to do repeatedly
    callback(err, result);
}
async.times(5, function(n, next) {
    recursiveAction(n, function(err, result) {
        next(err, result);
    });
}, function(err, results) {
    // we should now have 5 result
});
```

Dies wird parallel genannt. Wenn Sie es **einzeln** aufrufen möchten, verwenden Sie: **async.timesSeries**

async.each (Um das Datenfeld effizient zu behandeln)

Wenn wir ein Array von Daten behandeln wollen, ist es besser, **async.each** zu verwenden. Wenn wir etwas mit allen Daten durchführen möchten und den endgültigen Rückruf erhalten möchten, sobald alles fertig ist, ist diese Methode nützlich. Dies wird parallel behandelt.

```
function createUser(userName, callback)
{
    //create user in db
    callback(null)//or error based on creation
}

var arrayOfData = ['Ritu', 'Sid', 'Tom'];
async.each(arrayOfData, function(eachUserName, callback) {

    // Perform operation on each user.
    console.log('Creating user '+eachUserName);
    //Returning callback is must. Else it wont get the final callback, even if we miss to
return one callback
    createUser(eachUserName, callback);

}, function(err) {
    //If any of the user creation failed may throw error.
    if( err ) {
      // One of the iterations produced an error.
      // All processing will now stop.
      console.log('unable to create user');
    } else {
      console.log('All user created successfully');
    }
});
```

Sie können **async.eachSeries** gleichzeitig verwenden

async.series (Ereignisse einzeln behandeln)

/ In async.series werden alle Funktionen in Reihe ausgeführt und die konsolidierten Ausgaben jeder Funktion werden an den endgültigen Rückruf übergeben. zB /

var async = required ('async'); async.series ([function (callback) {console.log ('First Execute ..'); callback (null, 'userPersonalData');}, function (callback) {console.log ('Second Execute ..'); callback (null, 'userDependentData');}], function (err, result) {console.log (result);});

//Ausgabe:

First Execute .. Second Execute .. ['userPersonalData', 'userDependentData'] // Ergebnis

async.js online lesen: https://riptutorial.com/de/node-js/topic/3972/async-js

Kapitel 9: Asynchrone Programmierung

Einführung

Node ist eine Programmiersprache, in der alles asynchron laufen kann. Nachfolgend finden Sie einige Beispiele und die typischen Aspekte des asynchronen Arbeitens.

Syntax

- doSomething ([args], function ([argsCB]) {/ * tun etwas, wenn sie fertig sind * /});
- doSomething ([args], ([argsCB]) => {/ * tun Sie etwas, wenn Sie fertig sind * /});

Examples

Rückruffunktionen

Callback-Funktionen in JavaScript

Rückruffunktionen sind in JavaScript üblich. Callback-Funktionen sind in JavaScript möglich, da Funktionen erstklassige Bürger sind .

Synchrone Rückrufe.

Rückruffunktionen können synchron oder asynchron sein. Da asynchrone Callback-Funktionen komplexer sein können, finden Sie hier ein einfaches Beispiel für eine synchrone Callback-Funktion.

```
// a function that uses a callback named `cb` as a parameter
function getSyncMessage(cb) {
    cb("Hello World!");
}

console.log("Before getSyncMessage call");
// calling a function and sending in a callback function as an argument.
getSyncMessage(function(message) {
    console.log(message);
});
console.log("After getSyncMessage call");
```

Die Ausgabe für den obigen Code lautet:

```
> Before getSyncMessage call
> Hello World!
> After getSyncMessage call
```

Zuerst gehen wir Schritt für Schritt durch, wie der obige Code ausgeführt wird. Dies ist mehr für diejenigen, die das Konzept von Rückrufen nicht bereits verstehen, wenn Sie es bereits verstanden haben. Sie können diesen Absatz gerne überspringen. Zuerst wird der Code analysiert und dann wird zunächst die Zeile 6 ausgeführt, die `Before getSyncMessage call` an die Konsole `Before getSyncMessage call` . Dann Linie 8 ausgeführt, der die Funktion aufruft `getSyncMessage` in einer anonymen Funktion als Argument für den Parameter namens Senden `cb` in der `getSyncMessage` Funktion. Die Ausführung erfolgt nun in der Funktion `getSyncMessage` in Zeile 3, die die soeben übergebene Funktion `cb` ausführt. Dieser Aufruf sendet eine Argumentzeichenfolge "Hello World" für die mit Parameter bezeichnete `message` in der in anonymous übergebenen Funktion. Die Ausführung geht dann zu Zeile 9, die `Hello World!` zur Konsole. Dann durchläuft die Ausführung den Prozess des Verlassens des Callstacks (siehe auch), wobei die Zeile 10, dann die Zeile 4 und dann schließlich die Zeile 11 wieder erreicht werden.

Einige Informationen, die Sie im Allgemeinen über Rückrufe kennen sollten:

- Die Funktion, die Sie als Callback an eine Funktion senden, kann null, einmal oder mehrmals aufgerufen werden. Auf die Umsetzung kommt es an.
- Die Callback-Funktion kann synchron oder asynchron und möglicherweise sowohl synchron als auch asynchron aufgerufen werden.
- Wie bei normalen Funktionen sind die Namen, die Sie Ihrer Funktion geben, nicht wichtig, aber die Reihenfolge ist. In Zeile 8 könnte die Parameter `message` beispielsweise mit dem Namen `statement` , `msg` oder, wenn Sie unsinnig sind, so etwas wie `jellybean` . Sie sollten also wissen, welche Parameter in Ihrem Callback gesendet werden, damit Sie sie mit den richtigen Namen in der richtigen Reihenfolge anzeigen können.

Asynchrone Rückrufe.

Zu beachten ist, dass JavaScript standardmäßig synchron ist. In der Umgebung (API, Browser, Node.js usw.) gibt es jedoch APIs, die es asynchron machen könnten (mehr dazu hier).

Einige häufige Dinge, die in JavaScript-Umgebungen, die Rückrufe akzeptieren, asynchron sind:

- Veranstaltungen
- setTimeout
- setInterval
- die Abruf-API
- Versprechen

Jede Funktion, die eine der oben genannten Funktionen verwendet, kann mit einer Funktion umschlossen werden, die einen Rückruf übernimmt, und der Rückruf wäre dann ein asynchroner Rückruf (obwohl das Versprechen eines Versprechens mit einer Funktion, die einen Rückruf entgegennimmt, wahrscheinlich als Anti-Muster betrachtet wird.) es gibt mehr bevorzugte Wege, um Versprechen zu behandeln).

In Anbetracht dieser Informationen können wir eine asynchrone Funktion ähnlich der obigen synchronen Funktion erstellen.

```
// a function that uses a callback named `cb` as a parameter
function getAsyncMessage(cb) {
    setTimeout(function () { cb("Hello World!") }, 1000);
}

console.log("Before getSyncMessage call");
// calling a function and sending in a callback function as an argument.
getAsyncMessage(function(message) {
    console.log(message);
});
console.log("After getSyncMessage call");
```

Welche druckt Folgendes auf die Konsole:

```
> Before getSyncMessage call
> After getSyncMessage call
// pauses for 1000 ms with no output
> Hello World!
```

Die Zeilenausführung geht an die Protokolle der Zeile 6 "Vor dem Aufruf von getSyncMessage". Dann wird die Ausführung in Zeile 8 mit dem Aufruf von getAsyncMessage und einem Rückruf für den Parameter `cb` . Dann wird Zeile 3 ausgeführt, die setTimeout mit einem Callback als erstem Argument und der Nummer 300 als zweitem Argument aufruft. `setTimeout` tut alles, was es tut, und hält an diesem Callback fest, sodass es später in 1000 Millisekunden aufgerufen werden kann. `setTimeout` das Timeout eingerichtet und die 1000 Millisekunden angehalten wurde, übergibt es die Ausführung an die Stelle, an der es unterbrochen wurde, und geht in Zeile 4 weiter , dann die Leitung 11, und dann pausiert für 1 Sekunde und setTimeout ruft dann seine Rückruffunktion , die 3 die Ausführung zurück auf Linie nimmt , wo `getAsyncMessages` Rückruf mit dem Wert „Hallo , Welt“ für seine Parameter aufgerufen wird `message` , die dann auf die Konsole auf der Leitung angemeldet ist 9 .

Rückruffunktionen in Node.js

NodeJS verfügt über asynchrone Callbacks und stellt normalerweise zwei Parameter für Ihre Funktionen bereit, die üblicherweise als `err` und `data` . Ein Beispiel zum Lesen eines Dateitextes.

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

fs.readFile("./test.txt", "utf8", function(err, data) {
    if(err) {
        // handle the error
    } else {
        // process the file text given with data
    }
});
```

Dies ist ein Beispiel für einen Rückruf, der einmal aufgerufen wird.

Es ist empfehlenswert, den Fehler irgendwie zu behandeln, selbst wenn Sie ihn nur protokollieren oder werfen. Das Andernfalls ist nicht erforderlich, wenn Sie werfen oder zurückkehren, und es kann entfernt werden, um die Einrückung zu verringern, solange Sie die Ausführung der aktuellen

Funktion im if-Vorgang beenden, indem Sie etwa das Werfen oder das Zurückkehren ausführen.

Obwohl es häufig üblich ist, `err` zu sehen, können `data` nicht immer der Fall sein, dass Ihre Rückrufe dieses Muster verwenden. Am besten sollten Sie sich die Dokumentation ansehen.

Ein weiterer Beispielrückruf stammt aus der Express-Bibliothek (Express 4.x):

```
// this code snippet was on http://expressjs.com/en/4x/api.html
const express = require('express');
const app = express();

// this app.get method takes a url route to watch for and a callback
// to call whenever that route is requested by a user.
app.get('/', function(req, res){
  res.send('hello world');
});

app.listen(3000);
```

Dieses Beispiel zeigt einen Rückruf, der mehrmals aufgerufen wird. Der Rückruf verfügt über zwei Objekte als Parameter, die hier als `req` und `res` Diese Namen entsprechen jeweils der Anforderung und der Antwort. Sie bieten Möglichkeiten, die eingehende Anforderung anzuzeigen und die Antwort festzulegen, die an den Benutzer gesendet wird.

Wie Sie sehen, gibt es verschiedene Möglichkeiten, mit Hilfe eines Rückrufs Sync- und Asynchron-Code in JavaScript auszuführen, und Callbacks sind in JavaScript sehr allgegenwärtig.

Code-Beispiel

Frage: Wie wird der Code ausgegeben und warum?

```
setTimeout(function() {
    console.log("A");
}, 1000);

setTimeout(function() {
    console.log("B");
}, 0);

getDataFromDatabase(function(err, data) {
    console.log("C");
    setTimeout(function() {
        console.log("D");
    }, 1000);
});

console.log("E");
```

Ausgabe: Dies ist sicher bekannt: `EBAD . C` ist unbekannt, wann es protokolliert wird.

Erläuterung: Der Compiler wird bei den `setTimeout` und `getDataFromDatabase` nicht `getDataFromDatabase` . Die erste Zeile, die er protokolliert, ist `E` Die Callback-Funktionen *(erstes Argument von `setTimeout`)* werden nach dem gesetzten Timeout asynchron ausgeführt!

Mehr Details:

1. `E` hat kein `setTimeout`
2. `B` hat ein Zeitlimit von 0 Millisekunden
3. `A` hat ein festgelegtes Timeout von 1000 Millisekunden
4. `D` müssen eine Datenbank anfordern, nachdem es muss `D` 1000 Millisekunden warten , so dass es nach kommt `A` .
5. `C` ist unbekannt, weil es unbekannt ist, wenn die Daten der Datenbank angefordert werden. Es könnte vor oder nach `A` .

Async-Fehlerbehandlung

Versuchen Sie, zu fangen

Fehler müssen immer behandelt werden. Wenn Sie eine synchrone Programmierung verwenden, können Sie einen `try catch` . Das funktioniert aber nicht, wenn Sie asynchron arbeiten! Beispiel:

```
try {
    setTimeout(function() {
        throw new Error("I'm an uncaught error and will stop the server!");
    }, 100);
}
catch (ex) {
    console.error("This error will not be work in an asynchronous situation: " + ex);
}
```

Async-Fehler werden nur innerhalb der Rückruffunktion behandelt!

Arbeitsmöglichkeiten

v0,8

Event-Handler

Die ersten Versionen von Node.JS erhielten einen Event-Handler.

```
process.on("UncaughtException", function(err, data) {
    if (err) {
        // error handling
    }
});
```

v0,8

Domains

Innerhalb einer Domain werden die Fehler über die Ereignis-Emitter freigegeben. Dadurch werden alle Fehler, Timer und Callback-Methoden implizit nur innerhalb der Domäne registriert. Durch einen Fehler ein Fehlerereignis senden und die Anwendung nicht zum Absturz bringen.

```
var domain = require("domain");
var d1 = domain.create();
var d2 = domain.create();

d1.run(function() {
    d2.add(setTimeout(function() {
        throw new Error("error on the timer of domain 2");
    }, 0));
});

d1.on("error", function(err) {
    console.log("error at domain 1: " + err);
});

d2.on("error", function(err) {
    console.log("error at domain 2: " + err);
});
```

Rückruf Hölle

Callback Hell (auch eine Pyramide des Untergangs oder Bumerang-Effekts) entsteht, wenn Sie zu viele Callback-Funktionen in eine Callback-Funktion einbetten. Hier ein Beispiel zum Lesen einer Datei (in ES6).

```
const fs = require('fs');
let filename = `${__dirname}/myfile.txt`;

fs.exists(filename, exists => {
    if (exists) {
        fs.stat(filename, (err, stats) => {
            if (err) {
                throw err;
            }
            if (stats.isFile()) {
                fs.readFile(filename, null, (err, data) => {
                    if (err) {
                        throw err;
                    }
                    console.log(data);
                });
            }
            else {
                throw new Error("This location contains not a file");
            }
        });
    }
    else {
        throw new Error("404: file not found");
    }
});
```

So vermeiden Sie "Callback Hell"

Es wird empfohlen, nicht mehr als 2 Rückruffunktionen zu verschachteln. Dies hilft Ihnen, die Lesbarkeit des Codes aufrechtzuerhalten, und ist in Zukunft viel einfacher zu pflegen. Wenn Sie mehr als zwei Rückrufe verschachteln müssen, verwenden Sie stattdessen verteilte Ereignisse .

Es gibt auch eine Bibliothek namens async , mit der Callbacks und deren Ausführung verwaltet werden können, die auf npm verfügbar sind. Es erhöht die Lesbarkeit des Rückrufcodes und gibt Ihnen mehr Kontrolle über Ihren Callback-Code-Fluss, einschließlich der Möglichkeit, diese parallel oder hintereinander auszuführen.

Native Versprechungen

v6.0.0

Versprechen sind ein Werkzeug für die asynchrone Programmierung. In JavaScript sind Versprechungen für ihre `then` Methoden bekannt. Versprechen haben zwei Hauptzustände "ausstehend" und "erledigt". Sobald ein Versprechen „erledigt“ ist, kann es nicht wieder „ausstehend“ sein. Dies bedeutet, dass Versprechen meistens für Ereignisse gut sind, die nur einmal vorkommen. Der "erledigte" Staat hat zwei Zustände, die auch "gelöst" und "abgelehnt" sind. Sie können ein neues Versprechen erstellen, indem Sie das `new` Schlüsselwort verwenden und eine Funktion an den Konstruktor `new Promise(function (resolve, reject) {})` .

Die an den Promise-Konstruktor übergebene Funktion erhält immer einen ersten und einen zweiten Parameter, die üblicherweise als `resolve` und `reject` bezeichnet werden. Die Benennung dieser beiden Parameter ist üblich, aber sie versprechen das Versprechen entweder in den Zustand "gelöst" oder "abgelehnt". Wenn eines dieser beiden genannt wird, geht das Versprechen von "ausstehend" auf "erledigt" über. `resolve` wird aufgerufen , wenn die gewünschte Aktion, die oft asynchron ist, durchgeführt worden ist und `reject` verwendet wird , wenn die Aktion errored hat.

Im unteren Bereich ist eine Funktion, die ein Versprechen zurückgibt.

```
function timeout (ms) {
  return new Promise(function (resolve, reject) {
    setTimeout(function () {
      resolve("It was resolved!");
    }, ms)
  });
}

timeout(1000).then(function (dataFromPromise) {
  // logs "It was resolved!"
  console.log(dataFromPromise);
})

console.log("waiting...");
```

Konsolenausgabe

```
waiting...
// << pauses for one second>>
It was resolved!
```

Wenn das Timeout aufgerufen wird, wird die an den Promise-Konstruktor übergebene Funktion ohne Verzögerung ausgeführt. Dann wird die Methode setTimeout ausgeführt und der Callback wird in den nächsten `ms` Millisekunden ausgelöst, in diesem Fall `ms=1000` . Da der Callback zum setTimeout noch nicht ausgelöst wird, gibt die Timeout-Funktion die Kontrolle an den aufrufenden Bereich zurück. Die Kette der `then` Methoden werden dann später genannt gespeichert werden , wenn / falls das Versprechen gelöst hat. Gäbe es `catch` hier Methoden würden sie auch gespeichert werden, sondern würden gefeuert werden , wenn / falls das Versprechen ‚ablehnt'.

Das Skript druckt dann "Warten ...". Eine Sekunde später ruft der setTimeout seinen Callback auf, der die Auflösungsfunktion mit der Zeichenfolge "Es wurde aufgelöst!" Aufruft. Dieser String wird dann in der vergangen `then` Methode Callback und wird dann den Benutzer angemeldet.

In dem gleichen Sinn können Sie die asynchrone Funktion setTimeout umschließen, die einen Rückruf erfordert. Sie können jede einzelne asynchrone Aktion mit einem Versprechen umschließen.

Weitere Informationen zu Versprechungen finden Sie in der JavaScript-Dokumentation Versprechungen .

Asynchrone Programmierung online lesen: https://riptutorial.com/de/node-js/topic/8813/asynchrone-programmierung

Kapitel 10: Ausführen von Dateien oder Befehlen mit untergeordneten Prozessen

Syntax

- child_process.exec (Befehl [, Optionen] [, Rückruf])
- child_process.execFile (datei [, args] [, options] [, callback])
- child_process.fork (modulePath [, args] [, options])
- child_process.spawn (Befehl [, Argumente] [, Optionen])
- child_process.execFileSync (datei [, args] [, options])
- child_process.execSync (Befehl [, Optionen])
- child_process.spawnSync (Befehl [, args] [, options])

Bemerkungen

Wenn Sie mit `ChildProcess` Prozessen arbeiten, geben alle asynchronen Methoden eine Instanz von `ChildProcess` , während alle synchronen Versionen die Ausgabe dessen `ChildProcess` , was ausgeführt wurde. Wie andere synchrone Operationen in Node.js, wenn ein Fehler auftritt, *wird* es werfen.

Examples

Einen neuen Prozess starten, um einen Befehl auszuführen

Verwenden Sie `child_process.spawn()` um einen neuen Prozess zu erzeugen, in dem Sie eine *ungepufferte* Ausgabe benötigen (z. B. lang laufende Prozesse, bei denen die Ausgabe über einen bestimmten Zeitraum gedruckt werden kann und nicht sofort `child_process.spawn()` .

Diese Methode erzeugt einen neuen Prozess mit einem gegebenen Befehl und einem Array von Argumenten. Der Rückgabewert ist eine Instanz von `ChildProcess` , die wiederum die Eigenschaften `stdout` und `stderr` bereitstellt. Beide Streams sind Instanzen von `stream.Readable` .

Der folgende Code entspricht der Ausführung des Befehls `ls -lh /usr` .

```
const spawn = require('child_process').spawn;
const ls = spawn('ls', ['-lh', '/usr']);

ls.stdout.on('data', (data) => {
  console.log(`stdout: ${data}`);
});

ls.stderr.on('data', (data) => {
  console.log(`stderr: ${data}`);
});

ls.on('close', (code) => {
```

```
  console.log(`child process exited with code ${code}`);
});
```

Ein weiterer Beispielbefehl:

```
zip -0vr "archive" ./image.png
```

Könnte geschrieben werden als:

```
spawn('zip', ['-0vr', '"archive"', './image.png']);
```

Eine Shell starten, um einen Befehl auszuführen

Um einen Befehl in einer Shell auszuführen, in der Sie eine gepufferte Ausgabe benötigen (dh es ist kein Stream), verwenden Sie `child_process.exec` . Wenn Sie beispielsweise den Befehl `cat *.js file | wc -l` , ohne Optionen, das würde so aussehen:

```
const exec = require('child_process').exec;
exec('cat *.js file | wc -l', (err, stdout, stderr) => {
  if (err) {
    console.error(`exec error: ${err}`);
    return;
  }

  console.log(`stdout: ${stdout}`);
  console.log(`stderr: ${stderr}`);
});
```

Die Funktion akzeptiert bis zu drei Parameter:

```
child_process.exec(command[, options][, callback]);
```

Der Befehlsparameter ist eine Zeichenfolge und ist erforderlich, während das Optionsobjekt und der Rückruf optional sind. Wenn kein Optionsobjekt angegeben ist, verwendet `exec` standardmäßig Folgendes:

```
{
  encoding: 'utf8',
  timeout: 0,
  maxBuffer: 200*1024,
  killSignal: 'SIGTERM',
  cwd: null,
  env: null
}
```

Das Optionsobjekt unterstützt auch einen `shell` Parameter (standardmäßig `/bin/sh` unter UNIX und `cmd.exe` unter Windows), eine `uid` Option zum Festlegen der Benutzeridentität des Prozesses und eine `gid` Option für die Gruppenidentität.

Der Rückruf, der aufgerufen wird, wenn der Befehl ausgeführt wurde, wird mit den drei

Argumenten `(err, stdout, stderr)` aufgerufen. Wenn der Befehl erfolgreich ausgeführt wird, ist `err` `null` , andernfalls ist es eine Instanz von `Error` , wobei `err.code` der `err.code` des Prozesses und `err.signal` das Signal ist, das zur Beendigung gesendet wurde.

Die Argumente `stdout` und `stderr` sind die Ausgabe des Befehls. Es wird mit der im options-Objekt angegebenen Kodierung dekodiert (Standard: `string`), kann aber ansonsten als `Buffer` werden.

Es gibt auch eine synchrone Version von `exec` , nämlich `execSync` . Die synchrone Version nimmt keinen Rückruf an und gibt `stdout` anstelle einer Instanz von `ChildProcess` . Wenn bei der synchronen Version ein Fehler auftritt, *wird* Ihr Programm ausgelöst und angehalten. Es sieht aus wie das:

```
const execSync = require('child_process').execSync;
const stdout = execSync('cat *.js file | wc -l');
console.log(`stdout: ${stdout}`);
```

Einen Prozess starten, um eine ausführbare Datei auszuführen

Wenn Sie eine Datei ausführen `child_process.execFile` , beispielsweise eine ausführbare Datei, verwenden Sie `child_process.execFile` . Anstatt eine Shell wie `child_process.exec` , wird direkt ein neuer Prozess erstellt, der etwas effizienter ist als die Ausführung eines Befehls. Die Funktion kann wie folgt verwendet werden:

```
const execFile = require('child_process').execFile;
const child = execFile('node', ['--version'], (err, stdout, stderr) => {
  if (err) {
    throw err;
  }

  console.log(stdout);
});
```

Im Gegensatz zu `child_process.exec` akzeptiert diese Funktion bis zu vier Parameter, wobei der zweite Parameter ein Array von Argumenten ist, die Sie der ausführbaren Datei übergeben möchten:

```
child_process.execFile(file[, args][, options][, callback]);
```

Ansonsten sind die Optionen und das Callback-Format ansonsten identisch mit `child_process.exec` . Gleiches gilt für die synchrone Version der Funktion:

```
const execFileSync = require('child_process').execFileSync;
const stdout = execFileSync('node', ['--version']);
console.log(stdout);
```

Ausführen von Dateien oder Befehlen mit untergeordneten Prozessen online lesen: https://riptutorial.com/de/node-js/topic/2726/ausfuhren-von-dateien-oder-befehlen-mit-untergeordneten-prozessen

Kapitel 11: Ausnahmebehandlung

Examples

Behandlung von Ausnahmen in Node.Js

Node.js hat drei grundlegende Möglichkeiten, Ausnahmen / Fehler zu behandeln:

1. **try** - **catch** block
2. **Fehler** als erstes Argument für einen `callback`
3. `emit` ein **Fehlerereignis** unter Verwendung eventEmitter

try-catch wird verwendet, um die Ausnahmen der synchronen Codeausführung abzufangen. Wenn der Anrufer (oder der Anrufer des Anrufers ...) try / catch verwendet hat, kann er den Fehler abfangen. Wenn keiner der Anrufer Try-Catch hatte, stürzt das Programm ab.
Wenn Try-Catch für einen asynchronen Vorgang verwendet wird und eine Ausnahme vom Rückruf der asynchronen Methode ausgelöst wurde, wird sie nicht von try-catch abgefangen. Um eine Ausnahme von einem asynchronen Vorgangscallback abzufangen, werden vorzugsweise *Versprechungen verwendet* .
Beispiel, um es besser zu verstehen

```
// ** Example - 1  **
function doSomeSynchronousOperation(req, res) {
    if(req.body.username === ''){
        throw new Error('User Name cannot be empty');
    }
    return true;
}

// calling the method above
try {
    // synchronous code
    doSomeSynchronousOperation(req, res)
catch(e) {
    //exception handled here
    console.log(e.message);
}

// ** Example - 2 **
function doSomeAsynchronousOperation(req, res, cb) {
    // imitating async operation
    return setTimeout(function(){
        cb(null, []);
    },1000);
}

try {
    // asynchronous code
    doSomeAsynchronousOperation(req, res, function(err, rs){
        throw new Error("async operation exception");
    })
} catch(e) {
     // Exception will not get handled here
```

```
     console.log(e.message);
}
// The exception is unhandled and hence will cause application to break
```

Callbacks werden meistens in Node.js verwendet, da Callback ein Ereignis asynchron liefert. Der Benutzer übergibt Ihnen eine Funktion (den Rückruf), und Sie rufen sie später auf, wenn der asynchrone Vorgang abgeschlossen ist.
Das übliche Muster ist, dass der Rückruf als *Rückruf (err, result)* aufgerufen wird, wobei nur err und result nicht null sind, je nachdem, ob die Operation erfolgreich war oder fehlgeschlagen ist.

```
function doSomeAsynchronousOperation(req, res, callback) {
   setTimeout(function(){
        return callback(new Error('User Name cannot be empty'));
   }, 1000);
   return true;
}

doSomeAsynchronousOperation(req, res, function(err, result) {
   if (err) {
       //exception handled here
       console.log(err.message);
   }

   //do some stuff with valid data
});
```

emit In komplizierteren Fällen kann die Funktion selbst anstelle eines Rückrufs ein EventEmitter-Objekt zurückgeben, und es wird erwartet, dass der Aufrufer auf Fehlerereignisse auf dem Emitter wartet.

```
const EventEmitter = require('events');

function doSomeAsynchronousOperation(req, res) {
    let myEvent = new EventEmitter();

    // runs asynchronously
    setTimeout(function(){
        myEvent.emit('error', new Error('User Name cannot be empty'));
    }, 1000);

    return myEvent;
}

// Invoke the function
let event = doSomeAsynchronousOperation(req, res);

event.on('error', function(err) {
    console.log(err);
});

event.on('done', function(result) {
    console.log(result); // true
});
```

Unhanded Exception Management

Da Node.js in einem einzelnen Prozess ausgeführt wird, müssen nicht erfasste Ausnahmen beim Entwickeln von Anwendungen berücksichtigt werden.

Ausnahmen stumm behandeln

Die meisten Leute lassen die Fehler von node.js-Servern unbemerkt verschlingen.

- Die Ausnahme wird stumm behandelt

```
process.on('uncaughtException', function (err) {
  console.log(err);
});
```

Das ist schlecht , es wird aber funktionieren:

- Die Ursache wird weiterhin unbekannt bleiben, da dies nicht zur Lösung der Ursache der Ausnahme (Fehler) beiträgt.
- Wenn eine Datenbankverbindung (Pool) aus irgendeinem Grund geschlossen wird, führt dies zu einer ständigen Fehlerweitergabe. Dies bedeutet, dass der Server ausgeführt wird, die Verbindung zu db jedoch nicht wiederhergestellt wird.

Rückkehr zum Ausgangszustand

Im Falle einer "uncaughtException" empfiehlt es sich, den Server neu zu starten und in den **Ausgangszustand zu versetzen** , von dem wir wissen, dass er funktionieren wird. Eine Ausnahme wird protokolliert, die Anwendung wird beendet. Da sie jedoch in einem Container ausgeführt wird, der sicherstellt, dass der Server ausgeführt wird, wird ein Neustart des Servers (Rückkehr zum ursprünglichen Betriebszustand) erreicht.

- Installieren Sie das forever (oder ein anderes CLI-Tool, um sicherzustellen, dass der Knotenserver kontinuierlich ausgeführt wird).

```
npm install forever -g
```

- Starten Sie den Server für immer

```
forever start app.js
```

  Grund, warum es gestartet wird und warum wir für immer verwenden, ist, nachdem der Server für immer **beendet wurde. Der** Prozess startet den Server erneut.

- Server neu starten

```
process.on('uncaughtException', function (err) {
    console.log(err);
```

```
    // some logging mechanisam
    // ....

    process.exit(1); // terminates process
});
```

Nebenbei gab es eine Möglichkeit, Ausnahmen mit **Clustern und Domänen zu behandeln** .

Domains sind hier mehr Informationen veraltet.

Fehler und Versprechen

Versprechungen behandeln Fehler anders als synchroner oder Callback-Code.

```
const p = new Promise(function (resolve, reject) {
    reject(new Error('Oops'));
});

// anything that is `reject`ed inside a promise will be available through catch
// while a promise is rejected, `.then` will not be called
p
    .then(() => {
        console.log("won't be called");
    })
    .catch(e => {
        console.log(e.message); // output: Oops
    })
    // once the error is caught, execution flow resumes
    .then(() => {
        console.log('hello!'); // output: hello!
    });
```

Derzeit führen Fehler, die in einem Versprechen eingeworfen werden und nicht erfasst werden, dazu, dass der Fehler verschluckt wird, was die Ermittlung des Fehlers erschweren kann. Dies kann gelöst mit Fusseln Tool wie eslint oder indem sichergestellt haben Sie immer eine `catch` -Klausel.

Dieses Verhalten wird in Knoten 8 zugunsten der Beendigung des Knotenprozesses abgelehnt.

Ausnahmebehandlung online lesen: https://riptutorial.com/de/node-js/topic/2819/ausnahmebehandlung

Kapitel 12: Befehlszeilenargumente analysieren

Examples

Aktion (Verb) und Werte übergeben

```
const options = require("commander");

options
  .option("-v, --verbose", "Be verbose");

options
  .command("convert")
  .alias("c")
  .description("Converts input file to output file")
  .option("-i, --in-file <file_name>", "Input file")
  .option("-o, --out-file <file_name>", "Output file")
  .action(doConvert);

options.parse(process.argv);

if (!options.args.length) options.help();

function doConvert(options){
    //do something with options.inFile and options.outFile
};
```

Boolesche Schalter übergeben

```
const options = require("commander");

options
  .option("-v, --verbose")
  .parse(process.argv);

if (options.verbose){
  console.log("Let's make some noise!");
}
```

Befehlszeilenargumente analysieren online lesen: https://riptutorial.com/de/node-js/topic/6174/befehlszeilenargumente-analysieren

Kapitel 13: Bei Änderungen automatisch laden

Examples

Quellcode-Änderungen mit nodemon automatisch laden

Das nodemon-Paket ermöglicht es Ihnen, Ihr Programm automatisch neu zu laden, wenn Sie Dateien im Quellcode ändern.

Nodemon global installieren

`npm install -g nodemon` (oder `npm i -g nodemon`)

Nodemon lokal installieren

Falls Sie es nicht global installieren möchten

`npm install --save-dev nodemon` (oder `npm i -D nodemon`)

Nodemon verwenden

Führen Sie Ihr Programm mit `nodemon entry.js` (oder `nodemon entry`) aus.

Dies ersetzt die übliche Verwendung von `node entry.js` (oder `node entry`).

Sie können Ihr nodemon-Startup auch als npm-Skript hinzufügen. Dies kann nützlich sein, wenn Sie Parameter angeben und nicht jedes Mal eingeben müssen.

Fügen Sie **package.json hinzu:**

```
"scripts": {
  "start": "nodemon entry.js -devmode -something 1"
}
```

Auf diese Weise können Sie `npm start` einfach von Ihrer Konsole aus verwenden.

Browsersync

Überblick

Browsersync ist ein Tool, mit dem Live-Dateien betrachtet und Browser neu geladen werden können. Es ist als NPM-Paket erhältlich .

Installation

Um Browsersync zu installieren, müssen Sie zunächst Node.js und NPM installieren. Weitere Informationen finden Sie in der SO-Dokumentation zum Installieren und Ausführen von Node.js.

Sobald Ihr Projekt eingerichtet ist, können Sie Browsersync mit dem folgenden Befehl installieren:

```
$ npm install browser-sync -D
```

Dadurch wird Browsersync im lokalen `node_modules` Verzeichnis installiert und in Ihren Entwicklerabhängigkeiten `node_modules` .

Wenn Sie es global installieren möchten, verwenden Sie anstelle der `-g -D` Flag `-g` .

Windows-Benutzer

Wenn Sie Probleme bei der Installation von Browsersync unter Windows haben, müssen Sie möglicherweise Visual Studio installieren, damit Sie auf die Build-Tools zur Installation von Browsersync zugreifen können. Sie müssen dann die verwendete Version von Visual Studio wie folgt angeben:

```
$ npm install browser-sync --msvs_version=2013 -D
```

Dieser Befehl gibt die Version von Visual Studio 2013 an.

Grundlegende Verwendung

Um Ihre Site automatisch neu zu laden, wenn Sie eine JavaScript-Datei in Ihrem Projekt ändern, verwenden Sie den folgenden Befehl:

```
$ browser-sync start --proxy "myproject.dev" --files "**/*.js"
```

Ersetzen Sie `myproject.dev` durch die Webadresse, die Sie für den Zugriff auf Ihr Projekt verwenden. Browsersync gibt eine alternative Adresse aus, über die über den Proxy auf Ihre Site zugegriffen werden kann.

Fortgeschrittene Verwendung

Neben der oben beschriebenen Befehlszeilenschnittstelle kann Browsersync auch mit Grunt.js und Gulp.js verwendet werden .

Grunt.js

Die Verwendung von Grunt.js erfordert ein Plugin, das wie folgt installiert werden kann:

```
$ npm install grunt-browser-sync -D
```

Dann fügen Sie diese Zeile zu Ihrer `gruntfile.js` :

```
grunt.loadNpmTasks('grunt-browser-sync');
```

Gulp.js

Browsersync arbeitet als CommonJS- Modul, sodass kein Gulp.js-Plugin erforderlich ist. Fordern Sie einfach das Modul so an:

```
var browserSync = require('browser-sync').create();
```

Sie können jetzt die Browsersync-API verwenden , um sie an Ihre Anforderungen anzupassen.

API

Die Browsersync-API finden Sie hier: https://browsersync.io/docs/api

Bei Änderungen automatisch laden online lesen: https://riptutorial.com/de/node-js/topic/1743/bei-anderungen-automatisch-laden

Kapitel 14: Benötigen()

Einführung

Diese Dokumentation konzentriert sich auf die Erläuterung der Verwendung und der von NodeJS in ihrer Sprache eingeschlossenen Anweisung request `require()` .

Require ist ein Import bestimmter Dateien oder Pakete, die mit NodeJS-Modulen verwendet werden. Es wird verwendet, um die Codestruktur und -nutzung zu verbessern. `require()` wird für Dateien verwendet, die lokal installiert werden, mit einer direkten Route von der Datei, die `require` .

Syntax

- module.exports = {testFunction: testFunction};
- var test_file = required ('./ testFile.js'); // Lassen Sie uns eine Datei namens `testFile`
- test_file.testFunction (unsere_Daten); // Lass `testFile` die Funktion `testFunction`

Bemerkungen

Die Verwendung von `require()` ermöglicht die Strukturierung von Code ähnlich wie bei der Verwendung von Klassen und öffentlichen Methoden in Java. Wenn eine Funktion " `.export` " ist, kann es `require` , dass sie in einer anderen Datei verwendet wird. Wenn eine Datei nicht `.export` , kann sie nicht in einer anderen Datei verwendet werden.

Examples

Beginn der Anforderung () mit einer Funktion und Datei

Require ist eine Anweisung, die Node in gewissem Sinne als `getter` Funktion interpretiert. Angenommen, Sie haben eine Datei mit dem Namen `analysis.js` , und das Innere Ihrer Datei sieht folgendermaßen aus:

```
function analyzeWeather(weather_data) {
  console.log('Weather information for ' + weather_data.time + ': ');
  console.log('Rainfall: ' + weather_data.precip);
  console.log('Temperature: ' + weather_data.temp);
  //More weather_data analysis/printing...
}
```

Diese Datei enthält nur die Methode `analyzeWeather(weather_data)` . Wenn Sie diese Funktion verwenden möchten, muss sie entweder in dieser Datei verwendet werden oder in die Datei kopiert werden, von der sie verwendet werden soll. Node hat jedoch ein sehr nützliches Tool für die Code- und Dateiorganisation, also für Module, enthalten .

Um unsere Funktion nutzen zu können, müssen wir die Funktion zuerst durch eine Anweisung am Anfang `export` . Unsere neue Datei sieht so aus,

```
module.exports = {
  analyzeWeather: analyzeWeather
}
function analyzeWeather(weather_data) {
  console.log('Weather information for ' + weather_data.time + ': ');
  console.log('Rainfall: ' + weather_data.precip);
  console.log('Temperature: ' + weather_data.temp);
  //More weather_data analysis/printing...
}
```

Mit dieser kleinen Anweisung von `module.exports` kann unsere Funktion jetzt außerhalb der Datei verwendet werden. Alles, was Sie noch tun müssen, ist zu verwenden `require()` .

Wenn `require` eine Funktion oder Datei ing, ist die Syntax sehr ähnlich. Dies wird normalerweise am Anfang der Datei ausgeführt und auf `var` oder `const` , um sie in der gesamten Datei zu verwenden. Zum Beispiel haben wir eine andere Datei (auf derselben Ebene wie `analyze.js` namens `handleWeather.js` , die folgendermaßen aussieht:

```
const analysis = require('./analysis.js');

weather_data = {
  time: '01/01/2001',
  precip: 0.75,
  temp: 78,
  //More weather data...
};
analysis.analyzeWeather(weather_data);
```

In dieser Datei verwenden wir `require()` , um unsere Datei `analysis.js` zu packen. Bei Verwendung rufen wir nur die Variable oder Konstante auf, die dieser `require` zugewiesen ist, und verwenden die Funktion, die exportiert wird.

Beginnen Sie mit request () mit einem NPM-Paket

Knotens `require` ist auch sehr hilfreich , wenn im Tandem mit einem verwendet NPM - Paket . Nehmen wir zum Beispiel, möchten Sie die NPM - Paket verwenden `require` in einer Datei mit dem Namen `getWeather.js` . Nachdem NPM Ihr Paket über Ihre Befehlszeile (`git install request`) `git install request` , können Sie es verwenden. Ihre `getWeather.js` Datei könnte folgendermaßen aussehen:

```
var https = require('request');

//Construct your url variable...
https.get(url, function(error, response, body) {
  if (error) {
    console.log(error);
  } else {
    console.log('Response => ' + response);
    console.log('Body => ' + body);
  }
```

```
});
```

Wenn diese Datei ausgeführt wird, ist es zunächst `require` ,s (Importe) das Paket , das Sie gerade genannt installiert `request` . In der `request` gibt es viele Funktionen, auf die Sie jetzt zugreifen können, von denen eine als `get` . In den nächsten Zeilen wird die Funktion verwendet, um eine HTTP-GET-Anforderung zu erstellen .

Benötigen() online lesen: https://riptutorial.com/de/node-js/topic/10742/benotigen--

Kapitel 15: Bereitstellen von Node.js-Anwendungen in der Produktion

Examples

NODE_ENV = "Produktion" einstellen

Produktionsbereitstellungen variieren in vielerlei Hinsicht. Eine Standardkonvention bei der Bereitstellung in der Produktion besteht darin, eine Umgebungsvariable namens `NODE_ENV` zu definieren und den Wert auf *"Produktion" zu setzen* .

Laufzeitflags

Jeder in Ihrer Anwendung ausgeführte Code (einschließlich externer Module) kann den Wert von `NODE_ENV` :

```
if(process.env.NODE_ENV === 'production') {
    // We are running in production mode
} else {
   // We are running in development mode
}
```

Abhängigkeiten

Wenn die Umgebungsvariable `NODE_ENV` auf *'production' gesetzt ist, werden* alle `devDependencies` in Ihrer *package.json*- Datei beim Ausführen von `npm install` vollständig ignoriert. Sie können dies auch mit einem `--production` Flag `--production` :

```
npm install --production
```

Zur Einstellung von `NODE_ENV` Sie eine dieser Methoden verwenden

Methode 1: Setze NODE_ENV für alle Knoten-Apps

Windows:

```
set NODE_ENV=production
```

Linux oder anderes Unix-basiertes System:

```
export NODE_ENV=production
```

Dadurch wird `NODE_ENV` für die aktuelle Bash-Sitzung festgelegt. `NODE_ENV` für alle Apps, die nach

dieser Anweisung gestartet werden, `NODE_ENV` auf `production` .

Methode 2: Setze NODE_ENV für die aktuelle App

```
NODE_ENV=production node app.js
```

Dadurch wird `NODE_ENV` für die aktuelle App festgelegt. Dies ist hilfreich, wenn wir unsere Apps in verschiedenen Umgebungen testen möchten.

Methode 3: Erstellen `.env` eine `.env` Datei und verwenden Sie sie

Hierfür wird die hier erläuterte Idee verwendet. In diesem Beitrag finden Sie eine ausführlichere Erklärung.

Grundsätzlich erstellen Sie eine `.env` Datei und führen ein bash-Skript aus, um sie in der Umgebung `.env` .

Um das Schreiben eines Bash-Skripts zu vermeiden, können Sie mit dem Paket env- `.env` die in der `.env` Datei definierten Umgebungsvariablen `.env` .

```
env-cmd .env node app.js
```

Methode 4: Verwenden Sie das `cross-env` Paket

Mit diesem Paket können Umgebungsvariablen für jede Plattform auf eine Weise festgelegt werden.

Nach der Installation mit npm können Sie es `package.json` wie folgt zu Ihrem Bereitstellungsskript in `package.json` hinzufügen:

```
"build:deploy": "cross-env NODE_ENV=production webpack"
```

App mit dem Prozessmanager verwalten

Es ist empfehlenswert, NodeJS-Apps, die von Prozessmanagern gesteuert werden, auszuführen. Der Prozessmanager hilft, die Anwendung für immer am Leben zu erhalten, bei einem Neustart neu zu starten, ohne Ausfallzeiten neu zu laden und die Verwaltung zu vereinfachen. Die leistungsstärksten von ihnen (wie PM2) verfügen über einen integrierten Lastausgleich. Mit PM2 können Sie auch Anwendungsprotokollierung, -überwachung und -cluster verwalten.

PM2-Prozessmanager

PM2 installieren:

```
npm install pm2 -g
```

Der Prozess kann im Cluster-Modus mit integriertem Load Balancer gestartet werden, um die Last zwischen den Prozessen zu verteilen:

`pm2 start app.js -i 0 --name "api"` (*-i* gibt die Anzahl der zu `pm2 start app.js -i 0 --name "api"` Prozesse an. Wenn es 0 ist, basiert die Prozessnummer auf der CPU-Kernenanzahl)

Bei mehreren Benutzern in der Produktion muss es für PM2 einen einzigen Punkt geben. Dem pm2-Befehl muss daher ein Speicherort vorangestellt werden (für die PM2-Konfiguration). Andernfalls wird für jeden Benutzer mit der Konfiguration in dem entsprechenden Basisverzeichnis ein neuer pm2-Prozess erzeugt. Und es wird inkonsistent sein.

Verwendung: `PM2_HOME=/etc/.pm2 pm2 start app.js`

Bereitstellung mit PM2

`PM2` ist ein Produktionsprozessmanager für `Node.js` Anwendungen, mit dem Sie Anwendungen für immer am Leben erhalten und ohne Ausfallzeiten neu laden können. Mit PM2 können Sie auch Anwendungsprotokollierung, -überwachung und -cluster verwalten.

Installieren Sie `pm2` global.

```
npm install -g pm2
```

Führen Sie dann die `node.js` App mit `PM2`.

```
pm2 start server.js --name "my-app"
```

```
$ pm2 start app.js --name my-app
[PM2] restartProcessId process id 0
┌──────────┬────┬──────┬───────┬────────┬─────────┬────────┬────────────┬──────────┐
│ App name │ id │ mode │ pid   │ status │ restart │ uptime │ memory     │ watching │
├──────────┼────┼──────┼───────┼────────┼─────────┼────────┼────────────┼──────────┤
│ my-app   │ 0  │ fork │ 64029 │ online │ 1       │ 0s     │ 17.816 MB  │ disabled │
└──────────┴────┴──────┴───────┴────────┴─────────┴────────┴────────────┴──────────┘
 Use the `pm2 show <id|name>` command to get more details about an app.
```

Die folgenden Befehle sind nützlich, wenn Sie mit `PM2` .

Alle laufenden Prozesse auflisten:

```
pm2 list
```

Stoppen Sie eine App:

```
pm2 stop my-app
```

App neu starten:

```
pm2 restart my-app
```

So zeigen Sie detaillierte Informationen zu einer App an:

```
pm2 show my-app
```

So entfernen Sie eine App aus der Registrierung von PM2:

```
pm2 delete my-app
```

Bereitstellung mit dem Prozessmanager

Prozessmanager wird im Allgemeinen in der Produktion verwendet, um eine nodejs-App bereitzustellen. Die Hauptfunktionen eines Prozessmanagers sind ein Neustart des Servers bei einem Absturz, Überprüfen des Ressourcenverbrauchs, Verbesserung der Laufzeitleistung, Überwachung usw.

Einige der populärsten Prozessmanager, die von der Knoten-Community erstellt werden, sind für immer PM2 usw.

Forvever

`forever` ist ein Befehlszeilenschnittstellentool, um sicherzustellen, dass ein bestimmtes Skript kontinuierlich ausgeführt wird. Die einfache Benutzeroberfläche von `forever` macht es ideal für kleinere Implementierungen von `Node.js` Apps und -Skripts.

`forever` überwacht Ihren Prozess und startet ihn neu, wenn er abstürzt.

Für `forever` global installieren.

```
$ npm install -g forever
```

Anwendung ausführen:

```
$ forever start server.js
```

Dadurch wird der Server gestartet und eine ID für den Prozess angegeben (beginnt bei 0).

Applikation neustarten :

```
$ forever restart 0
```

Hier ist `0` die ID des Servers.

Anwendung stoppen:

```
$ forever stop 0
```

Ähnlich wie beim Neustart ist `0` die ID des Servers. Sie können auch eine Prozess-ID oder einen

Skriptnamen anstelle der von forever angegebenen ID eingeben.

Für weitere Befehle: https://www.npmjs.com/package/forever

Verwenden verschiedener Eigenschaften / Konfigurationen für verschiedene Umgebungen wie dev, qa, staging usw.

Große Anwendungen erfordern oft unterschiedliche Eigenschaften, wenn sie in unterschiedlichen Umgebungen ausgeführt werden. Wir können dies erreichen, indem Sie Argumente an die NodeJs-Anwendung übergeben und dasselbe Argument im Node-Prozess verwenden, um bestimmte Umgebungseigenschaftsdateien zu laden.

Angenommen, wir haben zwei Eigenschaftsdateien für unterschiedliche Umgebungen.

- dev.json

```
{
    "PORT": 3000,
    "DB": {
        "host": "localhost",
        "user": "bob",
        "password": "12345"
    }
}
```

- qa.json

```
{
    "PORT": 3001,
    "DB": {
        "host": "where_db_is_hosted",
        "user": "bob",
        "password": "54321"
    }
}
```

Der folgende Code in der Anwendung wird die entsprechende Eigenschaftsdatei exportieren, die wir verwenden möchten.

```
process.argv.forEach(function (val) {
    var arg = val.split("=");
    if (arg.length > 0) {
        if (arg[0] === 'env') {
            var env = require('./' + arg[1] + '.json');
            exports.prop = env;
        }
    }
});
```

Wir geben der Bewerbung folgende Argumente

```
node app.js env=dev
```

Wenn wir Prozessmanager *für immer verwenden, dann ist* es so einfach wie

```
forever start app.js env=dev
```

Cluster nutzen

Eine einzelne Instanz von Node.js wird in einem einzelnen Thread ausgeführt. Um Multi-Core-Systeme zu nutzen, möchte der Benutzer manchmal ein Cluster von Node.js-Prozessen starten, um die Last zu handhaben.

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

    var numCPUs = require('os').cpus().length;

        if (cluster.isMaster) {
          // In real life, you'd probably use more than just 2 workers,
          // and perhaps not put the master and worker in the same file.
          //
          // You can also of course get a bit fancier about logging, and
          // implement whatever custom logic you need to prevent DoS
          // attacks and other bad behavior.
          //
          // See the options in the cluster documentation.
          //
          // The important thing is that the master does very little,
          // increasing our resilience to unexpected errors.
            console.log('your server is working on ' + numCPUs + ' cores');

             for (var i = 0; i < numCPUs; i++) {
                    cluster.fork();
                  }

          cluster.on('disconnect', function(worker) {
            console.error('disconnect!');
            //clearTimeout(timeout);
            cluster.fork();
          });


        } else {
            require('./app.js');

        }
```

Bereitstellen von Node.js-Anwendungen in der Produktion online lesen: https://riptutorial.com/de/node-js/topic/2975/bereitstellen-von-node-js-anwendungen-in-der-produktion

Kapitel 16: Bereitstellung der Node.js-Anwendung ohne Ausfallzeiten

Examples

Bereitstellung mit PM2 ohne Ausfallzeiten.

`ecosystem.json`

```
{
    "name": "app-name",
    "script": "server",
    "exec_mode": "cluster",
    "instances": 0,
    "wait_ready": true
    "listen_timeout": 10000,
    "kill_timeout": 5000,
}
```

```
wait_ready
```

Warten Sie nicht erneut auf das Warten auf Abhörereignis, sondern warten Sie auf process.send ('ready');

```
listen_timeout
```

Zeit in ms vor dem Erzwingen eines erneuten Ladens, wenn die App nicht auf sie wartet.

```
kill_timeout
```

Zeit in ms vor dem Senden einer endgültigen SIGKLL.

`server.js`

```
const http = require('http');
const express = require('express');

const app = express();
const server = http.Server(app);
const port = 80;

server.listen(port, function() {
    process.send('ready');
});

process.on('SIGINT', function() {
    server.close(function() {
        process.exit(0);
    });
});
```

Möglicherweise müssen Sie warten, bis Ihre Anwendung Verbindungen zu Ihren DBs / caches / workers / was auch immer hergestellt hat. PM2 muss warten, bevor Ihre Bewerbung als online betrachtet wird. Dazu müssen Sie `wait_ready: true` in einer Prozessdatei `wait_ready: true` . Dadurch wird PM2 auf dieses Ereignis hören. In Ihrer Anwendung müssen Sie `process.send('ready');` hinzufügen `process.send('ready');` wenn Sie möchten, dass Ihre Bewerbung als fertig betrachtet wird.

Wenn ein Prozess von PM2 gestoppt / erneut gestartet wird, werden einige Systemsignale in einer bestimmten Reihenfolge an Ihren Prozess gesendet.

Zuerst wird ein `SIGINT` an Ihre Prozesse gesendet, ein Signal, das Sie erkennen können, dass Ihr Prozess gestoppt wird. Wenn Ihre Anwendung nicht vor 1.6s (anpassbar) beendet wird, erhält sie ein `SIGKILL` Signal, um den Prozess zu beenden. Wenn Ihre Anwendung also Statuszustände oder Jobs bereinigen muss, können Sie das `SIGINT` Signal `SIGINT` , um Ihre Anwendung auf das Beenden vorbereiten.

Bereitstellung der Node.js-Anwendung ohne Ausfallzeiten online lesen: https://riptutorial.com/de/node-js/topic/9752/bereitstellung-der-node-js-anwendung-ohne-ausfallzeiten

Kapitel 17: Bluebird Versprechen

Examples

Nodeback-Bibliothek in Versprechen konvertieren

```
const Promise = require('bluebird'),
      fs = require('fs')

Promise.promisifyAll(fs)

// now you can use promise based methods on 'fs' with the Async suffix
fs.readFileAsync('file.txt').then(contents => {
  console.log(contents)
}).catch(err => {
  console.error('error reading', err)
})
```

Funktionale Versprechen

Beispiel einer Karte:

```
Promise.resolve([ 1, 2, 3 ]).map(el => {
   return Promise.resolve(el * el) // return some async operation in real world
})
```

Beispiel für einen Filter:

```
Promise.resolve([ 1, 2, 3 ]).filter(el => {
  return Promise.resolve(el % 2 === 0) // return some async operation in real world
}).then(console.log)
```

Beispiel reduzieren:

```
Promise.resolve([ 1, 2, 3 ]).reduce((prev, curr) => {
  return Promise.resolve(prev + curr) // return some async operation in real world
}).then(console.log)
```

Coroutinen (Generatoren)

```
const promiseReturningFunction = Promise.coroutine(function* (file) {
  const data = yield fs.readFileAsync(file) // this returns a Promise and resolves to the file
contents

  return data.toString().toUpperCase()
})

promiseReturningFunction('file.txt').then(console.log)
```

Automatische Ressourcenentsorgung (Promise.using)

```
function somethingThatReturnsADisposableResource() {
  return getSomeResourceAsync(...).disposer(resource => {
    resource.dispose()
  })
}

Promise.using(somethingThatReturnsADisposableResource(), resource => {
  // use the resource here, the disposer will automatically close it when Promise.using exits
})
```

In Serie ausführen

```
Promise.resolve([1, 2, 3])
  .mapSeries(el => Promise.resolve(el * el)) // in real world, use Promise returning async
function
  .then(console.log)
```

Bluebird Versprechen online lesen: https://riptutorial.com/de/node-js/topic/6728/bluebird-versprechen

Kapitel 18: Callback Hölle vermeiden

Examples

Async-Modul

Die Quelle kann von GitHub heruntergeladen werden. Alternativ können Sie mit npm installieren:

$ npm install --save async

Neben der Verwendung von Bower:

$ bower async installieren

Beispiel:

```
var async = require("async");
async.parallel([
    function(callback) { ... },
    function(callback) { ... }
], function(err, results) {
    // optional callback
});
```

Async-Modul

Zum Glück gibt es Bibliotheken wie Async.js, um das Problem einzudämmen. Async fügt Ihrem Code eine dünne Schicht von Funktionen hinzu, kann jedoch die Komplexität erheblich reduzieren, indem die Rückrufverschachtelung vermieden wird.

In Async gibt es viele Hilfsmethoden, die in verschiedenen Situationen verwendet werden können, z. B. Serien, Parallel, Wasserfall usw. Jede Funktion hat einen bestimmten Anwendungsfall. Nehmen Sie sich also etwas Zeit, um herauszufinden, welche in welchen Situationen hilfreich ist.

So gut wie Async auch ist, es ist nicht perfekt. Es ist sehr einfach, sich durch das Kombinieren von Serien, Parallelen, für immer usw. mitreißen zu lassen. Dann sind Sie wieder da, wo Sie mit chaotischem Code angefangen haben. Seien Sie vorsichtig, um nicht vorzeitig zu optimieren. Nur weil einige asynchrone Aufgaben parallel ausgeführt werden können, müssen sie nicht immer ausgeführt werden. Da der Knoten nur aus einem einzigen Thread besteht, führt die parallele Ausführung von Tasks bei Verwendung von Async kaum zu einer Leistungssteigerung.

Die Quelle steht unter https://github.com/caolan/async zum Download zur Verfügung. Alternativ können Sie mit npm installieren:

$ npm install --save async

Neben der Verwendung von Bower:

$ bower async installieren

Asyncs Wasserfall Beispiel:

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

var myFile = '/tmp/test';

async.waterfall([
    function(callback) {
        fs.readFile(myFile, 'utf8', callback);
    },
    function(txt, callback) {
        txt = txt + '\nAppended something!';
        fs.writeFile(myFile, txt, callback);
    }
], function (err, result) {
    if(err) return console.log(err);
    console.log('Appended text!');
});
```

Callback Hölle vermeiden online lesen: https://riptutorial.com/de/node-js/topic/10045/callback-holle-vermeiden

Kapitel 19: Cassandra-Integration

Examples

Hallo Welt

Für den Zugriff auf Cassandra kann das Cassandra `cassandra-driver` von DataStax verwendet werden. Es unterstützt alle Funktionen und kann leicht konfiguriert werden.

```
const cassandra = require("cassandra-driver");
const clientOptions = {
    contactPoints: ["host1", "host2"],
    keyspace: "test"
};

const client = new cassandra.Client(clientOptions);

const query = "SELECT hello FROM world WHERE name = ?";
client.execute(query, ["John"], (err, results) => {
    if (err) {
      return console.error(err);
    }

    console.log(results.rows);
});
```

Cassandra-Integration online lesen: https://riptutorial.com/de/node-js/topic/5949/cassandra-integration

Kapitel 20: CLI

Syntax

- node [Optionen] [V8-Optionen] [script.js | -e "Skript"] [Argumente]

Examples

Befehlszeilenoptionen

```
-v, --version
```

Hinzugefügt in: v0.1.3 Version des Knotens drucken.

```
-h, --help
```

Hinzugefügt in: v0.1.3 Befehlszeilenoptionen für Knoten drucken. Die Ausgabe dieser Option ist weniger detailliert als dieses Dokument.

```
-e, --eval "script"
```

Hinzugefügt in: v0.5.2 Bewerten Sie das folgende Argument als JavaScript. Die in der REPL vordefinierten Module können auch im Skript verwendet werden.

```
-p, --print "script"
```

Hinzugefügt in: v0.6.4 Identisch mit -e, gibt jedoch das Ergebnis aus.

```
-c, --check
```

Hinzugefügt in: v5.0.0 Syntax Überprüfen Sie das Skript, ohne es auszuführen.

```
-i, --interactive
```

Hinzugefügt in: v0.7.7 Öffnet die REPL, auch wenn stdin kein Terminal zu sein scheint.

```
-r, --require module
```

Hinzugefügt in: v1.6.0 Laden Sie das angegebene Modul beim Start vor.

Folgt den Modulauflösungsregeln von (). Das Modul kann entweder ein Pfad zu einer Datei oder ein Knotenmodulname sein.

```
--no-deprecation
```

Hinzugefügt in: v0.8.0 Warnungen für die Stille-Abschreibung.

```
--trace-deprecation
```

Hinzugefügt in: v0.8.0 Stapelspuren für Abwertungen drucken.

```
--throw-deprecation
```

Hinzugefügt in: v0.11.14 Wurffehler für Verfall.

```
--no-warnings
```

Hinzugefügt in: v6.0.0 Deaktivieren Sie alle Prozesswarnungen (einschließlich Abschreibungen).

```
--trace-warnings
```

Hinzugefügt in: v6.0.0 Druckstapelverfolgungen für Prozesswarnungen (einschließlich Abwertungen).

```
--trace-sync-io
```

Hinzugefügt in: v2.1.0 Gibt eine Stapelablaufverfolgung aus, wenn nach dem ersten Durchlauf der Ereignisschleife synchrone E / A erkannt werden.

```
--zero-fill-buffers
```

Hinzugefügt in: v6.0.0 Füllt automatisch alle neu zugewiesenen Buffer- und SlowBuffer-Instanzen mit Nullen.

```
--preserve-symlinks
```

Hinzugefügt in: v6.3.0 Weist den Modullader an, beim Auflösen und Zwischenspeichern von Modulen symbolische Links beizubehalten.

Wenn Node.js standardmäßig ein Modul aus einem Pfad lädt, der symbolisch mit einem anderen Speicherort auf der Festplatte verknüpft ist, dereferenziert Node.js die Verknüpfung und verwendet den tatsächlichen "realen Pfad" des Moduls auf der Festplatte als Bezeichner und als Root-Pfad zum Auffinden anderer Abhängigkeitsmodule. In den meisten Fällen ist dieses Standardverhalten akzeptabel. Wenn Sie symbolisch verknüpfte Peer-Abhängigkeiten verwenden, wie im folgenden Beispiel veranschaulicht, führt das Standardverhalten dazu, dass eine Ausnahme ausgelöst wird, wenn moduleA versucht, moduleB als Peer-Abhängigkeit zu fordern:

```
{appDir}
 ├── app
 │   ├── index.js
 │   └── node_modules
 │       ├── moduleA -> {appDir}/moduleA
 │       └── moduleB
```

```
│           ├── index.js
│           └── package.json
└── moduleA
    ├── index.js
    └── package.json
```

Das Befehlszeilenflag "--preserve-symlinks" weist Node.js an, den symlink-Pfad für Module im Gegensatz zum realen Pfad zu verwenden, sodass symbolisch verknüpfte Peer-Abhängigkeiten gefunden werden können.

Beachten Sie jedoch, dass die Verwendung von --preserve-symlinks andere Nebenwirkungen haben kann. Insbesondere können symbolisch verknüpfte systemeigene Module möglicherweise nicht geladen werden, wenn diese von mehr als einem Ort im Abhängigkeitsbaum verknüpft werden (Node.js würde diese als zwei separate Module anzeigen und versuchen, das Modul mehrmals zu laden, wodurch eine Ausnahme ausgelöst wird).

```
--track-heap-objects
```

Hinzugefügt in: v2.4.0 Verfolgen Sie Heap-Objektzuordnungen für Heap-Snapshots.

```
--prof-process
```

Hinzugefügt in: v6.0.0 Process v8-Profilerausgabe, die mit der V8-Option --prof generiert wurde.

```
--v8-options
```

Hinzugefügt in: v0.1.3 Befehlszeilenoptionen für v8 drucken.

Hinweis: Mit den Optionen von Version 8 können Wörter durch Bindestriche (-) oder Unterstriche (_) getrennt werden.

Zum Beispiel entspricht --stack-trace-limit --stack_trace_limit.

```
--tls-cipher-list=list
```

Hinzugefügt in: v4.0.0 Geben Sie eine alternative Standardliste für die TLS-Verschlüsselung an. (Erfordert, dass Node.js mit Crypto-Unterstützung erstellt wird. (Standard))

```
--enable-fips
```

Hinzugefügt in: v6.0.0 Aktivieren Sie FIPS-konforme Verschlüsselung beim Start. (Erfordert, dass Node.js mit ./configure --openssl-fips erstellt wird.)

```
--force-fips
```

Hinzugefügt in: v6.0.0 FIPS-konforme Verschlüsselung beim Start erzwingen. (Kann nicht aus Skriptcode deaktiviert werden.) (Gleiche Anforderungen wie --enable-fips)

```
--icu-data-dir=file
```

Hinzugefügt in: v0.11.15 Geben Sie den Ladepfad der ICU-Daten an. (überschreibt NODE_ICU_DATA)

```
Environment Variables

NODE_DEBUG=module[,…]
```

Hinzugefügt in: v0.1.32 ',' - getrennte Liste der Kernmodule, die Debuginformationen drucken sollen.

```
NODE_PATH=path[:…]
```

Hinzugefügt in: v0.1.32 ':' - getrennte Liste von Verzeichnissen, die dem Modulsuchpfad vorangestellt sind.

Hinweis: Unter Windows ist dies stattdessen eine durch ";" getrennte Liste.

```
NODE_DISABLE_COLORS=1
```

Hinzugefügt in: v0.3.0 Bei Einstellung auf 1 werden in REPL keine Farben verwendet.

```
NODE_ICU_DATA=file
```

Hinzugefügt in: v0.11.15 Datenpfad für ICU-Daten (Intl-Objekt). Wird Linked-In-Daten erweitern, wenn sie mit Unterstützung für Small-ICE kompiliert werden.

```
NODE_REPL_HISTORY=file
```

Hinzugefügt in: v5.0.0 Pfad zu der Datei, in der der persistente REPL-Verlauf gespeichert wird. Der Standardpfad ist ~ / .node_repl_history, der von dieser Variablen überschrieben wird. Wenn Sie den Wert auf eine leere Zeichenfolge ("" oder "") setzen, wird der permanente REPL-Verlauf deaktiviert.

CLI online lesen: https://riptutorial.com/de/node-js/topic/6013/cli

Kapitel 21: Client-Server-Kommunikation

Examples

/ w Express, jQuery und Jade

```
//'client.jade'

//a button is placed down; similar in HTML
button(type='button', id='send_by_button') Modify data

    #modify Lorem ipsum Sender

    //loading jQuery; it can be done from an online source as well
    script(src='./js/jquery-2.2.0.min.js')

    //AJAX request using jQuery
    script
        $(function () {
            $('#send_by_button').click(function (e) {
                e.preventDefault();

                //test: the text within brackets should appear when clicking on said button
                //window.alert('You clicked on me.  - jQuery');

                //a variable and a JSON initialized in the code
                var predeclared = "Katamori";
                var data = {
                    Title: "Name_SenderTest",
                    Nick: predeclared,
                    FirstName: "Zoltan",
                    Surname: "Schmidt"
                };

                //an AJAX request with given parameters
                $.ajax({
                    type: 'POST',
                    data: JSON.stringify(data),
                    contentType: 'application/json',
                    url: 'http://localhost:7776/domaintest',

                    //on success, received data is used as 'data' function input
                    success: function (data) {
                        window.alert('Request sent; data received.');

                        var jsonstr = JSON.stringify(data);
                        var jsonobj = JSON.parse(jsonstr);

                        //if the 'nick' member of the JSON does not equal to the predeclared
string (as it was initialized), then the backend script was executed, meaning that
communication has been established
                        if(data.Nick != predeclared){
                            document.getElementById("modify").innerHTML = "JSON changed!\n" +
jsonstr;
                        };

                    }
```

```
                });
            });
            });

//'domaintest_route.js'

var express = require('express');
var router = express.Router();

//an Express router listening to GET requests - in this case, it's empty, meaning that nothing
is displayed when you reach 'localhost/domaintest'
router.get('/', function(req, res, next) {
});

//same for POST requests - notice, how the AJAX request above was defined as POST
router.post('/', function(req, res) {
    res.setHeader('Content-Type', 'application/json');

    //content generated here
    var some_json = {
        Title: "Test",
        Item: "Crate"
    };

    var result = JSON.stringify(some_json);

    //content got 'client.jade'
    var sent_data = req.body;
    sent_data.Nick = "ttony33";

    res.send(sent_data);

});


module.exports = router;
```

// basierend auf einem persönlich verwendeten gist:
https://gist.github.com/Katamori/5c9850f02e4baf6e9896

Client-Server-Kommunikation online lesen: https://riptutorial.com/de/node-js/topic/6222/client-server-kommunikation

Kapitel 22: Cluster-Modul

Syntax

- const cluster = required ("Cluster")
- cluster.fork ()
- cluster.isMaster
- cluster.isWorker
- cluster.schedulingPolicy
- cluster.setupMaster (Einstellungen)
- cluster.settings
- cluster.worker // in worker
- cluster.workers // in master

Bemerkungen

Beachten Sie, dass `cluster.fork()` einen `cluster.fork()` Prozess erzeugt, der mit der Ausführung des aktuellen Skripts von Anfang an beginnt, im Gegensatz zum Systemaufruf `fork()` in *C,* der den aktuellen Prozess `cluster.fork()` und die Anweisung nach dem Systemaufruf in parent und fortsetzt Kindprozess.

Die Node.js Dokumentation hat eine komplette Anleitung zum Cluster hier

Examples

Hallo Welt

Dies ist Ihre `cluster.js` :

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  // Fork workers.
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
} else {
  // Workers can share any TCP connection
  // In this case it is an HTTP server
  require('./server.js')();
}
```

Dies ist dein `server.js` :

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

function startServer() {
    const server = http.createServer((req, res) => {
      res.writeHead(200);
      res.end('Hello Http');
    });

    server.listen(3000);
}

if(!module.parent) {
    // Start server if file is run directly
    startServer();
} else {
    // Export server, if file is referenced via cluster
    module.exports = startServer;
}
```

In diesem Beispiel hosten wir einen einfachen Webserver, wir starten jedoch Arbeiter (untergeordnete Prozesse) mithilfe des integrierten **Cluster-** Moduls. Die Anzahl der Prozesswähler hängt von der Anzahl der verfügbaren CPU-Kerne ab. Dies ermöglicht einer Node.js-Anwendung, Multi-Core-CPUs zu nutzen, da eine einzelne Instanz von Node.js in einem einzigen Thread ausgeführt wird. Die Anwendung teilt jetzt den Port 8000 über alle Prozesse hinweg. Lasten werden standardmäßig automatisch unter Verwendung der Round-Robin-Methode zwischen Arbeitern verteilt.

Cluster-Beispiel

Eine einzelne Instanz von `Node.js` in einem einzelnen Thread ausgeführt. Um die Vorteile von Multicore-Systemen zu nutzen, kann die Anwendung in einem Cluster von Node.js-Prozessen gestartet werden, um die Last zu handhaben.

Mit dem `cluster` Modul können Sie auf einfache Weise untergeordnete Prozesse erstellen, die alle Serverports gemeinsam nutzen.

Im folgenden Beispiel erstellen Sie den Worker Child-Prozess im Hauptprozess, der die Last über mehrere Kerne hinweg übernimmt.

Beispiel

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length; //number of CPUS

if (cluster.isMaster) {
  // Fork workers.
  for (var i = 0; i < numCPUs; i++) {
    cluster.fork();    //creating child process
  }

  //on exit of cluster
```

```
  cluster.on('exit', (worker, code, signal) => {
      if (signal) {
        console.log(`worker was killed by signal: ${signal}`);
      } else if (code !== 0) {
        console.log(`worker exited with error code: ${code}`);
      } else {
        console.log('worker success!');
      }
  });
} else {
  // Workers can share any TCP connection
  // In this case it is an HTTP server
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('hello world\n');
  }).listen(3000);
}
```

Cluster-Modul online lesen: https://riptutorial.com/de/node-js/topic/2817/cluster-modul

Kapitel 23: CSV-Parser in Knoten js

Einführung

Das Einlesen von Daten aus einer CSV-Datei kann auf verschiedene Weise erfolgen. Eine Lösung besteht darin, die `csv` Datei in ein Array zu lesen. Von dort aus können Sie an dem Array arbeiten.

Examples

FS zum Einlesen einer CSV verwenden

fs ist das Dateisystem-API im Knoten. Wir können die Methode readFile für unsere fs-Variable verwenden, ihr eine `data.csv` Datei, ein Format und eine Funktion übergeben, die das `csv` zur weiteren Verarbeitung lesen und `data.csv` .

Dies setzt voraus, dass sich eine Datei namens `data.csv` im selben Ordner befindet.

```
'use strict'

const fs = require('fs');

fs.readFile('data.csv', 'utf8', function (err, data) {
  var dataArray = data.split(/\r?\n/);
  console.log(dataArray);
});
```

Sie können das Array jetzt wie jedes andere verwenden, um daran zu arbeiten.

CSV-Parser in Knoten js online lesen: https://riptutorial.com/de/node-js/topic/9162/csv-parser-in-knoten-js

Kapitel 24: Dateisystem-E / A

Bemerkungen

In Node.js werden ressourcenintensive Operationen wie E / A *asynchron ausgeführt* , haben jedoch ein *synchrones* Gegenstück (z. B. gibt es ein `fs.readFile` und sein Gegenstück ist `fs.readFileSync`). Da es sich bei Node nur um einen Thread handelt, sollten Sie bei der Verwendung *synchroner* Operationen vorsichtig sein, da sie den gesamten Prozess blockieren.

Wenn ein Prozess durch einen synchronen Vorgang blockiert wird, wird der gesamte Ausführungszyklus (einschließlich der Ereignisschleife) angehalten. Das bedeutet, dass anderer asynchroner Code, einschließlich Events und Event-Handler, nicht ausgeführt wird und Ihr Programm so lange wartet, bis der einzelne Blockierungsvorgang abgeschlossen ist.

Es gibt geeignete Verwendungen sowohl für synchrone als auch für asynchrone Operationen. Es muss jedoch darauf geachtet werden, dass sie ordnungsgemäß verwendet werden.

Examples

Schreiben in eine Datei mit writeFile oder writeFileSync

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

// Save the string "Hello world!" in a file called "hello.txt" in
// the directory "/tmp" using the default encoding (utf8).
// This operation will be completed in background and the callback
// will be called when it is either done or failed.
fs.writeFile('/tmp/hello.txt', 'Hello world!', function(err) {
  // If an error occurred, show it and return
  if(err) return console.error(err);
  // Successfully wrote to the file!
});

// Save binary data to a file called "binary.txt" in the current
// directory. Again, the operation will be completed in background.
var buffer = new Buffer([ 0x48, 0x65, 0x6c, 0x6c, 0x6f ]);
fs.writeFile('binary.txt', buffer, function(err) {
  // If an error occurred, show it and return
  if(err) return console.error(err);
  // Successfully wrote binary contents to the file!
});
```

`fs.writeFileSync` verhält sich ähnlich wie `fs.writeFile` , nimmt jedoch keinen Rückruf an, da es synchron ausgeführt wird, und blockiert daher den Haupt-Thread. Die meisten node.js-Entwickler bevorzugen die asynchronen Varianten, die praktisch keine Verzögerung bei der Programmausführung verursachen.

Anmerkung: Das Sperren des Hauptthreads ist in node.js unangebracht. Die Synchronfunktion sollte nur beim Debuggen verwendet werden oder wenn keine anderen Optionen verfügbar sind.

```
// Write a string to another file and set the file mode to 0755
try {
  fs.writeFileSync('sync.txt', 'anni', { mode: 0o755 });
} catch(err) {
  // An error occurred
  console.error(err);
}
```

Asynchron aus Dateien lesen

Verwenden Sie das Dateisystemmodul für alle Dateivorgänge:

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

Mit Kodierung

Lesen `hello.txt` in diesem Beispiel `hello.txt` aus dem Verzeichnis `/tmp` . Dieser Vorgang wird im Hintergrund abgeschlossen und der Rückruf erfolgt nach Abschluss oder Fehler:

```
fs.readFile('/tmp/hello.txt', { encoding: 'utf8' }, (err, content) => {
  // If an error occurred, output it and return
  if(err) return console.error(err);

  // No error occurred, content is a string
  console.log(content);
});
```

Ohne Kodierung

Lesen Sie die Binärdatei `binary.txt` asynchron im Hintergrund aus dem aktuellen Verzeichnis. Beachten Sie, dass wir die Option 'encoding' nicht festlegen - dies verhindert, dass Node.js den Inhalt in einen String decodiert:

```
fs.readFile('binary', (err, binaryContent) => {
  // If an error occurred, output it and return
  if(err) return console.error(err);

  // No error occurred, content is a Buffer, output it in
  // hexadecimal representation.
  console.log(content.toString('hex'));
});
```

Relative Pfade

Beachten Sie, dass Ihr Skript im Allgemeinen mit einem beliebigen aktuellen Arbeitsverzeichnis ausgeführt werden kann. Um eine Datei relativ zum aktuellen Skript zu adressieren, verwenden Sie `__dirname` oder `__filename` :

```
fs.readFile(path.resolve(__dirname, 'someFile'), (err, binaryContent) => {
  //Rest of Function
}
```

Verzeichnisinhalt mit readdir oder readdirSync auflisten

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

// Read the contents of the directory /usr/local/bin asynchronously.
// The callback will be invoked once the operation has either completed
// or failed.
fs.readdir('/usr/local/bin', (err, files) => {
  // On error, show it and return
  if(err) return console.error(err);

  // files is an array containing the names of all entries
  // in the directory, excluding '.' (the directory itself)
  // and '..' (the parent directory).

  // Display directory entries
  console.log(files.join(' '));
});
```

Als `readdirSync` gibt es eine synchrone Variante, die den Haupt-Thread blockiert und somit die gleichzeitige Ausführung von asynchronem Code verhindert. Die meisten Entwickler vermeiden synchrone E / A-Funktionen, um die Leistung zu verbessern.

```
let files;

try {
  files = fs.readdirSync('/var/tmp');
} catch(err) {
  // An error occurred
  console.error(err);
}
```

Verwendung eines Generators

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

// Iterate through all items obtained via
// 'yield' statements
// A callback is passed to the generator function because it is required by
// the 'readdir' method
function run(gen) {
  var iter = gen((err, data) => {
    if (err) { iter.throw(err); }

    return iter.next(data);
  });

  iter.next();
}

const dirPath = '/usr/local/bin';
```

```
// Execute the generator function
run(function* (resume) {
  // Emit the list of files in the directory from the generator
  var contents = yield fs.readdir(dirPath, resume);
  console.log(contents);
});
```

Synchron aus einer Datei lesen

Für alle Dateioperationen benötigen Sie das Dateisystemmodul:

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

Einen String lesen

`fs.readFileSync` verhält sich ähnlich wie `fs.readFile` , nimmt jedoch keinen Rückruf an, da es synchron ausgeführt wird, und blockiert daher den Haupt-Thread. Die meisten node.js-Entwickler bevorzugen die asynchronen Varianten, die praktisch keine Verzögerung bei der Programmausführung verursachen.

Wenn eine `encoding` angegeben wird, wird eine Zeichenfolge zurückgegeben, andernfalls wird ein `Buffer` zurückgegeben.

```
// Read a string from another file synchronously
let content;
try {
  content = fs.readFileSync('sync.txt', { encoding: 'utf8' });
} catch(err) {
  // An error occurred
  console.error(err);
}
```

Löschen einer Datei mit unlink oder unlinkSync

Datei asynchron löschen:

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

fs.unlink('/path/to/file.txt', function(err) {
  if (err) throw err;

  console.log('file deleted');
});
```

Sie können es auch synchron löschen *:

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

fs.unlinkSync('/path/to/file.txt');
```

```
console.log('file deleted');
```

* Vermeiden Sie synchrone Methoden, da sie den gesamten Prozess blockieren, bis die Ausführung abgeschlossen ist.

Eine Datei mit Streams in einen Puffer lesen

Während das Lesen von Inhalten aus einer Datei mit der Methode `fs.readFile()` bereits asynchron ist, `fs.readFile()` wir manchmal die Daten in einem Stream und nicht in einem einfachen Rückruf abrufen. Dies ermöglicht es uns, diese Daten an andere Standorte weiterzuleiten oder am Ende auf einmal zu verarbeiten.

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

// Store file data chunks in this array
let chunks = [];
// We can use this variable to store the final data
let fileBuffer;

// Read file into stream.Readable
let fileStream = fs.createReadStream('text.txt');

// An error occurred with the stream
fileStream.once('error', (err) => {
    // Be sure to handle this properly!
    console.error(err);
});

// File is done being read
fileStream.once('end', () => {
    // create the final data Buffer from data chunks;
    fileBuffer = Buffer.concat(chunks);

    // Of course, you can do anything else you need to here, like emit an event!
});

// Data is flushed from fileStream in chunks,
// this callback will be executed for each chunk
fileStream.on('data', (chunk) => {
    chunks.push(chunk); // push data chunk to array

    // We can perform actions on the partial data we have so far!
});
```

Überprüfen Sie die Berechtigungen einer Datei oder eines Verzeichnisses

`fs.access()` bestimmt, ob ein Pfad vorhanden ist und welche Berechtigungen ein Benutzer für die Datei oder das Verzeichnis in diesem Pfad hat. `fs.access` gibt kein Ergebnis zurück. Wenn kein Fehler zurückgegeben wird, ist der Pfad vorhanden und der Benutzer verfügt über die gewünschten Berechtigungen.

Die Berechtigungsmodi sind als Eigenschaft für das `fs` Objekt `fs.constants`

- `fs.constants.F_OK` - Hat Lese- / Schreib- / Ausführungsberechtigungen (Wenn kein Modus

angegeben ist, ist dies der Standard).

- `fs.constants.R_OK` - Hat Leseberechtigungen
- `fs.constants.W_OK` - Verfügt über Schreibberechtigungen
- `fs.constants.X_OK` - hat Ausführungsberechtigungen (funktioniert genauso wie `fs.constants.F_OK` unter Windows)

Asynchron

```
var fs = require('fs');
var path = '/path/to/check';

// checks execute permission
fs.access(path, fs.constants.X_OK, (err) => {
    if (err) {
        console.log("%s doesn't exist", path);
    } else {
        console.log('can execute %s', path);
    }
});
// Check if we have read/write permissions
// When specifying multiple permission modes
// each mode is separated by a pipe : `|`
fs.access(path, fs.constants.R_OK | fs.constants.W_OK, (err) => {
    if (err) {
        console.log("%s doesn't exist", path);
    } else {
        console.log('can read/write %s', path);
    }
});
```

Synchron

`fs.access` hat auch eine synchrone Version `fs.accessSync` . Wenn Sie `fs.accessSync` , müssen Sie es in einen try / catch-Block einschließen.

```
// Check write permission
try {
    fs.accessSync(path, fs.constants.W_OK);
    console.log('can write %s', path);
}
catch (err) {
    console.log("%s doesn't exist", path);
}
```

Vermeiden von Race-Bedingungen beim Erstellen oder Verwenden eines vorhandenen Verzeichnisses

Aufgrund der asynchronen Natur des Knotens müssen Sie zunächst ein Verzeichnis erstellen oder verwenden:

1. `fs.stat()` mit `fs.stat()` Existenz `fs.stat()`
2. abhängig von den Ergebnissen der Existenzprüfung erstellen oder verwenden,

kann zu einer Race-Bedingung führen, wenn der Ordner zwischen dem Zeitpunkt der Prüfung und dem Zeitpunkt der Erstellung erstellt wird. Die unten `fs.mkdir()` Methode `fs.mkdir()` und `fs.mkdirSync()` in Wrapper zum `fs.mkdirSync()` Fehlern, die die Ausnahme passieren lassen, wenn der Code `EEXIST` (bereits vorhanden). Wenn der Fehler etwas anderes ist, wie `EPERM` (Pemission denied), werfen Sie einen Fehler aus, oder übergeben Sie ihn, wie es die nativen Funktionen tun.

Asynchrone Version mit `fs.mkdir()`

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

function mkdir (dirPath, callback) {
  fs.mkdir(dirPath, (err) => {
    callback(err && err.code !== 'EEXIST' ? err : null);
  });
}

mkdir('./existingDir', (err) => {

  if (err)
    return console.error(err.code);

  // Do something with `./existingDir` here

});
```

Synchronversion mit `fs.mkdirSync()`

```
function mkdirSync (dirPath) {
  try {
    fs.mkdirSync(dirPath);
  } catch(e) {
    if ( e.code !== 'EEXIST' ) throw e;
  }
}

mkdirSync('./existing-dir');
// Do something with `./existing-dir` now
```

Überprüfen, ob eine Datei oder ein Verzeichnis vorhanden ist

Asynchron

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

fs.stat('path/to/file', function(err) {
    if (!err) {
        console.log('file or directory exists');
    }
    else if (err.code === 'ENOENT') {
        console.log('file or directory does not exist');
    }
```

```
});
```

Synchron

Hier müssen wir den Funktionsaufruf in einen `try/catch` Block einschließen, um Fehler zu behandeln.

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

try {
    fs.statSync('path/to/file');
    console.log('file or directory exists');
}
catch (err) {
  if (err.code === 'ENOENT') {
    console.log('file or directory does not exist');
  }
}
```

Klonen einer Datei mithilfe von Streams

Dieses Programm veranschaulicht, wie Sie eine Datei mithilfe von lesbaren und beschreibbaren Streams mithilfe von `createReadStream()` und den vom Dateisystemmodul bereitgestellten `createWriteStream()` Funktionen `createWriteStream()` .

```
//Require the file System module
var fs = require('fs');

/*
  Create readable stream to file in current directory (__dirname) named 'node.txt'
  Use utf8 encoding
  Read the data in 16-kilobyte chunks
*/
var readable = fs.createReadStream(__dirname + '/node.txt', { encoding: 'utf8', highWaterMark:
16 * 1024 });

// create writable stream
var writable = fs.createWriteStream(__dirname + '/nodeCopy.txt');

// Write each chunk of data to the writable stream
readable.on('data', function(chunk) {
    writable.write(chunk);
});
```

Kopieren von Dateien durch Piping-Streams

Dieses Programm kopiert eine Datei mit einem lesbaren und einem beschreibbaren Stream mit der Funktion `pipe()` , die von der Stream-Klasse bereitgestellt wird

```
// require the file system module
var fs = require('fs');

/*
```

```
    Create readable stream to file in current directory named 'node.txt'
    Use utf8 encoding
    Read the data in 16-kilobyte chunks
*/
var readable = fs.createReadStream(__dirname + '/node.txt', { encoding: 'utf8', highWaterMark:
16 * 1024 });

// create writable stream
var writable = fs.createWriteStream(__dirname + '/nodePipe.txt');

// use pipe to copy readable to writable
readable.pipe(writable);
```

Inhalt einer Textdatei ändern

Beispiel. Es wird das Wort ersetzen `email` - `name index.txt replace(/email/gim, 'name') email` an einen `name` in einer Textdatei `index.txt` mit einfachen RegExp `replace(/email/gim, 'name')` - `replace(/email/gim, 'name')`

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

fs.readFile('index.txt', 'utf-8', function(err, data) {
    if (err) throw err;

    var newValue = data.replace(/email/gim, 'name');

    fs.writeFile('index.txt', newValue, 'utf-8', function(err, data) {
        if (err) throw err;
        console.log('Done!');
    })
})
```

Ermitteln der Zeilenanzahl einer Textdatei

app.js

```
const readline = require('readline');
const fs = require('fs');

var file = 'path.to.file';
var linesCount = 0;
var rl = readline.createInterface({
    input: fs.createReadStream(file),
    output: process.stdout,
    terminal: false
});
rl.on('line', function (line) {
    linesCount++; // on each linebreak, add +1 to 'linesCount'
});
rl.on('close', function () {
    console.log(linesCount); // print the result when the 'close' event is called
});
```

Verwendungszweck:

Knoten App

Eine Datei Zeile für Zeile lesen

app.js

```
const readline = require('readline');
const fs = require('fs');

var file = 'path.to.file';
var rl = readline.createInterface({
    input: fs.createReadStream(file),
    output: process.stdout,
    terminal: false
});

rl.on('line', function (line) {
    console.log(line) // print the content of the line on each linebreak
});
```

Verwendungszweck:

Knoten App

Dateisystem-E / A online lesen: https://riptutorial.com/de/node-js/topic/489/dateisystem-e---a

Kapitel 25: Datei-Upload

Examples

Einzelnes Datei-Upload mit Multer

Erinnere dich an

- Ordner zum Hochladen erstellen (`uploads` im Beispiel).
- Installieren Sie Mulder `npm i -S multer`

`server.js` :

```
var express =   require("express");
var multer  =   require('multer');
var app         =   express();
var fs = require('fs');

app.get('/',function(req,res){
      res.sendFile(__dirname + "/index.html");
});

var storage =   multer.diskStorage({
  destination: function (req, file, callback) {
    fs.mkdir('./uploads', function(err) {
        if(err) {
            console.log(err.stack)
        } else {
            callback(null, './uploads');
        }
    })
  },
  filename: function (req, file, callback) {
    callback(null, file.fieldname + '-' + Date.now());
  }
});

app.post('/api/file',function(req,res){
    var upload = multer({ storage : storage}).single('userFile');
    upload(req,res,function(err) {
        if(err) {
            return res.end("Error uploading file.");
        }
        res.end("File is uploaded");
    });
});

app.listen(3000,function(){
    console.log("Working on port 3000");
});
```

`index.html` :

```
<form id        =  "uploadForm"
```

```
     enctype   =  "multipart/form-data"
     action    =  "/api/file"
     method    =  "post"
>
<input type="file" name="userFile" />
<input type="submit" value="Upload File" name="submit">
</form>
```

Hinweis:

Um eine Datei mit der Erweiterung hochzuladen, können Sie die integrierte Bibliothek des Pfads Node.js verwenden

Dafür benötigen Sie nur den `path` zur Datei `server.js` :

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

und ändern:

```
callback(null, file.fieldname + '-' + Date.now());
```

Hinzufügen einer Dateierweiterung auf folgende Weise:

```
callback(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname));
```

So filtern Sie den Upload nach Erweiterung:

In diesem Beispiel erfahren Sie, wie Sie Dateien hochladen, um nur bestimmte Erweiterungen zuzulassen.

Zum Beispiel nur Bildererweiterungen. `var upload = multer({ storage : storage}).single('userFile');` einfach zu `var upload = multer({ storage : storage}).single('userFile');` fileFilter-Bedingung

```
var upload = multer({
    storage: storage,
    fileFilter: function (req, file, callback) {
        var ext = path.extname(file.originalname);
        if(ext !== '.png' && ext !== '.jpg' && ext !== '.gif' && ext !== '.jpeg') {
            return callback(new Error('Only images are allowed'))
        }
        callback(null, true)
    }
}).single('userFile');
```

Jetzt können Sie nur Bilddateien mit den Erweiterungen `png` , `jpg` , `gif` oder `jpeg` hochladen

Verwendung eines beeindruckenden Moduls

Installieren Sie das Modul und lesen Sie die **Dokumente**

```
npm i formidable@latest
```

Beispiel eines Servers am 8080-Port

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

http.createServer(function(req, res) {
  if (req.url == '/upload' && req.method.toLowerCase() == 'post') {
    // parse a file upload
    var form = new formidable.IncomingForm();

    form.parse(req, function(err, fields, files) {
      if (err)
          do-smth; // process error

      // Copy file from temporary place
      // var fs = require('fs');
      // fs.rename(file.path, <targetPath>, function (err) { ... });

      // Send result on client
      res.writeHead(200, {'content-type': 'text/plain'});
      res.write('received upload:\n\n');
      res.end(util.inspect({fields: fields, files: files}));
    });

    return;
  }

  // show a file upload form
  res.writeHead(200, {'content-type': 'text/html'});
  res.end(
    '<form action="/upload" enctype="multipart/form-data" method="post">'+
    '<input type="text" name="title"><br>'+
    '<input type="file" name="upload" multiple="multiple"><br>'+
    '<input type="submit" value="Upload">'+
    '</form>'
  );
}).listen(8080);
```

Datei-Upload online lesen: https://riptutorial.com/de/node-js/topic/4080/datei-upload

Kapitel 26: Datenbank (MongoDB mit Mongoose)

Examples

Mungo-Verbindung

Stellen Sie sicher, dass Mongodb zuerst ausgeführt wird! `mongod --dbpath data/`

package.json

```
"dependencies": {
    "mongoose": "^4.5.5",
}
```

server.js (ECMA 6)

```
import mongoose from 'mongoose';

mongoose.connect('mongodb://localhost:27017/stackoverflow-example');
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'DB connection error!'));
```

server.js (ECMA 5.1)

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

mongoose.connect('mongodb://localhost:27017/stackoverflow-example');
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'DB connection error!'));
```

Modell

Definieren Sie Ihre Modelle:

app / models / user.js (ECMA 6)

```
import mongoose from 'mongoose';

const userSchema = new mongoose.Schema({
    name: String,
    password: String
});

const User = mongoose.model('User', userSchema);

export default User;
```

app / model / user.js (ECMA 5.1)

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

var userSchema = new mongoose.Schema({
    name: String,
    password: String
});

var User = mongoose.model('User', userSchema);

module.exports = User
```

Daten einfügen

ECMA 6:

```
const user = new User({
   name: 'Stack',
   password: 'Overflow',
   }) ;

user.save((err) => {
    if (err) throw err;

    console.log('User saved!');
});
```

ECMA5.1:

```
var user = new User({
   name: 'Stack',
   password: 'Overflow',
   }) ;

user.save(function (err) {
    if (err) throw err;

    console.log('User saved!');
});
```

Daten lesen

ECMA6:

```
User.findOne({
    name: 'stack'
}, (err, user) => {
    if (err) throw err;

    if (!user) {
        console.log('No user was found');
    } else {
        console.log('User was found');
    }
```

```
});
```

ECMA5.1:

```
User.findOne({
    name: 'stack'
}, function (err, user) {
    if (err) throw err;

    if (!user) {
        console.log('No user was found');
    } else {
        console.log('User was found');
    }
});
```

Datenbank (MongoDB mit Mongoose) online lesen: https://riptutorial.com/de/node-js/topic/6411/datenbank--mongodb-mit-mongoose-

Kapitel 27: Debuggen der Node.js-Anwendung

Examples

Core node.js Debugger und Knoteninspektor

Core Debugger verwenden

Node.js enthält ein integriertes nicht grafisches Debugging-Dienstprogramm. Starten Sie die Anwendung mit dem folgenden Befehl, um den Build im Debugger zu starten:

```
node debug filename.js
```

Betrachten Sie die folgende einfache Anwendung Node.js, die in `debugDemo.js`

```
'use strict';

function addTwoNumber(a, b){
// function returns the sum of the two numbers
debugger
  return a + b;
}

var result = addTwoNumber(5, 9);
console.log(result);
```

Der Keyword- `debugger` stoppt den Debugger an dieser Stelle im Code.

Befehlsreferenz

1. Treten

```
cont, c - Continue execution
next, n - Step next
step, s - Step in
out, o - Step out
```

2. Haltepunkte

```
setBreakpoint(), sb() - Set breakpoint on current line
setBreakpoint(line), sb(line) - Set breakpoint on specific line
```

Führen Sie den folgenden Befehl aus, um den obigen Code zu debuggen

```
node debug debugDemo.js
```

Sobald der obige Befehl ausgeführt wird, sehen Sie die folgende Ausgabe. Um die Debugger-Schnittstelle zu verlassen, geben Sie `process.exit()`

```
ankuranand:~/workspace/nodejs/nodejsDebugging $ node debug debugDemo.js
< Debugger listening on port 5858
debug> . ok
break in debugDemo.js:3
  1 // A Demo Code Showing the basic capabilities of the nodejs  debugging module
  2
> 3 'use strict';
  4
  5 function addTwoNumber(a, b){
debug> n
break in debugDemo.js:11
  9 }
 10
>11 let result = addTwoNumber(5, 9);
 12 console.log(result);
 13
debug> c
break in debugDemo.js:7
  5 function addTwoNumber(a, b){
  6 // function returns the sum of the two numbers
> 7 debugger
  8   return a + b;
  9 }
debug> c
< 14
debug> process.exit()
ankuranand:~/workspace/nodejs/nodejsDebugging $
```

Verwenden Sie den Befehl `watch(expression)` , um die Variable oder den Ausdruck hinzuzufügen, deren Wert Sie überwachen möchten, und `restart` , um die App und das Debugging `restart` zu starten.

Verwenden Sie `repl` , um Code interaktiv einzugeben. Der Repl-Modus hat denselben Kontext wie die Zeile, die Sie debuggen. Auf diese Weise können Sie den Inhalt von Variablen untersuchen und Codezeilen testen. Drücken Sie `Ctrl+C` , um die Debug-Replik zu verlassen.

Verwenden des eingebauten Knoteninspektors

v6.3.0

Sie können den in v8 integrierten Inspektor ausführen. Das Knoteninspektor- Plug-In wird nicht mehr benötigt.

Übergeben Sie einfach die Inspektorflagge und Sie erhalten eine URL zum Inspektor

```
node --inspect server.js
```

Verwenden des Knoteninspektors

Installieren Sie den Knoteninspektor:

```
npm install -g node-inspector
```

Führen Sie Ihre App mit dem Befehl node-debug aus:

```
node-debug filename.js
```

Danach klick in Chrome:

```
http://localhost:8080/debug?port=5858
```

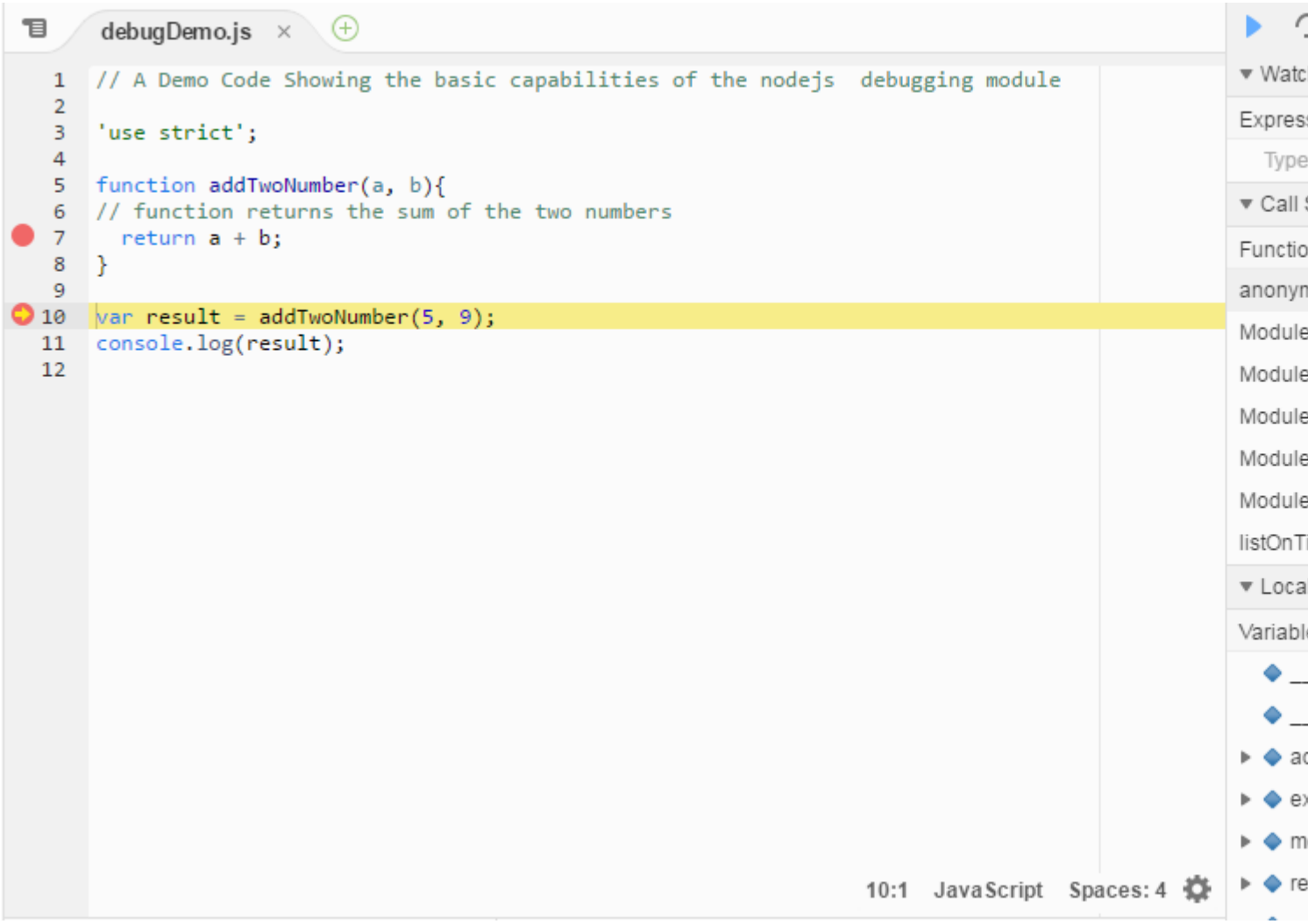
Manchmal ist Port 8080 auf Ihrem Computer nicht verfügbar. Möglicherweise erhalten Sie folgende Fehlermeldung:

> Der Server kann nicht bei 0.0.0.0:8080 gestartet werden. Fehler: EACCES abhören.

Starten Sie in diesem Fall den Knoteninspektor mit dem folgenden Befehl an einem anderen Port.

```
$node-inspector --web-port=6500
```

Sie werden so etwas sehen:



Debuggen der Node.js-Anwendung online lesen: https://riptutorial.com/de/node-js/topic/5900/debuggen-der-node-js-anwendung

Kapitel 28: Deinstallation von Node.js

Examples

Deinstallieren Sie Node.js unter Mac OSX vollständig

Geben Sie in Terminal auf Ihrem Mac-Betriebssystem die folgenden zwei Befehle ein:

```
lsbom -f -l -s -pf /var/db/receipts/org.nodejs.pkg.bom | while read f; do  sudo rm
/usr/local/${f}; done

sudo rm -rf /usr/local/lib/node /usr/local/lib/node_modules /var/db/receipts/org.nodejs.*
```

Deinstallieren Sie Node.js unter Windows

Um Node.js unter Windows zu deinstallieren, verwenden Sie Software wie folgt:

1. Öffnen `Add or Remove Programs` im Startmenü die Option Software.
2. Suchen Sie nach `Node.js`

Windows 10:

3. Klicken Sie auf Node.js.
4. Klicken Sie auf Deinstallieren.
5. Klicken Sie auf die neue Schaltfläche Deinstallieren.

Windows 7-8.1:

3. Klicken Sie unter Node.js auf die Schaltfläche Deinstallieren.

Deinstallation von Node.js online lesen: https://riptutorial.com/de/node-js/topic/2821/deinstallation-von-node-js

Kapitel 29: ECMAScript 2015 (ES6) mit Node.js

Examples

const / let Deklarationen

Im Gegensatz zu `var` sind `const` / `let` eher an den lexikalischen als an den Funktionsumfang gebunden.

```
{
  var x = 1 // will escape the scope
  let y = 2 // bound to lexical scope
  const z = 3 // bound to lexical scope, constant
}

console.log(x) // 1
console.log(y) // ReferenceError: y is not defined
console.log(z) // ReferenceError: z is not defined
```

Führen Sie RunKit aus

Pfeilfunktionen

Pfeilfunktionen binden sich automatisch an den lexikalischen Bereich dieses Codes.

```
performSomething(result => {
  this.someVariable = result
})
```

vs

```
performSomething(function(result) {
  this.someVariable = result
}.bind(this))
```

Pfeil Funktionsbeispiel

Betrachten wir dieses Beispiel, das die Quadrate der Zahlen 3, 5 und 7 ausgibt:

```
let nums = [3, 5, 7]
let squares = nums.map(function (n) {
  return n * n
})
console.log(squares)
```

Führen Sie RunKit aus

Die an `.map` Funktion kann auch als `.map` geschrieben werden, indem das `function` und stattdessen der Pfeil `=>` hinzugefügt wird:

```
let nums = [3, 5, 7]
let squares = nums.map((n) => {
  return n * n
})
console.log(squares)
```

Führen Sie RunKit aus

Dies kann jedoch noch präziser geschrieben werden. Wenn der Funktionskörper nur aus einer Anweisung besteht und diese den Rückgabewert berechnet, können die geschweiften Klammern des Umschließens des Funktionskörpers sowie das Schlüsselwort `return` .

```
let nums = [3, 5, 7]
let squares = nums.map(n => n * n)
console.log(squares)
```

Führen Sie RunKit aus

Zerstörung

```
    let [x,y, ...nums] = [0, 1, 2, 3, 4, 5, 6];
console.log(x, y, nums);

let {a, b, ...props} = {a:1, b:2, c:3, d:{e:4}}
console.log(a, b, props);

let dog = {name: 'fido', age: 3};
let {name:n, age} = dog;
console.log(n, age);
```

fließen

```
/* @flow */

function product(a: number, b: number){
  return a * b;
}

const b = 3;
let c = [1,2,3,,{}];
let d = 3;

import request from 'request';

request('http://dev.markitondemand.com/MODApis/Api/v2/Quote/json?symbol=AAPL', (err, res,
payload)=>{
  payload = JSON.parse(payload);
  let {LastPrice} = payload;
  console.log(LastPrice);
});
```

ES6-Klasse

```
class Mammel {
  constructor(legs){
    this.legs = legs;
  }
  eat(){
    console.log('eating...');
  }
  static count(){
    console.log('static count...');
  }
}

class Dog extends Mammel{
  constructor(name, legs){
    super(legs);
    this.name = name;
  }
  sleep(){
    super.eat();
    console.log('sleeping');
  }
}

let d = new Dog('fido', 4);
d.sleep();
d.eat();
console.log('d', d);
```

ECMAScript 2015 (ES6) mit Node.js online lesen: https://riptutorial.com/de/node-js/topic/6732/ecmascript-2015--es6--mit-node-js

Kapitel 30: Einfache REST-basierte CRUD-API

Examples

REST-API für CRUD in Express 3+

```
var express = require("express"),
    bodyParser = require("body-parser"),
    server = express();

//body parser for parsing request body
server.use(bodyParser.json());
server.use(bodyParser.urlencoded({ extended: true }));

//temperary store for `item` in memory
var itemStore = [];

//GET all items
server.get('/item', function (req, res) {
    res.json(itemStore);
});

//GET the item with specified id
server.get('/item/:id', function (req, res) {
    res.json(itemStore[req.params.id]);
});

//POST new item
server.post('/item', function (req, res) {
    itemStore.push(req.body);
    res.json(req.body);
});

//PUT edited item in-place of item with specified id
server.put('/item/:id', function (req, res) {
    itemStore[req.params.id] = req.body
    res.json(req.body);
});

//DELETE item with specified id
server.delete('/item/:id', function (req, res) {
    itemStore.splice(req.params.id, 1)
    res.json(req.body);
});

//START SERVER
server.listen(3000, function () {
    console.log("Server running");
})
```

Einfache REST-basierte CRUD-API online lesen: https://riptutorial.com/de/node-js/topic/5850/einfache-rest-basierte-crud-api

Kapitel 31: Erste Schritte mit der Knotenprofilerstellung

Einführung

Das Ziel dieses Beitrags ist es, mit der Profilierung von nodejs-Anwendung zu beginnen und zu verstehen, wie diese Ergebnisse zur Erfassung eines Fehlers oder eines Speicherverlusts sinnvoll sind. Eine Anwendung, bei der ein Knoten ausgeführt wird, ist nichts anderes als eine V8-Engine, die in vieler Hinsicht einer Website ähnelt, die in einem Browser ausgeführt wird. Grundsätzlich können alle Metriken erfasst werden, die mit einem Website-Prozess für eine Knotenanwendung zusammenhängen.

Mein bevorzugtes Werkzeug ist Chrome Devtools oder Chrome Inspector, der mit dem Node-Inspector gekoppelt ist.

Bemerkungen

Der Knoteninspektor kann keine Verbindung zum Knoten bebug herstellen. In diesem Fall können Sie den Debug-Haltepunkt in devtools nicht abrufen. Versuchen Sie, die Registerkarte devtools mehrmals zu aktualisieren, und warten Sie einige Sekunden, bis sich der Debug-Modus befindet.

Wenn nicht, starten Sie den Knoteninspektor von der Befehlszeile aus neu.

Examples

Profilieren einer einfachen Knotenanwendung

Schritt 1 : Installieren Sie das Node-Inspector-Paket global mit npm auf Ihrem Computer

```
$ npm install -g node-inspector
```

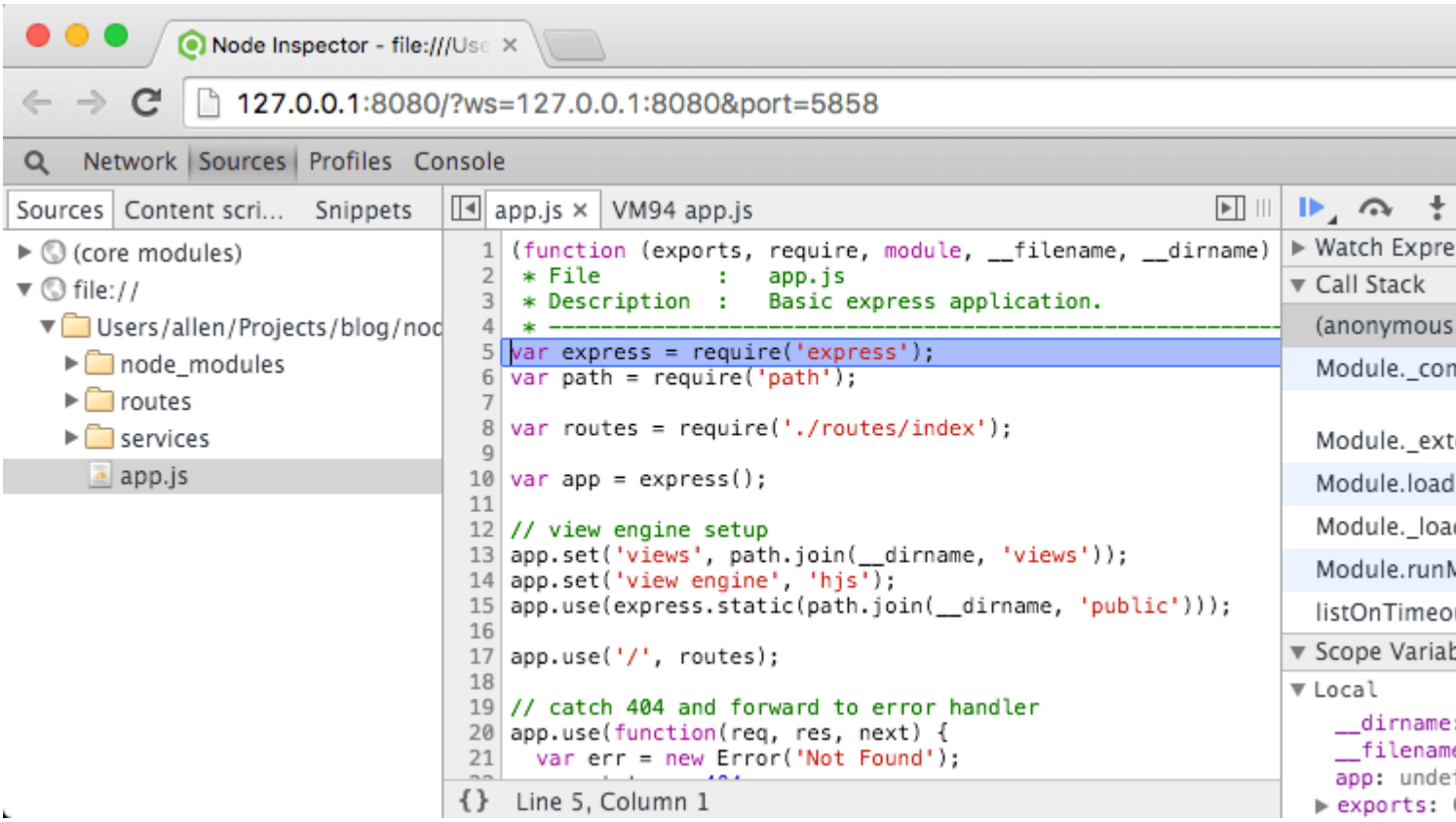
Schritt 2 : Starten Sie den Node-Inspector-Server

```
$ node-inspector
```

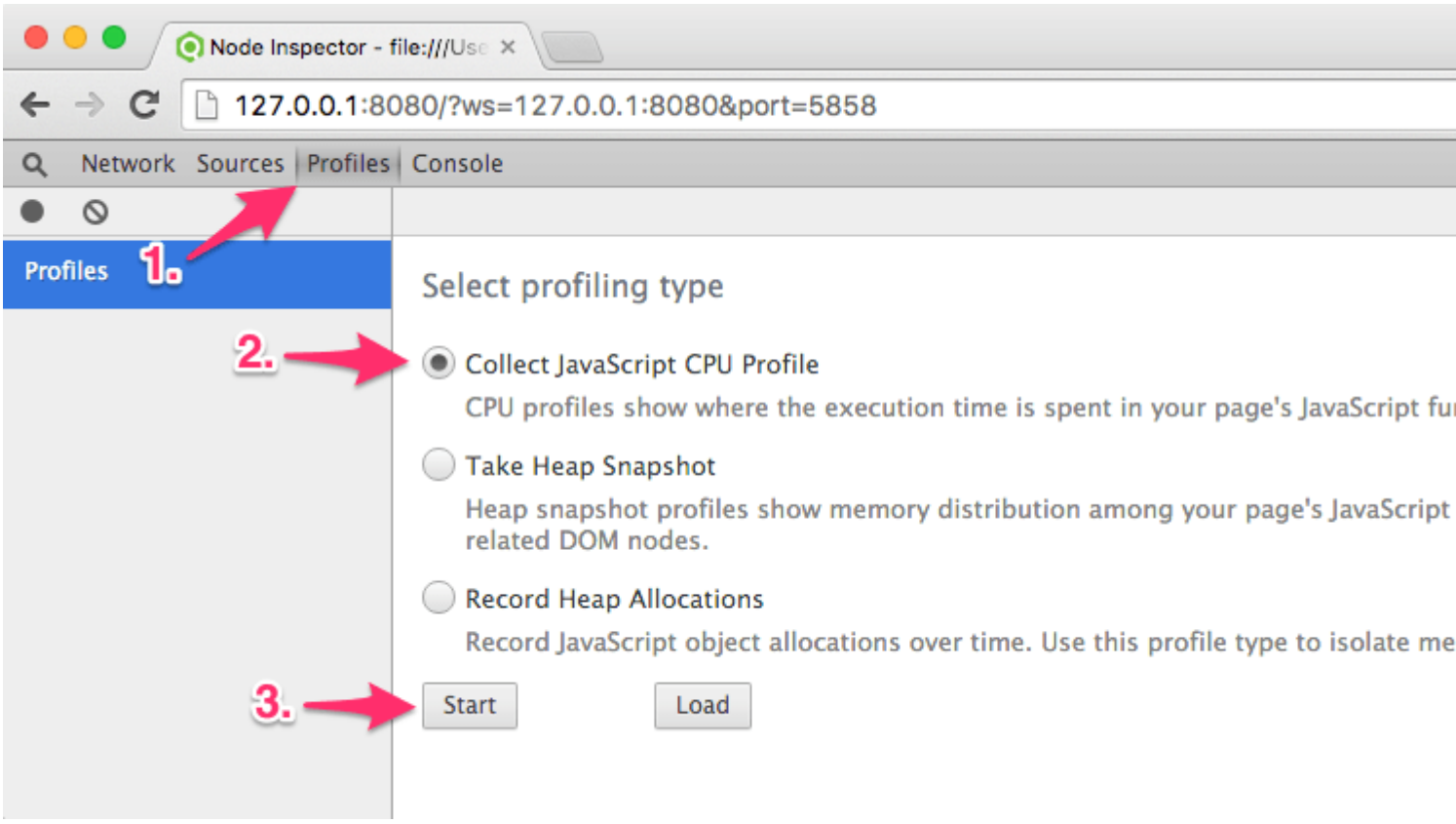
Schritt 3 : Beginnen Sie mit dem Debuggen Ihrer Knotenanwendung

```
$ node --debug-brk your/short/node/script.js
```

Schritt 4 : Öffnen Sie http://127.0.0.1:8080/?port=5858 im Chrome-Browser. Im linken Bereich sehen Sie eine Chrom-Dev-Tools-Schnittstelle mit dem Quellcode Ihrer Anwendung. Da wir beim Debuggen der Anwendung die Option debug break verwendet haben, wird die Ausführung des Codes in der ersten Codezeile angehalten.



Schritt 5 : Dies ist der einfache Teil, in dem Sie zur Registerkarte "Profiling" wechseln und mit der Profilerstellung der Anwendung beginnen. Wenn Sie das Profil für eine bestimmte Methode oder einen bestimmten Ablauf abrufen möchten, stellen Sie sicher, dass die Codeausführung kurz vor der Ausführung des Codeteils umbrochen wird.



Schritt 6 : Nachdem Sie Ihr CPU-Profil oder Heap-Dump / Snapshot oder Ihre Heap-Zuweisung

aufgezeichnet haben, können Sie die Ergebnisse in demselben Fenster anzeigen oder auf dem lokalen Laufwerk speichern, um sie später zu analysieren oder mit anderen Profilen zu vergleichen.

In diesen Artikeln können Sie wissen, wie Sie die Profile lesen:

- CPU-Profile lesen
- Chrome CPU-Profiler und Heap-Profiler

Erste Schritte mit der Knotenprofilerstellung online lesen: https://riptutorial.com/de/node-js/topic/9347/erste-schritte-mit-der-knotenprofilerstellung

Kapitel 32: Erstellen einer Node.js-Bibliothek, die sowohl Versprechen als auch Fehler beim ersten Rückruf unterstützt

Einführung

Viele Leute arbeiten gern mit Versprechen und / oder Async / erwarten die Syntax, aber wenn Sie ein Modul schreiben, wäre es für einige Programmierer hilfreich, klassische Callback-Methoden zu unterstützen. Anstatt zwei Module oder zwei Funktionsgruppen zu erstellen oder den Programmierer dazu zu bringen, Ihr Modul zu versprechen, kann Ihr Modul beide Programmiermethoden unterstützen, indem es bluebirds asCallback () oder Qs nodeify () verwendet.

Examples

Beispielmodul und entsprechendes Programm mit Bluebird

math.js

```
'use strict';

const Promise = require('bluebird');

module.exports = {

  // example of a callback-only method
  callbackSum: function(a, b, callback) {
    if (typeof a !== 'number')
      return callback(new Error('"a" must be a number'));
    if (typeof b !== 'number')
      return callback(new Error('"b" must be a number'));

    return callback(null, a + b);
  },

  // example of a promise-only method
  promiseSum: function(a, b) {
    return new Promise(function(resolve, reject) {
      if (typeof a !== 'number')
        return reject(new Error('"a" must be a number'));
      if (typeof b !== 'number')
        return reject(new Error('"b" must be a number'));
      resolve(a + b);
    });
  },

  // a method that can be used as a promise or with callbacks
  sum: function(a, b, callback) {
    return new Promise(function(resolve, reject) {
```

```
      if (typeof a !== 'number')
        return reject(new Error('"a" must be a number'));
      if (typeof b !== 'number')
        return reject(new Error('"b" must be a number'));
      resolve(a + b);
    }).asCallback(callback);
  },

};
```

index.js

```
'use strict';

const math = require('./math');


// classic callbacks

math.callbackSum(1, 3, function(err, result) {
  if (err)
    console.log('Test 1: ' + err);
  else
    console.log('Test 1: the answer is ' + result);
});

math.callbackSum(1, 'd', function(err, result) {
  if (err)
    console.log('Test 2: ' + err);
  else
    console.log('Test 2: the answer is ' + result);
});


// promises

math.promiseSum(2, 5)
.then(function(result) {
  console.log('Test 3: the answer is ' + result);
})
.catch(function(err) {
  console.log('Test 3: ' + err);
});

math.promiseSum(1)
.then(function(result) {
  console.log('Test 4: the answer is ' + result);
})
.catch(function(err) {
  console.log('Test 4: ' + err);
});


// promise/callback method used like a promise

math.sum(8, 2)
.then(function(result) {
  console.log('Test 5: the answer is ' + result);
})
.catch(function(err) {
```

```
  console.log('Test 5: ' + err);
});


// promise/callback method used with callbacks

math.sum(7, 11, function(err, result) {
  if (err)
    console.log('Test 6: ' + err);
  else
    console.log('Test 6: the answer is ' + result);
});


// promise/callback method used like a promise with async/await syntax

(async () => {

  try {
    let x = await math.sum(6, 3);
    console.log('Test 7a: ' + x);

    let y = await math.sum(4, 's');
    console.log('Test 7b: ' + y);

  } catch(err) {
    console.log(err.message);
  }

})();
```

Erstellen einer Node.js-Bibliothek, die sowohl Versprechen als auch Fehler beim ersten Rückruf unterstützt online lesen: https://riptutorial.com/de/node-js/topic/9874/erstellen-einer-node-js-bibliothek--die-sowohl-versprechen-als-auch-fehler-beim-ersten-ruckruf-unterstutzt

Kapitel 33: Eventloop

Einführung

In diesem Beitrag werden wir diskutieren, wie das Konzept von Eventloop entstand und wie es für Hochleistungsserver und ereignisgesteuerte Anwendungen wie GUIs verwendet werden kann.

Examples

Wie sich das Konzept der Ereignisschleife entwickelt hat.

Eventloop im Pseudocode

Eine Ereignisschleife ist eine Schleife, die auf Ereignisse wartet und dann auf diese Ereignisse reagiert

```
while true:
    wait for something to happen
    react to whatever happened
```

Beispiel für einen Single-Thread-HTTP-Server ohne Ereignisschleife

```
    while true:
    socket = wait for the next TCP connection
    read the HTTP request headers from (socket)
    file_contents = fetch the requested file from disk
    write the HTTP response headers to (socket)
    write the (file_contents) to (socket)
    close(socket)
```

Hier ist eine einfache Form eines HTTP-Servers, der ein einzelner Thread ist, jedoch keine Ereignisschleife. Das Problem hier ist, dass es wartet, bis jede Anforderung abgeschlossen ist, bevor mit der Verarbeitung der nächsten begonnen wird. Wenn es eine Weile dauert, um die HTTP-Anforderungsheader zu lesen oder die Datei von der Festplatte abzurufen, sollten wir in der Lage sein, die nächste Anforderung zu verarbeiten, während wir warten, bis diese abgeschlossen ist.

Die gebräuchlichste Lösung besteht darin, das Programm multithreadig zu machen.

Beispiel für einen Multithread-HTTP-Server

ohne Ereignisschleife

```
function handle_connection(socket):
    read the HTTP request headers from (socket)
    file_contents = fetch the requested file from disk
    write the HTTP response headers to (socket)
    write the (file_contents) to (socket)
    close(socket)
while true:
    socket = wait for the next TCP connection
    spawn a new thread doing handle_connection(socket)
```

Jetzt haben wir unseren kleinen HTTP-Server mit mehreren Threads versehen. Auf diese Weise können wir sofort zur nächsten Anforderung übergehen, da die aktuelle Anforderung in einem Hintergrundthread ausgeführt wird. Viele Server, einschließlich Apache, verwenden diesen Ansatz.

Aber es ist nicht perfekt. Eine Einschränkung ist, dass Sie nur so viele Threads erzeugen können. Bei Workloads, bei denen Sie über eine große Anzahl von Verbindungen verfügen, aber jede Verbindung nur gelegentlich Aufmerksamkeit erfordert, ist das Multithread-Modell nicht besonders leistungsfähig. Die Lösung für diese Fälle ist die Verwendung einer Ereignisschleife:

Beispiel eines HTTP-Servers mit Ereignisschleife

```
while true:
    event = wait for the next event to happen
    if (event.type == NEW_TCP_CONNECTION):
        conn = new Connection
        conn.socket = event.socket
        start reading HTTP request headers from (conn.socket) with userdata = (conn)
    else if (event.type == FINISHED_READING_FROM_SOCKET):
        conn = event.userdata
        start fetching the requested file from disk with userdata = (conn)
    else if (event.type == FINISHED_READING_FROM_DISK):
        conn = event.userdata
        conn.file_contents = the data we fetched from disk
        conn.current_state = "writing headers"
        start writing the HTTP response headers to (conn.socket) with userdata = (conn)
    else if (event.type == FINISHED_WRITING_TO_SOCKET):
        conn = event.userdata
        if (conn.current_state == "writing headers"):
            conn.current_state = "writing file contents"
            start writing (conn.file_contents) to (conn.socket) with userdata = (conn)
        else if (conn.current_state == "writing file contents"):
            close(conn.socket)
```

Hoffentlich ist dieser Pseudocode verständlich. Hier ist was los ist: Wir warten darauf, dass etwas passiert. Immer wenn eine neue Verbindung erstellt wird oder eine bestehende Verbindung unsere Aufmerksamkeit erfordert, gehen wir darauf ein und warten dann wieder. Auf diese Weise

arbeiten wir gut, wenn es viele Verbindungen gibt und jede nur selten Aufmerksamkeit erfordert.

In einer echten Anwendung (nicht Pseudocode), die unter Linux ausgeführt wird, wird der Teil "Warten auf das nächste Ereignis" durch Aufrufen des Systemaufrufs poll () oder epoll () implementiert. Die Teile "Lesen / Schreiben von etwas in einen Socket" werden durch Aufruf der Systemaufrufe recv () oder send () im nicht blockierenden Modus implementiert.

Referenz:

[1]. "Wie funktioniert eine Ereignisschleife?" [Online]. Verfügbar: https://www.quora.com/How-does-an-event-loop-work

Eventloop online lesen: https://riptutorial.com/de/node-js/topic/8652/eventloop

Kapitel 34: Event-Sender

Bemerkungen

Wenn ein Ereignis "ausgelöst" wird (was gleichbedeutend mit "Veröffentlichen eines Ereignisses" oder "Ausgeben eines Ereignisses") ist, wird jeder Listener synchron (Quelle) zusammen mit den zugehörigen Begleitdaten aufgerufen, die an `emit()` wurden Egal wie viele Argumente Sie einreichen:

```
myDog.on('bark', (howLoud, howLong, howIntense) => {
  // handle the event
})
myDog.emit('bark', 'loudly', '5 seconds long', 'fiercely')
```

Die Hörer werden in der Reihenfolge ihrer Registrierung angerufen:

```
myDog.on('urinate', () => console.log('My first thought was "Oh-no"'))
myDog.on('urinate', () => console.log('My second thought was "Not my lawn :)"'))
myDog.emit('urinate')
// The console.logs will happen in the right order because they were registered in that order.
```

Wenn Sie jedoch erst einen Listener abfeuern `prependListener()` , bevor alle anderen Listener bereits hinzugefügt wurden, können Sie `prependListener()` wie `prependListener()` :

```
myDog.prependListener('urinate', () => console.log('This happens before my first and second
thoughts, even though it was registered after them'))
```

Wenn Sie ein Ereignis anhören möchten, aber nur `once` `prependOnceListener` möchten, können Sie `once` statt `on` oder `prependOnceListener` anstelle von `prependListener` . Nachdem das Ereignis ausgelöst wurde und der Listener aufgerufen wurde, wird der Listener automatisch entfernt und beim nächsten Auslösen des Ereignisses nicht mehr aufgerufen.

Wenn Sie alle Zuhörer entfernen und von vorne beginnen möchten, können Sie genau das tun:

```
myDog.removeAllListeners()
```

Examples

HTTP Analytics durch einen Ereignisgeber

Im HTTP- `server.js` (zB `server.js`):

```
const EventEmitter = require('events')
const serverEvents = new EventEmitter()

// Set up an HTTP server
const http = require('http')
```

```
const httpServer = http.createServer((request, response) => {
  // Handler the request...
  // Then emit an event about what happened
  serverEvents.emit('request', request.method, request.url)
});

// Expose the event emitter
module.exports = serverEvents
```

In Supervisor-Code (z. B. `supervisor.js`):

```
const server = require('./server.js')
// Since the server exported an event emitter, we can listen to it for changes:
server.on('request', (method, url) => {
  console.log(`Got a request: ${method} ${url}`)
})
```

Immer wenn der Server eine Anforderung erhält, gibt er ein Ereignis aus, das als `request` das der Supervisor wartet. Der Supervisor kann dann auf das Ereignis reagieren.

Grundlagen

Ereignis-Emitters sind in Node integriert und für pub-sub, ein Muster, bei dem ein *Publisher* Ereignisse auslöst, auf die *Abonnenten* zuhören und darauf reagieren können. Im Node-Jargon werden Verleger als *Event-Emitters bezeichnet* . Sie geben Ereignisse aus, während Abonnenten als *Listener bezeichnet* werden und auf die Ereignisse reagieren.

```
// Require events to start using them
const EventEmitter = require('events').EventEmitter;
// Dogs have events to publish, or emit
class Dog extends EventEmitter {};
class Food {};

let myDog = new Dog();

// When myDog is chewing, run the following function
myDog.on('chew', (item) => {
  if (item instanceof Food) {
    console.log('Good dog');
  } else {
    console.log(`Time to buy another ${item}`);
  }
});

myDog.emit('chew', 'shoe'); // Will result in console.log('Time to buy another shoe')
const bacon = new Food();
myDog.emit('chew', bacon); // Will result in console.log('Good dog')
```

Im obigen Beispiel ist der Hund der Herausgeber / EventEmitter, während die Funktion, die das Element überprüft, der Abonnent / Listener war. Sie können auch mehr Zuhörer machen:

```
myDog.on('bark', () => {
  console.log('WHO\'S AT THE DOOR?');
  // Panic
```

```
});
```

Es können auch mehrere Listener für ein einzelnes Ereignis vorhanden sein, und es können sogar Listener entfernt werden:

```
myDog.on('chew', takeADeepBreathe);
myDog.on('chew', calmDown);
// Undo the previous line with the next one:
myDog.removeListener('chew', calmDown);
```

Wenn Sie eine Veranstaltung nur einmal anhören möchten, können Sie Folgendes verwenden:

```
myDog.once('chew', pet);
```

Dadurch wird der Listener automatisch ohne Racebedingungen entfernt.

Rufen Sie die Namen der Ereignisse ab, die abonniert werden

Die Funktion **EventEmitter.eventNames ()** gibt ein Array mit den Namen der aktuell abonnierten Ereignisse zurück.

```
const EventEmitter = require("events");
class MyEmitter extends EventEmitter{}

var emitter = new MyEmitter();

emitter
.on("message", function(){ //listen for message event
    console.log("a message was emitted!");
})
.on("message", function(){ //listen for message event
    console.log("this is not the right message");
})
.on("data", function(){ //listen for data event
    console.log("a data just occured!!");
});

console.log(emitter.eventNames()); //=> ["message","data"]
emitter.removeAllListeners("data");//=> removeAllListeners to data event
console.log(emitter.eventNames()); //=> ["message"]
```

Führen Sie RunKit aus

Erhalten Sie die Anzahl der Hörer, die für ein bestimmtes Ereignis registriert sind

Die Funktion Emitter.listenerCount (eventName) gibt die Anzahl der Listener zurück, die aktuell auf das als Argument angegebene Ereignis warten

```
const EventEmitter = require("events");
class MyEmitter extends EventEmitter{}
var emitter = new MyEmitter();
```

```
emitter
.on("data", ()=>{ // add listener for data event
    console.log("data event emitter");
});

console.log(emitter.listenerCount("data"))    // => 1
console.log(emitter.listenerCount("message")) // => 0

emitter.on("message", function mListener(){ //add listener for message event
    console.log("message event emitted");
});
console.log(emitter.listenerCount("data"))    // => 1
console.log(emitter.listenerCount("message")) // => 1

emitter.once("data", (stuff)=>{ //add another listener for data event
    console.log(`Tell me my ${stuff}`);
})

console.log(emitter.listenerCount("data"))   // => 2
console.log(emitter.listenerCount("message"))// => 1
```

Event-Sender online lesen: https://riptutorial.com/de/node-js/topic/1623/event-sender

Kapitel 35: Garnpaket-Manager

Einführung

Yarn ist ein Paketmanager für Node.js, ähnlich wie bei npm. Obwohl es viele Gemeinsamkeiten gibt, gibt es einige wichtige Unterschiede zwischen Yarn und npm.

Examples

Garneinbau

In diesem Beispiel werden die verschiedenen Methoden zum Installieren von Yarn für Ihr Betriebssystem erläutert.

Mac OS

Homebrew

```
brew update
brew install yarn
```

MacPorts

```
sudo port install yarn
```

Hinzufügen von Garn zu Ihrem PFAD

Fügen Sie Ihrem bevorzugten Shell-Profil Folgendes hinzu (`.profile` , `.bashrc` , `.zshrc` usw.)

```
export PATH="$PATH:`yarn global bin`"
```

Windows

Installateur

Installieren Sie zunächst Node.js, falls es noch nicht installiert ist.

Laden Sie das Yarn-Installationsprogramm als `.msi` von der Yarn-Website herunter .

Schokoladig

```
choco install yarn
```

Linux

Debian / Ubuntu

Stellen Sie sicher, dass Node.js für Ihre Distribution installiert ist, oder führen Sie die folgenden Schritte aus

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Konfigurieren Sie das YarnPkg-Repository

```
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list
```

Garn installieren

```
sudo apt-get update && sudo apt-get install yarn
```

CentOS / Fedora / RHEL

Installieren Sie Node.js, falls noch nicht installiert

```
curl --silent --location https://rpm.nodesource.com/setup_6.x | bash -
```

Garn installieren

```
sudo wget https://dl.yarnpkg.com/rpm/yarn.repo -O /etc/yum.repos.d/yarn.repo
sudo yum install yarn
```

Bogen

Installieren Sie Garn über AUR.

Beispiel mit Hof:

```
yaourt -S yarn
```

Solus

```
sudo eopkg install yarn
```

Alle Distributionen

Fügen Sie Ihrem bevorzugten Shell-Profil Folgendes hinzu (`.profile` , `.bashrc` , `.zshrc` usw.)

```
export PATH="$PATH:`yarn global bin`"
```

Alternative Installationsmethode

Shell-Skript

```
curl -o- -L https://yarnpkg.com/install.sh | bash
```

oder geben Sie eine zu installierende Version an

```
curl -o- -L https://yarnpkg.com/install.sh | bash -s -- --version [version]
```

Tarball

```
cd /opt
wget https://yarnpkg.com/latest.tar.gz
tar zvxf latest.tar.gz
```

Npm

Wenn Sie npm bereits installiert haben, starten Sie einfach

```
npm install -g yarn
```

Nach der Installation

Überprüfen Sie die installierte Version von Yarn, indem Sie sie ausführen

```
yarn --version
```

Ein Basispaket erstellen

Der Befehl `package.json` `yarn init` führt Sie durch die Erstellung einer `package.json` Datei, um einige Informationen zu Ihrem Paket zu konfigurieren. Dies ist ähnlich zu dem Befehl `npm init` in npm.

Erstellen Sie ein neues Verzeichnis, in dem sich Ihr Paket befindet, und navigieren Sie zu einem neuen Verzeichnis. Führen Sie anschließend den `yarn init`

```
mkdir my-package && cd my-package
yarn init
```

Beantworten Sie die folgenden Fragen in der CLI

```
question name (my-package): my-package
question version (1.0.0):
question description: A test package
question entry point (index.js):
question repository url:
question author: StackOverflow Documentation
question license (MIT):
success Saved package.json
  Done in 27.31s.
```

Dadurch wird eine `package.json` Datei ähnlich der folgenden erzeugt

```
{
  "name": "my-package",
  "version": "1.0.0",
  "description": "A test package",
  "main": "index.js",
  "author": "StackOverflow Documentation",
  "license": "MIT"
}
```

Jetzt können wir versuchen, eine Abhängigkeit hinzuzufügen. Die Grundsyntax dafür ist `yarn add [package-name]`

Führen Sie die folgenden Schritte aus, um ExpressJS zu installieren

```
yarn add express
```

Dadurch wird Ihrem `package.json` ein Abschnitt für `dependencies` `package.json` und ExpressJS hinzugefügt

```
"dependencies": {
    "express": "^4.15.2"
}
```

Installieren Sie das Paket mit dem Garn

Yarn verwendet die gleiche Registrierung wie npm. Das bedeutet, dass jedes auf npm verfügbare Paket auf Yarn gleich ist.

Um ein Paket zu installieren, führen `yarn add package` .

Wenn Sie eine bestimmte Version des Pakets benötigen, können Sie `yarn add package@version` .

Wenn die zu installierende Version mit einem Tag versehen wurde, können Sie `yarn add package@tag` .

Garnpaket-Manager online lesen: https://riptutorial.com/de/node-js/topic/9441/garnpaket-manager

Kapitel 36: grunzen

Bemerkungen

Lesen Sie weiter:

Das Handbuch zum Installieren von Grunt enthält detaillierte Informationen zur Installation bestimmter, in der Entwicklung oder in der Entwicklung befindlicher Versionen von Grunt und Grunt-cli.

Das Handbuch zum Konfigurieren von Aufgaben enthält ausführliche Erläuterungen zum Konfigurieren von Aufgaben, Zielen, Optionen und Dateien in der Gruntfile sowie eine Erläuterung von Vorlagen, Globbing-Mustern und dem Importieren externer Daten.

Das Handbuch zum Erstellen von Aufgaben listet die Unterschiede zwischen den Arten von Grunt-Aufgaben auf und zeigt eine Reihe von Beispielaufgaben und Konfigurationen.

Examples

Einführung in GruntJs

Grunt ist ein JavaScript Task Runner, der zur Automatisierung sich wiederholender Aufgaben wie Minifizierung, Kompilierung, Komponententest, Flusen usw. verwendet wird.

Um zu beginnen, müssen Sie die Befehlszeilenschnittstelle (CLI) von Grunt global installieren.

```
npm install -g grunt-cli
```

Vorbereiten eines neuen Grunt-Projekts: Bei einem typischen Setup werden zwei Dateien zu Ihrem Projekt hinzugefügt: package.json und die Gruntfile.

package.json: Diese Datei wird von npm verwendet, um Metadaten für Projekte zu speichern, die als npm-Module veröffentlicht wurden. Sie werden grunt und die grunt-Plugins, die Ihr Projekt benötigt, als devDependencies in dieser Datei auflisten.

Gruntfile: Diese Datei hat den Namen Gruntfile.js und wird zum Konfigurieren oder Definieren von Aufgaben und zum Laden von Grunt-Plugins verwendet.

```
Example package.json:

{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
```

```
  }
}
```

Beispiel für eine Gruntdatei:

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

Gruntplugins installieren

Abhängigkeit hinzufügen

Um ein Gruntplugin verwenden zu können, müssen Sie es zunächst als Abhängigkeit zu Ihrem Projekt hinzufügen. Verwenden wir das jshint-Plugin als Beispiel.

```
npm install grunt-contrib-jshint --save-dev
```

Mit `--save-dev` Option `--save-dev` wird das Plugin in `package.json` . Auf diese Weise wird das Plugin nach einer `npm install` immer `npm install` .

Laden des Plugins

Sie können Ihr Plugin mit `loadNpmTasks` in die Gruntfile-Datei `loadNpmTasks` .

```
grunt.loadNpmTasks('grunt-contrib-jshint');
```

Aufgabe konfigurieren

Sie konfigurieren die Aufgabe in der Gruntdatei, indem `jshint` dem an `grunt.initConfig` Objekt eine Eigenschaft namens `jshint grunt.initConfig` .

```
grunt.initConfig({
```

```
  jshint: {
    all: ['Gruntfile.js', 'lib/**/*.js', 'test/**/*.js']
  }
});
```

Vergessen Sie nicht, dass Sie andere Eigenschaften für andere Plugins haben können, die Sie verwenden.

Aufgabe ausführen

Um die Aufgabe einfach mit dem Plugin auszuführen, können Sie die Befehlszeile verwenden.

```
grunt jshint
```

Oder Sie können `jshint` zu einer anderen Aufgabe hinzufügen.

```
grunt.registerTask('default', ['jshint']);
```

Die Standardaufgabe wird mit dem Befehl grunt im Terminal ohne Optionen ausgeführt.

grunzen online lesen: https://riptutorial.com/de/node-js/topic/6059/grunzen

Kapitel 37: Guter Codierstil

Bemerkungen

Ich würde einem Anfänger empfehlen, mit diesem Codierstil zu beginnen. Und wenn irgendjemand einen besseren Weg vorschlagen kann (ich habe mich für diese Technik entschieden und arbeitet effizient in einer App, die von mehr als 100.000 Benutzern verwendet wird), können Sie sich jederzeit für Vorschläge entscheiden. TIA.

Examples

Grundprogramm für die Anmeldung

In diesem Beispiel wird erläutert, dass der **node.js-** Code zur besseren Übersichtlichkeit in verschiedene **Module / Ordner** unterteilt wird. Wenn Sie diese Technik anwenden, können andere Entwickler den Code leichter verstehen, da er direkt auf die betreffende Datei zugreifen kann, anstatt den gesamten Code durchzugehen. Wenn Sie in einem Team arbeiten und zu einem späteren Zeitpunkt ein neuer Entwickler beitritt, wird es für ihn einfacher, den Code selbst zu finden.

index.js : - Diese Datei verwaltet die Serververbindung.

```
//Import Libraries
var express = require('express'),
    session = require('express-session'),
    mongoose = require('mongoose'),
    request = require('request');

//Import custom modules
var userRoutes = require('./app/routes/userRoutes');
var config = require('./app/config/config');

//Connect to Mongo DB
mongoose.connect(config.getDBString());

//Create a new Express application and Configure it
var app = express();

//Configure Routes
app.use(config.API_PATH, userRoutes());

//Start the server
app.listen(config.PORT);
console.log('Server started at - '+ config.URL+ ":" +config.PORT);
```

config.js : - Diese Datei verwaltet alle Konfigurationsparameter, die durchgehend gleich bleiben.

```
var config = {
VERSION: 1,
BUILD: 1,
```

```
URL: 'http://127.0.0.1',
API_PATH : '/api',
PORT : process.env.PORT || 8080,
DB : {
    //MongoDB configuration
    HOST : 'localhost',
    PORT : '27017',
    DATABASE : 'db'
},

/*
 * Get DB Connection String for connecting to MongoDB database
 */
getDBString : function(){
    return 'mongodb://'+ this.DB.HOST +':'+ this.DB.PORT +'/'+ this.DB.DATABASE;
},

/*
 * Get the http URL
 */
getHTTPUrl : function(){
    return 'http://' + this.URL + ":" + this.PORT;
}

module.exports = config;
```

user.js : - **Modelldatei,** in der das Schema definiert ist

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

//Schema for User
var UserSchema = new Schema({
    name: {
        type: String,
    //    required: true
    },
    email: {
        type: String
    },
    password: {
        type: String,
        //required: true
    },
    dob: {
        type: Date,
        //required: true
    },
    gender: {
        type: String, // Male/Female
    //    required: true
    }
});

//Define the model for User
var User;
if(mongoose.models.User)
    User = mongoose.model('User');
else
    User = mongoose.model('User', UserSchema);
```

```
//Export the User Model
module.exports = User;
```

userController : - Diese Datei enthält die Funktion für die Benutzeranmeldung

```
var User = require('../models/user');
var crypto = require('crypto');

//Controller for User
var UserController = {

    //Create a User
    create: function(req, res){
        var repassword = req.body.repassword;
        var password = req.body.password;
        var userEmail = req.body.email;

        //Check if the email address already exists
        User.find({"email": userEmail}, function(err, usr){
            if(usr.length > 0){
                //Email Exists

                res.json('Email already exists');
                return;
            }
            else
            {
                //New Email

                //Check for same passwords
                if(password != repassword){
                    res.json('Passwords does not match');
                    return;
                }

                //Generate Password hash based on sha1
                var shasum = crypto.createHash('sha1');
                shasum.update(req.body.password);
                var passwordHash = shasum.digest('hex');

                //Create User
                var user = new User();
                user.name = req.body.name;
                user.email = req.body.email;
                user.password = passwordHash;
                user.dob = Date.parse(req.body.dob) || "";
                user.gender = req.body.gender;

                //Validate the User
                user.validate(function(err){
                    if(err){
                        res.json(err);
                        return;
                    }else{
                        //Finally save the User
                        user.save(function(err){
                            if(err)
                            {
                                res.json(err);
```

```
                                return;
                            }

                            //Remove Password before sending User details
                            user.password = undefined;
                            res.json(user);
                            return;
                        });
                    }
                });
            }
        });
    }

}

module.exports = UserController;
```

userRoutes.js : - Dies ist die Route für den userController

```
var express = require('express');
var UserController = require('../controllers/userController');

//Routes for User
var UserRoutes = function(app)
{
    var router = express.Router();

router.route('/users')
    .post(UserController.create);

return router;

}

module.exports = UserRoutes;
```

Das obige Beispiel mag zu groß erscheinen, aber wenn ein Anfänger bei node.js mit einer kleinen Mischung aus Expresswissen versucht, dies durchzugehen, wird es leicht und wirklich hilfreich sein.

Guter Codierstil online lesen: https://riptutorial.com/de/node-js/topic/6489/guter-codierstil

Kapitel 38: Hacken

Examples

Neue Erweiterungen zu request () hinzufügen

Sie können neue Erweiterungen zu `require.extensions require()` hinzufügen, indem Sie `require.extensions` .

Für ein **XML-** Beispiel:

```
// Add .xml for require()
require.extensions['.xml'] = (module, filename) => {
    const fs = require('fs')
    const xml2js = require('xml2js')

    module.exports = (callback) => {
        // Read required file.
        fs.readFile(filename, 'utf8', (err, data) => {
            if (err) {
                callback(err)
                return
            }
            // Parse it.
            xml2js.parseString(data, (err, result) => {
                callback(null, result)
            })
        })
    }
}
```

Wenn der Inhalt von `hello.xml` wie folgt lautet:

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>
    <bar>baz</bar>
    <qux />
</foo>
```

Sie können es lesen und durch `require()` analysieren:

```
require('./hello')((err, xml) {
    if (err)
        throw err;
    console.log(err);
})
```

Es druckt `{ foo: { bar: [ 'baz' ], qux: [ '' ] } }` .

Hacken online lesen: https://riptutorial.com/de/node-js/topic/6645/hacken

Kapitel 39: Halten Sie eine Knotenanwendung ständig aktiv

Examples

Verwenden Sie PM2 als Prozessmanager

Mit PM2 können Sie Ihre nodejs-Skripts für immer ausführen. Falls Ihre Anwendung abstürzt, wird sie auch von PM2 neu gestartet.

Installieren Sie PM2 global, um Ihre nodejs-Instanzen zu verwalten

```
npm install pm2 -g
```

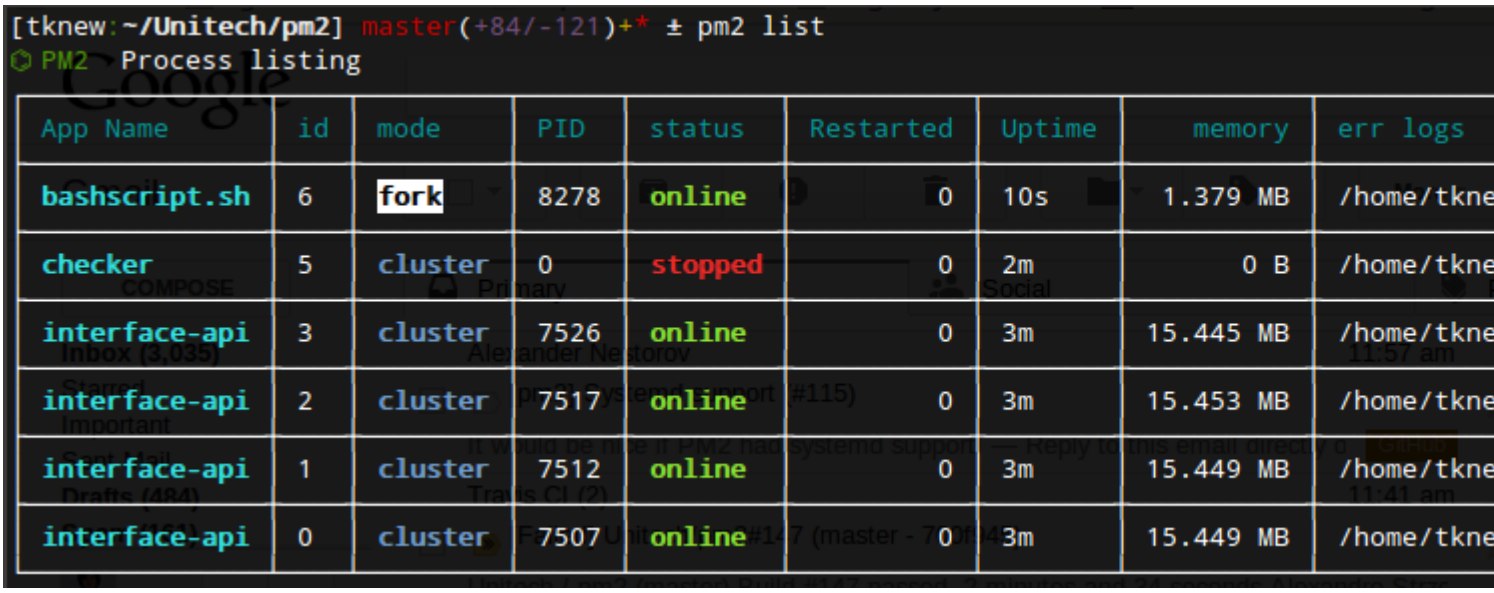
Navigieren Sie zu dem Verzeichnis, in dem sich Ihr nodejs-Skript befindet, und führen Sie den folgenden Befehl jedes Mal aus, wenn Sie eine von pm2 zu überwachende nodejs-Instanz starten möchten:

```
pm2 start server.js --name "app1"
```

Nützliche Befehle zur Überwachung des Prozesses

1. Listen Sie alle von pm2 verwalteten nodejs-Instanzen auf

   ```
   pm2 list
   ```



2. Stoppen Sie eine bestimmte nodejs-Instanz

   ```
   pm2 stop <instance named>
   ```

3. Löschen Sie eine bestimmte nodejs-Instanz

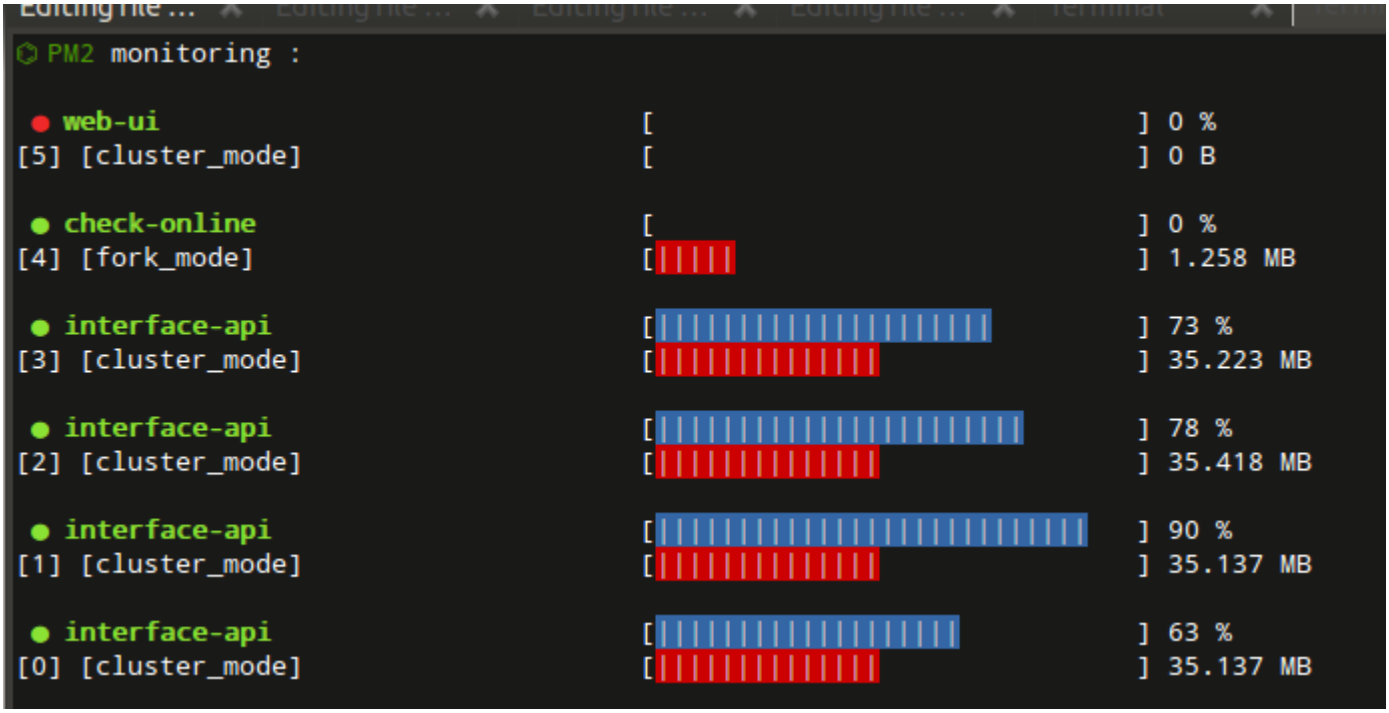
```
pm2 delete <instance name>
```

4. Starten Sie eine bestimmte nodejs-Instanz erneut

```
pm2 restart <instance name>
```

5. Überwachen aller nodejs-Instanzen

```
pm2 monit
```



6. Stoppen Sie pm2

```
pm2 kill
```

7. Im Gegensatz zum Neustart, der den Prozess abbricht und neu startet, wird beim Nachladen eine Nachladung von 0 Sekunden erreicht

```
pm2 reload <instance name>
```

8. Protokolle anzeigen

```
pm2 logs <instance_name>
```

Einen Forever-Daemon ausführen und stoppen

Um den Prozess zu starten:

```
$ forever start index.js
warn:    --minUptime not set. Defaulting to: 1000ms
warn:    --spinSleepTime not set. Your script will exit if it does not stay up for at least
1000ms
info:    Forever processing file: index.js
```

Liste der laufenden Forever-Instanzen:

```
$ forever list
info:    Forever processes running

|data: | index | uid | command          | script      |forever pid|id   | logfile
|uptime        |
|------|-------|-----|------------------|-------------|-----------|-----|--------------------
---|--------------|
|data: | [0]   |f4Kt |/usr/bin/nodejs   | src/index.js|2131       |
2146|/root/.forever/f4Kt.log | 0:0:0:11.485 |
```

Stoppen Sie den ersten Vorgang:

```
$ forever stop 0

$ forever stop 2146

$ forever stop --uid f4Kt

$ forever stop --pidFile 2131
```

Dauerlauf mit nohup

Eine Alternative zu ewig unter Linux ist nohup.

So starten Sie eine Nohup-Instanz

1. cd an den Ort von `app.js` oder `www app.js`
2. `nohup nodejs app.js &`

Um den Prozess zu beenden

1. Führen Sie `ps -ef|grep nodejs`
2. `kill -9 <the process number>`

Prozessmanagement mit Forever

Installation

```
npm install forever -g
cd /node/project/directory
```

Verwendungen

```
forever start app.js
```

Halten Sie eine Knotenanwendung ständig aktiv online lesen: https://riptutorial.com/de/node-js/topic/2820/halten-sie-eine-knotenanwendung-standig-aktiv

Kapitel 40: http

Examples

http-Server

Ein einfaches Beispiel für einen HTTP-Server.

Schreiben Sie folgenden Code in die Datei http_server.js:

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

var httpPort = 80;

http.createServer(handler).listen(httpPort, start_callback);

function handler(req, res) {

    var clientIP = req.connection.remoteAddress;
    var connectUsing = req.connection.encrypted ? 'SSL' : 'HTTP';
    console.log('Request received: '+ connectUsing + ' ' + req.method + ' ' + req.url);
    console.log('Client IP: ' + clientIP);

    res.writeHead(200, "OK", {'Content-Type': 'text/plain'});
    res.write("OK");
    res.end();
    return;
}

function start_callback(){
    console.log('Start HTTP on port ' + httpPort)
}
```

Führen Sie dann an Ihrem http_server.js-Verzeichnis den folgenden Befehl aus:

```
node http_server.js
```

Sie sollten dieses Ergebnis sehen:

```
> Start HTTP on port 80
```

Jetzt müssen Sie Ihren Server testen. Öffnen Sie Ihren Internetbrowser und navigieren Sie zu dieser URL:

```
http://127.0.0.1:80
```

Wenn Ihr Computer mit einem Linux-Server läuft, können Sie es folgendermaßen testen:

```
curl 127.0.0.1:80
```

Sie sollten folgendes Ergebnis sehen:

```
ok
```

In Ihrer Konsole, die die App ausführt, werden Sie folgende Ergebnisse sehen:

```
> Request received: HTTP GET /
> Client IP: ::ffff:127.0.0.1
```

http client

ein grundlegendes Beispiel für einen http-Client:

Schreiben Sie den folgenden Code in die Datei http_client.js:

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

var options = {
  hostname: '127.0.0.1',
  port: 80,
  path: '/',
  method: 'GET'
};

var req = http.request(options, function(res) {
    console.log('STATUS: ' + res.statusCode);
    console.log('HEADERS: ' + JSON.stringify(res.headers));
    res.setEncoding('utf8');
    res.on('data', function (chunk) {
        console.log('Response: ' + chunk);
    });
    res.on('end', function (chunk) {
        console.log('Response ENDED');
    });
});

req.on('error', function(e) {
    console.log('problem with request: ' + e.message);
});


req.end();
```

Führen Sie dann an Ihrem http_client.js-Verzeichnis den folgenden Befehl aus:

```
node http_client.js
```

Sie sollten dieses Ergebnis sehen:

```
> STATUS: 200
> HEADERS: {"content-type":"text/plain","date":"Thu, 21 Jul 2016 11:27:17
GMT","connection":"close","transfer-encoding":"chunked"}
> Response: OK
> Response ENDED
```

Hinweis: Dieses Beispiel hängt vom Beispiel eines http-Servers ab.

http online lesen: https://riptutorial.com/de/node-js/topic/2973/http

Kapitel 41: Knoten-JS-Lokalisierung

Einführung

Es ist sehr einfach, Lokalisierungsknoten zu verwalten

Examples

Verwenden des i18n-Moduls zum Verwalten der Lokalisierung in Knoten js app

Leichtes einfaches Übersetzungsmodul mit dynamischem Json-Speicher. Unterstützt einfache Vanilla node.js-Apps und sollte mit jedem Framework (wie express, restify und wahrscheinlich mehr) funktionieren, das eine app.use () -Methode bereitstellt, die res und req -Objekte übergeben. Verwendet die gängige __ ('...') - Syntax in App und Vorlagen. Speichert Sprachdateien in json-Dateien, die mit dem json-Format von webtranslateit kompatibel sind. Fügt neue Zeichenfolgen schnell hinzu, wenn sie zum ersten Mal in Ihrer App verwendet werden. Keine zusätzliche Analyse erforderlich.

drücken Sie + i18n-node + cookieParser aus und vermeiden Sie Parallelitätsprobleme

```
// usual requirements
var express = require('express'),
    i18n = require('i18n'),
    app = module.exports = express();

i18n.configure({
  // setup some locales - other locales default to en silently
  locales: ['en', 'ru', 'de'],

  // sets a custom cookie name to parse locale settings from
  cookie: 'yourcookiename',

  // where to store json files - defaults to './locales'
  directory: __dirname + '/locales'
});

app.configure(function () {
  // you will need to use cookieParser to expose cookies to req.cookies
  app.use(express.cookieParser());

  // i18n init parses req for language headers, cookies, etc.
  app.use(i18n.init);

});

// serving homepage
app.get('/', function (req, res) {
  res.send(res.__('Hello World'));
});

// starting server
```

```
if (!module.parent) {
  app.listen(3000);
}
```

Knoten-JS-Lokalisierung online lesen: https://riptutorial.com/de/node-js/topic/9594/knoten-js-lokalisierung

Kapitel 42: Knotenserver ohne Framework

Bemerkungen

Obwohl Node über viele Rahmenbedingungen verfügt, die Ihnen bei der Einrichtung Ihres Servers helfen, hauptsächlich:

Express : Das am häufigsten verwendete Framework

Gesamt : Das ALL-IN-ONE-UNITY-Framework, das alles enthält und nicht von anderen Frameworks oder Modulen abhängig ist.

Es gibt jedoch keine Einheitsgröße, daher muss der Entwickler möglicherweise seinen eigenen Server ohne weitere Abhängigkeiten aufbauen.

Wenn auf die App über einen externen Server zugegriffen wurde, könnte CORS ein Problem sein. Es wurde ein Code bereitgestellt, um dies zu vermeiden.

Examples

Framework-loser Knotenserver

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

http.createServer(function (request, response) {
console.log('request ', request.url);

var filePath = '.' + request.url;
if (filePath == './')
    filePath = './index.html';

var extname = String(path.extname(filePath)).toLowerCase();
var contentType = 'text/html';
var mimeTypes = {
    '.html': 'text/html',
    '.js': 'text/javascript',
    '.css': 'text/css',
    '.json': 'application/json',
    '.png': 'image/png',
    '.jpg': 'image/jpg',
    '.gif': 'image/gif',
    '.wav': 'audio/wav',
    '.mp4': 'video/mp4',
    '.woff': 'application/font-woff',
    '.ttf': 'applilcation/font-ttf',
    '.eot': 'application/vnd.ms-fontobject',
    '.otf': 'application/font-otf',
    '.svg': 'application/image/svg+xml'
};
```

```
contentType = mimeTypes[extname] || 'application/octect-stream';

fs.readFile(filePath, function(error, content) {
    if (error) {
        if(error.code == 'ENOENT'){
            fs.readFile('./404.html', function(error, content) {
                response.writeHead(200, { 'Content-Type': contentType });
                response.end(content, 'utf-8');
            });
        }
        else {
            response.writeHead(500);
            response.end('Sorry, check with the site admin for error: '+error.code+' ..\n');
            response.end();
        }
    }
    else {
        response.writeHead(200, { 'Content-Type': contentType });
        response.end(content, 'utf-8');
    }
 });

}).listen(8125);
console.log('Server running at http://127.0.0.1:8125/');
```

Überwindung von CORS-Problemen

```
// Website you wish to allow to connect to
response.setHeader('Access-Control-Allow-Origin', '*');

// Request methods you wish to allow
response.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');

// Request headers you wish to allow
response.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');

// Set to true if you need the website to include cookies in the requests sent
// to the API (e.g. in case you use sessions)
response.setHeader('Access-Control-Allow-Credentials', true);
```

Knotenserver ohne Framework online lesen: https://riptutorial.com/de/node-js/topic/5910/knotenserver-ohne-framework

Kapitel 43: Koa Framework v2

Examples

Hallo Weltbeispiel

```
const Koa = require('koa')

const app = new Koa()

app.use(async ctx => {
   ctx.body = 'Hello World'
})

app.listen(8080)
```

Fehlerbehandlung bei der Verwendung von Middleware

```
app.use(async (ctx, next) => {
  try {
    await next() // attempt to invoke the next middleware downstream
  } catch (err) {
    handleError(err, ctx) // define your own error handling function
  }
})
```

Koa Framework v2 online lesen: https://riptutorial.com/de/node-js/topic/6730/koa-framework-v2

Kapitel 44: Leistungsherausforderungen

Examples

Verarbeiten von lange laufenden Abfragen mit Node

Da es sich bei Node um einen Singlethread handelt, ist eine Problemumgehung erforderlich, wenn es zu langwierigen Berechnungen kommt.

Hinweis: Dies ist ein Beispiel für die Ausführung. Vergessen Sie nicht, jQuery zu installieren und die erforderlichen Module zu installieren.

Hauptlogik dieses Beispiels:

1. Client sendet Anforderung an den Server.
2. Der Server startet die Routine in einer separaten Knoteninstanz und sendet eine sofortige Antwort mit der zugehörigen Task-ID zurück.
3. Der Client sendet ständig Überprüfungen an den Server, um Statusaktualisierungen der angegebenen Task-ID zu überprüfen.

Projektstruktur:

```
project
|   package.json
|   index.html
|
├──js
|      main.js
|      jquery-1.12.0.min.js
|
└──srv
    |    app.js
    ├── models
    |      task.js
    └── tasks
           data-processor.js
```

app.js:

```
var express     = require('express');
var app         = express();
var http        = require('http').Server(app);
var mongoose    = require('mongoose');
var bodyParser  = require('body-parser');

var childProcess= require('child_process');

var Task        = require('./models/task');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

```
app.use(express.static(__dirname + '/../'));

app.get('/', function(request, response){
    response.render('index.html');
});

//route for the request itself
app.post('/long-running-request', function(request, response){
    //create new task item for status tracking
    var t = new Task({ status: 'Starting ...' });

    t.save(function(err, task){
        //create new instance of node for running separate task in another thread
        taskProcessor = childProcess.fork('./srv/tasks/data-processor.js');

        //process the messages comming from the task processor
        taskProcessor.on('message', function(msg){
            task.status = msg.status;
            task.save();
        }.bind(this));

        //remove previously openned node instance when we finished
        taskProcessor.on('close', function(msg){
            this.kill();
        });

        //send some params to our separate task
        var params = {
            message: 'Hello from main thread'
        };

        taskProcessor.send(params);
        response.status(200).json(task);
    });
});

//route to check is the request is finished the calculations
app.post('/is-ready', function(request, response){
    Task
        .findById(request.body.id)
        .exec(function(err, task){
            response.status(200).json(task);
        });
});

mongoose.connect('mongodb://localhost/test');
http.listen('1234');
```

task.js:

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

var taskSchema = mongoose.Schema({
    status: {
        type: String
    }
});

mongoose.model('Task', taskSchema);
```

```
module.exports  = mongoose.model('Task');
```

Datenprozessor.js:

```
process.on('message', function(msg){
    init = function(){
        processData(msg.message);
    }.bind(this)();

    function processData(message){
        //send status update to the main app
        process.send({ status: 'We have started processing your data.' });

        //long calculations ..
        setTimeout(function(){
            process.send({ status: 'Done!' });

            //notify node, that we are done with this task
            process.disconnect();
        }, 5000);
    }
});

process.on('uncaughtException',function(err){
    console.log("Error happened: " + err.message + "\n" + err.stack + ".\n");
    console.log("Gracefully finish the routine.");
});
```

index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <script src="./js/jquery-1.12.0.min.js"></script>
        <script src="./js/main.js"></script>
    </head>
    <body>
        <p>Example of processing long-running node requests.</p>
        <button id="go" type="button">Run</button>

        <br />

        <p>Log:</p>
        <textarea id="log" rows="20" cols="50"></textarea>
    </body>
</html>
```

main.js:

```
$(document).on('ready', function(){

    $('#go').on('click', function(e){
        //clear log
        $("#log").val('');

        $.post("/long-running-request", {some_params: 'params' })
            .done(function(task){
```

```
                $("#log").val( $("#log").val() + '\n' + task.status);

                //function for tracking the status of the task
                function updateStatus(){
                    $.post("/is-ready", {id: task._id })
                        .done(function(response){
                            $("#log").val( $("#log").val() + '\n' + response.status);

                            if(response.status != 'Done!'){
                                checkTaskTimeout = setTimeout(updateStatus, 500);
                            }
                        });
                }

                //start checking the task
                var checkTaskTimeout = setTimeout(updateStatus, 100);
            });
    });
});
```

package.json:

```
{
  "name": "nodeProcessor",
  "dependencies": {
    "body-parser": "^1.15.2",
    "express": "^4.14.0",
    "html": "0.0.10",
    "mongoose": "^4.5.5"
  }
}
```

Haftungsausschluss: Dieses Beispiel soll Ihnen eine grundlegende Idee geben. Um es in der Produktionsumgebung einsetzen zu können, sind Verbesserungen erforderlich.

Leistungsherausforderungen online lesen: https://riptutorial.com/de/node-js/topic/6325/leistungsherausforderungen

Kapitel 45: Liefern Sie HTML oder eine andere Datei

Syntax

- response.sendFile (Dateiname, Optionen, Funktion (Err) {});

Examples

Geben Sie HTML an den angegebenen Pfad aus

So erstellen Sie einen Express-Server und `index.html` standardmäßig `index.html` (leerer Pfad `/`) und `page1.html` für `/page1` Pfad.

Ordnerstruktur

```
project root
|    server.js
|____views
     |    index.html
     |    page1.html
```

server.js

```
var express = require('express');
var path = require('path');
var app = express();

// deliver index.html if no file is requested
app.get("/", function (request, response) {
  response.sendFile(path.join(__dirname, 'views/index.html'));
});

// deliver page1.html if page1 is requested
app.get('/page1', function(request, response) {
    response.sendFile(path.join(__dirname, 'views', 'page1.html', function(error) {
        if (error) {
            // do something in case of error
            console.log(err);
            response.end(JSON.stringify({error:"page not found"}));
        }
    });
});

app.listen(8080);
```

Beachten Sie, dass `sendFile()` nur eine statische Datei als Antwort streamt und keine Möglichkeit

bietet, sie zu ändern. Wenn Sie eine HTML-Datei bereitstellen und dynamische Daten einbeziehen möchten, müssen Sie eine *Vorlagen-Engine* wie Pug, Moustache oder EJS verwenden.

Liefern Sie HTML oder eine andere Datei online lesen: https://riptutorial.com/de/node-js/topic/6538/liefern-sie-html-oder-eine-andere-datei

Kapitel 46: Lodash

Einführung

Lodash ist eine praktische JavaScript-Dienstprogrammbibliothek.

Examples

Eine Sammlung filtern

Das folgende Codefragment zeigt die verschiedenen Möglichkeiten, wie Sie mit lodash nach Objekten filtern können.

```
let lodash = require('lodash');

var countries = [
    {"key": "DE", "name": "Deutschland", "active": false},
    {"key": "ZA", "name": "South Africa", "active": true}
];

var filteredByFunction = lodash.filter(countries, function (country) {
    return country.key === "DE";
});
// => [{"key": "DE", "name": "Deutschland"}];

var filteredByObjectProperties = lodash.filter(countries, { "key": "DE" });
// => [{"key": "DE", "name": "Deutschland"}];

var filteredByProperties = lodash.filter(countries, ["key", "ZA"]);
// => [{"key": "ZA", "name": "South Africa"}];

var filteredByProperty = lodash.filter(countries, "active");
// => [{"key": "ZA", "name": "South Africa"}];
```

Lodash online lesen: https://riptutorial.com/de/node-js/topic/9161/lodash

Kapitel 47: Loopback - REST-basierter Anschluss

Einführung

Restbasierte Konnektoren und Umgang mit ihnen. Wir alle wissen, dass Loopback REST-basierte Verbindungen nicht elegant macht

Examples

Hinzufügen eines webbasierten Connectors

```
// Dieses Beispiel erhält die Antwort von iTunes
{
  "sich ausruhen": {
    "name": "rest",
    "Verbinder": "Rest",
    "debug": wahr,
    "Optionen": {
      "useQuerystring": true,
      "Zeitüberschreitung": 10000
      "Header": {
        "akzeptiert": "application / json",
        "Inhaltstyp": "Anwendung / Json"
      }
    },
    "Vorgänge": [
      {
        "Vorlage": {
          "method": "GET",
          "url": "https://itunes.apple.com/search",
          "Abfrage": {
            "Begriff": "{Schlüsselwort}",
            "Land": "{Land = IN}",
            "media": "{itemType = music}",
            "limit": "{limit = 10}",
            "explizit": "falsch"
          }
        },
        "Funktionen": {
          "Suche": [
            "Stichwort",
            "Land",
            "Gegenstandsart",
            "Grenze"
          ]
        }
      },
      {
        "Vorlage": {
          "method": "GET",
          "url": "https://itunes.apple.com/lookup",
          "Abfrage": {
```

```
            "Ich tat}"
          }
        },
        "Funktionen": {
          "findById": [
            "Ich würde"
          ]
        }
      }
    ]
  }
}
```

Loopback - REST-basierter Anschluss online lesen: https://riptutorial.com/de/node-js/topic/9234/loopback---rest-basierter-anschluss

Kapitel 48: Metallschmied

Examples

Bauen Sie ein einfaches Blog auf

Vorausgesetzt, Sie haben node und npm installiert und verfügbar, erstellen Sie einen Projektordner mit einem gültigen `package.json` . Installieren Sie die notwendigen Abhängigkeiten:

```
npm install --save-dev metalsmith metalsmith-in-place handlebars
```

Erstellen `build.js` im Stammverzeichnis Ihres Projektordners eine Datei mit dem Namen `build.js` , die Folgendes enthält:

```
var metalsmith = require('metalsmith');
var handlebars = require('handlebars');
var inPlace = require('metalsmith-in-place');

Metalsmith(__dirname)
  .use(inPlace('handlebars'))
  .build(function(err) {
    if (err) throw err;
    console.log('Build finished!');
  });
```

Erstellen Sie im Stammverzeichnis Ihres Projektordners einen Ordner namens `src` . Erstellen Sie `index.html` in `src` , die Folgendes enthält:

```
---
title: My awesome blog
---
<h1>{{ title }}</h1>
```

Beim Ausführen des `node build.js` werden nun alle Dateien in `src` . Nachdem Sie diesen Befehl ausgeführt haben, befindet sich `index.html` in Ihrem Build-Ordner mit folgendem Inhalt:

```
<h1>My awesome blog</h1>
```

Metallschmied online lesen: https://riptutorial.com/de/node-js/topic/6111/metallschmied

Kapitel 49: Mit der Konsole interagieren

Syntax

- console.log ([Daten] [, ...])
- console.error ([data] [, ...])
- Konsole.Zeit (Label)
- console.timeEnd (label)

Examples

Protokollierung

Konsolenmodul

Ähnlich wie in der Browserumgebung von JavaScript bietet node.js ein **Konsolenmodul** , das einfache Protokollierungs- und Debugging-Möglichkeiten bietet.

Die wichtigsten Methoden des Konsolenmoduls sind `console.log` , `console.error` und `console.time` . Es gibt aber noch einige andere wie `console.info` .

console.log

Die Parameter werden mit einer neuen Zeile auf die Standardausgabe (`stdout`) gedruckt.

```
console.log('Hello World');
```

```
> console.log('Hello World')
Hello World
```

Konsole.Fehler

Die Parameter werden mit einer neuen Zeile zum Standardfehler (`stderr`) gedruckt.

```
console.error('Oh, sorry, there is an error.');
```

```
> console.error("Oh, sorry, error");
Oh, sorry, error
```

console.time, console.timeEnd

`console.time` startet einen Timer mit einem eindeutigen Label, mit dem die Dauer einer Operation berechnet werden kann. Wenn Sie `console.timeEnd` mit demselben Label aufrufen, wird der Timer `console.timeEnd` und die verstrichene Zeit in Millisekunden in `stdout` .

```
> console.time("label");
undefined
> console.timeEnd("label");
label: 9297.320ms
```

Prozessmodul

Es ist möglich, mit dem **Prozessmodul direkt** in die Standardausgabe der Konsole zu schreiben. Daher existiert die Methode `process.stdout.write` . Im Gegensatz zu `console.log` diese Methode vor der Ausgabe keine neue Zeile hinzu.

Im folgenden Beispiel wird die Methode zweimal aufgerufen, aber zwischen ihren Ausgaben wird keine neue Zeile hinzugefügt.

```
> process.stdout.write("123");process.stdout.write("456");
123456true
```

Formatierung

Terminalcodes (Steuercodes) können verwendet **werden** , um bestimmte Befehle auszugeben, beispielsweise das Wechseln der Farben oder das Positionieren des Cursors.

```
> console.log("\033[31mThis will be red");
This will be red
```

Allgemeines

Bewirken	Code
Zurücksetzen	`\033[0m`
Hicolor	`\033[1m`
Unterstreichen	`\033[4m`
Inverse	`\033[7m`

Schriftfarben

Bewirken	Code
Schwarz	`\033[30m`
rot	`\033[31m`
Grün	`\033[32m`

Bewirken	Code
Gelb	`\033[33m`
Blau	`\033[34m`
Magenta	`\033[35m`
Cyan	`\033[36m`
Weiß	`\033[37m`

Hintergrundfarben

Bewirken	Code
Schwarz	`\033[40m`
rot	`\033[41m`
Grün	`\033[42m`
Gelb	`\033[43m`
Blau	`\033[44m`
Magenta	`\033[45m`
Cyan	`\033[46m`
Weiß	`\033[47m`

Mit der Konsole interagieren online lesen: https://riptutorial.com/de/node-js/topic/5935/mit-der-konsole-interagieren

Kapitel 50: Mitteilungen

Einführung

Wenn Sie eine Web-App-Benachrichtigung erstellen möchten, sollten Sie Push.js oder das SoneSignal-Framework für Web / Mobile-App verwenden.

Push ist der schnellste Weg, Javascript-Benachrichtigungen bereitzustellen. Die Benachrichtigungs-API ist eine recht neue Erweiterung der offiziellen Spezifikation und ermöglicht es modernen Browsern wie Chrome, Safari, Firefox und IE 9+, Benachrichtigungen an den Desktop eines Benutzers zu senden.

Sie müssen Socket.io und ein Backend-Framework verwenden. Ich werde Express für dieses Beispiel verwenden.

Parameter

Modul / Rahmen	Beschreibung
node.js / express	Einfaches Backe-End-Framework für Node.js-Anwendung, sehr einfach zu bedienen und extrem leistungsstark
Socket.io	Socket.IO ermöglicht die bidirektionale ereignisbasierte Kommunikation in Echtzeit. Es funktioniert auf jeder Plattform, jedem Browser oder Gerät, wobei Zuverlässigkeit und Geschwindigkeit gleichermaßen im Vordergrund stehen.
Push.js	Das weltweit vielseitigste Desktop-Benachrichtigungs-Framework
OneSignal	Nur ein weiteres Formular für Push-Benachrichtigungen für Apple-Geräte
Firebase	Firebase ist die mobile Plattform von Google, mit der Sie schnell qualitativ hochwertige Apps entwickeln und Ihr Geschäft steigern können.

Examples

Webbenachrichtigung

Zuerst müssen Sie das Push.js- Modul installieren.

```
$ npm install push.js --save
```

Oder importieren Sie es über CDN in Ihre Front-End-App

```
<script src="./push.min.js"></script> <!-- CDN link -->
```

Wenn du damit fertig bist, solltest du gut sein. So sollte es aussehen, wenn Sie eine einfache Benachrichtigung machen möchten:

```
Push.create('Hello World!')
```

Ich gehe davon aus, dass Sie wissen, wie Sie Socket.io mit Ihrer App einrichten . Hier ist ein Codebeispiel meiner Backend-App mit Express:

```
var app = require('express')();
var server = require('http').Server(app);
var io = require('socket.io')(server);

server.listen(80);

app.get('/', function (req, res) {
  res.sendfile(__dirname + '/index.html');
});

io.on('connection', function (socket) {

  socket.emit('pushNotification', { success: true, msg: 'hello' });

});
```

Nachdem der Server vollständig eingerichtet ist, sollten Sie in der Lage sein, mit den Front-End-Elementen fortzufahren. Jetzt müssen wir nur noch Socket.io CDN importieren und diesen Code zu meiner *index.html*- Datei hinzufügen:

```
<script src="../socket.io.js"></script> <!-- CDN link -->
<script>
  var socket = io.connect('http://localhost');
  socket.on('pushNotification', function (data) {
    console.log(data);
    Push.create("Hello world!", {
        body: data.msg, //this should print "hello"
        icon: '/icon.png',
        timeout: 4000,
        onClick: function () {
            window.focus();
            this.close();
        }
    });
  });
</script>
```

Dort gehen Sie, jetzt sollten Sie in der Lage sein , Ihre Mitteilung angezeigt werden , kann dies auch auf jedem Android - Gerät funktioniert, und wenn u verwenden wollen Firebase Cloud Messaging, können Sie es mit diesem Modul verwenden, hier ist Link zu diesem Beispiel von Nick geschrieben (Konzept von Push.js)

Apfel

Beachten Sie, dass dies auf Apple-Geräten nicht funktioniert (ich habe sie nicht alle getestet), aber wenn Sie Push-Benachrichtigungen erstellen möchten, überprüfen Sie das OneSignal-Plugin.

Mitteilungen online lesen: https://riptutorial.com/de/node-js/topic/10892/mitteilungen

Kapitel 51: Modul in node.js exportieren und importieren

Examples

Verwenden eines einfachen Moduls in node.js

Was ist ein node.js-Modul (Link zum Artikel):

> Ein Modul kapselt zugehörigen Code in einer einzigen Codeeinheit. Beim Erstellen eines Moduls kann dies als Verschieben aller zugehörigen Funktionen in eine Datei interpretiert werden.

Nun sehen wir ein Beispiel. Stellen Sie sich vor, alle Dateien befinden sich in demselben Verzeichnis:

Datei: `printer.js`

```
"use strict";

exports.printHelloWorld = function (){
    console.log("Hello World!!!");
}
```

Eine andere Art der Verwendung von Modulen:

Datei `animals.js`

```
"use strict";

module.exports = {
    lion: function() {
        console.log("ROAARR!!!");
    }

};
```

Datei: `app.js`

Führen Sie diese Datei aus, indem Sie in Ihr Verzeichnis gehen und `node app.js` eingeben: `node app.js`

```
"use strict";

//require('./path/to/module.js') node which module to load
var printer = require('./printer');
var animals = require('./animals');

printer.printHelloWorld(); //prints "Hello World!!!"
```

```
animals.lion(); //prints "ROAARR!!!"
```

Importe in ES6 verwenden

Node.js ist gegen moderne Versionen von V8 gebaut. Durch die Aktualisierung der neuesten Versionen dieser Engine stellen wir sicher, dass neue Funktionen der JavaScript-Spezifikation ECMA-262 den Entwicklern von Node.js zeitnah zur Verfügung gestellt werden und dass die Leistung und Stabilität ständig verbessert werden.

Alle Funktionen von ECMAScript 2015 (ES6) sind in drei Gruppen für Versand-, Bereitstellungs- und Statusfeatures unterteilt:

Alle Versandfunktionen, die V8 für stabil hält, sind standardmäßig in Node.js aktiviert und erfordern KEINE Laufzeitkennzeichen. Inszenierte Features, bei denen es sich um beinahe abgeschlossene Features handelt, die vom V8-Team nicht als stabil angesehen werden, erfordern ein Laufzeitflag: --harmony. In Bearbeitung befindliche Features können einzeln durch das jeweilige Harmonieflag aktiviert werden, obwohl dies nur zu Testzwecken dringend empfohlen wird. Hinweis: Diese Flags werden von V8 verfügbar gemacht und können sich ohne Abmahnungsbenachrichtigung ändern.

Derzeit unterstützt ES6 Import-Anweisungen nativ. Hier klicken

Wenn wir also eine Datei namens `fun.js` ...

```
export default function say(what){
  console.log(what);
}

export function sayLoud(whoot) {
  say(whoot.toUpperCase());
}
```

… Und wenn es eine andere Datei namens `app.js` in der wir unsere zuvor definierten Funktionen verwenden möchten, gibt es drei Möglichkeiten, sie zu importieren.

Standard importieren

```
import say from './fun';
say('Hello Stack Overflow!!');  // Output: Hello Stack Overflow!!
```

Importiert die `say()` Funktion, da sie in der Quelldatei als Standardexport markiert ist (`export default …`).

Genannte Importe

```
import { sayLoud } from './fun';
sayLoud('JS modules are awesome.'); // Output: JS MODULES ARE AWESOME.
```

Benannte Importe ermöglichen es uns, genau die Teile eines Moduls zu importieren, die wir

tatsächlich benötigen. Wir tun dies, indem wir sie explizit benennen. In unserem Fall benennen Sie `sayLoud` innerhalb der Import-Anweisung in geschweiften Klammern.

Gebündelter Import

```
import * as i from './fun';
i.say('What?'); // Output: What?
i.sayLoud('Whoot!'); // Output: WHOOT!
```

Wenn wir alles haben wollen, ist dies der Weg. Durch die Verwendung der Syntax `* as i` mit der `import` Anweisung ein Objekt `i` , das alle Exporte unseres `fun` Moduls als entsprechend benannte Eigenschaften enthält.

Pfade

Beachten Sie, dass Sie Ihre Importpfade explizit als *relative Pfade* markieren müssen, auch wenn sich die zu importierende Datei in demselben Verzeichnis befindet wie die Datei, in die Sie mit `./` importieren. Importiert von unpräfixierten Pfaden wie

```
import express from 'express';
```

wird in den lokalen und globalen `node_modules` Ordnern `node_modules` und gibt einen Fehler aus, wenn keine passenden Module gefunden werden.

Exportieren mit ES6-Syntax

Dies ist das Äquivalent des anderen Beispiels, verwendet jedoch stattdessen ES6.

```
export function printHelloWorld() {
  console.log("Hello World!!!");
}
```

Modul in node.js exportieren und importieren online lesen: https://riptutorial.com/de/node-js/topic/1173/modul-in-node-js-exportieren-und-importieren

Kapitel 52: Module exportieren und konsumieren

Bemerkungen

Während in Node.js im Allgemeinen alles asynchron ausgeführt wird, gehört `require()` nicht dazu. Da Module in der Praxis nur einmal geladen werden müssen, ist dies eine Sperroperation und sollte ordnungsgemäß verwendet werden.

Module werden nach dem ersten Laden zwischengespeichert. Wenn Sie ein Modul in der Bearbeitung bearbeiten, müssen Sie dessen Eintrag im Modulcache löschen, um die neuen Änderungen verwenden zu können. Das heißt, selbst wenn ein Modul aus dem Modulcache gelöscht wird, wird das Modul selbst nicht mit Müll gefüllt, daher sollte in Produktionsumgebungen darauf geachtet werden.

Examples

Laden und Verwenden eines Moduls

Ein Modul kann "importiert" werden oder anderweitig "erforderlich" von der Funktion `require()` . Um beispielsweise das mit Node.js gelieferte `http` Modul zu laden, kann Folgendes verwendet werden:

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

Abgesehen von Modulen, die mit der Laufzeitumgebung ausgeliefert werden, können Sie auch von npm installierte Module, z. Wenn Sie express bereits über `npm install express` auf Ihrem System `npm install express` , können Sie einfach Folgendes schreiben:

```
const express = require('express');
```

Sie können auch Module hinzufügen, die Sie als Teil Ihrer Anwendung selbst geschrieben haben. In diesem Fall `lib.js` eine Datei namens `lib.js` im selben Verzeichnis wie die aktuelle Datei ein:

```
const mylib = require('./lib');
```

Beachten Sie, dass Sie die Erweiterung weglassen können, und `.js` wird davon ausgegangen, dass `.js` wird. Nachdem Sie ein Modul geladen haben, wird die Variable mit einem Objekt gefüllt, das die aus der erforderlichen Datei veröffentlichten Methoden und Eigenschaften enthält. Ein vollständiges Beispiel:

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

// The `http` module has the property `STATUS_CODES`
```

```
console.log(http.STATUS_CODES[404]); // outputs 'Not Found'

// Also contains `createServer()`
http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<html><body>Module Test</body></html>');
  res.end();
}).listen(80);
```

Ein hallo-world.js-Modul erstellen

Node stellt die Schnittstelle `module.exports` bereit, um Funktionen und Variablen für andere Dateien `module.exports` zu machen. Am einfachsten ist es, nur ein Objekt (Funktion oder Variable) zu exportieren, wie im ersten Beispiel gezeigt.

hallo-world.js

```
module.exports = function(subject) {
    console.log('Hello ' + subject);
};
```

Wenn der gesamte Export nicht ein einzelnes Objekt sein soll, können Funktionen und Variablen als Eigenschaften des `exports` werden. Die drei folgenden Beispiele veranschaulichen dies auf unterschiedliche Weise:

- hello-venus.js: Die Funktionsdefinition wird separat durchgeführt und dann als Eigenschaft von `module.exports`
- hello-jupiter.js: Die Funktionsdefinitionen werden direkt als Wert der Eigenschaften von `module.exports`
- hallo-mars.js: Die Funktionsdefinition wird direkt als `exports` deklariert. `module.exports` ist eine kurze Version von `module.exports`

hallo-venus.js

```
function hello(subject) {
    console.log('Venus says Hello ' + subject);
}

module.exports = {
    hello: hello
};
```

hallo-jupiter.js

```
module.exports = {
    hello: function(subject) {
      console.log('Jupiter says hello ' + subject);
    },

    bye: function(subject) {
      console.log('Jupiter says goodbye ' + subject);
    }
};
```

hallo-mars.js

```
exports.hello = function(subject) {
    console.log('Mars says Hello ' + subject);
};
```

Modul mit Verzeichnisnamen wird geladen

Wir haben ein Verzeichnis namens `hello` das die folgenden Dateien enthält:

index.js

```
// hello/index.js
module.exports = function(){
    console.log('Hej');
};
```

main.js

```
// hello/main.js
// We can include the other files we've defined by using the `require()` method
var hw = require('./hello-world.js'),
    hm = require('./hello-mars.js'),
    hv = require('./hello-venus.js'),
    hj = require('./hello-jupiter.js'),
    hu = require('./index.js');

// Because we assigned our function to the entire `module.exports` object, we
// can use it directly
hw('World!'); // outputs "Hello World!"

// In this case, we assigned our function to the `hello` property of exports, so we must
// use that here too
hm.hello('Solar System!'); // outputs "Mars says Hello Solar System!"

// The result of assigning module.exports at once is the same as in hello-world.js
hv.hello('Milky Way!'); // outputs "Venus says Hello Milky Way!"

hj.hello('Universe!'); //  outputs "Jupiter says hello Universe!"
hj.bye('Universe!'); // outputs "Jupiter says goodbye Universe!"

hu(); //output 'hej'
```

Den Cache des Moduls ungültig machen

In der Entwicklung stellen Sie möglicherweise fest, dass die Verwendung von `require()` auf demselben Modul immer dasselbe Modul zurückgibt, selbst wenn Sie Änderungen an dieser Datei vorgenommen haben. Dies liegt daran, dass Module beim ersten Laden zwischengespeichert werden und alle nachfolgenden Modullasten aus dem Cache geladen werden.

Um dieses Problem zu `delete` , müssen Sie den Eintrag im Cache `delete` . Wenn Sie beispielsweise ein Modul geladen haben:

```
var a = require('./a');
```

Sie könnten dann den Cache-Eintrag löschen:

```
var rpath = require.resolve('./a.js');
delete require.cache[rpath];
```

Und dann nochmal das Modul benötigen:

```
var a = require('./a');
```

Beachten Sie, dass dies in der Produktion nicht empfohlen wird, da beim `delete` nur die Referenz auf das geladene Modul und nicht die geladenen Daten selbst `delete` werden. Das Modul wird nicht aufgesammelt, daher kann eine falsche Verwendung dieser Funktion zu Speicherlecks führen.

Bauen Sie Ihre eigenen Module

Sie können auch auf ein Objekt verweisen, um Methoden öffentlich zu exportieren und kontinuierlich an dieses Objekt anzuhängen:

```
const auth = module.exports = {}
const config = require('../config')
const request = require('request')

auth.email = function (data, callback) {
  // Authenticate with an email address
}

auth.facebook = function (data, callback) {
  // Authenticate with a Facebook account
}

auth.twitter = function (data, callback) {
  // Authenticate with a Twitter account
}

auth.slack = function (data, callback) {
  // Authenticate with a Slack account
}

auth.stack_overflow = function (data, callback) {
  // Authenticate with a Stack Overflow account
}
```

Um eines dieser Module zu verwenden, benötigen Sie das Modul wie gewohnt:

```
const auth = require('./auth')

module.exports = function (req, res, next) {
  auth.facebook(req.body, function (err, user) {
    if (err) return next(err)
```

```
    req.user = user
    next()
  })
}
```

Jedes Modul wird nur einmal gespritzt

NodeJS führt das Modul nur dann aus, wenn Sie es zum ersten Mal benötigen. Alle weiteren erforderlichen Funktionen verwenden dasselbe Objekt erneut, sodass der Code nicht zu einem anderen Zeitpunkt im Modul ausgeführt wird. Außerdem speichert Node die Module beim ersten Laden mithilfe von requir. Dies verringert die Anzahl der Dateilesevorgänge und hilft, die Anwendung zu beschleunigen.

myModule.js

```
console.log(123) ;
exports.var1 = 4 ;
```

index.js

```
var a=require('./myModule') ; // Output 123
var b=require('./myModule') ; // No output
console.log(a.var1) ; // Output 4
console.log(b.var1) ; // Output 4
a.var2 = 5 ;
console.log(b.var2) ; // Output 5
```

Laden von Modulen aus Knoten_Modulen

Module können `require` d ohne relative Pfade zu benutzen , indem sie in einem speziellen Verzeichnis namens setzen `node_modules` .

Zum Beispiel, `require` ein Modul namens `foo` aus einer Datei `index.js` , können Sie die folgende Verzeichnisstruktur verwenden:

```
index.js
 \- node_modules
  \- foo
   |- foo.js
   \- package.json
```

Module sollten zusammen mit einer `package.json` Datei in einem `package.json` abgelegt werden. Das `main` der `package.json` Datei sollte für das Modul an den Einspeisepunkt Punkt - das ist die Datei , die importiert wird , wenn ein Anwender `require('your-module')` . `main` standardmäßig `index.js` wenn nicht angegeben. Alternativ können Sie auf Dateien relativ zu Ihrem Modul verweisen, indem Sie einfach den relativen Pfad an den `require` Aufruf anhängen: `require('your-module/path/to/file')` .

Die Module können auch werden `require` d von `node_modules` Verzeichnissen der Dateisystemhierarchie. Wenn wir die folgende Verzeichnisstruktur haben:

```
my-project
\- node_modules
 |- foo   // the foo module
  \- ...
 \- baz   // the baz module
  \- node_modules
   \- bar   // the bar module
```

wir in der Lage, `require` Sie das Modul `foo` aus einer beliebigen Datei innerhalb `bar` mit `require('foo')` .

Beachten Sie, dass der Knoten nur mit dem Modul übereinstimmt, das der Datei in der Dateisystemhierarchie am nächsten ist, beginnend mit (dem aktuellen Verzeichnis / den Knotenmodulen der Datei). Knoten ordnet Verzeichnisse auf diese Weise bis zum Dateisystemstammverzeichnis zu.

Sie können entweder neue Module von der npm-Registry oder anderen npm-Registries installieren oder eigene erstellen.

Ordner als Modul

Module können auf mehrere .js-Dateien in demselben Ordner aufgeteilt werden. Ein Beispiel in einem Ordner *my_module* :

function_one.js

```
module.exports = function() {
  return 1;
}
```

function_two.js

```
module.exports = function() {
  return 2;
}
```

index.js

```
exports.f_one = require('./function_one.js');
exports.f_two = require('./function_two.js');
```

Ein Modul wie dieses wird verwendet, indem auf den Ordnernamen Bezug genommen wird:

```
var split_module = require('./my_module');
```

Beachten Sie, dass der Knoten versucht, ein Modul aus dem Ordner *node_modules* zu laden, wenn Sie es erforderlich machen, indem Sie `./` oder einen Hinweis auf einen Pfad zu einem Ordner aus dem Argument für die Funktionsanforderung *weglassen* .

Alternativ können Sie im selben Ordner eine `package.json` Datei mit folgendem Inhalt erstellen:

```
{
    "name": "my_module",
    "main": "./your_main_entry_point.js"
}
```

Auf diese Weise müssen Sie die Hauptmoduldatei nicht "index" nennen.

Module exportieren und konsumieren online lesen: https://riptutorial.com/de/node-js/topic/547/module-exportieren-und-konsumieren

Kapitel 53: Mongodb-Integration

Syntax

- db. *collection* .insertOne (*Dokument* , *Optionen (w, wtimeout, j, serializeFuntions, forceServerObjectId, bypassDocumentValidation)* , *Rückruf*
- db. *collection* .insertMany (*[Dokumente]* , *Optionen (w, wtimeout, j, serializeFuntions, forceServerObjectId, bypassDocumentValidation)* , *Rückruf*)
- db. *collection* .find (*Abfrage*)
- db. *collection* .updateOne (*Filter* , *Update* , *Optionen (Upsert, W, Wtimeout, J, BypassDocumentValidation)* , *Callback*)
- db. *collection* .updateMany (*Filter* , *Update* , *Optionen (Upsert, W, Wtimeout, J)* , *Callback*)
- db. *collection* .deleteOne (*Filter* , *Optionen (Upsert, W, Wtimeout, J)* , *Rückruf*)
- db. *collection* .deleteMany (*Filter* , *Optionen (Upsert, W, Wtimeout, J)* , *Rückruf*)

Parameter

Parameter	Einzelheiten
dokumentieren	Ein Javascript-Objekt, das ein Dokument darstellt
Unterlagen	Eine Reihe von Dokumenten
Abfrage	Ein Objekt, das eine Suchabfrage definiert
Filter	Ein Objekt, das eine Suchabfrage definiert
Ruf zurück	Funktion, die aufgerufen wird, wenn die Operation abgeschlossen ist
Optionen	*(optional)* Optionale Einstellungen *(Standardeinstellung: null)*
w	*(optional)* Der Schreibvorgang
wtimeout	*(optional)* Das Timeout für den Schreibvorgang. *(Standardeinstellung: null)*
j	*(optional)* Geben Sie einen Journalschreibvorgang an *(Standardeinstellung: false).*
upsert	*(optional)* Aktualisierungsvorgang *(Standardeinstellung: false)*
multi	*(optional)* Aktualisieren Sie ein / alle Dokumente *(Standardeinstellung: false).*
serializeFunctions	*(optional)* Serialisierungsfunktionen für beliebige Objekte

Parameter	Einzelheiten
	(Standardeinstellung: false)
forceServerObjectId	*(optional)* Server zwingen, _id-Werte anstelle des Treibers zuzuweisen *(Standardeinstellung: false)*
bypassDocumentValidation	*(optional)* Zulassen, dass der Treiber die Schemavalidierung in MongoDB 3.2 oder höher umgeht *(Standard: false)*

Examples

Verbinden Sie sich mit MongoDB

Verbinden Sie sich mit MongoDB, drucken Sie "Connected!" und schließen Sie die Verbindung.

```
const MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function(err, db) { // MongoClient method 'connect'
    if (err) throw new Error(err);
    console.log("Connected!");
    db.close(); // Don't forget to close the connection when you are done
});
```

MongoClient-Methode `Connect()`

MongoClient.connect (*URL* , *Optionen* , *Rückruf*)

Streit	Art	Beschreibung
`url`	Schnur	Eine Zeichenfolge, die die IP-Adresse, den Port und die Datenbank des Servers angibt
`options`	Objekt	*(optional)* Optionale Einstellungen *(Standardeinstellung: null)*
`callback`	Funktion	Funktion, die aufgerufen werden soll, wenn der Verbindungsversuch ausgeführt wird

Die `callback` benötigt zwei Argumente

- `err` : Error - Wenn ein Fehler auftritt , das `err` Argument definiert werden
- `db` : object - Die MongoDB-Instanz

Ein Dokument einfügen

Legen Sie ein Dokument mit dem Namen 'myFirstDocument' ein und legen Sie **2** Eigenschaften,

`greetings` und `farewell`

```
const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function (err, db) {
  if (err) throw new Error(err);
  db.collection('myCollection').insertOne({ // Insert method 'insertOne'
    "myFirstDocument": {
      "greetings": "Hellu",
      "farewell": "Bye"
    }
  }, function (err, result) {
    if (err) throw new Error(err);
    console.log("Inserted a document into the myCollection collection!");
    db.close(); // Don't forget to close the connection when you are done
  });
});
```

Erhebungsmethode `insertOne()`

db.collection (*collection*) .insertOne (*Dokument* , *Optionen* , *Rückruf*)

Streit	Art	Beschreibung
`collection`	Schnur	Eine Zeichenfolge, die die Sammlung angibt
`document`	Objekt	Das Dokument, das in die Sammlung eingefügt werden soll
`options`	Objekt	*(optional)* Optionale Einstellungen *(Standardeinstellung: null)*
`callback`	Funktion	Funktion, die aufgerufen wird, wenn der Einfügevorgang abgeschlossen ist

Die `callback` benötigt zwei Argumente

- `err` : Error - Wenn ein Fehler auftritt , das `err` Argument definiert werden
- `result` : object - Ein Objekt, das Details zum Einfügevorgang enthält

Lesen Sie eine Sammlung

Holen Sie sich alle Dokumente in der Sammlung 'myCollection' und drucken Sie sie auf der Konsole aus.

```
const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function (err, db) {
  if (err) throw new Error(err);
  var cursor = db.collection('myCollection').find(); // Read method 'find'
```

```
  cursor.each(function (err, doc) {
    if (err) throw new Error(err);
    if (doc != null) {
      console.log(doc); // Print all documents
    } else {
      db.close(); // Don't forget to close the connection when you are done
    }
  });
});
```

Sammelmethode `find()`

db.collection (*collection*) .find ()

Streit	Art	Beschreibung
`collection`	Schnur	Eine Zeichenfolge, die die Sammlung angibt

Aktualisieren Sie ein Dokument

Suchen Sie ein Dokument mit der Eigenschaft `{ greetings: 'Hellu' }` und ändern Sie es in `{ greetings: 'Whut?' }`

```
const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function (err, db) {
    if (err) throw new Error(err);
    db.collection('myCollection').updateOne({ // Update method 'updateOne'
        greetings: "Hellu" },
        { $set: { greetings: "Whut?" }},
        function (err, result) {
            if (err) throw new Error(err);
            db.close(); // Don't forget to close the connection when you are done
        });
});
```

Erfassungsmethode `updateOne()`

db.collection (*collection*) .updateOne (*filtern* , *aktualisieren* , *Optionen* . *Rückruf*)

Parameter	Art	Beschreibung
`filter`	Objekt	Legt das Auswahlkriterium fest
`update`	Objekt	Gibt die anzuwendenden Änderungen an
`options`	Objekt	*(optional)* Optionale Einstellungen *(Standardeinstellung: null)*
`callback`	Funktion	Funktion, die aufgerufen wird, wenn die Operation abgeschlossen ist

Die `callback` benötigt zwei Argumente

- `err` : Error - Wenn ein Fehler auftritt , das err Argument definiert werden
- `db` : object - Die MongoDB-Instanz

Dokument löschen

Löschen Sie ein Dokument mit der Eigenschaft `{ greetings: 'Whut?' }`

```
const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function (err, db) {
    if (err) throw new Error(err);
    db.collection('myCollection').deleteOne(// Delete method 'deleteOne'
        { greetings: "Whut?" },
        function (err, result) {
            if (err) throw new Error(err);
            db.close(); // Don't forget to close the connection when you are done
    });
});
```

Erhebungsmethode `deleteOne()`

db.collection (*collection*) .deleteOne (*Filter* , *Optionen* , *Rückruf*)

Parameter	Art	Beschreibung
`filter`	Objekt	Ein Dokument, das das Auswahlkriterium angibt
`options`	Objekt	*(optional)* Optionale Einstellungen *(Standardeinstellung: null)*
`callback`	Funktion	Funktion, die aufgerufen wird, wenn die Operation abgeschlossen ist

Die `callback` benötigt zwei Argumente

- `err` : Error - Wenn ein Fehler auftritt , das err Argument definiert werden
- `db` : object - Die MongoDB-Instanz

Mehrere Dokumente löschen

Löschen Sie ALLE Dokumente, deren Abschiedseigenschaft auf "OK" gesetzt ist.

```
const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017/test';

MongoClient.connect(url, function (err, db) {
    if (err) throw new Error(err);
    db.collection('myCollection').deleteMany(// MongoDB delete method 'deleteMany'
```

```
        { farewell: "okay" }, // Delete ALL documents with the property 'farewell: okay'
        function (err, result) {
            if (err) throw new Error(err);
            db.close(); // Don't forget to close the connection when you are done
    });
});
```

Erhebungsmethode `deleteMany()`

db.collection (*collection*) .deleteMany (*Filter* , *Optionen* , *Rückruf*)

Parameter	Art	Beschreibung
`filter`	dokumentieren	Ein Dokument, das das Auswahlkriterium angibt
`options`	Objekt	*(optional)* Optionale Einstellungen *(Standardeinstellung: null)*
`callback`	Funktion	Funktion, die aufgerufen wird, wenn die Operation abgeschlossen ist

Die `callback` benötigt zwei Argumente

- `err` : Error - Wenn ein Fehler auftritt , das err Argument definiert werden
- `db` : object - Die MongoDB-Instanz

Einfach verbinden

```
MongoDB.connect('mongodb://localhost:27017/databaseName', function(error, database) {
   if(error) return console.log(error);
   const collection = database.collection('collectionName');
   collection.insert({key: 'value'}, function(error, result) {
      console.log(error, result);
   });
});
```

Einfach verbinden mit Versprechen

```
const MongoDB = require('mongodb');

MongoDB.connect('mongodb://localhost:27017/databaseName')
    .then(function(database) {
        const collection = database.collection('collectionName');
        return collection.insert({key: 'value'});
    })
    .then(function(result) {
        console.log(result);
    });
    ```
```

Mongodb-Integration online lesen: https://riptutorial.com/de/node-js/topic/5002/mongodb-
```

integration

Kapitel 54: MongoDB-Integration für Node.js / Express.js

Einführung

MongoDB ist dank der Hilfe des MEAN-Stacks eine der beliebtesten NoSQL-Datenbanken. Die Schnittstelle zu einer Mongo-Datenbank von einer Express-App aus ist schnell und einfach, sobald Sie die irgendwie verständliche Abfragesyntax verstanden haben. Wir werden Mongoose benutzen, um uns zu helfen.

Bemerkungen

Weitere Informationen finden Sie hier: http://mongoosejs.com/docs/guide.html

Examples

MongoDB installieren

```
npm install --save mongodb
npm install --save mongoose //A simple wrapper for ease of development
```

In Ihrer Server-Datei (normalerweise index.js oder server.js)

```
const express = require('express');
const mongodb = require('mongodb');
const mongoose = require('mongoose');
const mongoConnectString = 'http://localhost/database name';

mongoose.connect(mongoConnectString, (err) => {
  if (err) {
    console.log('Could not connect to the database');
  }
});
```

Erstellen eines Mungo-Modells

```
const Schema = mongoose.Schema;
const ObjectId = Schema.Types.ObjectId;

const Article = new Schema({
  title: {
    type: String,
    unique: true,
    required: [true, 'Article must have title']
  },
  author: {
    type: ObjectId,
```

```
    ref: 'User'
  }
});

module.exports = mongoose.model('Article, Article);
```

Lassen Sie uns das analysieren. MongoDB und Mongoose verwenden als Datenformat JSON (eigentlich BSON, das ist hier jedoch irrelevant). Oben habe ich ein paar Variablen gesetzt, um das Tippen zu reduzieren.

Ich erstelle ein `new Schema` und ordne es einer Konstanten zu. Es ist ein einfaches JSON, und jedes Attribut ist ein anderes Objekt mit Eigenschaften, die dazu beitragen, ein konsistenteres Schema durchzusetzen. Unique zwingt neue Instanzen, die in die Datenbank eingefügt werden, eindeutig zu sein. Dies ist ideal, um zu verhindern, dass ein Benutzer mehrere Konten für einen Dienst erstellt.

Erforderlich ist ein anderes, als Array deklariert. Das erste Element ist der boolesche Wert und das zweite Element die Fehlermeldung, falls der Wert, der eingefügt oder aktualisiert wird, nicht vorhanden ist.

ObjectIds werden für Beziehungen zwischen Modellen verwendet. Beispiele sind "Benutzer haben viele Kommentare". Andere Attribute können anstelle von ObjectId verwendet werden. Zeichenfolgen wie ein Benutzername sind ein Beispiel.

Wenn Sie das Modell zur Verwendung mit Ihren API-Routen exportieren, können Sie auf Ihr Schema zugreifen.

Abfragen Ihrer Mongo-Datenbank

Eine einfache GET-Anfrage. Nehmen wir an, das Modell aus dem obigen Beispiel befindet sich in der Datei `./db/models/Article.js` .

```
const express = require('express');
const Articles = require('./db/models/Article');

module.exports = function (app) {
  const routes = express.Router();

  routes.get('/articles', (req, res) => {
    Articles.find().limit(5).lean().exec((err, doc) => {
      if (doc.length > 0) {
        res.send({ data: doc });
      } else {
        res.send({ success: false, message: 'No documents retrieved' });
      }
    });
  });

app.use('/api', routes);
};
```

Wir können die Daten jetzt aus unserer Datenbank abrufen, indem Sie eine HTTP-Anforderung an

diesen Endpunkt senden. Einige wichtige Dinge jedoch:

1. Limit macht genau so, wie es aussieht. Ich bekomme nur 5 Dokumente zurück.
2. Lean entledigt sich dem rohen BSON und reduziert so die Komplexität und den Overhead. Nicht benötigt. Aber nützlich
3. Wenn Sie `find` anstelle von `findOne` , `findOne` , dass `doc.length` größer als 0 ist. Dies ist darauf zurückzuführen, dass `find` immer ein Array zurückgibt. Ein leeres Array wird also Ihren Fehler nur behandeln, wenn er auf Länge geprüft wird
4. Ich persönlich schicke die Fehlermeldung gerne in diesem Format. Ändern Sie es nach Ihren Bedürfnissen. Gleiches für das zurückgegebene Dokument.
5. Der Code in diesem Beispiel wird unter der Annahme geschrieben, dass Sie ihn in einer anderen Datei und nicht direkt auf dem Express-Server gespeichert haben. Um dies auf dem Server aufzurufen, fügen Sie diese in Ihren Servercode ein:

```
const app = express();
require('./path/to/this/file')(app) //
```

MongoDB-Integration für Node.js / Express.js online lesen: https://riptutorial.com/de/node-js/topic/9020/mongodb-integration-fur-node-js---express-js

Kapitel 55: MSSQL-Integration

Einführung

Um jede Datenbank mit nodejs zu integrieren, benötigen Sie ein Treiberpaket oder Sie können es ein npm-Modul nennen, das Ihnen eine grundlegende API für die Verbindung mit der Datenbank und die Durchführung von Interaktionen bietet. Gleiches gilt für die mssql-Datenbank. Hier werden wir mssql mit nodejs integrieren und einige grundlegende Abfragen für SQL-Tabellen durchführen.

Bemerkungen

Wir haben davon ausgegangen, dass auf dem lokalen Computer eine lokale Instanz des mssql-Datenbankservers ausgeführt wird. Sie können auf dieses Dokument verweisen, um dasselbe zu tun.

Stellen Sie außerdem sicher, dass der entsprechende Benutzer mit den hinzugefügten Berechtigungen erstellt wurde.

Examples

Verbindung mit SQL über. Mssql npm Modul

Wir beginnen mit dem Erstellen einer einfachen Knotenanwendung mit einer Basisstruktur, verbinden sich dann mit der lokalen SQL Server-Datenbank und führen einige Abfragen für diese Datenbank durch.

Schritt 1: Erstellen Sie ein Verzeichnis / einen Ordner mit dem Namen des Projekts, das Sie erstellen möchten. Initialisieren Sie eine Knotenanwendung mit dem Befehl *npm init* , der eine package.json im aktuellen Verzeichnis erstellt.

```
mkdir mySqlApp
//folder created
cd mwSqlApp
//change to newly created directory
npm init
//answer all the question ..
npm install
//This will complete quickly since we have not added any packages to our app.
```

Schritt 2: Jetzt erstellen wir eine App.js-Datei in diesem Verzeichnis und installieren einige Pakete, die wir benötigen, um eine Verbindung zu sql db herzustellen.

```
sudo gedit App.js
//This will create App.js file , you can use your fav. text editor :)
npm install --save mssql
//This will install the mssql package to you app
```

Schritt 3: Jetzt fügen wir unserer Anwendung eine grundlegende Konfigurationsvariable hinzu, die vom mssql-Modul zum Herstellen einer Verbindung verwendet wird.

```
console.log("Hello world, This is an app to connect to sql server.");
var config = {
        "user": "myusername", //default is sa
        "password": "yourStrong(!)Password",
        "server": "localhost", // for local machine
        "database": "staging", // name of database
        "options": {
            "encrypt": true
        }
      }

sql.connect(config, err => {
    if(err){
        throw err ;
    }
    console.log("Connection Successful !");

    new sql.Request().query('select 1 as number', (err, result) => {
        //handle err
        console.dir(result)
        // This example uses callbacks strategy for getting results.
    })

});

sql.on('error', err => {
    // ... error handler
    console.log("Sql database connection error " ,err);
})
```

Schritt 4: Dies ist der einfachste Schritt, in dem wir die Anwendung starten und die Anwendung eine Verbindung zum SQL-Server herstellt und einige einfache Ergebnisse ausgibt.

```
node App.js
// Output :
// Hello world, This is an app to connect to sql server.
// Connection Successful !
// 1
```

> Um Versprechungen oder async zur Abfrageausführung zu verwenden, beziehen Sie sich auf die offiziellen Dokumente des mssql-Pakets:
>
> - Versprechen
> - Async / Warten

MSSQL-Integration online lesen: https://riptutorial.com/de/node-js/topic/9884/mssql-integration

Kapitel 56: Multithreading

Einführung

Node.js wurde als Einzelthread konzipiert. Für alle praktischen Zwecke werden Anwendungen, die mit Node gestartet werden, in einem einzigen Thread ausgeführt.

Node.js selbst führt jedoch mehrere Threads aus. E / A-Operationen und dergleichen werden von einem Thread-Pool ausgeführt. Ferner wird jede Instanz einer Knotenanwendung in einem anderen Thread ausgeführt. Um Multithread-Anwendungen auszuführen, werden mehrere Instanzen gestartet.

Bemerkungen

Das Verständnis der Ereignisschleife ist wichtig, um zu verstehen, wie und warum mehrere Threads verwendet werden.

Examples

Cluster

Mit dem `cluster` Modul kann dieselbe Anwendung mehrmals gestartet werden.

Clustering ist wünschenswert, wenn die verschiedenen Instanzen den gleichen Ausführungsfluss haben und nicht voneinander abhängig sind. In diesem Szenario haben Sie einen Master, der die Gabeln und die Gabeln (oder Kinder) starten kann. Die Kinder arbeiten unabhängig voneinander und haben einen Bereich mit Ram und Ereignisschleife.

Das Einrichten von Clustern kann für Websites / APIs von Vorteil sein. Jeder Thread kann jeden Kunden bedienen, da er nicht von anderen Threads abhängt. Eine Datenbank (wie Redis) würde zum Teilen von Cookies verwendet, da **Variablen nicht geteilt werden können!** zwischen den Fäden.

```
// runs in each instance
var cluster = require('cluster');
var numCPUs = require('os').cpus().length;

console.log('I am always called');

if (cluster.isMaster) {
    // runs only once (within the master);
    console.log('I am the master, launching workers!');
    for(var i = 0; i < numCPUs; i++) cluster.fork();

} else {
    // runs in each fork
    console.log('I am a fork!');
```

```
    // here one could start, as an example, a web server

}

console.log('I am always called as well');
```

Kindprozess

Untergeordnete Prozesse sind der Weg zu gehen, wenn Prozesse unabhängig voneinander mit unterschiedlicher Initialisierung und Bedenken ausgeführt werden sollen. Wie Gabeln in Clustern wird ein `child_process` in seinem Thread ausgeführt. Im Gegensatz zu Gabeln kann er jedoch mit seinem übergeordneten `child_process` kommunizieren.

Die Kommunikation verläuft in beide Richtungen, sodass Eltern und Kinder auf Nachrichten warten und Nachrichten senden können.

Elternteil (../parent.js)

```
var child_process = require('child_process');
console.log('[Parent]', 'initalize');

var child1 = child_process.fork(__dirname + '/child');
child1.on('message', function(msg) {
    console.log('[Parent]', 'Answer from child: ', msg);
});

// one can send as many messages as one want
child1.send('Hello'); // Hello to you too :)
child1.send('Hello'); // Hello to you too :)

// one can also have multiple children
var child2 = child_process.fork(__dirname + '/child');
```

Kind (../child.js)

```
// here would one initialize this child
// this will be executed only once
console.log('[Child]', 'initalize');

// here one listens for new tasks from the parent
process.on('message', function(messageFromParent) {

    //do some intense work here
    console.log('[Child]', 'Child doing some intense work');

    if(messageFromParent == 'Hello') process.send('Hello to you too :)');
    else process.send('what?');

})
```

Neben der Nachricht können Sie viele Ereignisse wie "Fehler", "Verbunden" oder "Trennen" abhören.

Das Starten eines untergeordneten Prozesses hat bestimmte Kosten zur Folge. Man möchte so

wenig wie möglich davon laichen.

Multithreading online lesen: https://riptutorial.com/de/node-js/topic/10592/multithreading

Kapitel 57: Mungo-Bibliothek

Examples

Verbinden Sie sich mit MongoDB mit Mongoose

Installieren Sie zuerst Mongoose mit:

```
npm install mongoose
```

`server.js` Sie es `server.js` als Abhängigkeiten zu `server.js` :

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
```

Als Nächstes erstellen Sie das Datenbankschema und den Namen der Sammlung:

```
var schemaName = new Schema({
    request: String,
    time: Number
}, {
    collection: 'collectionName'
});
```

Erstellen Sie ein Modell und stellen Sie eine Verbindung zur Datenbank her:

```
var Model = mongoose.model('Model', schemaName);
mongoose.connect('mongodb://localhost:27017/dbName');
```

Starten Sie anschließend MongoDB und führen Sie `server.js` mit dem `node server.js`

Um zu überprüfen, ob wir erfolgreich eine Verbindung zur Datenbank hergestellt haben, können wir die Ereignisse `open` , `error` des `mongoose.connection` Objekts verwenden.

```
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
  // we're connected!
});
```

Speichern Sie Daten in MongoDB mithilfe der Routen von Mongoose und Express.js

Konfiguration

Installieren Sie zuerst die erforderlichen Pakete mit:

```
npm install express cors mongoose
```

Code

`server.js` Datei `server.js` dann Abhängigkeiten `server.js` , erstellen Sie das Datenbankschema und den Namen der Sammlung, erstellen Sie einen Express.js-Server und stellen Sie eine Verbindung zu MongoDB her

```
var express = require('express');
var cors = require('cors'); // We will use CORS to enable cross origin domain requests.
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var app = express();

var schemaName = new Schema({
    request: String,
    time: Number
}, {
    collection: 'collectionName'
});

var Model = mongoose.model('Model', schemaName);
mongoose.connect('mongodb://localhost:27017/dbName');

var port = process.env.PORT || 8080;
app.listen(port, function() {
    console.log('Node.js listening on port ' + port);
});
```

Fügen Sie nun Express.js-Routen hinzu, die wir zum Schreiben der Daten verwenden werden:

```
app.get('/save/:query', cors(), function(req, res) {
    var query = req.params.query;

    var savedata = new Model({
        'request': query,
        'time': Math.floor(Date.now() / 1000) // Time of save the data in unix timestamp
format
    }).save(function(err, result) {
        if (err) throw err;

        if(result) {
            res.json(result)
        }
    })
})
```

Hier ist die `query` der Parameter `<query>` aus der eingehenden HTTP-Anforderung, der in MongoDB gespeichert wird:

```
var savedata = new Model({
    'request': query,
    //...
```

Wenn beim Versuch, in MongoDB zu schreiben, ein Fehler auftritt, wird auf der Konsole eine Fehlermeldung angezeigt. Wenn alles erfolgreich ist, werden die gespeicherten Daten im JSON-Format auf der Seite angezeigt.

```
//...
}).save(function(err, result) {
    if (err) throw err;

    if(result) {
        res.json(result)
    }
})
//...
```

Jetzt müssen Sie MongoDB starten und Ihre `server.js` Datei mit dem `node server.js` .

Verwendungszweck

Um dies zum Speichern von Daten zu verwenden, rufen Sie in Ihrem Browser die folgende URL auf:

```
http://localhost:8080/save/<query>
```

Wo `<query>` ist die neue Anforderung , die Sie speichern möchten.

Beispiel:

```
http://localhost:8080/save/JavaScript%20is%20Awesome
```

Ausgabe im JSON-Format:

```
{
    __v: 0,
    request: "JavaScript is Awesome",
    time: 1469411348,
    _id: "57957014b93bc8640f2c78c4"
}
```

Finden Sie Daten in MongoDB mithilfe von Mongoose- und Express.js-Routen

Konfiguration

Installieren Sie zuerst die erforderlichen Pakete mit:

```
npm install express cors mongoose
```

Code

`server.js` Abhängigkeiten `server.js` , erstellen Sie das Datenbankschema und den Namen der Sammlung, erstellen Sie einen Express.js-Server und stellen Sie eine Verbindung zu MongoDB her

```
var express = require('express');
var cors = require('cors'); // We will use CORS to enable cross origin domain requests.
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var app = express();

var schemaName = new Schema({
    request: String,
    time: Number
}, {
    collection: 'collectionName'
});

var Model = mongoose.model('Model', schemaName);
mongoose.connect('mongodb://localhost:27017/dbName');

var port = process.env.PORT || 8080;
app.listen(port, function() {
    console.log('Node.js listening on port ' + port);
});
```

Fügen Sie nun Express.js-Routen hinzu, die wir zur Abfrage der Daten verwenden werden:

```
app.get('/find/:query', cors(), function(req, res) {
    var query = req.params.query;

    Model.find({
        'request': query
    }, function(err, result) {
        if (err) throw err;
        if (result) {
            res.json(result)
        } else {
            res.send(JSON.stringify({
                error : 'Error'
            }))
        }
    })
})
```

Angenommen, die folgenden Dokumente befinden sich in der Sammlung des Modells:

```
{
        "_id" : ObjectId("578abe97522ad414b8eeb55a"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710551
}
{
```

```
        "_id" : ObjectId("578abe9b522ad414b8eeb55b"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710555
}
{
        "_id" : ObjectId("578abea0522ad414b8eeb55c"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710560
}
```

Das Ziel ist es, alle Dokumente, die `"JavaScript is Awesome"` unter der `"JavaScript is Awesome"` `"request"` zu finden und anzuzeigen.

Starten Sie dazu MongoDB und führen Sie `server.js` mit dem `node server.js` :

Verwendungszweck

Um Daten zu finden, rufen Sie die folgende URL in einem Browser auf:

```
http://localhost:8080/find/<query>
```

Wobei `<query>` die Suchabfrage ist.

Beispiel:

```
http://localhost:8080/find/JavaScript%20is%20Awesome
```

Ausgabe:

```
[{
    _id: "578abe97522ad414b8eeb55a",
    request: "JavaScript is Awesome",
    time: 1468710551,
    __v: 0
},
{
    _id: "578abe9b522ad414b8eeb55b",
    request: "JavaScript is Awesome",
    time: 1468710555,
    __v: 0
},
{
    _id: "578abea0522ad414b8eeb55c",
    request: "JavaScript is Awesome",
    time: 1468710560,
    __v: 0
}]
```

Finden Sie Daten in MongoDB mit Mongoose, Express.js Routes und $ text Operator

Konfiguration

Installieren Sie zuerst die erforderlichen Pakete mit:

```
npm install express cors mongoose
```

Code

`server.js` Abhängigkeiten `server.js` , erstellen Sie das Datenbankschema und den Namen der Sammlung, erstellen Sie einen Express.js-Server und stellen Sie eine Verbindung zu MongoDB her

```
var express = require('express');
var cors = require('cors'); // We will use CORS to enable cross origin domain requests.
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var app = express();

var schemaName = new Schema({
    request: String,
    time: Number
}, {
    collection: 'collectionName'
});

var Model = mongoose.model('Model', schemaName);
mongoose.connect('mongodb://localhost:27017/dbName');

var port = process.env.PORT || 8080;
app.listen(port, function() {
    console.log('Node.js listening on port ' + port);
});
```

Fügen Sie nun Express.js-Routen hinzu, die wir zur Abfrage der Daten verwenden werden:

```
app.get('/find/:query', cors(), function(req, res) {
    var query = req.params.query;

    Model.find({
        'request': query
    }, function(err, result) {
        if (err) throw err;
        if (result) {
            res.json(result)
        } else {
            res.send(JSON.stringify({
                error : 'Error'
            }))
        }
    })
})
```

Angenommen, die folgenden Dokumente befinden sich in der Sammlung des Modells:

```
{
        "_id" : ObjectId("578abe97522ad414b8eeb55a"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710551
}
{
        "_id" : ObjectId("578abe9b522ad414b8eeb55b"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710555
}
{
        "_id" : ObjectId("578abea0522ad414b8eeb55c"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710560
}
```

Das Ziel besteht darin, alle Dokumente zu finden und anzuzeigen, die nur das Wort `"JavaScript"` unter der `"request"` .

Erstellen Sie dazu zunächst einen *Textindex* für `"request"` in der Sammlung. `server.js` Sie hierzu den folgenden Code zu `server.js` :

```
schemaName.index({ request: 'text' });
```

Und ersetzen Sie:

```
    Model.find({
        'request': query
    }, function(err, result) {
```

Mit:

```
Model.find({
    $text: {
        $search: query
    }
}, function(err, result) {
```

Hier verwenden wir `$text` und `$search` MongoDB-Operatoren für das Finden aller Dokumente in collection `collectionName` die mindestens ein Wort aus der angegebenen Suchabfrage enthalten.

Verwendungszweck

Um Daten zu finden, rufen Sie die folgende URL in einem Browser auf:

```
http://localhost:8080/find/<query>
```

Wobei `<query>` die Suchabfrage ist.

Beispiel:

```
http://localhost:8080/find/JavaScript
```

Ausgabe:

```
[{
    _id: "578abe97522ad414b8eeb55a",
    request: "JavaScript is Awesome",
    time: 1468710551,
    __v: 0
},
{
    _id: "578abe9b522ad414b8eeb55b",
    request: "JavaScript is Awesome",
    time: 1468710555,
    __v: 0
},
{
    _id: "578abea0522ad414b8eeb55c",
    request: "JavaScript is Awesome",
    time: 1468710560,
    __v: 0
}]
```

Indizes in Modellen.

MongoDB unterstützt sekundäre Indizes. In Mongoose definieren wir diese Indizes innerhalb unseres Schemas. Die Definition von Indizes auf Schemaebene ist erforderlich, wenn zusammengesetzte Indizes erstellt werden müssen.

Mungo-Verbindung

```
var strConnection = 'mongodb://localhost:27017/dbName';
var db = mongoose.createConnection(strConnection)
```

Ein grundlegendes Schema erstellen

```
var Schema = require('mongoose').Schema;
var usersSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true
    },
    email: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    created: {
        type: Date,
```

```
        default: Date.now
    }
});

var usersModel = db.model('users', usersSchema);
module.exports = usersModel;
```

Standardmäßig fügt mongoose unserem Modell zwei neue Felder hinzu, auch wenn diese nicht im Modell definiert sind. Diese Felder sind:

_Ich würde

Mongoose weist jedem Ihrer Schemas standardmäßig ein _id-Feld zu, wenn eines nicht an den Schemakonstruktor übergeben wird. Der zugewiesene Typ ist eine ObjectId, die mit dem Standardverhalten von MongoDB übereinstimmt. Wenn Sie nicht möchten, dass eine _id zu Ihrem Schema hinzugefügt wird, können Sie sie mit dieser Option deaktivieren.

```
var usersSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true
    }, {
        _id: false
});
```

__v oder versionKey

Der versionKey ist eine Eigenschaft, die bei der erstmaligen Erstellung durch Mongoose in jedem Dokument festgelegt wird. Dieser Schlüsselwert enthält die interne Revision des Dokuments. Der Name dieser Dokumenteigenschaft ist konfigurierbar.

Sie können dieses Feld einfach in der Modellkonfiguration deaktivieren:

```
var usersSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true
    }, {
    versionKey: false
});
```

Zusammengesetzte Indizes

Neben den von Mongoose erstellten Indizes können wir weitere Indizes erstellen.

```
usersSchema.index({username: 1 });
usersSchema.index({email: 1 });
```

In diesem Fall verfügt unser Modell über zwei weitere Indizes, einen für das Feld Benutzername und einen für das E-Mail-Feld. Wir können jedoch zusammengesetzte Indizes erstellen.

```
usersSchema.index({username: 1, email: 1 });
```

Auswirkungen auf den Index

Standardmäßig ruft mongoose immer den configureIndex für jeden Index nacheinander auf und gibt ein "Index" -Ereignis für das Modell aus, wenn alle Aufrufe von sichuerindex erfolgreich waren oder ein Fehler auftrat.

In MongoDB ist sureIndex seit der Version 3.0.0 veraltet. Jetzt ist es ein Alias für createIndex.

Es wird empfohlen, das Verhalten zu deaktivieren, indem Sie die Option autoIndex Ihres Schemas auf false oder global für die Verbindung setzen, indem Sie die Option config.autoIndex auf false setzen.

```
usersSchema.set('autoIndex', false);
```

Nützliche Mongoose-Funktionen

Mongoose enthält einige eingebaute Funktionen, die auf dem standard `find()` aufbauen.

```
doc.find({'some.value':5},function(err,docs){
    //returns array docs
});

doc.findOne({'some.value':5},function(err,doc){
    //returns document doc
});

doc.findById(obj._id,function(err,doc){
    //returns document doc
});
```

Finden Sie Daten in Mongodb mit Versprechungen

Konfiguration

Installieren Sie zuerst die erforderlichen Pakete mit:

```
npm install express cors mongoose
```

Code

`server.js` Abhängigkeiten `server.js` , erstellen Sie das Datenbankschema und den Namen der Sammlung, erstellen Sie einen Express.js-Server und stellen Sie eine Verbindung zu MongoDB her

```
var express = require('express');
var cors = require('cors'); // We will use CORS to enable cross origin domain requests.
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var app = express();

var schemaName = new Schema({
    request: String,
    time: Number
}, {
    collection: 'collectionName'
});

var Model = mongoose.model('Model', schemaName);
mongoose.connect('mongodb://localhost:27017/dbName');

var port = process.env.PORT || 8080;
app.listen(port, function() {
    console.log('Node.js listening on port ' + port);
});

app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});

app.use(function(req, res, next) {
  res.status(404).send('Sorry cant find that!');
});
```

Fügen Sie nun Express.js-Routen hinzu, die wir zur Abfrage der Daten verwenden werden:

```
app.get('/find/:query', cors(), function(req, res, next) {
    var query = req.params.query;

    Model.find({
        'request': query
    })
    .exec() //remember to add exec, queries have a .then attribute but aren't promises
    .then(function(result) {
        if (result) {
            res.json(result)
        } else {
            next() //pass to 404 handler
        }
    })
    .catch(next) //pass to error handler
})
```

Angenommen, die folgenden Dokumente befinden sich in der Sammlung des Modells:

```
{
        "_id" : ObjectId("578abe97522ad414b8eeb55a"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710551
}
{
        "_id" : ObjectId("578abe9b522ad414b8eeb55b"),
```

```
        "request" : "JavaScript is Awesome",
        "time" : 1468710555
}
{
        "_id" : ObjectId("578abea0522ad414b8eeb55c"),
        "request" : "JavaScript is Awesome",
        "time" : 1468710560
}
```

Das Ziel ist es, alle Dokumente, die `"JavaScript is Awesome"` unter der `"JavaScript is Awesome"` `"request"` zu finden und anzuzeigen.

Starten Sie dazu MongoDB und führen Sie `server.js` mit dem `node server.js` :

Verwendungszweck

Um Daten zu finden, rufen Sie die folgende URL in einem Browser auf:

```
http://localhost:8080/find/<query>
```

Wobei `<query>` die Suchabfrage ist.

Beispiel:

```
http://localhost:8080/find/JavaScript%20is%20Awesome
```

Ausgabe:

```
[{
    _id: "578abe97522ad414b8eeb55a",
    request: "JavaScript is Awesome",
    time: 1468710551,
    __v: 0
},
{
    _id: "578abe9b522ad414b8eeb55b",
    request: "JavaScript is Awesome",
    time: 1468710555,
    __v: 0
},
{
    _id: "578abea0522ad414b8eeb55c",
    request: "JavaScript is Awesome",
    time: 1468710560,
    __v: 0
}]
```

Mungo-Bibliothek online lesen: https://riptutorial.com/de/node-js/topic/3486/mungo-bibliothek

Kapitel 58: MySQL-Integration

Einführung

In diesem Thema erfahren Sie, wie Sie Node.js mithilfe des MYSQL-Datenbankverwaltungstools integrieren. Sie werden verschiedene Möglichkeiten kennenlernen, um mithilfe eines nodejs-Programms und -Skripts eine Verbindung zu Daten herzustellen und mit ihnen zu interagieren.

Examples

Fragen Sie ein Verbindungsobjekt mit Parametern ab

Wenn Sie benutzergenerierte Inhalte in der SQL verwenden möchten, ist dies mit Parametern erledigt. Für die Suche nach Benutzern mit dem Namen `aminadav` sollten Sie beispielsweise `aminadav` tun:

```
var username = 'aminadav';
var querystring = 'SELECT name, email from users where name = ?';
connection.query(querystring, [username], function(err, rows, fields) {
  if (err) throw err;
  if (rows.length) {
    rows.forEach(function(row) {
      console.log(row.name, 'email address is', row.email);
    });
  } else {
    console.log('There were no results.');
  }
});
```

Verwenden eines Verbindungspools

ein. Mehrere Abfragen gleichzeitig ausführen

Alle Abfragen in der MySQL-Verbindung werden nacheinander ausgeführt. Das bedeutet, wenn Sie 10 Abfragen durchführen möchten und jede Abfrage 2 Sekunden dauert, dauert es 20 Sekunden, bis die gesamte Ausführung abgeschlossen ist. Die Lösung besteht darin, 10 Verbindungen zu erstellen und jede Abfrage in einer anderen Verbindung auszuführen. Dies kann automatisch über den Verbindungspool erfolgen

```
var pool  = mysql.createPool({
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bobby',
  password        : 'pass',
  database        : 'schema'
});

for(var i=0;i<10;i++){
```

```
  pool.query('SELECT ` as example', function(err, rows, fields) {
    if (err) throw err;
    console.log(rows[0].example); //Show 1
  });
 }
```

Es werden alle 10 Abfragen parallel ausgeführt.

Wenn Sie den `pool` verwenden, brauchen Sie die Verbindung nicht mehr. Sie können den Pool direkt abfragen. Das MySQL-Modul sucht nach der nächsten freien Verbindung, um Ihre Abfrage auszuführen.

b. Mehrmandantenfähigkeit auf Datenbankservern mit verschiedenen darauf gehosteten Datenbanken.

Multitenancy ist heutzutage eine häufige Anforderung an Unternehmensanwendungen. Das Erstellen eines Verbindungspools für jede Datenbank im Datenbankserver wird nicht empfohlen. Stattdessen können Sie einen Verbindungspool mit dem Datenbankserver erstellen und dann bei Bedarf zwischen auf dem Datenbankserver gehosteten Datenbanken wechseln.

Angenommen, unsere Anwendung verfügt über unterschiedliche Datenbanken für jedes auf dem Datenbankserver gehostete Unternehmen. Wir werden eine Verbindung zu der jeweiligen Firmendatenbank herstellen, wenn der Benutzer die Anwendung besucht. Hier ist das Beispiel, wie das geht: -

```
var pool  = mysql.createPool({
      connectionLimit : 10,
      host            : 'example.org',
      user            : 'bobby',
      password        : 'pass'
    });

pool.getConnection(function(err, connection){
    if(err){
        return cb(err);
    }
    connection.changeUser({database : "firm1"});
    connection.query("SELECT * from history", function(err, data){
        connection.release();
        cb(err, data);
    });
});
```

Lassen Sie mich das Beispiel aufschlüsseln: -

Bei der Definition der Poolkonfiguration habe ich nicht den Datenbanknamen angegeben, sondern nur den Datenbankserver

```
{
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bobby',
```

```
  password        : 'pass'
}
```

Wenn wir also die spezifische Datenbank auf dem Datenbankserver verwenden möchten, bitten wir die Verbindung, die Datenbank zu treffen, indem Sie Folgendes verwenden: -

```
    connection.changeUser({database : "firm1"});
```

Die offizielle Dokumentation können Sie hier einsehen

Verbinden Sie sich mit MySQL

Eine der einfachsten Möglichkeiten , um MySQL zu verbinden , ist die Verwendung `mysql` - Modul. Dieses Modul übernimmt die Verbindung zwischen der Node.js-App und dem MySQL-Server. Sie können es wie jedes andere Modul installieren:

```
npm install --save mysql
```

Jetzt müssen Sie eine MySQL-Verbindung erstellen, die Sie später abfragen können.

```
const mysql      = require('mysql');
const connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'me',
  password : 'secret',
  database : 'database_schema'
});

connection.connect();

// Execute some query statements
// I.e. SELECT * FROM FOO

connection.end();
```

Im nächsten Beispiel erfahren Sie, wie Sie das `connection` abfragen.

Fragen Sie ein Verbindungsobjekt ohne Parameter ab

Sie senden die Abfrage als String und erhalten als Antwort einen Rückruf mit der Antwort. Der Rückruf gibt `error` , Array von `rows` und Feldern aus. Jede Zeile enthält die gesamte Spalte der zurückgegebenen Tabelle. Hier ist ein Ausschnitt für die folgende Erklärung.

```
connection.query('SELECT name,email from users', function(err, rows, fields) {
  if (err) throw err;

  console.log('There are:', rows.length,' users');
  console.log('First user name is:',rows[0].name)
});
```

Führen Sie eine Reihe von Abfragen mit einer einzelnen Verbindung aus

einem Pool aus

Es kann Situationen geben, in denen Sie einen Pool von MySQL-Verbindungen eingerichtet haben, aber Sie haben mehrere Abfragen, die Sie nacheinander ausführen möchten:

```
SELECT 1;
SELECT 2;
```

Sie *könnten dann* einfach mit `pool.query` wie anderswo ausgeführt laufen. Wenn Sie jedoch nur eine freie Verbindung im Pool haben, müssen Sie warten, bis eine Verbindung verfügbar ist, bevor Sie die zweite Abfrage ausführen können.

Sie können jedoch eine aktive Verbindung aus dem Pool beibehalten und so viele Abfragen ausführen, wie Sie möchten, indem Sie eine einzige Verbindung mit `pool.getConnection` :

```
pool.getConnection(function (err, conn) {
  if (err) return callback(err);

  conn.query('SELECT 1 AS seq', function (err, rows) {
    if (err) throw err;

    conn.query('SELECT 2 AS seq', function (err, rows) {
      if (err) throw err;

      conn.release();
      callback();
    });
  });
});
```

Anmerkung: Sie müssen daran denken, die Verbindung `release` , da sonst eine MySQL-Verbindung für den Rest des Pools verfügbar ist!

Weitere Informationen zum Pooling von MySQL-Verbindungen finden Sie in den MySQL-Dokumenten .

Gibt die Abfrage zurück, wenn ein Fehler auftritt

Sie können die ausgeführte Abfrage an das `err` anhängen, wenn ein Fehler auftritt:

```
var q = mysql.query('SELECT `name` FROM `pokedex` WHERE `id` = ?', [ 25 ], function (err,
result) {
  if (err) {
    // Table 'test.pokedex' doesn't exist
    err.query = q.sql; // SELECT `name` FROM `pokedex` WHERE `id` = 25
    callback(err);
  }
  else {
    callback(null, result);
  }
});
```

Verbindungspool exportieren

```
// db.js

const mysql = require('mysql');

const pool = mysql.createPool({
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bob',
  password        : 'secret',
  database        : 'my_db'
});

module.export = {
  getConnection: (callback) => {
    return pool.getConnection(callback);
  }
}
```

```
// app.js

const db = require('./db');

db.getConnection((err, conn) => {
  conn.query('SELECT something from sometable', (error, results, fields) => {
    // get the results
    conn.release();
  });
});
```

MySQL-Integration online lesen: https://riptutorial.com/de/node-js/topic/1406/mysql-integration

Kapitel 59: Mysql-Verbindungspool

Examples

Verwenden eines Verbindungspools ohne Datenbank

Multitenancy auf Datenbankservern mit mehreren darauf gehosteten Datenbanken erreichen.

Multitenancy ist heutzutage eine häufige Anforderung an Unternehmensanwendungen, und das Erstellen eines Verbindungspools für jede Datenbank im Datenbankserver wird nicht empfohlen. Stattdessen können Sie stattdessen einen Verbindungspool mit dem Datenbankserver erstellen und dann bei Bedarf zwischen den auf dem Datenbankserver gehosteten Datenbanken wechseln.

Angenommen, unsere Anwendung verfügt über unterschiedliche Datenbanken für jedes auf dem Datenbankserver gehostete Unternehmen. Wir werden eine Verbindung zu der jeweiligen Firmendatenbank herstellen, wenn der Benutzer die Anwendung besucht. Hier ist das Beispiel, wie das geht: -

```
var pool  = mysql.createPool({
      connectionLimit : 10,
      host            : 'example.org',
      user            : 'bobby',
      password        : 'pass'
    });

pool.getConnection(function(err, connection){
    if(err){
        return cb(err);
    }
    connection.changeUser({database : "firm1"});
    connection.query("SELECT * from history", function(err, data){
        connection.release();
        cb(err, data);
    });
});
```

Lassen Sie mich das Beispiel aufschlüsseln: -

Bei der Definition der Poolkonfiguration habe ich nicht den Datenbanknamen angegeben, sondern nur den Datenbankserver

```
{
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bobby',
  password        : 'pass'
}
```

Wenn wir also die spezifische Datenbank auf dem Datenbankserver verwenden möchten, bitten wir die Verbindung, die Datenbank zu treffen, indem Sie Folgendes verwenden: -

```
connection.changeUser({database : "firm1"});
```

Die offizielle Dokumentation können Sie hier einsehen

Mysql-Verbindungspool online lesen: https://riptutorial.com/de/node-js/topic/6353/mysql-verbindungspool

Kapitel 60: N-API

Einführung

Die N-API ist eine neue und bessere Möglichkeit, native Module für NodeJS zu erstellen. Die N-API befindet sich noch in einem frühen Stadium, daher kann es zu inkonsistenter Dokumentation kommen.

Examples

Hallo an N-API

Dieses Modul registriert Hallo-Funktion am Hallo-Modul. hallo Funktion druckt Hallo Welt auf Konsole mit `printf` und kehrt `1373` von der nativen Funktion in den Javascript-Aufrufer zurück.

```
#include <node_api.h>
#include <stdio.h>


napi_value say_hello(napi_env env, napi_callback_info info)
{
    napi_value retval;

    printf("Hello world\n");

    napi_create_number(env, 1373, &retval);

    return retval;
}

void init(napi_env env, napi_value exports, napi_value module, void* priv)
{
    napi_status status;
    napi_property_descriptor desc = {
        /*
         * String describing the key for the property, encoded as UTF8.
         */
        .utf8name = "hello",
        /*
         * Set this to make the property descriptor object's value property
         * to be a JavaScript function represented by method.
         * If this is passed in, set value, getter and setter to NULL (since these members
won't be used).
         */
        .method = say_hello,
        /*
         * A function to call when a get access of the property is performed.
         * If this is passed in, set value and method to NULL (since these members won't be
used).
         * The given function is called implicitly by the runtime when the property is
accessed
         * from JavaScript code (or if a get on the property is performed using a N-API call).
         */
```

```
        .getter = NULL,
        /*
         * A function to call when a set access of the property is performed.
         * If this is passed in, set value and method to NULL (since these members won't be
used).
         * The given function is called implicitly by the runtime when the property is set
         * from JavaScript code (or if a set on the property is performed using a N-API call).
         */
        .setter = NULL,
        /*
         * The value that's retrieved by a get access of the property if the property is a
data property.
         * If this is passed in, set getter, setter, method and data to NULL (since these
members won't be used).
         */
        .value = NULL,
        /*
         * The attributes associated with the particular property. See
napi_property_attributes.
         */
        .attributes = napi_default,
        /*
         * The callback data passed into method, getter and setter if this function is
invoked.
         */
        .data = NULL
    };
    /*
     * This method allows the efficient definition of multiple properties on a given object.
     */
    status = napi_define_properties(env, exports, 1, &desc);

    if (status != napi_ok)
        return;
}


NAPI_MODULE(hello, init)
```

N-API online lesen: https://riptutorial.com/de/node-js/topic/10539/n-api

Kapitel 61: Node.js (express.js) mit angle.js Beispielcode

Einführung

Dieses Beispiel zeigt, wie Sie eine einfache Express-App erstellen und dann AngularJS bedienen.

Examples

Unser Projekt erstellen

Wir sind gut zu gehen, also laufen wir wieder von der Konsole:

```
mkdir our_project
cd our_project
```

Jetzt sind wir an dem Ort, an dem unser Code leben wird. Um das Hauptarchiv unseres Projekts zu erstellen, können Sie es ausführen

Ok, aber wie erstellen wir das Express-Skelett-Projekt?

Es ist einfach:

```
npm install -g express express-generator
```

Linux-Distributoren und Mac sollten **sudo verwenden** , um dies zu installieren, da sie im Verzeichnis nodejs installiert sind, auf das nur der **root-** Benutzer zugreifen kann. Wenn alles gut gelaufen ist, können wir endlich das Express-App-Skelett erstellen, einfach ausführen

```
express
```

Dieser Befehl erstellt in unserem Ordner eine Express-Beispielanwendung. Die Struktur ist wie folgt:

```
bin/
public/
routes/
views/
app.js
package.json
```

Wenn wir **npm starten,** gehen Sie zu http: // localhost: 3000 und sehen die Express-App betriebsbereit. Fairerweise haben wir eine Express-App ohne allzu große Umstände generiert. .

Wie funktioniert Express, kurz?

Express ist ein Framework, das auf **Nodejs aufgebaut** ist. Die offizielle Dokumentation finden Sie auf der Express-Site . Für unseren Zweck müssen wir jedoch wissen, dass **Express** für die Wiedergabe der Startseite unserer Anwendung beispielsweise http: // localhost: 3000 / home verantwortlich ist . In der kürzlich erstellten App können wir Folgendes überprüfen:

```
FILE: routes/index.js
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Was dieser Code uns sagt , ist , dass , wenn der Benutzer geht auf http: // localhost: 3000 muss die **Indexansicht** machen und einen **JSON** mit einem Titel - Eigenschaft und Wert Express übergeben. Wenn wir jedoch das Views-Verzeichnis überprüfen und index.jade öffnen, können wir Folgendes sehen:

```
extends layout
block content
    h1= title
    p Welcome to #{title}
```

Dies ist eine weitere leistungsstarke Express-Funktion, **Template Engines** . Sie ermöglichen das Rendern von Inhalten auf der Seite durch Übergabe von Variablen oder das Erben einer anderen Vorlage, sodass Ihre Seiten kompakter und für andere verständlicher sind. Die Dateierweiterung ist **.jade** , soweit ich weiß, dass **Jade** den Namen für **Pug** geändert hat, im Grunde ist es die gleiche Template-Engine, jedoch mit einigen Updates und Kernmodifikationen.

Pug installieren und Express Template Engine aktualisieren.

Ok, um Pug als Template Engine für unser Projekt zu verwenden, müssen wir Folgendes ausführen:

```
npm install --save pug
```

Dadurch wird Pug als Abhängigkeit von unserem Projekt installiert und in **package.json gespeichert** . Um es zu benutzen, müssen wir die Datei **app.js ändern** :

```
var app = express();
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

Und ersetzen Sie die Sichtlinienmaschine durch Mops, und das ist alles. Wir können unser Projekt erneut mit **npm start** ausführen und wir werden sehen, dass alles gut funktioniert.

Wie passt AngularJS in all das?

AngularJS ist ein Javascript- **MVW** -Framework (Model-View-Whatever), das hauptsächlich zum Erstellen von **SPA** (Simple Page Application) verwendet wird. Die Installation ist relativ einfach. Sie können die AngularJS-Website **besuchen** und die neueste Version **v1.6.4 herunterladen** .

Nach dem Herunterladen von AngularJS, wenn die Datei in unseren **öffentlichen / javascripts-** Ordner in unserem Projekt kopiert werden soll, eine kleine Erklärung. Dies ist der Ordner, der die statischen Assets unserer Website, Bilder, CSS, Javacipt-Dateien usw. bereitstellt. Natürlich ist dies über die **app.js-** Datei konfigurierbar, aber wir machen es einfach. Jetzt erstellen wir eine Datei namens **ng-app.js** , die Datei, in der sich unsere Anwendung befindet, in unserem öffentlichen Ordner "javascripts", genau dort, wo AngularJS lebt. Um AngularJS auf den **neuesten** Stand zu bringen, müssen wir den Inhalt von **views / layout.pug** wie folgt ändern:

```
doctype html
html(ng-app='first-app')
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body(ng-controller='indexController')
    block content

  script(type='text-javascript', src='javascripts/angular.min.js')
  script(type='text-javascript', src='javascripts/ng-app.js')
```

Was machen wir hier? **Nun** , wir enthalten AngularJS Core und unsere kürzlich erstellte Datei **ng-app.js.** Wenn also die Vorlage gerendert wird, wird AngularJS **aktiviert. Beachten** Sie die Verwendung der **ng-app-** Direktive AngularJS, dass dies unser Anwendungsname ist und dabei bleiben sollte.
Der Inhalt unserer **ng-app.js** ist also:

```
angular.module('first-app', [])
       .controller('indexController', ['$scope', indexController]);

function indexController($scope) {
    $scope.name = 'sigfried';
}
```

Wir verwenden hier die grundlegendste AngularJS-Funktion, die bidirektionale **Datenbindung. Dadurch** können wir den Inhalt unserer Ansicht und des Controllers sofort aktualisieren. Dies ist eine sehr einfache Erklärung. Sie können jedoch eine Recherche in Google oder StackOverflow durchführen wie es wirklich funktioniert

Wir haben also die grundlegenden Blöcke unserer AngularJS-Anwendung, aber es gibt etwas, das wir tun müssen. Wir müssen unsere index.pug-Seite aktualisieren, um die Änderungen unserer Winkel-App zu sehen. Lassen Sie uns dies tun:

```
extends layout
block content
  div(ng-controller='indexController')
    h1= title
    p Welcome {{name}}
    input(type='text' ng-model='name')
```

Hier binden wir nur die Eingabe an unseren definierten Eigenschaftsnamen im AngularJS-Bereich in unserem Controller:

```
$scope.name = 'sigfried';
```

Der Zweck davon ist, dass wenn wir den Text in der Eingabe ändern, der Absatz oben den Inhalt innerhalb von {{Name}} aktualisiert, dies wird **Interpolation genannt** , ein weiteres AngularJS-Feature, um unseren Inhalt in der Vorlage darzustellen.

Nun ist alles eingerichtet, wir können jetzt **npm starten.** Gehen Sie zu http: // localhost: 3000 und sehen Sie unsere Express-Anwendung, die die Seite bedient, und AngularJS, die das Front-End der Anwendung verwaltet.

Node.js (express.js) mit angle.js Beispielcode online lesen: https://riptutorial.com/de/node-js/topic/9757/node-js--express-js--mit-angle-js-beispielcode

Kapitel 62: Node.js als Dienst ausführen

Einführung

Im Gegensatz zu vielen anderen Webservern wird Node nicht als Standarddienst installiert. In der Produktion ist es jedoch besser, es als dæmon laufen zu lassen, das von einem init-System verwaltet wird.

Examples

Node.js als systemd dæmon

systemd ist das *De-facto*- Init-System in den meisten Linux-Distributionen. Nachdem der Knoten für die Ausführung mit systemd konfiguriert wurde, können Sie ihn mit dem `service` verwalten.

Erstens benötigt es eine Konfigurationsdatei, erstellen wir sie. Für Debian-basierte Distributionen ist dies in `/etc/systemd/system/node.service`

```
[Unit]
Description=My super nodejs app

[Service]
# set the working directory to have consistent relative paths
WorkingDirectory=/var/www/app

# start the server file (file is relative to WorkingDirectory here)
ExecStart=/usr/bin/node serverCluster.js

# if process crashes, always try to restart
Restart=always

# let 500ms between the crash and the restart
RestartSec=500ms

# send log tot syslog here (it doesn't compete with other log config in the app itself)
StandardOutput=syslog
StandardError=syslog

# nodejs process name in syslog
SyslogIdentifier=nodejs

# user and group starting the app
User=www-data
Group=www-data

# set the environement (dev, prod…)
Environment=NODE_ENV=production


[Install]
# start node at multi user system level (= sysVinit runlevel 3)
WantedBy=multi-user.target
```

Es ist jetzt möglich, die App zu starten, zu stoppen und erneut zu starten:

```
service node start
service node stop
service node restart
```

Um systemd `systemctl enable node` beim Booten automatisch zu starten, geben Sie `systemctl enable node` : `systemctl enable node` .

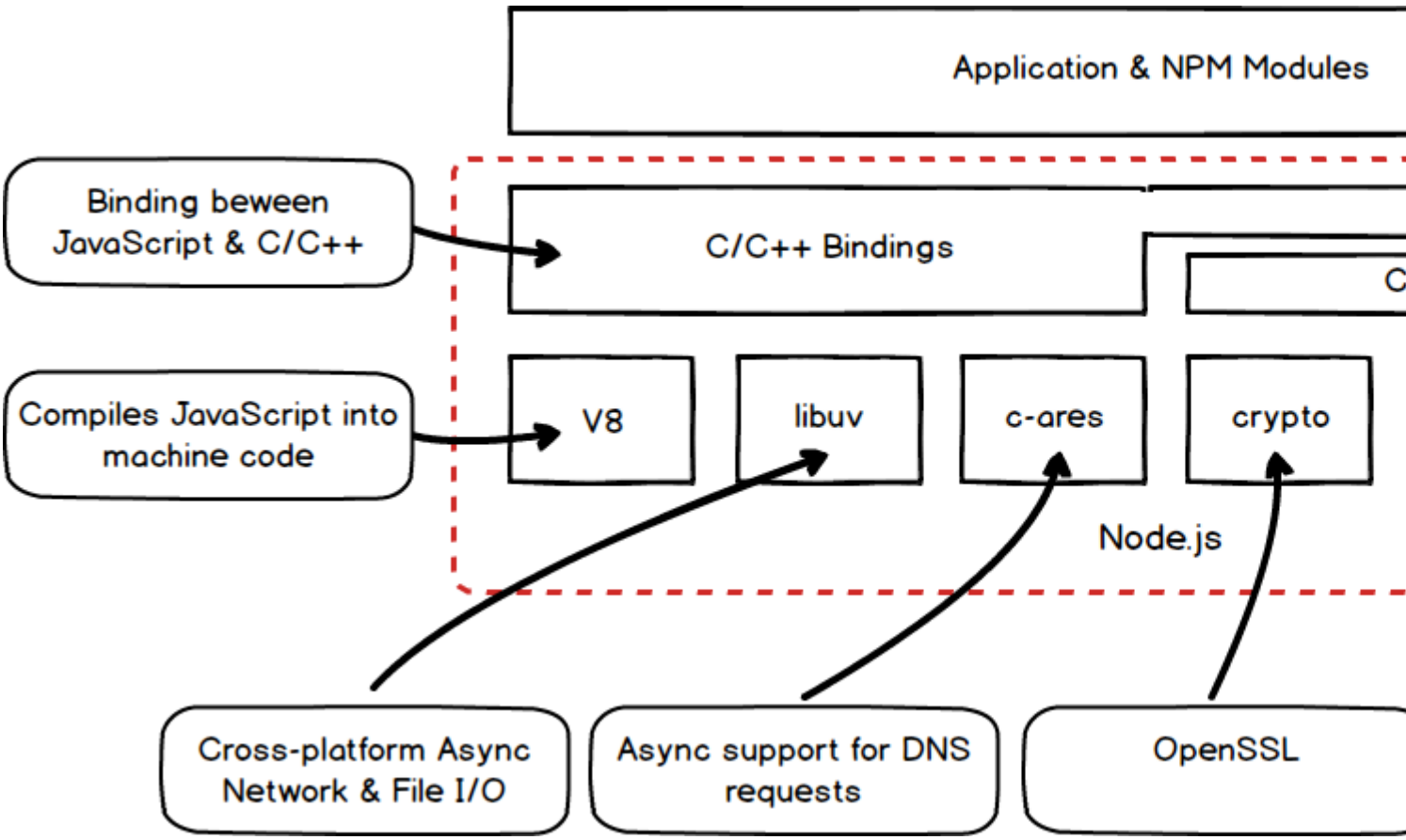
Das ist alles, Knoten läuft jetzt als dæmon.

Node.js als Dienst ausführen online lesen: https://riptutorial.com/de/node-js/topic/9258/node-js-als-dienst-ausfuhren

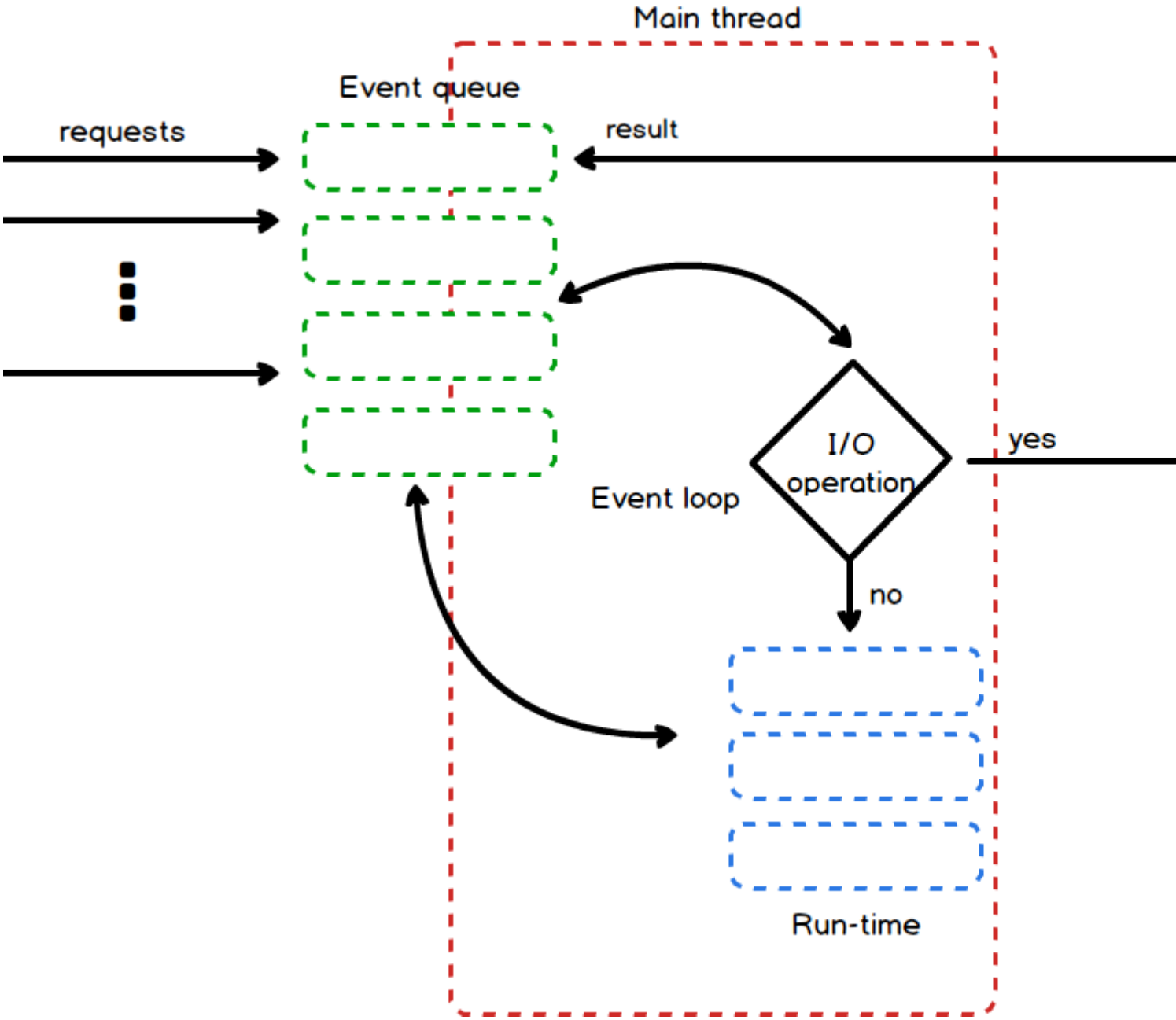
Kapitel 63: Node.js Architektur & Inneres

Examples

Node.js - unter der Haube



Node.js - in Bewegung



Node.js Architektur & Inneres online lesen: https://riptutorial.com/de/node-js/topic/5892/node-js-architektur--amp--inneres

Kapitel 64: Node.js Design Fundamental

Examples

Die Philosophie von Node.js

Kleiner Kern , **Kleines Modul** : -

Erstellen Sie kleine Module und Module für einen bestimmten Zweck, nicht nur in Bezug auf die Codegröße, sondern auch in Bezug auf den Umfang, der einem einzigen Zweck dient

```
    a - "Small is beautiful"
    b - "Make each program do one thing well."
```

Das Reaktormuster

Das Reactor Pattern ist das Herzstück der `node.js` Asynchronität. Das System konnte mit einer Reihe von Ereignisgeneratoren und Ereignishandlern mithilfe einer Ereignisschleife, die kontinuierlich ausgeführt wird, als Single-Thread-Prozess implementiert werden.

Die nicht blockierende E / A-Engine von Node.js - libuv -

Das Observer Pattern (EventEmitter) führt eine Liste von Angehörigen / Beobachtern und benachrichtigt diese

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

var ringBell = function ringBell()
{
  console.log('tring tring tring');
}
eventEmitter.on('doorOpen', ringBell);

eventEmitter.emit('doorOpen');
```

Node.js Design Fundamental online lesen: https://riptutorial.com/de/node-js/topic/6274/node-js-design-fundamental

Kapitel 65: Node.js installieren

Examples

Installieren Sie Node.js unter Ubuntu

Verwenden des apt Paketmanagers

```
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install npm
sudo ln -s /usr/bin/nodejs /usr/bin/node

# the node & npm versions in apt are outdated. This is how you can update them:
sudo npm install -g npm
sudo npm install -g n
sudo n stable # (or lts, or a specific version)
```

Verwenden Sie die neueste Version einer bestimmten Version (z. B. LTS 6.x) direkt von Nodesource

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
apt-get install -y nodejs
```

Legen Sie für die richtige Installation von globalen npm-Modulen das persönliche Verzeichnis für sie fest (da keine Sudo und EACCES-Fehler erforderlich sind):

```
mkdir ~/.npm-global
echo "export PATH=~/.npm-global/bin:$PATH" >> ~/.profile
source ~/.profile
npm config set prefix '~/.npm-global'
```

Node.js unter Windows installieren

Standardinstallation

Alle Node.js-Binärdateien, Installationsprogramme und Quelldateien können hier heruntergeladen werden .

Sie können nur die `node.exe` Laufzeitumgebung herunterladen oder das Windows-Installationsprogramm (`.msi`) verwenden, mit dem auch `npm` (der empfohlene Paketmanager für Node.js) installiert und Pfade konfiguriert werden.

Installation durch Paketmanager

Sie können auch den Paketmanager *Chocolatey* (Software Management Automation) installieren.

```
# choco install nodejs.install
```

Weitere Informationen zur aktuellen Version finden Sie im choco-Repository hier .

Verwenden des Node Version Manager (nvm)

Node Version Manager , auch als nvm bezeichnet, ist ein Bash-Skript, das die Verwaltung mehrerer Node.js-Versionen vereinfacht.

Verwenden Sie zur Installation von nvm das mitgelieferte Installationsskript:

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

Für Windows gibt es ein nvm-windows-Paket mit einem Installationsprogramm. Diese GithHub-Seite enthält Details zur Installation und Verwendung des Pakets nvm-windows.

Führen Sie nach der Installation von nvm "nvm on" über die Befehlszeile aus. Dadurch kann nvm die Knotenversionen steuern.

Hinweis: Möglicherweise müssen Sie Ihr Terminal neu starten, um den neu installierten Befehl `nvm` zu erkennen.

Dann installieren Sie die neueste Node-Version:

```
$ nvm install node
```

Sie können auch eine bestimmte Node-Version installieren, indem Sie die Versionen Major, Minor und / oder Patch übergeben:

```
$ nvm install 6
$ nvm install 4.2
```

So listen Sie die für die Installation verfügbaren Versionen auf:

```
$ nvm ls-remote
```

Sie können dann die Version wechseln, indem Sie die Version auf dieselbe Weise übergeben, wie Sie dies bei der Installation tun:

```
$ nvm use 5
```

Sie können eine bestimmte Version von Node, die Sie installiert haben, als **Standardversion** festlegen, indem Sie Folgendes eingeben:

```
$ nvm alias default 4.2
```

Geben Sie Folgendes ein, um eine Liste der Knotenversionen anzuzeigen, die auf Ihrem Computer installiert sind:

```
$ nvm ls
```

Um projektspezifische Knotenversionen zu verwenden, können Sie die Version in der .nvmrc-Datei speichern. Auf diese Weise ist der Beginn der Arbeit mit einem anderen Projekt weniger fehleranfällig, nachdem das Projekt aus seinem Repository abgerufen wurde.

```
$ echo "4.2" > .nvmrc
$ nvm use
Found '/path/to/project/.nvmrc' with version <4.2>
Now using node v4.2 (npm v3.7.3)
```

Wenn Node über nvm installiert wird, müssen wir keine globalen Pakete mit `sudo` installieren, da diese im Home-Ordner installiert sind. Somit funktioniert der `npm i -g http-server` ohne Erlaubnisfehler.

Installieren Sie Node.js From Source mit dem APT-Paketmanager

Voraussetzungen

```
sudo apt-get install build-essential
sudo apt-get install python

[optional]
sudo apt-get install git
```

Holen Sie sich Quelle und bauen Sie

```
cd ~
git clone https://github.com/nodejs/node.git
```

ODER Für die neueste Version von LTS Node.js 6.10.2

```
cd ~
wget https://nodejs.org/dist/v6.3.0/node-v6.10.2.tar.gz
tar -xzvf node-v6.10.2.tar.gz
```

Wechseln Sie in das Quellverzeichnis wie in `cd ~/node-v6.10.2`

```
./configure
make
sudo make install
```

Installation von Node.js auf dem Mac mit dem Paketmanager

Homebrew

Sie können Node.js mit dem Homebrew- Paketmanager installieren.

Beginnen Sie mit dem Aktualisieren von brew:

```
brew update
```

Möglicherweise müssen Sie Berechtigungen oder Pfade ändern. Führen Sie das am besten aus, bevor Sie fortfahren:

```
brew doctor
```

Als Nächstes können Sie Node.js installieren, indem Sie Folgendes ausführen:

```
brew install node
```

Sobald Node.js installiert ist, können Sie die installierte Version überprüfen, indem Sie Folgendes ausführen:

```
node -v
```

Macports

Sie können node.js auch über Macports installieren.

Aktualisieren Sie es zunächst, um sicherzustellen, dass auf die neuesten Pakete verwiesen wird:

```
sudo port selfupdate
```

Dann installieren Sie nodejs und npm

```
sudo port install nodejs npm
```

Sie können den Knoten jetzt direkt über die CLI ausführen, indem Sie den `node` aufrufen. Sie können auch Ihre aktuelle Knotenversion mit überprüfen

```
node -v
```

Installation mit dem MacOS X Installer

Sie finden die Installationsprogramme auf der Download-Seite von Node.js. Normalerweise empfiehlt Node.js zwei Versionen von Node, die LTS-Version (Langzeitunterstützung) und die aktuelle Version (neueste Version). Wenn Sie mit Node noch nicht vertraut sind, wählen Sie

einfach das LTS und klicken Sie auf die Schaltfläche `Macintosh Installer` , um das Paket herunterzuladen.

Wenn Sie nach weiteren NodeJS-Releases suchen möchten, klicken Sie hier , wählen Sie Ihr Release aus und klicken Sie auf Download. `.pkg` Sie auf der Download-Seite nach einer Datei mit der Erweiterung `.pkg` .

Nachdem Sie die Datei heruntergeladen haben (mit der Erweiterung `.pkg` ofcourse), doppelklicken Sie darauf, um sie zu installieren. Das mit `Node.js` und `npm` gepackte Installationsprogramm installiert standardmäßig beide. Das Installationspaket kann jedoch durch Anklicken der Schaltfläche `customize` im Schritt `Installation Type customize` . Ansonsten folgen Sie einfach den Installationsanweisungen, es ist ziemlich einfach.

Überprüfen Sie, ob der Knoten installiert ist

Offenes `terminal` (wenn Sie nicht wissen, wie Sie Ihr Terminal öffnen, sehen Sie sich dieses Wikihow an). Im Terminaltyp `node --version` dann ein. Ihr Terminal sieht folgendermaßen aus, wenn Node installiert ist:

```
$ node --version
v7.2.1
```

Die Version `v7.2.1` ist Ihre Node.js-Version. Wenn Sie den Nachrichtenbefehl `command not found: node` statt dessen, bedeutet dies, dass ein Problem mit Ihrer Installation `v7.2.1` .

Node.js auf Raspberry PI installieren

Um v6.x zu installieren, aktualisieren Sie die Pakete

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
```

Verwenden des apt Paketmanagers

```
sudo apt-get install -y nodejs
```

Installation mit Node Version Manager unter Fish Shell mit Oh My Fish!

Knoten Version Manager (`nvm`) vereinfacht die Verwaltung von Node.js Versionen, deren Installation und beseitigt die Notwendigkeit für `sudo` , wenn sie mit Paketen zu tun (zB `npm install ...`). Fish Shell (`fish`) " *ist eine intelligente und benutzerfreundliche Befehlszeilen-Shell für OS X, Linux und den Rest der Familie* ", die bei Programmierern eine beliebte Alternative zu herkömmlichen Shells wie `bash` . Oh My Fish (`omf`) ermöglicht schließlich das Anpassen und Installieren von Paketen in Fish Shell.

In diesem Handbuch wird davon ausgegangen, dass Sie bereits Fish als Shell verwenden .

Installiere nvm

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.4/install.sh | bash
```

Installieren Sie Oh My Fish

```
curl -L https://github.com/oh-my-fish/oh-my-fish/raw/master/bin/install | fish
```

(Hinweis: Sie werden jetzt aufgefordert, Ihr Terminal neu zu starten. Fahren Sie jetzt fort.)

Installieren Sie Plugin-NVM für Oh My Fish

Wir installieren plugin-nvm über Oh My Fish, um die `nvm` Funktionen in der Fish-Shell `nvm` :

```
omf install nvm
```

Installieren Sie Node.js mit Node Version Manager

Sie können jetzt `nvm` . Sie können die Version von Node.js nach Ihren Wünschen installieren und verwenden. Einige Beispiele:

- Installieren Sie die neueste Node-Version: `nvm install node`
- Installieren Sie 6.3.1 speziell: `nvm install 6.3.1`
- Liste der installierten Versionen: `nvm ls`
- `nvm use 4.3.1` zu einem zuvor installierten 4.3.1: `nvm use 4.3.1`

Abschließende Anmerkungen

Denken Sie daran, dass wir `sudo` nicht mehr benötigen, wenn Sie Node.js mit dieser Methode verwenden. Knotenversionen, Pakete usw. werden in Ihrem Home-Verzeichnis installiert.

Installieren Sie Node.js vom Quellcode auf Centos, RHEL und Fedora

Voraussetzungen

- git
- `clang` und `clang++` 3.4 ^ oder `gcc` und `g++` 4.8 ^
- Python 2.6 oder 2.7
- GNU Make 3.81 ^

Quelle erhalten

Node.js v6.x LTS

```
git clone -b v6.x https://github.com/nodejs/node.git
```

Node.js v7.x

```
git clone -b v7.x https://github.com/nodejs/node.git
```

Bauen

```
cd node
./configure
make -jX
su -c make install
```

X - Die Anzahl der Prozessorkerne beschleunigt den Build erheblich

Aufräumen [Optional]

```
cd
rm -rf node
```

Node.js mit n installieren

Erstens gibt es einen wirklich schönen Wrapper zum Einrichten von `n` auf Ihrem System. Renn einfach:

```
curl -L https://git.io/n-install | bash
```

zu installieren `n` . Installieren Sie dann die Binärdateien auf verschiedene Arten:

neueste

```
n latest
```

stabil

```
n stable
```

lts

```
n lts
```

Jede andere Version

```
n <version>
```

zB `n 4.4.7`

Wenn diese Version bereits installiert ist, aktiviert dieser Befehl diese Version.

Versionswechsel

`n` selbst erzeugt eine Auswahlliste der installierten Binärdateien. Verwenden Sie hoch und runter, um den gewünschten zu finden und drücken Sie Enter, um ihn zu aktivieren.

Node.js installieren online lesen: https://riptutorial.com/de/node-js/topic/1294/node-js-installieren

Kapitel 66: Node.js Leistung

Examples

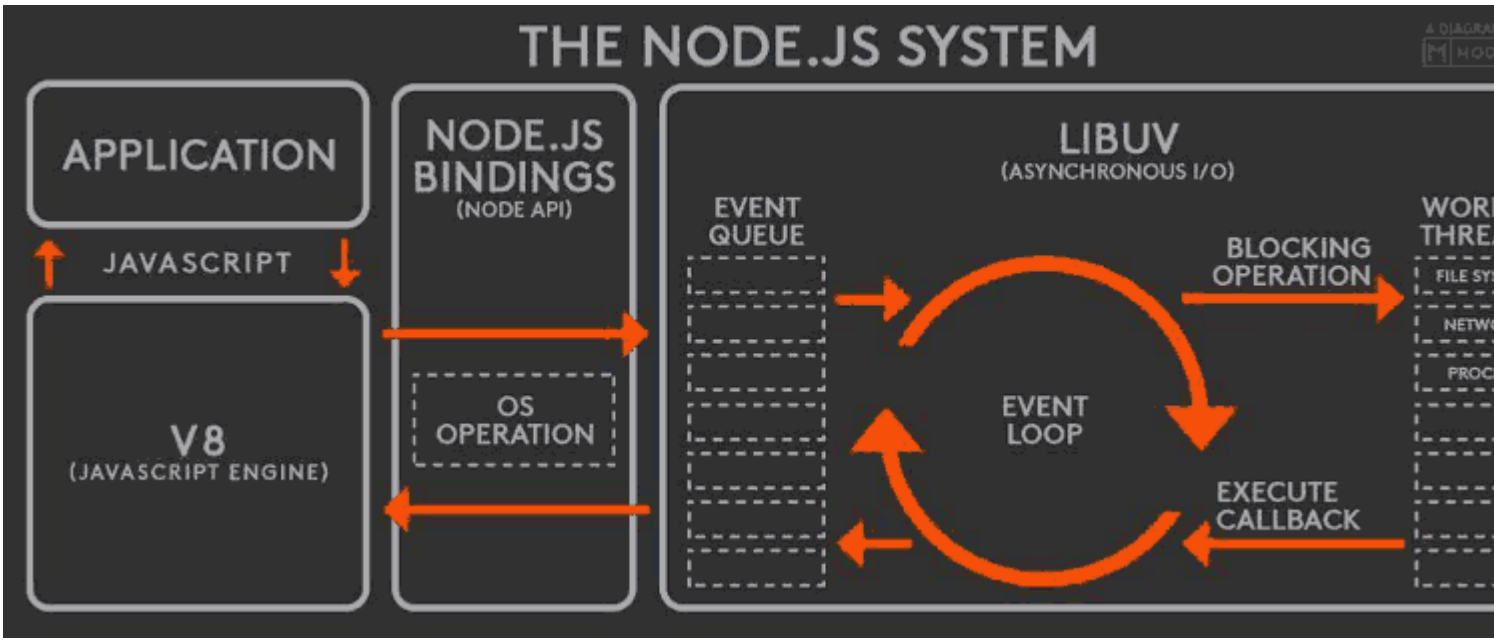
Ereignisschleife

Beispiel für das Blockieren

```
let loop = (i, max) => {
  while (i < max) i++
  return i
}

// This operation will block Node.js
// Because, it's CPU-bound
// You should be careful about this kind of code
loop(0, 1e+12)
```

Beispiel für nicht blockierende E / A-Operationen

```
let i = 0

const step = max => {
  while (i < max) i++
  console.log('i = %d', i)
}

const tick = max => process.nextTick(step, max)

// this will postpone tick run step's while-loop to event loop cycles
// any other IO-bound operation (like filesystem reading) can take place
// in parallel
tick(1e+6)
tick(1e+7)
console.log('this will output before all of tick operations. i = %d', i)
console.log('because tick operations will be postponed')
tick(1e+8)
```



Vereinfacht ausgedrückt handelt es sich bei Event Loop um einen Single-Threaded-Warteschlangenmechanismus, der Ihren CPU-gebundenen Code bis zum Ende seiner Ausführung und E / A-gebundenen Code auf nicht blockierende Weise ausführt.

In Node.js unter dem Teppich wird jedoch für einige Operationen der libuv Library Multithreading verwendet .

Überlegungen zur Leistung

- Nicht blockierende Vorgänge blockieren die Warteschlange nicht und wirken sich nicht auf die Leistung der Schleife aus.
- CPU-gebundene Vorgänge blockieren jedoch die Warteschlange. Sie sollten daher darauf achten, dass Sie keine CPU-gebundenen Vorgänge in Ihrem Node.js-Code ausführen.

Node.js blockiert die E / A nicht, da sie die Arbeit in den Betriebssystemkern verlagert. Wenn die E / A-Operation Daten (*als Ereignis*) bereitstellt, benachrichtigt sie Ihren Code mit Ihren bereitgestellten Rückrufen.

Erhöhen Sie maxSockets

Grundlagen

```
require('http').globalAgent.maxSockets = 25

// You can change 25 to Infinity or to a different value by experimenting
```

Node.js verwendet standardmäßig `maxSockets = Infinity` gleichzeitig (seit v0.12.0). Bis zum Knoten v0.12.0 war der Standardwert `maxSockets = 5` (siehe v0.11.0). Nach mehr als 5 Anforderungen werden sie in die Warteschlange gestellt. Wenn Sie Parallelität wünschen,

erhöhen Sie diese Anzahl.

Stellen Sie Ihren eigenen Agenten ein

`http` API verwendet einen **" Global Agent "** . Sie können Ihren eigenen Agenten zur Verfügung stellen. So was:

```
const http = require('http')
const myGloriousAgent = new http.Agent({ keepAlive: true })
myGloriousAgent.maxSockets = Infinity

http.request({ ..., agent: myGloriousAgent }, ...)
```

Socket Pooling vollständig ausschalten

```
const http = require('http')
const options = {.....}

options.agent = false

const request = http.request(options)
```

Fallstricke

- Sie sollten dasselbe für die `https` API tun, wenn Sie dieselben Effekte erzielen möchten
- Beachten Sie, dass AWS beispielsweise 50 anstelle von `Infinity` .

Gzip aktivieren

```
const http = require('http')
const fs   = require('fs')
const zlib = require('zlib')

http.createServer((request, response) => {
  const stream          = fs.createReadStream('index.html')
  const acceptsEncoding = request.headers['accept-encoding']

  let encoder = {
    hasEncoder     : false,
    contentEncoding: {},
    createEncoder  : () => throw 'There is no encoder'
  }

  if (!acceptsEncoding) {
    acceptsEncoding = ''
  }

  if (acceptsEncoding.match(/\bdeflate\b/)) {
```

```
    encoder = {
      hasEncoder     : true,
      contentEncoding: { 'content-encoding': 'deflate' },
      createEncoder  : zlib.createDeflate
    }
  } else if (acceptsEncoding.match(/\bgzip\b/)) {
    encoder = {
      hasEncoder     : true,
      contentEncoding: { 'content-encoding': 'gzip' },
      createEncoder  : zlib.createGzip
    }
  }

  response.writeHead(200, encoder.contentEncoding)

  if (encoder.hasEncoder) {
    stream = stream.pipe(encoder.createEncoder())
  }

  stream.pipe(response)

}).listen(1337)
```

Node.js Leistung online lesen: https://riptutorial.com/de/node-js/topic/9410/node-js-leistung

Kapitel 67: Node.js mit CORS

Examples

Aktivieren Sie CORS in express.js

Da node.js häufig zum Erstellen von APIs verwendet wird, kann die richtige Einstellung von CORS lebensrettend sein, wenn Sie die API von verschiedenen Domänen anfordern möchten.

Im Beispiel richten wir es für die umfassendere Konfiguration ein (autorisieren Sie alle Anforderungstypen von jeder Domäne aus).

In Ihrer server.js nach dem Initialisieren von express:

```
// Create express server
const app = express();

app.use((req, res, next) => {
    res.header('Access-Control-Allow-Origin', '*');

    // authorized headers for preflight requests
    // https://developer.mozilla.org/en-US/docs/Glossary/preflight_request
    res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type,
Accept');
    next();

    app.options('*', (req, res) => {
        // allowed XHR methods
        res.header('Access-Control-Allow-Methods', 'GET, PATCH, PUT, POST, DELETE, OPTIONS');
        res.send();
    });
});
```

Normalerweise wird der Knoten auf Produktionsservern hinter einem Proxy ausgeführt. Daher ist der Reverse-Proxy-Server (wie Apache oder Nginx) für die CORS-Konfiguration verantwortlich.

Um dieses Szenario bequem anzupassen, ist es möglich, node.js CORS nur während der Entwicklung zu aktivieren.

Dies kann leicht durch Überprüfen von `NODE_ENV` :

```
const app = express();

if (process.env.NODE_ENV === 'development') {
    // CORS settings
}
```

Node.js mit CORS online lesen: https://riptutorial.com/de/node-js/topic/9272/node-js-mit-cors

Kapitel 68: Node.JS mit ES6

Einführung

ES6, ECMAScript 6 oder ES2015 ist die neueste Spezifikation für JavaScript, mit der der Sprache etwas syntaktischer Zucker hinzugefügt wird. Es ist ein großes Update für die Sprache und führt viele neue Funktionen ein

Weitere Informationen zu Node und ES6 finden Sie auf der Website https://nodejs.org/de/docs/es6/.

Examples

Node ES6 Support und Erstellung eines Projekts mit Babel

Die gesamte ES6-Spezifikation ist noch nicht vollständig implementiert, sodass Sie nur einige der neuen Funktionen nutzen können. Eine Liste der aktuell unterstützten ES6-Funktionen finden Sie unter http://node.green/

Seit NodeJS v6 gibt es ziemlich gute Unterstützung. Wenn Sie also NodeJS v6 oder höher verwenden, können Sie ES6 genießen. Möglicherweise möchten Sie jedoch auch einige der nicht veröffentlichten Funktionen und einige darüber hinaus verwenden. Dafür benötigen Sie einen Transpiler

Es ist möglich, einen Transpiler zur Laufzeit auszuführen und zu bauen, alle ES6-Funktionen und mehr zu verwenden. Der beliebteste Transpiler für JavaScript heißt Babel

Babel können Sie alle Funktionen von der ES6 Spezifikation und einige zusätzliche nicht-in-spec Features mit ‚Stufe-0' verwenden , wie `import thing from 'thing` statt `var thing = require('thing')`

Wenn wir ein Projekt erstellen wollten, in dem wir "stage-0" -Funktionen wie Import verwenden, müssten wir Babel als Transpiler hinzufügen. In Projekten, in denen React und Vue und andere CommonJS-basierte Muster verwendet werden, wird häufig die Stufe 0 implementiert.

Erstellen Sie ein neues Knotenprojekt

```
mkdir my-es6-app
cd my-es6-app
npm init
```

Installieren Sie das ES6-Preset und die Stufe 0

```
npm install --save-dev babel-preset-es2015 babel-preset-stage-2 babel-cli babel-register
```

Erstellen Sie eine neue Datei mit dem Namen `server.js` und fügen Sie einen einfachen HTTP-Server hinzu.

```
import http from 'http'

http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'})
  res.end('Hello World\n')
}).listen(3000, '127.0.0.1')

console.log('Server running at http://127.0.0.1:3000/')
```

Beachten Sie, dass wir einen `import http from 'http'` Dies ist eine Stage-0-Funktion. Wenn dies funktioniert, bedeutet dies, dass der Transpiler ordnungsgemäß funktioniert.

Wenn Sie `node server.js` kann der Import nicht durchgeführt werden.

Erstellen Sie eine .babelrc-Datei im Stammverzeichnis Ihres Verzeichnisses, und fügen Sie die folgenden Einstellungen hinzu

```
{
  "presets": ["es2015", "stage-2"],
  "plugins": []
}
```

Sie können den Server jetzt mit dem `node src/index.js --exec babel-node`

Es ist keine gute Idee, einen Transpiler zur Laufzeit in einer Produktions-App auszuführen. Wir können jedoch einige Skripte in package.json implementieren, um die Arbeit mit ihnen zu erleichtern.

```
"scripts": {
    "start": "node dist/index.js",
    "dev": "babel-node src/index.js",
    "build": "babel src -d dist",
    "postinstall": "npm run build"
  },
```

Das obige wird bei `npm install` den transpiled-Code in das dist-Verzeichnis erstellen, damit `npm start` den transpiled-Code für unsere Produktions-App verwenden kann.

`npm run dev` bootet die Server- und Babel-Laufzeitumgebung. Dies ist in Ordnung und wird bevorzugt, wenn Sie lokal an einem Projekt arbeiten.

Wenn Sie noch einen Schritt weiter gehen, können Sie nodemon `npm install nodemon --save-dev` , um nach Änderungen zu `npm install nodemon --save-dev` die Knoten-App neu.

Dies beschleunigt die Arbeit mit Babel und NodeJS. Aktualisieren Sie in Ihrer package.json das Skript "dev", um nodemon zu verwenden

```
"dev": "nodemon src/index.js --exec babel-node",
```

Verwenden Sie JS es6 in Ihrer NodeJS-App

JS es6 (auch als es2015 bekannt) umfasst eine Reihe neuer Funktionen für JS, die darauf

abzielen, es intuitiver zu machen, wenn Sie OOP verwenden oder sich modernen Entwicklungsaufgaben stellen.

Voraussetzungen:

1. Schauen Sie sich die neuen es6-Funktionen unter http://es6-features.org an - es kann Ihnen klarstellen, ob Sie es wirklich in Ihrer nächsten NodeJS-App verwenden möchten
2. Überprüfen Sie den Kompatibilitätsgrad Ihrer Knotenversion unter http://node.green
3. Wenn alles in Ordnung ist - lass uns einlernen!

Hier ist ein sehr kurzes Beispiel einer einfachen `hello world` App mit JS es6

```
'use strict'

class Program
{
    constructor()
    {
        this.message = 'hello es6 :)';
    }

    print()
    {
        setTimeout(() =>
        {
            console.log(this.message);

            this.print();

        }, Math.random() * 1000);
    }
}

new Program().print();
```

Sie können dieses Programm ausführen und beobachten, wie dieselbe Nachricht immer und immer wieder gedruckt wird.

Jetzt .. lassen Sie es Zeile für Zeile aufteilen:

```
'use strict'
```

Diese Zeile ist tatsächlich erforderlich, wenn Sie js es6 verwenden möchten. `strict` Modus weist absichtlich eine andere Semantik als der normale Code auf (bitte lesen Sie mehr dazu im MDN - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode)

```
class Program
```

Unglaublich - ein `class` ! Nur zur schnellen Referenz - vor es6 bestand die einzige Möglichkeit,

eine Klasse in js zu definieren, mit dem Schlüsselwort ... `function` !

```
function MyClass() // class definition
{

}

var myClassObject = new MyClass(); // generating a new object with a type of MyClass
```

Bei Verwendung von OOP ist eine Klasse eine sehr grundlegende Fähigkeit, die den Entwickler dabei unterstützt, einen bestimmten Teil eines Systems darzustellen (Code zu zerlegen ist wichtig, wenn der Code größer wird. Beispiel: beim Schreiben von serverseitigem Code)

```
constructor()
{
    this.message = 'hello es6 :)';
}
```

Sie müssen zugeben - das ist ziemlich intuitiv! Dies ist der C'tor meiner Klasse - diese einzigartige "Funktion" wird jedes Mal ausgeführt, wenn ein Objekt aus dieser bestimmten Klasse erstellt wird (in unserem Programm - nur einmal).

```
print()
{
    setTimeout(() => // this is an 'arrow' function
    {
        console.log(this.message);

        this.print(); // here we call the 'print' method from the class template itself (a
recursion in this particular case)

    }, Math.random() * 1000);
}
```

Weil print im Klassenbereich definiert ist - es ist eigentlich eine Methode - die entweder vom Objekt der Klasse oder von innerhalb der Klasse selbst aufgerufen werden kann!

Bis jetzt haben wir unsere Klasse definiert .. Zeit, um sie zu benutzen:

```
new Program().print();
```

Welches ist wirklich gleich:

```
var prog = new Program(); // define a new object of type 'Program'

prog.print(); // use the program to print itself
```

Zusammenfassend: JS es6 kann Ihren Code vereinfachen - intuitiver und verständlicher machen (im Vergleich zur vorherigen Version von JS). Sie können versuchen, einen vorhandenen Code neu zu schreiben und den Unterschied zu erkennen

GENIESSEN :)

Node.JS mit ES6 online lesen: https://riptutorial.com/de/node-js/topic/5934/node-js-mit-es6

Kapitel 69: Node.js mit Oracle

Examples

Stellen Sie eine Verbindung zu Oracle DB her

Eine sehr einfache Möglichkeit, eine Verbindung zu einer ORACLE-Datenbank `oracledb` ist das `oracledb` Modul. Dieses Modul übernimmt die Verbindung zwischen Ihrer Node.js-App und dem Oracle-Server. Sie können es wie jedes andere Modul installieren:

```
npm install oracledb
```

Nun müssen Sie eine ORACLE-Verbindung erstellen, die Sie später abfragen können.

```
const oracledb = require('oracledb');

oracledb.getConnection(
  {
    user          : "oli",
    password      : "password",
    connectString : "ORACLE_DEV_DB_TNS_NAME"
  },
  connExecute
);
```

Der connectString "ORACLE_DEV_DB_TNA_NAME" kann in einer tnsnames.org-Datei im selben Verzeichnis oder in dem Ihr Oracle-Instant-Client installiert ist.

Wenn Sie keinen Oracle-Instant-Client auf Ihrem Entwicklungscomputer installiert haben, können Sie der `instant client installation guide` für Ihr Betriebssystem folgen.

Fragen Sie ein Verbindungsobjekt ohne Parameter ab

Use kann jetzt die connExecute-Funktion zum Ausführen einer Abfrage verwenden. Sie haben die Möglichkeit, das Abfrageergebnis als Objekt oder Array abzurufen. Das Ergebnis wird in console.log gedruckt.

```
function connExecute(err, connection)
{
    if (err) {
        console.error(err.message);
        return;
    }
    sql = "select 'test' as c1, 'oracle' as c2 from dual";
    connection.execute(sql, {}, { outFormat: oracledb.OBJECT }, // or oracledb.ARRAY
        function(err, result)
        {
            if (err) {
                console.error(err.message);
                connRelease(connection);
```

```
                return;
            }
            console.log(result.metaData);
            console.log(result.rows);
            connRelease(connection);
        });
}
```

Da wir eine Verbindung ohne Pooling verwendet haben, müssen wir die Verbindung wieder freigeben.

```
function connRelease(connection)
{
  connection.close(
    function(err) {
      if (err) {
        console.error(err.message);
      }
    });
}
```

Die Ausgabe für ein Objekt wird sein

```
[ { name: 'C1' }, { name: 'C2' } ]
[ { C1: 'test', C2: 'oracle' } ]
```

und die Ausgabe für ein Array wird sein

```
[ { name: 'C1' }, { name: 'C2' } ]
[ [ 'test', 'oracle' ] ]
```

Verwendung eines lokalen Moduls zur einfacheren Abfrage

Um das Abfragen von ORACLE-DB zu vereinfachen, können Sie Ihre Abfrage folgendermaßen aufrufen:

```
const oracle = require('./oracle.js');

const sql = "select 'test' as c1, 'oracle' as c2 from dual";
oracle.queryObject(sql, {}, {})
    .then(function(result) {
        console.log(result.rows[0]['C2']);
    })
    .catch(function(err) {
        next(err);
    });
```

Der Aufbau der Verbindung und deren Ausführung ist in dieser Datei oracle.js mit folgendem Inhalt enthalten:

```
'use strict';
const oracledb = require('oracledb');
```

```
const oracleDbRelease = function(conn) {
  conn.release(function (err) {
    if (err)
      console.log(err.message);
  });
};

function queryArray(sql, bindParams, options) {
    options.isAutoCommit = false; // we only do SELECTs

    return new Promise(function(resolve, reject) {
        oracledb.getConnection(
                  {
                    user          : "oli",
                    password      : "password",
                    connectString : "ORACLE_DEV_DB_TNA_NAME"
                })
        .then(function(connection){
            //console.log("sql log: " + sql + " params " + bindParams);
            connection.execute(sql, bindParams, options)
            .then(function(results) {
                resolve(results);
                process.nextTick(function() {
                    oracleDbRelease(connection);
                });
            })
            .catch(function(err) {
                reject(err);

                process.nextTick(function() {
                    oracleDbRelease(connection);
                        });
                    });
            })
            .catch(function(err) {
                reject(err);
            });
    });
}

function queryObject(sql, bindParams, options) {
    options['outFormat'] = oracledb.OBJECT; // default is oracledb.ARRAY
    return queryArray(sql, bindParams, options);
}

module.exports = queryArray;
module.exports.queryArray = queryArray;
module.exports.queryObject = queryObject;
```

Beachten Sie, dass Sie die beiden Methoden queryArray und queryObject für Ihr Oracle-Objekt aufrufen können.

Node.js mit Oracle online lesen: https://riptutorial.com/de/node-js/topic/8248/node-js-mit-oracle

Kapitel 70: Node.JS und MongoDB.

Bemerkungen

Dies sind die grundlegenden CRUD-Operationen für die Verwendung der Mongo-Datenbank mit Nodejs.

Frage: Gibt es andere Möglichkeiten, was Sie hier tun können?

Antwort: Ja, dafür gibt es zahlreiche Möglichkeiten.

Frage: Ist die Verwendung von Mungo notwendig?

Antwort: Nein. Es gibt andere Pakete, die Ihnen helfen können.

Frage: Wo bekomme ich eine vollständige Dokumentation von Mungo?

Antwort: Hier klicken

Examples

Verbindung zu einer Datenbank herstellen

Um eine Verbindung zu einer Mongo-Datenbank von der Knotenanwendung herzustellen, benötigen wir Mongoose.

Installation von Mongoose Gehen Sie zum Toot Ihrer Anwendung und installieren Sie Mongoose von

```
npm install mongoose
```

Als Nächstes verbinden wir uns mit der Datenbank.

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

//connect to the test database running on default mongod port of localhost
mongoose.connect('mongodb://localhost/test');



//Connecting with custom credentials
mongoose.connect('mongodb://USER:PASSWORD@HOST:PORT/DATABASE');


//Using Pool Size to define the number of connections opening
//Also you can use a call back function for error handling
mongoose.connect('mongodb://localhost:27017/consumers',
                 {server: { poolSize: 50 }},
                 function(err) {
                    if(err) {
```

```
                        console.log('error in this')
                        console.log(err);
                        // Do whatever to handle the error
                    } else {
                        console.log('Connected to the database');
                    }
                });
```

Neue Kollektion erstellen

Bei Mongoose wird alles von einem Schema abgeleitet. Lass uns ein Schema erstellen.

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

var Schema = mongoose.Schema;

var AutoSchema = new Schema({
    name : String,
    countOf: Number,
});
// defining the document structure

// by default the collection created in the db would be the first parameter we use (or the
plural of it)
module.exports = mongoose.model('Auto', AutoSchema);

// we can over write it and define the collection name by specifying that in the third
parameters.
module.exports = mongoose.model('Auto', AutoSchema, 'collectionName');


// We can also define methods in the models.
AutoSchema.methods.speak = function () {
  var greeting = this.name
    ? "Hello this is  " + this.name+ " and I have counts of "+ this.countOf
    : "I don't have a name";
  console.log(greeting);
}
mongoose.model('Auto', AutoSchema, 'collectionName');
```

Denken Sie daran, dass Methoden zum Schema hinzugefügt werden müssen, bevor Sie es wie oben beschrieben mit mongoose.model () kompilieren.

Dokumente einfügen

Um ein neues Dokument in die Sammlung einzufügen, erstellen wir ein Objekt des Schemas.

```
var Auto = require('models/auto')
var autoObj = new Auto({
    name: "NewName",
    countOf: 10
});
```

Wir speichern es wie folgt

```
autoObj.save(function(err, insertedAuto) {
    if (err) return console.error(err);
    insertedAuto.speak();
    // output: Hello this is NewName and I have counts of 10
});
```

Dadurch wird ein neues Dokument in die Sammlung eingefügt

lesen

Das Lesen von Daten aus der Sammlung ist sehr einfach. Alle Daten der Sammlung abrufen.

```
var Auto = require('models/auto')
Auto.find({}, function (err, autos) {
      if (err) return console.error(err);
       // will return a json array of all the documents in the collection
      console.log(autos);
})
```

Daten mit einer Bedingung lesen

```
Auto.find({countOf: {$gte: 5}}, function (err, autos) {
      if (err) return console.error(err);
       // will return a json array of all the documents in the collection whose count is
greater than 5
      console.log(autos);
})
```

Sie können auch den zweiten Parameter als Objekt aller Felder angeben, die Sie benötigen

```
Auto.find({},{name:1}, function (err, autos) {
      if (err) return console.error(err);
       // will return a json array of name field of all the documents in the collection
      console.log(autos);
})
```

Ein Dokument in einer Sammlung suchen.

```
Auto.findOne({name:"newName"}, function (err, auto) {
      if (err) return console.error(err);
     //will return the first object of the document whose name is "newName"
      console.log(auto);
})
```

Ein Dokument in einer Sammlung nach ID suchen.

```
Auto.findById(123, function (err, auto) {
      if (err) return console.error(err);
     //will return the first json object of the document whose id is 123
      console.log(auto);
})
```

Aktualisierung

Zur Aktualisierung von Sammlungen und Dokumenten können wir eine der folgenden Methoden verwenden:

Methoden

- aktualisieren()
- updateOne ()
- updateMany ()
- replaceOne ()

Aktualisieren()

Die update () -Methode ändert ein oder mehrere Dokumente (Aktualisierungsparameter).

```
db.lights.update(
   { room: "Bedroom" },
   { status: "On" }
)
```

Bei diesem Vorgang wird in der Sammlung "Lichter" nach einem Dokument **gesucht,** in dem der `room` **Schlafzimmer ist** *(erster Parameter)* . Es aktualisiert dann die passenden Dokumente `status` - Eigenschaft auf **On** *(2.e Parameter)* und gibt ein Write Objekt , das wie folgt aussieht:

```
{ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }
```

UpdateOne

Die UpdateOne () -Methode ändert EIN Dokument (Aktualisierungsparameter).

```
db.countries.update(
   { country: "Sweden" },
   { capital: "Stockholm" }
)
```

Bei diesem Vorgang wird die Sammlung "Länder" nach einem Dokument durchsucht, in dem das `country` **Schweden ist** *(1. Parameter)* . Es aktualisiert dann die passenden Dokumente Eigentum `capital` **Stockholm** *(2.e Parameter)* und gibt ein Write Objekt , das wie folgt aussieht:

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

UpdateMany

Die UpdateMany () -Methode ändert mehrere Dokumente (Aktualisierungsparameter).

```
db.food.updateMany(
   { sold: { $lt: 10 } },
   { $set: { sold: 55 } }
)
```

Dieser Vorgang aktualisiert alle Dokumente *(in einer ‚Lebensmittel' Sammlung)* , wo `sold` ist **weniger als 10** * (1. Parameter) durch Einstellung `sold` bis **55.** Dann wird ein WriteResult-Objekt zurückgegeben, das wie folgt aussieht:

```
{ "acknowledged" : true, "matchedCount" : a, "modifiedCount" : b }
```

a = Anzahl übereinstimmender Dokumente
b = Anzahl der modifizierten Dokumente

Ersetzen Sie eine

Ersetzt das erste übereinstimmende Dokument (Ersatzdokument)

Diese Beispielsammlung mit dem Namen **Länder** enthält 3 Dokumente:

```
{ "_id" : 1, "country" : "Sweden" }
{ "_id" : 2, "country" : "Norway" }
{ "_id" : 3, "country" : "Spain" }
```

Die folgende Operation ersetzt das Dokument `{ country: "Spain" }` durch Dokument `{ country: "Finland" }`

```
db.countries.replaceOne(
   { country: "Spain" },
   { country: "Finland" }
)
```

Und kehrt zurück:

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Die Beispielsammlung **Länder** jetzt enthält:

```
{ "_id" : 1, "country" : "Sweden" }
{ "_id" : 2, "country" : "Norway" }
{ "_id" : 3, "country" : "Finland" }
```

Löschen

Das Löschen von Dokumenten aus einer Sammlung in Moose erfolgt auf folgende Weise.

```
Auto.remove({_id:123}, function(err, result){
    if (err) return console.error(err);
    console.log(result); // this will specify the mongo default delete result.
});
```

Node.JS und MongoDB. online lesen: https://riptutorial.com/de/node-js/topic/7505/node-js-und-mongodb-

Kapitel 71: Node.js v6 Neue Funktionen und Verbesserungen

Einführung

Der Knoten 6 wird zur neuen LTS-Version des Knotens. Durch die Einführung der neuen ES6-Standards können wir eine Reihe von Verbesserungen der Sprache feststellen. Wir werden einige der neuen Funktionen und Beispiele für deren Implementierung durchgehen.

Examples

Standardfunktionsparameter

```
function addTwo(a, b = 2) {
    return a + b;
}

addTwo(3) // Returns the result 5
```

Mit dem Hinzufügen von Standardfunktionsparametern können Sie nun Argumente optional machen und sie standardmäßig auf einen Wert Ihrer Wahl setzen.

Restparameter

```
function argumentLength(...args) {
    return args.length;
}

argumentLength(5) // returns 1
argumentLength(5, 3) //returns 2
argumentLength(5, 3, 6) //returns 3
```

Wenn Sie das letzte Argument Ihrer Funktion mit ... voranstellen, werden alle an die Funktion übergebenen Argumente als Array gelesen. In diesem Beispiel übergeben wir mehrere Argumente und erhalten die Länge des Arrays, das aus diesen Argumenten erstellt wird.

Spread Operator

```
function myFunction(x, y, z) { }
var args = [0, 1, 2];
myFunction(...args);
```

Die Spread-Syntax ermöglicht die Erweiterung eines Ausdrucks an Stellen, an denen mehrere Argumente (für Funktionsaufrufe) oder mehrere Elemente (für Array-Literale) oder mehrere Variablen erwartet werden. Genau wie die Rest-Parameter geben Sie Ihrem Array einfach ein ...

Pfeilfunktionen

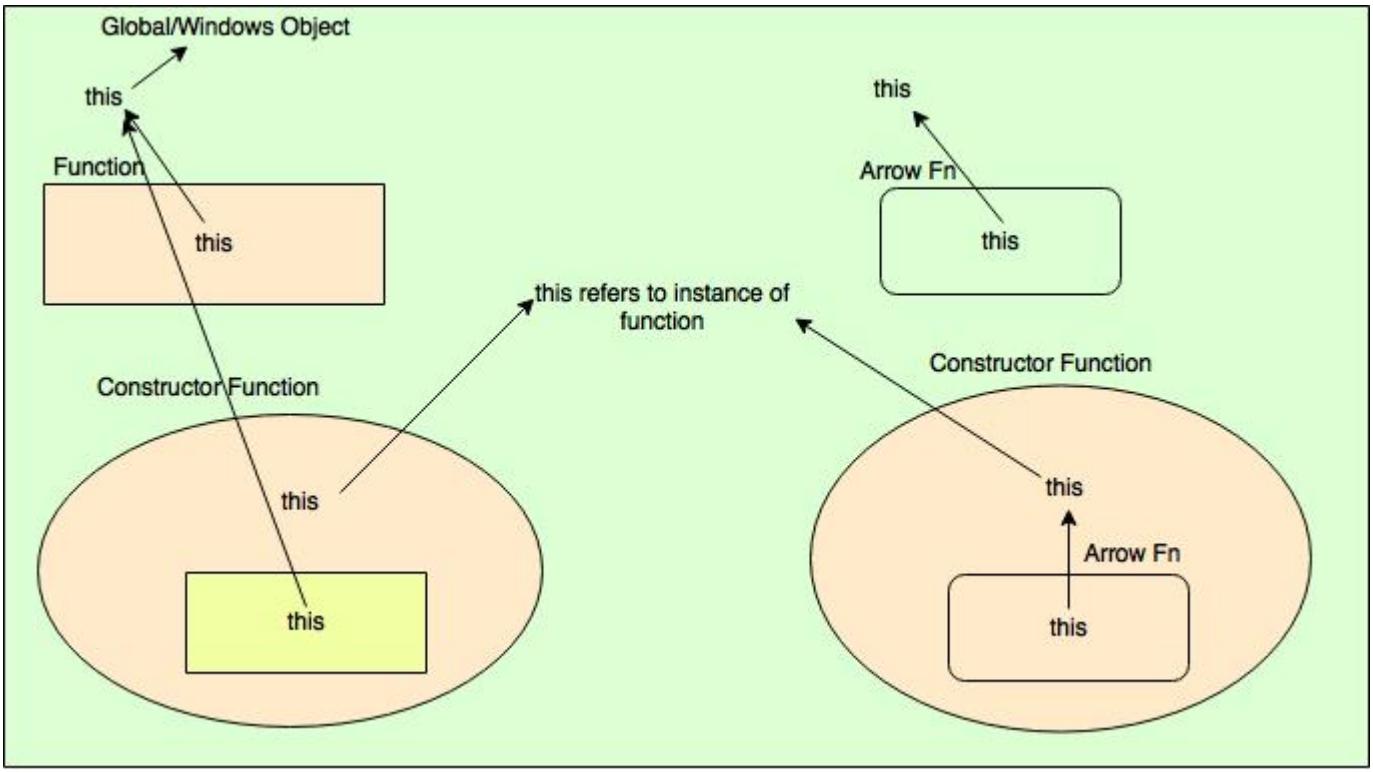
Die Pfeilfunktion ist die neue Art, eine Funktion in ECMAScript 6 zu definieren.

```
// traditional way of declaring and defining function
var sum = function(a,b)
{
    return a+b;
}

// Arrow Function
let sum = (a, b)=> a+b;

//Function defination using multiple lines
let checkIfEven = (a) => {
    if( a % 2 == 0 )
        return true;
    else
        return false;
}
```

"this" in der Pfeilfunktion

Diese Funktion bezieht sich auf die Instanz - Objekt verwendet, um diese Funktion aufzurufen , aber ***diese*** in Pfeil Funktion ist gleich *dieser* der Funktion , in der Pfeil Funktion definiert ist.

Verstehen wir mit Diagramm



Verständnis anhand von Beispielen.

```
var normalFn = function(){
   console.log(this) // refers to global/window object.
```

```
}

var arrowFn = () => console.log(this); // refers to window or global object as function is
defined in scope of global/window object

var service = {

    constructorFn : function(){

        console.log(this); //  refers to service as service object used to call method.

        var nestedFn = function(){
            console.log(this); // refers window or global object because no instance object
was used to call this method.
        }
        nestedFn();
    },

    arrowFn : function(){
        console.log(this); // refers to service as service object was used to call method.
        let fn = () => console.log(this); // refers to service object as arrow function
defined in function which is called using instance object.
        fn();
    }
}

// calling defined functions
constructorFn();
arrowFn();
service.constructorFn();
service.arrowFn();
```

In der Pfeilfunktion ist *dies der* lexikalische Gültigkeitsbereich. *Dies* ist der Funktionsumfang, in dem die Pfeilfunktion definiert ist.
Das erste Beispiel ist die traditionelle Art und Weise Funktionen zu definieren und somit bezieht sich *dies* auf *global / Fenster* - Objekt.
Im zweiten Beispiel wird *diese* Funktion innerhalb der Pfeilfunktion verwendet. *Dies* bezieht sich auf den Gültigkeitsbereich, in dem sie definiert ist (dh Fenster oder globales Objekt). Im dritten Beispiel ist *dies* ein Serviceobjekt, da die Funktion mit dem Serviceobjekt aufgerufen wird.
Im vierten Beispiel wurde die Funktion arrow in der Funktion definiert und aufgerufen, deren Gültigkeitsbereich *service* ist. Daher wird das *Service*- Objekt gedruckt.

Hinweis: - Das globale Objekt wird in Node.Js und Windows-Objekten im Browser gedruckt.

Node.js v6 Neue Funktionen und Verbesserungen online lesen: https://riptutorial.com/de/node-js/topic/8593/node-js-v6-neue-funktionen-und-verbesserungen

Kapitel 72: Node.js-Code für STDIN und STDOUT, ohne eine Bibliothek zu verwenden

Einführung

Dies ist ein einfaches Programm in node.js, in das der Benutzer Eingaben vornimmt und an die Konsole druckt.

Das **Prozessobjekt** ist ein globales Objekt, das Informationen zum aktuellen Node.js-Prozess liefert und diese steuert. Als globales Element steht es Node.js-Anwendungen immer zur Verfügung, ohne „request ()“ zu verwenden.

Examples

Programm

Die **process.stdin-** Eigenschaft gibt einen lesbaren Stream zurück, der stdin entspricht oder diesem zugeordnet ist.

Die **process.stdout -Eigenschaft** gibt einen beschreibbaren Stream zurück, der stdout entspricht oder diesem zugeordnet ist.

```
process.stdin.resume()
console.log('Enter the data to be displayed ');
process.stdin.on('data', function(data) { process.stdout.write(data) })
```

Node.js-Code für STDIN und STDOUT, ohne eine Bibliothek zu verwenden online lesen: https://riptutorial.com/de/node-js/topic/8961/node-js-code-fur-stdin-und-stdout--ohne-eine-bibliothek-zu-verwenden

Kapitel 73: Node.js-Fehlerverwaltung

Einführung

Wir werden lernen, Fehlerobjekte zu erstellen und Fehler in Node.js zu werfen und zu behandeln

Zukünftige Änderungen bezogen sich auf bewährte Vorgehensweisen bei der Fehlerbehandlung.

Examples

Fehlerobjekt erstellen

neuer Fehler (Nachricht)

Erstellt ein neues Fehlerobjekt, bei dem der Wert `message` auf `message` property des erstellten Objekts festgelegt wird. Normalerweise werden die `message` als String an den Error-Konstruktor übergeben. Wenn das `message` jedoch object und kein String ist, ruft der Error-Konstruktor die `.toString()` -Methode des übergebenen Objekts auf und setzt diesen Wert auf `message` Eigenschaft des erstellten `.toString()` .

```
var err = new Error("The error message");
console.log(err.message); //prints: The error message
console.log(err);
//output
//Error: The error message
//    at ...
```

Jedes Fehlerobjekt verfügt über eine Stapelablaufverfolgung. Der Stack-Trace enthält die Informationen zur Fehlermeldung und zeigt an, wo der Fehler aufgetreten ist (die obige Ausgabe zeigt den Fehler-Stack). Sobald das Fehlerobjekt erstellt wurde, erfasst das System den Stack-Trace des Fehlers in der aktuellen Zeile. Um den Stack-Trace abzurufen, verwenden Sie die stack -Eigenschaft aller erstellten Fehlerobjekte. Unten sind zwei Zeilen identisch:

```
console.log(err);
console.log(err.stack);
```

Fehler werfen

Auslöser Fehler bedeutet Ausnahme Wenn keine Ausnahme behandelt wird, stürzt der Knotenserver ab.

Die folgende Zeile wirft einen Fehler:

```
throw new Error("Some error occurred");
```

oder

```
var err = new Error("Some error occurred");
throw err;
```

oder

```
throw "Some error occurred";
```

Das letzte Beispiel (das Auslösen von Strings) ist nicht empfehlenswert und wird nicht empfohlen (werfen Sie immer Fehler, die Instanzen eines Fehlerobjekts sind).

Beachten Sie `throw` dass das System in dieser Zeile abstürzt, wenn Sie einen Fehler auslösen (wenn keine Ausnahmebehandlungsroutinen vorhanden sind). Danach wird kein Code ausgeführt.

```
var a = 5;
var err = new Error("Some error message");
throw err; //this will print the error stack and node server will stop
a++; //this line will never be executed
console.log(a); //and this one also
```

Aber in diesem Beispiel:

```
var a = 5;
var err = new Error("Some error message");
console.log(err); //this will print the error stack
a++;
console.log(a); //this line will be executed and will print 6
```

versuchen Sie ... catch block

try ... catch-Block ist für die Behandlung von Ausnahmen gedacht. Ausnahmebedingungen bedeuten, dass der ausgelöste Fehler nicht der Fehler ist.

```
try {
    var a = 1;
    b++; //this will cause an error because be is undefined
    console.log(b); //this line will not be executed
} catch (error) {
    console.log(error); //here we handle the error caused in the try block
}
```

Im `try` Block verursachen `b++` einen Fehler und dieser Fehler wird an `catch` Block übergeben, der dort gehandhabt werden kann oder sogar den gleichen Fehler im catch-Block werfen kann oder eine kleine Bitänderung vornehmen und dann werfen kann. Schauen wir uns das nächste Beispiel an.

```
try {
    var a = 1;
    b++;
    console.log(b);
} catch (error) {
    error.message = "b variable is undefined, so the undefined can't be incremented"
```

```
    throw error;
}
```

Im obigen Beispiel haben wir die `message` des `error` geändert und den geänderten `error` .

Sie können jeden Fehler in Ihrem try-Block durchgehen und im catch-Block behandeln:

```
try {
    var a = 1;
    throw new Error("Some error message");
    console.log(a); //this line will not be executed;
} catch (error) {
    console.log(error); //will be the above thrown error
}
```

Node.js-Fehlerverwaltung online lesen: https://riptutorial.com/de/node-js/topic/8590/node-js-fehlerverwaltung

Kapitel 74: Nodejs Geschichte

Einführung

Hier werden wir über die Geschichte von Node.js, Versionsinformationen und deren aktuellen Status diskutieren.

Examples

Wichtige Ereignisse in jedem Jahr

2009

- 3. März: Das Projekt wurde als "Knoten" bezeichnet
- 1. Oktober: Erste sehr frühe Vorschau von npm, dem Node-Paket Manager
- 8. November: Ryan Dahls (Schöpfer von Node.js) Original Node.js Talk auf der JSConf 2009

2010

- Express: Ein Node.js-Webentwicklungs-Framework
- Erstausgabe von Socket.io
- 28. April: Experimentelle Node.js-Unterstützung bei Heroku
- 28. Juli: Ryan Dahls Google Tech Talk auf Node.js
- 20. August: Node.js 0.2.0 veröffentlicht

2011

- 31. März: Node.js Guide
- 1. Mai: npm 1.0: veröffentlicht
- 1. Mai: Ryan Dahls AMA auf Reddit
- 10. Juli: Das Node-Beginner-Buch, eine Einführung in Node.js, ist abgeschlossen .
    - Ein umfassendes Node.js-Tutorial für Anfänger.
- 16. August: LinkedIn verwendet Node.js
    - LinkedIn hat seine komplett überarbeitete mobile App mit neuen Funktionen und neuen Teilen unter der Haube eingeführt.
- 5. Oktober: Ryan Dahl spricht über die Geschichte von Node.js und warum er sie erstellt hat
- 5. Dezember: Node.js in Produktion bei Uber
    - Curtis Chambers, Uber-Engineering-Manager, erklärt, warum sein Unternehmen seine Anwendung mit Node.js komplett überarbeitet hat, um die Effizienz zu steigern und die Partner- und Kundenerfahrung zu verbessern.

2012

- 30. Januar: Der Schöpfer von Node.j, Ryan Dahl, tritt vom täglichen Alltag von Node zurück
- 25. Juni: Node.js v0.8.0 [stable] ist raus
- 20. Dezember: Hapi, ein Node.js-Framework wird veröffentlicht

2013

- 30. April: Der MEAN-Stack: MongoDB, ExpressJS, AngularJS und Node.js
- 17. Mai: Wie haben wir die erste Node.js-Anwendung von eBay erstellt?
- 15. November: PayPal veröffentlicht Kraken, ein Node.js-Framework
- 22. November: Node.js Memory Leak bei Walmart
    - Eran Hammer von Wal-Mart-Laboren kam zum Kernteam von Node.js und beschwerte sich über ein Gedächtnisleck, das er seit Monaten aufgespürt hatte.
- 19. Dezember: Koa - Web-Framework für Node.js

2014

- 15. Januar: TJ Fontaine übernimmt das Node-Projekt
- 23. Oktober: Node.js-Beirat
    - Joyent und mehrere Mitglieder der Node.js-Community kündigten einen Vorschlag für ein Node.js-Beratungsgremium als nächsten Schritt hin zu einem vollständig offenen Steuerungsmodell für das Open-Source-Projekt Node.js an.
- 19. November: Node.js in Flammen-Diagrammen - Netflix
- 28. November: IO.js - Evented I / O für V8 Javascript

2015

Q1

- 14. Januar: IO.js 1.0.0
- 10. Februar: Joyent will die Node.js Foundation gründen
    - Joyent, IBM, Microsoft, PayPal, Treue, SAP und The Linux Foundation unterstützen gemeinsam die Node.js-Community mit neutraler und offener Governance
- 27. Februar: Abstimmungsvorschlag für IO.js und Node.js

Q2

- 14. April: npm Private Module
- 28. Mai: Node Lead TJ Fontaine tritt zurück und verlässt Joyent
- 13. Mai: Node.js und io.js werden unter der Node Foundation zusammengeführt

Q3

- 2. August: Trace - Node.js Leistungsüberwachung und Debugging
    - Trace ist ein visualisiertes Microservice-Überwachungstool, mit dem Sie alle Metriken erhalten, die Sie für den Betrieb von Microservices benötigen.
- 13. August: 4.0 ist die neue 1.0

Q4

- 12. Oktober: Node v4.2.0, erste Langzeit-Support-Version
- 8. Dezember: Apigee, RisingStack und Yahoo treten der Node.js Foundation bei
- 8. und 9. Dezember: Node Interactive
    - Die erste jährliche Node.js-Konferenz der Node.js Foundation

2016

Q1

- 10. Februar: Express wird zum Inkubationsprojekt
- 23. März: Der Vorfall auf der linken Seite
- 29. März: Google Cloud Platform tritt der Node.js Foundation bei

Q2

- 26. April: npm hat 210.000 Benutzer

Q3

- 18. Juli: CJ Silverio wird CTO von npm
- 1. August: Trace, die Debugging-Lösung Node.js wird allgemein verfügbar
- 15. September: Der erste Node Interactive in Europa

Q4

- 11. Oktober: Der Garnpaketmanager wurde veröffentlicht
- 18. Oktober: Node.js 6 wird zur LTS-Version

Referenz

1. "Geschichte von Node.js in einer Timeline" [Online]. Verfügbar: [https://blog.risingstack.com/history-of-node-js]

Nodejs Geschichte online lesen: https://riptutorial.com/de/node-js/topic/8653/nodejs-geschichte

Kapitel 75: NodeJS mit Redis

Bemerkungen

Wir haben die grundlegenden und am häufigsten verwendeten Operationen in node_redis behandelt. Mit diesem Modul können Sie die volle Leistungsfähigkeit von Redis nutzen und wirklich ausgeklügelte Node.js-Apps erstellen. Mit dieser Bibliothek können Sie viele interessante Dinge erstellen, z. B. eine starke Caching-Schicht, ein leistungsstarkes Pub / Sub-Messagingsystem und vieles mehr. Um mehr über die Bibliothek zu erfahren, lesen Sie die Dokumentation .

Examples

Fertig machen

node_redis ist, wie Sie vielleicht schon vermutet haben, der Redis-Client für Node.js. Sie können es über npm mit dem folgenden Befehl installieren.

```
npm install redis
```

Nachdem Sie das node_redis-Modul installiert haben, können Sie loslegen. Erstellen wir eine einfache Datei, app.js, und sehen Sie, wie Sie von Node.js aus eine Verbindung mit Redis herstellen.

app.js

```
var redis = require('redis');
client = redis.createClient(); //creates a new client
```

Standardmäßig verwendet redis.createClient () 127.0.0.1 und 6379 als Hostnamen und Port. Wenn Sie einen anderen Host / Port haben, können Sie diesen wie folgt angeben:

```
var client = redis.createClient(port, host);
```

Jetzt können Sie eine Aktion ausführen, sobald eine Verbindung hergestellt wurde. Grundsätzlich müssen Sie nur auf Verbindungsereignisse warten, wie unten gezeigt.

```
client.on('connect', function() {
    console.log('connected');
});
```

Der folgende Ausschnitt ist also in app.js enthalten:

```
var redis = require('redis');
var client = redis.createClient();
```

```
client.on('connect', function() {
    console.log('connected');
});
```

Geben Sie im Appliance node app ein, um die App auszuführen. Stellen Sie sicher, dass Ihr Redis-Server betriebsbereit ist, bevor Sie dieses Snippet ausführen.

Schlüsselwertpaare speichern

Nun, da Sie wissen, wie Sie mit Redis von Node.js aus eine Verbindung herstellen, sehen Sie, wie Schlüssel-Wert-Paare in Redis-Speicher gespeichert werden.

Saiten speichern

Alle Redis-Befehle werden als verschiedene Funktionen für das Client-Objekt verfügbar gemacht. Um eine einfache Zeichenfolge zu speichern, verwenden Sie folgende Syntax:

```
client.set('framework', 'AngularJS');
```

Oder

```
client.set(['framework', 'AngularJS']);
```

Die obigen Ausschnitte speichern einen einfachen String AngularJS im Schlüsselrahmen. Sie sollten beachten, dass beide Snippets dasselbe tun. Der einzige Unterschied besteht darin, dass der erste eine variable Anzahl von Argumenten übergibt, während der spätere ein args-Array an die `client.set()` Funktion `client.set()` . Sie können auch einen optionalen Rückruf übergeben, um eine Benachrichtigung zu erhalten, wenn der Vorgang abgeschlossen ist:

```
client.set('framework', 'AngularJS', function(err, reply) {
  console.log(reply);
});
```

Wenn der Vorgang aus irgendeinem Grund fehlgeschlagen ist, repräsentiert das Argument `err` für den Rückruf den Fehler. Gehen Sie folgendermaßen vor, um den Wert des Schlüssels abzurufen:

```
client.get('framework', function(err, reply) {
    console.log(reply);
});
```

`client.get()` können Sie einen in Redis gespeicherten Schlüssel abrufen. Auf den Wert des Schlüssels kann über das Rückrufargument Reply zugegriffen werden. Wenn der Schlüssel nicht vorhanden ist, ist der Wert der Antwort leer.

Hash speichern

Das Speichern einfacher Werte löst Ihr Problem oft nicht. Sie müssen Hashes (Objekte) in Redis speichern. Dafür können Sie die `hmset()` Funktion wie folgt verwenden:

```
client.hmset('frameworks', 'javascript', 'AngularJS', 'css', 'Bootstrap', 'node', 'Express');

client.hgetall('frameworks', function(err, object) {
    console.log(object);
});
```

Das obige Snippet speichert einen Hash in Redis, der jede Technologie ihrem Framework zuordnet. Das erste Argument für `hmset()` ist der Name des Schlüssels. Nachfolgende Argumente repräsentieren Schlüsselwertpaare. In ähnlicher Weise wird `hgetall()` verwendet, um den Wert des Schlüssels abzurufen. Wenn der Schlüssel gefunden wird, enthält das zweite Argument des Rückrufs den Wert, der ein Objekt ist.

Beachten Sie, dass Redis keine verschachtelten Objekte unterstützt. Alle Eigenschaftswerte im Objekt werden vor dem Speichern in Strings umgewandelt. Sie können auch die folgende Syntax verwenden, um Objekte in Redis zu speichern:

```
client.hmset('frameworks', {
    'javascript': 'AngularJS',
    'css': 'Bootstrap',
    'node': 'Express'
});
```

Ein optionaler Rückruf kann auch weitergeleitet werden, um zu erfahren, wann der Vorgang abgeschlossen ist.

Alle Funktionen (Befehle) können mit Groß- / Kleinschreibung aufgerufen werden. Beispielsweise sind `client.hmset()` und `client.HMSET()` gleich. Listen speichern

Wenn Sie eine Liste mit Elementen speichern möchten, können Sie Redis-Listen verwenden. Um eine Liste zu speichern, verwenden Sie folgende Syntax:

```
client.rpush(['frameworks', 'angularjs', 'backbone'], function(err, reply) {
    console.log(reply); //prints 2
});
```

Das obige Snippet erstellt eine Liste namens Frameworks und schiebt zwei Elemente dorthin. Die Länge der Liste beträgt also jetzt zwei. Wie Sie sehen, habe ich ein `args` Array an `rpush` . Das erste Element des Arrays stellt den Namen des Schlüssels dar, während der Rest die Elemente der Liste darstellt. Sie können auch `lpush()` anstelle von `rpush()` , um die Elemente nach links zu verschieben.

Um die Elemente der Liste `lrange()` können Sie die `lrange()` Funktion wie folgt verwenden:

```
client.lrange('frameworks', 0, -1, function(err, reply) {
    console.log(reply); // ['angularjs', 'backbone']
});
```

Beachten Sie, dass Sie alle Elemente der Liste erhalten, indem Sie -1 als drittes Argument an `lrange()` . Wenn Sie eine Untermenge der Liste wünschen, sollten Sie hier den Endindex übergeben.

Sets speichern

Sets ähneln Listen, aber der Unterschied ist, dass sie keine Duplikate zulassen. Wenn Sie also keine doppelten Elemente in Ihrer Liste haben möchten, können Sie einen Satz verwenden. So können wir unser vorheriges Snippet so ändern, dass ein Satz anstelle einer Liste verwendet wird.

```
client.sadd(['tags', 'angularjs', 'backbonejs', 'emberjs'], function(err, reply) {
    console.log(reply); // 3
});
```

Wie Sie sehen, erstellt die Funktion `sadd()` einen neuen Satz mit den angegebenen Elementen. Hier ist die Länge des Sets drei. Um die Mitglieder des Sets `smembers()` , verwenden Sie die Funktion `smembers()` wie folgt:

```
client.smembers('tags', function(err, reply) {
    console.log(reply);
});
```

Dieses Snippet ruft alle Mitglieder des Sets ab. Beachten Sie, dass die Reihenfolge beim Abrufen der Mitglieder nicht beibehalten wird.

Dies ist eine Liste der wichtigsten Datenstrukturen, die in jeder von Redis betriebenen App gefunden werden. Neben Strings, Listen, Sets und Hashwerten können Sie sortierte Sets, HyperLogLogs und mehr in Redis speichern. Wenn Sie eine vollständige Liste von Befehlen und Datenstrukturen wünschen, besuchen Sie die offizielle Redis-Dokumentation. Denken Sie daran, dass fast alle Redis-Befehle für das Client-Objekt verfügbar gemacht werden, das vom node_redis-Modul angeboten wird.

Einige wichtigere Operationen, die von node_redis unterstützt werden.

Vorhandensein von Schlüsseln prüfen

Manchmal müssen Sie möglicherweise überprüfen, ob ein Schlüssel bereits vorhanden ist, und entsprechend vorgehen. Um dies zu tun, können Sie wie folgt die `exists()` Funktion verwenden:

```
client.exists('key', function(err, reply) {
    if (reply === 1) {
        console.log('exists');
    } else {
        console.log('doesn\'t exist');
    }
});
```

Schlüssel löschen und ablaufen lassen

Manchmal müssen Sie einige Schlüssel löschen und sie neu initialisieren. Um die Tasten zu löschen, können Sie den Befehl del wie folgt verwenden:

```
client.del('frameworks', function(err, reply) {
    console.log(reply);
```

```
});
```

Sie können einem vorhandenen Schlüssel auch eine Ablaufzeit wie folgt zuweisen:

```
client.set('key1', 'val1');
client.expire('key1', 30);
```

Das obige Snippet weist dem Schlüssel key1 eine Ablaufzeit von 30 Sekunden zu.

Inkrementieren und Dekrementieren

Redis unterstützt auch das Inkrementieren und Dekrementieren von Schlüsseln. Um einen Schlüssel zu inkrementieren, verwenden Sie die Funktion `incr()` wie folgt:

```
client.set('key1', 10, function() {
    client.incr('key1', function(err, reply) {
        console.log(reply); // 11
    });
});
```

Die `incr()` Funktion erhöht einen Schlüsselwert um 1. Wenn Sie einen anderen Betrag `incrby()` , können Sie die Funktion `incrby()` verwenden. Um eine Taste zu dekrementieren, können Sie die Funktionen `decr()` und `decrby()` .

NodeJS mit Redis online lesen: https://riptutorial.com/de/node-js/topic/7107/nodejs-mit-redis

Kapitel 76: NodeJs Routing

Einführung

Einrichten eines einfachen Express-Webservers unter dem Knoten js und Erkundung des Express-Routers.

Bemerkungen

Mit Hilfe des Express Routers können Sie schließlich Routing-Funktionen in Ihrer Anwendung verwenden. Die Implementierung ist einfach.

Examples

Express-Webserver-Routing

Express-Webserver erstellen

Der Express-Server hat sich als nützlich erwiesen und durch viele Benutzer und die Community besticht. Es wird populär.

Lässt einen Express-Server erstellen. Für die Paketverwaltung und Flexibilität für die Abhängigkeit verwenden wir NPM (Node Package Manager).

1. Wechseln Sie in das Projektverzeichnis und erstellen Sie die package.json-Datei. ***package.json*** {"name": "expressRouter", "version": "0.0.1", "scripts": {"start": "node Server.js"}, "abhängigkeiten": {"express": "^ 4.12.3 "}}

2. Speichern Sie die Datei und installieren Sie die Expressabhängigkeit mit dem folgenden Befehl *npm install* . Dadurch werden node_modules in Ihrem Projektverzeichnis zusammen mit der erforderlichen Abhängigkeit erstellt.

3. Lassen Sie uns einen Express-Webserver erstellen. Wechseln Sie in das Projektverzeichnis und erstellen Sie die Datei server.js. ***server.js***

   var express = erfordern ("express"); var app = express ();

// Erstellen eines Router () - Objekts

var router = express.Router ();

// Alle Routen hier angeben, dies ist für die Startseite.

```
router.get("/",function(req,res){
res.json({"message" : "Hello World"});
```

});

app.use ("/ api", Router);

// Höre dir diesen Port an

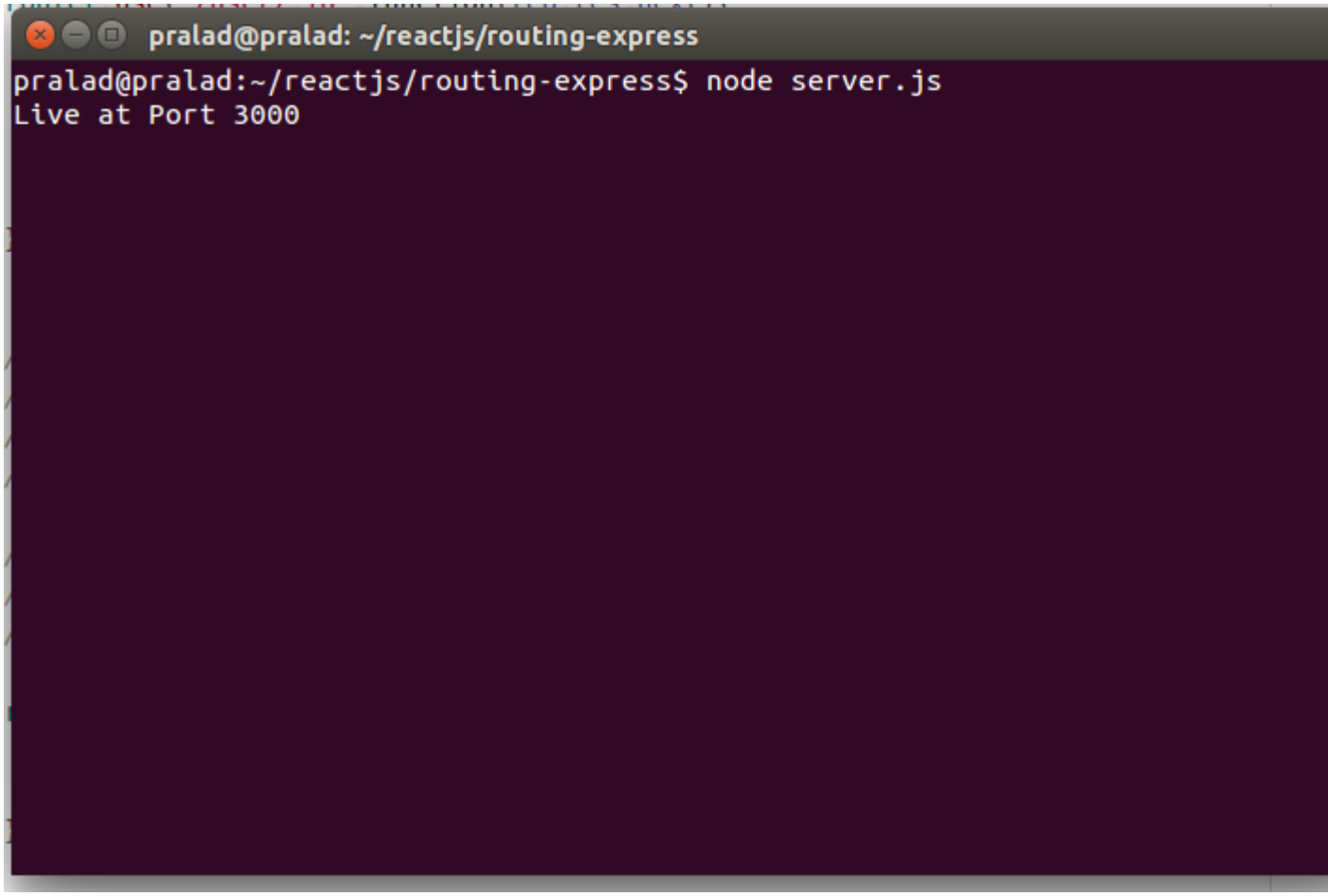
app.listen (3000, function () {console.log ("Live at Port 3000");});

```
For more detail on setting node server you can see [here][1].
```

4. Führen Sie den Server aus, indem Sie den folgenden Befehl eingeben.

   Knoten server.js

   Wenn der Server erfolgreich läuft, werden Sie so etwas sehen.



   .

5. Gehen Sie jetzt zum Browser oder Postboten und stellen Sie eine Anfrage

   http: // localhost: 3000 / api /

   Die Ausgabe wird sein

.

Das ist alles, die Grundlagen des Express-Routings.

Lassen Sie uns nun mit GET, POST usw. umgehen.

Ändern Sie Ihre yous server.js-Datei wie

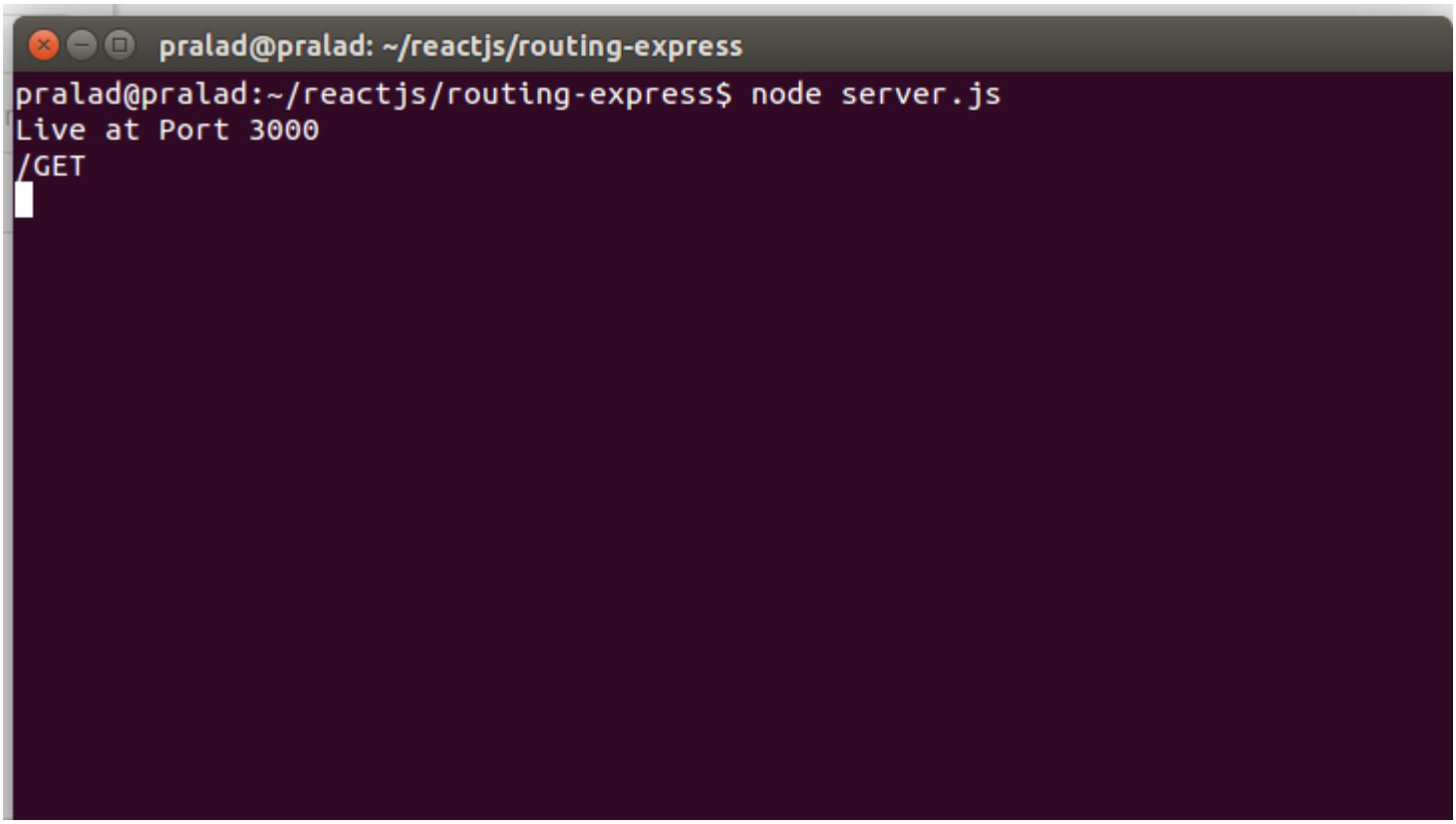
```
var express = require("express");
var app = express();

//Creating Router() object

var router = express.Router();

// Router middleware, mentioned it before defining routes.

router.use(function(req,res,next) {
  console.log("/" + req.method);
  next();
});

// Provide all routes here, this is for Home page.

router.get("/",function(req,res){
  res.json({"message" : "Hello World"});
});


app.use("/api",router);


app.listen(3000,function(){
  console.log("Live at Port 3000");
```

```
});
```

Wenn Sie jetzt den Server neu starten und die Anforderung an stellen

```
http://localhost:3000/api/
```

Sie werden so etwas sehen

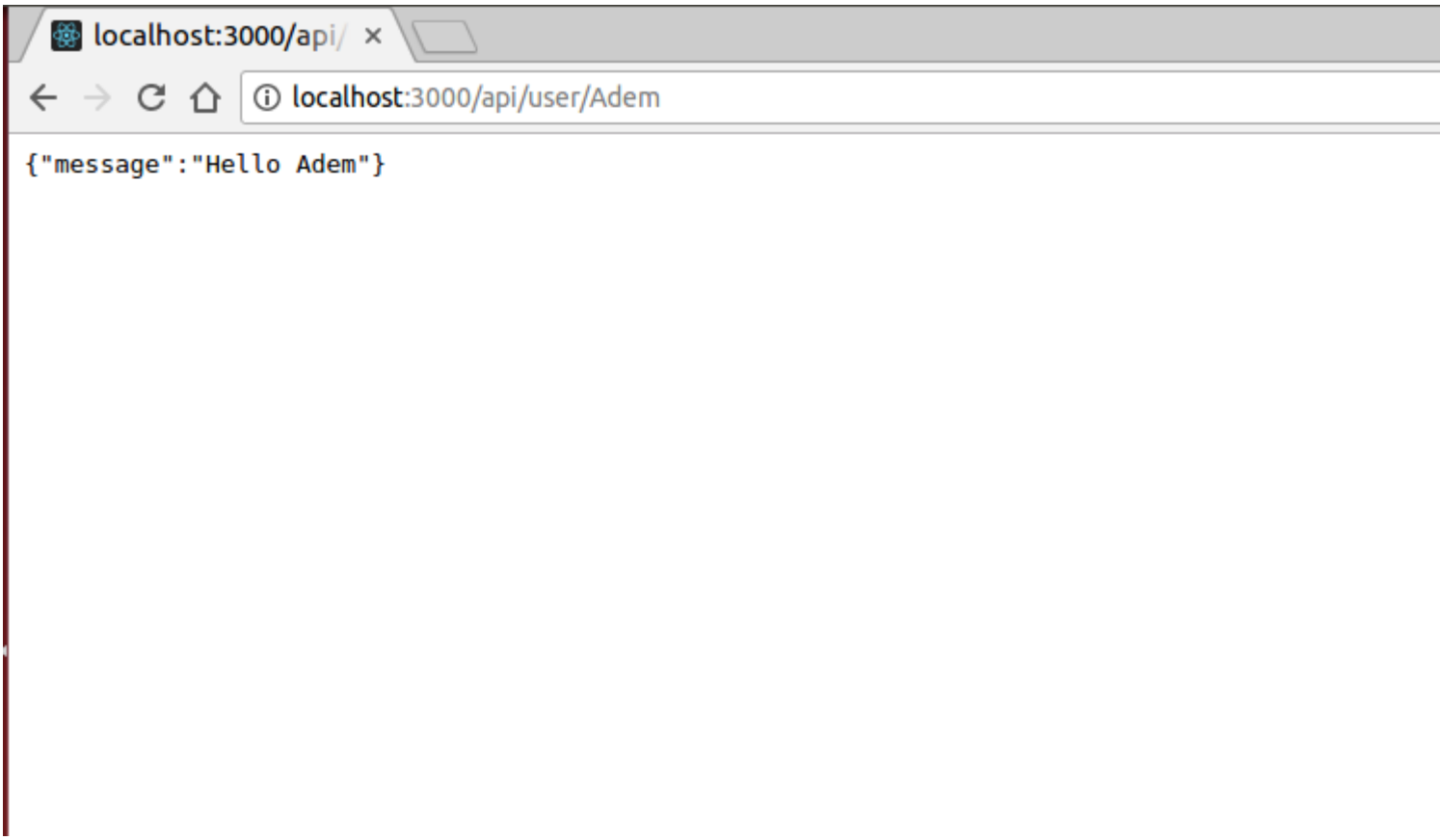


Zugriff auf Parameter im Routing

Sie können auf den Parameter auch von der URL aus zugreifen, z. B. *http://example.com/api/:name/* . Name-Parameter können also Zugriff sein. Fügen Sie den folgenden Code in Ihre server.js ein

```
router.get("/user/:id",function(req,res){
  res.json({"message" : "Hello "+req.params.id});
});
```

Starten Sie nun den Server neu und gehen Sie zu [http: // localhost: 3000 / api / user / Adem] [4]



.

NodeJs Routing online lesen: https://riptutorial.com/de/node-js/topic/9846/nodejs-routing

Kapitel 77: NodeJS-Anfängerhandbuch

Examples

Hallo Welt !

`helloworld.js` den folgenden Code in einen Dateinamen `helloworld.js`

```
console.log("Hello World");
```

Speichern Sie die Datei und führen Sie sie über Node.js aus:

```
node helloworld.js
```

NodeJS-Anfängerhandbuch online lesen: https://riptutorial.com/de/node-js/topic/7693/nodejs-anfangerhandbuch

Kapitel 78: NodeJS-Frameworks

Examples

Webserver-Frameworks

ausdrücken

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

Koa

```
var koa = require('koa');
var app = koa();

app.use(function *(next){
  var start = new Date;
  yield next;
  var ms = new Date - start;
  console.log('%s %s - %s', this.method, this.url, ms);
});

app.use(function *(){
  this.body = 'Hello World';
});

app.listen(3000);
```

Befehlszeilenschnittstellen-Frameworks

Commander.js

```
var program = require('commander');

program
  .version('0.0.1')

program
  .command('hi')
  .description('initialize project configuration')
  .action(function(){
```

```
        console.log('Hi my Friend!!!');
});

program
  .command('bye [name]')
  .description('initialize project configuration')
  .action(function(name){
        console.log('Bye ' + name + '. It was good to see you!');
});

program
  .command('*')
  .action(function(env){
    console.log('Enter a Valid command');
    terminate(true);
});

program.parse(process.argv);
```

Vorpal.js

```
const vorpal = require('vorpal')();

vorpal
  .command('foo', 'Outputs "bar".')
  .action(function(args, callback) {
    this.log('bar');
    callback();
  });

vorpal
  .delimiter('myapp$')
  .show();
```

NodeJS-Frameworks online lesen: https://riptutorial.com/de/node-js/topic/6042/nodejs-frameworks

Kapitel 79: npm

Einführung

Node Package Manager (npm) bietet die folgenden zwei Hauptfunktionen: Online-Repositorys für node.js-Pakete / -Module, die auf search.nodejs.org durchsucht werden können. Befehlszeilenhilfsprogramm zum Installieren von Node.js-Paketen, Ausführen der Versionsverwaltung und Abhängigkeitsverwaltung für Node.js-Pakete.

Syntax

- npm <Befehl> wobei <Befehl> einer der folgenden Werte ist:
  - Nutzer hinzufügen
  - Nutzer hinzufügen
  - apihelp
  - Autor
  - Behälter
  - Bugs
  - c
  - Zwischenspeicher
  - Fertigstellung
  - Konfig
  - ddp
  - Deduplizierung
  - missbilligen
  - docs
  - bearbeiten
  - erkunden
  - FAQ
  - finden
  - Find-Dupes
  - erhalten
  - Hilfe
  - Hilfe-Suche
  - Zuhause
  - ich
  - Installieren
  - Info
  - drin
  - ist nicht alles
  - Probleme
  - la
  - Verknüpfung
  - Liste

- ll
- l
- Anmeldung
- ls
- veraltet
- Inhaber
- Pack
- Präfix
- Pflaume
- veröffentlichen
- r
- rb
- wieder aufbauen
- Löschen
- Repo
- Neustart
- rm
- Wurzel
- Skript ausführen
- s
- se
- Suche
- einstellen
- Show
- Schrumpffolie
- Star
- Sterne
- Start
- halt
- Submodul
- Etikett
- Prüfung
- tst
- un
- deinstallieren
- Verknüpfung aufheben
- nicht veröffentlichen
- Unstar
- oben
- aktualisieren
- v
- Ausführung
- Aussicht
- Wer bin ich

Parameter

Parameter	Beispiel
Zugriff	`npm publish --access=public`
Behälter	`npm bin -g`
bearbeiten	`npm edit connect`
Hilfe	`npm help init`
drin	`npm init`
Installieren	`npm install`
Verknüpfung	`npm link`
Pflaume	`npm prune`
veröffentlichen	`npm publish ./`
Neustart	`npm restart`
Start	`npm start`
halt	`npm start`
aktualisieren	`npm update`
Ausführung	`npm version`

Examples

Pakete installieren

Einführung

Paket ist ein Begriff, der von npm zur Bezeichnung von Tools verwendet wird, die Entwickler für ihre Projekte verwenden können. Dies umfasst alles von Bibliotheken und Frameworks wie jQuery und AngularJS bis hin zu Task-Läufern wie Gulp.js. Die Pakete werden in einem Ordner mit dem Namen `node_modules` , der auch eine `package.json` Datei enthält. Diese Datei enthält Informationen zu allen Paketen, einschließlich Abhängigkeiten. Dies sind zusätzliche Module, die zur Verwendung eines bestimmten Pakets erforderlich sind.

Npm verwendet die Befehlszeile sowohl zum Installieren als auch zum Verwalten von Paketen. Benutzer, die versuchen, npm zu verwenden, sollten sich mit den grundlegenden Befehlen ihres Betriebssystems auskennen, z.

NPM installieren

Beachten Sie, dass zur Installation von Paketen NPM installiert sein muss.

Die empfohlene Methode zur Installation von NPM ist die Verwendung eines der Installationsprogramme auf der Download-Seite Node.js. Sie können überprüfen, ob bereits node.js installiert ist, indem Sie entweder den Befehl `npm -v` oder den Befehl `npm version` ausführen.

Überprüfen Sie nach der Installation von NPM über das Installationsprogramm Node.js, ob Updates verfügbar sind. Dies ist darauf zurückzuführen, dass NPM häufiger als das Installationsprogramm Node.js aktualisiert wird. Um nach Updates zu suchen, führen Sie den folgenden Befehl aus:

```
npm install npm@latest -g
```

Wie installiere ich Pakete?

Um ein oder mehrere Pakete zu installieren, gehen Sie folgendermaßen vor:

```
npm install <package-name>
# or
npm i <package-name>...

# e.g. to install lodash and express
npm install lodash express
```

> **Hinweis** : Dadurch wird das Paket in dem Verzeichnis installiert, in dem sich die Befehlszeile gerade befindet. Daher ist es wichtig zu prüfen, ob das entsprechende Verzeichnis ausgewählt wurde

Wenn sich in Ihrem aktuellen Arbeitsverzeichnis bereits eine `package.json` Datei befindet und darin Abhängigkeiten definiert sind, `package.json npm install` install alle in der Datei aufgelisteten Abhängigkeiten automatisch auf und installiert sie. Sie können auch die Kurzversion des Befehls `npm install` : `npm i`

Wenn Sie eine bestimmte Version eines Pakets installieren möchten, verwenden Sie Folgendes:

```
npm install <name>@<version>

# e.g. to install version 4.11.1 of the package lodash
npm install lodash@4.11.1
```

Wenn Sie eine Version installieren möchten, die einem bestimmten Versionsbereich entspricht, verwenden Sie:

```
npm install <name>@<version range>

# e.g. to install a version which matches "version >= 4.10.1" and "version < 4.11.1"
# of the package lodash
npm install lodash@">=4.10.1 <4.11.1"
```

Wenn Sie die neueste Version installieren möchten, verwenden Sie:

```
npm install <name>@latest
```

Die obigen Befehle werden im zentralen `npm` Repository unter npmjs.com nach Paketen suchen . Wenn Sie nicht von der `npm` Registry installieren `npm` , werden andere Optionen unterstützt, z.

```
# packages distributed as a tarball
npm install <tarball file>
npm install <tarball url>

# packages available locally
npm install <local path>

# packages available as a git repository
npm install <git remote url>

# packages available on GitHub
npm install <username>/<repository>

# packages available as gist (need a package.json)
npm install gist:<gist-id>

# packages from a specific repository
npm install --registry=http://myreg.mycompany.com <package name>

# packages from a related group of packages
# See npm scope
npm install @<scope>/<name>(@<version>)

# Scoping is useful for separating private packages hosted on private registry from
# public ones by setting registry for specific scope
npm config set @mycompany:registry http://myreg.mycompany.com
npm install @mycompany/<package name>
```

Normalerweise werden Module lokal in einem Ordner namens `node_modules` , der sich in Ihrem aktuellen Arbeitsverzeichnis befindet. Dies ist das Verzeichnis, das von `require()` zum Laden von Modulen verwendet wird, um sie für Sie verfügbar zu machen.

Wenn Sie bereits eine `package.json` Datei erstellt haben, können Sie die Option `--save` (`--save -S`) oder eine ihrer Varianten verwenden, um das installierte Paket `package.json` als abhängige `package.json` zu `package.json` hinzuzufügen. Wenn das Paket von einer anderen Person installiert wird, liest `npm` automatisch Abhängigkeiten aus der Datei `package.json` und installiert die aufgelisteten Versionen. Beachten Sie, dass Sie Ihre Abhängigkeiten weiterhin hinzufügen und verwalten können, indem Sie die Datei später bearbeiten. Daher ist es normalerweise ratsam, Abhängigkeiten zu verfolgen, z. B. mit:

```
npm install --save <name> # Install dependencies
```

```
# or
npm install -S <name> # shortcut version --save
# or
npm i -S <name>
```

Befolgen Sie den folgenden Befehl, um Pakete zu installieren und nur dann zu speichern, wenn sie für die Entwicklung, nicht für die Ausführung der Anwendung und nicht für die Ausführung der Anwendung benötigt werden.

```
npm install --save-dev <name> # Install dependencies for development purposes
# or
npm install -D <name> # shortcut version --save-dev
# or
npm i -D <name>
```

Abhängigkeiten installieren

Einige Module stellen nicht nur eine Bibliothek zur Verfügung, sondern auch eine oder mehrere Binärdateien, die über die Befehlszeile verwendet werden sollen. Sie können diese Pakete zwar weiterhin lokal installieren, es wird jedoch häufig bevorzugt, sie global zu installieren, damit die Befehlszeilenprogramme aktiviert werden können. In diesem Fall verknüpft `npm` die Binärdateien automatisch mit den entsprechenden Pfaden (z. B. `/usr/local/bin/<name>`), sodass sie von der Befehlszeile aus verwendet werden können. Um ein Paket global zu installieren, verwenden Sie:

```
npm install --global <name>
# or
npm install -g <name>
# or
npm i -g <name>

# e.g. to install the grunt command line tool
npm install -g grunt-cli
```

Wenn Sie eine Liste aller installierten Pakete und ihrer zugehörigen Versionen im aktuellen Arbeitsbereich anzeigen möchten, verwenden Sie Folgendes:

```
npm list
npm list <name>
```

Durch das Hinzufügen eines optionalen Namensarguments kann die Version eines bestimmten Pakets überprüft werden.

Hinweis: Wenn Sie beim Versuch, ein npm-Modul global zu installieren, auf Berechtigungsprobleme stoßen, widerstehen Sie der Versuchung, ein `sudo npm install -g ...` , um das Problem zu lösen. Es ist gefährlich, Skripts von Drittanbietern für die Ausführung auf Ihrem System mit erhöhten Berechtigungen zuzulassen. Das Berechtigungsproblem kann bedeuten, dass Sie ein Problem mit der Art und Weise haben, wie `npm` selbst installiert wurde.

Wenn Sie an der Installation von Node in Sandbox-Benutzerumgebungen interessiert sind, möchten Sie möglicherweise versuchen, nvm zu verwenden .

Wenn Sie über Build-Tools oder andere Entwicklungsabhängigkeiten verfügen (z. B. Grunt), möchten Sie möglicherweise, dass diese nicht mit der von Ihnen bereitgestellten Anwendung gebündelt werden. In diesem Fall sollten Sie es als Entwicklungsabhängigkeit haben, die in der `package.json` unter `devDependencies` . Verwenden Sie `--save-dev` (oder `-D`), um ein Paket als Entwicklungsabhängigkeit zu installieren.

```
npm install --save-dev <name> // Install development dependencies which is not included in
production
# or
npm install -D <name>
```

Sie werden sehen, dass das Paket dann zu den `devDependencies` Ihrer `package.json` .

Um Abhängigkeiten eines heruntergeladenen / geklonten node.js-Projekts zu installieren, können Sie einfach Folgendes verwenden

```
npm install
# or
npm i
```

npm liest automatisch die Abhängigkeiten von `package.json` und installiert sie.

NPM hinter einem Proxyserver

Wenn Ihr Internetzugang über einen Proxyserver erfolgt, müssen Sie möglicherweise die npm-Installationsbefehle ändern, die auf Remote-Repositorys zugreifen. npm verwendet eine Konfigurationsdatei, die über die Befehlszeile aktualisiert werden kann:

```
npm config set
```

Sie können Ihre Proxy-Einstellungen in den Einstellungen Ihres Browsers finden. Nachdem Sie die Proxy-Einstellungen erhalten haben (Server-URL, Port, Benutzername und Kennwort); Sie müssen Ihre npm-Konfigurationen wie folgt konfigurieren.

```
$ npm config set proxy http://<username>:<password>@<proxy-server-url>:<port>
$ npm config set https-proxy http://<username>:<password>@<proxy-server-url>:<port>
```

`username` , `password` , `port` sind Felder optional. Nachdem Sie diese Einstellungen vorgenommen haben, funktionieren Ihre `npm install` , `npm i -g` usw. ordnungsgemäß.

Bereiche und Repositories

```
# Set the repository for the scope "myscope"
npm config set @myscope:registry http://registry.corporation.com
```

```
# Login at a repository and associate it with the scope "myscope"
npm adduser --registry=http://registry.corporation.com --scope=@myscope

# Install a package "mylib" from the scope "myscope"
npm install @myscope/mylib
```

Wenn der Name Ihres eigenen Pakets mit `@myscope` beginnt und der Bereich "myscope" einem anderen Repository zugeordnet ist `npm publish` lädt `npm publish` Ihr Paket stattdessen in dieses Repository hoch.

Sie können diese Einstellungen auch in einer `.npmrc` Datei `.npmrc` :

```
@myscope:registry=http://registry.corporation.com
//registry.corporation.com/:_authToken=xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxx
```

Dies ist nützlich, wenn Sie den Build auf einem CI-Server automatisieren möchten, z

Pakete deinstallieren

Um ein oder mehrere lokal installierte Pakete zu deinstallieren, verwenden Sie:

```
npm uninstall <package name>
```

Der Deinstallationsbefehl für npm enthält fünf Aliase, die ebenfalls verwendet werden können:

```
npm remove <package name>
npm rm <package name>
npm r <package name>

npm unlink <package name>
npm un <package name>
```

Wenn Sie das Paket als Teil der Deinstallation aus der `package.json` Datei entfernen `package.json` , verwenden Sie das Flag `--save` (Abkürzung: `-S`):

```
npm uninstall --save <package name>
npm uninstall -S <package name>
```

Verwenden `--save-dev` für eine Entwicklungsabhängigkeit das `--save-dev` (Kurzform: `-D`):

```
npm uninstall --save-dev <package name>
npm uninstall -D <package name>
```

Verwenden `--save-optional` für eine optionale Abhängigkeit das `--save-optional` (Kürzel: `-O`):

```
npm uninstall --save-optional <package name>
npm uninstall -O <package name>
```

Für Pakete, die global installiert werden, verwenden Sie das Flag `--global` (Kürzel: `-g`):

```
npm uninstall -g <package name>
```

Grundlegende semantische Versionierung

Bevor Sie ein Paket veröffentlichen, müssen Sie es versionieren. npm unterstützt semantische Versionierung , dh es gibt **Patches, Minor und Major** Releases.

Wenn Ihr Paket beispielsweise die Version 1.2.3 hat, um die Version zu ändern, müssen Sie:

1. Patch-Version: `npm version patch` => 1.2.4
2. Nebenversion: `npm version minor` => 1.3.0
3. Hauptversion: `npm version major` => 2.0.0

Sie können eine Version auch direkt angeben mit:

`npm version 3.1.4` => 3.1.4

Wenn Sie eine Paketversion mit einem der obigen npm-Befehle festlegen, ändert npm das Versionsfeld der package.json-Datei, schreibt es fest und erstellt außerdem ein neues Git-Tag mit dem Präfix "v", als ob Sie es wären habe den Befehl erteilt:

```
git tag v3.1.4
```

Im Gegensatz zu anderen Paketmanagern wie Bower ist die npm-Registrierung nicht darauf angewiesen, dass für jede Version Git-Tags erstellt werden. Wenn Sie jedoch gerne mit Tags arbeiten, sollten Sie daran denken, das neu erstellte Tag zu verschieben, nachdem Sie die Paketversion angehoben haben:

`git push origin master` (um die Änderung auf package.json zu verschieben)

`git push origin v3.1.4` (um das neue Tag zu drücken)

Oder Sie können dies auf einen Schlag mit:

```
git push origin master --tags
```

Paketkonfiguration einrichten

Node.js- `package.json` sind in einer Datei namens `package.json` , die Sie im Stammverzeichnis jedes Projekts finden. Sie können eine neue Konfigurationsdatei einrichten, indem Sie Folgendes aufrufen:

```
npm init
```

Dadurch wird versucht, das aktuelle Arbeitsverzeichnis für Git-Repository-Informationen (falls vorhanden) und Umgebungsvariablen zu lesen, um einige der Platzhalterwerte für Sie zu versuchen und automatisch zu vervollständigen. Andernfalls wird ein Eingabedialogfeld für die grundlegenden Optionen angezeigt.

Wenn Sie eine `package.json` mit Standardwerten erstellen `package.json` , verwenden Sie:

```
npm init --yes
# or
npm init -y
```

Wenn Sie `package.json` für ein Projekt `package.json` , das nicht als npm-Paket veröffentlicht werden soll (dh, nur um die Abhängigkeiten `package.json`), können Sie diese Absicht in Ihrer `package.json` Datei `package.json` :

1. Legen Sie optional die `private` Eigenschaft auf true fest, um eine versehentliche Veröffentlichung zu verhindern.
2. Setzen Sie optional die `license` Eigenschaft auf "UNLICENSED", um anderen das Recht zur Verwendung Ihres Pakets zu verweigern.

Um ein Paket zu installieren und automatisch in `package.json` zu `package.json` , verwenden Sie:

```
npm install --save <package>
```

Das Paket und die zugehörigen Metadaten (z. B. die Paketversion) werden in Ihren Abhängigkeiten angezeigt. Wenn Sie if als Entwicklungsabhängigkeit speichern (mithilfe von `--save-dev`), wird das Paket stattdessen in Ihren `devDependencies` .

Mit diesem bare-bones `package.json` werden beim Installieren oder Aktualisieren von Paketen Warnmeldungen `package.json` , die Sie darüber `package.json` , dass Ihnen eine Beschreibung und das Repository-Feld fehlen. Diese Meldungen können zwar ignoriert werden, Sie können sie jedoch entfernen, indem Sie die package.json in einem beliebigen Texteditor öffnen und dem JSON-Objekt die folgenden Zeilen hinzufügen:

```
[...]
"description": "No description",
"repository": {
  "private": true
},
[...]
```

Paket veröffentlichen

Stellen Sie zunächst sicher, dass Sie Ihr Paket konfiguriert haben (wie in Paketkonfiguration einrichten beschrieben). Dann müssen Sie bei npmjs angemeldet sein.

Wenn Sie bereits einen npm-Benutzer haben

```
npm login
```

Wenn Sie keinen Benutzer haben

```
npm adduser
```

Um zu überprüfen, ob Ihr Benutzer im aktuellen Client registriert ist

```
npm config ls
```

Danach, wenn Ihr Paket zur Veröffentlichung bereit ist, verwenden Sie es

```
npm publish
```

Und du bist fertig.

Wenn Sie eine neue Version veröffentlichen müssen, stellen Sie sicher, dass Sie Ihre Paketversion aktualisieren, wie in der semantischen Versionierung von Basic angegeben. Andernfalls können Sie das Paket nicht über `npm` veröffentlichen.

```
{
    name: "package-name",
    version: "1.0.4"
}
```

Skripte ausführen

Sie können Skripte in Ihrer `package.json` , zum Beispiel:

```
{
  "name": "your-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "author": "",
  "license": "ISC",
  "dependencies": {},
  "devDependencies": {},
  "scripts": {
    "echo": "echo hello!"
  }
}
```

Um das `echo` Skript auszuführen, führen Sie `npm run echo` über die Befehlszeile aus. Willkürliche Skripte, wie beispielsweise `echo` oben, müssen mit `npm run <script name>` . npm verfügt auch über eine Reihe von offiziellen Skripts, die in bestimmten Lebensphasen des Pakets ausgeführt werden (z. B. `preinstall`). Hier finden Sie die gesamte Übersicht darüber, wie npm Skriptfelder behandelt.

npm-Skripts werden am häufigsten für das Starten eines Servers, das Erstellen des Projekts und das Ausführen von Tests verwendet. Hier ist ein realistischeres Beispiel:

```
  "scripts": {
    "test": "mocha tests",
    "start": "pm2 start index.js"
  }
```

In den `scripts` Einträge, Kommandozeilen - Programme wie `mocha` funktionieren , wenn entweder global oder lokal installiert. Wenn der Befehlszeileneintrag nicht im Systempfad vorhanden ist,

überprüft npm Ihre lokal installierten Pakete.

Wenn Ihre Skripts sehr lang werden, können Sie sie wie folgt in Teile aufteilen:

```
  "scripts": {
    "very-complex-command": "npm run chain-1 && npm run chain-2",
    "chain-1": "webpack",
    "chain-2": "node app.js"
  }
```

Überflüssige Pakete entfernen

Führen Sie den folgenden Befehl aus, um überflüssige Pakete (Pakete, die installiert sind, aber nicht in der Abhängigkeitsliste enthalten) zu entfernen:

```
npm prune
```

Um alle `dev` Pakete zu entfernen, fügen `--production` Flag " `--production` :

```
npm prune --production
```

Mehr dazu

Aktuell installierte Pakete auflisten

Um eine Liste (Baumansicht) der aktuell installierten Pakete zu erstellen, verwenden Sie

```
npm list
```

ls , **la** und **ll** sind Aliase des **Listenbefehls** . Die Befehle la und ll zeigen erweiterte Informationen wie Beschreibung und Repository.

Optionen

Das Antwortformat kann durch Übergeben von Optionen geändert werden.

```
npm list --json
```

- **json** - Zeigt Informationen im Json-Format an
- **long** - Zeigt erweiterte Informationen an
- **parseable** - Zeigt eine analysierbare Liste anstelle eines Baums an
- **global** - Zeigt global installierte Pakete an
- **Tiefe** - Maximale Anzeigetiefe des Abhängigkeitsbaums
- **dev** / **development** - Zeigt devDependencies an
- **prod** / **production** - Zeigt Abhängigkeiten

Wenn Sie möchten, können Sie auch die Homepage des Pakets aufrufen.

```
npm home <package name>
```

Aktualisieren von npm und Paketen

Da npm selbst ein Node.js-Modul ist, kann es mit sich selbst aktualisiert werden.

Wenn das Betriebssystem Windows ist, muss die Eingabeaufforderung als Admin ausgeführt werden

```
npm install -g npm@latest
```

Wenn Sie nach aktualisierten Versionen suchen möchten, können Sie Folgendes tun:

```
npm outdated
```

Um ein bestimmtes Paket zu aktualisieren:

```
npm update <package name>
```

> Dadurch wird das Paket gemäß den Einschränkungen in package.json auf die neueste Version aktualisiert

Falls Sie auch die aktualisierte Version in package.json sperren möchten:

```
npm update <package name> --save
```

Sperren von Modulen auf bestimmte Versionen

Standardmäßig installiert npm die neueste verfügbare Version von Modulen entsprechend der semantischen Version der jeweiligen Abhängigkeiten. Dies kann problematisch sein, wenn ein Modulautor sich nicht an Semver hält und beispielsweise bahnbrechende Änderungen in einem Modulupdate einführt.

`node_modules` , um die Version der einzelnen Abhängigkeiten (und die Versionen ihrer Abhängigkeiten usw.) auf die bestimmte lokal installierte Version im Ordner `node_modules`

```
npm shrinkwrap
```

Daraufhin wird neben Ihrem `package.json` ein `npm-shrinkwrap.json` `package.json` dem die spezifischen Versionen der Abhängigkeiten aufgeführt sind.

Einrichten für global installierte Pakete

Sie können `npm install -g` , um ein Paket "global" zu installieren. In der Regel wird dazu eine ausführbare Datei installiert, die Sie Ihrem Ausführungspfad hinzufügen können. Zum Beispiel:

```
npm install -g gulp-cli
```

Wenn Sie Ihren Pfad aktualisieren, können Sie `gulp` direkt anrufen.

In vielen Betriebssystemen versucht `npm install -g` , in ein Verzeichnis zu schreiben, in das Ihr Benutzer möglicherweise nicht schreiben kann, beispielsweise nach `/usr/bin` . Sie sollten `sudo npm install` in diesem Fall **nicht** verwenden, da möglicherweise ein Sicherheitsrisiko besteht, wenn Sie beliebige Skripts mit `sudo` Der Root-Benutzer erstellt möglicherweise Verzeichnisse in Ihrem Zuhause, in die Sie nicht schreiben können.

Sie können `npm` über Ihre Konfigurationsdatei `~/.npmrc` mitteilen, wohin globale Module installiert werden `~/.npmrc` . Dies ist das `prefix` das Sie mit dem `npm prefix` anzeigen können.

```
prefix=~/.npm-global-modules
```

Dies wird das Präfix verwenden, wenn Sie `npm install -g` ausführen. Sie können auch `npm install --prefix ~/.npm-global-modules` , um das Präfix bei der `npm install --prefix ~/.npm-global-modules` . Wenn das Präfix Ihrer Konfiguration entspricht, müssen Sie `-g` nicht verwenden.

Um das global installierte Modul verwenden zu können, muss es sich in Ihrem Pfad befinden:

```
export PATH=$PATH:~/.npm-global-modules/bin
```

Wenn Sie nun `npm install -g gulp-cli` ausführen, können Sie `gulp` .

Hinweis: Wenn Sie `npm install` (ohne `-g`), ist das Präfix das Verzeichnis mit `package.json` oder das aktuelle Verzeichnis, wenn in der Hierarchie kein Verzeichnis gefunden wird. Dadurch wird auch ein Verzeichnis `node_modules/.bin` , das die ausführbaren Dateien enthält. Wenn Sie eine ausführbare Datei verwenden möchten, die für ein Projekt spezifisch ist, müssen Sie `npm install -g` . Sie können die in `node_modules/.bin` .

Verknüpfen von Projekten für schnelleres Debugging und Entwicklung

Das Erstellen von Abhängigkeiten für Projekte kann manchmal eine langwierige Aufgabe sein. Verwenden Sie den `npm link` anstatt eine Paketversion in NPM zu veröffentlichen und die Abhängigkeit zum Testen der Änderungen zu installieren. `npm link` erstellt einen symbolischen `npm link` sodass der neueste Code in einer lokalen Umgebung getestet werden kann. Dies erleichtert das Testen globaler Tools und Projektabhängigkeiten, indem der neueste Code ausgeführt wird, bevor eine veröffentlichte Version erstellt wird.

Hilfstext

```
NAME
       npm-link - Symlink a package folder

SYNOPSIS
         npm link (in package dir)
         npm link [<@scope>/]<pkg>[@<version>]

         alias: npm ln
```

Schritte zum Verknüpfen von Projektabhängigkeiten

Beachten Sie beim Erstellen der Abhängigkeitsverknüpfung, dass der Name des Pakets im übergeordneten Projekt angegeben wird.

1. CD in ein Abhängigkeitsverzeichnis (zB: `cd ../my-dep`)
2. `npm link`
3. CD in das Projekt, das die Abhängigkeit verwendet
4. `npm link my-dep` oder wenn der Namespace `npm link @namespace/my-dep`

Schritte zum Verknüpfen eines globalen Tools

1. CD in das Projektverzeichnis (zB: `cd eslint-watch`)
2. `npm link`
3. Verwenden Sie das Werkzeug
4. `esw --quiet`

Probleme, die auftreten können

Das Verknüpfen von Projekten kann manchmal zu Problemen führen, wenn das Abhängigkeits- oder globale Tool bereits installiert ist. `npm uninstall (-g) <pkg>` und das Ausführen des `npm link` löst normalerweise alle auftretenden Probleme.

npm online lesen: https://riptutorial.com/de/node-js/topic/482/npm

Kapitel 80: nvm - Knotenversionsmanager

Bemerkungen

Die in den obigen Beispielen verwendeten URLs verweisen auf eine bestimmte Version von Node Version Manager. Die neueste Version unterscheidet sich höchstwahrscheinlich von dem, auf das verwiesen wird. Um nvm mit der neuesten Version zu installieren, klicken Sie hier, um auf nvm unter GitHub zuzugreifen. Dort erhalten Sie die neuesten URLs.

Examples

Installieren Sie NVM

Sie können `curl` :

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

Oder Sie können `wget` :

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

Überprüfen Sie die NVM-Version

Um zu überprüfen, ob nvm installiert wurde, führen Sie folgende Schritte aus:

```
command -v nvm
```

welches sollte 'nvm' ausgeben, wenn die Installation erfolgreich war.

Eine bestimmte Node-Version installieren

Auflistung der verfügbaren Remote-Versionen zur Installation

```
nvm ls-remote
```

Remote-Version installieren

```
nvm install <version>
```

Zum Beispiel

```
nvm install 0.10.13
```

Verwenden einer bereits installierten Knotenversion

So listen Sie verfügbare lokale Versionen des Knotens über NVM auf:

```
nvm ls
```

Wenn zum Beispiel `nvm ls` zurückgegeben wird:

```
$ nvm ls
     v4.3.0
     v5.5.0
```

Sie können zu `v5.5.0` wechseln mit:

```
nvm use v5.5.0
```

Installieren Sie nvm unter Mac OSX

INSTALLATIONSPROZESS

Sie können Node Version Manager mit git, curl oder wget installieren. Sie führen diese Befehle in **Terminal** unter **Mac OSX aus** .

Curl-Beispiel:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

wget Beispiel:

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

Testen Sie, dass der NVM ordnungsgemäß installiert wurde

Um zu testen, ob nvm ordnungsgemäß installiert wurde, schließen Sie das Terminal, öffnen Sie es erneut, und geben Sie `nvm` . Wenn Sie **die** Meldung **nvm: command not found erhalten** , verfügt Ihr Betriebssystem möglicherweise nicht über die erforderliche **.bash_profile-** Datei. `touch ~/.bash_profile` in Terminal `touch ~/.bash_profile` und führen Sie das obige Installationsskript erneut aus.

Wenn der **Befehl nvm:** immer noch **nicht gefunden wird** , versuchen Sie Folgendes:

- `nano .bashrc` in Terminal `nano .bashrc` . Sie sollten ein Exportskript sehen, das fast identisch mit dem folgenden ist:

  export NVM_DIR = "/ Benutzer / johndoe / .nvm" [-s "$ NVM_DIR / nvm.sh"] &&. "$ NVM_DIR / nvm.sh"

- Kopieren Sie das **Exportskript** und entfernen Sie es aus **.bashrc**
- Speichern und Schließen der .bashrc-Datei (STRG + O - Eingabe - STRG + X)

- `nano .bash_profile` , um das Bash-Profil zu öffnen
- Fügen Sie das kopierte Exportskript in einer neuen Zeile in das Bash-Profil ein
- Speichern und Schließen des Bash-Profils (STRG + O - Eingabe - STRG + X)
- `nano .bashrc` schließlich `nano .bashrc` , um die **.bashrc-** Datei erneut zu öffnen
- Fügen Sie die folgende Zeile in die Datei ein:

  source ~ / .nvm / nvm.sh

- Speichern und schließen (STRG + O - Enter - STRG + X)
- `nvm` Terminal neu und geben Sie `nvm` , um zu testen, ob es funktioniert

Alias für Knotenversion setzen

Wenn Sie einen Aliasnamen für die Version des installierten Knotens festlegen möchten, führen Sie folgende Schritte aus:

```
nvm alias <name> <version>
```

Vergleiche mit Unalias:

```
nvm unalias <name>
```

Eine richtige Verwendung wäre, wenn Sie eine andere Version als die stabile Version als Standardalias festlegen möchten. `default` Alias-Versionen werden standardmäßig auf der Konsole geladen.

Mögen:

```
nvm alias default 5.0.1
```

Dann ist bei jedem Start von **Konsole / Terminal** 5.0.1 standardmäßig vorhanden.

Hinweis:

```
nvm alias # lists all aliases created on nvm
```

Führen Sie einen beliebigen Befehl in einer Subshell mit der gewünschten Version des Knotens aus

Listen Sie alle installierten Knotenversionen auf

```
nvm ls
    v4.5.0
    v6.7.0
```

Führen Sie den Befehl mit einer beliebigen installierten Knotenversion aus

```
nvm run 4.5.0 --version or nvm exec 4.5.0 node --version
Running node v4.5.0 (npm v2.15.9)
v4.5.0
```

```
nvm run 6.7.0 --version or nvm exec 6.7.0 node --version
Running node v6.7.0 (npm v3.10.3)
v6.7.0
```

---

mit Alias

```
nvm run default --version or nvm exec default node --version
Running node v6.7.0 (npm v3.10.3)
v6.7.0
```

Um die LTS-Version des Knotens zu installieren

```
nvm install --lts
```

Versionswechsel

```
nvm use v4.5.0 or nvm use stable ( alias )
```

nvm - Knotenversionsmanager online lesen: https://riptutorial.com/de/node-js/topic/2823/nvm---knotenversionsmanager

Kapitel 81: OAuth 2.0

Examples

OAuth 2 mit Redis-Implementierung - grant_type: Kennwort

In diesem Beispiel verwende ich oauth2 in rest api mit der redis-Datenbank

> **Wichtig:** Sie müssen die redis-Datenbank auf Ihrem Computer installieren, sie von hier für Linux-Benutzer herunterladen und von hier aus die Windows-Version installieren, und wir werden die redis manager desktop app verwenden. Installieren Sie sie hier .

Nun müssen wir unseren node.js-Server so einstellen, dass er die redis-Datenbank verwendet.

- **Server-Datei erstellen: app.js**

```
    var express = require('express'),
  bodyParser = require('body-parser'),
  oauthserver = require('oauth2-server'); // Would be: 'oauth2-server'

var app = express();

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.oauth = oauthserver({
  model: require('./routes/Oauth2/model'),
  grants: ['password', 'refresh_token'],
  debug: true
});

// Handle token grant requests
app.all('/oauth/token', app.oauth.grant());

app.get('/secret', app.oauth.authorise(), function (req, res) {
  // Will require a valid access_token
  res.send('Secret area');
});

app.get('/public', function (req, res) {
  // Does not require an access_token
  res.send('Public area');
});

// Error handling
app.use(app.oauth.errorHandler());

app.listen(3000);
```

- **Erstellen Sie ein Oauth2-Modell in den Routen / Oauth2 / model.js**

```
 var model = module.exports,
  util = require('util'),
  redis = require('redis');

var db = redis.createClient();

var keys = {
  token: 'tokens:%s',
  client: 'clients:%s',
  refreshToken: 'refresh_tokens:%s',
  grantTypes: 'clients:%s:grant_types',
  user: 'users:%s'
};

model.getAccessToken = function (bearerToken, callback) {
  db.hgetall(util.format(keys.token, bearerToken), function (err, token) {
    if (err) return callback(err);

    if (!token) return callback();

    callback(null, {
      accessToken: token.accessToken,
      clientId: token.clientId,
      expires: token.expires ? new Date(token.expires) : null,
      userId: token.userId
    });
  });
};

model.getClient = function (clientId, clientSecret, callback) {
  db.hgetall(util.format(keys.client, clientId), function (err, client) {
    if (err) return callback(err);

    if (!client || client.clientSecret !== clientSecret) return callback();

    callback(null, {
      clientId: client.clientId,
      clientSecret: client.clientSecret
    });
  });
};

model.getRefreshToken = function (bearerToken, callback) {
  db.hgetall(util.format(keys.refreshToken, bearerToken), function (err, token) {
    if (err) return callback(err);

    if (!token) return callback();

    callback(null, {
      refreshToken: token.accessToken,
      clientId: token.clientId,
      expires: token.expires ? new Date(token.expires) : null,
      userId: token.userId
    });
  });
};

model.grantTypeAllowed = function (clientId, grantType, callback) {
  db.sismember(util.format(keys.grantTypes, clientId), grantType, callback);
};
```

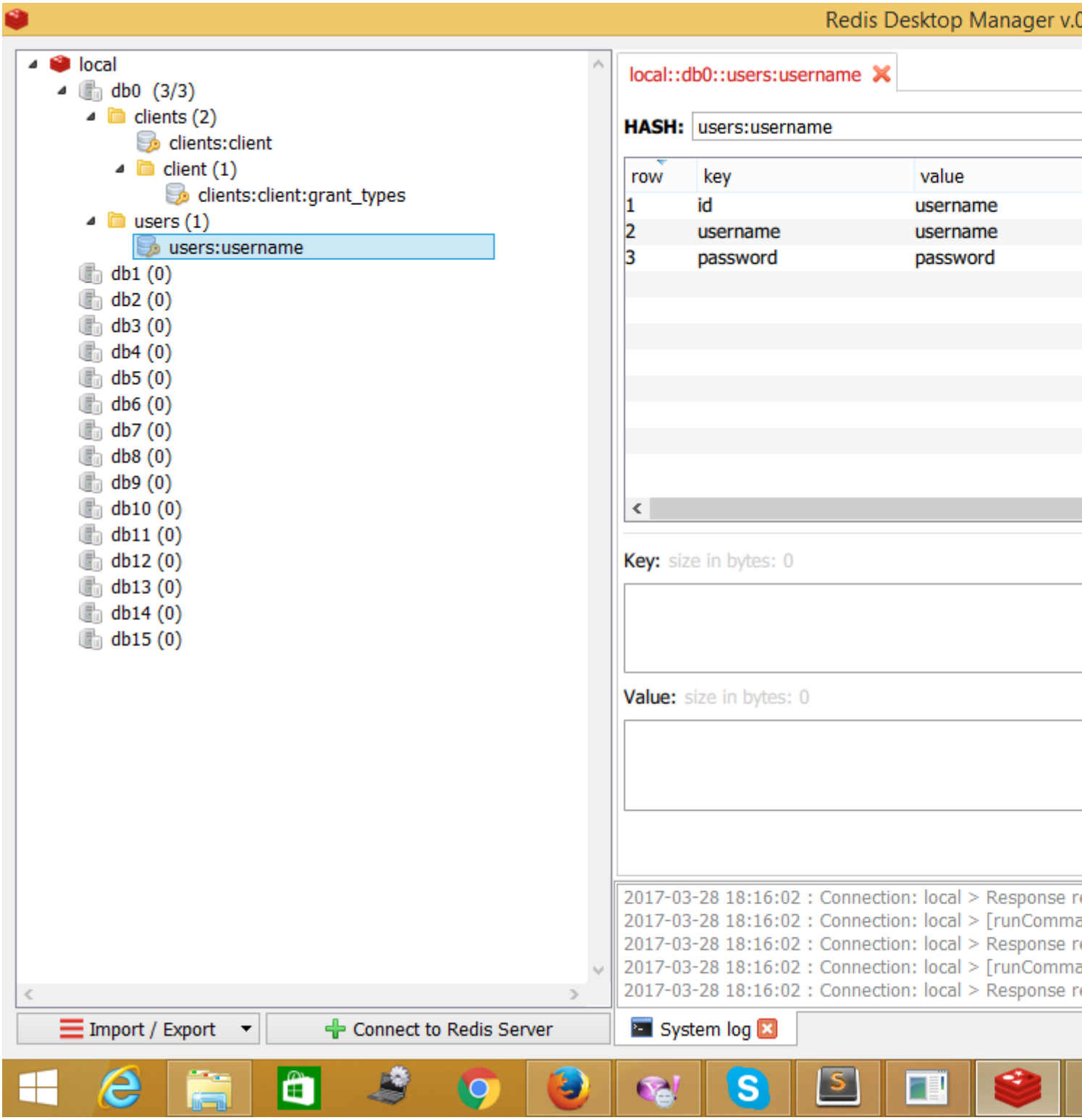
```
model.saveAccessToken = function (accessToken, clientId, expires, user, callback) {
  db.hmset(util.format(keys.token, accessToken), {
    accessToken: accessToken,
    clientId: clientId,
    expires: expires ? expires.toISOString() : null,
    userId: user.id
  }, callback);
};

model.saveRefreshToken = function (refreshToken, clientId, expires, user, callback) {
  db.hmset(util.format(keys.refreshToken, refreshToken), {
    refreshToken: refreshToken,
    clientId: clientId,
    expires: expires ? expires.toISOString() : null,
    userId: user.id
  }, callback);
};

model.getUser = function (username, password, callback) {
  db.hgetall(util.format(keys.user, username), function (err, user) {
    if (err) return callback(err);

    if (!user || password !== user.password) return callback();

    callback(null, {
      id: username
    });
  });
};
```

Sie müssen lediglich redis auf Ihrem Computer installieren und die folgende Knotendatei ausführen

```
  #! /usr/bin/env node

var db = require('redis').createClient();

db.multi()
  .hmset('users:username', {
    id: 'username',
    username: 'username',
    password: 'password'
  })
  .hmset('clients:client', {
    clientId: 'client',
    clientSecret: 'secret'
  })//clientId + clientSecret to base 64 will generate Y2xpZW50OnNlY3JldA==
  .sadd('clients:client:grant_types', [
    'password',
    'refresh_token'
  ])
  .exec(function (errs) {
    if (errs) {
      console.error(errs[0].message);
      return process.exit(1);
    }

    console.log('Client and user added successfully');
    process.exit();
  });
```

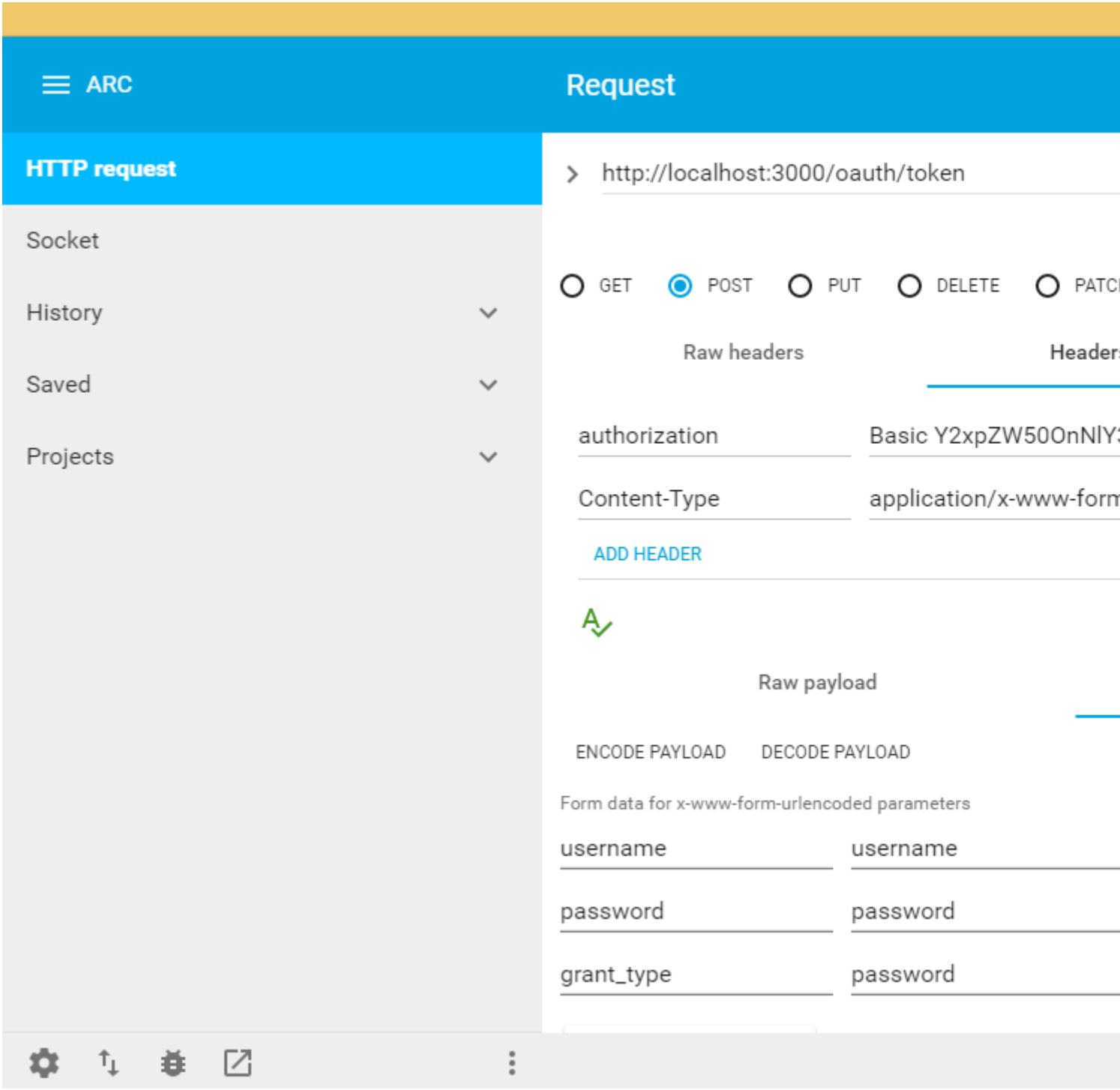
Hinweis : Diese Datei legt die Anmeldeinformationen für Ihr Frontend fest, um das Token anzufordern

Beispiel einer redis-Datenbank nach Aufruf der obigen Datei:



Anfrage wird wie folgt sein:

Beispielanruf an API



Header:

1. Autorisierung: Standard gefolgt von dem Kennwort, das beim ersten Setup festgelegt wurde:

   ein. clientId + secretId in base64

2. Datenformular:

   Benutzername: Benutzer, der das Anforderungstoken verwendet

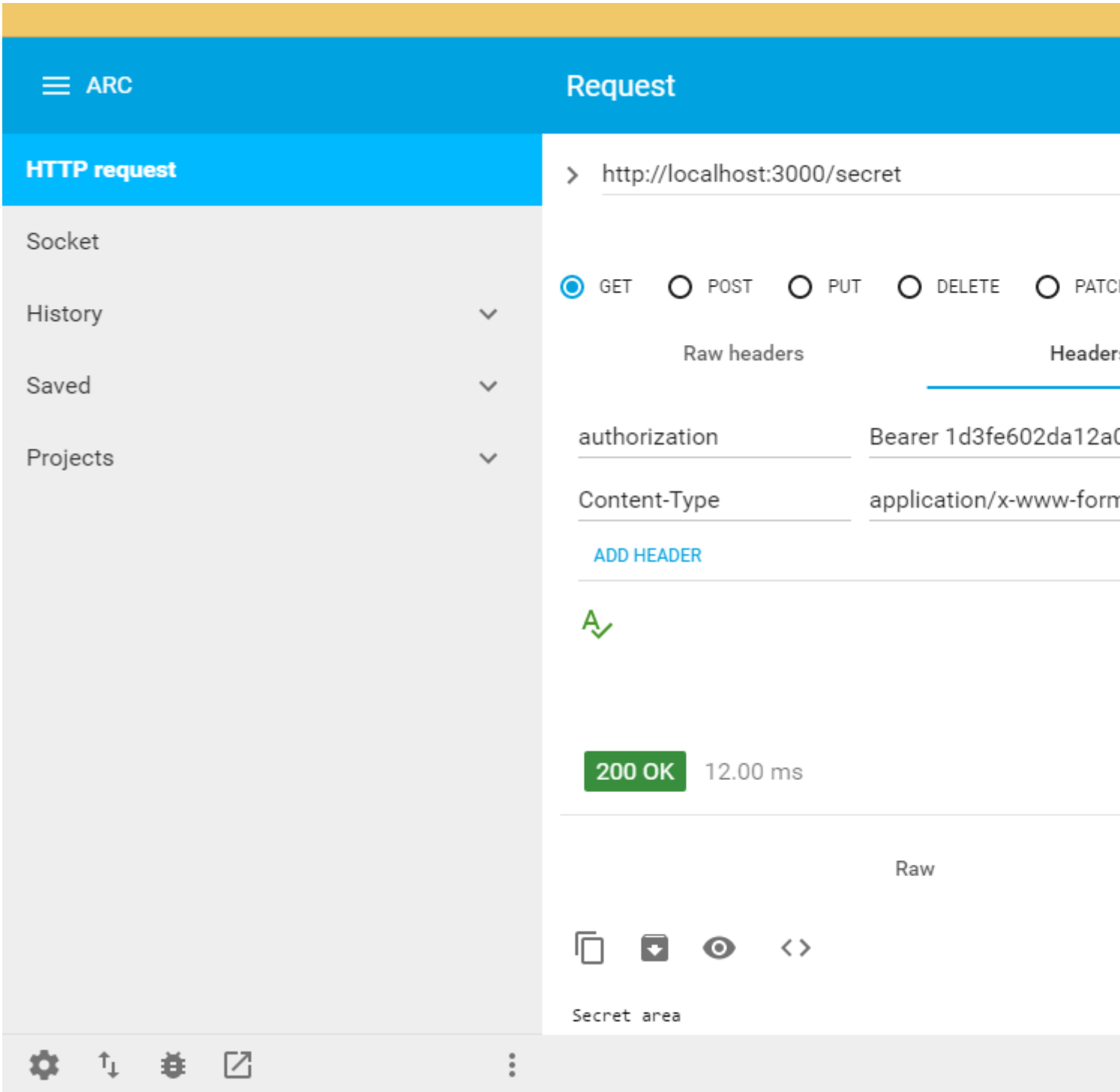
   Passwort: Benutzerpasswort

   grant_type: hängt von den gewünschten Optionen ab. Ich wähle passwod, wobei

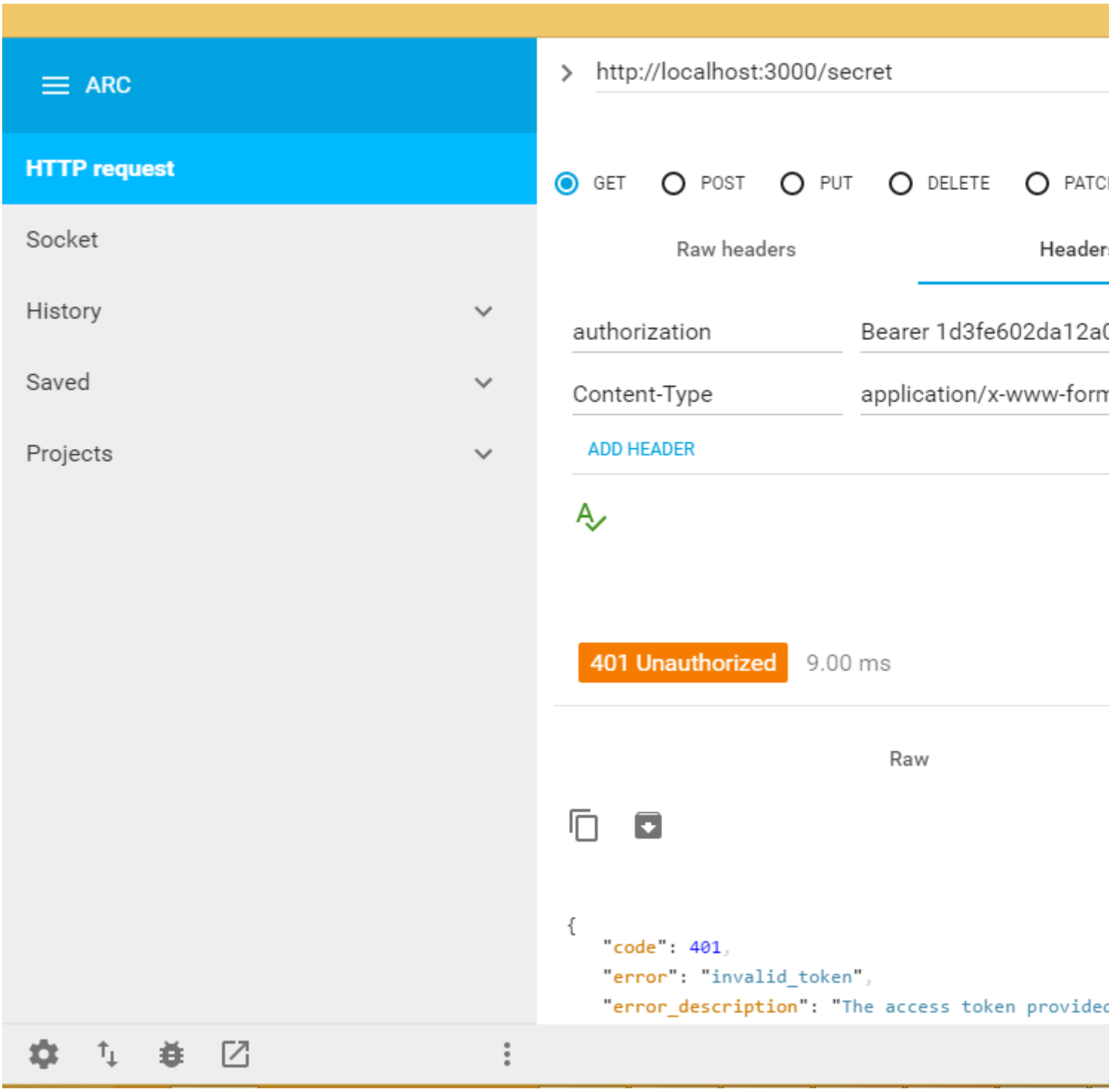
nur Benutzername und Passwort in redis erstellt werden müssen. Data on redis ist wie folgt:

```
{
  "access_token":"1d3fe602da12a086ecb2b996fd7b7ae874120c4f",
  "token_type":"bearer", // Will be used to access api + access+token e.g. bearer
1d3fe602da12a086ecb2b996fd7b7ae874120c4f
  "expires_in":3600,
  "refresh_token":"b6ad56e5c9aba63c85d7e21b1514680bbf711450"
}
```

Wir müssen also unsere API anrufen und einige gesicherte Daten mit unserem soeben erstellten Zugriffstoken beschaffen, siehe unten:

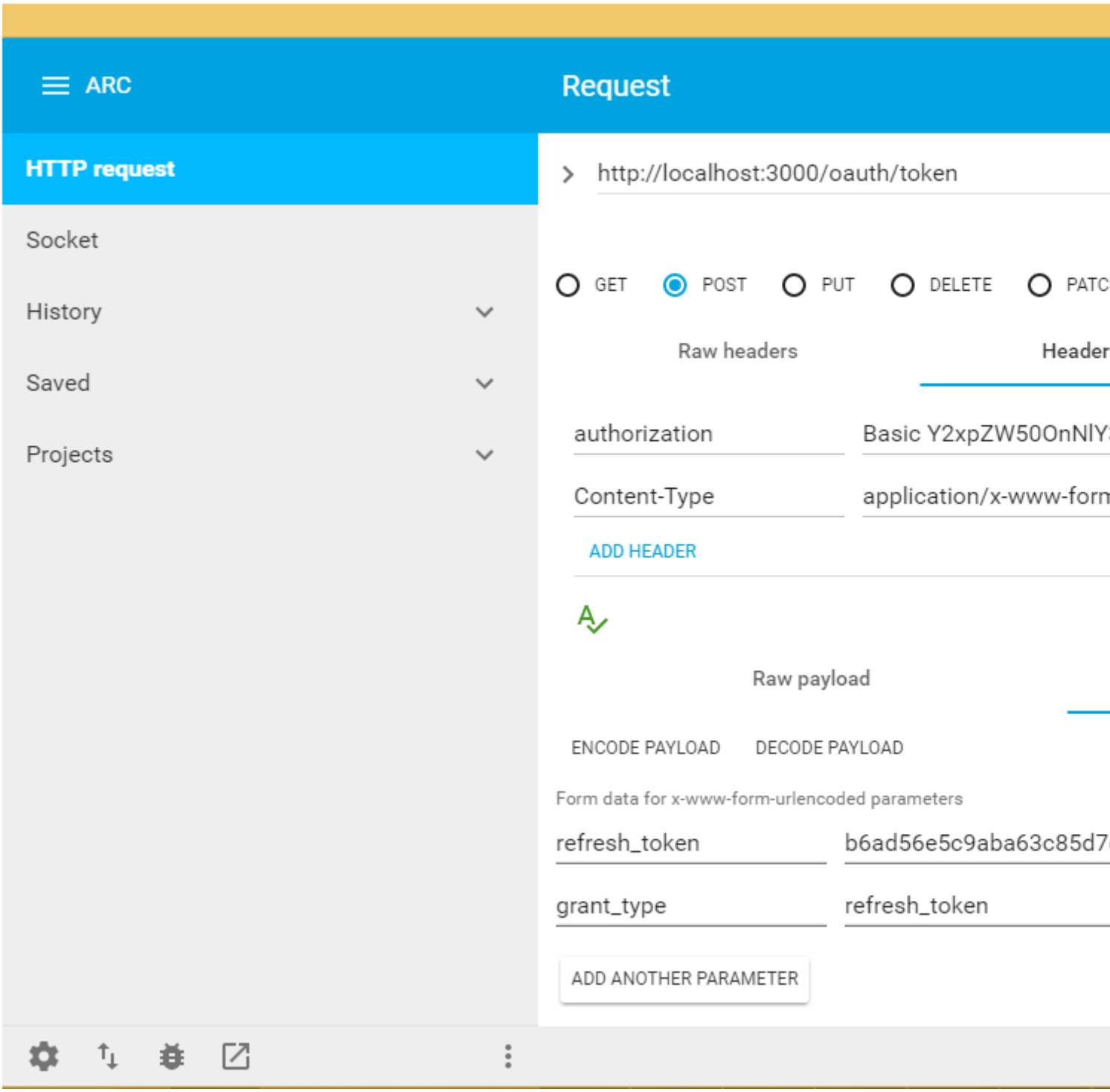


Wenn das Token abläuft, gibt die API einen Fehler aus, bei dem das Token abläuft und Sie keinen Zugriff auf einen der API-Aufrufe haben, siehe Abbildung unten:



> Mal sehen, was zu tun ist, wenn das Token abläuft. Lassen Sie mich es zuerst erklären. Wenn das Zugriffstoken abläuft, ist ein Refresh_Token in redis vorhanden, das auf das abgelaufene Access_Token verweist. Was wir also brauchen, ist, oauth / token erneut mit dem Refresh_token-Grant_type aufzurufen und den zu setzen Wenn Sie die Basis-ClientId: clientsecret (auf Basis 64!) autorisieren und schließlich das refresh_token senden, wird ein neues access_token mit neuen Ablaufdaten generiert.

Das folgende Bild zeigt, wie Sie ein neues Zugriffstoken erhalten:



Hoffe zu helfen!

OAuth 2.0 online lesen: https://riptutorial.com/de/node-js/topic/9566/oauth-2-0

Kapitel 82: package.json

Bemerkungen

Sie können `package.json` mit erstellen

```
npm init
```

was Sie über grundlegende Fakten über Ihre Projekte stellen, einschließlich Lizenz - Kennung .

Examples

Grundlegende Projektdefinition

```
{
    "name": "my-project",
    "version": "0.0.1",
    "description": "This is a project.",
    "author": "Someone <someone@example.com>",
    "contributors": [{
        "name": "Someone Else",
        "email": "else@example.com"
    }],
    "keywords": ["improves", "searching"]
}
```

Feld	Beschreibung
Name	ein **erforderliches** Feld für die Installation eines Pakets. Muss klein geschrieben werden, ein einzelnes Wort ohne Leerzeichen. (Bindestriche und Unterstriche erlaubt)
Ausführung	ein **erforderliches** Feld für die Paketversion mit semantischer Versionierung .
Beschreibung	eine kurze Beschreibung des Projekts
Autor	gibt den Autor des Pakets an
Mitwirkende	ein Array von Objekten, eines für jeden Beitragenden
Schlüsselwörter	eine Reihe von Zeichenfolgen, dies wird den Leuten helfen, Ihr Paket zu finden

Abhängigkeiten

"Abhängigkeiten": {"Modulname": "0.1.0"}

- **genau** : `0.1.0` installiert diese spezifische Version des Moduls.
- **neueste untergeordnete Version** : `^0.1.0` installiert die neueste untergeordnete Version, beispielsweise `0.2.0` , installiert jedoch kein Modul mit einer höheren Hauptversion, z. B. `1.0.0`
- **neuester Patch** : `0.1.x` oder `~0.1.0` installiert die neueste verfügbare Patch-Version, z. B. `0.1.4` , installiert jedoch kein Modul mit einer höheren Haupt- oder Nebenversion, z. B. `0.2.0` oder `1.0.0` .
- **Wildcard** : `*` installiert die neueste Version des Moduls.
- **Git Repository** : Folgendes wird ein Tarball aus dem Hauptzweig eines Git Repo installieren. Ein `#sha` , `#tag` oder `#branch` kann ebenfalls angegeben werden:
  - **GitHub** : `user/project` oder `user/project#v1.0.0`
  - **URL** : `git://gitlab.com/user/project.git` oder `git://gitlab.com/user/project.git#develop`
- **lokaler Pfad** : `file:../lib/project`

Nachdem Sie sie zu package.json hinzugefügt haben, verwenden Sie den Befehl `npm install` in Ihrem Projektverzeichnis im Terminal.

devDependencies

```
"devDependencies": {
    "module-name": "0.1.0"
}
```

Für Abhängigkeiten, die nur für die Entwicklung erforderlich sind, wie zum Beispiel das Testen von Styling-Proxies ext. Diese dev-Abhängigkeiten werden nicht installiert, wenn "npm install" im Produktionsmodus ausgeführt wird.

Skripte

Sie können Skripts definieren, die vor oder nach einem anderen Skript ausgeführt werden können oder ausgelöst werden.

```
{
  "scripts": {
    "pretest": "scripts/pretest.js",
    "test": "scripts/test.js",
    "posttest": "scripts/posttest.js"
  }
}
```

In diesem Fall können Sie das Skript ausführen, indem Sie einen der folgenden Befehle ausführen:

```
$ npm run-script test
$ npm run test
$ npm test
$ npm t
```

Vordefinierte Skripte

Skriptname	Beschreibung
vorveröffentlichen	Ausführen, bevor das Paket veröffentlicht wird.
veröffentlichen, postpublish	Ausführen, nachdem das Paket veröffentlicht wurde.
vorinstallieren	Führen Sie den Vorgang aus, bevor das Paket installiert wird.
installieren, nachinstallieren	Ausführen, nachdem das Paket installiert wurde.
deinstallieren, deinstallieren	Ausführen, bevor das Paket deinstalliert wird.
postuninstall	Nach der Deinstallation des Pakets ausführen.
Vorversion, Version	Führen Sie den Vorgang aus, bevor Sie die Paketversion stoßen.
Postversion	Führen Sie nach dem Bump die Paketversion aus.
Vortest, Test, Posttest	Führen Sie den Befehl `npm test`
Stoppen Sie, stoppen Sie, stoppen Sie	Führen Sie den Befehl `npm stop`
Prestart, Start, Poststart	Führen Sie den Befehl `npm start`
Neustart, Neustart, Neustart	Wird mit dem Befehl `npm restart` ausgeführt

Benutzerdefinierte Skripte

Sie können auch Ihre eigenen Skripts auf die gleiche Weise definieren wie mit den vordefinierten Skripts:

```
{
  "scripts": {
    "preci": "scripts/preci.js",
    "ci": "scripts/ci.js",
    "postci": "scripts/postci.js"
  }
}
```

In diesem Fall können Sie das Skript ausführen, indem Sie einen der folgenden Befehle ausführen:

```
$ npm run-script ci
$ npm run ci
```

Benutzerdefinierte Skripte unterstützt auch *Pre-* und *Post* - Skripte, wie im obigen Beispiel gezeigt.

Erweiterte Projektdefinition

Einige der zusätzlichen Attribute werden von der npm-Website wie `repository` , `bugs` oder `homepage` analysiert und im Infobox für diese Pakete angezeigt

```
{
  "main": "server.js",
  "repository" :  {
    "type": "git",
    "url": "git+https://github.com/<accountname>/<repositoryname>.git"
  },
  "bugs": {
    "url": "https://github.com/<accountname>/<repositoryname>/issues"
  },
  "homepage": "https://github.com/<accountname>/<repositoryname>#readme",
  "files": [
    "server.js", // source files
    "README.md", // additional files
    "lib" // folder with all included files
  ]
}
```

Feld	Beschreibung
Main	Einstiegsskript für dieses Paket. Dieses Skript wird zurückgegeben, wenn ein Benutzer das Paket benötigt.
Repository	Ort und Typ des öffentlichen Repositorys
Bugs	Bugtracker für dieses Paket (zB github)
Startseite	Homepage für dieses Paket oder das allgemeine Projekt
Dateien	Liste der Dateien und Ordner, die heruntergeladen werden sollten, wenn ein Benutzer eine `npm install <packagename>`

Paket.json erkunden

Eine `package.json` Datei, die normalerweise im Projektstammverzeichnis vorhanden ist, enthält Metadaten zu Ihrer App oder Ihrem Modul sowie die Liste der Abhängigkeiten, die von npm bei der Ausführung von `npm install` .

Um eine `package.json` zu initialisieren, geben `package.json` `npm init` an der Eingabeaufforderung ein.

Um eine `package.json` mit Standardwerten zu erstellen, verwenden Sie:

```
npm init --yes
# or
npm init -y
```

Um ein Paket zu installieren und in `package.json` zu `package.json` verwenden Sie:

```
npm install {package name} --save
```

Sie können auch die Kurzschreibweise verwenden:

```
 npm i -S {package name}
```

NPM-Aliasnamen `-S` an `--save` und `-D` an `--save-dev` , um sie in Ihren Produktions- bzw. Entwicklungsabhängigkeiten zu speichern.

Das Paket wird in Ihren Abhängigkeiten angezeigt. Wenn Sie `--save-dev` anstelle von `--save` , wird das Paket in Ihren devDependencies angezeigt.

Wichtige Eigenschaften von `package.json` :

```
{
  "name": "module-name",
  "version": "10.3.1",
  "description": "An example module to illustrate the usage of a package.json",
  "author": "Your Name <your.name@example.org>",
  "contributors": [{
    "name": "Foo Bar",
    "email": "foo.bar@example.com"
  }],
  "bin": {
    "module-name": "./bin/module-name"
  },
  "scripts": {
    "test": "vows --spec --isolate",
    "start": "node index.js",
    "predeploy": "echo About to deploy",
    "postdeploy": "echo Deployed",
    "prepublish": "coffee --bare --compile --output lib/foo src/foo/*.coffee"
  },
  "main": "lib/foo.js",
  "repository": {
    "type": "git",
    "url": "https://github.com/username/repo"
  },
  "bugs": {
    "url": "https://github.com/username/issues"
  },
  "keywords": [
    "example"
  ],
  "dependencies": {
    "express": "4.2.x"
  },
  "devDependencies": {
    "assume": "<1.0.0 || >=2.3.1 <2.4.5 || >=2.5.2 <3.0.0"
  },
```

```
  "peerDependencies": {
    "moment": ">2.0.0"
  },
  "preferGlobal": true,
  "private": true,
  "publishConfig": {
    "registry": "https://your-private-hosted-npm.registry.domain.com"
  },
  "subdomain": "foobar",
  "analyze": true,
  "license": "MIT",
  "files": [
    "lib/foo.js"
  ]
}
```

Informationen zu wichtigen Eigenschaften:

```
name
```

Der eindeutige Name Ihres Pakets sollte in Kleinbuchstaben angegeben werden. Diese Eigenschaft ist erforderlich und Ihr Paket kann nicht ohne installiert werden.

1. Der Name darf höchstens 214 Zeichen umfassen.
2. Der Name darf nicht mit einem Punkt oder Unterstrich beginnen.
3. Neue Pakete dürfen keine Großbuchstaben enthalten.

```
version
```

Die Version des Pakets wird durch Semantic Versioning (Semver) angegeben. Dabei wird davon ausgegangen, dass eine Versionsnummer als MAJOR.MINOR.PATCH geschrieben wird und Sie das Inkrement erhöhen:

1. MAJOR-Version, wenn Sie inkompatible API-Änderungen vornehmen
2. MINOR-Version, wenn Sie Funktionalität rückwärtskompatibel hinzufügen
3. PATCH-Version, wenn Sie rückwärtskompatible Fehlerbehebungen vornehmen

```
description
```

Die Beschreibung des Projekts. Versuchen Sie es kurz und prägnant zu halten.

```
author
```

Der Autor dieses Pakets.

```
bin
```

Ein Objekt, mit dem Binärskripts aus Ihrem Paket angezeigt werden. Das Objekt nimmt an, dass der Schlüssel der Name des Binärskripts und der Wert ein relativer Pfad zum Skript ist.

Diese Eigenschaft wird von Paketen verwendet, die eine CLI (Befehlszeilenschnittstelle) enthalten.

```
script
```

Ein Objekt, das zusätzliche npm-Befehle verfügbar macht. Das Objekt nimmt an, dass der Schlüssel der Befehl npm ist und der Wert der Skriptpfad ist. Diese Skripts können ausgeführt werden, wenn Sie `npm run {command name}` oder `npm run-script {command name}` .

Pakete, die eine Befehlszeilenschnittstelle enthalten und lokal installiert sind, können ohne relativen Pfad aufgerufen werden. Anstatt `./node-modules/.bin/mocha` , können Sie also direkt `mocha` aufrufen.

```
main
```

Der Haupteinstiegspunkt für Ihr Paket. Beim Aufruf von `require('{module name}')` im Knoten handelt es sich um die tatsächlich benötigte Datei.

Es wird dringend empfohlen, dass das Anfordern der Hauptdatei keine Nebeneffekte erzeugt. Das Anfordern der Hauptdatei sollte beispielsweise keinen HTTP-Server starten oder eine Verbindung zu einer Datenbank herstellen. Stattdessen sollten Sie in Ihrem Hauptskript etwas wie `exports.init = function () {...}` erstellen.

```
keywords
```

Eine Reihe von Keywords, die Ihr Paket beschreiben. Diese helfen den Leuten, Ihr Paket zu finden.

```
devDependencies
```

Dies sind die Abhängigkeiten, die nur für die Entwicklung und den Test Ihres Moduls vorgesehen sind. Die Abhängigkeiten werden automatisch installiert, sofern nicht die `NODE_ENV=production` festgelegt wurde. Wenn dies der Fall ist, können Sie diese Pakete weiterhin mit `npm install --dev`

```
peerDependencies
```

Wenn Sie dieses Modul verwenden, listet peerDependencies die Module auf, die Sie neben diesem Modul installieren müssen. Zum Beispiel muss `moment-timezone` neben `moment` installiert `moment` da es sich um ein Plugin für Moment handelt, auch wenn es nicht direkt `require("moment")` .

```
preferGlobal
```

Eine Eigenschaft, die angibt, dass diese Seite die globale `npm install -g {module-name}` mit `npm install -g {module-name}` . Diese Eigenschaft wird von Paketen verwendet, die eine CLI (Befehlszeilenschnittstelle) enthalten.

In allen anderen Situationen sollten Sie diese Eigenschaft NICHT verwenden.

```
publishConfig
```

Die publishConfig ist ein Objekt mit Konfigurationswerten, die zum Veröffentlichen von Modulen verwendet werden. Die eingestellten Konfigurationswerte setzen Ihre Standard-npm-Konfiguration außer Kraft.

Die häufigste Verwendung von `publishConfig` ist das Veröffentlichen des Pakets in einer privaten npm-Registrierung, sodass Sie weiterhin die Vorteile von npm nutzen können, jedoch nicht für private Pakete. Dazu legen Sie einfach die URL Ihrer privaten npm als Wert für den Registrierungsschlüssel fest.

```
files
```

Dies ist ein Array aller Dateien, die in das veröffentlichte Paket aufgenommen werden sollen. Es kann entweder ein Dateipfad oder ein Ordnerpfad verwendet werden. Der gesamte Inhalt eines Ordnerpfads wird eingeschlossen. Dies reduziert die Gesamtgröße Ihres Pakets, indem nur die korrekten Dateien zum Verteilen eingeschlossen werden. Dieses Feld arbeitet mit einer `.npmignore` .

Quelle

package.json online lesen: https://riptutorial.com/de/node-js/topic/1515/package-json

Kapitel 83: Passintegration

Bemerkungen

Passwort muss **immer** gehasht werden. Eine einfache Möglichkeit, Kennwörter mit **NodeJS** zu sichern, wäre die Verwendung **des** Moduls **bcrypt-nodejs** .

Examples

Fertig machen

Der Passport muss mit der Middleware `passport.initialize()` initialisiert werden. Zur Verwendung von Anmeldesitzungen ist `passport.session()` Middleware erforderlich.

Beachten Sie, dass die Methoden `passport.serialize()` und `passport.deserializeUser()` definiert werden müssen. **Passport** serialisiert und deserialisiert Benutzerinstanzen für die Sitzung

```
const express = require('express');
const session = require('express-session');
const passport = require('passport');
const cookieParser = require('cookie-parser');
const app = express();

// Required to read cookies
app.use(cookieParser());

passport.serializeUser(function(user, next) {
    // Serialize the user in the session
    next(null, user);
});

passport.deserializeUser(function(user, next) {
    // Use the previously serialized user
        next(null, user);
});

// Configuring express-session middleware
app.use(session({
    secret: 'The cake is a lie',
    resave: true,
    saveUninitialized: true
}));

// Initializing passport
app.use(passport.initialize());
app.use(passport.session());

// Starting express server on port 3000
app.listen(3000);
```

Lokale Authentifizierung

Das **lokale Passport-** Modul wird zur Implementierung einer lokalen Authentifizierung verwendet.

> Mit diesem Modul können Sie sich mit einem Benutzernamen und einem Kennwort in Ihren Node.js-Anwendungen authentifizieren.

Registrieren des Benutzers:

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

// A named strategy is used since two local strategy are used :
// one for the registration and the other to sign-in
passport.use('localSignup', new LocalStrategy({
    // Overriding defaults expected parameters,
    // which are 'username' and 'password'
    usernameField: 'email',
    passwordField: 'password',
    passReqToCallback: true // allows us to pass back the entire request to the callback    .
},
function(req, email, password, next) {
    // Check in database if user is already registered
    findUserByEmail(email, function(user) {
        // If email already exists, abort registration process and
        // pass 'false' to the callback
        if (user) return next(null, false);
        // Else, we create the user
        else {
            // Password must be hashed !
            let newUser = createUser(email, password);

            newUser.save(function() {
                // Pass the user to the callback
                return next(null, newUser);
            });
        }
    });
});
```

Beim Benutzer anmelden:

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

passport.use('localSignin', new LocalStrategy({
    usernameField : 'email',
    passwordField : 'password',
},
function(email, password, next) {
    // Find the user
    findUserByEmail(email, function(user) {
        // If user is not found, abort signing in process
        // Custom messages can be provided in the verify callback
        // to give the user more details concerning the failed authentication
        if (!user)
            return next(null, false, {message: 'This e-mail address is not associated with any
account.'});
        // Else, we check if password is valid
        else {
```

```
            // If password is not correct, abort signing in process
            if (!isPasswordValid(password)) return next(null, false);
            // Else, pass the user to callback
            else return next(null, user);
        }
    });
});
```

Routen erstellen:

```
// ...
app.use(passport.initialize());
app.use(passport.session());

// Sign-in route
// Passport strategies are middlewares
app.post('/login', passport.authenticate('localSignin', {
    successRedirect: '/me',
    failureRedirect: '/login'
});

// Sign-up route
app.post('/register', passport.authenticate('localSignup', {
    successRedirect: '/',
    failureRedirect: '/signup'
});

// Call req.logout() to log out
app.get('/logout', function(req, res) {
    req.logout();
    res.redirect('/');
});

app.listen(3000);
```

Facebook-Authentifizierung

Das **Pass-Facebook-** Modul wird zur Implementierung einer **Facebook-** Authentifizierung verwendet. Wenn der Benutzer bei der Anmeldung nicht vorhanden ist, wird er erstellt.

Umsetzungsstrategie:

```
const passport = require('passport');
const FacebookStrategy = require('passport-facebook').Strategy;

// Strategy is named 'facebook' by default
passport.use({
    clientID: 'yourclientid',
    clientSecret: 'yourclientsecret',
    callbackURL: '/auth/facebook/callback'
},
// Facebook will send a token and user's profile
function(token, refreshToken, profile, next) {
    // Check in database if user is already registered
    findUserByFacebookId(profile.id, function(user) {
        // If user exists, returns his data to callback
        if (user) return next(null, user);
```

```
        // Else, we create the user
        else {
            let newUser = createUserFromFacebook(profile, token);

            newUser.save(function() {
                // Pass the user to the callback
                return next(null, newUser);
            });
        }
    });
});
```

Routen erstellen:

```
// ...
app.use(passport.initialize());
app.use(passport.session());

// Authentication route
app.get('/auth/facebook', passport.authenticate('facebook', {
    // Ask Facebook for more permissions
    scope : 'email'
}));

// Called after Facebook has authenticated the user
app.get('/auth/facebook/callback',
    passport.authenticate('facebook', {
        successRedirect : '/me',
        failureRedirect : '/'
}));


//...

app.listen(3000);
```

Einfache Benutzername-Passwort Authentifizierung

In Ihren Routen / index.js

Hier ist der `user` das Modell für das Benutzerschema

```
router.post('/login', function(req, res, next) {
    if (!req.body.username || !req.body.password) {
        return res.status(400).json({
            message: 'Please fill out all fields'
        });
    }

    passport.authenticate('local', function(err, user, info) {
        if (err) {
            console.log("ERROR : " + err);
            return next(err);
        }

        if(user) (
```

```
            console.log("User Exists!")
            //All the data of the user can be accessed by user.x
            res.json({"success" : true});
            return;
        } else {
            res.json({"success" : false});
            console.log("Error" + errorResponse());
            return;
        }
    })(req, res, next);
});
```

Google Passport-Authentifizierung

Wir haben in npm ein einfaches Modul für den Brillenauthentifizierungsnamen **passport-google-oauth20 verfügbar**

Betrachten Sie das folgende Beispiel. In diesem Beispiel wurde ein Ordner erstellt, nämlich config, der die Dateien passport.js und google.js im Stammverzeichnis enthält. In Ihrer app.js enthalten Sie Folgendes

```
var express = require('express');
var session = require('express-session');
var passport = require('./config/passport'); // path where the passport file placed
var app = express();
passport(app);
```

// anderer Code zum Initialisieren des Servers, Fehlerhandle

Fügen Sie in der Datei passport.js im Ordner config den folgenden Code ein

```
var passport = require  ('passport'),
google = require('./google'),
User = require('./../model/user'); // User is the  mongoose model

module.exports = function(app){
  app.use(passport.initialize());
  app.use(passport.session());
  passport.serializeUser(function(user, done){
      done(null, user);
  });
  passport.deserializeUser(function (user, done) {
     done(null, user);
  });
  google();
};
```

In der google.js-Datei im selben Konfigurationsordner finden Sie Folgendes

```
var passport = require('passport'),
GoogleStrategy = require('passport-google-oauth20').Strategy,
User = require('./../model/user');
module.exports = function () {
passport.use(new GoogleStrategy({
        clientID: 'CLIENT ID',
```

```
        clientSecret: 'CLIENT SECRET',
        callbackURL: "http://localhost:3000/auth/google/callback"
    },
    function(accessToken, refreshToken, profile, cb) {
        User.findOne({ googleId : profile.id }, function (err, user) {
            if(err){
                return cb(err, false, {message : err});
            }else {
                if (user != '' && user != null) {
                    return cb(null, user, {message : "User "});
                } else {
                    var username  = profile.displayName.split(' ');
                    var userData = new User({
                        name : profile.displayName,
                        username : username[0],
                        password : username[0],
                        facebookId : '',
                        googleId : profile.id,
                    });
                    // send email to user just in case required to send the newly created
                    // credentails to user for future login without using google login
                    userData.save(function (err, newuser) {
                        if (err) {
                            return cb(null, false, {message : err + " !!! Please try again"});
                        }else{
                            return cb(null, newuser);
                        }
                    });
                }
            }
        });
      }
  ));
};
```

Wenn sich der Benutzer nicht in der DB befindet, erstellen Sie in diesem Beispiel einen neuen Benutzer in DB für die lokale Referenz. Verwenden Sie dazu den Feldnamen googleId im Benutzermodell.

Passintegration online lesen: https://riptutorial.com/de/node-js/topic/7666/passintegration

Kapitel 84: passport.js

Einführung

Passport ist ein beliebtes Berechtigungsmodul für Knoten. In einfachen Worten werden alle Autorisierungsanforderungen Ihrer Benutzer in Ihrer App verarbeitet. Passport unterstützt über 300 Strategien, so dass Sie das Login problemlos mit Facebook / Google oder einem anderen sozialen Netzwerk integrieren können. Die Strategie, die wir hier besprechen werden, ist die Local, bei der Sie einen Benutzer anhand Ihrer eigenen Datenbank registrierter Benutzer (mit Benutzername und Kennwort) authentifizieren.

Examples

Beispiel für LocalStrategy in passport.js

```
var passport = require('passport');
var LocalStrategy = require('passport-local').Strategy;

passport.serializeUser(function(user, done) { //In serialize user you decide what to store in
the session. Here I'm storing the user id only.
  done(null, user.id);
});

passport.deserializeUser(function(id, done) { //Here you retrieve all the info of the user
from the session storage using the user id stored in the session earlier using serialize user.
  db.findById(id, function(err, user) {
    done(err, user);
    });
});

passport.use(new LocalStrategy(function(username, password, done) {
    db.findOne({'username':username},function(err,student){
        if(err)return done(err,{message:message});//wrong roll_number or password;
        var pass_retrieved = student.pass_word;
        bcrypt.compare(password, pass_retrieved, function(err3, correct) {
          if(err3){
            message = [{"msg": "Incorrect Password!"}];
            return done(null,false,{message:message});  // wrong password
          }
          if(correct){
              return done(null,student);
          }
        });
    });
}));

app.use(session({ secret: 'super secret' })); //to make passport remember the user on other
pages too.(Read about session store. I used express-sessions.)
app.use(passport.initialize());
app.use(passport.session());

app.post('/',passport.authenticate('local',{successRedirect:'/users' failureRedirect: '/'}),
    function(req,res,next){
```

```
});
```

passport.js online lesen: https://riptutorial.com/de/node-js/topic/8812/passport-js

Kapitel 85: POST-Anforderung in Node.js behandeln

Bemerkungen

Node.js verwendet Streams , um eingehende Daten zu verarbeiten.

Zitieren aus den Dokumenten,

> Ein Stream ist eine abstrakte Schnittstelle zum Arbeiten mit Streaming-Daten in Node.js. Das Stream-Modul bietet eine Basis-API, mit der sich Objekte, die die Stream-Schnittstelle implementieren, leicht erstellen lassen.

Verwenden Sie zur Verarbeitung im Anforderungshauptteil einer POST-Anforderung das `request` , bei dem es sich um einen lesbaren Stream handelt. Datenströme werden als emittierte `data` auf dem `request` - Objekt.

```
request.on('data', chunk => {
  buffer += chunk;
});
request.on('end', () => {
  // POST request body is now available as `buffer`
});
```

Einfach einen leeren Puffer Zeichenfolge erstellen und die Pufferdaten anhängen , wie es über empfangene `data` Ereignisse.

HINWEIS

1. Pufferdaten auf empfangenen `data` Ereignisse vom Typ Buffer
2. Erstellen Sie eine neue Pufferzeichenfolge, um gepufferte Daten aus Datenereignissen **für jede Anforderung** zu sammeln, dh, erstellen Sie eine `buffer` im Anforderungshandler.

Examples

Beispiel eines node.js-Servers, der nur POST-Anforderungen verarbeitet

```
'use strict';

const http = require('http');

const PORT = 8080;
const server = http.createServer((request, response) => {
  let buffer = '';
  request.on('data', chunk => {
    buffer += chunk;
  });
  request.on('end', () => {
```

```
    const responseString = `Received string ${buffer}`;
    console.log(`Responding with: ${responseString}`);
    response.writeHead(200, "Content-Type: text/plain");
    response.end(responseString);
  });
}).listen(PORT, () => {
  console.log(`Listening on ${PORT}`);
});
```

POST-Anforderung in Node.js behandeln online lesen: https://riptutorial.com/de/node-js/topic/5676/post-anforderung-in-node-js-behandeln

Kapitel 86: PostgreSQL-Integration

Examples

Verbinden Sie sich mit PostgreSQL

`PostgreSQL` Modul npm verwenden.

Abhängigkeit von npm installieren

```
npm  install pg --save
```

Nun müssen Sie eine PostgreSQL-Verbindung erstellen, die Sie später abfragen können.

Angenommen, Database_Name = Studenten, Host = localhost und DB_User = Postgres

```
var pg = require("pg")
var connectionString = "pg://postgres:postgres@localhost:5432/students";
var client = new pg.Client(connectionString);
client.connect();
```

Abfrage mit Verbindungsobjekt

Wenn Sie ein Verbindungsobjekt für die Abfragedatenbank verwenden möchten, können Sie diesen Beispielcode verwenden.

```
var queryString = "SELECT name, age FROM students " ;
var query = client.query(queryString);

query.on("row", (row, result)=> {
result.addRow(row);
});

query.on("end", function (result) {
//LOGIC
});
```

PostgreSQL-Integration online lesen: https://riptutorial.com/de/node-js/topic/7706/postgresql-integration

Kapitel 87: Projektstruktur

Einführung

Die Struktur des nodejs-Projekts wird durch die persönlichen Präferenzen, die Projektarchitektur und die Modulinjektionsstrategie beeinflusst. Auch der ereignisbasierte Arc ', der einen dynamischen Modulinstanzierungsmechanismus verwendet. Um eine MVC-Struktur zu haben, ist es unbedingt erforderlich, den serverseitigen und den clientseitigen Quellcode zu trennen, da der clientseitige Code wahrscheinlich minimiert und an den Browser gesendet wird. Auf der Serverseite oder im Backend wird eine API zur Durchführung von CRUD-Vorgängen bereitgestellt

Bemerkungen

Das obige Projekt verwendet browserify- und vue.js-Module als Anwendungsbasisansicht und Minifikationsbibliotheken. Die Projektstruktur kann sich also geringfügig ändern, je nachdem, welches MVC-Framework Sie verwenden. Das Build-Verzeichnis in public muss den gesamten MVC-Code enthalten. Sie können eine Aufgabe haben, die dies für Sie erledigt.

Examples

Eine einfache nodejs-Anwendung mit MVC und API

- Der erste Hauptunterschied besteht zwischen den dynamisch erzeugten Verzeichnissen, die für das Hosting und den Quellverzeichnissen verwendet werden.
- Die Quellverzeichnisse verfügen je nach Konfigurationsumfang über eine Konfigurationsdatei oder einen Konfigurationsordner. Dazu gehören die Umgebungskonfiguration und die Konfiguration der Geschäftslogik, die Sie in das Konfigurationsverzeichnis einfügen können.

```
|-- Config
       |-- config.json
       |-- appConfig
           |-- pets.config
           |-- payment.config
```

- Nun die wichtigsten Verzeichnisse, in denen wir zwischen dem Server / Backend und den Frontend-Modulen unterscheiden. Die 2 Verzeichnisse *Server* und *Webapp* stellen das Backend bzw. Frontend dar, die wir in ein Quellverzeichnis aufnehmen können. *src* .

  > Sie können für Ihre Server- oder Webappeauswahl unterschiedliche Namen wählen, je nachdem, was für Sie sinnvoll ist. Stellen Sie sicher, dass Sie es nicht zu lang oder zu komplex machen möchten, da es sich um die interne Projektstruktur handelt.

- Innerhalb des *Serververzeichnisses* können Sie den Controller, App.js / index.js, verwenden,

der Ihre Hauptknotenjs-Datei und der Startpunkt ist. Das Serververzeichnis. kann auch das *dto*- Verzeichnis haben, in dem sich alle Datenübertragungsobjekte befinden, die von API-Controllern verwendet werden.

```
|-- server
     |-- dto
         |-- pet.js
         |-- payment.js
     |-- controller
         |-- PetsController.js
         |-- PaymentController.js
     |-- App.js
```

- Das webapp-Verzeichnis kann in zwei große Teile *public* und *mvc unterteilt* werden. Dies wird wiederum durch die von Ihnen gewünschte Build-Strategie beeinflusst. Wir verwenden den Browser, bauen den MVC-Teil von webapp und minimieren den Inhalt des *mvc*-Verzeichnisses einfach.

  | - webapp | - public | - mvc

- Jetzt kann das öffentliche Verzeichnis alle statischen Ressourcen, Bilder, CSS (Sie können auch Saas-Dateien haben) und vor allem die HTML-Dateien enthalten.

```
|-- public
    |-- build  // will contianed minified scripts(mvc)
    |-- images
        |-- mouse.jpg
        |-- cat.jpg
    |-- styles
        |-- style.css
    |-- views
        |-- petStore.html
        |-- paymentGateway.html
        |-- header.html
        |-- footer.html
    |-- index.html
```

- Das *mvc*- Verzeichnis enthält die Front-End-Logik einschließlich der *Modelle* , der *View-Controller* und aller anderen *Utils*- Module, die Sie als Teil der Benutzeroberfläche benötigen. Auch das index.js oder das shell.js, was auch immer Sie benötigen, ist ebenfalls Teil dieses Verzeichnisses.

```
|-- mvc
    |-- controllers
        |-- Dashborad.js
        |-- Help.js
        |-- Login.js
    |-- utils
    |-- index.js
```

Abschließend sieht die gesamte Projektstruktur wie folgt aus. Und eine einfache Build-Aufgabe wie *gulp browserify* minimiert die MVC-Skripts und veröffentlicht sie im *öffentlichen* Verzeichnis. Dieses öffentliche Verzeichnis können wir dann über ***express.use (satic ('public'))*** api als

statische Ressource ***bereitstellen*** .

```
|-- node_modules
|-- src
    |-- server
        |-- controller
        |-- App.js   // node app
    |-- webapp
        |-- public
            |-- styles
            |-- images
            |-- index.html
        |-- mvc
            |-- controller
            |-- shell.js  // mvc shell
|-- config
|-- Readme.md
|-- .gitignore
|-- package.json
```

Projektstruktur online lesen: https://riptutorial.com/de/node-js/topic/9935/projektstruktur

Kapitel 88: Remote-Debugging in Node.JS

Examples

NodeJS-Laufkonfiguration

Um das Remote-Debugging von Node einzurichten, führen Sie einfach den Node-Prozess mit dem Flag `--debug` . Sie können einen Port hinzufügen, auf dem der Debugger ausgeführt werden soll, indem Sie `--debug=<port>` .

Wenn Ihr Knotenprozess startet, sollten Sie die Nachricht sehen

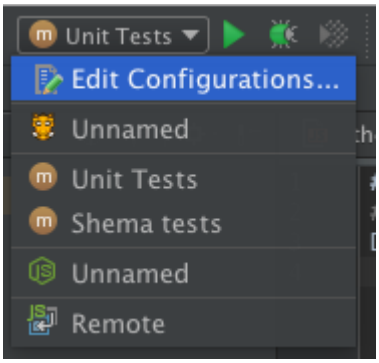
```
Debugger listening on port <port>
```

Was Ihnen sagen wird, dass alles gut ist.

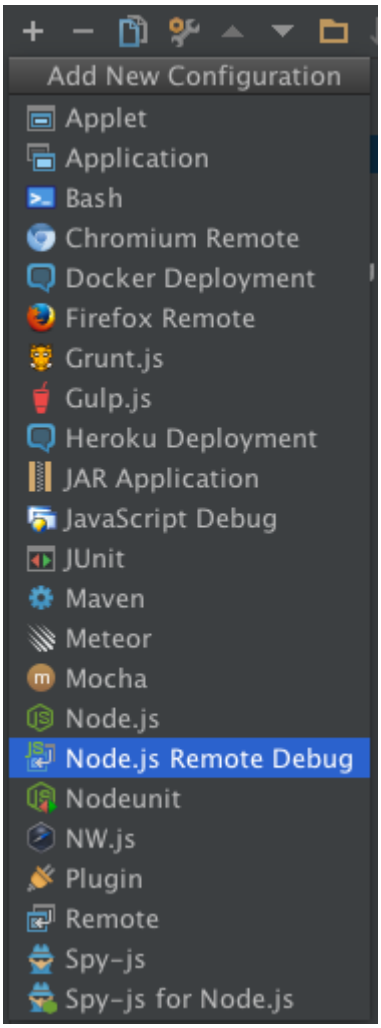
Dann richten Sie das Remote-Debugging-Ziel in Ihrer spezifischen IDE ein.

IntelliJ / Webstorm-Konfiguration

1. Stellen Sie sicher, dass das NodeJS-Plugin aktiviert ist
2. Wählen Sie Ihre Laufkonfigurationen (Bildschirm)



3. Wählen Sie **+** > **Node.js Remote Debug aus**



4. Stellen Sie sicher, dass Sie den oben ausgewählten Port sowie den richtigen Host eingeben

Sobald diese konfiguriert sind, führen Sie einfach das Debug-Ziel wie gewohnt aus, und die Haltepunkte werden angehalten.

Verwenden Sie den Proxy zum Debuggen über den Port unter Linux

Wenn Sie Ihre Anwendung unter Linux starten, verwenden Sie den Proxy zum Debuggen über Port. Beispiel:

```
socat TCP-LISTEN:9958,fork TCP:127.0.0.1:5858 &
```

Verwenden Sie dann den Port 9958 für das Remote-Debugging.

Remote-Debugging in Node.JS online lesen: https://riptutorial.com/de/node-js/topic/6335/remote-debugging-in-node-js

Kapitel 89: Restful API Design: Best Practices

Examples

Fehlerbehandlung: Holen Sie sich alle Ressourcen

Wie gehen Sie mit Fehlern um, anstatt sie an der Konsole zu protokollieren?

Schlechter Weg:

```
Router.route('/')
  .get((req, res) => {
    Request.find((err, r) => {
      if(err){
        console.log(err)
      } else {
        res.json(r)
      }
    })
  })
  .post((req, res) => {
    const request = new Request({
      type: req.body.type,
      info: req.body.info
    });
    request.info.user = req.user._id;
    console.log("ABOUT TO SAVE REQUEST", request);
    request.save((err, r) => {
      if (err) {
        res.json({ message: 'there was an error saving your r' });
      } else {
        res.json(r);
      }
    });
  });
```

Besserer Weg:

```
Router.route('/')
 .get((req, res) => {
  Request.find((err, r) => {
   if(err){
    console.log(err)
  } else {
    return next(err)
  }
})
})
.post((req, res) => {
const request = new Request({
  type: req.body.type,
  info: req.body.info
});
request.info.user = req.user._id;
console.log("ABOUT TO SAVE REQUEST", request);
```

```
request.save((err, r) => {
  if (err) {
    return next(err)
  } else {
    res.json(r);
  }
});
});
```

Restful API Design: Best Practices online lesen: https://riptutorial.com/de/node-js/topic/6490/restful-api-design--best-practices

Kapitel 90: Route-Controller-Service-Struktur für ExpressJS

Examples

Verzeichnisstruktur für Modellrouten, Controller, Dienste

```
├──models
|   ├──user.model.js
├──routes
|   ├──user.route.js
├──services
|   ├──user.service.js
├──controllers
|   ├──user.controller.js
```

Für eine modulare Codestruktur sollte die Logik in diese Verzeichnisse und Dateien unterteilt werden.

> **Modelle** - Die Schemadefinition des Modells
>
> **Routen** - Die API leitet Karten an die Controller weiter
>
> **Controller** - Die Controller handhaben die gesamte Logik hinter der Validierung von Anforderungsparametern, der Abfrage und dem Senden von Antworten mit korrekten Codes.
>
> **Dienste** - Die Dienste enthalten die Datenbankabfragen und das Zurückgeben von Objekten oder das Auslösen von Fehlern

Dieser Codierer schreibt am Ende mehr Codes. Aber am Ende werden die Codes viel einfacher zu pflegen und zu trennen sein.

Code-Struktur für Modellrouten-Controller-Dienste

user.model.js

```
var mongoose = require('mongoose')

const UserSchema  = new mongoose.Schema({
    name: String
})

const User = mongoose.model('User', UserSchema)

module.exports = User;
```

user.routes.js

```
var express = require('express');
var router = express.Router();

var UserController = require('../controllers/user.controller')

router.get('/', UserController.getUsers)

module.exports = router;
```

user.controllers.js

```
var UserService = require('../services/user.service')

exports.getUsers = async function (req, res, next) {
    // Validate request parameters, queries using express-validator

    var page = req.params.page ? req.params.page : 1;
    var limit = req.params.limit ? req.params.limit : 10;
    try {
        var users = await UserService.getUsers({}, page, limit)
        return res.status(200).json({ status: 200, data: users, message: "Succesfully Users
Retrieved" });
    } catch (e) {
        return res.status(400).json({ status: 400, message: e.message });
    }
}
```

user.services.js

```
var User = require('../models/user.model')

exports.getUsers = async function (query, page, limit) {

    try {
        var users = await User.find(query)
        return users;
    } catch (e) {
        // Log Errors
        throw Error('Error while Paginating Users')
    }
}
```

Route-Controller-Service-Struktur für ExpressJS online lesen: https://riptutorial.com/de/node-js/topic/10785/route-controller-service-struktur-fur-expressjs

Kapitel 91: Rückruf an Versprechen

Examples

Rückruf versprechen

Callback-basiert:

```
db.notification.email.find({subject: 'promisify callback'}, (error, result) => {
   if (error) {
       console.log(error);
   }

   // normal code here
});
```

Hierbei wird die promisifyAll-Methode von Bluebird verwendet, um zu versprechen, was herkömmlich auf Callback basierender Code wie oben beschrieben ist. bluebird wird eine Versprechungsversion aller Methoden im Objekt erstellen. An diese Versprechungsbasierten Methodennamen wurde Async angehängt:

```
let email = bluebird.promisifyAll(db.notification.email);

email.findAsync({subject: 'promisify callback'}).then(result => {

    // normal code here
})
.catch(console.error);
```

Wenn nur bestimmte Methoden zugesagt werden müssen, verwenden Sie einfach das Versprechen:

```
let find = bluebird.promisify(db.notification.email.find);

find({locationId: 168}).then(result => {

    // normal code here
});
.catch(console.error);
```

Es gibt einige Bibliotheken (z. B. MassiveJS), die nicht angekündigt werden können, wenn das unmittelbare Objekt der Methode nicht an den zweiten Parameter übergeben wird. Übergeben Sie in diesem Fall einfach das unmittelbare Objekt der Methode, das für den zweiten Parameter angekündigt werden muss, und fügt es in die Kontexteigenschaft ein.

```
let find = bluebird.promisify(db.notification.email.find, { context: db.notification.email });

find({locationId: 168}).then(result => {

    // normal code here
```

```
});
.catch(console.error);
```

Einen Callback manuell versprechen

Manchmal ist es erforderlich, eine Callback-Funktion manuell zu veröffentlichen. Dies kann in einem Fall der Fall sein, in dem der Rückruf nicht dem Standard- Error-First-Format folgt oder wenn zusätzliche Logik erforderlich ist, um zu versprechen:

Beispiel mit fs.exists (Pfad, Rückruf) :

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

var existsAsync = function(path) {
  return new Promise(function(resolve, reject) {
    fs.exists(path, function(exists) {
      // exists is a boolean
      if (exists) {
        // Resolve successfully
        resolve();
      } else {
        // Reject with error
        reject(new Error('path does not exist'));
      }
    });
});

// Use as a promise now
existsAsync('/path/to/some/file').then(function() {
  console.log('file exists!');
}).catch(function(err) {
  // file does not exist
  console.error(err);
});
```

setTimeout versprochen

```
function wait(ms) {
    return new Promise(function (resolve, reject) {
        setTimeout(resolve, ms)
    })
}
```

Rückruf an Versprechen online lesen: https://riptutorial.com/de/node-js/topic/2346/ruckruf-an-versprechen

Kapitel 92: Senden eines Dateistreams an den Client

Examples

Mit fs und pipe statische Dateien vom Server streamen

Ein guter VOD-Dienst (Video On Demand) sollte mit den Grundlagen beginnen. Angenommen, Sie haben ein Verzeichnis auf Ihrem Server, auf das nicht öffentlich zugegriffen werden kann. Sie möchten jedoch über eine Art Portal oder Paywall den Benutzern den Zugriff auf Ihre Medien ermöglichen.

```
var movie = path.resolve('./public/' + req.params.filename);

        fs.stat(movie, function (err, stats) {

            var range = req.headers.range;

            if (!range) {

                return res.sendStatus(416);

            }

            //Chunk logic here
            var positions = range.replace(/bytes=/, "").split("-");
            var start = parseInt(positions[0], 10);
            var total = stats.size;
            var end = positions[1] ? parseInt(positions[1], 10) : total - 1;
            var chunksize = (end - start) + 1;

            res.writeHead(206, {

                'Transfer-Encoding': 'chunked',

                "Content-Range": "bytes " + start + "-" + end + "/" + total,

                "Accept-Ranges": "bytes",

                "Content-Length": chunksize,

                "Content-Type": mime.lookup(req.params.filename)

            });

            var stream = fs.createReadStream(movie, { start: start, end: end, autoClose: true
})

                .on('end', function () {

                    console.log('Stream Done');

                })
```

```
                .on("error", function (err) {
                    res.end(err);
                })
                .pipe(res, { end: true });
        });
```

Das obige Snippet enthält eine grundlegende Übersicht darüber, wie Sie Ihr Video an einen Client streamen möchten. Die Chunk-Logik hängt von verschiedenen Faktoren ab, einschließlich Netzwerkverkehr und Latenz. Es ist wichtig, die Futtergröße gegen die Quantität abzuwägen.

Schließlich kann der .pipe-Aufruf node.js wissen, dass er eine Verbindung zum Server aufrechterhalten und bei Bedarf weitere Chunks senden soll.

Streaming mit fluent-ffmpeg

Sie können flent-ffmpeg auch zum Konvertieren von .mp4-Dateien in .flv-Dateien oder andere Typen verwenden:

res.contentType ('flv');

```
    var pathToMovie = './public/' + req.params.filename;
    var proc = ffmpeg(pathToMovie)
        .preset('flashvideo')
        .on('end', function () {
            console.log('Stream Done');
        })
        .on('error', function (err) {
            console.log('an error happened: ' + err.message);
            res.send(err.message);
        })
        .pipe(res, { end: true });
```

Senden eines Dateistreams an den Client online lesen: https://riptutorial.com/de/node-js/topic/6994/senden-eines-dateistreams-an-den-client

Kapitel 93: Sequelize.js

Examples

Installation

Stellen Sie sicher, dass Sie zuerst Node.js und npm installiert haben. Dann installieren Sie sequelize.js mit npm

```
npm install --save sequelize
```

Sie müssen auch unterstützte Datenbank-Node.js-Module installieren. Sie müssen nur den von Ihnen verwendeten installieren

Für `MYSQL` und `Mariadb`

```
npm install --save mysql
```

Für `PostgreSQL`

```
npm install --save pg pg-hstore
```

Für `SQLite`

```
 npm install --save sqlite
```

Für `MSSQL`

```
npm install --save tedious
```

Nachdem Sie das Setup installiert haben, können Sie eine neue Sequalize-Instanz hinzufügen und erstellen.

ES5-Syntax

```
var Sequelize = require('sequelize');
var sequelize = new Sequelize('database', 'username', 'password');
```

ES6 stage-0 Babel-Syntax

```
import Sequelize from 'sequelize';
const sequelize = new Sequelize('database', 'username', 'password');
```

Sie haben jetzt eine Instanz von Sequelize verfügbar. Sie könnten, wenn Sie sich geneigt fühlen, einen anderen Namen nennen, z

```
var db = new Sequelize('database', 'username', 'password');
```

oder

```
var database = new Sequelize('database', 'username', 'password');
```

dieser Teil ist Ihr Vorrecht. Sobald Sie dies installiert haben, können Sie es in Ihrer Anwendung gemäß der API-Dokumentation http://docs.sequelizejs.com/de/v3/api/sequelize/ verwenden.

Der nächste Schritt nach der Installation besteht darin, ein eigenes Modell einzurichten

Modelle definieren

Es gibt zwei Möglichkeiten, Modelle in Sequelize zu definieren. mit `sequelize.define(...)` oder `sequelize.import(...)` . Beide Funktionen geben ein sequenzielles Modellobjekt zurück.

1. sequelize.define (Modellname, Attribute, [Optionen])

Dies ist der Weg, wenn Sie alle Ihre Modelle in einer Datei definieren möchten oder wenn Sie die Modelldefinition besser steuern möchten.

```
/* Initialize Sequelize */
const config = {
    username: "database username",
    password: "database password",
    database: "database name",
    host: "database's host URL",
    dialect: "mysql" // Other options are postgres, sqlite, mariadb and mssql.
}
var Sequelize = require("sequelize");
var sequelize = new Sequelize(config);

/* Define Models */
sequelize.define("MyModel", {
    name: Sequelize.STRING,
    comment: Sequelize.TEXT,
    date: {
        type: Sequelize.DATE,
        allowNull: false
    }
});
```

Die Dokumentation und weitere Beispiele finden Sie in der Dokumentation zu Doclets oder in der Dokumentation zu sequelize.com .

---

2. sequelize.import (Pfad)

Wenn Ihre Modelldefinitionen für jede in eine Datei unterteilt sind, ist der `import` Ihr Freund. In der Datei, in der Sie Sequelize initialisieren, müssen Sie den Import wie folgt aufrufen:

```
/* Initialize Sequelize */
// Check previous code snippet for initialization

/* Define Models */
sequelize.import("./models/my_model.js"); // The path could be relative or absolute
```

In Ihren Modelldefinitionsdateien sieht Ihr Code dann etwa so aus:

```
module.exports = function(sequelize, DataTypes) {
    return sequelize.define("MyModel", {
        name: DataTypes.STRING,
        comment: DataTypes.TEXT,
        date: {
            type: DataTypes.DATE,
            allowNull: false
        }
    });
};
```

Weitere Informationen zur Verwendung von `import` finden Sie in dem Beispiel für das Sequelize-Beispiel zu GitHub .

Sequelize.js online lesen: https://riptutorial.com/de/node-js/topic/7705/sequelize-js

Kapitel 94: Sicherung von Node.js-Anwendungen

Examples

Verhindern von Cross Site Request Forgery (CSRF)

CSRF ist ein Angriff, der den Endbenutzer zwingt, unerwünschte Aktionen in einer Webanwendung auszuführen, in der er derzeit authentifiziert ist.

Dies kann passieren, weil bei jeder Anfrage Cookies an eine Website gesendet werden - auch wenn diese Anfragen von einer anderen Website stammen.

Wir können das `csurf` Modul verwenden, um ein csrf-Token zu erstellen und es zu überprüfen.

Beispiel

```
var express = require('express')
var cookieParser = require('cookie-parser')    //for cookie parsing
var csrf = require('csurf')    //csrf module
var bodyParser = require('body-parser')    //for body parsing

// setup route middlewares
var csrfProtection = csrf({ cookie: true })
var parseForm = bodyParser.urlencoded({ extended: false })

// create express app
var app = express()

// parse cookies
app.use(cookieParser())

app.get('/form', csrfProtection, function(req, res) {
  // generate and pass the csrfToken to the view
  res.render('send', { csrfToken: req.csrfToken() })
})

app.post('/process', parseForm, csrfProtection, function(req, res) {
  res.send('data is being processed')
})
```

Wenn wir also auf `GET /form` zugreifen, wird das csrf-Token `csrfToken` an die Ansicht übergeben.

Legen Sie nun in der Ansicht den csrfToken-Wert als Wert eines versteckten Eingabefelds mit dem Namen `_csrf` .

zB für `handlebar`

```
<form action="/process" method="POST">
    <input type="hidden" name="_csrf" value="{{csrfToken}}">
    Name: <input type="text" name="name">
```

```
    <button type="submit">Submit</button>
</form>
```

zB für `jade` Templates

```
form(action="/process" method="post")
    input(type="hidden", name="_csrf", value=csrfToken)

    span Name:
    input(type="text", name="name", required=true)
    br

    input(type="submit")
```

zB für `ejs` Vorlagen

```
<form action="/process" method="POST">
    <input type="hidden" name="_csrf" value="<%= csrfToken %>">
    Name: <input type="text" name="name">
    <button type="submit">Submit</button>
</form>
```

SSL / TLS in Node.js

Wenn Sie SSL / TLS in Ihrer Node.js-Anwendung verwenden möchten, müssen Sie an dieser Stelle auch für die Aufrechterhaltung des Schutzes gegen SSL / TLS-Angriffe verantwortlich sein. In vielen Server-Client-Architekturen wird SSL / TLS auf einem Reverse-Proxy beendet, um sowohl die Anwendungskomplexität als auch den Umfang der Sicherheitskonfiguration zu reduzieren.

Wenn Ihre Node.js-Anwendung SSL / TLS verarbeiten soll, kann dies durch Laden der Schlüssel- und Zertifizierungsdateien gesichert werden.

Wenn Ihr Zertifikatsanbieter eine Zertifizierungsstellenkette (Certificate Authority, CA) benötigt, kann er der Option " `ca` " als Array hinzugefügt werden. Eine Kette mit mehreren Einträgen in einer einzigen Datei muss in mehrere Dateien aufgeteilt und in derselben Reihenfolge in das Array eingefügt werden, da Node.js derzeit nicht mehrere ca-Einträge in einer Datei unterstützt. Im folgenden Code finden Sie ein Beispiel für die Dateien `1_ca.crt` und `2_ca.crt` . Wenn das `ca` Array erforderlich und nicht ordnungsgemäß festgelegt ist, zeigen Client-Browser möglicherweise Meldungen an, dass sie die Authentizität des Zertifikats nicht überprüfen konnten.

Beispiel

```
const https = require('https');
const fs = require('fs');

const options = {
  key: fs.readFileSync('privatekey.pem'),
  cert: fs.readFileSync('certificate.pem'),
  ca: [fs.readFileSync('1_ca.crt'), fs.readFileSync('2_ca.crt')]
};
```

```
https.createServer(options, (req, res) => {
  res.writeHead(200);
  res.end('hello world\n');
}).listen(8000);
```

HTTPS verwenden

Das minimale Setup für einen HTTPS-Server in Node.js wäre ungefähr so:

```
const https = require('https');
const fs = require('fs');

const httpsOptions = {
    key: fs.readFileSync('path/to/server-key.pem'),
    cert: fs.readFileSync('path/to/server-crt.pem')
};

const app = function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}

https.createServer(httpsOptions, app).listen(4433);
```

Wenn Sie auch HTTP-Anfragen unterstützen möchten, müssen Sie nur diese kleine Änderung vornehmen:

```
const http = require('http');
const https = require('https');
const fs = require('fs');

const httpsOptions = {
    key: fs.readFileSync('path/to/server-key.pem'),
    cert: fs.readFileSync('path/to/server-crt.pem')
};

const app = function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}

http.createServer(app).listen(8888);
https.createServer(httpsOptions, app).listen(4433);
```

Einrichten eines HTTPS-Servers

Nachdem Sie node.js auf Ihrem System installiert haben, folgen Sie den nachstehenden Anweisungen, um einen einfachen Webserver mit Unterstützung für HTTP und HTTPS zu starten!

Schritt 1: Erstellen Sie eine Zertifizierungsstelle

1. Erstellen Sie den Ordner, in dem Sie Ihren Schlüssel und Ihr Zertifikat speichern möchten:

   ```
   mkdir conf
   ```

2. Gehen Sie zu diesem Verzeichnis:

```
cd conf
```

---

3. `ca.cnf` diese `ca.cnf` Datei, um sie als Konfigurationsverknüpfung zu verwenden:

```
wget https://raw.githubusercontent.com/anders94/https-authorized-clients/master/keys/ca.cnf
```

---

4. Erstellen Sie eine neue Zertifizierungsstelle mit dieser Konfiguration:

```
openssl req -new -x509 -days 9999 -config ca.cnf -keyout ca-key.pem -out ca-cert.pem
```

---

5. Jetzt, da wir unsere Zertifizierungsstelle in `ca-key.pem` und `ca-cert.pem` , generieren wir einen privaten Schlüssel für den Server:

```
openssl genrsa -out key.pem 4096
```

---

6. `server.cnf` diese `server.cnf` Datei, um sie als Konfigurationsverknüpfung zu verwenden:

```
wget https://raw.githubusercontent.com/anders94/https-authorized-
clients/master/keys/server.cnf
```

---

7. Generieren Sie die Zertifikatsignierungsanforderung mit dieser Konfiguration:

```
openssl req -new -config server.cnf -key key.pem -out csr.pem
```

---

8. unterschreibe die Anfrage:

```
openssl x509 -req -extfile server.cnf -days 999 -passin "pass:password" -in csr.pem -CA ca-
cert.pem -CAkey ca-key.pem -CAcreateserial -out cert.pem
```

Schritt 2: Installieren Sie Ihr Zertifikat als Stammzertifikat

1. Kopieren Sie Ihr Zertifikat in den Ordner Ihrer Stammzertifikate:

```
sudo cp ca-crt.pem /usr/local/share/ca-certificates/ca-crt.pem
```

---

2. CA Store aktualisieren:

```
sudo update-ca-certificates
```

Sichere express.js 3 Anwendung

Die Konfiguration zum Herstellen einer sicheren Verbindung mit express.js (Seit Version 3):

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

```
var https = require('https');
var privateKey  = fs.readFileSync('sslcert/server.key', 'utf8');
var certificate = fs.readFileSync('sslcert/server.crt', 'utf8');

// Define your key and cert

var credentials = {key: privateKey, cert: certificate};
var express = require('express');
var app = express();

// your express configuration here

var httpServer = http.createServer(app);
var httpsServer = https.createServer(credentials, app);

// Using port 8080 for http and 8443 for https

httpServer.listen(8080);
httpsServer.listen(8443);
```

Auf diese Weise stellen Sie dem nativen http / https-Server Express-Middleware zur Verfügung

Wenn Sie möchten, dass Ihre App auf Ports unter 1024 ausgeführt wird, müssen Sie den Befehl sudo (nicht empfohlen) oder einen Reverse-Proxy (z. B. nginx, haproxy) verwenden.

Sicherung von Node.js-Anwendungen online lesen: https://riptutorial.com/de/node-js/topic/3473/sicherung-von-node-js-anwendungen

Kapitel 95: Socket.io Kommunikation

Examples

"Hallo Welt!" mit Socketnachrichten.

Installieren Sie Knotenmodule

```
npm install express
npm install socket.io
```

Node.js Server

```
const express = require('express');
const app = express();
const server = app.listen(3000,console.log("Socket.io Hello World server started!"));
const io = require('socket.io')(server);

io.on('connection', (socket) => {
    //console.log("Client connected!");
    socket.on('message-from-client-to-server', (msg) => {
        console.log(msg);
    })
    socket.emit('message-from-server-to-client', 'Hello World!');
});
```

Browser-Client

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Hello World with Socket.io</title>
  </head>
  <body>
    <script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>
    <script>
      var socket = io("http://localhost:3000");
      socket.on("message-from-server-to-client", function(msg) {
          document.getElementById('message').innerHTML = msg;
      });
      socket.emit('message-from-client-to-server', 'Hello World!');
    </script>
    <p>Socket.io Hello World client started!</p>
    <p id="message"></p>
  </body>
</html>
```

Socket.io Kommunikation online lesen: https://riptutorial.com/de/node-js/topic/4261/socket-io-kommunikation

Kapitel 96: Streams verwenden

Parameter

Parameter	Definition
Lesbarer Stream	Typ des Streams, aus dem Daten gelesen werden können
Beschreibbarer Stream	Typ des Streams, in den Daten geschrieben werden können
Duplex-Stream	Typ des Streams, der sowohl lesbar als auch schreibbar ist
Stream umwandeln	Typ des Duplex-Streams, der Daten konvertieren kann, während sie gelesen und dann geschrieben werden

Examples

Daten aus Textdatei mit Streams lesen

Die E / A im Knoten ist asynchron, sodass bei der Interaktion mit der Festplatte und dem Netzwerk Callbacks an Funktionen übergeben werden. Sie könnten versucht sein, Code zu schreiben, der eine Datei von der Festplatte abruft:

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

var server = http.createServer(function (req, res) {
    fs.readFile(__dirname + '/data.txt', function (err, data) {
        res.end(data);
    });
});
server.listen(8000);
```

Dieser Code funktioniert, ist aber sperrig und puffert die gesamte Datei data.txt für jede Anforderung in den Speicher, bevor das Ergebnis an die Clients zurückgeschrieben wird. Wenn data.txt sehr groß ist, kann Ihr Programm viel Speicherplatz beanspruchen, da es gleichzeitig viele Benutzer bedient, insbesondere für Benutzer mit langsamen Verbindungen.

Die Benutzererfahrung ist ebenfalls schlecht, da Benutzer warten müssen, bis die gesamte Datei in den Speicher Ihres Servers gepuffert ist, bevor sie Inhalte empfangen können.

Glücklicherweise sind beide Argumente (req, res) Streams, was bedeutet, dass wir dies mit fs.createReadStream () anstelle von fs.readFile () viel besser schreiben können:

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

```
var server = http.createServer(function (req, res) {
    var stream = fs.createReadStream(__dirname + '/data.txt');
    stream.pipe(res);
});
server.listen(8000);
```

Hier sorgt .pipe () für das Abhören von 'data' und 'end' Ereignissen aus dem fs.createReadStream (). Dieser Code ist nicht nur sauberer, sondern die Datei "data.txt" wird jetzt für jeden einzelnen Block sofort in die Clients geschrieben, sobald sie von der Festplatte empfangen werden.

Piping-Streams

Lesbare Streams können zu "Pipes" oder zu schreibbaren Streams verbunden werden. Dadurch wird der Datenfluss vom Quellstream zum Zielstream ohne großen Aufwand durchgeführt.

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

var readable = fs.createReadStream('file1.txt')
var writable = fs.createWriteStream('file2.txt')

readable.pipe(writable) // returns writable
```

Wenn beschreibbare Streams auch lesbare Streams sind, dh wenn sie *Duplex*- Streams sind, können Sie sie an andere beschreibbare Streams weiterleiten.

```
var zlib = require('zlib')

fs.createReadStream('style.css')
  .pipe(zlib.createGzip()) // The returned object, zlib.Gzip, is a duplex stream.
  .pipe(fs.createWriteStream('style.css.gz')
```

Lesbare Streams können auch in mehrere Streams geleitet werden.

```
var readable = fs.createReadStream('source.css')
readable.pipe(zlib.createGzip()).pipe(fs.createWriteStream('output.css.gz'))
readable.pipe(fs.createWriteStream('output.css')
```

Beachten Sie, dass Sie synchron (gleichzeitig) an die Ausgabeströme weiterleiten müssen, bevor Daten "fließen". Andernfalls können unvollständige Daten gestreamt werden.

Beachten Sie auch, dass Stream-Objekte `error` ausgeben können. Vergewissern Sie sich, dass Sie diese Ereignisse in *jedem* Stream nach Bedarf verantwortungsbewusst behandeln:

```
var readable = fs.createReadStream('file3.txt')
var writable = fs.createWriteStream('file4.txt')
readable.pipe(writable)
readable.on('error', console.error)
writable.on('error', console.error)
```

Erstellen Sie einen eigenen lesbaren / beschreibbaren Stream

Wir werden Stream-Objekte sehen, die von Modulen wie FS usw. zurückgegeben werden. Was aber, wenn wir unser eigenes Streamable-Objekt erstellen möchten?

Um ein Stream-Objekt zu erstellen, müssen Sie das von NodeJs bereitgestellte Stream-Modul verwenden

```
    var fs = require("fs");
    var stream = require("stream").Writable;

    /*
     *  Implementing the write function in writable stream class.
     *  This is the function which will be used when other stream is piped into this
     *  writable stream.
     */
    stream.prototype._write = function(chunk, data){
        console.log(data);
    }

    var customStream = new stream();

    fs.createReadStream("am1.js").pipe(customStream);
```

Dies gibt uns unseren eigenen benutzerdefinierten Stream. wir können alles innerhalb der *_write*-Funktion *implementieren* . Die obige Methode funktioniert in NodeJs 4.xx-Version, aber in NodeJs 6.x **ES6 haben** Klassen eingeführt, weshalb sich die Syntax geändert hat. Unten ist der Code für die 6.x-Version von NodeJs

```
    const Writable = require('stream').Writable;

    class MyWritable extends Writable {
      constructor(options) {
        super(options);
      }

      _write(chunk, encoding, callback) {
        console.log(chunk);
      }
    }
```

Warum Streams?

Schauen wir uns die folgenden zwei Beispiele an, um den Inhalt einer Datei zu lesen:

Die erste, die eine asynchrone Methode zum Lesen einer Datei verwendet und eine Rückruffunktion bereitstellt, die aufgerufen wird, sobald die Datei vollständig in den Speicher eingelesen ist:

```
fs.readFile(`${__dirname}/utils.js`, (err, data) => {
  if (err) {
    handleError(err);
  } else {
    console.log(data.toString());
  }
})
```

Und die zweite, die `streams` verwendet, um den Inhalt der Datei Stück für Stück zu lesen:

```
var fileStream = fs.createReadStream(`${__dirname}/file`);
var fileContent = '';
fileStream.on('data', data => {
  fileContent += data.toString();
})

fileStream.on('end', () => {
  console.log(fileContent);
})

fileStream.on('error', err => {
  handleError(err)
})
```

Es ist erwähnenswert, dass beide Beispiele **dasselbe tun** . Was ist dann der Unterschied?

- Der erste ist kürzer und sieht eleganter aus
- Mit der zweiten können Sie die Datei bearbeiten, **während** sie gelesen wird (!).

Wenn die Dateien, mit denen Sie arbeiten, klein sind, gibt es keinen wirklichen Effekt bei der Verwendung von `streams` , aber was passiert, wenn die Datei groß ist? (so groß, dass es 10 Sekunden dauert, um es in den Speicher zu lesen)

Ohne `streams` Sie warten und absolut nichts tun (es sei denn, Ihr Prozess erledigt anderes), bis die 10 Sekunden vergangen sind und die Datei **vollständig gelesen** ist. Erst dann können Sie die Datei verarbeiten.

Mit `streams` Sie den Inhalt der Datei Stück für Stück, **sobald sie verfügbar sind.** Dadurch können Sie die Datei verarbeiten, **während** sie gelesen wird.

Das obige Beispiel veranschaulicht nicht, wie `streams` für Arbeiten verwendet werden können, die beim Callback-Modus nicht ausgeführt werden können. Schauen wir uns also ein anderes Beispiel an:

Ich möchte eine herunterladen `gzip` - Datei, entpacken und seinen Inhalt auf die Festplatte speichern. In Anbetracht der `url` der Datei müssen Sie `url` tun:

- Laden Sie die Datei herunter
- Entpacke die Datei
- Speichern Sie es auf der Festplatte

Hier ist eine [kleine Datei] [1], die in meinem `S3` Speicher gespeichert ist. Der folgende Code führt das oben auf die Rückrufweise aus.

```
var startTime = Date.now()
s3.getObject({Bucket: 'some-bucket', Key: 'tweets.gz'}, (err, data) => {
  // here, the whole file was downloaded

  zlib.gunzip(data.Body, (err, data) => {
    // here, the whole file was unzipped
```

```
    fs.writeFile(`${__dirname}/tweets.json`, data, err => {
      if (err) console.error(err)

      // here, the whole file was written to disk
      var endTime = Date.now()
      console.log(`${endTime - startTime} milliseconds`) // 1339 milliseconds
    })
  })
})

// 1339 milliseconds
```

So sieht es mit `streams` :

```
s3.getObject({Bucket: 'some-bucket', Key: 'tweets.gz'}).createReadStream()
  .pipe(zlib.createGunzip())
  .pipe(fs.createWriteStream(`${__dirname}/tweets.json`));

// 1204 milliseconds
```

Ja, bei kleinen Dateien geht es nicht schneller - die getestete Datei `80KB` . Das Testen einer größeren Datei mit `71MB` MB (`382MB` MB entpackt) zeigt, dass die `streams` Version viel schneller ist

- Es dauerte 20925 Millisekunden zum Download `71MB` , entpacken und dann schreiben `382MB` auf der Festplatte - **mit der Callback Mode.**
- Im Vergleich dazu dauerte es 13434 Millisekunden, um dasselbe bei Verwendung der `streams` Version zu tun (35% schneller, für eine nicht so große Datei).

Streams verwenden online lesen: https://riptutorial.com/de/node-js/topic/2974/streams-verwenden

Kapitel 97: Synchronous vs. Asynchronous Programmierung in nodejs

Examples

Async verwenden

Das Paket async stellt Funktionen für asynchronen Code bereit.

Mit der Auto- Funktion können Sie asynchrone Beziehungen zwischen zwei oder mehr Funktionen definieren:

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

async.auto({
    get_data: function(callback) {
        console.log('in get_data');
        // async code to get some data
        callback(null, 'data', 'converted to array');
    },
    make_folder: function(callback) {
        console.log('in make_folder');
        // async code to create a directory to store a file in
        // this is run at the same time as getting the data
        callback(null, 'folder');
    },
    write_file: ['get_data', 'make_folder', function(results, callback) {
        console.log('in write_file', JSON.stringify(results));
        // once there is some data and the directory exists,
        // write the data to a file in the directory
        callback(null, 'filename');
    }],
    email_link: ['write_file', function(results, callback) {
        console.log('in email_link', JSON.stringify(results));
        // once the file is written let's email a link to it...
        // results.write_file contains the filename returned by write_file.
        callback(null, {'file':results.write_file, 'email':'user@example.com'});
    }]
}, function(err, results) {
    console.log('err = ', err);
    console.log('results = ', results);
});
```

Dieser Code könnte synchron erstellt worden sein, indem einfach `get_data` , `make_folder` , `write_file` und `email_link` in der richtigen Reihenfolge `email_link` werden. Async verfolgt die Ergebnisse für Sie und wenn ein Fehler aufgetreten ist (erster Parameter des `callback` ungleich `null`), wird die Ausführung der anderen Funktionen angehalten.

Synchronous vs. Asynchronous Programmierung in nodejs online lesen: https://riptutorial.com/de/node-js/topic/8287/synchronous-vs--asynchronous-programmierung-in-nodejs

Kapitel 98: TCP-Sockets

Examples

Ein einfacher TCP-Server

```
// Include Nodejs' net module.
const Net = require('net');
// The port on which the server is listening.
const port = 8080;

// Use net.createServer() in your code. This is just for illustration purpose.
// Create a new TCP server.
const server = new Net.Server();
// The server listens to a socket for a client to make a connection request.
// Think of a socket as an end point.
server.listen(port, function() {
    console.log(`Server listening for connection requests on socket localhost:${port}`.);
});

// When a client requests a connection with the server, the server creates a new
// socket dedicated to that client.
server.on('connection', function(socket) {
    console.log('A new connection has been established.');

    // Now that a TCP connection has been established, the server can send data to
    // the client by writing to its socket.
    socket.write('Hello, client.');

    // The server can also receive data from the client by reading from its socket.
    socket.on('data', function(chunk) {
        console.log(`Data received from client: ${chunk.toString()`.});
    });

    // When the client requests to end the TCP connection with the server, the server
    // ends the connection.
    socket.on('end', function() {
        console.log('Closing connection with the client');
    });

    // Don't forget to catch error, for your own sake.
    socket.on('error', function(err) {
        console.log(`Error: ${err}`);
    });
});
```

Ein einfacher TCP-Client

```
// Include Nodejs' net module.
const Net = require('net');
// The port number and hostname of the server.
const port = 8080;
const host = 'localhost';

// Create a new TCP client.
```

```
const client = new Net.Socket();
// Send a connection request to the server.
client.connect({ port: port, host: host }), function() {
    // If there is no error, the server has accepted the request and created a new
    // socket dedicated to us.
    console.log('TCP connection established with the server.');

    // The client can now send data to the server by writing to its socket.
    client.write('Hello, server.');
});

// The client can also receive data from the server by reading from its socket.
client.on('data', function(chunk) {
    console.log(`Data received from the server: ${chunk.toString()}.`);

    // Request an end to the connection after the data has been received.
    client.end();
});

client.on('end', function() {
    console.log('Requested an end to the TCP connection');
});
```

TCP-Sockets online lesen: https://riptutorial.com/de/node-js/topic/6545/tcp-sockets

Kapitel 99: Umgebung

Examples

Zugriff auf Umgebungsvariablen

Die Eigenschaft `process.env` gibt ein Objekt zurück, das die Benutzerumgebung enthält.

Es gibt ein Objekt wie dieses zurück:

```
{
  TERM: 'xterm-256color',
  SHELL: '/usr/local/bin/bash',
  USER: 'maciej',
  PATH: '~/.bin/:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin',
  PWD: '/Users/maciej',
  EDITOR: 'vim',
  SHLVL: '1',
  HOME: '/Users/maciej',
  LOGNAME: 'maciej',
  _: '/usr/local/bin/node'
}
```

```
process.env.HOME // '/Users/maciej'
```

Wenn Sie die Umgebungsvariable `FOO` auf `foobar` , können Sie auf `foobar` :

```
process.env.FOO // 'foobar'
```

Befehlszeilenargumente von process.argv

process.argv ist ein Array, das die Befehlszeilenargumente enthält. Das erste Element ist `node` , das zweite Element ist der Name der JavaScript-Datei. Die nächsten Elemente sind zusätzliche Befehlszeilenargumente.

Code-Beispiel:

Ausgabesumme aller Befehlszeilenargumente

`index.js`

```
var sum = 0;
for (i = 2; i < process.argv.length; i++) {
    sum += Number(process.argv[i]);
}

console.log(sum);
```

Verwendungsbeispiel:

```
node index.js 2 5 6 7
```

Die Ausgabe wird `20`

Eine kurze Erklärung zum Code:

Hier in der for-Schleife `for (i = 2; i < process.argv.length; i++)` beginnt die Schleife mit 2, da die ersten beiden Elemente im process.argv-Array **immer** `['path/to/node.exe', 'path/to/js/file', ...]`

Konvertierung in number `Number(process.argv[i])` da Elemente im Array process.argv **immer** eine Zeichenfolge sind

Verwenden verschiedener Eigenschaften / Konfigurationen für verschiedene Umgebungen wie dev, qa, staging usw.

Große Anwendungen erfordern oft unterschiedliche Eigenschaften, wenn sie in unterschiedlichen Umgebungen ausgeführt werden. Wir können dies erreichen, indem Sie Argumente an die NodeJs-Anwendung übergeben und dasselbe Argument im Node-Prozess verwenden, um bestimmte Umgebungseigenschaftsdateien zu laden.

Angenommen, wir haben zwei Eigenschaftsdateien für unterschiedliche Umgebungen.

- dev.json

```
{
    PORT : 3000,
    DB : {
        host : "localhost",
        user : "bob",
        password : "12345"
    }
}
```

- qa.json

```
{
        PORT : 3001,
        DB : {
            host : "where_db_is_hosted",
            user : "bob",
            password : "54321"
        }
}
```

Der folgende Code in der Anwendung wird die entsprechende Eigenschaftsdatei exportieren, die wir verwenden möchten.

Angenommen, der Code befindet sich in environment.js

```
process.argv.forEach(function (val, index, array) {
    var arg = val.split("=");
    if (arg.length > 0) {
        if (arg[0] === 'env') {
            var env = require('./' + arg[1] + '.json');
            module.exports = env;
        }
    }
});
```

Wir geben der Bewerbung folgende Argumente

```
node app.js env=dev
```

Wenn wir Prozessmanager *für immer verwenden, dann ist* es so einfach wie

```
forever start app.js env=dev
```

So verwenden Sie die Konfigurationsdatei

```
 var env= require("environment.js");
```

Laden von Umgebungseigenschaften aus einer "Eigenschaftendatei"

- Eigenschaftsleser installieren:

```
npm install properties-reader --save
```

- Erstellen Sie ein **Verzeichnis env** , um Ihre Eigenschaftendateien zu speichern:

```
mkdir env
```

- Erstellen von **environment.js** :

```
process.argv.forEach(function (val, index, array) {
    var arg = val.split("=");
    if (arg.length > 0) {
        if (arg[0] === 'env') {
            var env = require('./env/' + arg[1] + '.properties');
            module.exports = env;
        }
    }
});
```

- Beispiel-Eigenschaftendatei für **development.properties** :

```
# Dev properties
[main]
```

```
# Application port to run the node server
app.port=8080

[database]
# Database connection to mysql
mysql.host=localhost
mysql.port=2500
...
```

- Verwendungsbeispiel der geladenen Eigenschaften:

```
var enviorment = require('./environments');
var PropertiesReader = require('properties-reader');
var properties = new PropertiesReader(enviorment);

var someVal = properties.get('main.app.port');
```

- Express-Server starten

```
npm start env=development
```

oder

```
npm start env=production
```

Umgebung online lesen: https://riptutorial.com/de/node-js/topic/2340/umgebung

Kapitel 100: Unit-Test-Frameworks

Examples

Mokka synchron

```
describe('Suite Name', function() {
  describe('#method()', function() {
    it('should run without an error', function() {
      expect([ 1, 2, 3 ].length).to.be.equal(3)
    })
  })
})
```

Mocha asynchron (Rückruf)

```
var expect = require("chai").expect;
describe('Suite Name', function() {
  describe('#method()', function() {
    it('should run without an error', function(done) {
      testSomething(err => {
        expect(err).to.not.be.equal(null)
        done()
      })
    })
  })
})
```

Mokka asynchron (Versprechen)

```
describe('Suite Name', function() {
  describe('#method()', function() {
    it('should run without an error', function() {
      return doSomething().then(result => {
         expect(result).to.be.equal('hello world')
      })
    })
  })
})
```

Mocha asynchron (async / await)

```
const { expect } = require('chai')

describe('Suite Name', function() {
  describe('#method()', function() {
    it('should run without an error', async function() {
      const result = await answerToTheUltimateQuestion()
      expect(result).to.be.equal(42)
    })
  })
```

```
})
```

Unit-Test-Frameworks online lesen: https://riptutorial.com/de/node-js/topic/6731/unit-test-frameworks

Kapitel 101: Verbinden Sie sich mit Mongodb

Einführung

MongoDB ist ein kostenloses und quelloffenes, open-source-dokumentorientiertes Datenbankprogramm. Als NoSQL-Datenbankprogramm eingestuft, verwendet MongoDB JSON-ähnliche Dokumente mit Schemas.

Weitere Informationen finden Sie unter https://www.mongodb.com/

Syntax

- MongoClient.connect ('mongodb: //127.0.0.1: 27017 / crud', function (err, db) {// tut etwas hier});

Examples

Ein einfaches Beispiel zum Verbinden von MongoDB von Node.JS

```
MongoClient.connect('mongodb://localhost:27017/myNewDB',function (err,db) {
    if(err)
        console.log("Unable to connect DB. Error: " + err)
    else
        console.log('Connected to DB');

    db.close();
});
```

myNewDB ist DB-Name. Wenn es nicht in der Datenbank vorhanden ist, wird es automatisch mit diesem Aufruf erstellt.

Einfache Möglichkeit, MongoDB mit dem Core Node.JS zu verbinden

```
  var MongoClient = require('mongodb').MongoClient;

    //connection with mongoDB
    MongoClient.connect("mongodb://localhost:27017/MyDb", function (err, db) {
          //check the connection
 if(err){
           console.log("connection failed.");
}else{
                    console.log("successfully connected to mongoDB.");
    });
```

Verbinden Sie sich mit Mongodb online lesen: https://riptutorial.com/de/node-js/topic/6280/verbinden-sie-sich-mit-mongodb

Kapitel 102: Verwenden von Browserfiy zum Beheben von "erforderlichen" Fehlern bei Browsern

Examples

Beispiel - file.js

In diesem Beispiel haben wir eine Datei namens **file.js.**

Nehmen wir an, Sie müssen eine URL mit JavaScript und dem NodeJS-Querystring-Modul analysieren.

Dazu müssen Sie nur die folgende Anweisung in Ihre Datei einfügen:

```
const querystring = require('querystring');
var ref = querystring.parse("foo=bar&abc=xyz&abc=123");
```

Was macht dieser Ausschnitt?

Zunächst erstellen wir ein Querystring-Modul, das Dienstprogramme zum Analysieren und Formatieren von URL-Abfragezeichenfolgen bereitstellt. Es ist erreichbar mit:

```
const querystring = require('querystring');
```

Dann analysieren wir eine URL mit der .parse () -Methode. Es analysiert eine URL-Abfragezeichenfolge (str) in eine Sammlung von Schlüssel- und Wertepaaren.

Die Abfragezeichenfolge `'foo=bar&abc=xyz&abc=123'` wird beispielsweise in Folgendes analysiert:

```
{  foo: 'bar',  abc: ['xyz', '123']   }
```

Leider ist in Browsern nicht die *erforderliche* Methode definiert, Node.js jedoch.

Installieren Sie Browserfy

Mit Browserify können Sie schreiben Code, der Anwendungen auf die gleiche Art und Weise *erfordern* , dass Sie es in Knoten verwenden würden. Wie lösen Sie das? Es ist einfach.

1. Installieren Sie zuerst den Knoten, der mit npm ausgeliefert wird. Dann mach:

   npm install -g **browserify**

2. Wechseln Sie in das Verzeichnis, in dem sich Ihre file.js befindet, und installieren Sie unser *Querystring*- Modul mit npm:

   npm install **querystring**

Hinweis: Wenn Sie das betreffende Verzeichnis nicht ändern, schlägt der Befehl fehl, da er die Datei nicht finden kann, die das Modul enthält.

3. Packen Sie jetzt alle erforderlichen Module rekursiv ab file.js in eine einzige Datei namens bundle.js (oder wie immer Sie sie benennen **möchten**), mit dem **Befehl browserify**

   browserify file.js -o bundle.js

Browserify analysiert die Abstract-Syntax-Struktur für Aufforderungen *()* , um den gesamten Abhängigkeitsgraphen Ihrer zu durchlaufen

4. Zum Schluss legen Sie ein einzelnes Tag in Ihre HTML-Datei ein und fertig!

```
<script src="bundle.js"></script>
```

Was passiert, ist, dass Sie eine Kombination aus Ihrer alten .js-Datei (also **Datei.js**) und Ihrer neu erstellten Datei **bundle.js erhalten** . Diese beiden Dateien werden in einer einzigen Datei zusammengeführt.

Wichtig

Bitte beachten Sie, dass Sie Änderungen an Ihrer Datei "file.js" vornehmen möchten und das Verhalten Ihres Programms nicht beeinflusst. **Ihre Änderungen werden nur wirksam, wenn Sie das neu erstellte Bundle.js bearbeiten**

Was bedeutet das?

Das heißt, wenn Sie **file.js** aus irgendwelchen Gründen bearbeiten **möchten** , haben die Änderungen keine Auswirkungen. Sie müssen **bundle.js** wirklich bearbeiten, da es sich um eine Zusammenführung von **bundle.js** und **file.js handelt** .

Verwenden von Browserfiy zum Beheben von "erforderlichen" Fehlern bei Browsern online lesen: https://riptutorial.com/de/node-js/topic/7123/verwenden-von-browserfiy-zum-beheben-von--erforderlichen--fehlern-bei-browsern

Kapitel 103: Verwenden von IISNode zum Hosten von Node.js-Webanwendungen in IIS

Bemerkungen

Virtuelles Verzeichnis / geschachtelte Anwendung mit Ansichtsfall

Wenn Sie Express zum Rendern von Ansichten mit einer View Engine verwenden, müssen Sie den `virtualDirPath` Wert an Ihre Ansichten übergeben

```
`res.render('index', { virtualDirPath: virtualDirPath });`
```

Der Grund dafür ist, dass Sie Hyperlinks zu anderen Ansichten erstellen, die von Ihrer App und statischen Ressourcenpfaden gehostet werden, um zu erfahren, wo die Site gehostet wird, ohne alle Ansichten nach der Bereitstellung ändern zu müssen. Dies ist einer der lästigsten und langwierigsten Fallstricke bei der Verwendung von virtuellen Verzeichnissen mit IISNode.

Versionen

Alle obigen Beispiele funktionieren mit

- Express v4.x
- IIS 7.x / 8.x
- Socket.io v1.3.x oder höher

Examples

Fertig machen

Mit IISNode können Node.js-Webanwendungen auf IIS 7/8 gehostet werden, genau wie dies bei einer .NET-Anwendung der Fall ist. Natürlich können Sie Ihren `node.exe` Prozess unter Windows selbst hosten. Warum sollten Sie dies tun, wenn Sie Ihre App einfach in IIS ausführen können.

IISNode wird handhabt mehrere Kerne, Prozess manageement die Skalierung über `node.exe` und Auto-Recycling Ihre IIS Anwendung , wenn Ihr App aktualisiert wird, um nur einige seiner zu nennen Vorteile .

Bedarf

Für IISNode gelten einige Anforderungen, bevor Sie Ihre Node.js-App in IIS hosten können.

1. Auf dem IIS-Host muss Node.js installiert sein, 32-Bit oder 64-Bit werden unterstützt.
2. Wenn IISNode x86 oder x64 installiert ist, sollte dies mit der Bitgröße Ihres IIS-Hosts übereinstimmen.
3. Das Microsoft URL-Rewrite-Modul für IIS, das auf Ihrem IIS-Host installiert ist.
    - Dies ist der Schlüssel, andernfalls funktionieren Anforderungen an Ihre Node.js-App nicht wie erwartet.
4. Eine `Web.config` im Stammverzeichnis Ihrer Node.js-App.
5. IISNode-Konfiguration über eine `iisnode.yml` Datei oder ein `<iisnode>` -Element in Ihrer `Web.config` .

Grundlegendes Hello World-Beispiel mit Express

Damit dieses Beispiel funktionieren kann, müssen Sie auf Ihrem IIS-Host eine IIS 7/8-App erstellen und das Verzeichnis mit der Node.js-Webanwendung als physisches Verzeichnis hinzufügen. Stellen Sie sicher, dass Ihre Anwendung / Anwendungspool-Identität auf die Installation von Node.js zugreifen kann. In diesem Beispiel wird die 64-Bit-Installation Node.js verwendet.

Projektstruktur

Dies ist die grundlegende Projektstruktur einer IISNode / Node.js-Webanwendung. Es sieht fast identisch mit jeder anderen Webanwendung als IISNode aus, abgesehen von der Hinzufügung von `Web.config` .

```
- /app_root
  - package.json
  - server.js
  - Web.config
```

server.js - Expressanwendung

```
const express = require('express');
const server = express();

// We need to get the port that IISNode passes into us
// using the PORT environment variable, if it isn't set use a default value
const port = process.env.PORT || 3000;

// Setup a route at the index of our app
server.get('/', (req, res) => {
    return res.status(200).send('Hello World');
});

server.listen(port, () => {
    console.log(`Listening on ${port}`);
});
```

Konfiguration & Web.config

Die `Web.config` ist wie jede andere IIS- `Web.config` außer dass die folgenden beiden Dinge vorhanden sein müssen: URL `<rewrite><rules>` und ein IISNode `<handler>` . Beide Elemente sind `<system.webServer>` Elemente des Elements `<system.webServer>` .

Aufbau

Sie können IISNode konfigurieren , indem eine mit `iisnode.yml` Datei oder durch Hinzufügen des `<iisnode>` Element als Kind von `<system.webServer>` in Ihrer `Web.config` . Beide dieser Konfiguration kann mit einem jedoch einen anderen, in diesem Fall in Verbindung verwendet werden, `Web.config` die angeben müssen `iisnode.yml` - Datei **und** alle Konfigurationskonflikte werden die sich aus der `iisnode.yml` Datei statt . Dieses Konfigurationsüberschreiben kann nicht umgekehrt passieren.

IISNode-Handler

Damit IIS wissen kann, dass `server.js` unsere Node.js-Webanwendung enthält, müssen wir dies explizit mitteilen. Dazu können Sie den IISNode `<handler>` zum `<handlers>` Handlers `<handlers>` - Element hinzufügen.

```
<handlers>
  <add name="iisnode" path="server.js" verb="*" modules="iisnode"/>
</handlers>
```

URL-Rewrite-Regeln

Der letzte Teil der Konfiguration besteht darin, sicherzustellen, dass der Verkehr, der für unsere Node.js-App in unserer IIS-Instanz bestimmt ist, zu IISNode geleitet wird. Ohne Regeln zum Umschreiben von `http://<host>/server.js` müssten wir unsere App besuchen, indem Sie zu `http://<host>/server.js` Schlimmer noch, wenn Sie eine von `server.js` bereitgestellte Ressource `server.js` Sie eine `404` . Aus diesem Grund ist das Umschreiben von URLs für IISNode-Webanwendungen erforderlich.

```
<rewrite>
    <rules>
        <!-- First we consider whether the incoming URL matches a physical file in the /public
folder -->
        <rule name="StaticContent" patternSyntax="Wildcard">
            <action type="Rewrite" url="public/{R:0}" logRewrittenUrl="true"/>
            <conditions>
                <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true"/>
            </conditions>
            <match url="*.*"/>
        </rule>

        <!-- All other URLs are mapped to the Node.js application entry point -->
        <rule name="DynamicContent">
```

```
            <conditions>
                <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="True"/>
            </conditions>
            <action type="Rewrite" url="server.js"/>
        </rule>
    </rules>
</rewrite>
```

Dies ist eine funktionierende `Web.config` Datei für dieses Beispiel , für eine 64-Bit-Installation von `Web.config`

Besuchen Sie jetzt Ihre IIS-Site und sehen Sie, wie Ihre Node.js-Anwendung funktioniert.

Verwenden eines virtuellen IIS-Verzeichnisses oder einer geschachtelten Anwendung über

Die Verwendung eines virtuellen Verzeichnisses oder einer geschachtelten Anwendung in IIS ist ein häufiges Szenario, das Sie bei der Verwendung von IISNode am ehesten nutzen möchten.

IISNode bietet keine direkte Unterstützung für virtuelle Verzeichnisse oder geschachtelte Anwendungen über die Konfiguration. Um dies zu erreichen, müssen wir eine Funktion von IISNode nutzen, die nicht Teil der Konfiguration ist und viel weniger bekannt ist. Alle Kinder des `<appSettings>` Elements mit dem `Web.config` werden das hinzugefügt `process.env` Objekt als Eigenschaften des appSetting Schlüssel.

Ermöglicht das Erstellen eines virtuellen Verzeichnisses in unseren `<appSettings>`

```
<appSettings>
  <add key="virtualDirPath" value="/foo" />
</appSettings>
```

In unserer Node.js-App können wir auf die Einstellung von `virtualDirPath` zugreifen

```
console.log(process.env.virtualDirPath); // prints /foo
```

Nun, da wir das Element `<appSettings>` zur Konfiguration verwenden können, können wir dies nutzen und in unserem Servercode verwenden.

```
// Access the virtualDirPath appSettings and give it a default value of '/'
// in the event that it doesn't exist or isn't set
var virtualDirPath = process.env.virtualDirPath || '/';

// We also want to make sure that our virtualDirPath
// always starts with a forward slash
if (!virtualDirPath.startsWith('/', 0))
  virtualDirPath = '/' + virtualDirPath;

// Setup a route at the index of our app
server.get(virtualDirPath, (req, res) => {
    return res.status(200).send('Hello World');
});
```

Wir können den virtualDirPath auch mit unseren statischen Ressourcen verwenden

```
// Public Directory
server.use(express.static(path.join(virtualDirPath, 'public')));
// Bower
server.use('/bower_components', express.static(path.join(virtualDirPath,
'bower_components')));
```

Lass uns das alles zusammenstellen

```
const express = require('express');
const server = express();

const port = process.env.PORT || 3000;

// Access the virtualDirPath appSettings and give it a default value of '/'
// in the event that it doesn't exist or isn't set
var virtualDirPath = process.env.virtualDirPath || '/';

// We also want to make sure that our virtualDirPath
// always starts with a forward slash
if (!virtualDirPath.startsWith('/', 0))
  virtualDirPath = '/' + virtualDirPath;

// Public Directory
server.use(express.static(path.join(virtualDirPath, 'public')));
// Bower
server.use('/bower_components', express.static(path.join(virtualDirPath,
'bower_components')));

// Setup a route at the index of our app
server.get(virtualDirPath, (req, res) => {
    return res.status(200).send('Hello World');
});

server.listen(port, () => {
    console.log(`Listening on ${port}`);
});
```

Socket.io mit IISNode verwenden

Damit Socket.io mit IISNode arbeiten kann, sind die einzigen Änderungen, die erforderlich sind, wenn kein virtuelles Verzeichnis / geschachtelte Anwendung verwendet wird, in der `Web.config` .

Da Socket.io Anforderungen sendet, die mit `/socket.io` , muss IISNode IIS mitteilen, dass diese ebenfalls mit IISNode behandelt werden sollten und nicht nur statische Dateianforderungen oder anderer Datenverkehr sind. Dies erfordert einen anderen `<handler>` als Standard-IISNode-Apps.

```
<handlers>
    <add name="iisnode-socketio" path="server.js" verb="*" modules="iisnode" />
</handlers>
```

Zusätzlich zu den Änderungen an den `<handlers>` wir eine zusätzliche URL-Umschreibregel hinzufügen. Die Umschreibungsregel sendet den gesamten `/socket.io` an unsere Server-Datei, auf der der Socket.io-Server ausgeführt wird.

```
<rule name="SocketIO" patternSyntax="ECMAScript">
    <match url="socket.io.+"/>
    <action type="Rewrite" url="server.js"/>
</rule>
```

Wenn Sie IIS 8 verwenden, müssen Sie Ihre webSockets-Einstellung in Ihrer `Web.config` deaktivieren und zusätzlich die obigen `Web.config` und Umschreiberegeln hinzufügen. Dies ist in IIS 7 nicht erforderlich, da es keine Unterstützung für webSocket gibt.

```
<webSocket enabled="false" />
```

Verwenden von IISNode zum Hosten von Node.js-Webanwendungen in IIS online lesen: https://riptutorial.com/de/node-js/topic/6003/verwenden-von-iisnode-zum-hosten-von-node-js-webanwendungen-in-iis

Kapitel 104: Vorlagen-Frameworks

Examples

Nunjucks

Serverseitige Engine mit Blockvererbung, Autoescaping, Makros, asynchroner Steuerung und mehr. Sehr inspiriert von jinja2, sehr ähnlich zu Twig (php).

Dokumente - http://mozilla.github.io/nunjucks/
Installiere - `npm i nunjucks`

Grundlegende Verwendung mit *Express* unten.

app.js

```
var express = require ('express');
var nunjucks  = require('nunjucks');

var app = express();
app.use(express.static('/public'));

// Apply nunjucks and add custom filter and function (for example).
var env = nunjucks.configure(['views/'], { // set folders with templates
    autoescape: true,
    express: app
});
env.addFilter('myFilter', function(obj, arg1, arg2) {
    console.log('myFilter', obj, arg1, arg2);
    // Do smth with obj
    return obj;
});
env.addGlobal('myFunc', function(obj, arg1) {
    console.log('myFunc', obj, arg1);
    // Do smth with obj
    return obj;
});

app.get('/', function(req, res){
    res.render('index.html', {title: 'Main page'});
});

app.get('/foo', function(req, res){
    res.locals.smthVar = 'This is Sparta!';
    res.render('foo.html', {title: 'Foo page'});
});

app.listen(3000, function() {
    console.log('Example app listening on port 3000...');
});
```

/views/index.html

```
<html>
<head>
    <title>Nunjucks example</title>
</head>
<body>
{% block content %}
{{title}}
{% endblock %}
</body>
</html>
```

/views/foo.html

```
{% extends "index.html" %}

{# This is comment #}
{% block content %}
    <h1>{{title}}</h1>
    {# apply custom function and next build-in and custom filters #}
    {{ myFunc(smthVar) | lower | myFilter(5, 'abc') }}
{% endblock %}
```

Vorlagen-Frameworks online lesen: https://riptutorial.com/de/node-js/topic/5885/vorlagen-frameworks

Kapitel 105: Web Apps mit Express

Einführung

Express ist ein minimales und flexibles Node.js-Webanwendungs-Framework, das robuste Funktionen zum Erstellen von Webanwendungen bietet.

Die offizielle Website von Express ist expressjs.com . Die Quelle ist auf GitHub zu finden.

Syntax

- app.get (Pfad [, Middleware], Rückruf [, Rückruf ...])
- app.put (Pfad [, Middleware], Rückruf [, Rückruf ...])
- app.post (Pfad [, Middleware], Rückruf [, Rückruf ...])
- app ['delete'] (Pfad [, Middleware], Rückruf [, Rückruf ...])
- app.use (Pfad [, Middleware], Rückruf [, Rückruf ...])
- app.use (Rückruf)

Parameter

Parameter	Einzelheiten
`path`	Gibt den Pfadteil oder die URL an, die der angegebene Rückruf verarbeiten soll.
`middleware`	Eine oder mehrere Funktionen, die vor dem Rückruf aufgerufen werden. Im Wesentlichen eine Verkettung mehrerer `callback` . Nützlich für eine spezifischere Behandlung, z. B. für die Autorisierung oder Fehlerbehandlung.
`callback`	Eine Funktion, die zur Verarbeitung von Anforderungen an den angegebenen `path` . Es wird als `callback(request, response, next)` . Hier werden `request` , `response` und `next` beschrieben.
`request`	Ein Objekt, das Details zur HTTP-Anforderung enthält, für die der Callback aufgerufen wird.
`response`	Ein Objekt, das verwendet wird, um anzugeben, wie der Server auf die Anforderung reagieren soll.
`next`	Ein Rückruf, der die Kontrolle an die nächste übereinstimmende Route weiterleitet. Es akzeptiert ein optionales Fehlerobjekt.

Examples

Fertig machen

Sie werden zuerst ein Verzeichnis erstellen müssen, greifen sie in der Shell und installieren Express mit npm , indem Sie `npm install express --save`

Erstellen Sie eine Datei und nennen Sie sie `app.js` Sie den folgenden Code hinzu, der einen neuen Express-Server erstellt und ihm einen Endpunkt (`/ping`) mit der `app.get` Methode `app.get` :

```
const express = require('express');

const app = express();

app.get('/ping', (request, response) => {
    response.send('pong');
});

app.listen(8080, 'localhost');
```

Um Ihr Skript auszuführen, verwenden Sie den folgenden Befehl in Ihrer Shell:

```
> node app.js
```

Ihre Anwendung akzeptiert Verbindungen auf dem Localhost-Port 8080. Wenn das Hostname-Argument für `app.listen` ist, akzeptiert der Server Verbindungen sowohl auf der IP-Adresse des Computers als auch auf Localhost. Wenn der Portwert 0 ist, weist das Betriebssystem einen verfügbaren Port zu.

Sobald Ihr Skript ausgeführt wird, können Sie es in einer Shell testen, um zu bestätigen, dass Sie die erwartete Antwort "pong" vom Server erhalten:

```
> curl http://localhost:8080/ping
pong
```

Sie können auch einen Webbrowser öffnen. Navigieren Sie zur URL http: // localhost: 8080 / ping , um die Ausgabe anzuzeigen

Grundlegendes Routing

Erstellen Sie zuerst eine Express-App:

```
const express = require('express');
const app = express();
```

Dann können Sie Routen wie folgt definieren:

```
app.get('/someUri', function (req, res, next) {})
```

Diese Struktur funktioniert für alle HTTP-Methoden und erwartet einen Pfad als erstes Argument und einen Handler für diesen Pfad, der die Anforderungs- und Antwortobjekte empfängt. Für die grundlegenden HTTP-Methoden sind dies die Routen

```
// GET www.domain.com/myPath
app.get('/myPath', function (req, res, next) {})

// POST www.domain.com/myPath
app.post('/myPath', function (req, res, next) {})

// PUT www.domain.com/myPath
app.put('/myPath', function (req, res, next) {})

// DELETE www.domain.com/myPath
app.delete('/myPath', function (req, res, next) {})
```

Sie können die vollständige Liste der unterstützten Verben hier überprüfen. Wenn Sie dasselbe Verhalten für eine Route und alle HTTP-Methoden definieren möchten, können Sie Folgendes verwenden:

```
app.all('/myPath', function (req, res, next) {})
```

oder

```
app.use('/myPath', function (req, res, next) {})
```

oder

```
app.use('*', function (req, res, next) {})

// * wildcard will route for all paths
```

Sie können Ihre Routendefinitionen für einen einzelnen Pfad verketten

```
app.route('/myPath')
  .get(function (req, res, next) {})
  .post(function (req, res, next) {})
  .put(function (req, res, next) {})
```

Sie können jeder HTTP-Methode auch Funktionen hinzufügen. Sie werden vor dem letzten Rückruf ausgeführt und übernehmen die Parameter (req, res, next) als Argumente.

```
// GET www.domain.com/myPath
app.get('/myPath', myFunction, function (req, res, next) {})
```

Ihre letzten Rückrufe können in einer externen Datei gespeichert werden, um zu vermeiden, dass zu viel Code in eine Datei eingefügt wird:

```
// other.js
exports.doSomething = function(req, res, next) {/* do some stuff */};
```

Und dann in der Datei mit Ihren Routen:

```
const other = require('./other.js');
app.get('/someUri', myFunction, other.doSomething);
```

Dadurch wird Ihr Code viel sauberer.

Informationen aus der Anfrage erhalten

Um Informationen von der anfordernden URL zu erhalten (beachten Sie, dass `req` das Anforderungsobjekt in der Handlerfunktion der Routen ist). Berücksichtigen Sie diese **`/settings/:user_id`** und dieses bestimmte Beispiel **`/settings/32135?field=name`**

```
// get the full path
req.originalUrl // => /settings/32135?field=name

// get the user_id param
req.params.user_id // => 32135

// get the query value of the field
req.query.field // => 'name'
```

Sie können auch Header der Anfrage erhalten

```
req.get('Content-Type')
// "text/plain"
```

Um das Erhalten anderer Informationen zu vereinfachen, können Sie Middlewares verwenden. Um beispielsweise die Body-Informationen der Anfrage zu erhalten, können Sie die Body-Parser-Middleware verwenden, die rohen Request-Body in ein verwendbares Format umwandelt.

```
var app = require('express')();
var bodyParser = require('body-parser');

app.use(bodyParser.json()); // for parsing application/json
app.use(bodyParser.urlencoded({ extended: true })); // for parsing application/x-www-form-
urlencoded
```

Angenommen, eine solche Anfrage

```
PUT /settings/32135
{
  "name": "Peter"
}
```

Sie können auf den veröffentlichten Namen wie folgt zugreifen

```
req.body.name
// "Peter"
```

In ähnlicher Weise können Sie von der Anfrage aus auf Cookies zugreifen, Sie benötigen außerdem eine Middleware wie einen Cookie-Parser

```
req.cookies.name
```

Modulare Expressanwendung

Um die Web-Anwendung für Webanwendungen modular zu gestalten, verwenden Sie Routerfabriken:

Modul:

```
// greet.js
const express = require('express');

module.exports = function(options = {}) { // Router factory
    const router = express.Router();

    router.get('/greet', (req, res, next) => {
        res.end(options.greeting);
    });

    return router;
};
```

Anwendung:

```
// app.js
const express = require('express');
const greetMiddleware = require('./greet.js');

express()
    .use('/api/v1/', greetMiddleware({ greeting:'Hello world' }))
    .listen(8080);
```

Dadurch wird Ihre Anwendung modular, anpassbar und der Code wiederverwendbar.

Wenn Sie auf `http://<hostname>:8080/api/v1/greet` zugreifen, wird `Hello world` ausgegeben

Komplizierteres Beispiel

Beispiel mit Services, die die Vorteile der Middleware-Fabrik zeigen.

Modul:

```
// greet.js
const express = require('express');

module.exports = function(options = {}) { // Router factory
    const router = express.Router();
    // Get controller
    const {service} = options;

    router.get('/greet', (req, res, next) => {
        res.end(
            service.createGreeting(req.query.name || 'Stranger')
        );
    });
```

```
    return router;
};
```

Anwendung:

```
// app.js
const express = require('express');
const greetMiddleware = require('./greet.js');

class GreetingService {
    constructor(greeting = 'Hello') {
        this.greeting = greeting;
    }

    createGreeting(name) {
        return `${this.greeting}, ${name}!`;
    }
}

express()
    .use('/api/v1/service1', greetMiddleware({
        service: new GreetingService('Hello'),
    }))
    .use('/api/v1/service2', greetMiddleware({
        service: new GreetingService('Hi'),
    }))
    .listen(8080);
```

Beim Zugriff auf `http://<hostname>:8080/api/v1/service1/greet?name=World` wird `Hello, World` ausgegeben und auf `http://<hostname>:8080/api/v1/service2/greet?name=World` Die Ausgabe ist `Hi, World` .

Template Engine verwenden

Template Engine verwenden

Mit dem folgenden Code wird Jade als Vorlagenmodul eingerichtet. (Hinweis: Jade wurde seit Dezember 2015 in `pug` umbenannt.)

```
const express = require('express');  //Imports the express module
const app = express();  //Creates an instance of the express module

const PORT = 3000; //Randomly chosen port

app.set('view engine','jade'); //Sets jade as the View Engine / Template Engine
app.set('views','src/views'); //Sets the directory where all the views (.jade files) are
stored.

//Creates a Root Route
app.get('/',function(req, res){
    res.render('index');  //renders the index.jade file into html and returns as a response.
The render function optionally takes the data to pass to the view.
});
```

```
//Starts the Express server with a callback
app.listen(PORT, function(err) {
    if (!err) {
        console.log('Server is running at port', PORT);
    } else {
        console.log(JSON.stringify(err));
    }
});
```

In ähnlicher Weise können auch andere Template Engines wie `Handlebars` (`hbs`) oder `ejs` . Denken Sie daran, auch die Template Engine von `npm install` . Für Lenker verwenden wir das `hbs` Paket, für Jade ein `jade` Paket und für EJS ein `ejs` Paket.

Beispiel für eine EJS-Vorlage

Mit EJS (wie andere Express-Vorlagen) können Sie Servercode ausführen und auf Ihre Servervariablen von Ihrem HTML-Code aus zugreifen.
In EJS wird " `<%` " als Starttag und " `%>` " als Endtag verwendet. Auf als Renderparameter übergebene Variablen kann mit `<%=var_name%>` zugegriffen werden
Zum Beispiel, wenn Sie in Ihrem Servercode ein Array mit Verbrauchsmaterialien haben
Sie können es mit schleifen

```
<h1><%= title %></h1>
   <ul>
<% for(var i=0; i<supplies.length; i++) { %>
    <li>
        <a href='supplies/<%= supplies[i] %>'>
            <%= supplies[i] %>
        </a>
    </li>
<% } %>
```

Wie Sie im Beispiel jedes Mal sehen können, wenn Sie zwischen serverseitigem Code und HTML wechseln, müssen Sie das aktuelle EJS-Tag schließen und später ein neues öffnen. Hier wollten wir ein `li` innerhalb des Befehls `for` erstellen am Ende des `for` und erstellen Sie ein neues Tag nur für die geschweiften Klammern
ein anderes Beispiel
Wenn wir die Standardeinstellung für die Eingabe als Variable auf der Serverseite festlegen möchten, verwenden wir `<%=`
zum Beispiel:

```
 Message:<br>
<input type="text" value="<%= message %>" name="message" required>
```

Hier ist die von Ihrem Server übergebene Nachrichtenvariable der Standardwert Ihrer Eingabe. Bitte beachten Sie, dass EJS eine Ausnahme auslöst, wenn Sie die Nachrichtenvariable nicht von Ihrem Server übergeben haben. Sie können Parameter mit `res.render('index', {message: message});` (für ejs-Datei mit dem Namen index.ejs).

In den EJS-Tags können Sie auch verwenden, `if` , `while` oder mit einem anderen Javascript-

Befehl.

JSON-API mit ExpressJS

```
var express = require('express');
var cors = require('cors'); // Use cors module for enable Cross-origin resource sharing

var app = express();
app.use(cors()); // for all routes

var port = process.env.PORT || 8080;

app.get('/', function(req, res) {
    var info = {
        'string_value': 'StackOverflow',
        'number_value': 8476
    }
    res.json(info);

    // or
    /* res.send(JSON.stringify({
        string_value: 'StackOverflow',
        number_value: 8476
    })) */

  //you can add a status code to the json response
   /* res.status(200).json(info) */
})

app.listen(port, function() {
    console.log('Node.js listening on port ' + port)
})
```

Auf `http://localhost:8080/` Ausgabeobjekt

```
{
    string_value: "StackOverflow",
    number_value: 8476
}
```

Statische Dateien bereitstellen

Beim Erstellen eines Webservers mit Express muss häufig eine Kombination aus dynamischen Inhalten und statischen Dateien bereitgestellt werden.

Beispielsweise können Sie index.html und script.js haben, bei denen es sich um statische Dateien im Dateisystem handelt.

Es ist üblich, Ordner mit dem Namen "public" für statische Dateien zu verwenden. In diesem Fall kann die Ordnerstruktur folgendermaßen aussehen:

```
project root
├── server.js
├── package.json
└── public
```

```
    ├── index.html
    └── script.js
```

So konfigurieren Sie Express für die Bereitstellung statischer Dateien:

```
const express = require('express');
const app = express();

app.use(express.static('public'));
```

Hinweis: Sobald der Ordner konfiguriert ist, sind index.html, script.js und alle Dateien im Ordner "public" unter dem Stammpfad verfügbar (Sie dürfen `/public/` in der URL angeben). Dies liegt daran, dass express relativ zu dem konfigurierten statischen Ordner nach Dateien sucht. Sie können das *virtuelle Pfadpräfix* wie *folgt* angeben:

```
app.use('/static', express.static('public'));
```

stellt die Ressourcen unter dem Präfix `/static/` .

Mehrere Ordner

Es ist möglich, mehrere Ordner gleichzeitig zu definieren:

```
app.use(express.static('public'));
app.use(express.static('images'));
app.use(express.static('files'));
```

Beim Bereitstellen der Ressourcen prüft Express den Ordner in der Definitionsreihenfolge. Bei Dateien mit demselben Namen wird die Datei im ersten übereinstimmenden Ordner bereitgestellt.

Benannte Routen im Django-Stil

Ein großes Problem ist, dass wertvolle benannte Routen von Express nicht standardmäßig unterstützt werden. Die Lösung besteht darin, ein unterstütztes Paket eines Drittanbieters zu installieren, z. B. Express-Reverse :

```
npm install express-reverse
```

Stecken Sie es in Ihr Projekt:

```
var app = require('express')();
require('express-reverse')(app);
```

Dann benutze es wie:

```
app.get('test', '/hello', function(req, res) {
  res.end('hello');
});
```

Der Nachteil dieses Ansatzes besteht darin, dass Sie das `route` Express-Modul nicht verwenden können, wie unter Erweiterte Router-Verwendung gezeigt . Die Problemumgehung besteht darin, Ihre `app` als Parameter an Ihre Router-Fabrik zu übergeben:

```
require('./middlewares/routing')(app);
```

Und benutze es wie:

```
module.exports = (app) => {
    app.get('test', '/hello', function(req, res) {
      res.end('hello');
    });
};
```

Von nun an können Sie herausfinden, wie Sie Funktionen definieren, um sie mit angegebenen benutzerdefinierten Namespaces zusammenzuführen und auf die entsprechenden Controller verweisen.

Fehlerbehandlung

Grundlegende Fehlerbehandlung

Standardmäßig sucht Express nach einer 'Fehleransicht' im Verzeichnis `/views` , die gerendert werden soll. Erstellen Sie einfach die 'Fehleransicht' und platzieren Sie sie im Verzeichnis 'views', um Fehler zu behandeln. Fehler werden mit der Fehlermeldung, dem Status und dem Stack-Trace geschrieben, zum Beispiel:

Ansichten / error.pug

```
html
  body
      h1= message
      h2= error.status
      p= error.stack
```

Erweiterte Fehlerbehandlung

Definieren Sie Ihre Fehlerbehandlungs-Middleware-Funktionen ganz am Ende des Middleware-Funktionsstapels. Diese haben vier Argumente anstelle von drei `(err, req, res, next)` zum Beispiel:

app.js

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;

    //pass error to the next matching route.
    next(err);
});
```

```
// handle error, print stacktrace
app.use(function(err, req, res, next) {
    res.status(err.status || 500);

    res.render('error', {
        message: err.message,
        error: err
    });
});
```

Sie können, wie bei normalen Middleware-Funktionen, mehrere Middleware-Funktionen zur Fehlerbehandlung definieren.

Mit Middleware und dem nächsten Callback

Express leitet einen `next` Rückruf an alle Routenhandler- und Middleware-Funktionen weiter, mit denen die Logik für einzelne Routen über mehrere Handler hinweg unterbrochen werden kann. Wenn Sie `next()` ohne Argumente aufrufen, wird express aufgefordert, mit der nächsten übereinstimmenden Middleware oder dem Routenhandler fortzufahren. Wenn Sie `next(err)` mit einem Fehler `next(err)` wird eine Middleware für die Fehlerbehandlung ausgelöst. Wenn Sie `next('route')` aufrufen, werden alle nachfolgenden Middleware auf der aktuellen Route umgangen und zur nächsten passenden Route gesprungen. Dies ermöglicht die Entkopplung der Domänenlogik in wiederverwendbare Komponenten, die eigenständig sind, einfacher zu testen und einfacher zu warten und zu ändern sind.

Mehrere übereinstimmende Routen

Anfragen an `/api/foo` oder `/api/bar` führen den ursprünglichen Handler aus, um das Mitglied nachzuschlagen, und übergeben dann die Kontrolle an den tatsächlichen Handler für jede Route.

```
app.get('/api', function(req, res, next) {
  // Both /api/foo and /api/bar will run this
  lookupMember(function(err, member) {
    if (err) return next(err);
    req.member = member;
    next();
  });
});

app.get('/api/foo', function(req, res, next) {
  // Only /api/foo will run this
  doSomethingWithMember(req.member);
});

app.get('/api/bar', function(req, res, next) {
  // Only /api/bar will run this
  doSomethingDifferentWithMember(req.member);
});
```

Fehlerhandler

Fehlerhandler sind Middleware mit der Signaturfunktion `function(err, req, res, next)` . Sie

können pro Route eingerichtet werden (z. B. `app.get('/foo', function(err, req, res, next)`). `app.get('/foo', function(err, req, res, next)` jedoch ein einzelner Fehlerhandler aus, der eine Fehlerseite darstellt.

```
app.get('/foo', function(req, res, next) {
  doSomethingAsync(function(err, data) {
    if (err) return next(err);
    renderPage(data);
  });
});

// In the case that doSomethingAsync return an error, this special
// error handler middleware will be called with the error as the
// first parameter.
app.use(function(err, req, res, next) {
  renderErrorPage(err);
});
```

Middleware

Jede der oben genannten Funktionen ist eigentlich eine Middleware-Funktion, die immer dann ausgeführt wird, wenn eine Anforderung mit der definierten Route übereinstimmt. Es können jedoch beliebig viele Middleware-Funktionen auf einer einzelnen Route definiert werden. Auf diese Weise kann die Middleware in separaten Dateien definiert und die gemeinsame Logik über mehrere Routen hinweg wiederverwendet werden.

```
app.get('/bananas', function(req, res, next) {
  getMember(function(err, member) {
    if (err) return next(err);
    // If there's no member, don't try to look
    // up data. Just go render the page now.
    if (!member) return next('route');
    // Otherwise, call the next middleware and fetch
    // the member's data.
    req.member = member;
    next();
  });
}, function(req, res, next) {
  getMemberData(req.member, function(err, data) {
    if (err) return next(err);
    // If this member has no data, don't bother
    // parsing it. Just go render the page now.
    if (!data) return next('route');
    // Otherwise, call the next middleware and parse
    // the member's data. THEN render the page.
    req.member.data = data;
    next();
  });
}, function(req, res, next) {
  req.member.parsedData = parseMemberData(req.member.data);
  next();
});

app.get('/bananas', function(req, res, next) {
  renderBananas(req.member);
});
```

In diesem Beispiel befindet sich jede Middleware-Funktion entweder in einer eigenen Datei oder in einer anderen Stelle in der Datei, sodass sie in anderen Routen wiederverwendet werden kann.

Fehlerbehandlung

Grundlegende Dokumente finden Sie **hier**

```
app.get('/path/:id(\\d+)', function (req, res, next) { // please note: "next" is passed
    if (req.params.id == 0) // validate param
        return next(new Error('Id is 0')); // go to first Error handler, see below

    // Catch error on sync operation
    var data;
    try {
        data = JSON.parse('/file.json');
    } catch (err) {
        return next(err);
    }

    // If some critical error then stop application
    if (!data)
        throw new Error('Smth wrong');

    // If you need send extra info to Error handler
    // then send custom error (see Appendix B)
    if (smth)
        next(new MyError('smth wrong', arg1, arg2))

    // Finish request by res.render or res.end
    res.status(200).end('OK');
});

// Be sure: order of app.use have matter
// Error handler
app.use(function(err, req, res, next)) {
    if (smth-check, e.g. req.url != 'POST')
        return next(err); // go-to Error handler 2.

    console.log(req.url, err.message);

    if (req.xhr) // if req via ajax then send json else render error-page
        res.json(err);
    else
        res.render('error.html', {error: err.message});
});

// Error handler 2
app.use(function(err, req, res, next)) {
    // do smth here e.g. check that error is MyError
    if (err instanceof MyError) {
        console.log(err.message, err.arg1, err.arg2);
    }
    ...
    res.end();
});
```

Anhang A

```
// "In Express, 404 responses are not the result of an error,
// so the error-handler middleware will not capture them."
// You can change it.
app.use(function(req, res, next) {
    next(new Error(404));
});
```

Anhang B

```
// How to define custom error
var util = require('util');
...
function MyError(message, arg1, arg2) {
    this.message = message;
    this.arg1 = arg1;
    this.arg2 = arg2;
    Error.captureStackTrace(this, MyError);
}
util.inherits(MyError, Error);
MyError.prototype.name = 'MyError';
```

Hook: Wie wird Code vor einer Anforderung und nach einer Res ausgeführt?

`app.use()` und Middleware für „vor“ und eine Kombination der verwendet werden schließen und beenden Ereignisse kann für „nach“ verwendet werden.

```
app.use(function (req, res, next) {
    function afterResponse() {
        res.removeListener('finish', afterResponse);
        res.removeListener('close', afterResponse);

        // actions after response
    }
    res.on('finish', afterResponse);
    res.on('close', afterResponse);

    // action before request
    // eventually calling `next()`
    next();
});
...
app.use(app.router);
```

Ein Beispiel dafür ist die Logger- Middleware, die standardmäßig nach der Antwort an das Protokoll angehängt wird.

`app.router` nur sicher, dass diese "Middleware" vor dem `app.router` da die Reihenfolge von Bedeutung ist.

Ursprünglicher Beitrag ist **hier**

POST-Anfragen bearbeiten

Genau wie Sie Get-Anforderungen in Express mit der app.get-Methode abwickeln, können Sie die app.post-Methode verwenden, um Post-Anforderungen zu bearbeiten.

Bevor Sie jedoch mit POST-Anforderungen umgehen können, müssen Sie die `body-parser` Middleware verwenden. Es analysiert einfach den Hauptteil von `POST` , `PUT` , `DELETE` und anderen Anforderungen.

`Body-Parser` Middleware analysiert den Hauptteil der Anforderung und `req.body` sie in ein in `req.body` verfügbares Objekt `req.body`

```
var bodyParser = require('body-parser');

const express = require('express');

const app = express();

// Parses the body for POST, PUT, DELETE, etc.
app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

app.post('/post-data-here', function(req, res, next){

    console.log(req.body); // req.body contains the parsed body of the request.

});

app.listen(8080, 'localhost');
```

Cookies mit Cookie-Parser setzen

Im Folgenden finden Sie ein Beispiel für das Setzen und Lesen von Cookies mithilfe des Cookie-Parser- Moduls:

```
var express = require('express');
var cookieParser = require('cookie-parser'); // module for parsing cookies
var app = express();
app.use(cookieParser());

app.get('/setcookie', function(req, res){
    // setting cookies
    res.cookie('username', 'john doe', { maxAge: 900000, httpOnly: true });
    return res.send('Cookie has been set');
});

app.get('/getcookie', function(req, res) {
    var username = req.cookies['username'];
    if (username) {
        return res.send(username);
    }

    return res.send('No cookie found');
});

app.listen(3000);
```

Benutzerdefinierte Middleware in Express

In Express können Sie Middlewares definieren, die zum Überprüfen von Anforderungen oder zum Setzen einiger Header als Antwort verwendet werden können.

```
app.use(function(req, res, next){ });    // signature
```

Beispiel

Der folgende Code fügt `user` zum Anforderungsobjekt hinzu und übergibt das Steuerelement an die nächste übereinstimmende Route.

```
var express = require('express');
var app = express();

//each request will pass through it
app.use(function(req, res, next){
    req.user = 'testuser';
    next();    // it will pass the control to next matching route
});

app.get('/', function(req, res){
    var user = req.user;
    console.log(user); // testuser
    return res.send(user);
});

app.listen(3000);
```

Fehlerbehandlung in Express

In Express können Sie einen einheitlichen Fehlerhandler für die Behandlung von Fehlern definieren, die in der Anwendung aufgetreten sind. Definieren Sie dann den Handler am Ende aller Routen und des Logikcodes.

Beispiel

```
var express = require('express');
var app = express();

//GET /names/john
app.get('/names/:name', function(req, res, next){
    if (req.params.name == 'john'){
        return res.send('Valid Name');
    } else{
        next(new Error('Not valid name'));    //pass to error handler
    }
});

//error handler
app.use(function(err, req, res, next){
    console.log(err.stack);    // e.g., Not valid name
    return res.status(500).send('Internal Server Occured');
});
```

```
app.listen(3000);
```

Middleware hinzufügen

Middleware-Funktionen sind Funktionen, die Zugriff auf das Anforderungsobjekt (req), das Antwortobjekt (res) und die nächste Middleware-Funktion im Request-Response-Zyklus der Anwendung haben.

Middleware-Funktionen können beliebigen Code ausführen, Änderungen an `res` und `req` Objekten vornehmen, den Antwortzyklus beenden und die nächste Middleware aufrufen.

Ein bekanntes Beispiel für Middleware ist das `cors` Modul. Um die CORS-Unterstützung hinzuzufügen, installieren Sie sie einfach, fordern Sie sie an und setzen Sie diese Zeile ein:

```
app.use(cors());
```

vor irgendwelchen Routern oder Routing-Funktionen.

Hallo Welt

Hier erstellen wir einen einfachen Hallo-Welt-Server mit Express. Routen:

- '/'
- '/ wiki'

Und für den Rest gibt "404", dh Seite nicht gefunden.

```
'use strict';

const port = process.env.PORT || 3000;

var app = require('express')();
    app.listen(port);

app.get('/',(req,res)=>res.send('HelloWorld!'));
app.get('/wiki',(req,res)=>res.send('This is wiki page.'));
app.use((req,res)=>res.send('404-PageNotFound'));
```

Hinweis: Wir haben 404-Route als letzte Route festgelegt, da Express-Routen in Reihenfolge stapelbar sind und für jede Anforderung nacheinander verarbeitet werden.

Web Apps mit Express online lesen: https://riptutorial.com/de/node-js/topic/483/web-apps-mit-express

Kapitel 106: Webbenachrichtigung senden

Examples

Webbenachrichtigung mit GCM (Google Cloud Messaging System) senden

In diesem Beispiel ist eine weite Verbreitung unter **PWAs** (Progressive Web Applications) bekannt, und in diesem Beispiel werden wir eine einfache Benachrichtigung über das Backend senden, die **NodeJS** und **ES6 verwendet**

1. Node-GCM-Modul `npm install node-gcm` : `npm install node-gcm`
2. Installieren Sie Socket.io: `npm install socket.io`
3. Erstellen Sie eine GCM-fähige Anwendung mithilfe von Google Console.
4. Grabe Ihre GCM-Anwendungs-ID (wird später benötigt)
5. Grabe Ihren GCM-Anwendungsgeheimcode.
6. Öffnen Sie Ihren bevorzugten Code-Editor und fügen Sie den folgenden Code hinzu:

```
'use strict';

const express = require('express');
const app = express();
const gcm = require('node-gcm');
app.io = require('socket.io')();

// [*] Configuring our GCM Channel.
const sender = new gcm.Sender('Project Secret');
const regTokens = [];
let message = new gcm.Message({
    data: {
        key1: 'msg1'
    }
});

// [*] Configuring our static files.
app.use(express.static('public/'));

// [*] Configuring Routes.
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/public/index.html');
});

// [*] Configuring our Socket Connection.
app.io.on('connection', socket => {
    console.log('we have a new connection ...');
    socket.on('new_user', (reg_id) => {
        // [*] Adding our user notification registration token to our list typically
hided in a secret place.
        if (regTokens.indexOf(reg_id) === -1) {
```

```
            regTokens.push(reg_id);

            // [*] Sending our push messages
            sender.send(message, {
                registrationTokens: regTokens
            }, (err, response) => {
                if (err) console.error('err', err);
                else console.log(response);
            });
        }
    })
});

module.exports = app
```

PS: Ich benutze hier einen speziellen Hack, damit Socket.io mit Express funktioniert, weil es einfach nicht außerhalb der Box funktioniert.

Erstellen Sie nun eine **.json-** Datei und benennen Sie sie: **Manifest.json** , öffnen Sie sie und **hinterlassen** Sie Folgendes:

```
{
    "name": "Application Name",
    "gcm_sender_id": "GCM Project ID"
}
```

Schließen Sie es und speichern Sie es in Ihrem **ROOT-** Verzeichnis der Anwendung.

PS: Die Datei Manifest.json muss sich im Stammverzeichnis befinden, sonst funktioniert sie nicht.

Im obigen Code mache ich Folgendes:

1. Ich habe eine normale index.html-Seite eingerichtet und gesendet, die auch socket.io verwenden wird.
2. Ich höre ein **Verbindungsereignis an** , das vom **Frontend aus** (**dh** meine **index.html-Seite)** ausgelöst wird (es wird ausgelöst, wenn ein neuer Client erfolgreich mit unserem vordefinierten Link verbunden ist).
3. Ich sende ein spezielles Token-Know-how als **Registrierungstoken** aus meiner index.html über das socket.io **new_user** -Ereignis. **Dieses** Token ist unser benutzer-eindeutiger Passcode. Jeder Code wird normalerweise von einem unterstützenden Browser für die **Webbenachrichtigungs-API** generiert (lesen Sie mehr Hier.
4. Ich verwende einfach das **node-gcm-** Modul, um meine Benachrichtigung zu senden, die später in **Service Workers** `behandelt und angezeigt wird.

Dies ist aus der Sicht von **NodeJS** . In anderen Beispielen werde ich zeigen, wie wir benutzerdefinierte Daten, Symbole usw. in unserer Push-Nachricht senden können.

PS: Die vollständige Funktionsdemo finden Sie hier.

Webbenachrichtigung senden online lesen: https://riptutorial.com/de/node-js/topic/6333/webbenachrichtigung-senden

Kapitel 107: WebSocket mit Node.JS verwenden

Examples

WebSocket installieren

Es gibt einige Möglichkeiten, WebSocket in Ihrem Projekt zu installieren. Hier sind einige Beispiele:

```
npm install --save ws
```

oder in Ihrem package.json mit:

```
"dependencies": {
  "ws": "*"
},
```

WebSocket's zu Ihren Dateien hinzufügen

Um ws zu Ihrer Datei hinzuzufügen, verwenden Sie einfach:

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

Verwenden von WebSocket und WebSocket Server

Um ein neues WebSocket zu öffnen, fügen Sie einfach Folgendes hinzu:

```
var WebSocket = require("ws");
var ws = new WebSocket("ws://host:8080/OptionalPathName);
// Continue on with your code...
```

Oder um einen Server zu öffnen, verwenden Sie:

```
var WebSocketServer = require("ws").Server;
var ws = new WebSocketServer({port: 8080, path: "OptionalPathName"});
```

Ein einfaches WebSocket Server-Beispiel

```
var WebSocketServer = require('ws').Server
, wss = new WebSocketServer({ port: 8080 }); // If you want to add a path as well, use path:
"PathName"

wss.on('connection', function connection(ws) {
  ws.on('message', function incoming(message) {
```

```
    console.log('received: %s', message);
  });

  ws.send('something');
});
```

WebSocket mit Node.JS verwenden online lesen: https://riptutorial.com/de/node-js/topic/6106/websocket-mit-node-js-verwenden

Kapitel 108: Weiterleiten von Ajax-Anforderungen mit Express.JS

Examples

Eine einfache Implementierung von AJAX

Sie sollten die grundlegende Vorlage für den Express-Generator haben

Fügen Sie in app.js Folgendes hinzu (Sie können es überall nach `var app = express.app()` hinzufügen):

```
app.post(function(req, res, next){
    next();
});
```

Fügen Sie nun in Ihrer index.js-Datei (oder ihrer entsprechenden Übereinstimmung) Folgendes hinzu:

```
router.get('/ajax', function(req, res){
    res.render('ajax', {title: 'An Ajax Example', quote: "AJAX is great!"});
});
router.post('/ajax', function(req, res){
    res.render('ajax', {title: 'An Ajax Example', quote: req.body.quote});
});
```

Erstellen Sie eine Datei `ajax.jade` / `ajax.pug` oder `ajax.ejs` im Verzeichnis `/views` , und fügen Sie `ajax.jade` hinzu:

Für Jade / PugJS:

```
extends layout
script(src="http://code.jquery.com/jquery-3.1.0.min.js")
script(src="/magic.js")
h1 Quote: !{quote}
form(method="post" id="changeQuote")
    input(type='text', placeholder='Set quote of the day', name='quote')
    input(type="submit", value="Save")
```

Für EJS:

```
<script src="http://code.jquery.com/jquery-3.1.0.min.js"></script>
<script src="/magic.js"></script>
<h1>Quote: <%=quote%> </h1>
<form method="post" id="changeQuote">
    <input type="text" placeholder="Set quote of the day" name="quote"/>
    <input type="submit" value="Save">
</form>
```

Erstellen Sie nun in `/public` eine Datei namens `magic.js`

```
$(document).ready(function(){
    $("form#changeQuote").on('submit', function(e){
        e.preventDefault();
        var data = $('input[name=quote]').val();
        $.ajax({
            type: 'post',
            url: '/ajax',
            data: data,
            dataType: 'text'
        })
        .done(function(data){
            $('h1').html(data.quote);
        });
    });
});
```

Und da hast du es! Wenn Sie auf Speichern klicken, ändert sich das Angebot!

Weiterleiten von Ajax-Anforderungen mit Express.JS online lesen: https://riptutorial.com/de/node-js/topic/6738/weiterleiten-von-ajax-anforderungen-mit-express-js

Kapitel 109: Wie werden Module geladen?

Examples

Globaler Modus

Wenn Sie Node mithilfe des Standardverzeichnisses im globalen Modus installiert haben, installiert NPM Pakete in `/usr/local/lib/node_modules` . Wenn Sie in der Shell Folgendes eingeben, sucht NPM nach der neuesten Version des Pakets mit dem Namen sax, lädt sie herunter und installiert sie im Verzeichnis `/usr/local/lib/node_modules/express` :

```
$ npm install -g express
```

Stellen Sie sicher, dass Sie über ausreichende Zugriffsrechte für den Ordner verfügen. Diese Module sind für alle Knotenprozesse verfügbar, die auf dieser Maschine ausgeführt werden

Im lokalen Modus installieren. Npm lädt Module herunter und installiert sie in den aktuellen Arbeitsordnern. Erstellen Sie dazu einen neuen Ordner namens `node_modules` Wenn Sie sich beispielsweise in `/home/user/apps/my_app` ein neuer Ordner mit dem Namen `node_modules` `/home/user/apps/my_app/node_modules` falls noch nicht vorhanden

Module laden

Wenn wir das Modul im Code referenzieren, sucht der Knoten zuerst den Ordner `node_module` innerhalb des referenzierten Ordners in der erforderlichen Anweisung. Wenn der `node_module` nicht relativ ist und kein Kernmodul ist, versucht Node, den Knoten im Ordner `node_modules` im aktuellen Ordner zu finden Verzeichnis. Wenn Sie beispielsweise Folgendes tun, versucht Node, nach der Datei `./node_modules/myModule.js` zu suchen:

```
var myModule = require('myModule.js');
```

Wenn Node die Datei nicht findet, wird sie im übergeordneten Ordner mit dem Namen `../node_modules/myModule.js` . Wenn es erneut fehlschlägt, wird der übergeordnete Ordner ausprobiert und so lange abgesenkt, bis er den Stammordner erreicht oder das erforderliche Modul gefunden hat.

Sie können die Erweiterung `.js` auch weglassen, wenn Sie `.js` In diesem Fall `.js` node die Erweiterung `.js` und sucht nach der Datei.

Laden eines Ordnermoduls

Sie können den Pfad für einen Ordner verwenden, um ein Modul wie folgt zu laden:

```
var myModule = require('./myModuleDir');
```

Wenn Sie dies tun, sucht Node in diesem Ordner. Der Knoten nimmt an, dass dieser Ordner ein Paket ist und versucht, nach einer Paketdefinition zu suchen. Diese `package.json` sollte eine Datei namens `package.json` . Wenn dieser Ordner keine `package.json` mit dem Namen `package.json` , nimmt der `package.json` den Standardwert `index.js` , und Node sucht in diesem Fall nach einer Datei unter dem Pfad `./myModuleDir/index.js` .

Der letzte Ausweg, falls das Modul in keinem der Ordner gefunden wird, ist der globale Modulinstallationsordner.

Wie werden Module geladen? online lesen: https://riptutorial.com/de/node-js/topic/7738/wie-werden-module-geladen-

Kapitel 110: Windows-Authentifizierung unter node.js

Bemerkungen

Es gibt mehrere andere Active Directory-APIS- `activedirectory2` , z. B. `activedirectory2` und `adldap` .

Examples

Aktiviertes Verzeichnis verwenden

Das folgende Beispiel stammt aus den vollständigen Dokumenten, die hier (GitHub) oder hier (NPM) verfügbar sind .

Installation

```
npm install --save activedirectory
```

Verwendungszweck

```
// Initialize
var ActiveDirectory = require('activedirectory');
var config = {
    url: 'ldap://dc.domain.com',
    baseDN: 'dc=domain,dc=com'
};
var ad = new ActiveDirectory(config);
var username = 'john.smith@domain.com';
var password = 'password';
// Authenticate
ad.authenticate(username, password, function(err, auth) {
    if (err) {
        console.log('ERROR: '+JSON.stringify(err));
        return;
    }
    if (auth) {
        console.log('Authenticated!');
    }
    else {
        console.log('Authentication failed!');
    }
});
```

Windows-Authentifizierung unter node.js online lesen: https://riptutorial.com/de/node-

js/topic/10612/windows-authentifizierung-unter-node-js

Kapitel 111: Zeile lesen

Syntax

- const readline = required ('readline')
- readline.close ()
- readline.pause ()
- readline.prompt ([preserveCursor])
- readline.question (Abfrage, Rückruf)
- readline.resume ()
- readline.setPrompt (Eingabeaufforderung)
- readline.write (Daten [, Schlüssel])
- readline.clearLine (Stream, Verzeichnis)
- readline.clearScreenDown (Stream)
- readline.createInterface (Optionen)
- readline.cursorTo (Stream, x, y)
- readline.emitKeypressEvents (Stream [, Schnittstelle])
- readline.moveCursor (Stream, Dx, Dy)

Examples

Zeilenweises Lesen von Dateien

```
const fs = require('fs');
const readline = require('readline');

const rl = readline.createInterface({
    input: fs.createReadStream('text.txt')
});

// Each new line emits an event - every time the stream receives \r, \n, or \r\n
rl.on('line', (line) => {
    console.log(line);
});

rl.on('close', () => {
    console.log('Done reading file');
});
```

Eingabeaufforderung für Benutzer über CLI

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.question('What is your name?', (name) => {
```

```
    console.log(`Hello ${name}!`);

    rl.close();
});
```

Zeile lesen online lesen: https://riptutorial.com/de/node-js/topic/1431/zeile-lesen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Node.js	4444, Abdelaziz Mokhnache, Abhishek Jain, Adam, Aeolingamenfel, Alessandro Trinca Tornidor, Aljoscha Meyer, Amila Sampath, Ankit Gomkale, Ankur Anand, arcs, Aule, B Thuy, baranskistad, Bundit J., Chandra Sekhar, Chezzwizz, Christopher Ronning, Community, Craig Ayre, David Gatti, Djizeus, Florian Hämmerle, Franck Dernoncourt, ganesshkumar, George Aidonidis, Harangue, hexacyanide, Iain Reid, Inanc Gumus, Jason , Jasper, Jeremy Banks, John Slegers, JohnnyCoder, Joshua Kleveter, KolesnichenkoDS, krishgopinath, Léo Martin, Majid, Marek Skiba, Matt Bush, Meinkraft, Michael Irigoyen, Mikhail, Milan Laslop, ndugger, Nick, olegzhermal, Peter Mortensen, RamenChef, Reborn, Rishikesh Chandra, Shabin Hashim, Shiven, Sibeesh Venu, sigfried, SteveLacy, Susanne Oberhauser, thefourtheye, theunexpected1, Tomás Cañibano, user2314737, Volodymyr Sichka, xam, zurfyx
2	Abhängigkeitsspritze	Niroshan Ranapathi
3	Anmutiges Herunterfahren	RamenChef, Sathish
4	Anwendungsfälle von Node.js	vintproykt
5	APIs mit Node.js erstellen	Mukesh Sharma
6	Arduino-Kommunikation mit nodeJs	sBanda
7	Async / Warten	Cami Rodriguez, Cody G., cyanbeam, Dave, David Xu, Dom Vinyard, m_callens, Manuel, nomanbinhussein, Toni Villena
8	async.js	David Knipe, devnull69, DrakaSAN, F. Kauder, jerry, lsampaio, Shriganesh Kolhe, Sky, walid
9	Asynchrone Programmierung	Ala Eddine JEBALI, cyanbeam, Florian Hämmerle, H. Pauwelyn, John, Marek Skiba, Native Coder, omgimanerd, slowdeath007
10	Ausführen von Dateien oder	guleria, hexacyanide, iSkore

	Befehlen mit untergeordneten Prozessen	
11	Ausnahmebehandlung	KlwntSingh, Nivesh, riyadhalnur, sBanda, sjmarshy, topheman
12	Befehlszeilenargumente analysieren	yrtimiD
13	Bei Änderungen automatisch laden	ch4nd4n, Dean Rather, Jonas S, Joshua Kleveter, Nivesh, Sanketh Katta, zurfyx
14	Benötigen()	Philip Cornelius Glover
15	Bereitstellen von Node.js-Anwendungen in der Produktion	Apidcloud, Brett Jackson, Community, Cristian Boariu, duncanhall, Florian Hämmerle, guleria, haykam, KlwntSingh, Mad Scientist, MatthieuLemoine, Mukesh Sharma, raghu, sjmarshy, tverdohleb, tyehia
16	Bereitstellung der Node.js-Anwendung ohne Ausfallzeiten	gentlejo
17	Bluebird Versprechen	David Xu
18	Callback Hölle vermeiden	tyehia
19	Cassandra-Integration	Vsevolod Goloviznin
20	CLI	Ze Rubeus
21	Client-Server-Kommunikation	Zoltán Schmidt
22	Cluster-Modul	Benjamin, Florian Hämmerle, Kid Binary, MayorMonty, Mukesh Sharma, riyadhalnur, Vsevolod Goloviznin
23	CSV-Parser in Knoten js	aisflat439
24	Dateisystem-E / A	4444, Accepted Answer, Aeolingamenfel, Christophe Marois, Craig Ayre, DrakaSAN, Duly Kinsky, Florian Hämmerle, gnerkus, Harshal Bhamare, hexacyanide, jakerella, Julien CROUZET, Louis Barranqueiro, midnightsyntax, Mikhail, peteb, Shiven, still_learning, Tim Jones, Tropic, Vsevolod Goloviznin, Zanon
25	Datei-Upload	Aikon Mogwai, Iceman, Mikhail, walid
26	Datenbank (MongoDB mit Mongoose)	zurfyx

27	Debuggen der Node.js-Anwendung	4444, Alister Norris, Ankur Anand, H. Pauwelyn, Matthew Shanley
28	Deinstallation von Node.js	John Vincent Jardin, RamenChef, snuggles08, Trevor Clarke
29	ECMAScript 2015 (ES6) mit Node.js	David Xu, Florian Hämmerle, Osama Bari
30	Einfache REST-basierte CRUD-API	Iceman
31	Erste Schritte mit der Knotenprofilerstellung	damitj07
32	Erstellen einer Node.js-Bibliothek, die sowohl Versprechen als auch Fehler beim ersten Rückruf unterstützt	Dave
33	Eventloop	Kelum Senanayake
34	Event-Sender	DrakaSAN, Duly Kinsky, Florian Hämmerle, jamescostian, MindlessRanger, Mothman
35	Garnpaket-Manager	Andrew Brooke, skiilaa
36	grunzen	Naeem Shaikh, Waterscroll
37	Guter Codierstil	Ajitej Kaushik, RamenChef
38	Hacken	signal
39	Halten Sie eine Knotenanwendung ständig aktiv	Alex Logan, Bearington, cyanbeam, Himani Agrawal, Mikhail, mscdex, optimus, pietrovismara, RamenChef, Sameer Srivastava, somebody, Taylor Swanson
40	http	Ahmed Metwally
41	Knoten-JS-Lokalisierung	Osama Bari
42	Knotenserver ohne Framework	Hasan A Yousef, Taylor Ackley
43	Koa Framework v2	David Xu
44	Leistungsherausforderungen	Antenka, SteveLacy
45	Liefern Sie HTML oder eine andere Datei	Himani Agrawal, RamenChef, user2314737

46	Lodash	M1kstur
47	Loopback - REST-basierter Anschluss	Roopesh
48	Metallschmied	RamenChef, vsjn3290ckjnaoij2jikndckjb
49	Mit der Konsole interagieren	ScientiaEtVeritas
50	Mitteilungen	Mario Rozic
51	Modul in node.js exportieren und importieren	AndrewLeonardi, Bharat, commonSenseCode, James Billingham, Oliver, sharif.io, Shog9
52	Module exportieren und konsumieren	Aminadav, Craig Ayre, cyanbeam, devnull69, DrakaSAN, Fenton, Florian Hämmerle, hexacyanide, Jason, jdrydn, Loufylouf, Louis Barranqueiro, m02ph3u5, Marek Skiba, MrWhiteNerdy, MSB, Pedro Otero, Shabin Hashim, tkone, uzaif
53	Mongodb-Integration	cyanbeam, FabianCook, midnightsyntax
54	MongoDB-Integration für Node.js / Express.js	William Carron
55	MSSQL-Integration	damitj07
56	Multithreading	arcs
57	Mungo-Bibliothek	Alex Logan, manuerumx, Mikhail, Naeem Shaikh, Qiong Wu, Simplans, Will
58	MySQL-Integration	Aminadav, Andrés Encarnación, Florian Hämmerle, Ivan Schwarz, jdrydn, JohnnyCoder, Kapil Vats, KlwntSingh, Marek Skiba, Rafael Gadotti Bachovas, RamenChef, Simplans, Sorangwala Abbasali, surjikal
59	Mysql-Verbindungspool	KlwntSingh
60	N-API	Parham Alvani
61	Node.js (express.js) mit angle.js Beispielcode	sigfried
62	Node.js als Dienst ausführen	Buzut
63	Node.js Architektur & Inneres	Ivan Hristov
64	Node.js Design	Ankur Anand, pietrovismara

	Fundamental	
65	Node.js installieren	Alister Norris, Aminadav, Anh Cao, asherbar, Batsu, Buzut, Chance Snow, Chezzwizz, Dmitriy Borisov, Florian Hämmerle, GilZ, guleria, hexacyanide, HungryCoder, Inanc Gumus, Jacek Labuda, John Vincent Jardin, Josh, KahWee Teng, Maciej Rostański, mmhyamin, Naing Lin Aung, NuSkooler, Shabin Hashim , Siddharth Srivastva, Sveratum, tandrewnichols, user2314737, user6939352, V1P3R, victorkohl
66	Node.js Leistung	Florian Hämmerle, Inanc Gumus
67	Node.js mit CORS	Buzut
68	Node.JS mit ES6	Inanc Gumus, xam, ymz, zurfyx
69	Node.js mit Oracle	oliolioli
70	Node.JS und MongoDB.	midnightsyntax, RamenChef, Satyam S
71	Node.js v6 Neue Funktionen und Verbesserungen	creyD, DominicValenciana, KlwntSingh
72	Node.js-Code für STDIN und STDOUT, ohne eine Bibliothek zu verwenden	Syam Pradeep
73	Node.js-Fehlerverwaltung	Karlen
74	Nodejs Geschichte	Kelum Senanayake
75	NodeJS mit Redis	evalsocket
76	NodeJs Routing	parlad neupane
77	NodeJS-Anfängerhandbuch	Niroshan Ranapathi
78	NodeJS-Frameworks	dthree
79	npm	Abhishek Jain, AJS, Amreesh Tyagi, Ankur Anand, Asaf Manassen, Ates Goral, ccnokes, CD.., Cristian Cavalli, David G., DrakaSAN, Eric Fortin, Everettss, Explosion Pills, Florian Hämmerle, George Bailey, hexacyanide, HungryCoder, Ionică Bizău, James Taylor, João Andrade, John Slegers, Jojodmo, Josh, Kid Binary, Loufylouf, m02ph3u5, Matt, Matthew Harwood, Mehdi El Fadil, Mikhail, Mindsers, Nick, notgiorgi, num8er, oscarm, Pete TNT, Philipp Flenker, Pieter Herroelen, Pyloid, QoP, Quill, Rafal Wiliński, RamenChef, Ratan

		Kumar, RationalDev, rdegges, refaelos, Rizowski, Shiven, Skanda, Sorangwala Abbasali, still_learning, subbu, the12, tlo, Un3qual, uzaif, VladNeacsu, Vsevolod Goloviznin, Wasabi Fan, Yerko Palma
80	nvm - Knotenversionsmanager	cyanbeam, guleria, John Vincent Jardin, Luis González, pranspach, Shog9, Tushar Gupta
81	OAuth 2.0	tyehia
82	package.json	Ankur Anand, Asaf Manassen, Chance Snow, efeder, Eric Smekens, Florian Hämmerle, Jaylem Chaudhari, Kornel, lauriys, mezzode, OzW, RamenChef, Robbie, Shabin Hashim, Simplans, SteveLacy, Sven 31415, Tomás Cañibano, user6939352, V1P3R, victorkohl
83	Passintegration	Ankit Rana, Community, Léo Martin, M. A. Cordeiro, Rupali Pemare, shikhar bansal
84	passport.js	Red
85	POST-Anforderung in Node.js behandeln	Manas Jayanth
86	PostgreSQL-Integration	Niroshan Ranapathi
87	Projektstruktur	damitj07
88	Remote-Debugging in Node.JS	Rick, VooVoo
89	Restful API Design: Best Practices	fresh5447, nilakantha singh deo
90	Route-Controller-Service-Struktur für ExpressJS	nomanbinhussein
91	Rückruf an Versprechen	Clement JACOB, Michael Buen, Sanketh Katta
92	Senden eines Dateistreams an den Client	Beshoy Hanna
93	Sequelize.js	Fikra, Niroshan Ranapathi, xam
94	Sicherung von Node.js-Anwendungen	akinjide, devnull69, Florian Hämmerle, John Slegers, Mukesh Sharma, Pauly Garcia, Peter G, pranspach, RamenChef, Simplans
95	Socket.io Kommunikation	Forivin, N.J.Dawson

96	Streams verwenden	cyanbeam, Duly Kinsky, efeder, johni, KlwntSingh, Max, Ze Rubeus
97	Synchronous vs. Asynchronous Programmierung in nodejs	Craig Ayre, Veger
98	TCP-Sockets	B Thuy
99	Umgebung	Chris, Freddie Coleman, KlwntSingh, Louis Barranqueiro, Mikhail, sBanda
100	Unit-Test-Frameworks	David Xu, Florian Hämmerle, skiilaa
101	Verbinden Sie sich mit Mongodb	FabianCook, Nainesh Raval, Shriganesh Kolhe
102	Verwenden von Browserfiy zum Beheben von "erforderlichen" Fehlern bei Browsern	Big Dude
103	Verwenden von IISNode zum Hosten von Node.js-Webanwendungen in IIS	peteb
104	Vorlagen-Frameworks	Aikon Mogwai
105	Web Apps mit Express	Aikon Mogwai, Alex Logan, alexi2, Andres C. Viesca, Aph, Asaf Manassen, Batsu, bekce, brianmearns, Community, Craig Ayre, Daniel Verem, devnull69, Everettss, Florian Hämmerle, H. Pauwelyn, Inanc Gumus, jemiloii, Kid Binary, kunerd, Marek Skiba, Mikhail, Mohit Gangrade, Mukesh Sharma, Naeem Shaikh, Niklas, Nivesh, noob, Ojen, Pasha Rumkin, Paul, Rafal Wiliński, Shabin Hashim, SteveLacy, tandrewnichols, Taylor Ackley, themole, tverdohleb, Vsevolod Goloviznin, xims, Yerko Palma
106	Webbenachrichtigung senden	Houssem Yahiaoui
107	WebSocket mit Node.JS verwenden	Rowan Harley
108	Weiterleiten von Ajax-Anforderungen mit Express.JS	RamenChef, SynapseTech

109	Wie werden Module geladen?	RamenChef, umesh
110	Windows-Authentifizierung unter node.js	CJ Harries
111	Zeile lesen	4444, Craig Ayre, Florian Hämmerle, peteb