無料電子ブック

学習

Node.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#node.js

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

1: Node.jsをいめる

Node.jsは、GoogleのV8 JavaScriptエンジンをするイベントベースのブロッキングI / Oフレームワークです。これは、クライアントとサーバーのでJavaScriptをできるため、コードのとコンテキストりえののをけるアプリケーションのにされます。オープンソースとクロスプラットフォームです。 Node.jsアプリケーションはなJavaScriptでかれており、Windows、LinuxなどのNode.jsでできます。

バージョン

バージョン	
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
v7.6.0	2017-02-21
v7.5.0	2017-01-31

バージョン	
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	20161019
v6.9.0	20161018
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
v6.3.1	2016-07-21
v6.3.0	201676

バージョン	
v6.2.2	2016616
v6.2.1	201606-02
v6.2.0	2016517
v6.1.0	2016-05-05
v6.0.0	2016-04-26
v5.12.0	2016623
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
v4.1	2015-09-17
v4.0	2015-09-08

バージョン	
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	2015-06-13
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
v0.8	2012-06-22
v0.7	2012-01-17

バージョン	
v0.6	2011114
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

Hello World HTTPサーバー

まず、このプラットフォームのNode.jsをインストールします。

このでは、 `Hello, World!`をするポート1337でリッスンするHTTPサーバーをします`Hello, World!` ブラウザにします。ポート1337をするわりに、のサービスでされていないのポートをできます。

`http`モジュールはNode.js コアモジュール Node.jsのソースにまれ、のリソースのインストールをとしないモジュールです。 `http`モジュールは、 `http.createServer()`メソッドをしてHTTPサーバーをするをします。アプリケーションをするには、のJavaScriptコードをむファイルをします。

```
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
```

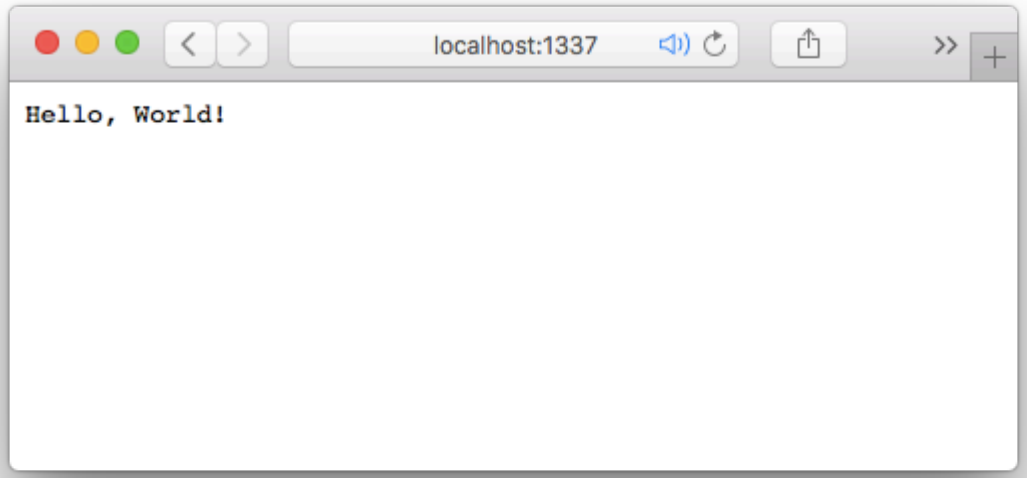
のファイルでファイルをします。この、 `hello.js`というをけると、ファイルがあるディレクトリにしてのコマンドをすることで、アプリケーションをできます。

```
node hello.js
```

されたサーバーは、ブラウザーでURL http// localhost1337またはhttp://127.0.0.1:1337でアクセス

できます。

のスクリーンショットにすように、に「Hello、World」テキストがされたなWebページがされます。



なオンラインの。

Hello Worldコマンドライン

Node.jsは、コマンドラインユーティリティのにもできます。のは、コマンドラインからのをみみ、Helloメッセージをします。

このコードをUnixシステムでするには

1. しいファイルをし、のコードをりけます。ファイルはです。
2. このファイルを`chmod 700 FILE_NAME`にする
3. アプリを`./APP_NAME David`する

Windowsでは、ステップ1をし、 `node APP_NAME David`それをし`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のインストールと

まず、コンピュータにNode.jsをインストールします。

Windows ダウンロードページにし、インストーラをダウンロード/します。

Mac ダウンロードページにし、インストーラをダウンロード/します。または、 `brew install node` をしてHomebrewでNodeをインストールすることもでき`brew install node`。 Homebrewは Macintoshけのコマンドラインパッケージマネージャであり、はHomebrewのWebサイトをしてください 。

Linux コマンドラインのインストールページでディストリビューションのにってください 。

ノードプログラムの

Node.jsプログラムをするには、 `node app.js`または`nodejs app.js`し`node app.js`。 ここで、 `app.js` は、ノードappのソースコードのファイルです。するスクリプトをつけるためにNodeに`.js`をめるはありません。

あるいは、UNIXベースのオペレーティングシステムでは、ノードプログラムをスクリプトとしてすることができます。これをうには、 `#!/usr/bin/env node`ようなNodeインタプリタをすシバンでめるがあり`#!/usr/bin/env node`。 このファイルもファイルとしてするがあります。これは`chmod`をってうことができます。スクリプトはコマンドラインからできるようになりました。

オンラインでアプリケーションをする

アプリケーションをNode.jsのホストにデプロイすると、このでは、サーバーをするためにできる `PORT`がされます。 `process.env.PORT`ポートをすると、アプリケーションにアクセスできます。

えば、

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

また、デバッグにオフラインでアクセスしたいは、のようにしてください

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

ここで、 `3000`はオフラインポートです。

NodeJSアプリケーションのデバッグ

ノードインスペクタをすることができます。 npmでインストールするには、のコマンドをします。

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

に、あなたはアプリケーションをデバッグすることができます

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

Githubのリポジトリはhttps://github.com/node-inspector/node-inspectorにあります。

ネイティブにデバッグする

また、のようにnode.jsをネイティブにデバッグすることもできます。

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

デバッガをのコードでにブレークするには、のようにします。

```
debugger;
```

はこちらをください 。

node.js 8では、のコマンドをします。

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

に、Google Chromeのバージョンで`about://inspect`き、NodeスクリプトをしてChromeのDevToolsのデバッグをします。

エクスプローラーきのHello World

のでは、Expressをしてポート3000でリッスンするHTTPサーバーをします。このサーバーは「Hello、World」とします。 Expressは、HTTP APIをするのになWebフレームワークです。

に、 `myApp`しいフォルダをします。 `myApp`り、のコードをむしいJavaScriptファイルをしますたとえば、 `hello.js`というをけて`hello.js`。 に、コマンドラインから`npm install --save express`をしてExpressモジュールをインストールします。 パッケージのインストールのについては、 このドキュメントをしてください 。

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

コマンドラインからのコマンドをします。

```
node hello.js
```

ブラウザをき、 `http://localhost:3000`または`http://127.0.0.1:3000`して、をします。

Expressフレームワークのについては、「 Web Apps with Express 」セクションをチェックしてください

Hello Worldルーティング

ノードをつHTTPサーバーをするをしたら、ユーザーがナビゲートしたパスについてを「う」をすることがです。このを「ルーティング」といいます。

これのもなは`if (request.url === 'some/path/here')`をチェックし、しいファイルでするをびすこ

とです。

これのはここでることができます

```
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);
```

しかし、このような "ルート"をしけると、なコールバックが1つだけすることになります。そのようななをんでいないので、これをクリーンアップできるかどうかをてみましょう。

まず、すべてのルートをオブジェクトにしましょう

```
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"');
    }
}
```

オブジェクトに2つのルートをしたので、ここでなコールバックでそれらをチェックできます

```
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);
```

すぐあなたのウェブサイトをナビゲートしようとするたびに、あなたのルートにそのパスのをし、それぞれのをびします。ルートがつからない、サバは404Not Foundでします。

また、HTTP Server APIをしたルーティングはにです。

これとのTCPとのないは、プライベートキーとオプションオブジェクトにするがあるパブリックだけです。

キーとをする

このセキュリティプロセスのは、プライベートキーのです。そしてこのはですかには、のにされるランダムノイズのセットです。には、1つのをし、それをってなものをすることができます。しかし、のものにしてなるキーをつことがベストプラクティスです。かがあなたのをむと、それはあなたののをむことにています。あなたが、ガレージ、オフィスなどをロックするためにじキーをしたをしてください。

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

をしたら、CSRをすることができます。このは、がのにされていることをしています。だからあなたのにするをするがあります。このはにされ、あなたのにされます。たちの、あなたがしたものはではありません。のステップでは、にするつもりであるからです。

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

ペーパーワークをさせたら、々はクールなであるとふるまうべきです。

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

プライベートキーとパブリックをしたので、2つのNodeJSアプリのなをできます。そして、サンプルコードでわかるように、になプロセスです。

たちはのをして、まったくなところ、はです。 NodeJSサーバーはデフォルトでこのようなをしません。そのため、のオプションrejectUnauthorizedfalseをしてをにするようにえるがあります。に ではこのをtrueにしないでください。

TLSソケットサーバー

```
'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ソケットクライアント

```
'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();

});
```

REPLのHello World

なしでびされると、Node.jsは " ノードシエル "ともばれるREPLRead-Eval-Print-Loopをします。

コマンドプロンプトでnodeしnode。

```
$ node
>
```

ノードのシェルプロンプトで>タイプの「Hello World」

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

Node.jsはブラウザでJavascriptをできるJavascriptエンジンGoogleのChromeV8エンジン、C ++でかれていますです。ノードのをするためにのライブラリがですが、エンジンにはをするのコアモジュールがしています。

、34のコアモジュールがノードにまれています

```
[ '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' ]
```

このリストは、Node documentation API https://nodejs.org/api/all.htmlJSONファイル https://nodejs.org/api/all.json からしました。

すべてのコアモジュールをでる

アサート

`assert`モジュールはをテストするためにすることができるアサーションテストのなセットをします。

バッファ

ECMAScript 2015ES6に`TypedArray`をするは、JavaScriptにはバイナリデ―タのストリ―ムをみんだりしたりするみがありませんでした。 `Buffer`クラスは、Node.js APIのとしてされ、TCPストリ―ムやファイルシステムなどのコンテキストでオクテットストリ―ムとやりりできるようにしました。

`TypedArray`がES6でされたので、 `Buffer`クラスは、Node.jsのユ―スケ―スにわせてされたで`Uint8Array` APIをします。

c / c ++ _ addons

Node.js Addonsは、CまたはC ++でかれたにリンクされたオブジェクトで、 `require()`をしてNode.jsにロ―ドでき、のNode.jsモジュ―ルとじようにできます。に、Node.jsでされているJavaScriptとC / C ++ライブラリとののインタフェ―スをするためにされます。

child_process

`child_process`モジュ―ルは、popen3にしているがではないでプロセスをするをします。

クラスタ

Node.jsののインスタンスは、のスレッドでされます。マルチコアシステムをするには、をするNode.jsプロセスのクラスタをするがあることがあります。クラスタモジュ―ルをすると、すべてのサ―バ―ポ―トをするプロセスをにできます。

コンソ―ル

`console`モジュ―ルは、ウェブブラウザによってされるJavaScriptコンソ―ルとのなデバッグコンソ―ルをする。

`crypto`モジュ―ルは、OpenSSLのハッシュ、HMAC、、、、のラッパ―セットをむをします。

deprecated_apis

aAPIのがであるとえられているか、bされたAPIがになっている、cのメジャ―リリ―スでAPIへののがされる、Node.jsはAPIをすることがあります。

DNS

`dns`モジュ―ルには、2つのなるカテゴリにするがまれています。

1. なオペレ―ティングシステムをしてをするで、ずしもネットワ―クをするはありません。このカテゴリにまれるは、 `dns.lookup()`です。
2. のDNSサ―バ―にしてをし、 にネットワ―クをしてDNSクエリをする。このカテゴリには、 `dns.lookup()` をく `dns`モジュ―ルのすべてのがまれます。

ドメイン

このモジュールはです。APIがすると、このモジュールはにされます。ほとんどのエンドユーザーは、このモジュールをするはありません。ドメインがするをにとするユーザーは、はそのにっているかもしれませんが、はのソリューションにするがあります。

イベント

Node.jsコアAPIのくは、ののオブジェクト「エミッタ」とばれますがにきイベントをし、Functionオブジェクト「リスナー」をびすようななイベントアーキテクチャをベースにされています。

fs

ファイルI / Oは、のPOSIXにするなラッパーによってされます。このモジュールをするには、`require('fs')`ます。すべてのメソッドは、とをちます。

http

Node.jsのHTTPインタフェースは、がであったプロトコルのくのをサポートするようにされています。に、チャンクエンコードされたなメッセージ。このインターフェースは、リクエストやレスポンスをバッファリングしないようにしています。ユーザーはデータをストリーミングすることができます。

https

HTTPSは、TLS / SSLのHTTPプロトコルです。 Node.jsでは、これはのモジュールとしてされています。

モジュール

Node.jsにはなモジュールローディングシステムがあります。 Node.jsでは、ファイルとモジュールは11でしていますファイルはのモジュールとしてわれます。

ネット

`net`モジュールは、ネットワークラッパーをします。サーバーとクライアントのをするためのストリームとばれますがまれています。このモジュールには`require('net');`をめることができます`require('net');`。

OS

`os`モジュールは、くのオペレーティングシステムのユーティリティメソッドをします。

パス

`path`モジュールは、ファイルおよびディレクトリパスをするためのユーティリティをします。

punycode

Node.jsにバンドルされている***punycode***モジュールのバージョンはです。

クエリ

querystringモジュールは、URLクエリのとのためのユーティリティをします。

みまれた

readlineモジュールは、Readableストリーム process.stdinなどからに1ずつデータをみむためのインターフェイスをします。

repl

replモジュールは、スタンドアロン·プログラムとしても、のアプリケーションでもなRead-Eval-Print-LoopREPLをします。

ストリーム

ストリームは、Node.jsのストリーミングデータをするためのなインターフェイスです。 streamモジュールは、ストリームインターフェイスをするオブジェクトをにできるAPIをします。

Node.jsによってされるくのストリームオブジェクトがあります。えば、HTTPサーバと process.stdoutへのリクエストはともストリームインスタンスです。

string_decoder

string_decoderモジュールは、エンコードされたマルチバイトUTF-8およびUTF-16をするで、BufferオブジェクトをストリングにデコードするためのAPIをします。

タイマー

timerモジュールは、あるのにびされるスケジューリングのグローバルAPIをします。タイマーはグローバルであるため、APIをするrequire('timers') timer require('timers')をびす require('timers')はありません。

Node.jsのタイマーは、Webブラウザーでされるタイマー APIとのAPIをしますが、Node.jsイベントループのりにされたのをします 。

tls_ssl

tlsモジュールは、OpenSSLのにされたTLSTransport Layer SecurityおよびSSLSecure Socket Layerプロトコルのをします。

トレース

トレースイベントは、V8、ノードコア、およびユーザーコードによってされたトレースをするメカニズムをします。

Node.jsアプリケーションのに--trace-events-enabledフラグをすと、トレースをにすることができます。

tty

`tty`モジュールは`tty.ReadStream`と`tty.WriteStream`クラスを。ほとんどの、このモジュールをするはありません。

dgram

`dgram`モジュールは、UDPデータグラムソケットのをします。

URL

`url`モジュールは、URLとのためのユーティリティをします。

ユーティリティ

`util`モジュールは、にNode.jsのAPIのニーズをサポートするようにされています。しかし、ユーティリティのくは、アプリケーションやモジュールにとってもです。

v8

`v8`モジュールは、Node.jsバイナリにみまれているV8のバージョンにのAPIをしています。

APIとはいつでもされることがあります。

VM

`vm`モジュールは、V8マシンのコンテキストでコードをコンパイルおよびするためのAPIをします。 JavaScriptコードはすぐにコンパイルしてすることも、コンパイルしてしてですることもできます。

vmモジュールはセキュリティではありません。 できないコードのにはしないでください 。

zlib

`zlib`モジュールは、GzipとDeflate / Inflateをしてされたをします。

なHTTPS Webサーバーをしてする

システムにnode.jsがインストールされたら、のにって、HTTPとHTTPSのをサポートしているWebサーバーをさせることができます。

1をする

1. キーとをするフォルダをします。

```
mkdir conf
```

---

2. そのディレクトリにします

```
cd conf
```

---

3. この`ca.cnf`ファイルをしてショートカットとしてする

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

---

4. このをしてしいをします。

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

---

5. `ca-key.pem`と`ca-cert.pem`にがあるので、サーバーのをしましょう

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

---

6. この`server.cnf`ファイルをしてショートカットとしてします。

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

---

7. のをしてをします。

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

---

8. リクエストにしてください

```
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
```

2をルートとしてインストールする

1. をルートのフォルダにコピーします。

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

---

2. CAストアをする

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

3ノードサーバーの

まず、のサーバーコードをむ`server.js`ファイルをします。

Node.jsのHTTPSサーバーののは、のようになります。

```
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);
```

httpリクエストをサポートしたいは、このさなをうがあります。

```
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. `server.js`があるディレクトリにします

   ```
   cd /path/to
   ```

---

2. `server.js`し`server.js`。

   ```
   node server.js
   ```

オンラインでNode.jsをいめるをむ https://riptutorial.com/ja/node-js/topic/340/node-jsをいめる

2: angular.jsをむNode.jsexpress.jsサンプルコード

き

このでは、なExpressアプリケーションをし、AngularJSをするをします。

Examples

たちのプロジェクトをする。

たちはくことができて、コンソールからもうします。

```
mkdir our_project
cd our_project
```

たちはコードがきるにいます。プロジェクトのメインアーカイブをするには、をします。

わかりましたが、たちはどのようにスケルトンプロジェクトをしましたか

それはです

```
npm install -g express express-generator
```

LinuxディストリビューションとMacは、 **root**ユーザーだけがアクセスなnodejsディレクトリにインストールされているため、 **sudo**をしてインストールするがあります。すべてがうまくいけば、にエクスプレスアプリスケルトンをし、

```
express
```

このコマンドは、エクスプローラーのサンプルアプリケーションをフォルダーにします。はのりです

```
bin/
public/
routes/
views/
app.js
package.json
```

すぐ**npm**をすると、 http// localhost3000にきます.**Express**アプリがしていることがわかります.

Expressのアプリは AngularJSとどのようにさせることができますか 。

にするは

Expressは**Nodej**のにされたフレームワークで、 Expressサイトでドキュメントをることができます。しかし、たちののためには、アプリケーションのホームページをレンダリングするために、http// localhost3000 / homeなど、するときに**Express**がをうことをるがあります。されたされたアプリから、をできます

```
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;
```

このコードがっていることは、ユーザーがhttp// localhost3000にくとき、 インデックスビューをレンダリングし、タイトルプロパティとExpressをつ**JSON**をすがあるということです。しかし、viewsディレクトリをしてindex.jadeをくと、のようにされます。

```
extends layout
block content
    h1= title
    p Welcome to #{title}
```

これは、 テンプレートエンジンであるなもう1つのなで、をしたりのテンプレートをしたりしてページのコンテンツをレンダリングすることができるため、のがページをよりコンパクトにできるようになります。ファイルのは、にじテンプレートエンジンが、いくつかのとコアのではある、のるりジェイドは パグのをっているよう**.jade**されます。

Pugのインストールと**Express**テンプレートエンジンの

さて、Pugをプロジェクトのテンプレートエンジンとしてするには、をするがあります

```
npm install --save pug
```

これにより、Pugがプロジェクトのとしてインストールされ、 **package.json**にされます。それをするには、 **app.js**ファイルをするがあります

```
var app = express();
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

そして、ビューエンジンのエンジンをpugにきえてください。たちは**npm start**でプロジェクトをやりすことができ、すべてがにしていることがわかります。

AngularJSはこれにどのようにしていますか

AngularJSは、に**SPA** Simple Page Applicationのインストールをにするためにされるjavascript **MVW** Model-View-Whateverフレームワークで、 AngularJSのWebサイトにアクセスして**v1.6.4**のバージョンをダウンロードできます。

AngularJSをダウンロードした、プロジェクトの**public / javascripts**フォルダにファイルをコピーするがあります。しがあります。これは、サイト、、CSS、javacriptファイルなどのをするフォルダです。もちろん、これは**app.js**ファイルでできますが、にしておきます。ここで、AngularJSがするjavascriptのパブリックフォルダに、アプリケーションがきる**ng-app.js**というのファイルをします。 AngularJSをするには、 **view / layout.pug**のをのようにするがあります。

```
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')
```

ここではをしていますかまあ、AngularJSコアとされたファイル**ng-app.js**をんでいるので、テンプレートがレンダリングされるとAngularJSがアップされ、 **ng-app**ディレクティブがされます。AngularJSこれはアプリケーションであり、それにするがあります。
ですから、 **ng-app.js**のはのようになります

```
angular.module('first-app', [])
       .controller('indexController', ['$scope', indexController]);

function indexController($scope) {
    $scope.name = 'sigfried';
}
```

ここでもなAngularJSの、 のデータバインディングをしています。これにより、ビューとコントローラのをにすることができます。これはになですが、GoogleやStackOverflowでしてどのようににします。

ですから、AngularJSアプリケーションのブロックをっていますが、ここでうべきことがあります。アプリのをするためにindex.pugページをするがあります。

```
extends layout
block content
  div(ng-controller='indexController')
    h1= title
```

```
    p Welcome {{name}}
    input(type='text' ng-model='name')
```

ここでは、コントローラのAngularJSスコープのみのプロパティにをバインドするだけです。

```
$scope.name = 'sigfried';
```

これは、のテキストをすると、のは{{name}}のをします。これはともばれ、テンプレートにコンテンツをレンダリングするもう1つのAngularJSです。

したがって、すべてがセットアップされているので、 **npm start**をすることができます。http://localhost3000にき、エクスプレスアプリケーションをし、アプリケーションフロントエンドをするAngularJSをしてください。

オンラインでangular.jsをむNode.jsexpress.jsサンプルコードをむ https://riptutorial.com/ja/node-js/topic/9757/angular-jsをむnode-js-express-js-サンプルコード

3: async.js

- コールバックは、のでするがあります。
- コールバックerr、result [、arg1 [、...]]
- このでは、まずエラーをすようにされ、でエラーをすることはできません。 `null`はエラーがないのです
- コールバックnull、myResult;
- あなたのコールバックは***err***や***result***よりもくのをむことができますが、の**waterfall**、seq、...
- コールバックnull、myResult、myCustomArgument;
- もちろん、エラーをします。あなたはそれをしなければならず、エラーをしなければなりません。
- コールバックエラー;

Examples

パラレルマルチタスキング

*async.parallel*tasks、afterTasksCallbackは、のタスクをしてし、すべてのタスクのをします コールバックのびしによってされます。

タスクがすると、 *async*はすべてのエラーとタスクのすべてのでメインコールバックをびします。

```
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);
  });
```

`["resultOfShortTime", "resultOfMediumTime", "resultOfLongTime"]`。

オブジェクトで`async.parallel()`をびす

タスクパラメータをオブジェクトできえることができます。この、はタスクとじキーをつオブジェクトにもなります。

いくつかのタスクをし、それぞれのをにつけることはにです。

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

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

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

のをする

にコールバックがされます。このコールバックは、のとしてエラーをしたり、それのをすことができます。いくつかのがコールバックにされた、これらのはとしてされます。

```
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);
  });
```

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

。

シリーズしたモノタスク

async.seriestasks、afterTasksCallbackはのタスクをします。タスクはゃにされます。 タスクがすると、 ***async***はすぐにをし、メインのコールバックにジャンプします。

タスクがにすると、 *async*は、すべてのエラーとすべてのタスクをむ「マスター」コールバックをびします。

```
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);
  });
```

["resultOfMediumTime", "resultOfShortTime", "resultOfLongTime"]。

オブジェクトで async.series() をびす

タスクパラメータをオブジェクトできえることができます。この、はタスクとじキーをつオブジェクトにもなります。

いくつかのタスクをし、それぞれのをにつけることはにです。

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

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

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

モノタスク

*async.waterfall*tasks、afterTasksCallbackはのタスクをします。タスクは々にされ、タスクのはのタスクにされます。 *async.series*として、タスクがした、 *async*はをし、メインのコールバックをちにびします。

タスクがにすると、 *async*は、すべてのエラーとすべてのタスクをむ「マスター」コールバックをびします。

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

`results`は、そののの`friendsResult`であるののの2のコールバックパラメータがまれます。

async.timesよりいでforループをするため

node.jsのループでをするには、いループの`for`ループをするのがよいでしょう。しかし、ループがいと、 `for`ループを`for`とがくなり、ノード·プロセスがハングアップするがあります。このようなシナリオでは、 **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
});
```

これはしてびされます。に1つずつびすは、 **async.timesSeries**をします。

async.eachデータのをにするため

々は、データのをしたいときは、よりい**async.each**をする。すべてのデータをってかをしたいときに、すべてがしたらのコールバックをしたいとき、このメソッドはです。これはしてされます。

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

に1つずつうには**async.eachSeries**をできます

async.series イベントを1つずつする

*/ async.series*では、すべてのがにされ、のがコールバックにされます。えば */*

var async = require 'async'; async.series[functioncallback{の..}; callbacknull、 'userPersonalData';}、コールバック{console.log 'Second Execute ..';コールバックnull、 'userDependentData';}]、エラー、{console.log;};

//

の.. 2の.. ['userPersonalData'、 'userDependentData'] //

オンラインでasync.jsをむ https://riptutorial.com/ja/node-js/topic/3972/async-js

4: Browserfiyをしてブラウザで「」エラーをする

Examples

- file.js

このでは、 **file.js**というファイルがあります。

JavaScriptとNodeJSクエリモジュールをしてURLをするがあるとしましょう。

これをうには、ファイルにのをするだけです。

```
const querystring = require('querystring');
var ref = querystring.parse("foo=bar&abc=xyz&abc=123");
```

このスニペットはをしていますか

に、URLクエリのとのためのユーティリティをするクエリモジュールをします。のでアクセスできます。

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

に、.parseメソッドをしてURLをします。 URLクエリstrをキーとのペアのにします。

たとえば、クエリ`'foo=bar&abc=xyz&abc=123'`はのようにされます。

```
{  foo: 'bar',  abc: ['xyz', '123']  }
```

ながら、ブラウザには*require*メソッドがされていませんが、Node.jsではされていません。

ブラウザをインストールする

Browserifyをすると、Nodeでするのとじで*require*をするコードをできます。だから、どうやってこれをするのですかそれはです。

1. npmにされているのインストールノード。それから

   npm install -g **browserify**

2. あなたのfile.jsがされているディレクトリにし、NPMとたちのクエリのモジュールをインス

トールします。

npmはクエリをインストールします

のディレクトリでしないと、モジュールをむファイルがつからないため、コマンドはします。

3. に、file.jsからまるなモジュールを、bundle.jsというのファイルまたはをけたいに、**browserify**コマンドでにバンドルします。

   browserify file.js -o bundle.js

Browserifyは、 *require*びしのためのをして、グラフをトラバースします。

4. にあなたのhtmlにのタグをドロップし、です

   ```
   <script src="bundle.js"></script>
   ```

あなたのい.jsファイル **file.js**とじ としくされた**bundle.js**ファイルのみわせがられます。これら2つのファイルは1つのファイルにされます。

---

file.jsをしたいは、プログラムのにはしません。 しくした**bundle.js**をすると、がになります

どういうですか

つまり 、らかので**file.js**をしたい、そのはをえません。 **bundle.js**と**file.js**をマージするので 、**bundle.js**をにするがあります 。

オンラインでBrowserfiyをしてブラウザで「 」エラーをするをむ https://riptutorial.com/ja/node-js/topic/7123/browserfiyをしてブラウザで--エラーをする

5: CLI

- ノード[オプション] [v8オプション] [script.js | -e "スクリプト"] []

Examples

コマンドラインオプション

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

されたバージョンv0.1.3ノードのバージョンをします。

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

されたバージョンv0.1.3ノードのコマンドラインオプション。このオプションのは、このドキュメントよりです。

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

されたバージョンv0.5.2のをJavaScriptとしてします。 REPLであらかじめされているモジュールは、スクリプトでもできます。

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

されたバージョンv0.6.4 -eとじですが、がされます。

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

されたバージョンv5.0.0スクリプトをせずにチェックします。

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

されたバージョンv0.7.7 stdinがのようにえなくてもREPLをきます。

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

されたバージョンv1.6.0にされたモジュールをプリロードします。

requireのモジュールにいます。 moduleは、ファイルへのパスまたはノードモジュールのいずれかです。

```
--no-deprecation
```

でされましたv0.8.0の。

```
--trace-deprecation
```

されたバ―ジョンv0.8.0のスタックトレ―スをします。

```
--throw-deprecation
```

でされましたv0.11.14のエラ―をげます。

```
--no-warnings
```

でされましたv6.0.0すべてのプロセスのをさせますをむ。

```
--trace-warnings
```

されたバ―ジョンv6.0.0プロセスのdeprecationsをむのスタックトレ―スをします。

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

されたバ―ジョンv2.1.0イベントル―プののターンのにI / Oがされるたびにスタックトレ―スをします。

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

されたバ―ジョンv6.0.0しくりてられたすべてのBufferおよびSlowBufferインスタンスをにゼロにりつぶします。

```
--preserve-symlinks
```

されたバ―ジョンv6.3.0モジュ―ルをしキャッシングするときにシンボリックリンクをするようにモジュ―ルロ―ダ―にします。

デフォルトでは、Node.jsがディスクのなるにシンボリックリンクされたパスからモジュ―ルをロ―ドすると、Node.jsはリンクをし、モジュ―ルののディスクの「のパス」をのとしてしますのモジュ―ルをつけるためのル―トパスとしてします。ほとんどの、このデフォルトのはされます。ただし、のにすように、シンボリックリンクピアをする、moduleAがmoduleBをピアとしてすると、デフォルトのによりがスロ―されます。

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

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

--preserve-symlinksコマンドラインフラグは、のパスとはにモジュールのシンボリックリンクパスをするようにNode.jsにし、シンボリックにリンクされたピアがつかるようにします。

ただし、--preserve-symlinksをすると、そののがじることにしてください。には、シンボリックリンクされたネィティブモジュールは、ツリーののからリンクされているとロードできませんNode.jsはそれらを2つのしたモジュールとみなし、モジュールをロードしてをスローします。

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

でされましたv2.4.0ヒープスナップショットのヒープオブジェクトりてをします。

```
--prof-process
```

されたバージョンv6.0.0 v8オプション--profをしてされたv8プロファイラをします。

```
--v8-options
```

されたバージョンv0.1.3 v8のコマンドラインオプションをします。

v8オプションをすると、をダッシュ - またはアンダースコア_でることができます。

たとえば、--stack-trace-limitは--stack_trace_limitとです。

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

されたバージョンv4.0.0のデフォルトのTLSリストをします。 サポートをしてNode.jsをするがありますデフォルト

```
--enable-fips
```

されたバージョンv6.0.0にFIPSのをにします。 Node.jsを./configure --openssl-fipsでビルドするがあります

```
--force-fips
```

でされましたv6.0.0にFIPSのをします。 スクリプトコードからにすることはできません--enable-fipsとじ

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

v0.11.15 ICUデータロードパスをします。 NODE_ICU_DATAをきする

```
Environment Variables

NODE_DEBUG=module[,…]
```

v0.1.32 '、' - デバッグをするコアモジュールのリスト。

```
NODE_PATH=path[:…]
```

されたバージョンv0.1.32 '' - モジュールのパスにきのディレクトリのりリスト。

Windowsでは、これはわりに ';'でられたリストです。

```
NODE_DISABLE_COLORS=1
```

されたバージョンv0.3.0 1にされた、REPLではされません。

```
NODE_ICU_DATA=file
```

されたバージョンv0.11.15 ICUIntlオブジェクトデータのデータパス。さなicuサポートでコンパイルされたときにリンクインデータをします。

```
NODE_REPL_HISTORY=file
```

されたバージョンv5.0.0なREPLをするためにされるファイルへのパス。デフォルトのパスは/.node_repl_historyで、このできされます。 をの ""または ""にすると、なREPLがになります。

オンラインでCLIをむ https://riptutorial.com/ja/node-js/topic/6013/cli

6: CORSをしたNode.js

Examples

express.jsでCORSをにする

node.jsはAPIをするためによくされるため、なるドメインからAPIをリクエストできるようにするには、なCORSがのになります。

では、よりいコンフィグレーションすべてのドメインからすべてのリクエストタイプをするのためにします。

エクスプレスをしたのserver.js

```
// 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();
    });
});
```

、ノードはサーバーのプロキシのでされます。したがって、プロキシサーバーApacheやNginxなどはCORSをします。

このシナリオをにさせるために、のnode.js CORSのみをにすることができます。

これは`NODE_ENV`をチェックすることでにえ`NODE_ENV`

```
const app = express();

if (process.env.NODE_ENV === 'development') {
    // CORS settings
}
```

オンラインでCORSをしたNode.jsをむ https://riptutorial.com/ja/node-js/topic/9272/corsをした-node-js

7: ES6をしたNode.JS

き

ES6、ECMAScript 6またはES2015はJavaScriptののであり、のをそのにしています。それはへのきなアップデートであり、くのをしていま す

NodeとES6のは、それぞれのサイトのhttps://nodejs.org/ja/docs/es6/にあります。

Examples

ノードES6 Babelによるプロジェクトのサポートと

ES6のはまだにはされていないので、のしかできません。 http://node.green/でサポートされているES6ののをることができます

NodeJS v6、かなりいサポートがありました。したがって、NodeJS v6をしているは、ES6をしむことができます。ただし、リリースされていないのとそれのをすることもできます。このためには、をするがあります

ランタイムにビルドをしてビルドし、すべてのES6などをすることができます。 JavaScriptのもなトランスバイザーはBabel

Babelでは、ES6のすべてのと、`'var thing = require('thing')`わりに`import thing from 'thing`-`import thing from 'thing`のような' stage-0 'の`import thing from 'thing`がされています

インポートなどの「ステージ0」をするプロジェクトをしたいは、バベルをトランスバータとしてするがあります。あなたは、とVueをったプロジェクトとのcommonJSベースのパターンがstage-0をかなりにするのをています。

しいノードプロジェクトをする

```
mkdir my-es6-app
cd my-es6-app
npm init
```

ES6プリセットとステージ0をしてください

```
npm install --save-dev babel-preset-es2015 babel-preset-stage-2 babel-cli babel-register
```

`server.js`というのしいファイルをし、なHTTPサーバーをします。

```
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/')
```

`import http from 'http'`をしていますが、これはステ―ジ0のです。すれば、しくしていることをします。

`node server.js`をすると、インポ―トのがわからなくなり`node server.js`。

ディレクトリのルートに.babelrcファイルをし、のをします

```
{
  "presets": ["es2015", "stage-2"],
  "plugins": []
}
```

これで、 `node src/index.js --exec babel-node`サ―バ―をできます`node src/index.js --exec babel-node`

プロダクションアプリでにトランスヒ―ラ―をするのはいではありません。ただし、package.jsonにスクリプトをして、いやすくすることができます。

```
"scripts": {
    "start": "node dist/index.js",
    "dev": "babel-node src/index.js",
    "build": "babel src -d dist",
    "postinstall": "npm run build"
  },
```

は`npm install`にdistiledディレクトリにトランスビルドされたコ―ドをビルドします。npm `npm start`プロダクションアプリにトランスビルドされたコ―ドをできるようにします。

`npm run dev`は、プロジェクトをロ―カルでするときにうまくする、サ―バ―とbabelランタイムをします。

さらにんでいくと、nodemon `npm install nodemon --save-dev`を`npm install nodemon --save-dev`してをし、ノ―ドアプリケ―ションをすることができます。

これにより、バ―ベルとNodeJSのがにします。あなたはpackage.jsonでに "dev"スクリプトをしてnodemonをします

```
"dev": "nodemon src/index.js --exec babel-node",
```

NodeJSアプリケ―ションでJS es6をする

JS es6es2015ともばれますは、OOPをしているときやなにしているときに、よりにするためのJSののセットです。

1. http://es6-features.orgのしいes6をチェックしてください。のNodeJSアプリケーションでにするがあるかどうかをにすることができます

2. http://node.greenでノードのバージョンのレベルをしてください

3. すべてがなら - コードをこう

JS es6をったな`hello world` appのサンプルです

```
'use strict'

class Program
{
    constructor()
    {
        this.message = 'hello es6 :)';
    }

    print()
    {
        setTimeout(() =>
        {
            console.log(this.message);

            this.print();

        }, Math.random() * 1000);
    }
}

new Program().print();
```

このプログラムをして、じメッセージをりしするをできます。

さて、ごとにけてみましょう。

```
'use strict'
```

このは、js es6をするににです。 `strict`モードは、に、のコードとはなるセマンティクスをっていますMDN - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

```
class Program
```

じられない - `class`キーワードなリファレンスのために - es6のに、jsのクラスをするのは... `function`キーワードであった

```
function MyClass() // class definition
{

}

var myClassObject = new MyClass(); // generating a new object with a type of MyClass
```

OOPをする、クラスはシステムののをするためにをするになですコードがきくなるとコードをすることがです。たとえば、サーバーサイドコードをくときなど

```
constructor()
{
    this.message = 'hello es6 :)';
}
```

あなたはめなければなりません - これはかなりですこれはのクラスのc'torです。このユニークな "は、こののクラスからオブジェクトがされるたびにしますプログラムでは1のみ

```
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);
}
```

printはクラススコープでされているため、にはメソッドです。クラスのオブジェクトかクラスからびすことができます。

だからまでにクラスをしました。する

```
new Program().print();
```

これはにのものとじです

```
var prog = new Program(); // define a new object of type 'Program'

prog.print(); // use the program to print itself
```

として、 JS es6はコードをすることができますのバージョンのJSとして、でわかりやすくなります。のコードをきしてみてください。

しい

オンラインでES6をしたNode.JSをむ https://riptutorial.com/ja/node-js/topic/5934/es6をしたnode-js

8: Express.JSでajaxリクエストをルーティングする

Examples

AJAXのな

あなたは、なジェネレータテンプレート

app.jsで、add `var app = express.app()`のどこにでもできます

```
app.post(function(req, res, next){
    next();
});
```

index.jsファイルまたはそれぞれのマッチにのをします。

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

`/views`ディレクトリに`ajax.jade` / `ajax.pug`または`ajax.ejs`ファイルをし、 `ajax.jade`にします。

/ PugJSの

```
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")
```

EJSの

```
<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>
```

さて、 `/public magic.js`とばれるファイルをします。

```
$(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);
        });
    });
});
```

そしてあなたはそれをっていますをクリックすると、もりがされます

オンラインでExpress.JSでajaxリクエストをルーティングするをむ https://riptutorial.com/ja/node-js/topic/6738/express-jsでajaxリクエストをルーティングする

9: ExpressJSのルートコントローラ - サービス

Examples

モデル - ルート - コントローラ - サービスディレクトリ

```
├──models
|   ├──user.model.js
├──routes
|   ├──user.route.js
├──services
|   ├──user.service.js
├──controllers
|   ├──user.controller.js
```

モジュラコードの、ロジックはこれらのディレクトリとファイルにするがあります。

> モデル - モデルのスキーマ
>
> ルート - APIルートはコントローラにマップされます
>
> コントローラー - コントローラーは、パラメーターの、、しいコードによるののにあるすべてのロジックをします。
>
> サービス - サービスには、データベースのクエリとりオブジェクトまたはエラーのスローがまれます。

このコーダーは、よりくのコードをくことになります。しかし、にコードははるかにとがします。

モデル - ルート - コントローラ - サービスコード

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')
    }
}
```

オンラインでExpressJSのルートコントローラ - サービスをむ https://riptutorial.com/ja/node-js/topic/10785/expressjsのルートコントローラ---サービス

10: ExpressをしたWebアプリケーション

き

Expressは、でなNode.js Webアプリケーションフレームワークであり、Webアプリケーションをするためのなをします。

Expressのウェブサイトはexpressjs.comです。ソースはGitHubにあります 。

- app.getpath [、middleware]、コールバック[、コールバック...]
- app.putpath [、middleware]、コールバック[、コールバック...]
- app.postpath [、middleware]、コールバック[、コールバック...]
- app ['delete']パス[、ミドルウェア]、コールバック[、コールバック...]
- app.usepath [、middleware]、コールバック[、コールバック...]
- app.useコールバック

パラメーター

パラメータ	
`path`	されたコールバックがするパスまたはURLをします。
`middleware`	コールバックのにびされる1つの。に、の`callback`の。やエラーなど、よりなにちます。
`callback`	された`path`へのをするためにされる。それはのようにびされる `callback(request, response, next)` 、 `request` 、 `response` 、および`next` 、にされています。
コールバック `request`	コールバックがするためにびされているHTTPリクエストにするをカプセルしているオブジェクトです。
`response`	サーバーがにどのようにするかをするためにされるオブジェクト。
`next`	のするルートにをすコールバック。オプションのエラーオブジェクトをけれます。

Examples

にディレクトリをし、シェルにアクセスし、 npmをしてExpressをインストールするがあります
`npm install express --save`

ファイルをし、 `app.js`というをけ、のコードをしてしいExpressサーバーをし、 `app.get`メソッド

をしてそのサーバーに1つのエンドポイント `/ping` をします。

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

const app = express();

app.get('/ping', (request, response) => {
    response.send('pong');
});

app.listen(8080, 'localhost');
```

スクリプトをするには、シェルでのコマンドをします。

```
> node app.js
```

アプリケーションはlocalhostポート8080でをけけます`app.listen`のhostnameをすると、serverはマシンのIPアドレスとlocalhostでをけれます。ポートのが0の、オペレーティングシステムはなポートをりてます。

スクリプトがされると、それをシェルでテストして、サーバーからされる「pong」をけることをできます。

```
> curl http://localhost:8080/ping
pong
```

また、Webブラウザをいてurl http// localhost8080 / pingにしてをすることもできます

なルーティング

まずエクスプレスアプリをします

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

に、このようなルートをすることができます

```
app.get('/someUri', function (req, res, next) {})
```

このはすべてのHTTPメソッドにしてし、のとしてパスがであり、そのパスのハンドラがオブジェクトとオブジェクトをけります。したがって、なHTTPメソッドの、これらはルートです

```
// 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) {})
```

ここでサポートされているのなリストをできます 。ルートとすべてのHTTPメソッドにしてじをするは、のようにできます。

```
app.all('/myPath', function (req, res, next) {})
```

または

```
app.use('/myPath', function (req, res, next) {})
```

または

```
app.use('*', function (req, res, next) {})

// * wildcard will route for all paths
```

1つのパスにしてのルートをさせることができます

```
app.route('/myPath')
  .get(function (req, res, next) {})
  .post(function (req, res, next) {})
  .put(function (req, res, next) {})
```

のHTTPメソッドにをすることもできます。それらはコールバックのにされ、パラメータreq、res、nextをとしてります。

```
// GET www.domain.com/myPath
app.get('/myPath', myFunction, function (req, res, next) {})
```

のコールバックをファイルにして、1つのファイルにコードをれすぎないようにすることができます。

```
// other.js
exports.doSomething = function(req, res, next) {/* do some stuff */};
```

そしてあなたのルートをんでいるファイルで

```
const other = require('./other.js');
app.get('/someUri', myFunction, other.doSomething);
```

これはコードをもっときれいにします。

リクエストからのの

リクエストしているURLからをする `req`はルートのハンドラのリクエストオブジェクトであるこ

とにしてください。このルート`/settings/:user_id`とこのの`/settings/32135?field=name`えてみましょう

```
// 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'
```

リクエストのヘッダーをすることもできます。

```
req.get('Content-Type')
// "text/plain"
```

のをするのをにするために、ミドルウェアをすることができます。たとえば、リクエストのボディをするには、ボディリクエストのをなフォーマットにするbody-parserミドルウェアをできます。

```
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
```

このようなリクエストをえてみましょう

```
PUT /settings/32135
{
  "name": "Peter"
}
```

あなたはこのようにされたにアクセスできます

```
req.body.name
// "Peter"
```

ので、リクエストからクッキーにアクセスできます。また、 クッキーパーサーのようなミドルウェアもです

```
req.cookies.name
```

モジュラーエクスプレスアプリケーション

Webアプリケーションのモジュラルーターファクトリをするには

モジュール

```
// 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;
};
```

```
// app.js
const express = require('express');
const greetMiddleware = require('./greet.js');

express()
    .use('/api/v1/', greetMiddleware({ greeting:'Hello world' }))
    .listen(8080);
```

これにより、アプリケーションをモジュールし、カスタマイズし、コードをできるようにします。

`http://<hostname>:8080/api/v1/greet`アクセスすると、は`Hello world`なり`Hello world`

よりな

ミドルウェアののをすサービスの

モジュール

```
// 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;
};
```

```
// 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);
```

アクセスする`http://<hostname>:8080/api/v1/service1/greet?name=World`がされます`Hello, World`とアクセスし`http://<hostname>:8080/api/v1/service2/greet?name=World`はが`Hi, World`ます。

テンプレートエンジンの

テンプレートエンジンの

のコードはJadeをテンプレートエンジンとしてします。 201512、Jadeのは`pug`されています。

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

に、 `Handlebars` `hbs` や`ejs`などののテンプレートエンジンもできます。 Template Engineも`npm`

`install`てください。ハンドルバーは`hbs`パッケージ、 `jade`には`jade`パッケージ、 `ejs`には`ejs`パッケージがあります。

EJSテンプレートの

EJSのExpressテンプレートとでは、サーバーコードをし、HTMLからサーバーにアクセスできます。
EJSでは、タグとして "`<%`"、タグとして "`%>`"をしてしました。レンダリングパラメータとしてされるは、 `<%=var_name%>`
たとえば、サーバーコードにアレイがある
それをしてループすることができます

```
<h1><%= title %></h1>
   <ul>
<% for(var i=0; i<supplies.length; i++) { %>
    <li>
        <a href='supplies/<%= supplies[i] %>'>
            <%= supplies[i] %>
        </a>
    </li>
<% } %>
```

このでは、サーバーのコードとHTMLをりえるたびに、のEJSタグをじてでしいタグをくがあるため、ここでは`for`コマンドに`li`をしてEJSタグをじるがありました`for`のにかっこのしいタグをするもうつの
のデフォルトのバージョンをサーバーからのにするは、 `<%=`
えば

```
 Message:<br>
<input type="text" value="<%= message %>" name="message" required>
```

ここでは、サーバーからされるメッセージがのデフォルトになります。サーバーからメッセージをさなかった、EJSはをスローします。 `res.render('index', {message: message});`をしてパラメータをすことができます`res.render('index', {message: message});` index.ejsというejsファイルの。

EJSタグでは、 `if` 、 `while`またはのjavascriptコマンドをすることもできます。

ExpressJSをしたJSON API

```
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)
})
```

`http://localhost:8080/` outputオブジェクトの

```
{
    string_value: "StackOverflow",
    number_value: 8476
}
```

ファイルの

ExpressをしてWebサーバーをする、コンテンツとファイルをみわせてするがあります。

たとえば、ファイルシステムにされているファイルであるindex.htmlとscript.jsがあるとします。

ファイルをつために 'public'というのフォルダをするのがです。この、フォルダはのようになります。

```
project root
├── server.js
├── package.json
└── public
    ├── index.html
    └── script.js
```

これは、ファイルをするようにExpressをするです。

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

app.use(express.static('public'));
```

フォルダがされると、index.html、script.js、および "public"フォルダのすべてのファイルがルートパスでになりますurlには`/public/`をしないでください。これは、エクスプレスがされたフォルダにするファイルをするためです。 パスプレフィックスは 、のようにできます。

```
app.use('/static', express.static('public'));
```

リソースを`/static/` prefixのでできるようにします。

のフォルダ

にのフォルダをすることはです

```
app.use(express.static('public'));
app.use(express.static('images'));
app.use(express.static('files'));
```

リソースをするとき、Expressはにフォルダーをします。じのファイルのは、にするフォルダのファイルがされます。

Django-styleのきルート

きなの1つは、なきルートが、Expressのすぐでサポートされていないことです。は、サポートされているサードパーティのパッケージをインストールすることですたとえば、 express-reverse

```
npm install express-reverse
```

あなたのプロジェクトにプラグインしてください

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

に、のようにします。

```
app.get('test', '/hello', function(req, res) {
  res.end('hello');
});
```

このアプローチのは、 なルータのにすように、 `route` Expressモジュールをできないことです。このをするには、ルータのファクトリにパラメータとして`app`をす

```
require('./middlewares/routing')(app);
```

それをのようにします。

```
module.exports = (app) => {
    app.get('test', '/hello', function(req, res) {
      res.end('hello');
    });
};
```

これからは、をしてされたカスタムとマージし、なコントローラをすをすることができます。

エラー

なエラー

デフォルトでは、Expressはレンダリングする`/views`ディレクトリの「エラー」ビューをします。「エラー」ビューをして、エラーをするためにビューディレクトリにするだけです。エラーには、エラーメッセージ、ステータス、およびスタックトレースがきまれます。たとえば、のようになります。

views / error.pug

```
html
  body
      h1= message
      h2= error.status
      p= error.stack
```

なエラー

ミドルウェアスタックのに、エラーミドルウェアをします。これらには、3つの`(err, req, res, next)`わりに4つのがあります。

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
    });
});
```

のミドルウェアとじように、いくつかのエラーミドルウェアをすることができます。

ミドルウェアとのコールバックの

Expressは、のハンドラでルートのロジックをするためにできるすべてのルート・ハンドラおよびミドルウェアに`next`コールバックをします。きなしで`next()`をびすと、のするミドルウェアまたはルートハンドラにするようにexpressにします。エラーで`next(err)`をびすと、エラーハンドラミドルウェアがトリガされます。 `next('route')`をびすと、のルートののミドルウェアをバイパスし、

のするルートにジャンプします。これにより、ドメインロジックをでテストがで、やがなコンポーネントにすることができます。

のするルート

`/api/foo`または`/api/bar`へのリクエストは、ハンドラをしてメンバをし、ルートののハンドラにをします。

```
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);
});
```

エラーハンドラ

エラーハンドラは、`function(err, req, res, next)`つミドルウェアです。 `app.get('/foo', function(err, req, res, next)` などのルートごとにできますが、はエラーページをレンダリングするのエラーハンドラでです。

```
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);
});
```

ミドルウェア

のは、にはがされたルートとするたびにされるミドルウェアですが、1つのルートにののミドルウェアをできます。これにより、ミドルウェアを々のファイルでし、ロジックをのルートにわたってすることができます。

```
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);
});
```

このでは、ミドルウェアは、それのファイルまたはファイルののののいずれかになり、のルートでできるようになります。

エラー

なドキュメントはここにあります

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

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));
});
```

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';
```

フックのreqのとのresのにコードをする

`app.use()`とミドルウェアは "before"にでき、 closeイベントとfinishイベントのみわせは "after"にできます。

```
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);
```

このは、デフォルトでにログにされるロガーミドルウェアです。

この"ミドルウェア"は、がになるように`app.router`にされていることをしてください。

オリジナルはこちら

POSTリクエストの

app.getメソッドをしてExpressでリクエストをするのとじように、app.postメソッドをしてポストリクエストをできます。

しかし、POSTリクエストをするに、 `body-parser`ミドルウェアをするがあります。に`POST` 、 `PUT` 、 `DELETE`などののをします。

`Body-Parser`ミドルウェアはリクエストのをし、 `req.body`なオブジェクトにします

```
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');
```

Cookieパーサーでクッキーをする

は、 cookie-parserモジュールをしてCookieをおよびみむです。

```
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);
```

Expressのカスタムミドルウェア

Expressでは、リクエストのチェックやのヘッダーのにできるミドルウェアをできます。

```
app.use(function(req, res, next){ });    // signature
```

のコードは、 `user`をリクエストオブジェクトにし、のするルートにコントロールをします。

```
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);
```

Expressでのエラー

Expressでは、アプリケーションでしたエラーをするエラーハンドラをできます。すべてのルートとロジックコードのにハンドラをします。

```
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);
```

ミドルウェアの

ミドルウエアは、アプリケーションのサイクルでオブジェクトreq、オブジェクトres、およびのミドルウエアにアクセスできるです。

ミドルウエアのは、のコードをし、 resおよびreqオブジェクトをし、サイクルをし、のミドルウエアをびすことができます。

になミドルウエアのはcorsモジュールです。 CORSサポートをするには、にインストールして、それをとし、のをします。

```
app.use(cors());
```

ルータまたはルーティングのに

こんにちは

ここでは、Expressをしてなhello worldサーバーをします。ルート

- '/'
- '/ wiki'

そして、りは "404"、つまりページがつかりません。

```
'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'));
```

エクスプレスはにをみね、それぞれのにしてにするため、ルートとして404ルートをしています。

オンラインでExpressをしたWebアプリケーションをむ https://riptutorial.com/ja/node-js/topic/483/expressをしたwebアプリケーション

11: HTMLやそののファイルをする

- response.sendFileファイル、オプション、エラー{};

Examples

されたパスでHTMLをする

ここではExpressサーバをし、するです`index.html` のパスデフォルトでは`/` 、および`page1.html` `/page1`パス。

フォルダ

```
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);
```

`sendFile()`はレスポンスとしてファイルをストリームするだけで、するはありません。 HTMLファイルをしていて、そのファイルにデータをめるは、Pug、Mustache、EJSなどのテンプレートエンジンをするがあります。

オンラインでHTMLやそののファイルをするをむ https://riptutorial.com/ja/node-js/topic/6538/html

やそののファイルをする

12: http

Examples

httpサーバー

HTTPサーバーのな。

http_server.jsファイルにのコードをします。

```
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)
}
```

http_server.jsのからこのコマンドをします。

```
node http_server.js
```

のがされます。

```
> Start HTTP on port 80
```

はサーバーをテストするがあります。インターネットブラウザをいてこのURLにするがあります

```
http://127.0.0.1:80
```

Linuxサーバをしているマシンであれば、のようにテストすることができます。

```
curl 127.0.0.1:80
```

のがされます。

```
ok
```

あなたのコンソ―ルでは、アプリケ―ションをすると、このがされます

```
> Request received: HTTP GET /
> Client IP: ::ffff:127.0.0.1
```

http クライアント

httpクライアントのな

のコ―ドをhttp_client.jsファイルにきみます。

```
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();
```

http_client.jsのからこのコマンドをします。

```
node http_client.js
```

のがされます。

```
> 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
```

このは、httpサーバーのにしています。

オンラインでhttpをむ https://riptutorial.com/ja/node-js/topic/2973/http

13: IISNodeをしてIISでNode.js Webアプリケーションをホストする

ビューをったディレクトリ/ネストされたアプリケーション

View EngineをしてExpressをしてビューをレンダリングする、ビューに`virtualDirPath`をすがあります

```
`res.render('index', { virtualDirPath: virtualDirPath });`
```

これをうは、アプリケーションとリソースパスによるのビューホストへのハイパーリンクを、にすべてのビューをすることなくサイトがホストされているをることができるようにするためです。これは、IISNodeでディレクトリをする、よりでなとしの1つです。

バージョン

のすべてのは

- Express v4.x
- IIS 7.x / 8.x
- Socket.io v1.3.x

Examples

IISNodeをすると、.NETアプリケーションのようにNode.js Web AppsをIIS 7/8でホストすることができます。もちろん、Windowsで`node.exe`プロセスをでホストすることはできますが、IISでアプリケーションをするだけのはですか。

IISNodeはのコアのプロセスmanageementオーバースケーリングする`node.exe`、およびをリサイクルアプリがされるたびに、ちょうどそのにあなたのIISアプリケーションのを。

IISでNode.jsアプリケーションをホストするには、IISNodeにいくつかのがあります。

1. Node.jsは、32ビットまたは64ビットのいずれかのIISホストにインストールするがあります。
2. IISNodeはx86またはx64をインストールしましたが、これはIISホストのビットとするがあります。
3. IISホストにインストールされたIISのMicrosoft URLきえモジュール。

- これはです。そうしないと、Node.jsアプリケーションへのリクエストはどおりにしません。

4. Node.jsアプリケーションのルートフォルダにある`Web.config`。
5. `Web.config`の`iisnode.yml`ファイルまたは`<iisnode>`をしたIISNode。

Expressをしたな Hello Worldの

このをにするには、IISホストでIIS 7/8アプリケーションをし、Node.js Web Appをむディレクトリをディレクトリとしてするがあります。アプリケーション/アプリケーションプールIDがNode.jsのインストールにアクセスできることをします。このでは、Node.jsの64ビットインストールをしています。

プロジェクトストラクチャ

これは、IISNode / Node.js Webアプリケーションのなプロジェクトです。 `Web.config`するは、IISNodeのWebアプリケーションとほぼじ`Web.config`。

```
- /app_root
  - package.json
  - server.js
  - Web.config
```

server.js - Expressアプリケーション

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

とWeb.config

`Web.config`は、のIIS `Web.configWeb.config`が、URL `<rewrite><rules>`とIISNode `<handler>` 2つがしなければなりません。これらのはとも`<system.webServer>`のです。

IISNodeは、 `iisnode.yml`ファイルをするか、 `<iisnode>`を`Web.config` `<system.webServer>`としてすることでできます。これらののがしかし、このには、いにみわせてすることができます`Web.config`するがあります`iisnode.yml`ファイルを**AND** ののからるだろう`iisnode.yml`わりにファイル 。 こののオーバーライドは、もありえません。

IISNodeハンドラ

IISが`server.js`にNode.js Web Appが`server.js`れていることをるには、にそれをえるがあります。これをうには、IISNode `<handler>`を`<handlers>`にし`<handlers>`。

```
<handlers>
  <add name="iisnode" path="server.js" verb="*" modules="iisnode"/>
</handlers>
```

URLきえルール

ののは、Node.jsアプリケーションがIISインスタンスにられることをしたトラフィックがIISNodeにけられていることをすることです。 URLきえルールがなければ、 `http://<host>/server.js`にアクセスしてアプリケーションをするがあり`http://<host>/server.js`さらにいことに、 `server.js`からされるリソースをリクエストしようとすると`404`がされます。これは、IISNode WebアプリケーションでURLのきえがなです。

```
<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>
```

これは、 64ビットのNode.jsインストールにされた、 こののの`Web.config`ファイルです 。

これで、IISサイトにアクセスし、Node.jsアプリケーションがすることをします。

IISディレクトリまたはネストされたアプリケーションをする

IISでディレクトリまたはネストされたアプリケーションをするのがなシナリオであり、おそらくIISNodeをするときにしたいとえています。

IISNodeは、ディレクトリまたはネストされたアプリケーションをでサポートしていないため、これをするためにのではなく、あまりられていないIISNodeのをするがあります。 AppSettingキーをして、 `Web.config`をつ`<appSettings>`のすべてのがプロパティとして`process.env`オブジェクトにされます。

`<appSettings>`ディレクトリにディレクトリをします。

```
<appSettings>
  <add key="virtualDirPath" value="/foo" />
</appSettings>
```

Node.jsアプリケーションで、 `virtualDirPath`にアクセスできます

```
console.log(process.env.virtualDirPath); // prints /foo
```

これでの`<appSettings>`をできるようになりました。これをして、サーバーコードでできます。

```
// 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');
});
```

virtualDirPathをリソースとともにすることもできます

```
// Public Directory
server.use(express.static(path.join(virtualDirPath, 'public')));
// Bower
server.use('/bower_components', express.static(path.join(virtualDirPath,
'bower_components')));
```

それらのすべてをまとめることができます

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

IISNodeでSocket.ioをする

IISNodeでSocket.ioをするには、 `Web.config`にディレクトリ/ネストされたアプリケーションをしないになのみが`Web.config`。

`/socket.io`は/socket.ioでまるをするので、IISNodeはIISにもするがあります。これはIISNodeもするがあり、ファイルやそののトラフィックではありません。これには、IISNodeアプリケーションとはなる`<handler>`がです。

```
<handlers>
    <add name="iisnode-socketio" path="server.js" verb="*" modules="iisnode" />
</handlers>
```

`<handlers>`にえて、のURLきえルールもするがあります。きえルールは、すべての`/socket.io`トラフィックを、Socket.ioサーバーがされているサーバーファイルにします。

```
<rule name="SocketIO" patternSyntax="ECMAScript">
    <match url="socket.io.+"/>
    <action type="Rewrite" url="server.js"/>
</rule>
```

IIS 8をしているは、 `Web.config` WebSocketsをにし、のハンドラとルールをきえるがあります。これはwebSocketのサポートがないためIIS 7ではです。

```
<webSocket enabled="false" />
```

オンラインでIISNodeをしてIISでNode.js Webアプリケーションをホストするをむ
https://riptutorial.com/ja/node-js/topic/6003/iisnodeをしてiisでnode-js-webアプリケーションをホストする

14: Koa Framework v2

Examples

Hello Worldの

```
const Koa = require('koa')

const app = new Koa()

app.use(async ctx => {
   ctx.body = 'Hello World'
})

app.listen(8080)
```

ミドルウェアをしたエラー

```
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
  }
})
```

オンラインでKoa Framework v2をむ https://riptutorial.com/ja/node-js/topic/6730/koa-framework-v2

15: MSSQLの

き

のデータベースをnodejsとするには、ドライバパッケージがです。また、npmモジュールとふこともできます。このモジュールは、データベースにしてをするためのなAPIをします。 mssqlデータベースでもじことがえます。ここでは、mssqlとnodejsをし、SQLタブでなクエリをします。

ローカルマシンでmssqlデータベースサーバーのローカルインスタンスをすることをしています。 このをしてじことをうことができます。

また、をしてしたなユーザーもしてください。

Examples

SQLでの。 mssql npmモジュール

まず、をつなノードアプリケーションをし、ローカルのSQL Serverデータベースにし、そのデータベースにしていくつかのクエリをします。

ステップ1するプロジェクトのでディレクトリ/フォルダをします。 *npm init*コマンドをしてノードアプリケーションをすると、のディレクトリにpackage.jsonがされます。

```
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.
```

ステップ2は、このディレクトリにApp.jsファイルをし、sql dbにするがあるパッケージをいくつかインストールします。

```
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
```

ステップ3ここでは、mssqlモジュールがをするためにするをアプリケーションにします。

```
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);
})
```

ステップ4これはアプリケーションをするもなステップです。アプリケーションはSQL Serverにし、なをします。

```
node App.js
// Output :
// Hello world, This is an app to connect to sql server.
// Connection Successful !
// 1
```

クエリのにまたはをするには、mssqlパッケージのドキュメントをしてください。

-
- /

オンラインでMSSQLのをむ https://riptutorial.com/ja/node-js/topic/9884/mssqlの

16: MySQLのプール

Examples

データベースなしのプールの

のデータベースがホストされているデータベースサーバーでマルチテナントをする。

、エンタープライズアプリケーションのマルチテナントがなであり、データベースサーバーのデータベースにプールをすることはおめしません。ですから、データベースサーバーとのプールをし、にじてデータベースサーバーにホストされているデータベースでりえることができます。

たちのアプリケーションが、データベースサバにホストされているごとになるデータベースをっているとします。ユーザーがアプリケーションにヒットすると、それぞれのデータベースにします。ここでそれをうのです

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

はをしましょう

プールをするとき、はデータベースをえず、データベースサバだけをえました。

```
{
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bobby',
  password        : 'pass'
}
```

データベースサーバーののデータベースをするは、のコマンドをしてヒットデータベースへのをねます。

```
    connection.changeUser({database : "firm1"});
```

あなたはのをここですることができます

オンラインでMySQLのプールをむ https://riptutorial.com/ja/node-js/topic/6353/mysqlのプール

17: MySQLの

き

このトピックでは、MYSQLデータベースツールをしてNode.jsとするをします。 nodejsプログラムとスクリプトをして、mysqlにするデータにしてするさまざまなをびます。

Examples

パラメータをしてオブジェクトをする

SQLでユーザーコンテンツをするは、パラメータをしています。たとえば、 `aminadav`というのユーザーをするは、 `aminadav`ようにします。

```
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.');
  }
});
```

プールの

a。のクエリをにする

MySQLのすべてのクエリは々にされます。つまり、10のクエリをし、クエリが2かかるは、をするのに20かかります。は、10のをし、それぞれのクエリをなるですることです。これは、プールをしてにうことができます

```
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
  });
```

```
 }
```

10のクエリをすべてにします。

`pool`をするときは、はありません。プールにいわせることができます。 MySQLモジュールは、のきをしてクエリをします。

b。なるデータベースがホストされているデータベースサーバーでのマルチテナントの

、エンタープライズアプリケーションのとしてマルチテナントがあり、データベースサーバーのデータベースにプールをすることはおめしません。ですから、データベースサーバーとのプールをし、にじてデータベースサーバーにホストされているデータベースでりえることができます。

たちのアプリケーションが、データベースサーバにホストされているごとになるデータベースをっているとします。ユーザーがアプリケーションにヒットすると、それぞれのデータベースにします。これをうのをにします。

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

はをしましょう

プールをするとき、はデータベースをえず、データベースサーバだけをえました。

```
{
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bobby',
  password        : 'pass'
}
```

データベースサーバーののデータベースをするは、のコマンドをしてヒットデータベースへのをねます。

```
    connection.changeUser({database : "firm1"});
```

あなたはのをここですることができます

MySQLにする

MySQLにするもなの1つは、 `mysql`モジュールをすることです。このモジュールは、Node.jsアプリケーションとMySQLサーバのをします。のモジュールとにインストールできます

```
npm install --save mysql
```

はmysqlをしなければなりません。これをでいわせることができます。

```
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();
```

のでは、 `connection`オブジェクトをするをします。

パラメータなしでオブジェクトをする

クエリをとしてし、をむコールバックをします。コールバックは`error` 、 `rows`とフィールドのを`rows`ます。には、されたテーブルのすべてのがまれます。ののためのです。

```
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)
});
```

プールからののでのクエリをする

MySQLのプールをセットアップしているがあるかもしれませんが、あなたはにしたいいくつかのクエリをっています

```
SELECT 1;
SELECT 2;
```

`pool.query` をしてすることもできますが 、プーールにきが1つしかないは、2のクエリをするにがになるまでつがあります。

ただし、 `pool.getConnection`をして1つのをしたいは、プーールからアクティブなをし、くのクエリをできます。

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

を`release`することをれないでください。そうしないと、プーールのりのでなMySQLが1つなくなります。

MySQLのプーールのについては、MySQLのドキュメントをしてください 。

エラーがしたときにクエリをします。

あなたは、あなたにされたクエリすることができ`err`エラーがしたときにオブジェクトを

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

プーールのエクスポート

```
// 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();
  });
});
```

オンラインでMySQLのをむ https://riptutorial.com/ja/node-js/topic/1406/mysqlの

18: N-API

き

N-APIは、NodeJSのネイティブモジュールをするためのしくれたです。 N-APIはにあるため、のないドキュメントがあるがあります。

Examples

こんにちはN-API

このモジュールはhelloモジュールのhelloをします。 helloは`printf`でコンソールでHello worldをし、ネイティブからjavascriptびしに`1373`をします。

```
#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)
```

オンラインでN-APIをむ https://riptutorial.com/ja/node-js/topic/10539/n-api

19: Node.js / Express.jsのMongoDBインテグレーション

き

MongoDBは、MEANスタックのおかげで、ものあるNoSQLデータベースの1つです。 Expressアプリケーションからのmongoデータベースとのインターフェースは、ちょっとしたをすればすばやくです。たちはMongooseをってたちをけます。

はこちらをください http : //mongoosejs.com/docs/guide.html

Examples

MongoDBのインストール

```
npm install --save mongodb
npm install --save mongoose //A simple wrapper for ease of development
```

サーバーファイルindex.jsまたはserver.jsという

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

マングースモデルの

```
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);
```

これをしましょう。 MongoDBとMongooseはJSONはBSONですが、ここではですをデ―タとしてしています。には、タイピングをらすためのいくつかのをしました。

は`new Schema`をし、それをにりてます。これはなJSONです。は、のあるスキ―マをするのにつプロパティをつのオブジェクトです。 Uniqueは、しいインスタンスをデ―タベ―スにして、らかににするようします。これは、ユ―ザ―がサ―ビスでのアカウントをするのをぐのにです。

はのもので、としてされています。のはブ―ルで、2のはまたはされるがしないはエラ―メッセ―ジです。

ObjectIdsは、モデルのにされます。たとえば、「ユ―ザ―にはくのコメントがあります。ObjectIdのわりにのをすることもできます。ユ―ザ―のようなはです。

に、APIルート―でするためにモデルをエクスポ―トすると、スキ―マにアクセスできます。

Mongoデ―タベ―スのクエリ

なGETリクエスト。ののモデルが`./db/models/Article.js`ファイルにあるとします。

```
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);
};
```

このエンドポイントにHTTPリクエストをすることで、デ―タベ―スからデ―タをできるようになりました。しかし、いくつかのな

1. はそれがどんなものかをにします。は5つのをしているだけです。
2. LeanはのBSONからかをりき、さとオ―バ―ヘッドをらします。です。しかし、です。
3. するは`find`のではなく、 `findOne` 、ことをして`doc.length`これが0よりもきい`find`にをします。それはさがチェックされていないり、そののは、あなたのエラ―をしません。
4. はにそのでエラ―メッセ―ジをするのがきです。ニ―ズにわせてしてください。されたドキュメントでもじです。

5. こののコードは、エクスプレス·サーバーにくのではなく、のファイルにいたことをとしてかれています。 サーバーでこれをびすには、サーバーコードにのをめます。

```
const app = express();
require('./path/to/this/file')(app) //
```

オンラインでNode.js / Express.jsのMongoDBインテグレーションをむ
https://riptutorial.com/ja/node-js/topic/9020/node-js---express-jsのmongodbインテグレーション

20: Node.js v6と

き

ノード6がノードのしいLTSバージョンになります。しいES6のにより、にするいくつかのがられます。されたのいくつかをし、のをします。

Examples

デフォルトのパラメータ

```
function addTwo(a, b = 2) {
    return a + b;
}

addTwo(3) // Returns the result 5
```

のパラメータをすることで、をオプションにして、デフォルトをあなたがんだにすることができます。

りのパラメータ

```
function argumentLength(...args) {
    return args.length;
}

argumentLength(5) // returns 1
argumentLength(5, 3) //returns 2
argumentLength(5, 3, 6) //returns 3
```

して、ののによって...にされるすべてのをとしてみまれます。このでは、のをし、それらのからされたのさをします。

スプレッドオペレータ

```
function myFunction(x, y, z) { }
var args = [0, 1, 2];
myFunction(...args);
```

では、のびしのまたはのリテラルのまたはのがなでをできます。りのパラメータのように、あなたのをにきします...

Arrowは、ECMAScript 6でをするしいです。

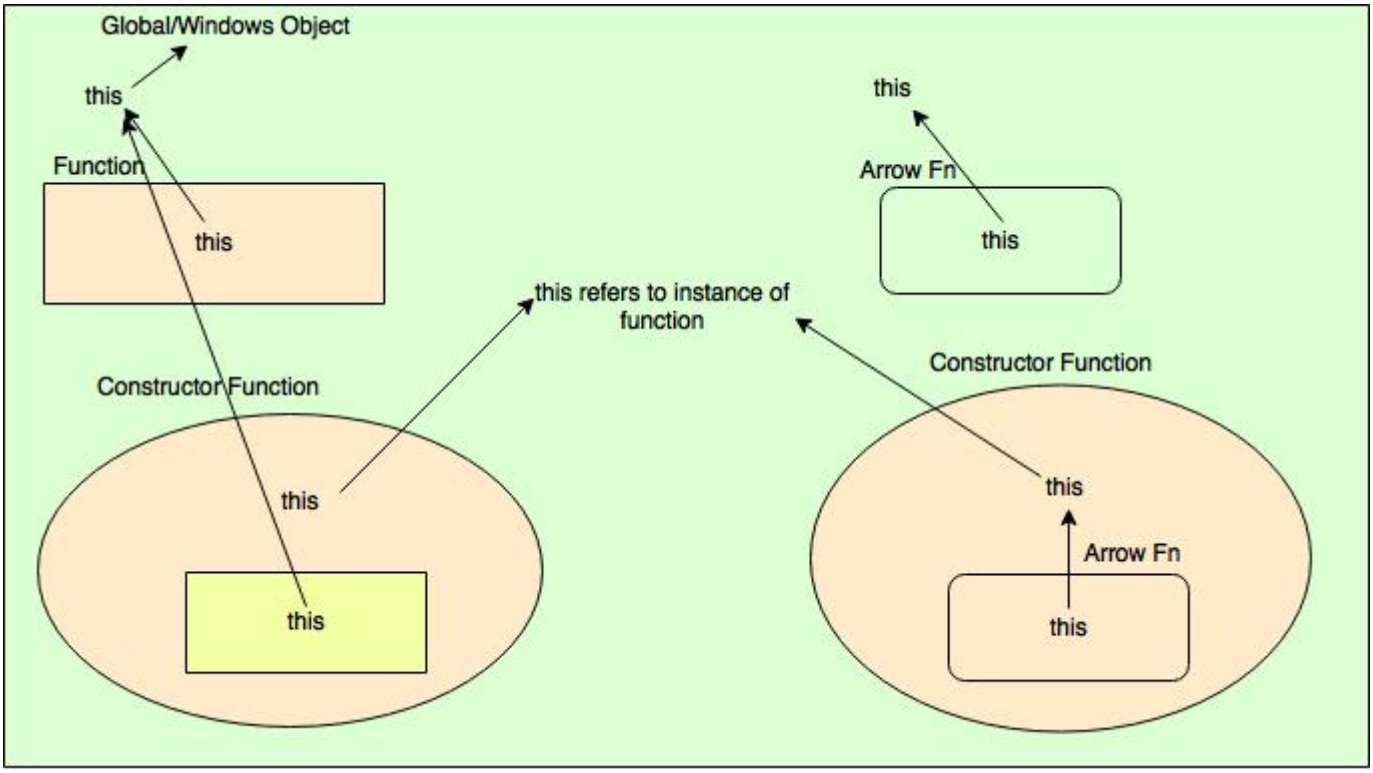
```
// traditional way of declaring and defining function
```

```
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;
}
```

の「this」

これは、でそのをびすためにされるインスタンス･オブジェクトをすが、これは 、のがされたのにしいです。

をってしよう



をってする。

```
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();
```

では、 これは、lexical scopeであり、がされているのスコープです。
のは、をするなであり、したがって、 これはグローバル/ウィンドウオブジェクトをしています。

2のでは、これはでされ、したがって、これは 、Windowsまたはグローバルオブジェクトである にされるをします。 3のでは、 これは、サービスオブジェクトがをびすためにされるため、サービスオブジェクトです。
4のでは、arrowはスコープが*service*であるからされ、びされます。したがって*service*オブジェクトをします。

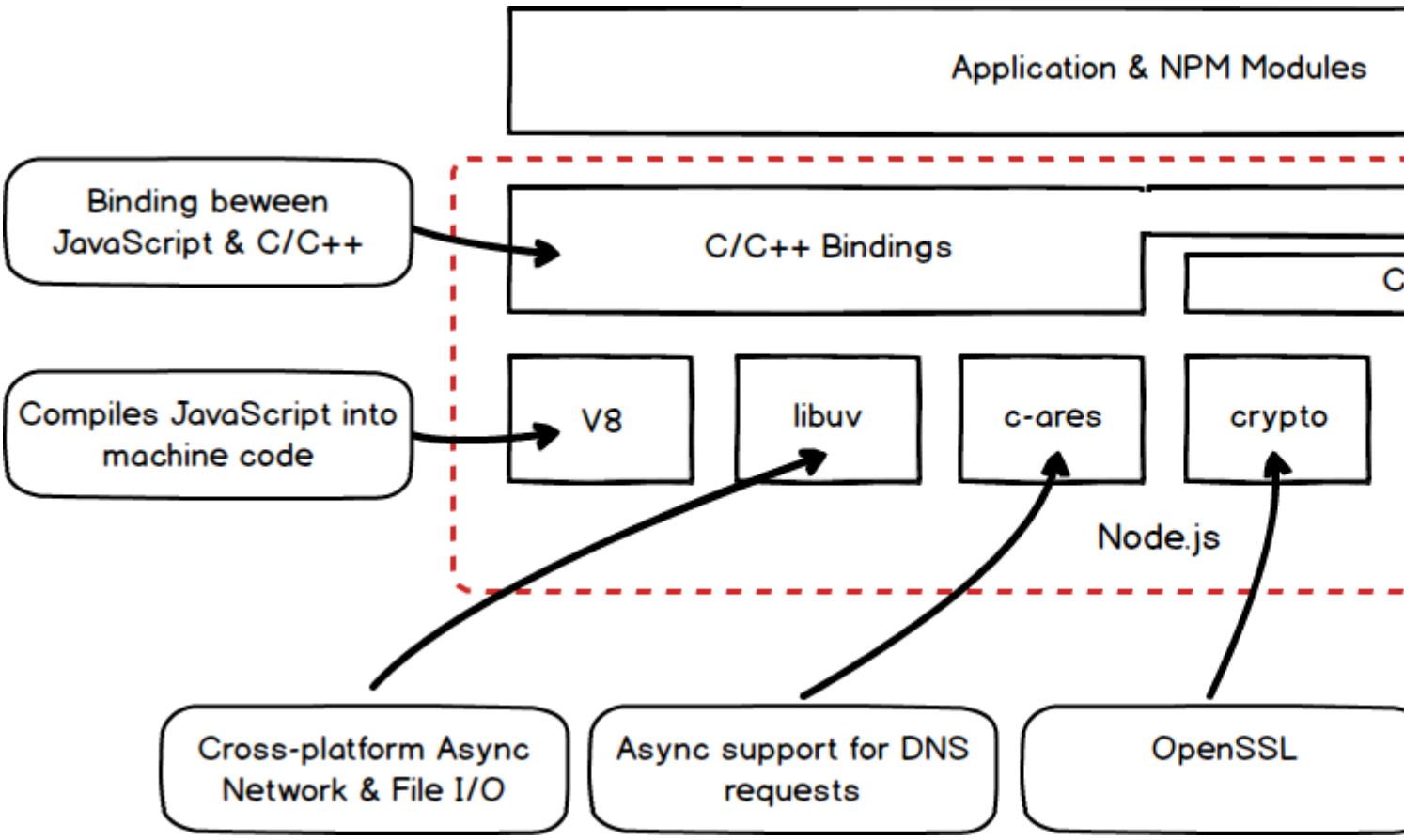
- グローバルオブジェクトは*Node.Js*に、*Windows*オブジェクトはブラウザにされます。

オンラインでNode.js v6とをむ https://riptutorial.com/ja/node-js/topic/8593/node-js-v6と

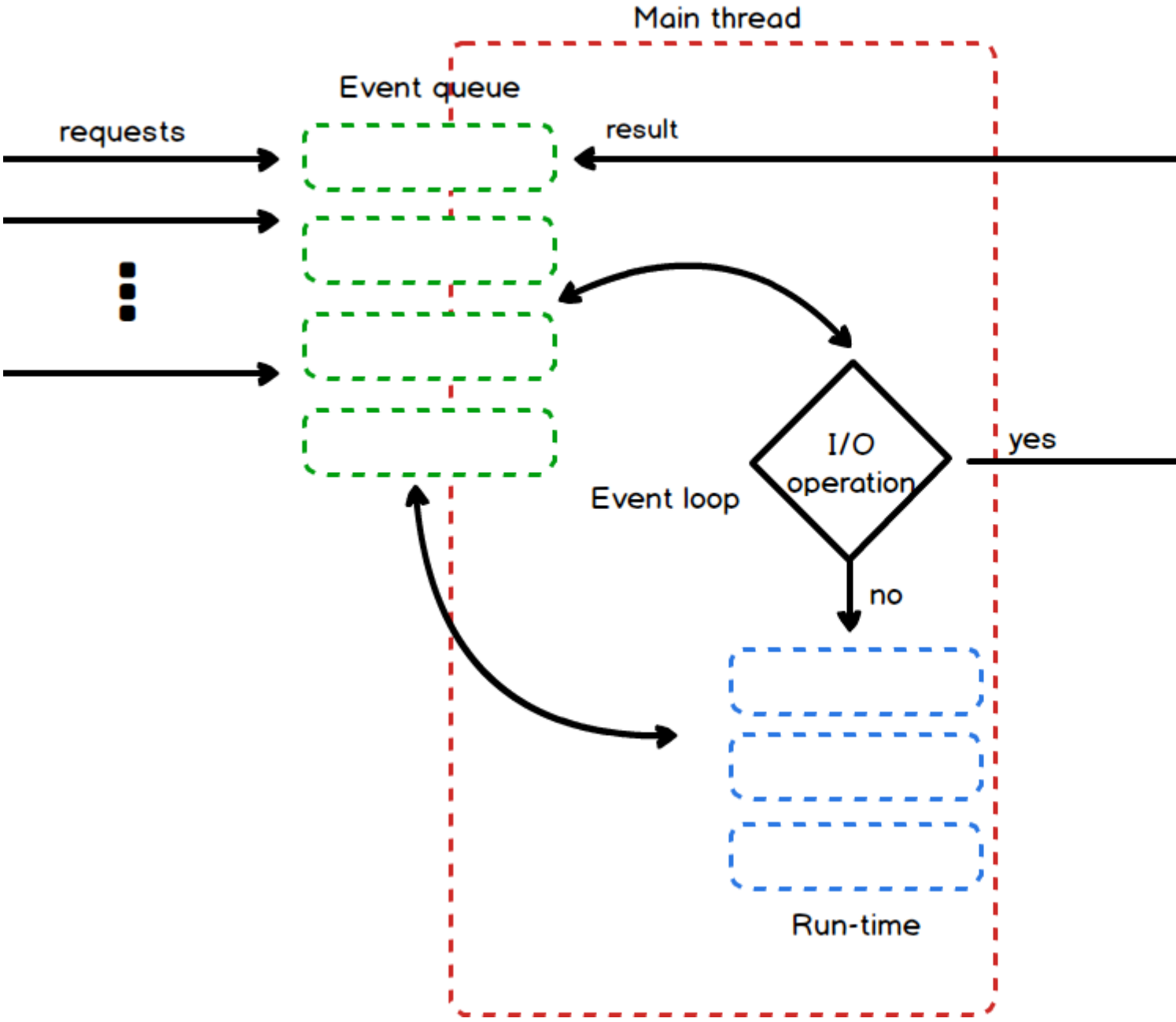
21: Node.jsアーキテクチャと

Examples

Node.js - フードの



Node.js - いている



オンラインでNode.jsアーキテクチャとをむ https://riptutorial.com/ja/node-js/topic/5892/node-jsアーキテクチャと

22: Node.jsアプリケーションのセキュリティ

Examples

クロスサイトリクエストCSRFの

CSRFはエンドユーザーに、されているWebアプリケーションでなアクションをさせるです。

これは、CookieがWebサイトにされるたびにリクエストがのサイトからされたでも、Cookieがされるためにします。

`csurf`モジュールをしてcsrfトークンをし、することができます。

```
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')
})
```

したがって、 `GET /form`にアクセスすると、csrfトークン`csrfToken`がビューに`csrfToken`れます。

さて、ビューで、のしフィールドのとしてcsrfTokenをし`_csrf`。

`handlebar`テンプレート

```
<form action="/process" method="POST">
    <input type="hidden" name="_csrf" value="{{csrfToken}}">
    Name: <input type="text" name="name">
    <button type="submit">Submit</button>
</form>
```

`jade`テンプレート

```
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")
```

ejsテンプレート

```
<form action="/process" method="POST">
    <input type="hidden" name="_csrf" value="<%= csrfToken %>">
    Name: <input type="text" name="name">
    <button type="submit">Submit</button>
</form>
```

Node.jsのSSL / TLS

Node.jsアプリケーションでSSL / TLSをすることをしたは、このでSSL / TLSをするもあるとえてください。くのサーバークライアントアーキテクチャでは、SSL / TLSはリバースプロキシでするため、アプリケーションのさがされ、セキュリティのがされます。

Node.jsアプリケーションでSSL / TLSをするがあるは、キーファイルとファイルをロードしてすることができます。

プロバイダにCAチェーンがなは、 caオプションでとしてできます。 1つのファイルにのエントリがあるチェーンはのファイルにされ、Node.jsは1つのファイルにのcaエントリをサポートしていないため、じでにするがあります。ファイル1_ca.crtおよび2_ca.crtのをのコードでしています。 caがで、しくされていない、クライアントブラウザはのをできなかったというメッセージをすることがあります。

```
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の

Node.jsのHTTPSサーバーののは、のようになります。

```
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);
```

httpリクエストをサポートしたいは、このさなをうがあります。

```
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);
```

HTTPSサーバーの

システムにnode.jsがインストールされたら、のにって、HTTPとHTTPSのをサポートしているWebサーバーをしてください。

1をする

1. キーとをするフォルダをします。

   ```
   mkdir conf
   ```

---

2. そのディレクトリにします

   ```
   cd conf
   ```

---

3. この`ca.cnf`ファイルをしてショートカットとしてする

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

---

4. このをしてしいをします。

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

---

5. `ca-key.pem`と`ca-cert.pem`にがあるので、サ―バ―のをしましょう

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

---

6. この`server.cnf`ファイルをしてショ―トカットとしてします。

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

---

7. のをしてをします。

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

---

8. リクエストにしてください

```
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
```

2をルートとしてインストールする

1. をルートのフォルダにコピーします。

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

---

2. CAストアをする

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

Secure express.js 3アプリケーション

express.jsをしてなをうためのバージョン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);
```

このようにして、ネイティブのhttp / httpsサーバーにミドルウェアをします

1024のポートでアプリケーションをしたいは、sudoコマンドされませんまたはリバースプロキシnginx、haproxyなどをするがあります。

オンラインでNode.jsアプリケーションのセキュリティをむ https://riptutorial.com/ja/node-js/topic/3473/node-jsアプリケーションのセキュリティ

23: Node.jsアプリケーションのデバッグ

Examples

コアnode.jsデバッガとノードインスペクタ

コアデバッガの

Node.jsはグラフィカルでないデバッグユーティリティでビルドをします。デバッガでビルドをするには、のコマンドをしてアプリケーションをします。

```
node debug filename.js
```

debugDemo.jsまれているのなNode.jsアプリケーションをえてみましょう

```
'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);
```

キーワードdebuggerは、コードのそのでdebuggerをします。

コマンドリファレンス

1. ステッピング

```
cont, c - Continue execution
next, n - Step next
step, s - Step in
out, o - Step out
```

2. ブレークポイント

```
setBreakpoint(), sb() - Set breakpoint on current line
setBreakpoint(line), sb(line) - Set breakpoint on specific line
```

のコードをデバッグするには、のコマンドをします。

```
node debug debugDemo.js
```

のコマンドがされると、のがされます。デバッガインタフェースをするには、 `process.exit()` とし
`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 $
```

`watch(expression)` コマンドをして、をして `restart` してアプリをし、デバッグするまたはをします
。

にコードをするには、 `repl` をし `repl` 。 replモードは、デバッグしているとじコンテキストをちます。これにより、のをべ、コードをテストすることができます。 `Ctrl+C` をして、デバッグ･レプリケーションをします。

みみノードインスペクタの

v6.3.0

ノードのv8インスペクタをみむことができます ノードインスペクタプラグインはもうありません
。

インスペクタのフラグをすだけで、インスペクタのURLがされます

```
node --inspect server.js
```

ノードインスペクタの

ノードインスペクタをインストールします。

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

node-debugコマンドでアプリケーションをします

```
node-debug filename.js
```

その、Chromeでヒット

```
http://localhost:8080/debug?port=5858
```

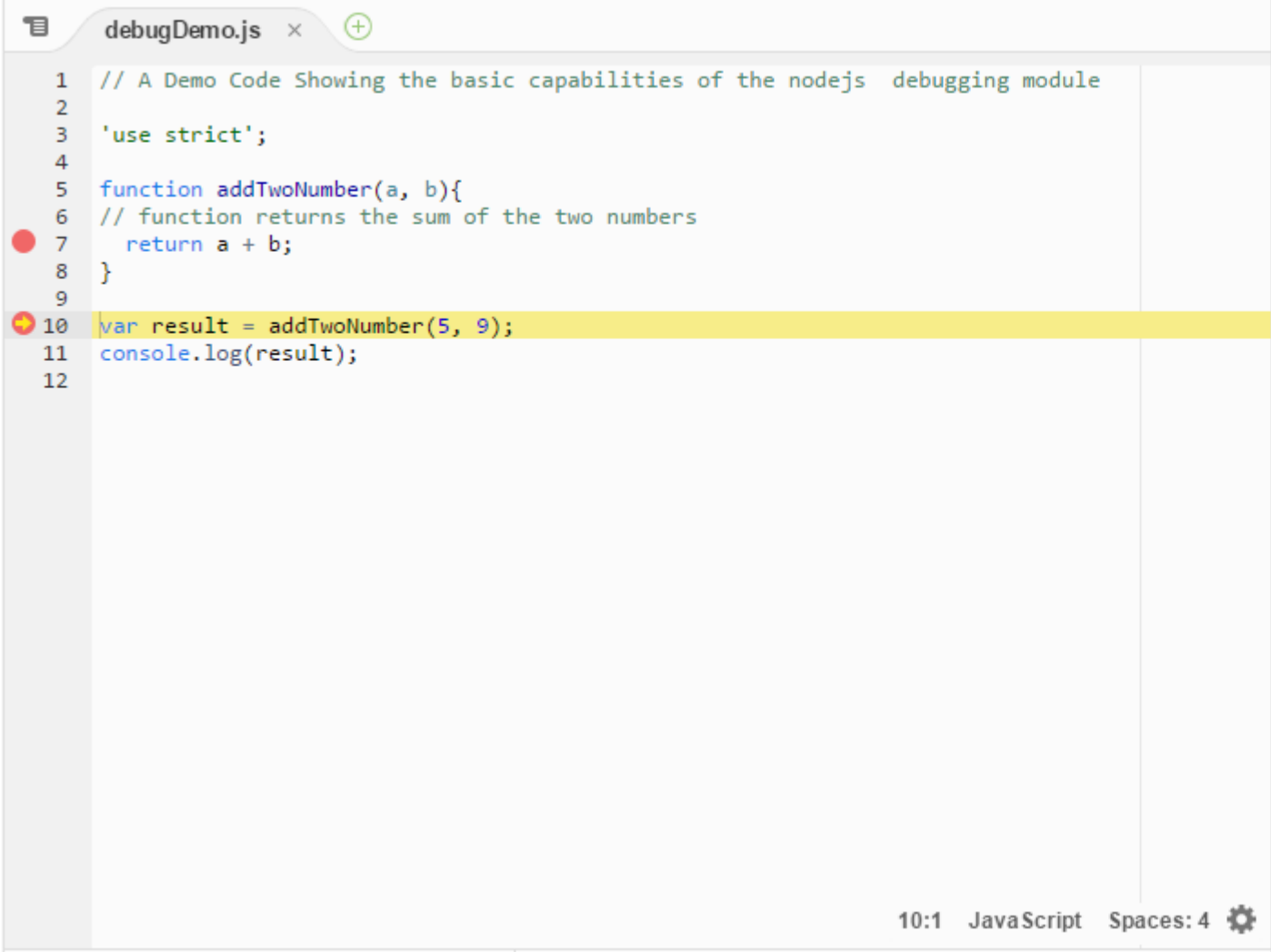
によっては、ポート8080がごのコンピューターでできないことがあります。のエラーがすることがあります。

    0.0.0.0:8080でサーバーをできません。エラーEACCESをきます。

この、のコマンドをして、のポートでノードインスペクタをします。

```
$node-inspector --web-port=6500
```

のようなものがされます



オンラインでNode.jsアプリケーションのデバッグをむ https://riptutorial.com/ja/node-js/topic/5900/node-jsアプリケーションのデバッグ

24: Node.jsエラー

き

Errorオブジェクトをすると、Node.jsでエラーをスローするをびます

エラーのベストプラクティスにするの。

Examples

エラーオブジェクトの

しいエラーメッセージ

`message`がされたオブジェクトの`message`プロパティにされているしいエラーオブジェクトをします。、`message`はErrorコンストラクタにとしてされます。しかし、`message`がではないオブジェクトの、Errorコンストラクタはされたオブジェクトの`.toString()`メソッドをびし、そのをされたエラーオブジェクトの`message`プロパティにします。

```
var err = new Error("The error message");
console.log(err.message); //prints: The error message
console.log(err);
//output
//Error: The error message
//    at ...
```

エラーオブジェクトにはスタックトレースがあります。スタックトレースには、エラーメッセージのがまれ、エラーがしたがされますのにはエラースタックがされます。エラーオブジェクトがされると、システムははののエラーのスタックトレースをします。スタックトレースをするには、されたエラーオブジェクトのスタックプロパティをします。の2はじです

```
console.log(err);
console.log(err.stack);
```

げるエラー

がされず、ノード・サーバーがクラッシュする、げみエラーはをします。

のはエラーをスローします

```
throw new Error("Some error occurred");
```

または

```
var err = new Error("Some error occurred");
throw err;
```

または

```
throw "Some error occurred";
```

のをげるはいではなく、されませんにErrorオブジェクトのインスタンスであるエラーをげる。

あなたがあなたのにエラーを`throw`た、システムはそのでクラッシュしますハンドラがない、そののにコードはされません。

```
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
```

しかし、このでは

```
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
```

try ... catchブロック

try ... catchブロックはをするためのもので、はエラーではなくスローされたエラーをします。

```
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
}
```

`try`ブロック`b++`エラーがし、そのエラーが`catch`ブロックにされます`catch`ブロックには、catchブロックでじエラーがスローされたり、スローされたり、スローされたりすることがあります。のをてみましょう。

```
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;
}
```

のでは、 `error`オブジェクトの`message`プロパティをし、された`error`スローし`error`。

tryブロックのエラーをすべてべ、catchブロックでできます。

```
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
}
```

オンラインでNode.jsエラーをむ https://riptutorial.com/ja/node-js/topic/8590/node-jsエラー

25: Node.jsデザインの

Examples

Node.jsの

コア 、 モジュール -

およびのモジュールを、コードサイズのみではなく、のをたすスコープのにする

```
    a - "Small is beautiful"
    b - "Make each program do one thing well."
```

パターン

Reactor Patternは`node.js`のです。にされるイベントループのけをりて、のイベントジェネレータとイベントハンドラをして、システムをシングルスレッドプロセスとしてすることをしました。

Node.jsのノンブロッキング**I / O**エンジン **- libuv -**

オブザーバー・パターン EventEmitterは、/オブザーバーのリストをし、それらにします

```
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');
```

オンラインでNode.jsデザインのをむ https://riptutorial.com/ja/node-js/topic/6274/node-jsデザインの

26: Node.jsでのAPIの

Examples

ExpressをしてAPIをする

`Node.js` apisは、 `Express` Webフレームワークでにできます。

のは、すべてのユーザーをするためのな`GET` APIをします。

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

ExpressをしたPOST API

のでは、 `Express`をして`POST` APIをします。このは、 `GET`にていますが、データをして`req.body`する`body-parser`をするがなります。

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

オンラインでNode.jsでのAPIのをむ https://riptutorial.com/ja/node-js/topic/5991/node-jsでのapiの

27: Node.jsでのPOSTリクエストの

Node.jsはストリームをしてデータをします。

ドキュメントからすると、

> ストリームは、Node.jsのストリーミングデータをするためのなインターフェイスです。ストリームモジュールは、ストリームインターフェイスをするオブジェクトをにできるAPIをします。

POSTのでするには、みりなストリームである`request`オブジェクトをします。データストリームは、 `request`オブジェクトの`data`イベントとしてされます。

```
  request.on('data', chunk => {
    buffer += chunk;
  });
  request.on('end', () => {
    // POST request body is now available as `buffer`
  });
```

にのバッファをし、 `data`イベントをしてしたバッファデータをします。

1. `data`イベントでされたバッファデータのタイプはBufferです
2. ごとにデータイベントからバッファリングされたデータをするためのしいバッファをします。つまり、ハンドラに`buffer`をします。

Examples

POSTリクエストをするnode.jsサーバのサンプル

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

オンラインでNode.jsでのPOSTリクエストのをむ https://riptutorial.com/ja/node-js/topic/5676/node-jsでのpostリクエストの

28: Node.JSでのWebSocketの

Examples

WebSocketをインストールする

あなたのプロジェクトにWebSocketをインストールするにはいくつかのがあります。はです

```
npm install --save ws
```

あなたのpackage.jsonでをします

```
"dependencies": {
  "ws": "*"
},
```

WebSocketをファイルにする

ファイルにwsをするには、のようにします

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

WebSocketとWebSocketサーバーの

しいWebSocketをくには、にのようにします

```
var WebSocket = require("ws");
var ws = new WebSocket("ws://host:8080/OptionalPathName);
// Continue on with your code...
```

または、サーバーをくには、のようにします。

```
var WebSocketServer = require("ws").Server;
var ws = new WebSocketServer({port: 8080, path: "OptionalPathName"});
```

シンプルなWebSocketサーバーの

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

オンラインでNode.JSでのWebSocketのをむ https://riptutorial.com/ja/node-js/topic/6106/node-jsでのwebsocketの

29: node.jsでのWindows

activedirectory2やadldapなど、いくつかのActive Directory APISがあります。

Examples

activedirectoryの

のは、ここGitHubまたはここNPMでなドキュメントからしたものです。

インストール

```
npm install --save activedirectory
```

```
// 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!');
    }
});
```

オンラインでnode.jsでのWindowsをむ https://riptutorial.com/ja/node-js/topic/10612/node-jsでのwindows

30: node.jsでのモジュールのエクスポートとインポート

Examples

node.jsのなモジュールをする

node.jsモジュールとは へのリンク

> モジュールは、するコードをのコードにカプセルします。モジュールをするときは、するすべてのをファイルにするとできます。

さあ、をてみましょう。すべてのファイルがじディレクトリにあるとします。

ファイル `printer.js`

```
"use strict";

exports.printHelloWorld = function (){
    console.log("Hello World!!!");
}
```

モジュールをするの

ファイル`animals.js`

```
"use strict";

module.exports = {
    lion: function() {
        console.log("ROAARR!!!");
    }

};
```

ファイル `app.js`

このファイルをするには、ディレクトリにし、のようにし`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!!!"
```

ES6でのインポートの

Node.jsは、V8のバージョンにしてされています。このエンジンのリリースをのにつことにより、JavaScript ECMA-262のがNode.jsにタイムリーにされ、なパフォーマンスとのがされます。

すべてのECMAScript 2015ES6のは、、ステージング、およびのの3つのグループにされています。

V8がしているとみなすすべてのは、Node.jsではデフォルトでオンになっており、ランタイムフラグはありません。 V8チームによってしているとはみなされないほぼしたであるステージは、ランタイムフラグをとします。--harmony。のフィーチャは、それぞれのハーモニーフラグによってにアクティブにすることができますが、テストのはけてください。これらのフラグはV8によってされており、なしでされるがあります。

、ES6はインポートステートメントをネイティブでサポートしていますす

ですから、 `fun.js`というファイルがあるとし`fun.js` ...

```
export default function say(what){
  console.log(what);
}

export function sayLoud(whoot) {
  say(whoot.toUpperCase());
}
```

...そして`app.js`というのファイルがあって、したをしたいは、それらをインポートするが3つあります。

デフォルトをインポート

```
import say from './fun';
say('Hello Stack Overflow!!');  // Output: Hello Stack Overflow!!
```

ソースファイルのデフォルトのエクスポートとしてマークされているため、 `say()`をインポートし`say()`デフォルトのエクスポート`export default …`

きインポート

```
import { sayLoud } from './fun';
sayLoud('JS modules are awesome.'); // Output: JS MODULES ARE AWESOME.
```

きインポートでは、になモジュールのをにインポートすることができます。これをにすることでいます。たちのは、importステートメントので`sayLoud`というをけます。

バンドルされたインポート

```
import * as i from './fun';
```

```
i.say('What?'); // Output: What?
i.sayLoud('Whoot!'); // Output: WHOOT!
```

たちがそれをっていたいなら、これがくです。 `* as i`をすることで、たちは`import`ステートメントをって、たちの`fun`モジュールのすべてのエクスポートをするきプロパティとしてするオブジェクト`i`をします。

パス

`./`をしてインポートするファイルとじディレクトリにインポートするファイルがするでも、インポートパスをパスとしてにマークするがあります。のようなされていないパスからのインポート

```
import express from 'express';
```

ローカルおよびグローバル`node_modules`フォルダでされ、するモジュールがつからないはエラーがします。

ES6のでエクスポートする

これはのとじですが、わりにES6をしています。

```
export function printHelloWorld() {
  console.log("Hello World!!!");
}
```

オンラインでnode.jsでのモジュールのエクスポートとインポートをむ
https://riptutorial.com/ja/node-js/topic/1173/node-jsでのモジュールのエクスポートとインポート

31: Node.JSとMongoDB。

これは、mongo dbとnodejをするためのなCRUDです。

ここでやることができるのはありますか

えはい、これをうはたくさんあります。

マングースをしていますか

いいえ。あなたにつそののパッケージがあります。

どこでmongooseのをできますか

え ここをクリック

Examples

データベースへの

ノードアプリケーションからmongoデータベースにするには、mongooseがです。

Mongooseをインストールするあなたのアプリケーションのトートにき、Mongooseをインストールする

```
npm install mongoose
```

に、データベースにします。

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

しいコレクションをする

Mongooseでは、すべてがスキーマからしています。スキーマをできます。

```
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');
```

のように、mongoose.modelでコンパイルするに、メソッドをスキーマにするがあります。

ドキュメントをする

コレクションにしいをするために、スキーマのオブジェクトをします。

```
var Auto = require('models/auto')
var autoObj = new Auto({
    name: "NewName",
    countOf: 10
});
```

たちはのようにします

```
autoObj.save(function(err, insertedAuto) {
    if (err) return console.error(err);
    insertedAuto.speak();
    // output: Hello this is NewName and I have counts of 10
```

```
});
```

これにより、しいがコレクションにされます

コレクションからデ―タをむことはにです。コレクションのすべてのデ―タをします。

```
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);
})
```

きのデ―タのみり

```
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);
})
```

なすべてのフィ―ルドのオブジェクトとして2のパラメ―タをすることもできます

```
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);
})
```

コレクションの1つのをする。

```
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);
})
```

コレクションの1つのドキュメントをIDでします。

```
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);
})
```

コレクションとドキュメントをするには、のいずれかのをできます。

メソッド

-

- updateOne
- updateMany
- replaceOne

---

*update*メソッドは、1つまたはのドキュメントパラメータをします。

```
db.lights.update(
   { room: "Bedroom" },
   { status: "On" }
)
```

このは、 `room`が**Bedroom** *1*パラメータであるを 'lights'コレクションでします 。に、するドキュメントの`status`プロパティを**On** *2*のパラメータにし、のようなWriteResultオブジェクトをします。

```
{ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }
```

UpdateOne

*UpdateOne*メソッドは、1つのドキュメントパラメータをします。

```
db.countries.update(
   { country: "Sweden" },
   { capital: "Stockholm" }
)
```

このでは、 'countries'コレクションで`country`が**Sweden** *1*パラメータのドキュメントがされます。その、するドキュメントプロパティの`capital`を**Stockholm** *2*のパラメータにし、のようなWriteResultオブジェクトをします。

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

UpdateMany

*UpdateMany*メソッドは、のドキュメントパラメータをします。

```
db.food.updateMany(
   { sold: { $lt: 10 } },
   { $set: { sold: 55 } }
)
```

どここのは、「」コレクションのすべてのドキュメントを`sold`することにより、10 *パラメータよりもさいです`sold` **55**へ。に、のようなWriteResultオブジェクトをします。

```
{ "acknowledged" : true, "matchedCount" : a, "modifiedCount" : b }
```

a =するの
b =されたの

ReplaceOne

にした

countriesというこのサンプルコレクションには、3つのドキュメントがまれています。

```
{ "_id" : 1, "country" : "Sweden" }
{ "_id" : 2, "country" : "Norway" }
{ "_id" : 3, "country" : "Spain" }
```

のは、`{ country: "Spain" }`を`{ country: "Finland" }`きえます

```
db.countries.replaceOne(
   { country: "Spain" },
   { country: "Finland" }
)
```

そして、

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

のにはがまれています

```
{ "_id" : 1, "country" : "Sweden" }
{ "_id" : 2, "country" : "Norway" }
{ "_id" : 3, "country" : "Finland" }
```

マングースのコレクションからをするには、のようにします。

```
Auto.remove({_id:123}, function(err, result){
    if (err) return console.error(err);
    console.log(result); // this will specify the mongo default delete result.
});
```

オンラインでNode.JSとMongoDB。をむ https://riptutorial.com/ja/node-js/topic/7505/node-jsとmongodb-

32: Node.jsのアンインストール

Examples

Mac OSXでNode.jsをにアンインストールする

Macオペレーティングシステムのターミナルで、の2つのコマンドをします。

```
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.*
```

WindowsでのNode.jsのアンインストール

WindowsでNode.jsをアンインストールするには、のようにプログラムのとをいます

1. スタートメニューから[`Add or Remove Programs`]をきます。
2. `Node.js`します。

ウィンドウズ**10**

3. Node.jsをクリックします。
4. [アンインストール]をクリックします。
5. しいアンインストールボタンをクリックします。

Windows 7-8.1

3. Node.jsのの「Uninstall」ボタンをクリックします。

オンラインでNode.jsのアンインストールをむ https://riptutorial.com/ja/node-js/topic/2821/node-jsのアンインストール

33: Node.jsのインストール

Examples

UbuntuにNode.jsをインストールする

aptパッケージマネージャをする

```
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)
```

のバージョンのバージョンえば、LTS 6.xを nodesourceからする

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
apt-get install -y nodejs
```

また、グローバルなnpmモジュールをしくインストールするには、それらのディレクトリをしますsudoのをし、EACCESエラーをします。

```
mkdir ~/.npm-global
echo "export PATH=~/.npm-global/bin:$PATH" >> ~/.profile
source ~/.profile
npm config set prefix '~/.npm-global'
```

WindowsにNode.jsをインストールする

インストール

すべてのNode.jsバイナリ、インストーラ、およびソースファイルをここからダウンロードできます 。

`node.exe`ランタイムのみをダウンロードするか、Windowsインストーラ `.msi` をして、 `npm` 、Node.jsのパッケージマネージャ、およびパスをインストールします。

パッケージマネージャによるインストール

パッケージマネージャ*Chocolatey* Software Management Automationによってインストールすることもできます。

```
# choco install nodejs.install
```

のバージョンのについては、chocoリポジトリをしてください 。

ノードバージョンマネージャ**nvm**の

ノードバージョンマネージャー nvmは、のNode.jsバージョンのをするbashスクリプトです。

nvmをインストールするには、されているインストールスクリプトをします。

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

Windowsのは、インストーラきのnvm-windowsパッケージがあります。このGithHubページには、nvm-windowsパッケージのインストールとにするがあります。

nvmをインストールしたら、コマンドラインから "nvm on"をします。これにより、nvmはノードのバージョンをできます。

しくインストールされた`nvm`コマンドをするために、をするがあります。

に、のNodeバージョンをインストールします。

```
$ nvm install node
```

メジャーバージョン、マイナーバージョン、および/またはパッチバージョンをして、のノードバージョンをインストールすることもできます。

```
$ nvm install 6
$ nvm install 4.2
```

インストールなバージョンをするには

```
$ nvm ls-remote
```

インストールとじように、バージョンをすことでバージョンをりえることができます

```
$ nvm use 5
```

インストールしたNodeののバージョンを、のようにしてデフォルトのバージョンにできます 。

```
$ nvm alias default 4.2
```

マシンにインストールされているノードのバージョンのリストをするには、のようにします。

```
$ nvm ls
```

プロジェクトのノードバージョンをするには、バージョンを.nvmrcファイルにします。こうすることで、のプロジェクトでをするに、リポジトリからフェッチしたにエラーがしにくくなります。

```
$ echo "4.2" > .nvmrc
$ nvm use
Found '/path/to/project/.nvmrc' with version <4.2>
Now using node v4.2 (npm v3.7.3)
```

Nodeがnvmでインストールされている、グローバルパッケージをインストールするために`sudo`をするはありません。ホームパッケージにインストールされているためです。したがって、 `npm i -g http-server`はパーミッションエラーなし で`npm i -g http-server`。

APTパッケージマネージャをしてSourceからNode.jsをインストールする

```
sudo apt-get install build-essential
sudo apt-get install python

[optional]
sudo apt-get install git
```

ソースをしてビルドする

```
cd ~
git clone https://github.com/nodejs/node.git
```

またはのLTS Node.jsバージョン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
```

`cd ~/node-v6.10.2`などのソースディレクトリに`cd ~/node-v6.10.2`

```
./configure
make
sudo make install
```

パッケージマネージャをしてMacにNode.jsをインストールする

Homebrewパッケージマネージャをしてインストールできます。

brewをしてします。

```
brew update
```

アクセスやパスをするがあるかもしれません。にむにこれをするのがです

```
brew doctor
```

にNode.jsをインストールするには、のコマンドをします。

```
brew install node
```

Node.jsがインストールされたら、のコマンドをして、インストールされているバージョンをできます。

```
node -v
```

Macports

Macportsでnode.jsをインストールすることもできます。

のパッケージがされていることをするためにまずそれをしてください

```
sudo port selfupdate
```

に、nodejsとnpmをインストールします。

```
sudo port install nodejs npm
```

あなたは、びすことにより、CLIをしてノードをすることができ`node`。また、のノードのバージョンを

```
node -v
```

MacOS Xインストーラをってインストールする

インストーラはNode.jsのダウンロードページにあります。、Node.jsはLTSバージョンサポートとのバージョンリリースの2つのバージョンのノードをします。 Nodeをいれていないは、LTSをして、 `Macintosh Installer`ボタンをクリックしてパッケージをダウンロードしてください。

あなたは、のNodeJSリリースをしにきたいは、ここでは、あなたのリリースをし、ダウンロードをクリックします。ダウンロードページから、が`.pkg`ファイルをします。

ファイルをダウンロードしたら`.pkg` ofcourse、ダブルクリックしてインストールします。 `Node.js`と`npm`いるインストーラは、デフォルトでパッケージはインストールされますが、 `Installation Type`ステップで`customize`ボタンをクリックすることで、インストールするものを`customize`ます。

それは、インストールにうだけでです。

ノードがインストールされていることをする

オープンterminal terminalをくがわからないは、このwikihowをてください。ターミナルタイプのnode --versionをします。 Nodeがインストールされている、はのようになります。

```
$ node --version
v7.2.1
```

v7.2.1はNode.jsのバージョンです。メッセージcommand not found: nodeなくcommand not found: nodeであるは、インストールにがあることをします。

Raspberry PIへのNode.jsのインストール

v6.xをインストールするには、パッケージをします。

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
```

aptパッケージマネージャをする

```
sudo apt-get install -y nodejs
```

のシェルのにノードマネージャーをインストールしてOh My Fish

ノード・バージョン・マネージャ nvm は、Node.jsバージョンのとインストールをにし、パッケージをうときにsudoのをりきますたとえば、 npm install ...。 Fish Shell fish は、*OS X*、Linux、およびそののファミリーけのスマートでいやすいコマンドラインシェルです。これは、 bashなどのなシェルにするプログラマのでのあるです。に、 Oh My Fish omf は、Fishシェルでパッケージをカスタマイズしてインストールすることができます。

このガイドでは、すでにあなたのシェルとして**Fish**をしていることをとしています。

nvmをインストールする

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.4/install.sh | bash
```

オー・マイ・フィッシュをインストールする

```
curl -L https://github.com/oh-my-fish/oh-my-fish/raw/master/bin/install | fish
```

このでをするようされますので、すぐしてください。

Oh My Fishのための**plugin-nvm**のインストール

はOh My Fishでplugin-nvmをインストールし、Fishシェルのnvmをします

```
omf install nvm
```

Node Version ManagerでNode.jsをインストールする

これで、 nvmをするがいました。あなたは、みのNode.jsのバージョンをインストールしてすることができます。いくつかの

- のノードバージョンを`nvm install node`
- 6.3.1をに`nvm install 6.3.1` `nvm install 6.3.1`
- インストールされているVerisonsのリスト `nvm ls`
- にインストールされた4.3.1にりえる `nvm use 4.3.1`

ファイナルノート

このメソッドをってNode.jsをうときに、もはやsudoがないことをえておいてくださいノードのバージョン、パッケージなどは、ホームディレクトリにインストールされます。

CentOS、RHEL、FedoraのソースからNode.jsをインストールする

- git
- `clang`と`clang++` 3.4 ^または`gcc`と`g++` 4.8 ^
- Python 2.6または2.7
- GNU Make 3.81 ^

ソースをする

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
```

ビルド

```
cd node
./configure
make -jX
su -c make install
```

X - プロセッサコアのがにえる

クリーンアップ[オプション]

```
cd
rm -rf node
```

nでNode.jsをインストールする

まず、あなたのシステムにnをするためのらしいラッパーがあります。とにかくれ

```
curl -L https://git.io/n-install | bash
```

nをインストールします。に、さまざまなでバイナリをインストールします。

```
n latest
```

した

```
n stable
```

lts

```
n lts
```

そののバージョン

```
n <version>
```

```
n 4.4.7
```

このバージョンがにインストールされている、このコマンドはそのバージョンをにします。

バージョンのりえ

nが、インストールされているバイナリのリストをします。とをしてのものをし、Enterをしてそれをにします。

オンラインでNode.jsのインストールをむ https://riptutorial.com/ja/node-js/topic/1294/node-jsのインストール

34: Node.jsのパフォーマンス

Examples

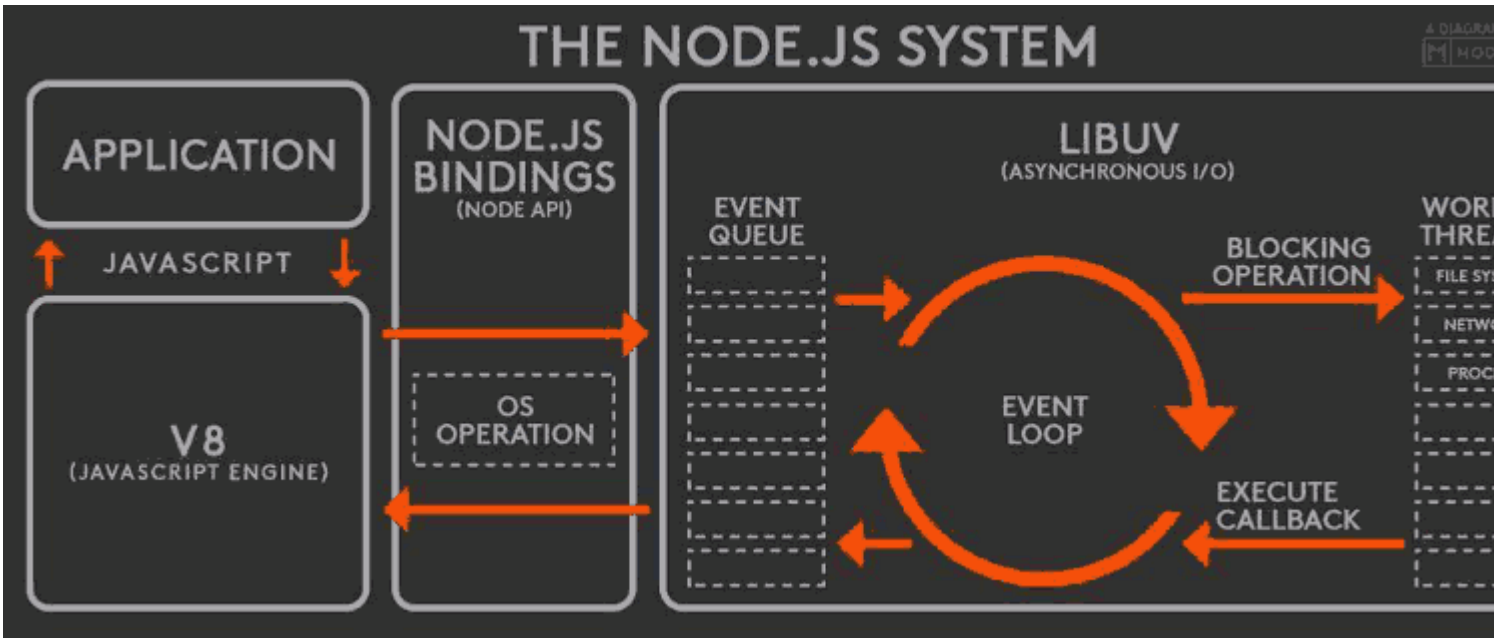
イベントループ

ブロッキングの

```
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)
```

ブロッキングIOの

```
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)
```



にえば、Event Loopは、をするまでCPUバインドコードをし、IOブロックコードをブロッキングでするシングルスレッドキューメカニズムです。

しかし、カーペットのにあるNode.jsは、 libuvライブラリをして、そののいくつかのためにマルチスレッドをします。

パフォーマンスにする

- ノンブロッキングはキューをブロックせず、ループのパフォーマンスにをえません。
- ただし、CPUバウンドではキューがブロックされるため、Node.jsコードでCPUバウンドをわないようにしてください。

Node.jsは、オペレーティングシステムカーネルにをさせるためIOをブロックにし、IOがデータをイベントとして すると、されたコールバックでコードをします。

maxSocketsをやす

```
require('http').globalAgent.maxSockets = 25

// You can change 25 to Infinity or to a different value by experimenting
```

Node.jsはデフォルトで`maxSockets = Infinity`をにしています v0.12.0。ノードv0.12.0まで、デフォルトは`maxSockets = 5` でした v0.11.0。したがって、5つのリクエストの、それらはキューにれられます。がなは、このをやしてください。

のエージェントをする

`http` APIは「 グローバルエージェント 」をしています。あなたのエージェントをすることができます。このような

```
const http = require('http')
const myGloriousAgent = new http.Agent({ keepAlive: true })
myGloriousAgent.maxSockets = Infinity

http.request({ ..., agent: myGloriousAgent }, ...)
```

ソケットプールをにする

```
const http = require('http')
const options = {.....}

options.agent = false

const request = http.request(options)
```

とし

- じがなは、 `https` APIでじことをうがあり`https`
- たとえば、 AWSでは`Infinity`ではなく、50がされることにしてください。

gzipをにする

```
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)
```

オンラインでNode.jsのパフォーマンスをむ https://riptutorial.com/ja/node-js/topic/9410/node-jsのパフォーマンス

35: Node.JSのリモートデバッグ

Examples

NodeJSのコンフィギュレーション

ノードリモートデバッグをするには、ノードプロセスを`--debug`フラグでするだけです。 `--debug=<port>`をして、デバッガをするポートをできます。

ノードプロセスがすると、メッセージがされます

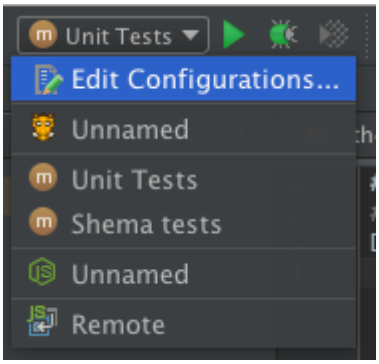
```
Debugger listening on port <port>
```

それは、すべてがいとあなたにえます。

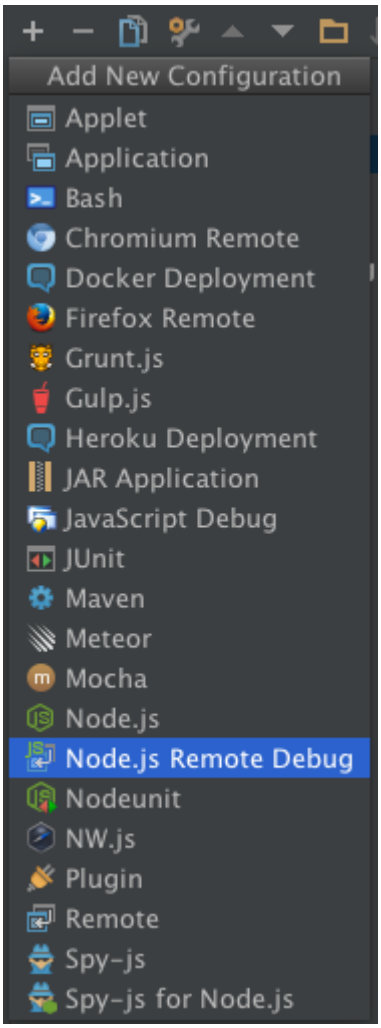
に、のIDEにリモートデバッグターゲットをします。

IntelliJ / Webstormの

1. NodeJSプラグインがになっていることをしてください
2. コンフィギュレーションをしてください



3. **+** > **Node.js** リモートデバッグを



4. でしたポートとしいホストをしてください

これらのがしたら、どおりデバッグターゲットをするだけで、ブレークポイントでします。

Linuxのポートでデバッグするためにプロキシをする

Linuxでアプリケーションをするは、ポートのデバッグにプロキシをします。たとえば、のようにします。

```
socat TCP-LISTEN:9958,fork TCP:127.0.0.1:5858 &
```

リモートデバッグにはポート9958をします。

オンラインでNode.JSのリモートデバッグをむ https://riptutorial.com/ja/node-js/topic/6335/node-jsのリモートデバッグ

36: Node.jsの

Examples

HTTPサーバー

```
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);
});
```

コマンドプロンプトをしたコンソール

```
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();
```

オンラインでNode.jsのをむ https://riptutorial.com/ja/node-js/topic/7703/node-jsの

37: node.jsをサーービスとしてする

き

くのWebサーバーとはなり、Nodeは、そのままのではサーービスとしてインストーールされません。しかし、プロダクションでは、initシステムによってされるdæmonとしてするがいです。

Examples

systemddæmonとしてのNode.js

systemdは、ほとんどのLinuxディストリビューションのの initシステムです。 Nodeがsystemdでするようにされた、 `service`コマンドをしてそれをすることができます。

まず、ファイルがです。しましょう。 Debianのベーースのディストリビューションのは、それがになります`/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
```

のようにして、それぞれアプリケーションの、、がになりました。

```
service node start
service node stop
service node restart
```

ににnodeをするようにsystemdにするには、に`systemctl enable node`。

つまり、ノードは、デーモンとしてされています。

オンラインでnode.jsをサービスとしてするをむ https://riptutorial.com/ja/node-js/topic/9258/node-jsをサービスとしてする

38: Node.jsをむECMAScript 2015ES6

Examples

const / let

varとはなり、 const / letはスコープではなくレキシカルスコープにバインドされています。

```
{
  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
```

RunKitで

は、にのコードの 'this'レキシカルスコープにバインドされます。

```
performSomething(result => {
  this.someVariable = result
})
```

```
performSomething(function(result) {
  this.someVariable = result
}.bind(this))
```

の

3,5、および7の2をするこのをえてみましょう。

```
let nums = [3, 5, 7]
let squares = nums.map(function (n) {
  return n * n
})
console.log(squares)
```

RunKitで

.mapされたfunctionは、 functionキーワードをし、わりに=>することによって、としてすることもできます。

```
let nums = [3, 5, 7]
let squares = nums.map((n) => {
  return n * n
```

```
})
console.log(squares)
```

RunKitで

しかし、これはさらににくことができます。が1つのステートメントのみでされ、そのステートメントがりをするは、をりすとreturnキーワードをりくことができます。

```
let nums = [3, 5, 7]
let squares = nums.map(n => n * n)
console.log(squares)
```

RunKitで

```
    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);
```

フロー

```
/* @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クラス

```
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);
```

オンラインでNode.jsをむECMAScript 2015ES6をむ https://riptutorial.com/ja/node-js/topic/6732/node-jsをむecmascript-2015-es6-

39: nodeJsとのArduinoの

き

Node.JsがArduino Unoとどのようにできるかをす。

Examples

シリアルポートのArduinoとのノードJs

ノードjsコード

このトピックをするサンプルは、シリアルポートでArduinoとするNode.jsサーバーです。

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

サンプル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 + '!');
});
```

サンプルエクスプレスサーバーの

```
node app.js
```

Arduinoコード

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

1. あなたのマシンにarduinoをします。
2. サーバーをする

ノードjsエクスプレスサーバでのビルドをします。

ledをオンにするには

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

ledをオフにするには

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

オンラインでnodeJsとのArduinoのをむ https://riptutorial.com/ja/node-js/topic/10509/nodejsとのarduinoの

40: Nodejsの

き

ここでは、Node.jsの、バージョン、およびのステータスについてします。

Examples

なイベント

2009

- 33 このプロジェクトは "ノード"
- 101 Nodeパッケージのnpmののにのプレビュー
  マネージャー
- 118 Ryan DahlさんNode.jsのオリジナルNode.js JSConf 2009トーク

2010

- ExpressNode.js Webフレームワーク
- リリースのSocket.io
- 428 HerokuでのなNode.jsサポート
- 728 Ryan DahlのGoogle Tech Talk on Node.js
- 820 Node.js 0.2.0リリース

2011

- 331Node.jsガイド
- 51 npm 1.0リリース
- 51 Ryan DahlのAMAReddit
- 710 Node.jsのであるNode Beginner Bookがしました 。
  - のためのなNode.jsチュートリアル。
- 816 LinkedInはNode.jsをします
  - LinkedInは、としいをした、にされたモバイルアプリをした。
- 105 Ryan DahlはNode.jsのについてし、なぜがそれをしたのか
- 125 UberでのにおけるNode.js
  - Uber Engineering Manager Curtis Chambersは、Node.jsをしてをめ、パートナーとをさせるためにがアプリケーションをにしたをします。

2012

- 130 Node.jsクリエイター、Ryan DahlがNodeのかられます
- 625 Node.js v0.8.0 [stable]がしました
- 1220 Hapi、Node.jsフレームワークがリリースされました

2013

- 430 MEANスタックMongoDB、ExpressJS、AngularJS、Node.js
- 517 eBayののNode.jsアプリケーションの
- 1115 PayPalがNode.jsフレームワークであるKrakenをリリース
- 1122 WalmartでのNode.jsメモリリーク
    - ウォルマートのEran HammerはNode.jsのコアチームに、かしていたメモリリークをえました。
- 1219Koa - Node.jsのWebフレームワーク

2014

- 115 TJ FontaineがNodeプロジェクトをきぎます
- 1023 Node.js
    - JoyentとNode.jsコミュニティのいくつかのメンバーは、Node.jsオープンソースプロジェクトのオープンガバナンスモデルへののステップとして、Node.jsのをしました。
- 1119 Flame GraphsのNode.js - Netflix
- 1128 IO.js - V8 JavascriptのI / Oのイベント

2015

Q1

- 114 IO.js 1.0.0
- 210 JoyentがNode.jsをするためにく
    - Joyent、IBM、Microsoft、PayPal、Fidelity、SAP、The Linux Foundationが、NeutralとOpen GovernanceをいてNode.jsコミュニティをサポート
- 227 IO.jsとNode.jsのリコンシリエーション

Q2

- 414 npmプライベートモジュール
- 528 ノードリードTJフォンテーヌがし、ジョイエントをれる

- 513 Node.jsとio.jsがNode Foundationでマージされています

Q3

- 82 Trace - Node.jsのパフォーマンスとデバッグ
    - トレースは、マイクロサービスをするになすべてのメトリックをするされたマイクロサービスツールです。
- 813 4.0がしい1.0です

Q4

- 1012 ノードv4.2.0、のサポートリリース
- 128 Apigee、RisingStack、YahooがNode.js Foundationに
- 128と9 ノードインタラクティブ
    - Node.js FoundationによるめてのNode.jsカンファレンス

2016

Q1

- 210 Expressはインキュベートされたプロジェクトになります
- 323 パッド
- 329 Google Cloud PlatformがNode.js Foundationに

Q2

- 426 npmのユーザーは210,000

Q3

- 718 CJ SilverioがnpmのCTOに
- 81 トレース、Node.jsのデバッグソリューションがにになる
- 915 ヨーロッパでのノードインタラクティブ

Q4

- 1011 パッケージマネージャーがリリースされました
- 1018 Node.js 6がLTSバージョンになりました

1. "タイムラインのNode.jsの" [オンライン][https://blog.risingstack.com/history-of-node-js]

オンラインでNodejsのをむ https://riptutorial.com/ja/node-js/topic/8653/nodejsの

41: NodeJSフレームワーク

Examples

Webサーバーフレームワーク

エクスプレス

```
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!');
});
```

コア

```
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);
```

コマンドラインインターフェイスフレームワーク

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();
```

オンラインでNodeJSフレームワークをむ https://riptutorial.com/ja/node-js/topic/6042/nodejsフレームワーク

42: NodeJsルーティング

き

ノードjsおよびExpressルーターののでなExpress Webサーバーをセットアップする。

に、Express Routerをすると、アプリケーションのルーティングをすることができ、がです。

Examples

Express Web Serverのルーティング

Express Webサーバーの

Expressサーバーがになり、くのユーザーとコミュニティがくりげています。がまっています。

Express Serverをできます。パッケージとのについてNPMNode Package Managerをします。

1. Projectディレクトリにし、package.jsonファイルをします。 ***package.json*** { "" "expressRouter"、 "バージョン" "0.0.1"、 "スクリプト"{ "" "ノードServer.js"}、 ""{ "" 「^ 4.12.3 "}}
2. ファイルをし、のコマンド*npm install*をしてなをインストールします。これにより、プロジェクトディレクトリにnode_modulesがされ、ながされます。
3. Express Web Serverをしましょう。 Projectディレクトリにし、server.jsファイルをします。 ***server.js***

   var express = require "express"; var app = express;

// Routerオブジェクトをする

varルータ= express.Router;

//ここにすべてのルートをします。これはホームページです。

```
router.get("/",function(req,res){
res.json({"message" : "Hello World"});
```

};

app.use "/ api"、router;

//このポートをく

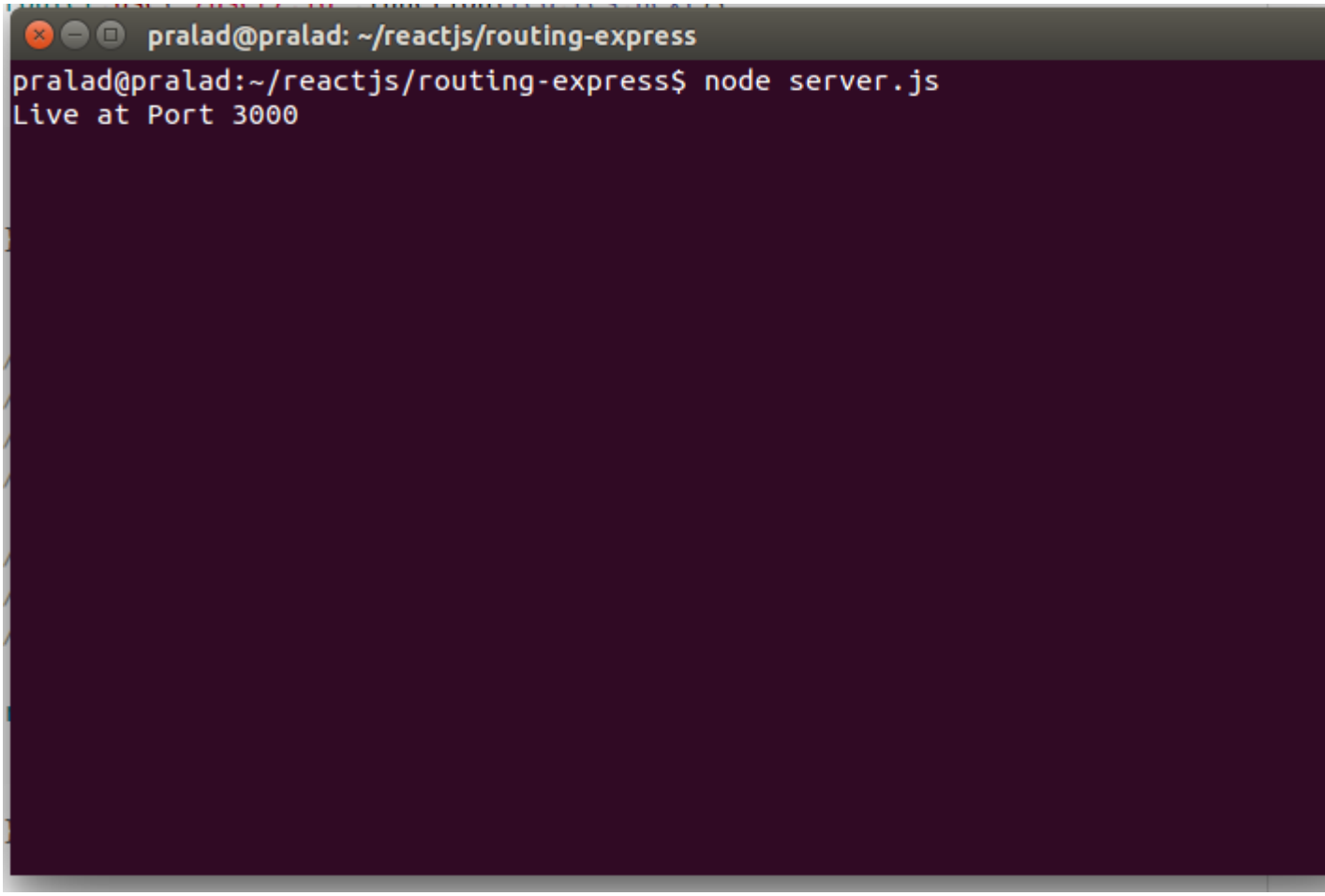
app.listen3000、function{console.log "ポート3000でライブ";};

```
For more detail on setting node server you can see [here][1].
```

4. のコマンドをして、サーバーをします。

   ノードserver.js

   サーバーがにされると、のようなメッセージがされます。



   。

5. ブラウザやにき、リクエストをしました

   http// localhost3000 / api /

   はのようになります。

。

つまり、Expressルーティングのです。

さあ、GET、POSTなどをいましょう。

あなたのようなserver.jsファイルをする

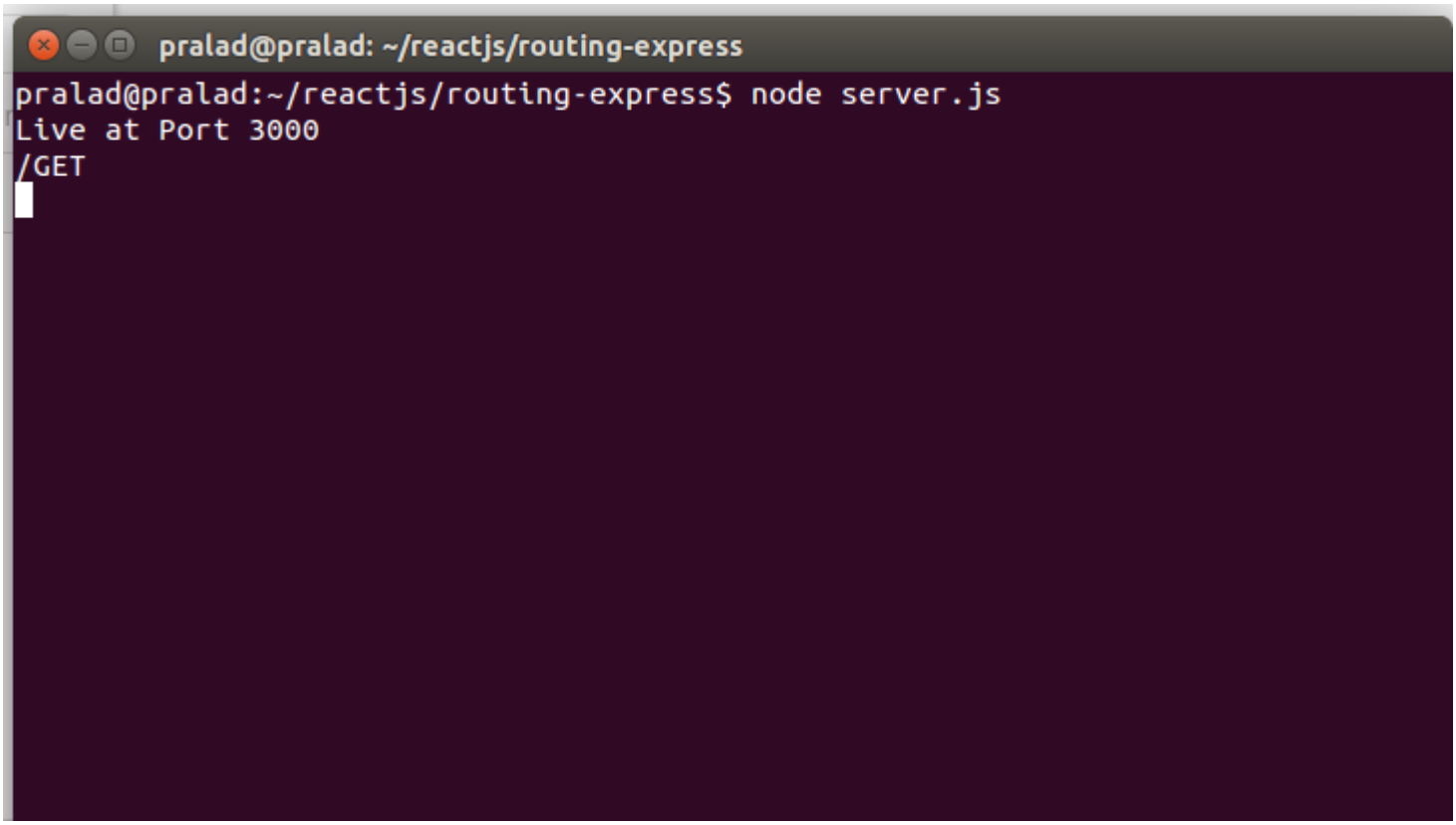
```
var express = require("express");
var app = express();

//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");
```

```
});
```

すぐサーバーをし、

```
http://localhost:3000/api/
```

あなたはかのようにえます

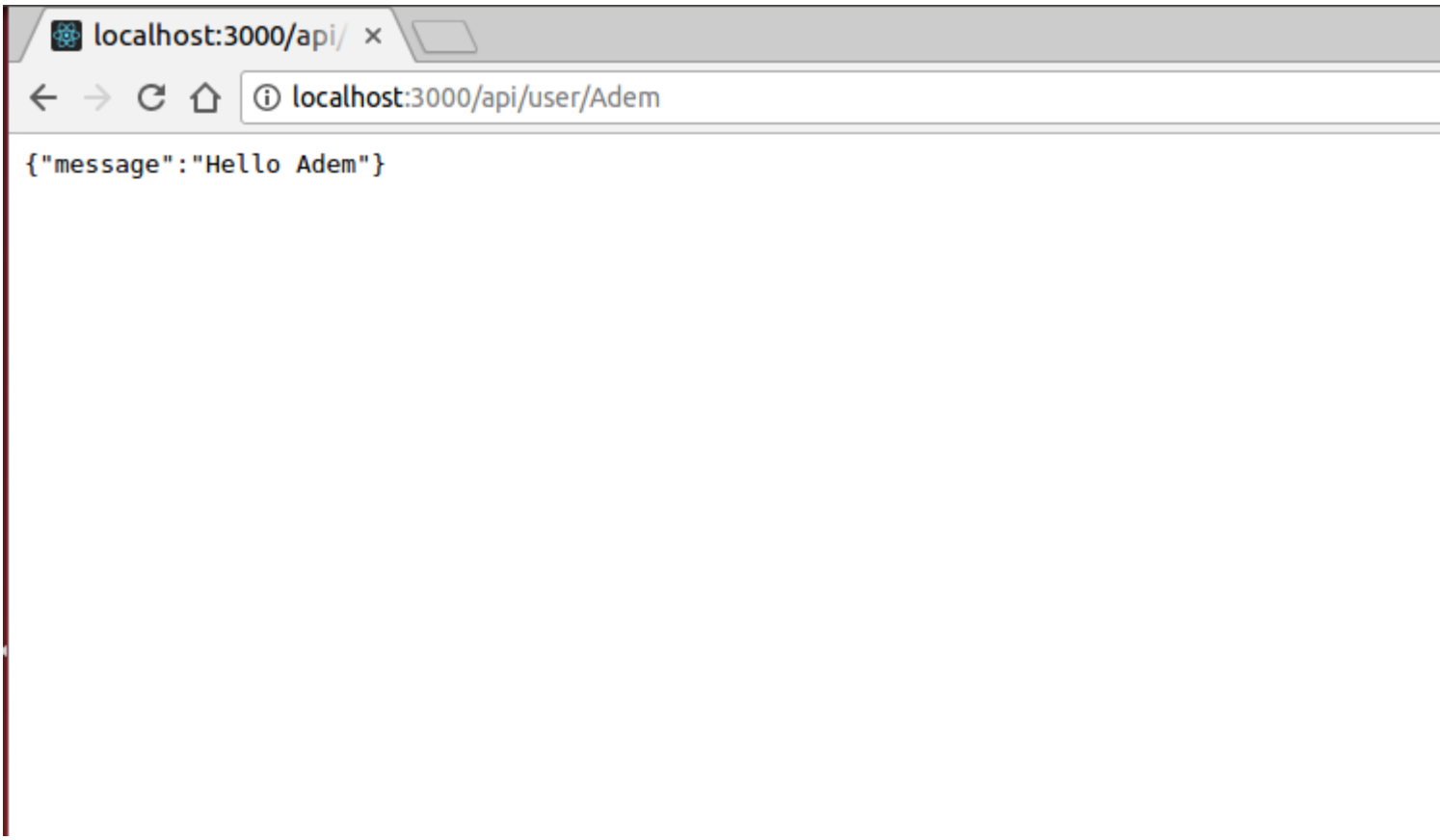


ルーティングでのパラメータへのアクセス

*http://example.com/api/:name/*のようにurlからパラメータにアクセスすることもできます。したがって、nameパラメータにアクセスできます。 server.jsにのコードをします

```
router.get("/user/:id",function(req,res){
  res.json({"message" : "Hello "+req.params.id});
});
```

すぐサーバーをし、[http// localhost3000 / api / user / Adem] [4]にくと、はのようになります



。

オンラインでNodeJsルーティングをむ https://riptutorial.com/ja/node-js/topic/9846/nodejsルーティング

43: NodeJSガイド

Examples

こんにちは

`helloworld.js`というファイルにのコードをします

```
console.log("Hello World");
```

ファイルをし、Node.jsでします。

```
node helloworld.js
```

オンラインでNodeJSガイドをむ https://riptutorial.com/ja/node-js/topic/7693/nodejsガイド

44: npm

き

Node Package Managernpmは、の2つのをします。search.nodejs.orgでなnode.jsパッケージ/モジュールのオンラインリポジトリ。 Node.jsパッケージをインストールするコマンドラインユーティリティは、Node.jsパッケージのバージョンとをいます。

- npm <command>ここで、<command>はのいずれかです。
  - ユーザーをする
  - ユーザーをする
  - apihelp
  -
  - ビン
  - バグ
  - c
  - キャッシュ
  -
  -
  - ddp
  -
  - する
  - ドキュメント
  -
  - する
  - よくある
  - つける
  - ファインダーをつける
  - する
  - けて
  - ヘルプ
  -
  -
  - インストール
  -
  - そのに
  - isntall
  -
  - ラ
  - リンク
  - リスト
  - ll
  - ln
  - ログイン

- ls
- れの
- オーナー
- パック
- プレフィックス
- プルーン
- する
- r
- rb
- する
- する
- レポ
-
- rm
- ルート
- スクリプト
- s
- セ
- サーチ
- セット
- ショー
-
-
-
-
- やめる
- サブモジュール
- タグ
- テスト
- TST
-
- アンインストール
- リンクをする
-
- アンスター
- アップ
-
- v
- バージョン
- る
- は

パラメーター

パラメータ	
アクセス	`npm publish --access=public`
ビン	`npm bin -g`
	`npm edit connect`
けて	`npm help init`
そのに	`npm init`
インストール	`npm install`
リンク	`npm link`
プルーン	`npm prune`
する	`npm publish ./`
	`npm restart`
	`npm start`
やめる	`npm start`
	`npm update`
バージョン	`npm version`

Examples

パッケージのインストール

き

パッケージは、がプロジェクトにできるツールをするためにnpmによってされるです。これには、jQueryやAngularJSなどのライブラリやフレームワークからGulp.jsなどのタスクランナーまでのすべてがまれます。パッケージははnode_modulesというフォルダにっています。このフォルダにはpackage.jsonファイルもまれています。このファイルには、のパッケージをするためになモジュールであるをむすべてのパッケージにするがまれています。

Npmはコマンドラインをしてパッケージのインストールとをうため、npmをしようとするユーザーはオペレーティングシステムのなコマンド、つまりディレクトリのとディレクトリのをることができます。

NPMのインストール

パッケージをインストールするには、NPMがインストールされているがあります。

NPMをインストールするには、 Node.jsダウンロードページのインストーラをすることをおめします。 `npm -v`または`npm version`コマンドをして、node.jsがすでにインストールされているかどうかをすることができます。

Node.jsインストーラをしてNPMをインストールしたは、ずアップデートをしてください。これは、NPMがNode.jsインストーラよりにされるためです。をするには、のコマンドをします。

```
npm install npm@latest -g
```

パッケージのインストール

1つまたはのパッケージをインストールするには、をします。

```
npm install <package-name>
# or
npm i <package-name>...

# e.g. to install lodash and express
npm install lodash express
```

> これにより、コマンドラインがするディレクトリにパッケージがインストールされるため、なディレクトリがされているかどうかをすることがです

のディレクトリに`package.json`ファイルがあり、がされている、 `npm install`はにファイルにリストされているすべてのをしてインストールします。また、 `npm install`コマンドのをすることもできます`npm i`

のバージョンのパッケージをインストールするは、のようにします。

```
npm install <name>@<version>

# e.g. to install version 4.11.1 of the package lodash
npm install lodash@4.11.1
```

のバージョンにするバージョンをインストールするは、のようにします。

```
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"
```

バージョンをインストールするには、のコマンドをします。

```
npm install <name>@latest
```

のコマンドは、 npmjs.comのの`npm`リポジトリにあるパッケージをします。 `npm`レジストリからインストールする`npm`がないは、のようなオプションがサポートされています

```
# 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>
```

、モジュールはのディレクトリにある`node_modules`というのフォルダにローカルにインストールされます。 `require()`がモジュールをできるようにみむためにするディレクトリです。

`package.json`ファイルをすでにしているは、 `--save` `-S` オプションまたはその1つをして、インストールされたパッケージをとして`package.json`ににすることができます。のがあなたのパッケージをインストールすると、 `npm`はに`package.json`ファイルからをみみ、リストされたバージョンをインストールします。でファイルをしてをしてすることもできますので、はをすることをおめします。たとえば、のようにします。

```
npm install --save <name> # Install dependencies
# or
npm install -S <name> # shortcut version --save
# or
npm i -S <name>
```

パッケージをインストールして、アプリケーションのになではなく、アプリケーションをするためではなく、になにのみします。のコマンドをします。

```
npm install --save-dev <name> # Install dependencies for development purposes
# or
npm install -D <name> # shortcut version --save-dev
# or
npm i -D <name>
```

のインストール

モジュールによっては、するライブラリをするだけでなく、コマンドラインでするための1つのバイナリもしています。それらのパッケージをローカルにインストールすることはできますが、コマンドラインツールをにできるように、これらのパッケージをグローバルにインストールすることをおめします。その、 `npm`はバイナリをなパスえば`/usr/local/bin/<name>` ににリンクしてコマンドラインからできるようにします。パッケージをグローバルにインストールするには、をします。

```
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
```

インストールされているすべてのパッケージとするバージョンのリストをのワークスペースにするには、のようにします。

```
npm list
npm list <name>
```

オプションのnameをすると、のパッケージのバージョンをできます。

---

npmモジュールをグローバルにインストールしようとしているときにのに`sudo npm install -g ...` したは、をするために`sudo npm install -g ...`を`sudo npm install -g ...`にしてください。されたでシステムでサードパーティのスクリプトをすることはです。のは、 `npm`のインストールにがあるがあります。サンドボックスされたユーザーにNodeをインストールするは、 nvmをしてみてください 。

ビルドツールやののGruntなどがあるは、それらをデプロイするアプリケーションにバンドルするはありません。その、 `devDependencies`の`package.json`リストされているにしたいとうでしょう。パッケージをのとしてインストールするには、 `--save-dev` または`-D` をします。

```
npm install --save-dev <name> // Install development dependencies which is not included in
production
# or
npm install -D <name>
```

そのパッケージがあなたの`package.json devDependencies`にされていることが`devDependencies`ます。

ダウンロード/されたnode.jsプロジェクトのをインストールするには、

```
npm install
# or
npm i
```

npmはに`package.json`からをみんでインストールします。

プロキシサーバーのにあるNPM

インターネットへのアクセスがプロキシサーバーのは、リモートリポジトリにアクセスするnpm installコマンドをするがあります。 npmは、コマンドラインでできるファイルをします

```
npm config set
```

ブラウザのパネルからプロキシをつけることができます。プロキシサーバーのURL、ポート、ユーザーとパスワードをしたら、のように、npmをするがあります。

```
$ 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`フィールドはオプションです。これらをすると、 `npm install`、 `npm i -g`などがにします。

スコープとリポジトリ

```
# 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
```

のパッケージのが`@myscope`でまり、スコープ "myscope"がのリポジトリにけられている、 `npm publish`はパッケージをそのリポジトリにアップロードします。

これらのを`.npmrc`ファイルにすることもできます。

```
@myscope:registry=http://registry.corporation.com
//registry.corporation.com/:_authToken=xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxx
```

これは、CIサーバーfeでビルドをするにです。

パッケージのアンインストール

1つまたはのローカルにインストールされたパッケージをアンインストールするには、

```
npm uninstall <package name>
```

npmのアンインストールコマンドには、の5つのエイリアスもできます。

```
npm remove <package name>
npm rm <package name>
npm r <package name>

npm unlink <package name>
npm un <package name>
```

アンインストールのとしてpackage.jsonファイルからパッケージをするは、 - --saveフラグ -S をします。

```
npm uninstall --save <package name>
npm uninstall -S <package name>
```

のの、-- --save-devフラグをします-D 。

```
npm uninstall --save-dev <package name>
npm uninstall -D <package name>
```

オプションののは、 - --save-optionalフラグをし--save-optional -O 。

```
npm uninstall --save-optional <package name>
npm uninstall -O <package name>
```

グローバルにインストールされたパッケージのは、 --globalグローバルフラグをし--global -g 。

```
npm uninstall -g <package name>
```

なセマンティックバージョン

パッケージをするに、パッケージをバージョンアップするがあります。 npmはセマンティックバージョニングをサポートしています。つまり、 パッチ、マイナー、メジャーリリースがあります。

たとえば、バージョンをするためにパッケージがバージョン1.2.3にあるは、のがです。

1. パッチリリース npm version patch => 1.2.4
2. マイナーリリース npm version minor => 1.3.0
3. メジャーリリース npm version major => 2.0.0

バージョンをすることもできます

`npm version 3.1.4` => 3.1.4

のnpmコマンドの1つをしてパッケージバージョンをすると、npmはpackage.jsonファイルのバージョンフィールドをしてコミットし、 "v"というをつしいGitタグをします。コマンドをしました

```
git tag v3.1.4
```

Bowerのようなのパッケージマネージャとはなり、npmレジストリはすべてのバージョンでされるGitタグにしません。しかし、タグをうのがきなは、パッケージバージョンをバンプしてからしくしたタグをしてください。

`git push origin master` をpackage.jsonにプッシュするため

`git push origin v3.1.4` しいタグをプッシュするため

または、のでこれをうことができます。

```
git push origin master --tags
```

パッケージの

Node.jsパッケージのは、プロジェクトのルートにある`package.json`というファイルにまれています。しいファイルをするには、のをびします。

```
npm init
```

これはGitリポジトリするののディレクトリとをみみ、あなたのプレースホルダのをしようとします。それのは、オプションのダイアログがされます。

デフォルトをして`package.json`をするには、のようにします。

```
npm init --yes
# or
npm init -y
```

npmパッケージとしてしないプロジェクトつまりをりげるのみに`package.json`をしているは、`package.json`ファイルにそのをえることができます

1. にじて、 `private`プロパティをtrueにして、ったパブリッシュをします。
2. にじて、 `license`プロパティを「」にして、のがパッケージをするをします。

パッケージをインストールして`package.json`にするには、のようにします。

```
npm install --save <package>
```

パッケージとするメタデータパッケージのバージョンなどがにされます。として `--save-dev`をし

てすると、そのパッケージはわりに`devDependencies`にされます。

このベアボーン`package.json`では、パッケージのインストールまたはアップグレードにメッセージがされ、とリポジトリフィールドがしていることがされます。これらのメッセージをすることはですが、のテキストエディタでpackage.jsonをき、のをJSONオブジェクトにすることで、メッセージをりくことができます。

```
[...]
"description": "No description",
"repository": {
  "private": true
},
[...]
```

パッケージをする

まず、パッケージをしたことをします パッケージのを。に、npmjsにログインするがあります。

すでにnpmユーザーがいる

```
npm login
```

ユーザーがいない

```
npm adduser
```

ユーザーがのクライアントにされていることをするには

```
npm config ls
```

その、パッケージをするができたら、

```
npm publish
```

そして、あなたはです。

しいバージョンをするがあるは、 なのバージョニングにされているように、パッケージのバージョンをするようにしてください。そうしないと、 `npm`はパッケージをすることはできません。

```
{
    name: "package-name",
    version: "1.0.4"
}
```

スクリプトの

`package.json`にスクリプトをすることができます。たとえば、のようになります。

```
{
  "name": "your-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "author": "",
  "license": "ISC",
  "dependencies": {},
  "devDependencies": {},
  "scripts": {
    "echo": "echo hello!"
  }
}
```

`echo`スクリプトをするには、コマンドラインから`npm run echo`を`npm run echo`ます。のようなのスクリプト、 `echo`は、でされなければなら`npm run <script name>`。 npmには、パッケージのライフサイクルのの `preinstall` でされるくのスクリプトもあります。 npmがスクリプトフィールドをどのようにうかのについては、 ここをしてください。

npmスクリプトは、サーバの、プロジェクトの、テストのなどのでもにされます。もっとながあります

```
  "scripts": {
    "test": "mocha tests",
    "start": "pm2 start index.js"
  }
```

`scripts`エントリでは、 `mocha`ようなコマンドラインプログラムは、グローバルにまたはローカルにインストールされたときにします。コマンドラインエントリがシステムPATHにしない、npmはローカルにインストールされたパッケージもチェックします。

スクリプトがにくなると、のようににすることができます。

```
  "scripts": {
    "very-complex-command": "npm run chain-1 && npm run chain-2",
    "chain-1": "webpack",
    "chain-2": "node app.js"
  }
```

なパッケージの

なパッケージリストにはインストールされていないパッケージをするには、のコマンドをします。

```
npm prune
```

すべての`dev`パッケージをするには、-- `--production`フラグをします。

```
npm prune --production
```

もっとしく

インストールされているパッケージの

インストールされているパッケージのリストツリービューをするには、

```
npm list
```

ls 、 **la** 、 **ll**は**list**コマンドのエイリアスです。 laとllコマンドは、とリポジトリのようなをします。

オプション

は、オプションをすことでできます。

```
npm list --json
```

- **json** - をjsonでします。
- **long** - をします。
- **parseable** - ツリーのわりになリストをする
- **global** - グローバルにインストールされたパッケージをします。
- - ツリーの
- **dev** / **development** - devDependenciesをします。
- **prod** / **production** - をします。

にじて、パッケージのホームページにすることもできます。

```
npm home <package name>
```

npmとパッケージの

npmはNode.jsモジュールなので、それをしてすることができます。

*OS*が*Windows*の、としてコマンドプロンプトをするがあります

```
npm install -g npm@latest
```

されたバージョンをするには、のをいます。

```
npm outdated
```

のパッケージをするには

```
npm update <package name>
```

これにより、package.jsonのにってパッケージがのバージョンにされます

package.jsonでされたバージョンをロックしたい

```
npm update <package name> --save
```

モジュールをのバージョンにロックする

デフォルトでは、npmはのセマンティックバージョンにってモジュールののなバージョンをインストールします。これは、モジュールがセマをしておらず、えばモジュールのをしたににになるがあります。

のバージョンおよびそののバージョンなどを`node_modules`フォルダにローカルにインストールされたのバージョンにロックするには、

```
npm shrinkwrap
```

これにより、ののバージョンをする`package.json`に`npm-shrinkwrap.json`されます。

グローバルにインストールされたパッケージのセットアップ

`npm install -g`をして、「グローバルに」パッケージをインストール`npm install -g`ことができます。これは、するパスにできるファイルをインストールするためにされます。えば

```
npm install -g gulp-cli
```

パスをすると、 `gulp`びすことができます。

くのOSでは、 `npm install -g`は、ユーザが`/usr/bin`などにきめないディレクトリにきもうとします。あなたはしないでください`sudo npm install`ってのスクリプトをしているのなセキュリティのリスクがあるので、このに`sudo`とrootユーザーは、あなたがのインストールがよりにこれにきむことはできませんあなたののディレクトリをすることがあります。

`npm`は、ファイル`~/.npmrc`をってグローバルモジュールをどこにインストールするかをできます。これは`prefix`とばれ、 `npm prefix`できます。

```
prefix=~/.npm-global-modules
```

これは、 `npm install -g`を`npm install -g`たびにをします。 `npm install --prefix ~/.npm-global-modules`をして、インストールにをすることもできます。がとじは、 `-g`をするはありません。

グローバルにインストールされたモジュールをするには、あなたのパスにあるがあります

```
export PATH=$PATH:~/.npm-global-modules/bin
```

`npm install -g gulp-cli`すると、 `gulp`をできるようになります。

`npm install -g`なし、は`package.json`のディレクトリ、またはにつからないはカレントディレクトリになります。また、ファイルをつ`node_modules/.bin`ディレクトリもされます。プロジェクトにのファイルをするは、 `npm install -g`をするはありません。 `node_modules/.bin`あるものをできます。

よりいデバッグとのためのプロジェクトのリンク

プロジェクトのをすることははにはなです。 NPMにパッケージ･バージョンをし、をインストールしてをテストするわりに、 `npm link`して`npm link`。 `npm link`はシンボリック`npm link`し、のコードをローカルでテストすることができます。これにより、されたバージョンをするにのコードをできるようにすることで、グローバルツールとプロジェクトのをにテストできます。

ヘルプテキスト

```
NAME
       npm-link - Symlink a package folder

SYNOPSIS
         npm link (in package dir)
         npm link [<@scope>/]<pkg>[@<version>]

         alias: npm ln
```

プロジェクトのをリンクする

リンクをするときは、パッケージがプロジェクトでされるものであることにしてください。

1. CDをディレクトリ `cd ../my-dep`
2. `npm link`
3. をするプロジェクトにCDをする
4. `npm link my-dep`またはnamespaced `npm link @namespace/my-dep`

グローバルツールをリンクする

1. CDをプロジェクトディレクトリに`cd eslint-watch cd eslint-watch`
2. `npm link`
3. ツールをする
4. `esw --quiet`

するのある

ツールまたはグローバルツールがにインストールされている、プロジェクトをリンクするとがすることがあります。 `npm uninstall (-g) <pkg>`をしてから`npm link`すると、するがされます。

オンラインでnpmをむ https://riptutorial.com/ja/node-js/topic/482/npm

45: nvm - ノードバージョンマネージャ

のでされているURLは、ノード·バージョン·マネージャののバージョンをしています。バージョンは、されているものとはなるがもいです。バージョンをしてnvmをインストールするには、 ここをクリックしてのURLをするGitHubのnvmにアクセスしてください。

Examples

NVMのインストール

`curl`をすることができます

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

あるいは、 `wget`をうこともできます

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

NVMのバージョンをする

nvmがインストールされていることをするには、のようにします。

```
command -v nvm
```

インストールがしたは 'nvm'をするはずです。

のノードバージョンのインストール

インストールなリモートバージョンの

```
nvm ls-remote
```

リモートバージョンのインストール

```
nvm install <version>
```

えば

```
nvm install 0.10.13
```

すでにインストールされているノードバージョンをする

NVMをしてなノードのローカルバージョンをするには

```
nvm ls
```

たとえば、 `nvm ls`がのようにすとします。

```
$ nvm ls
     v4.3.0
     v5.5.0
```

`v5.5.0`りえるには、 `v5.5.0`します。

```
nvm use v5.5.0
```

Mac OSXにnvmをインストールする

インストールプロセス

git、curl、またはwgetをしてNode Version Managerをインストールできます。これらのコマンドは、 **Mac OSX**の ターミナルでします。

カール

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

wgetの

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.31.3/install.sh | bash
```

NVMがしくインストールされているかどうかをテストする

nvmがしくインストールされたことをテストするには、Terminalをじてオープンし、 `nvm`とし`nvm`。 **nvmcommand not found**メッセージがされた、OSにな**.bash_profile**ファイルがないがあります。ターミナルで`touch ~/.bash_profile`をし、のインストールスクリプトをします。

それでも**nvmcommand**がつからないは、のコマンドをしてください。

- 「ターミナル」に「 `nano .bashrc`とし`nano .bashrc`。 のものとほぼじエクスポートスクリプトがされます。

  NVM_DIR = "/ Users / johndoe / .nvm" [-s "$ NVM_DIR / nvm.sh"] &&をエクスポートします。 "$ NVM_DIR / nvm.sh"

- エクスポートスクリプトをコピーし、 **.bashrc**からします。
- .bashrcファイルをしてじるCTRL + O - Enter - CTRL + X
- に、 `nano .bash_profile`として、Bashプロファイルをきます。
- コピーしたエクスポートスクリプトをBashプロファイルにしいにりける
-

してバッシュプロファイルをじるCTRL + O - Enter - CTRL + X
- に`nano .bashrc`として、 **.bashrc**ファイルをきます。
- のをファイルにりけます。

  ソース/ .nvm / nvm.sh

- してじるCTRL + O - Enter - CTRL + X
- ターミナルをし、 `nvm`として`nvm`しているかどうかテストします

ノードバージョンのをする

インストールされているノードのバージョンにいくつかのエイリアスをするは、のようにします。

```
nvm alias <name> <version>
```

unaliasとじように、のようにします。

```
nvm unalias <name>
```

デフォルトのエイリアスとしてのバージョンをしたいは、ながあります。 `default`エイリアスバージョンは`default`でコンソールにロードされます。

のように

```
nvm alias default 5.0.1
```

コンソール/がするたびに、デフォルトで5.0.1がします。

```
nvm alias # lists all aliases created on nvm
```

のコマンドをサブシェルでしたいノードのバージョン

インストールされているすべてのノードのバージョンをする

```
nvm ls
    v4.5.0
    v6.7.0
```

のノードがインストールされたバージョンをしてコマンドを

```
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
```

エイリアスをする

```
nvm run default --version or nvm exec default node --version
Running node v6.7.0 (npm v3.10.3)
v6.7.0
```

ノードLTSバージョンをインストールするには

```
nvm install --lts
```

バージョンのりえ

```
nvm use v4.5.0 or nvm use stable ( alias )
```

オンラインでnvm - ノードバージョンマネージャをむ https://riptutorial.com/ja/node-js/topic/2823/nvm----ノードバージョンマネージャ

46: OAuth 2.0

Examples

RedisによるOAuth 2 - grant_typepassword

このでは、りのapiでredisデータベースでoauth2をします

> あなたのマシンにredisデータベースをインストールするがあります。 ここからLinuxユーザーにダウンロードし、Windowsをインストールするにはここからダウンロードしてください 。 ここからredis managerデスクトップアプリケーションをします 。

ここで、redisデータベースをするようにnode.jsサーバーをするがあります。

- サーバーファイルの**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);
```

- ルート/ **Oauth2 / model.js**に**Oauth2**モデルをする

```
 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);
};
```

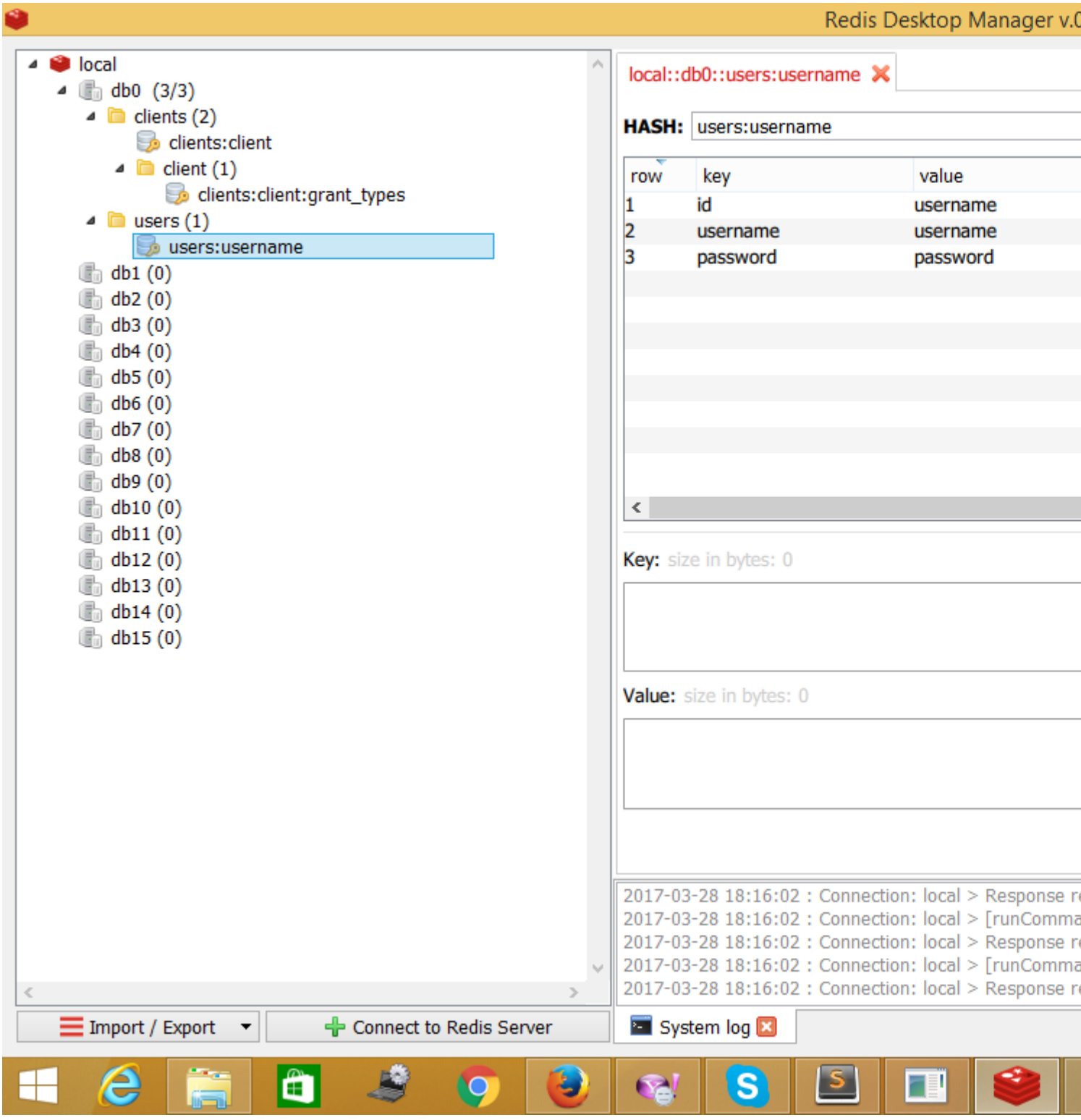
```
model.saveAccessToken = function (accessToken, clientId, expires, user, callback) {
  db.hmset(util.format(keys.token, accessToken), {
    accessToken: accessToken,
    clientId: clientId,
    expires: expires ? expires.toISOString() : null,
    userId: user.id
  }, callback);
};

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
    });
  });
};
```

マシンにredisをインストールし、のノードファイルをするだけです

```
  #! /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();
  });
```

このファイルはフロントエンドがトークンをするためのをします

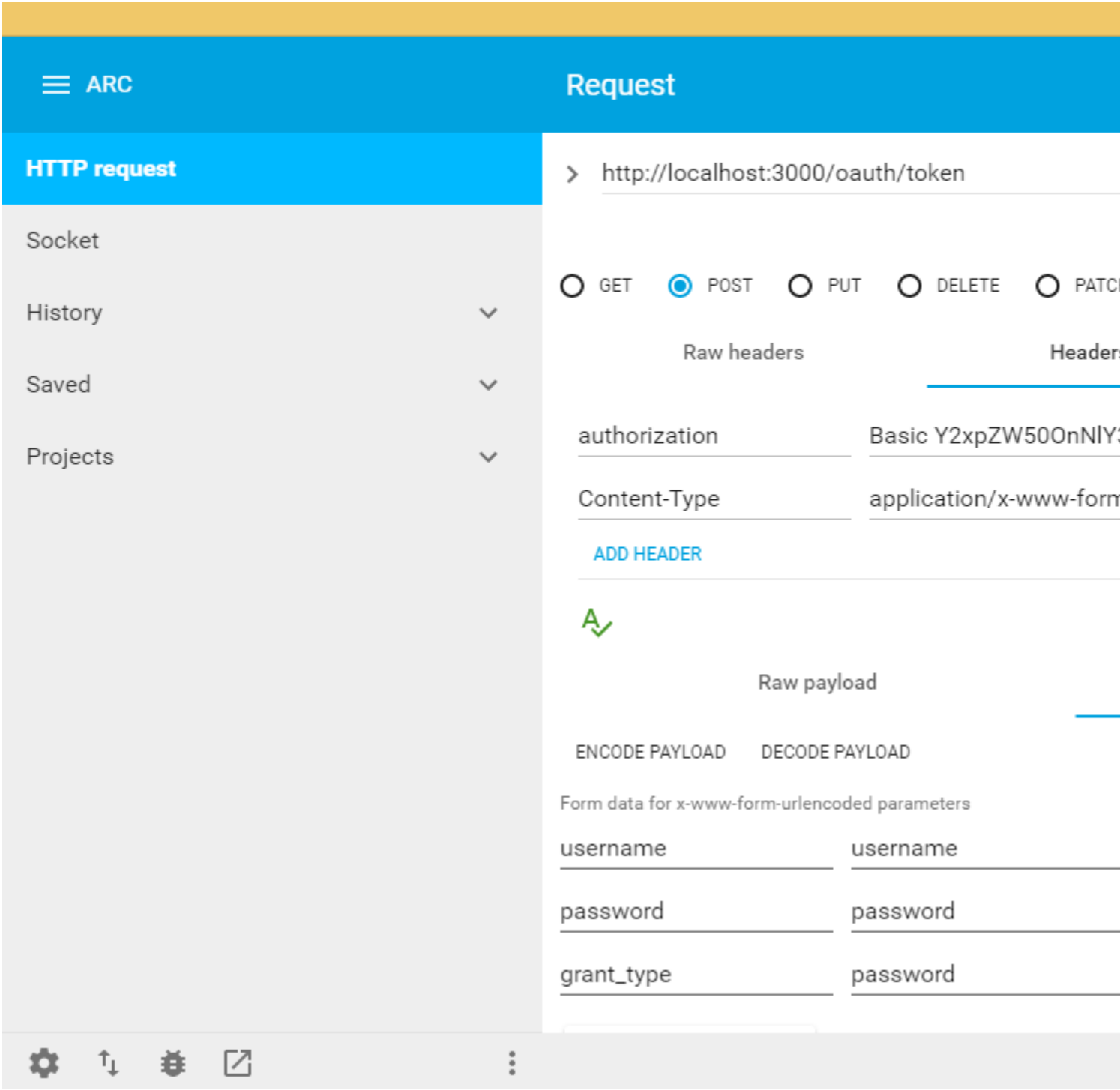
のファイルをびしたのサンプルredisデータベース



リクエストはのようになります

---

apiへのサンプルびし



ヘッダ

1. Redisをにセットアップしたときにされたパスワードのにパスワードがされます。

    a。 clientId + secretIdからbase64へ

2. データ

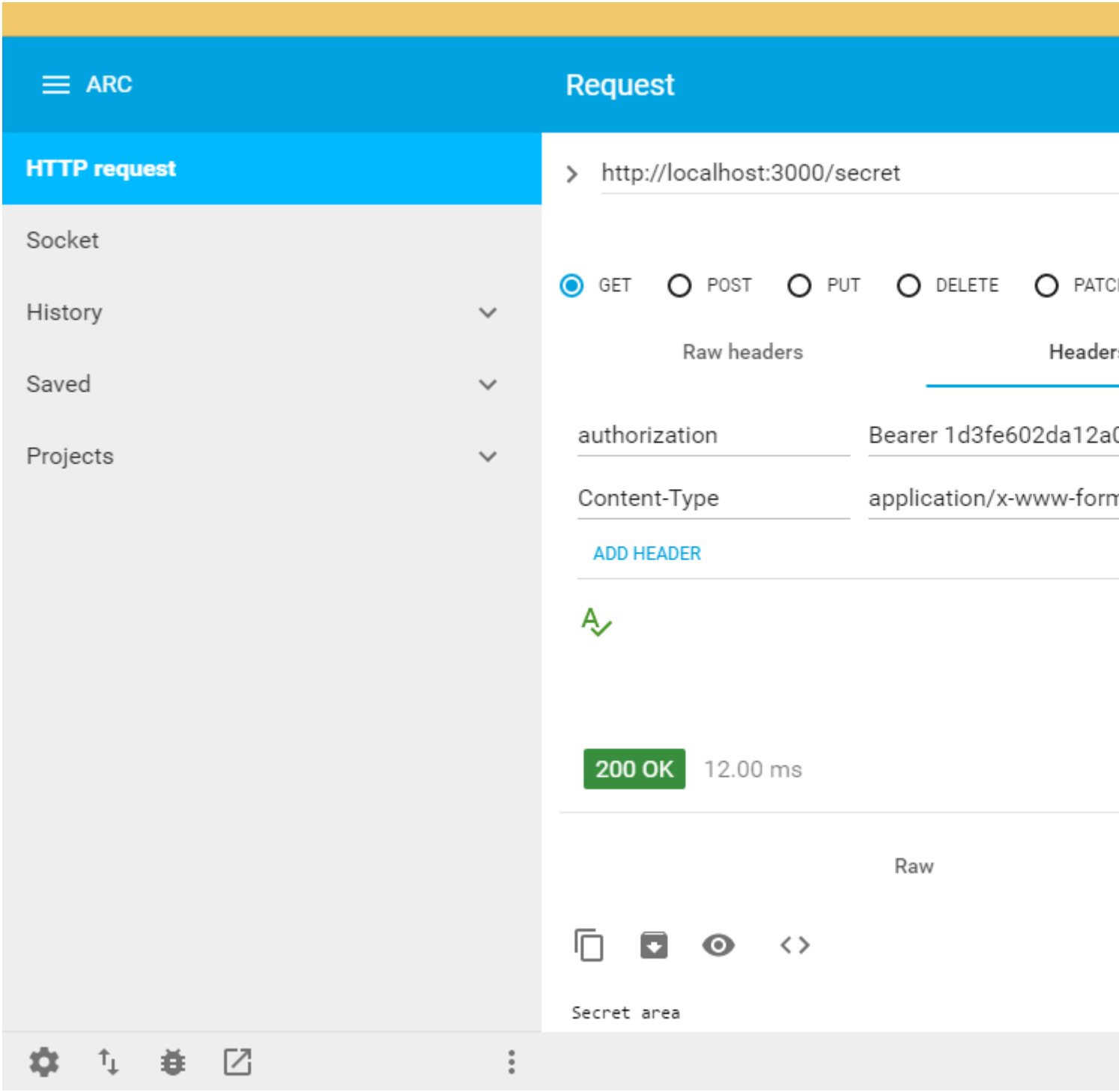
    ユーザーそのトークンをするユーザー

    パスワードユーザーパスワード

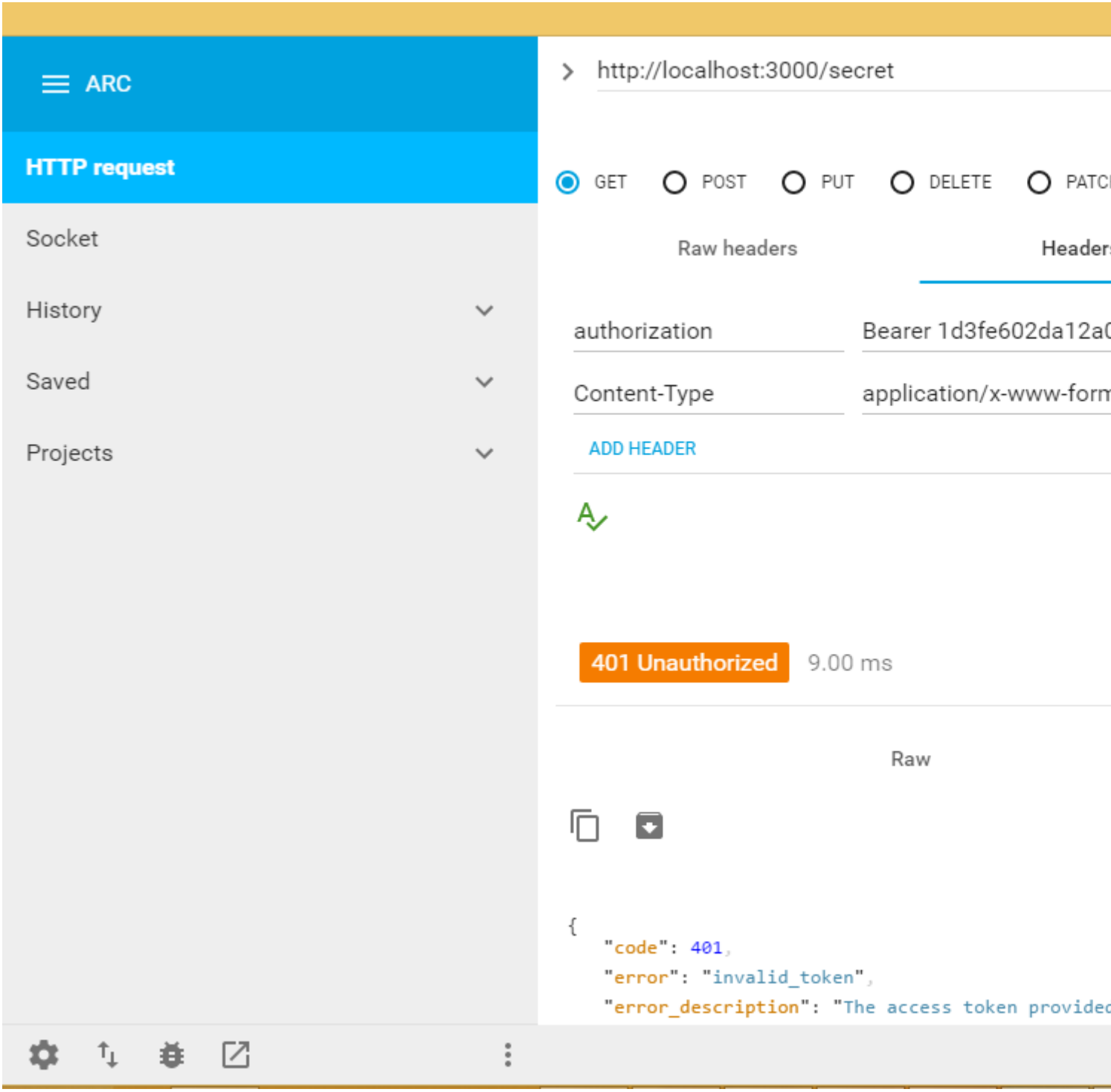
grant_typeどのオプションがなのかによってなりますが、はパスワードとパスワードだけをってredisでするpasswodをします。redisのデータはのようになります

```
{
  "access_token":"1d3fe602da12a086ecb2b996fd7b7ae874120c4f",
  "token_type":"bearer", // Will be used to access api + access+token e.g. bearer
1d3fe602da12a086ecb2b996fd7b7ae874120c4f
  "expires_in":3600,
  "refresh_token":"b6ad56e5c9aba63c85d7e21b1514680bbf711450"
}
```

したがって、たちはapiをびして、たちがしたアクセストークンでされたデータをするがあります。をしてください。

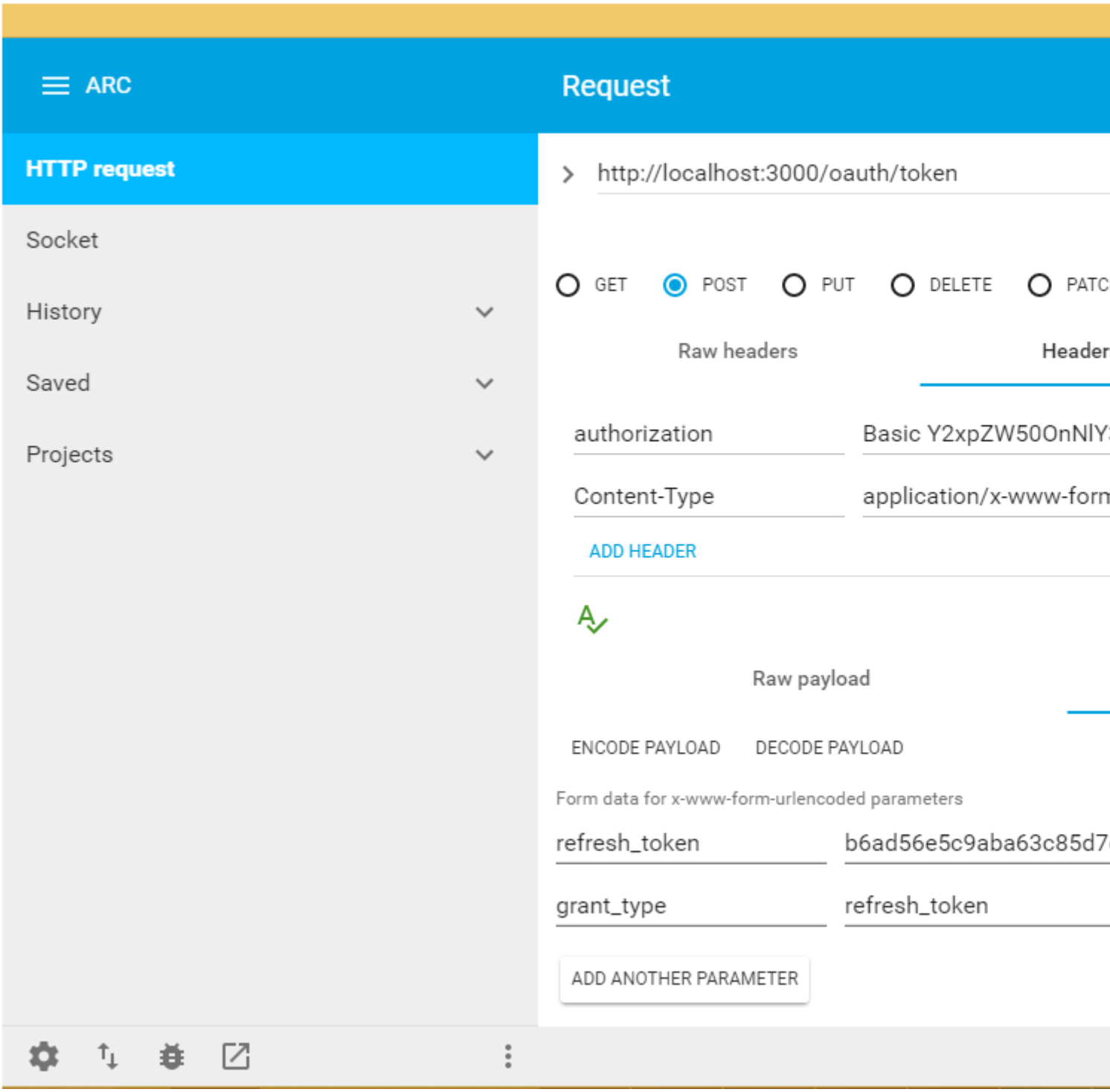


トークンがれになると、apiはトークンのがれていて、apiびしのいずれにもアクセスできないというエラーをげます。のをしてください



アクセストークンがれになった、トークンがれになったらどうするかをてみましょう。れのaccess_tokenをするredisにrefresh_tokenがするから、refresh_token grant_typeでoauth / tokenをもうびすがあります。 Basic clientIdclientsecretbase 64へのをい、にrefresh_tokenをすると、しいデータをつしいaccess_tokenがされます。

のは、しいアクセストークンをするをしています。



をけて

オンラインでOAuth 2.0をむ https://riptutorial.com/ja/node-js/topic/9566/oauth-2-0

47: OracleとNode.js

Examples

Oracle DBにする

ORACLEデータベースにするなは、 `oracledb`モジュールをする`oracledb`です。このモジュールは、Node.jsアプリケーションとOracleサーバーのをします。のモジュールとにインストールできます

```
npm install oracledb
```

これで、ORACLEをするがあります。これをでできます。

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

oracledb.getConnection(
  {
    user          : "oli",
    password      : "password",
    connectString : "ORACLE_DEV_DB_TNS_NAME"
  },
  connExecute
);
```

connectString "ORACLE_DEV_DB_TNA_NAME"は、oracleインスタント·クライアントがインストールされているディレクトリまたはじディレクトリのtnsnames.orgファイルにします。

マシンにインストールされているOracleインスタント·クライアントがないは、ごのオペレーティング·システムの`instant client installation guide`います。

パラメータなしでオブジェクトをする

useは、クエリをするためにconnExecute-Functionをするようになりました。クエリをオブジェクトまたはとしてするオプションがあります。はconsole.logにされます。

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

プールしていないをしていたので、をもうするがあります。

```
function connRelease(connection)
{
  connection.close(
    function(err) {
      if (err) {
        console.error(err.message);
      }
    });
}
```

オブジェクトのはのようになります。

```
[ { name: 'C1' }, { name: 'C2' } ]
[ { C1: 'test', C2: 'oracle' } ]
```

のはのようになります。

```
[ { name: 'C1' }, { name: 'C2' } ]
[ [ 'test', 'oracle' ] ]
```

なクエリのためにロ―カルモジュ―ルをする

ORACLE-DBからのせをするために、せをのようにびすことができます。

```
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);
    });
```

このoracle.jsファイルには、のようなコンテンツがまれています。

```
'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;
```

oracleオブジェクトでびすqueryArrayメソッドとqueryObjectメソッドのがあることにしてください。

オンラインでOracleとNode.jsをむ https://riptutorial.com/ja/node-js/topic/8248/oracleとnode-js

48: package.json

`package.json`をのようにすることができます。

```
npm init
```

ライセンスなど、プロジェクトにするなについてします。

Examples

プロジェクト

```
{
    "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"]
}
```

フィールド	
	パッケージをインストールするためになフィールド。、スペースなしのがです。ダッシュとアンダースコアはされています
バージョン	セマンティックバージョニングをしてパッケージバージョンになフィールド。
	プロジェクトのな
	パッケージのをする
	コントリビュータごとに1つのオブジェクトの
キーワード	の、これは々があなたのパッケージをつけるのにちます

"dependencies"{"module-name" "0.1.0"}

- `0.1.0`はモジュールののバージョンをインストールします。
- のマイナーバージョン `^0.1.0`はのマイナーバージョンたとえば`0.2.0` をインストールします

が、メジャーバージョン `1.0.0`モジュールはインストールしません
- のパッチ `0.1.x`または`~0.1.0`えば、できるのパッチバージョンをインストールします`0.1.4` 、しかし、よりいメジャーまたはマイナーバージョンでえばモジュールをインストールしません`0.2.0`または`1.0.0` 。
- ワイルドカード `*`はモジュールのバージョンをインストールします。
- **git**リポジトリは、 **git**リポジトリのマスターブランチからtarballをインストールします。`#sha` 、 `#tag`または`#branch`もできます
    - **GitHub** `user/project`または`user/project#v1.0.0`
    - **url** `git://gitlab.com/user/project.git`または`git://gitlab.com/user/project.git#develop`
- ローカルパス `file:../lib/project`

パッケージをpackage.jsonにしたら、ターミナルのプロジェクトディレクトリにある`npm install`コマンドをします。

devDependencies

```
"devDependencies": {
    "module-name": "0.1.0"
}
```

スタイリングプロキシのテストのように、のためだけになについては、ext。プロダクションモードで "npm install"をしている、これらのdev-dependenciesはインストールされません。

スクリプト

なスクリプト、またはのスクリプトのにトリガーされるスクリプトをできます。

```
{
  "scripts": {
    "pretest": "scripts/pretest.js",
    "test": "scripts/test.js",
    "posttest": "scripts/posttest.js"
  }
}
```

この、のコマンドのいずれかをしてスクリプトをできます。

```
$ npm run-script test
$ npm run test
$ npm test
$ npm t
```

あらかじめされたスクリプト

スクリプト	
	パッケージがされるにします。
し、でする	パッケージがされたにします。
プリインストール	パッケージがインストールされるにしてください。
インストール、ポストインストール	パッケージのインストールにします。
プリインストール、アンインストール	パッケージがアンインストールされるにします。
ポストアンインストール	パッケージがアンインストールされたにします。
バージョン、バージョン	パッケージをバンプするにしてください。
	パッケージをバンプしてからしてください。
テスト、テスト、テスト	`npm test`コマンドでする
プレストップ、、ポストストップ	`npm stop`コマンドでする
プレスタート、、ポストスタート	`npm start`コマンドでする
プレスタート、、ポストレタート	`npm restart`コマンドでする

ユーザーのスクリプト

あらかじめされたスクリプトとじようにのスクリプトをすることもできます。

```
{
  "scripts": {
    "preci": "scripts/preci.js",
    "ci": "scripts/ci.js",
    "postci": "scripts/postci.js"
  }
}
```

この、のコマンドのいずれかをしてスクリプトをできます。

```
$ npm run-script ci
$ npm run ci
```

ユーザースクリプトは、のにすように、 プレスクリプトとポストスクリプトもサポートしています。

プロジェクト

ののいくつかは、 `repository` 、 `bugs`または`homepage`ようなnpmウェブサイトによってされ、このパッケージのインフォボックスにされます

```
{
  "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
  ]
}
```

フィールド	
メイン	このパッケージのエントリースクリプト。このスクリプトは、ユーザーがパッケージをしたときにされます。
リポジトリ	リポジトリのとタイプ
バグ	このパッケージのバグトラッカーgithubなど
ホームページ	このパッケージまたはプロジェクトのホームページ
ファイル	ユーザーが`npm install <packagename>`したときにダウンロードされるファイルとフォルダのリスト`npm install <packagename>`

package.jsonの

はプロジェクトルートにする`package.json`ファイルには、アプリケーションまたはモジュールにするメタデータと、 `npm install`にnpmからインストールするのリストがまれています。

コマンドプロンプトで`package.json`をするには、 `npm init`とします。

デフォルトをして`package.json`をするには、のようにします。

```
npm init --yes
# or
npm init -y
```

パッケージをインストールして`package.json`するには

```
npm install {package name} --save
```

をすることもできます。

```
 npm i -S {package name}
```

NPMのエイリアス`-S`を`--save`と`-D`に`--save-dev`にすると、それぞれプロダクションまたはのをできます。

パッケージはにされます。あなたがしている`--save-dev`わりの`--save` 、パッケージには、あなたのdevDependenciesにされます。

`package.json`なプロパティ

```
{
  "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"
  ]
}
```

なプロパティにする

```
name
```

パッケージのの。にするがあります。このプロパティはであり、パッケージはパッケージがなければインストールされません。

1. は214でなければなりません。
2. はドットまたはアンダースコアでめることはできません。
3. しいパッケージにはにをできません。

```
version
```

パッケージのバージョンはSemantic Versioning semverでします。これは、バージョンがMAJOR.MINOR.PATCHとかれており、

1. のないAPIをうとMAJORバージョン
2. のあるでをすると、MINORバージョン
3. のあるバグをうのPATCHバージョン

```
description
```

プロジェクトの。くにしてください。

```
author
```

このパッケージの。

```
bin
```

パッケージからバイナリスクリプトをするためにされるオブジェクト。オブジェクトは、キーがバイナリスクリプトのであり、がスクリプトへのパスであることをとしています。

このプロパティは、CLIコマンドラインインターフェイスをむパッケージでされます。

```
script
```

のnpmコマンドをするオブジェクト。オブジェクトは、キーがnpmコマンドであり、がスクリプトパスであることをとしています。これらのスクリプトは、 `npm run {command name}`または`npm run-script {command name}`するとされます。

コマンドラインインターフェイスをみ、ローカルにインストールされたパッケージは、パスなしでびすことができます。したがって、`../node-modules/.bin/mocha mocha`をびすわりに、 `mocha`をびすことができます。

```
main
```

パッケージのメインエントリポイント。ノードで`require('{module name}')`をびす`require('{module name}')`、これはになファイルになります。

メインファイルをしてもがしないことをくおめします。たとえば、メインファイルをすると、HTTPサーバーをしたり、データベースにしたりしないでください。わりに、メインスクリプトに`exports.init = function () {...}`ようなものをするがあります。

```
keywords
```

あなたのパッケージをするキーワードの。これらは々があなたのパッケージをつけるのをけるでしょう。

```
devDependencies
```

これらは、モジュールのとテストのみをとしたです。は、 `NODE_ENV=production`がされていないり、にインストールされます。このようなは、 `npm install --dev`をしてこれらのパッケージを`npm install --dev`ことができます`npm install --dev`

```
peerDependencies
```

このモジュールをしている、peerDependenciesには、このモジュールのにインストールするがあるモジュールがリストされます。えば、 `moment-timezone`は、それが`require("moment")`しないでも`require("moment")`、 `moment-timezone`プラグインなので、 `moment`にインストールする`require("moment")`ます。

```
preferGlobal
```

このページが`npm install -g {module-name}`をしてグローバルにインストールされることをむことをすプロパティ。このプロパティは、CLIコマンドラインインターフェイスをむパッケージでされます。

それのは、このプロパティをしないでください。

```
publishConfig
```

publishConfigは、モジュールをするためにされるをつオブジェクトです。されたは、デフォルトのnpmをきします。

`publishConfig`のもないは、プライベートnpmレジストリにパッケージをすることです。そのため、npmのがありますが、プライベートパッケージのもあります。これはにプライベートnpmのURLをレジストリキーのとしてするだけです。

```
files
```

これは、パッケージにめるすべてのファイルのです。ファイルパスまたはフォルダパスのいずれかをできます。フォルダパスのすべてのがまれます。これにより、するしいファイルのみをめることによって、パッケージのサイズがされます。このフィールドは`.npmignore`ルールファイルとして`.npmignore`ます。

ソース

オンラインでpackage.jsonをむ https://riptutorial.com/ja/node-js/topic/1515/package-json

49: PostgreSQLの

Examples

PostgreSQLにする

`PostgreSQL` npmモジュールをしています。

をnpmからインストールする

```
npm  install pg --save
```

これでPostgreSQLをするがあります。これをでいわせることができます。

あなたはDatabase_Name = students、Host = localhost、DB_User = postgresとします

```
var pg = require("pg")
var connectionString = "pg://postgres:postgres@localhost:5432/students";
var client = new pg.Client(connectionString);
client.connect();
```

オブジェクトをしたクエリ

クエリデータベースのオブジェクトをするは、このサンプルコードをできます。

```
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
});
```

オンラインでPostgreSQLのをむ https://riptutorial.com/ja/node-js/topic/7706/postgresqlの

50: RedisとNodeJS

node_redisでででもにされるについてしました。このモジュールをすると、Redisのフルパワーをし、にされたNode.jsアプリケーションをできます。なキャッシングレイヤー、パワフルなPub / Subメッセージングシステムなど、くのいものをこのライブラリでできます。ライブラリのについては、 ドキュメントをしてください。

Examples

Node_redisは、 Node.jsのRedisクライアントです 。のコマンドをしてnpmでインストールできます。

```
npm install redis
```

node_redisモジュールをインストールしたら、すぐにけます。なファイルapp.jsをして、Node.jsからRedisにするをてみましょう

app.js

```
var redis = require('redis');
client = redis.createClient(); //creates a new client
```

デフォルトでは、redis.createClientはホストとポートとしてそれぞれ127.0.0.1と6379をします。のホスト/ポートをしているは、のようにできます。

```
var client = redis.createClient(port, host);
```

は、がされたら、らかのアクションをできます。には、のようにイベントをするだけです。

```
client.on('connect', function() {
    console.log('connected');
});
```

したがって、のスニペットがapp.jsにります

```
var redis = require('redis');
var client = redis.createClient();

client.on('connect', function() {
    console.log('connected');
});
```

さて、アプリケーションをするには、にnode appとしてください。このスニペットをするに、Redisサーバーがしていることをしてください。

キーとのペアの

Node.jsからRedisにするをったので、キーとのペアをRedisストレージにするをてみましょう。

ストリングの

すべてのRedisコマンドは、クライアントオブジェクトでなるとしてされています。なをするには、のをします。

```
client.set('framework', 'AngularJS');
```

または

```
client.set(['framework', 'AngularJS']);
```

のスニペットには、キーフレームワークにしてなAngularJSがされています。のスニペットがじことをうことにしてください。のいは、のものがのをし、でargsを`client.set()`にすことです。がしたら、オプションのコールバックをしてをけることもできます。

```
client.set('framework', 'AngularJS', function(err, reply) {
  console.log(reply);
});
```

なんらかのでがした、コールバックへの`err`はエラーをします。キーのをするには、のをいます。

```
client.get('framework', function(err, reply) {
    console.log(reply);
});
```

`client.get()`は、Redisにされているキーをします。キーのは、コールバックのをしてアクセスできます。キーがしない、replyのはになります。

ハッシュをする

くの、なをしてもはしません。 Redisにハッシュオブジェクトをするがあります。そのために、のように`hmset()`をすることができます

```
client.hmset('frameworks', 'javascript', 'AngularJS', 'css', 'Bootstrap', 'node', 'Express');

client.hgetall('frameworks', function(err, object) {
    console.log(object);
});
```

のスニペットは、をそのフレームワークにマップするRedisにハッシュをします。 `hmset()`ののはキーのです。のは、キーとのペアをします。に、 `hgetall()`をしてキーのをします。キーがつかった、コールバックの2のにはオブジェクトであるがされます。

Redisはネストされたオブジェクトをサポートしていないことにしてください。オブジェクトのす

べてのプロパティは、するににににされます。 Redisにオブジェクトをするには、のをすることもできます。

```
client.hmset('frameworks', {
    'javascript': 'AngularJS',
    'css': 'Bootstrap',
    'node': 'Express'
});
```

オプションのコールバックをして、がいつするかをることもできます。

すべてのコマンドは、の/でびすことができます。たとえば、 `client.hmset()`と`client.HMSET()`はじです。リストの

アイテムのリストをするは、Redisリストをできます。リストをするには、のをします。

```
client.rpush(['frameworks', 'angularjs', 'backbone'], function(err, reply) {
    console.log(reply); //prints 2
});
```

のスニペットは、frameworksというリストをし、それに2つのをプッシュします。したがって、リストのさは2になりました。ごのとおり、 `args`を`rpush`。ののはキーのをし、りはリストのをします。また、 `lpush()`わりに`rpush()`をしてをにプッシュすることもできます。

リストのをするには、 `lrange()`をのようにします。

```
client.lrange('frameworks', 0, -1, function(err, reply) {
    console.log(reply); // ['angularjs', 'backbone']
});
```

`lrange()` 3のとして-1をして、リストのすべてのをすることにしてください。リストのサブセットがなは、ここでインデックスをすがあります。

セットをする

セットはリストとていますが、をさないがなります。したがって、リストにするがないようにするには、セットをできます。のスニペットをリストのわりにセットをするようにするはのとおりです。

```
client.sadd(['tags', 'angularjs', 'backbonejs', 'emberjs'], function(err, reply) {
    console.log(reply); // 3
});
```

ごのように、 `sadd()`はされたでしいセットをします。ここで、のさは3である。セットのメンバーをするには、のように`smembers()`をします。

```
client.smembers('tags', function(err, reply) {
    console.log(reply);
});
```

このスニペットは、セットのすべてのメンバーをします。メンバーをするときにはがされないことにしてください。

これは、すべてのRedisアプリでつかったもなデータのリストでした。、リスト、セット、ハッシュとはに、ソートされたセット、hyperLogLogなどをRedisにすることができます。コマンドとデータのなリストについては、のRedisのドキュメントをごください。ほとんどすべてのRedisコマンドは、node_redisモジュールによってされるクライアントオブジェクトにされています。

node_redisによってサポートされるいくつかのよりな。

キーのの

によっては、キーがすでにするかどうかをし、それにじてをうがあります。これをうには、のようにexists()をexists()ます。

```
client.exists('key', function(err, reply) {
    if (reply === 1) {
        console.log('exists');
    } else {
        console.log('doesn\'t exist');
    }
});
```

キーのとれ

には、いくつかのキーをクリアして、それらをするがあります。キーをするには、のようにdelコマンドをします。

```
client.del('frameworks', function(err, reply) {
    console.log(reply);
});
```

のように、のキーにをすることもできます。

```
client.set('key1', 'val1');
client.expire('key1', 30);
```

のスニペットは、キーkey1に30のをりてます。

インクリメントとデクリメント

Redisは、キーとキーもサポートしています。キーをインクリメントするには、のようにincr()をします。

```
client.set('key1', 10, function() {
    client.incr('key1', function(err, reply) {
        console.log(reply); // 11
    });
});
```

`incr()`は、キーを1つずつインクリメントします。インクリメントするがあるは、 `incrby()`をします。に、キーをデクリメントするには、 `decr()`や`decrby()`などのをできます。

オンラインでRedisとNodeJSをむ https://riptutorial.com/ja/node-js/topic/7107/redisとnodejs

51: Require

き

このドキュメントでは、 NodeJSがそのでする`require()`のとについてします。

Requireは、NodeJSのモジュールでされるのファイルまたはパッケージのインポートです。これは、コードのとをするためにされます。 `require()`は、ローカルにインストールされたファイルでされ、 `require`ファイルからのルートがされます。

- module.exports = {testFunctiontestFunction};
- var test_file = require './ testFile.js'; // `testFile`というのファイルをしましょう
- test_file.testFunctionour_data; // `testFile`を`testFunction`

`require()`をう`require()` 、Javaがクラスやパブリックメソッドをうのとのでコードをすることができます。が`.export`は、のファイルで`require`が`require`ます。 ファイルが`.export`されていないは、のファイルですることはできません。

Examples

とファイルでのrequireのの

Requireは、Nodeがあるで`getter`としてするというです。たとえば、 `analysis.js`というのファイルがあり、ファイルのがのようになっているとします。

```
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...
}
```

このファイルには、 `analyzeWeather(weather_data)`メソッドのみがまれています。このをするは、このファイルのでするか、またはするファイルにコピーするがあります。しかし、Nodeには、モジュールであるコードとファイルのをするになツールがまれています 。

たちのをするためには、まずをのステートメントで`export`するがあります。たちのしいファイルはこのようにえますが、

```
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...
}
```

このさなmodule.exportsステートメントで、はファイルのでできるようになりました。 require()をうだけrequire()。

やファイルrequire、はにています。これは、ファイルのでされ、ファイルでするためにはvarまたはconstされます。たとえば、のようなのファイル handleWeather.jsというのanalyze.jsとじレベルにあります。

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

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

このファイルでは、 analysis.jsファイルをrequire()をしています。するとき、たちはこれにりてられたやをびすrequireとされているのどんなをします。

NPMパッケージでのrequireのの

ノードのrequireは、 NPMパッケージとにするにもにちます。たとえば、 getWeather.jsというのファイルでNPMパッケージrequireをしたいrequireします。 NPMがコマンドラインからパッケージをインストールした git install request 、あなたはそれをするがいました。あなたのgetWeather.jsファイルはこのようにえるかもしれませんが、

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

このファイルがされると、それはにrequireさんあなただけとばれるインストールされたパッケージrequest。 requestファイルのには、アクセスできるくのがあります。そのうちの1つはgetとばれget。のカップルのでは、 HTTP GETをうためにがされています 。

オンラインでRequireをむ https://riptutorial.com/ja/node-js/topic/10742/require--

52: Sequelize.js

Examples

インストール

Node.jsとnpmがにインストールされていることをしてください。に、sequelize.jsをnpmでインストールします。

```
npm install --save sequelize
```

サポートされているデータベースNode.jsモジュールもインストールするがあります。しているものだけをインストールするがあります

MYSQLとMariadb

```
npm install --save mysql
```

PostgreSQL

```
npm install --save pg pg-hstore
```

SQLite

```
 npm install --save sqlite
```

MSSQL

```
npm install --save tedious
```

インストールをセットアップしたら、しいSequalizeインスタンスをみんですることができます。

ES5の

```
var Sequelize = require('sequelize');
var sequelize = new Sequelize('database', 'username', 'password');
```

ES6 - 0バベル

```
import Sequelize from 'sequelize';
const sequelize = new Sequelize('database', 'username', 'password');
```

あなたはすぐなのインスタンスをっています。あなたは、あなたがそれほどうに

```
var db = new Sequelize('database', 'username', 'password');
```

または

```
var database = new Sequelize('database', 'username', 'password');
```

そのがあなたのです。これをインストールしたら、APIドキュメント
http://docs.sequelizejs.com/en/v3/api/sequelize/にってアプリケーションのでできます。

インストールののステップは、 あなたのモデルをセットアップすることです

モデルの

でモデルをするには、2つのがあります。 `sequelize.define(...)` 、または`sequelize.import(...)` 。
のは、sequelizeモデルオブジェクトをします。

1. sequelize.definemodelName、attributes、[options]

これは、すべてのモデルを1つのファイルでしたいや、モデルをにしたいにします。

```
/* 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
    }
});
```

ドキュメントとそののについては、チェックアウトドックレットのドキュメント 、または
sequelize.comのマニュアルを 。

2. sequelize.importpath

モデルがそれぞれのファイルにされている、 `import`はあなたのです。 Sequelizeをするファイル

では、importをのようにびすがあります

```
/* Initialize Sequelize */
// Check previous code snippet for initialization

/* Define Models */
sequelize.import("./models/my_model.js"); // The path could be relative or absolute
```

そして、あなたのモデルファイルでは、あなたのコードはのようになります

```
module.exports = function(sequelize, DataTypes) {
    return sequelize.define("MyModel", {
        name: DataTypes.STRING,
        comment: DataTypes.TEXT,
        date: {
            type: DataTypes.DATE,
            allowNull: false
        }
    });
};
```

`import`いのについては、GitHubの sequelizeのなをしてください 。

オンラインでSequelize.jsをむ https://riptutorial.com/ja/node-js/topic/7705/sequelize-js

53: Socket.ioコミュニケーション

Examples

"こんにちは"ソケットメッセージ。

ノードモジュールをインストールする

```
npm install express
npm install socket.io
```

Node.jsサーバー

```
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!');
});
```

ブラウザクライアント

```
<!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>
```

オンラインでSocket.ioコミュニケーションをむ https://riptutorial.com/ja/node-js/topic/4261/socket-ioコミュニケーション

54: TCPソケット

Examples

シンプルなTCPサーバー

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

なTCPクライアント

```
// 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');
});
```

オンラインでTCPソケットをむ https://riptutorial.com/ja/node-js/topic/6545/tcpソケット

55: Webをする

Examples

GCMGoogle Cloud Messaging Systemをしてウェブをする

このようなでは、 **PWA** プログレッシブWebアプリケーションのなをっています。このでは、**NodeJS**と**ES6**をしてなバックエンドをします

1. インストールノード-GCMモジュール `npm install node-gcm`
2. Socket.ioをインストールします。 npm `npm install socket.io`
3. Googleコンソールをしてgcmアプリケーションをします。
4. GCMアプリケーションIDにけするでになります
5. あなたのGCMアプリケーションのコードにけをしてください。
6. きなコードエディタをき、のコードをします

```
'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
```

PSSocket.ioをExpressでさせるためになハックをしています。これはにボックスでしないためです。

すぐ**.json**ファイルをし、を**Manifest.json**とし 、ファイルをき、

```
{
    "name": "Application Name",
    "gcm_sender_id": "GCM Project ID"
}
```

それをじて、アプリケーションの**ROOT**ディレクトリにします。

PSManifest.jsonファイルはルートディレクトリにあるがあります。そうしないとしません。

のコードでは、はのことをしています

1. はセットアップし、socket.ioもするのindex.htmlページをった。
2. フロントエンド 、つまり**index.html**の フロントエンドからびされたイベントをリッスンしています しいクライアントがみのリンクににされるとされます
3. はなトークンがsocket.io **new_user**イベントをしてのindex.htmlからのトークンとしてのをって、そのようなトークンは、のユーザーのユニークなパスコードされることだし、コードは、Web**API**についてサポートしているブラウザからされるきをみますここに。
4. はに**node-gcm**モジュールをってをします。はでされ、 サービスワーカーをってされます。

これは**NodeJS**のからのものです。のでは、プッシュメッセージにカスタムデータ、アイコンなどをするをします。

PSあなたはここでフルデモをつけることができます。

オンラインでWebをするをむ https://riptutorial.com/ja/node-js/topic/6333/webをする

56: イベントエミッタ

イベントが「する」「イベントの」または「イベントの」とじ、リスナーはして ソース 、 `emit()` にされたデータどのようにくのをすか

```
myDog.on('bark', (howLoud, howLong, howIntense) => {
  // handle the event
})
myDog.emit('bark', 'loudly', '5 seconds long', 'fiercely')
```

リスナーは、されたでびされます。

```
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.
```

しかし、に`prependListener()`リスナーがなは、すでにされているのリスナーのに、`prependListener()`ようにできます。

```
myDog.prependListener('urinate', () => console.log('This happens before my first and second
thoughts, even though it was registered after them'))
```

イベントをくがあるが、それについてだけきたいは、 `on`わりに`once` 、 `prependOnceListener`わりに`prependListener`できます。イベントがしてリスナーがびされると、リスナーはにされ、にイベントがするとびびされません。

に、すべてのリスナーをしてからやりすは、にってください。

```
myDog.removeAllListeners()
```

Examples

イベントエミッタによるHTTPアナリティクス

HTTPサーバーコード `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
```

スーパーバイザーコード `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}`)
})
```

サーバーがをけると、スーパーバイザがリッスンしている`request`というイベントがし、スーパーバイザがそのイベントにします。

イベントエミッタはノードにみまれており、 パブリッシャがイベントをし、 ユーザがいたりしたりできるパターンである、pub-subです。ノードでは、パブリッシャーはイベント・エミッターとばれ、 イベントはし、サブスクライバーはリスナーとばれ、イベントにします。

```
// 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')
```

のでは、dogはパブリッシャ/ EventEmitterであり、アイテムをチェックするはサブスクライバ/リスナでした。よりくのリスナーをることもできます

```
myDog.on('bark', () => {
  console.log('WHO\'S AT THE DOOR?');
  // Panic
});
```

1つのイベントにしてのリスナーがし、リスナーをすることもできます。

```
myDog.on('chew', takeADeepBreathe);
myDog.on('chew', calmDown);
// Undo the previous line with the next one:
myDog.removeListener('chew', calmDown);
```

ィベントを1だけくには、をできます。

```
myDog.once('chew', pet);
```

なしでにリスナーがされます。

しているイベントのをする

EventEmitter.eventNamesは、サブスクライブしているイベントのをむをします。

```
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"]
```

RunKitで

のイベントをするためにされたリスナーのをします。

Emitter.listenerCounteventNameは、としてえられたイベントをリスンしているリスナーのをします

```
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
```

オンラインでイベントエミッタをむ https://riptutorial.com/ja/node-js/topic/1623/イベントエミッタ

57: イベントループ

き

このでは、Eventloopのコンセプトがどのようにし、どのようにサーバーやGUIなどのイベントアプリケーションにできるかについてします。

Examples

イベントループのがどのようにしたか

コードによるイベントループ

イベントループは、イベントをってからそれらのイベントにするループです

```
while true:
    wait for something to happen
    react to whatever happened
```

イベントループのないスレッドのHTTPサーバーの

```
    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)
```

シングルスレッドのイベントループではないなHTTPサーバーのフォームです。ここでのは、がするまでしてから、ののをすることです。 HTTPヘッダーをみったり、ディスクからファイルをするのにがかかるは、ののがされるまでつがあります。

もなは、プログラムをマルチスレッドすることです。

イベントループのないマルチスレッドHTTPサーバーの

```
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)
```

これでさなHTTPサーバーをマルチスレッドしました。このでは、のリクエストがバックグラウンドスレッドでされているため、すぐにのリクエストにむことができます。 Apacheをむくのサーバーでは、このアプローチをしています。

しかしではありません。 1つのは、にくのスレッドしかできないことです。なのがあるワークロードでは、ごとにをうがありますが、マルチスレッドモデルはあまりうまくしません。そのようなのは、イベントループをすることです。

イベントループをつHTTPサーバーの

```
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)
```

うまくいけば、このコードはかりやすいでしょう。ここにはがこっているのですかたちはがこるのをちます。しいがされたり、のにがなには、それにしてからってください。そうすれば、くのがあり、それぞれがをうがほとんどないときはうまくします。

Linuxでされているのアプリケーションコードではないでは、「のイベントがこるのをつ」は、pollまたはepollシステムコールをびすことによってされます。「ソケットへのみきの」のは、ブロックモードでrecvまたはsendシステムコールをびすことによってされます。

[1]。 "イベントループはどのようにするのですか" [オンライン]。 https : //www.quora.com/How-

does-an-event-loop-work

オンラインでイベントループをむ https://riptutorial.com/ja/node-js/topic/8652/イベントループ

58: カサンドラ

Examples

こんにちは

Cassandraにアクセスするには、DataStaxから`cassandra-driver`モジュールをできます。すべてのをサポートしており、にできます。

```
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);
});
```

オンラインでカサンドラをむ https://riptutorial.com/ja/node-js/topic/5949/カサンドラ

59: クライアントとサーバーの

Examples

/ w Express、jQuery、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;
```

//にされるについて https : //gist.github.com/Katamori/5c9850f02e4baf6e9896

オンラインでクライアントとサーバーのをむ https://riptutorial.com/ja/node-js/topic/6222/クライアントとサーバーの

60: クライアントにファイルストリームをする

Examples

fsとパイプをしてサーバーからファイルをストリーミングする

いVODビデオ·オン·デマンドサービスは、からめるべきです。あなたのサーバーのディレクトリには、ユーザーがメディアにアクセスできるようにするらかのポータルやpaywallをして、にアクセスできないディレクトリがあるとします。

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

のスニペットは、ビデオをクライアントにストリーミングするのなです。チャンクロジックは、ネットワークトラフィックやレイテンシをむさまざまなにします。チャックサイズとのバランスをとることがです。

に、.pipeびしにより、node.jsは、サーバーとのをいたままにし、にじてのチャンクをすることをることができます。

Fluent-ffmpegをったストリーミング

flent-ffmpegをして、.mp4ファイルを.flvファイル、またはのタイプにすることもできます。

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 });
```

オンラインでクライアントにファイルストリームをするをむ https://riptutorial.com/ja/node-js/topic/6994/クライアントにファイルストリームをする

61: クラスタモジュール

- const cluster = require "cluster"
- cluster.fork
- cluster.isMaster
- cluster.isWorker
- cluster.schedulingPolicy
- cluster.setupMaster
- cluster.settings
- cluster.worker //ワーカー
- cluster.workers //マスターに

`cluster.fork()`は、のスクリプトをからしめるプロセスをすることにしてください。これは、 Cの`fork()`システムコールとはなり、のプロセスをクローンし、プロセス。

Node.jsのドキュメントには、 ここでのクラスタのよりなガイドがあります

Examples

こんにちは

これはあなたの`cluster.js`です

```
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')();
}
```

これがメインの`server.js`です

```
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;
}
```

このでは、Webサーバーをホストしていますが、みみのクラスタモジュールをしてプロセスをスピンアップします。 forkerプロセスのは、なCPUコアのにします。これにより、Node.jsののインスタンスがのスレッドでされるため、Node.jsアプリケーションでマルチコアCPUをすることができます。アプリケーションは、すべてのプロセスでポート8000をします。は、でRound-Robinメソッドをしてワーカーでにされます。

クラスタの

Node.jsのインスタンスは、のスレッドでされます。マルチコアシステムをするには、Node.jsプロセスのクラスタでアプリケーションをしてをします。

clusterモジュールをすると、すべてのサーバーポートをするプロセスをにできます。

のでは、のコアにわたるをするメインプロセスでワーカーのプロセスをします。

```
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);
}
```

オンラインでクラスタモジュールをむ https://riptutorial.com/ja/node-js/topic/2817/クラスタモジュール

62: グレースフルシャットダウン

Examples

グレースフルシャットダウン - SIGTERM

server.closeと**process.exit**をすることで、サーバーのをキャッチし、にシャットダウンできます。

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

オンラインでグレースフルシャットダウンをむ https://riptutorial.com/ja/node-js/topic/5996/グレースフルシャットダウン

63: コールバックをける

Examples

モジュール

ソースはGitHubからダウンロードできます。または、npmをしてインストールすることもできます。

$ npm install --save async

Bowerとに

$ bower install async

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

モジュール

ありがたいことに、Async.jsのようなライブラリはをしようとしています。 Asyncは、コードのにのいをしますが、コールバックのれをけることによってさをにらすことができます。

シリーズ、パラレル、ウォーターフォールなどのさまざまなでできるくのヘルパーメソッドがAsyncにします。にはのユースケースがあります。

Asyncは、かのように、ではありません。シリーズ、パラレル、などをみわせることでげすのはにです。そのであなたはなコードでまったところにります。でしないようにしてください。いくつかのタスクをしてできるからといって、ずしもそうであるとはりません。には、ノードはシングルスレッドのみであるため、Asyncをしてにタスクをすると、パフォーマンスはほとんどしません。

ソースはhttps://github.com/caolan/asyncからダウンロードできます 。または、npmをしてインストールすることもできます。

$ npm install --save async

Bowerとに

$ bower install async

の

```
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!');
});
```

オンラインでコールバックをけるをむ https://riptutorial.com/ja/node-js/topic/10045/コールバックをける

64: コマンドラインの

Examples

すアクションと

```
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
};
```

ブールスイッチをす

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

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

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

オンラインでコマンドラインのをむ https://riptutorial.com/ja/node-js/topic/6174/コマンドラインの

65: コンソールとの

- console.log[data] [、...]
- console.error[data] [、...]
- console.timelabel
- console.timeEndlabel

Examples

ロギング

コンソールモジュール

JavaScriptのブラウザにています。node.jsは、なロギングとデバッグのをするコンソールモジュールをします。

コンソールモジュールがするもなメソッドは、 `console.log` 、 `console.error` 、および`console.time` です。しかし、 `console.info`ようないくつかのものがあり`console.info` 。

console.log

パラメータはしいで `stdout` にされます。

```
console.log('Hello World');
```

```
> console.log('Hello World')
Hello World
```

console.error

パラメータは、エラー `stderr` にしいで`stderr`されます。

```
console.error('Oh, sorry, there is an error.');
```

```
> console.error("Oh, sorry, error");
Oh, sorry, error
```

console.time、console.timeEnd

`console.time`は、のをするためにできるのlableをつタイマーをします。 `console.timeEnd`をじラベルでびすと、タイマーはし、をミリで`stdout`ます。

```
> console.time("label");
undefined
> console.timeEnd("label");
label: 9297.320ms
```

プロセスモジュール

プロセスモジュールをして、コンソールのにきむことができます。このため、`process.stdout.write`というメソッドがし`process.stdout.write`。 `console.log`とはなり、このメソッドはのにをしません。

のでは、このメソッドは2びされますが、のにしいはされません。

```
> process.stdout.write("123");process.stdout.write("456");
123456true
```

ターミナルコードをして、のりえやカーソルのけなどののコマンドをすることができます。

```
> console.log("\033[31mThis will be red");
This will be red
```

	コード
リセット	`\033[0m`
ハイカラー	`\033[1m`
アンダーライン	`\033[4m`
	`\033[7m`

フォントの

	コード
ブラック	`\033[30m`
	`\033[31m`
	`\033[32m`
	`\033[33m`
	`\033[34m`
マゼンタ	`\033[35m`
シアン	`\033[36m`

	コード
	\033[37m

	コード
ブラック	\033[40m
	\033[41m
	\033[42m
	\033[43m
	\033[44m
マゼンタ	\033[45m
シアン	\033[46m
	\033[47m

オンラインでコンソールとのをむ https://riptutorial.com/ja/node-js/topic/5935/コンソールとの

66: シンプルなRESTベースのCRUD API

Examples

Express 3でのCRUDREST API

```
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");
})
```

オンラインでシンプルなRESTベースのCRUD APIをむ https://riptutorial.com/ja/node-js/topic/5850/シンプルなrestベースのcrud-api

67: ストリームの

パラメーター

パラメータ	
みりなストリーム	データをみむことができるストリームのタイプ
きみなストリーム	データをきむことができるストリームのタイプ
デュプレックスストリーム	みきなストリームタイプ
トランスフォームストリーム	みきのデータをできるストリームのタイプ

Examples

ストリームでTextFileからデータをみむ

ノードのI / Oはであるため、ディスクやネットワークとするにはコールバックをにすがあります。あなたは、のようにディスクからファイルをするコードをこうとするかもしれません。

```
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);
```

このコードはしますが、であり、をクライアントにきすに、すべてのにしてdata.txtファイルをメモリにバッファします。 data.txtがにきいは、にいのユーザーの、くのユーザーににサービスをするため、プログラムのメモリがくなります。

ユーザーは、コンテンツのをするに、ファイルがサーバーのメモリにバッファリングされるのをつがあるため、ユーザーエクスペリエンスがします。

いなことに、req、resはともストリームです。つまり、fs.readFileのわりにfs.createReadStreamをすると、よりいでできます。

```
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);
```

ここでは、.pipeはfs.createReadStreamから 'data'イベントと 'end'イベントをリスンします。このコードはよりされているだけでなく、data.txtファイルはディスクからされるとにに1つのチャンクにきまれます。

のれ

みりなストリームは、きみなストリームに「パイプ」またはできます。これにより、くのをすることなく、ソースストリームからデスティネーションストリームへのデータフローがわれます。

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

var readable = fs.createReadStream('file1.txt')
var writable = fs.createWriteStream('file2.txt')

readable.pipe(writable) // returns writable
```

きみなストリームがみみなストリームの、つまりデュプレックスストリームのは、のきみなストリームにききききできます。

```
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')
```

みみなストリームは、のストリームにパイプすることもできます。

```
var readable = fs.createReadStream('source.css')
readable.pipe(zlib.createGzip()).pipe(fs.createWriteStream('output.css.gz'))
readable.pipe(fs.createWriteStream('output.css')
```

データのれのにににストリームにパイプするがあることにしてください。そうしないと、なデータがストリーミングされるがあります。

また、ストリームオブジェクトはerrorイベントをerrorせることがerrorます。にじて、 すべてのストリームでこれらのイベントをにしてください。

```
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)
```

のみきなストリームをする

ストリームオブジェクトはfsなどのモジュールによってされますが、のストリームオブジェクトをするはどうなりますか。

Streamオブジェクトをするには、NodeJsがするストリームモジュールをするがあります

```
    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);
```

これにより、のカスタムきみなストリームがられます。 *_write*にはもできます。のメソッドはNodeJs 4.xxバージョンでしますが、NodeJs 6.x **ES6**ではクラスがされたため、がされました。は、NodeJsの6.xバージョンのコードです

```
    const Writable = require('stream').Writable;

    class MyWritable extends Writable {
      constructor(options) {
        super(options);
      }

      _write(chunk, encoding, callback) {
        console.log(chunk);
      }
    }
```

なぜストリーム

ファイルのをむためのの2つのをべてみましょう。

のものは、ファイルをみむためのメソッドをし、ファイルがメモリににみまれるとびされるコールバックをします。

```
fs.readFile(`${__dirname}/utils.js`, (err, data) => {
  if (err) {
    handleError(err);
  } else {
    console.log(data.toString());
  }
})
```

そして、ファイルのをみむために`streams`をする2のは、

```
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)
})
```

どちらのもまったくじことをするがあります。いはですか

- のものはく、よりエレガントにえる
- 2つは、ファイルをみんでいるにそのファイルをします

するファイルがさい、 `streams`をするときにのはありませんが、ファイルがきくなるとどうなりますか にきいので、それをメモリにみむのに10かかる

`streams`がなければ、10してファイルがにみまれ 、ファイルのがされるまで、あなたのプロセスがのことをしないり、もしないでっています。

`streams`をすると、ファイルのがになったときにすぐにされ、みみにそのファイルをできるようになります。

のは、コールバックファッションにくときにえないのために`streams`をどのようにできるかをしていないので、のをてみましょう

は`gzip`ファイルをダウンロードし、してそのをディスクにしたいといます。ファイルの`url`これはするがあります

- ファイルをダウンロードする
- ファイルをする
- ディスクにする

ここにの`S3`ストレージにされている[さなファイル] [1]があります。のコードは、コールバックのでをいます。

```
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
```

これは`streams`をしてえるです

```
s3.getObject({Bucket: 'some-bucket', Key: 'tweets.gz'}).createReadStream()
  .pipe(zlib.createGunzip())
  .pipe(fs.createWriteStream(`${__dirname}/tweets.json`));

// 1204 milliseconds
```

`80KB`、さなファイルをうときにはではありません。テストされたファイルのは`80KB`です。これをよりきなファイル `71MB` gzipされていない`382MB` でテストすると、 `streams`バージョンがはるかにであることが`382MB`ます

- これは、ダウンロードするには20925ミリをした`71MB`、それをしてからきみ`382MB` コールバックのファッションをして -ディスクへ。
- これとはに、 `streams`バージョンをする、じをうには13434ミリかかりましたそれほどきなファイルでは35

オンラインでストリームのをむ https://riptutorial.com/ja/node-js/topic/2974/ストリームの

68: ダウンタイムなしでNode.jsアプリケーションをする。

Examples

ダウンタイムなしでPM2をした。

ecosystem.json

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

```
wait_ready
```

リスン·イベントをつのをリロードするのではなく、process.send 'ready'をちます。

```
listen_timeout
```

アプリケーションがリッスンしていないにリロードをするまでのミリ。

```
kill_timeout
```

のSIGKLLをするまでのミリ。

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);
    });
});
```

アプリケーションがあなたのDB /キャッシュ/ワーカー/とのをするのをつがあるかもしれません。PM2は、アプリケーションをオンラインであるとみなすにするがあります。これをうには、プロセスファイルに`wait_ready: true`をするがあります。これにより、PM2はそのイベントをちけます。アプリケーションでは、 `process.send('ready');`をするがあり`process.send('ready');`あなたのアプリケーションをができているとなしたいとき。

プロセスがPM2によって/されると、システムのにはのでプロセスにされるものがあります。

まず`SIGINT`シグナルがプロセスにられます。プロセスがすることをるためにシグナルをまえることができます。 1.6sカスタマイズよりにアプリケーションがでしないは、に`SIGKILL`をけってプロセスをさせます。したがって、アプリケーションがらかのやジョブをクリーンアップするがあるは、 `SIGINT`シグナルをキャッチしてアプリケーションのをすることができます。

オンラインでダウンタイムなしでNode.jsアプリケーションをする。をむ
https://riptutorial.com/ja/node-js/topic/9752/ダウンタイムなしでnode-jsアプリケーションをする-

69: つぶやく

Gruntとgrunt-cliのの、の、またはのバージョンのインストールについてのなは、 gruntのインストールのガイドにあります。

タスクのガイドには、Gruntfileのタスク、ターゲット、オプション、ファイルの、テンプレートの、パターンのグロビング、データのインポートなどのながあります。

タスクのガイドには、Gruntタスクののいがされており、のサンプルタスクとがされています。

Examples

GruntJsの

GruntはJavaScriptタスクランナーであり、ミニフェーション、コンパイル、テスト、リンピングなどのりしタスクのにされます。

めに、Gruntのコマンドラインインターフェイスcliをグローバルにインストールしたいとうでしょう。

```
npm install -g grunt-cli
```

しい**Grunt**プロジェクトのなは、プロジェクトに2つのファイルpackage.jsonとGruntfileをすることです。

package.jsonこのファイルは、npmモジュールとしてされたプロジェクトのメタデータをするためにnpmによってされます。 gruntと、プロジェクトになGruntプラグインをこのファイルのdevDependenciesとしてします。

GruntfileこのファイルのはGruntfile.jsで、タスクのや、Gruntプラグインのみみにされます。

```
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"
  }
}
```

gruntfileの

```
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のインストール

をする

gruntpluginをするには、まずそれをプロジェクトにするものとしてするがあります。としてjshintプラグインをってみましょう。

```
npm install grunt-contrib-jshint --save-dev
```

`--save-dev`オプションをして`package.json`にプラグインをします。このでは、 `npm install`にプラグインがにインストールされます。

プラグインのみみ

あなたはしてgruntfileファイルにあなたのプラグインをみむことができ`loadNpmTasks`。

```
grunt.loadNpmTasks('grunt-contrib-jshint');
```

タスクの

gruntfileにタスクをすると、grunt.initConfigにされたオブジェクトに`jshint`というプロパティがされ`grunt.initConfig`。

```
grunt.initConfig({
  jshint: {
    all: ['Gruntfile.js', 'lib/**/*.js', 'test/**/*.js']
  }
});
```

あなたがしているのプラグインにしてのプロパティをつことができることをれないでください。

タスクの

プラグインをしてタスクをするには、コマンドラインをします。

```
grunt jshint
```

あるいは、のタスクに`jshint`をすることもできます。

```
grunt.registerTask('default', ['jshint']);
```

デフォルトのタスクはオプションなしでターミナルのgruntコマンドでされます。

オンラインでつぶやくをむ https://riptutorial.com/ja/node-js/topic/6059/つぶやく

70: データベースMongoDB with Mongoose

Examples

マングース

にmongodbをさせてください `mongod --dbpath data/`

package.json

```
"dependencies": {
    "mongoose": "^4.5.5",
}
```

server.jsECMA 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.jsECMA 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!'));
```

モデル

あなたのモデルをしてください

app / models / user.jsECMA 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.jsECMA 5.1

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

var userSchema = new mongoose.Schema({
    name: String,
    password: String
});

var User = mongoose.model('User', userSchema);

module.exports = User
```

データをする

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!');
});
```

データをみむ

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');
    }
});
```

オンラインでデータベースMongoDB with Mongooseをむ https://riptutorial.com/ja/node-js/topic/6411/データベース-mongodb-with-mongoose-

71: テンプレートフレームワーク

Examples

ナンジャクス

ブロック、オートエスケープ、マクロ、などをえたサーバーサイドエンジンjinja2ににされ、Twig phpににくています。

Docs - http://mozilla.github.io/nunjucks/
インストール - `npm i nunjucks`

のExpressでのな。

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 %}
```

オンラインでテンプレートフレームワークをむ https://riptutorial.com/ja/node-js/topic/5885/テンプレートフレームワーク

72: ノードjsのcsvパーサー

き

csvからのデータのみみは、さまざまなでできます。 1つのは、 `csv`ファイルをにみむことです。そこからのをうことができます。

Examples

FSをしたCSVでのみり

fsはノードのファイルシステムAPIです。 fsにreadFileメソッドをし、それを`data.csv`ファイル、format、およびにして、 `csv`をみり、してします。

これは、じフォルダに**`data.csv`**というのファイルがあることをとしています。

```
'use strict'

const fs = require('fs');

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

これでののようにをしてできます。

オンラインでノードjsのcsvパーサーをむ https://riptutorial.com/ja/node-js/topic/9162/ノードjsのcsvパーサー

73: ノードJSのローカリゼーション

き

そのローカライゼーションノードをすることはにです

Examples

i18nモジュールをしてノードjsアプリケーションのローカライゼーションをする

ダイナミックjsonストレージをえたのシンプルなモジュール。プレーンなvanilla node.jsアプリケーションをサポートし、resオブジェクトとreqオブジェクトをすapp.useメソッドをするのフレームワークexpress、restify、おそらくそれでするはずです。アプリとテンプレートでの__ '...'をします。 webtranslateit jsonとのあるjsonファイルにファイルをします。あなたのアプリでにされたときにしいをオンザフライでします。なはありません。

+ i18n-node + cookieParserをし、のをける

```
// 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);
```

```
}
```

オンラインでノードJSのローカリゼーションをむ https://riptutorial.com/ja/node-js/topic/9594/ノードjsのローカリゼーション

74: ノードアプリケーションをにさせる

Examples

PM2をプロセスマネージャとしてする

PM2では、nodejsスクリプトをにできます。アプリケーションがクラッシュした、PM2もそれをします。

node2インスタンスをするためにPM2をグローバルにインストールする

```
npm install pm2 -g
```

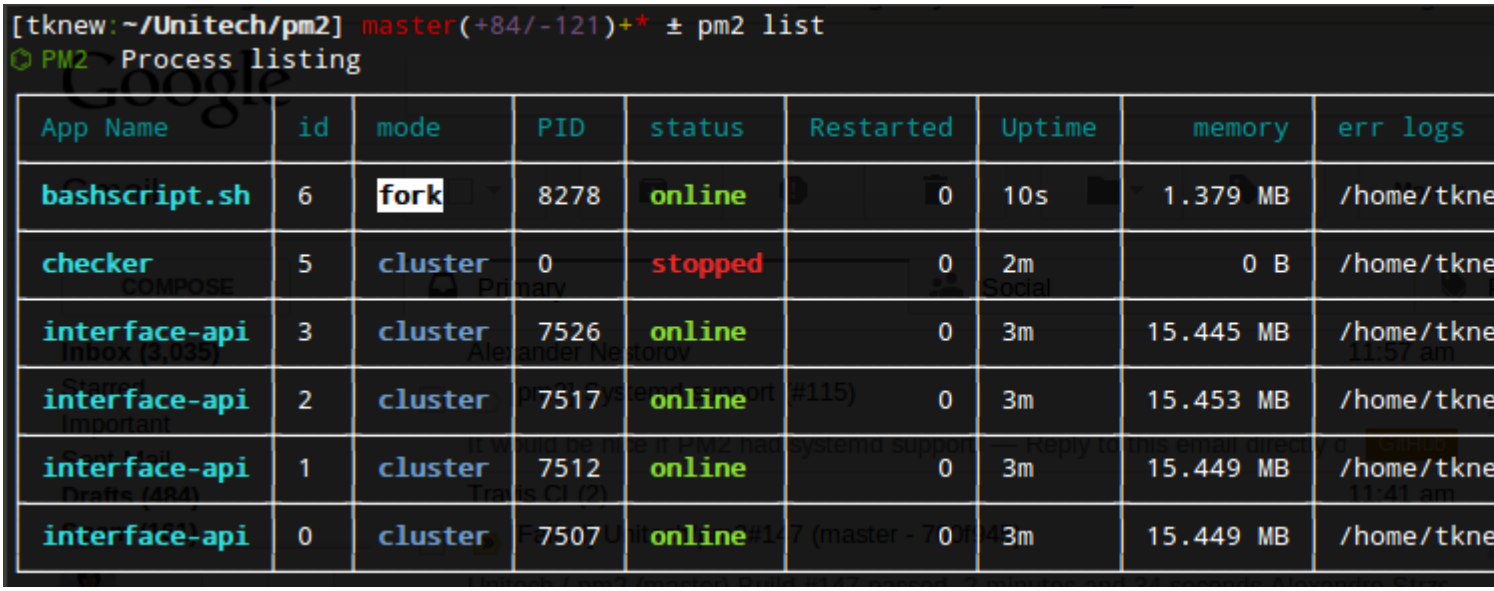
nodejsスクリプトがするディレクトリにし、node2インスタンスをpm2でするようにするたびに、のコマンドをします。

```
pm2 start server.js --name "app1"
```

プロセスをするのになコマンド

1. pm2でされているすべてのnodejsインスタンスをリストする

   ```
   pm2 list
   ```



2. のnodejsインスタンスをする

   ```
   pm2 stop <instance named>
   ```

3. のnodejsインスタンスをする

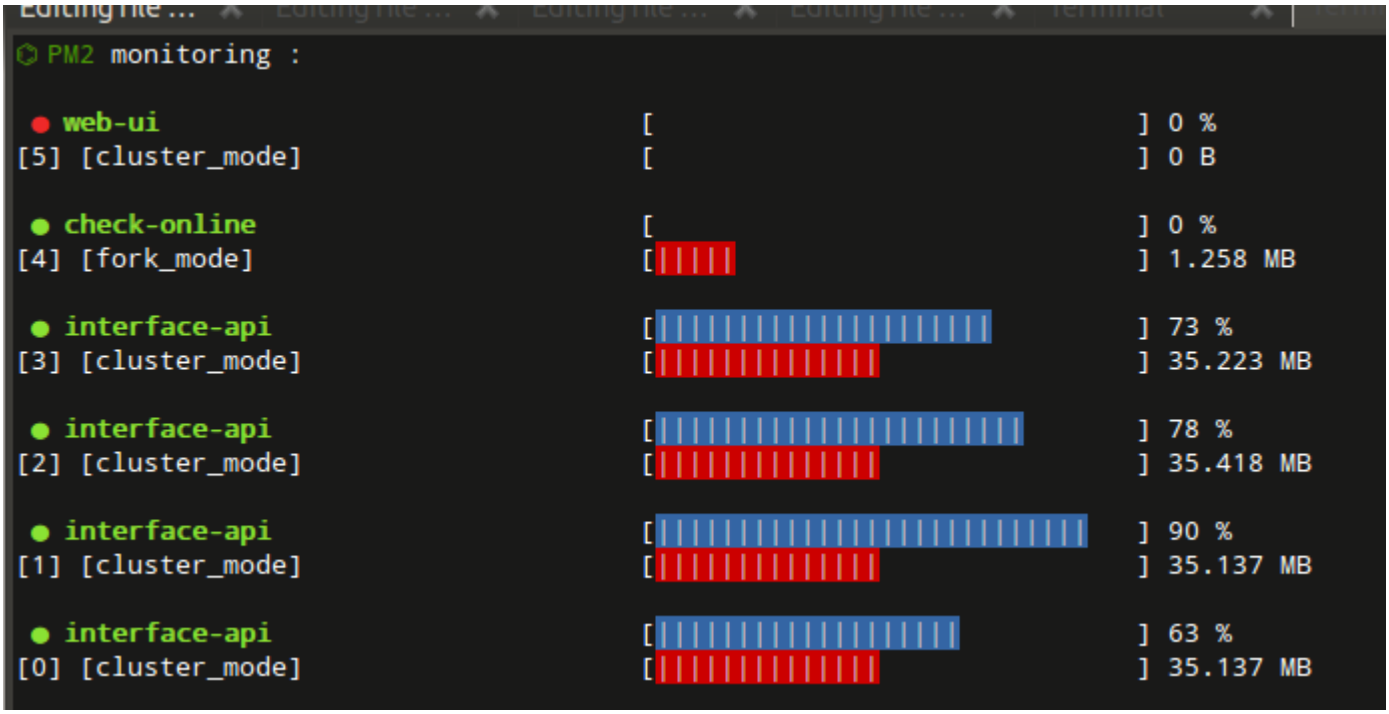
```
pm2 delete <instance name>
```

4. のnodejsインスタンスをする

```
pm2 restart <instance name>
```

5. すべてのnodejsインスタンスの

```
pm2 monit
```



6. pm2をする

```
pm2 kill
```

7. プロセスをしてするとはに、ロードは0のダウンタイムリロードをします

```
pm2 reload <instance name>
```

8. ログをする

```
pm2 logs <instance_name>
```

のデーモンのと

プロセスをするには

```
$ 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
```

のインスタンスをのリスト

```
$ 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 |
```

のプロセスをします。

```
$ forever stop 0

$ forever stop 2146

$ forever stop --uid f4Kt

$ forever stop --pidFile 2131
```

nohupによる

にLinuxにわるものはnohupです。

nohupインスタンスをするには

1. `app.js`または`www`フォルダのにcdします。
2. `nohup nodejs app.js &`する`nohup nodejs app.js &`

プロセスをするには

1. `ps -ef|grep nodejs`する
2. `kill -9 <the process number>`

のプロセスマンゲーション

インストール

```
npm install forever -g
cd /node/project/directory
```

```
forever start app.js
```

オンラインでノードアプリケーションをにさせるをむ https://riptutorial.com/ja/node-js/topic/2820/ノードアプリケーションをにさせる

75: ノードのプロファイリングをする

き

このは、nodejsアプリケーションのプロファイリングをし、このをしてバグやメモリリークをするです。アプリケーションをするnodejsは、くのでブラウザでするWebサイトにたv8エンジンプロセスにぎず、にノードアプリケーションのWebサイトプロセスにするすべてのメトリックをできます。

のみのツールは、chrome devtoolsまたはchrome inspectorとノードインスペクタをみわせたものです。

ノードインスペクタはノードbebugプロセスにアタッチできません。この、devtoolsでデバッグブレークポイントをできなくなります。devtoolsタブをリフレッシュして、デバッグモードになっているかどうかをってください。

しないは、コマンドラインからノードインスペクタをします。

Examples

なノードアプリケーションのプロファイリング

ステップ1 マシンにnpmをしてノードインスペクタパッケージをインストールする

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

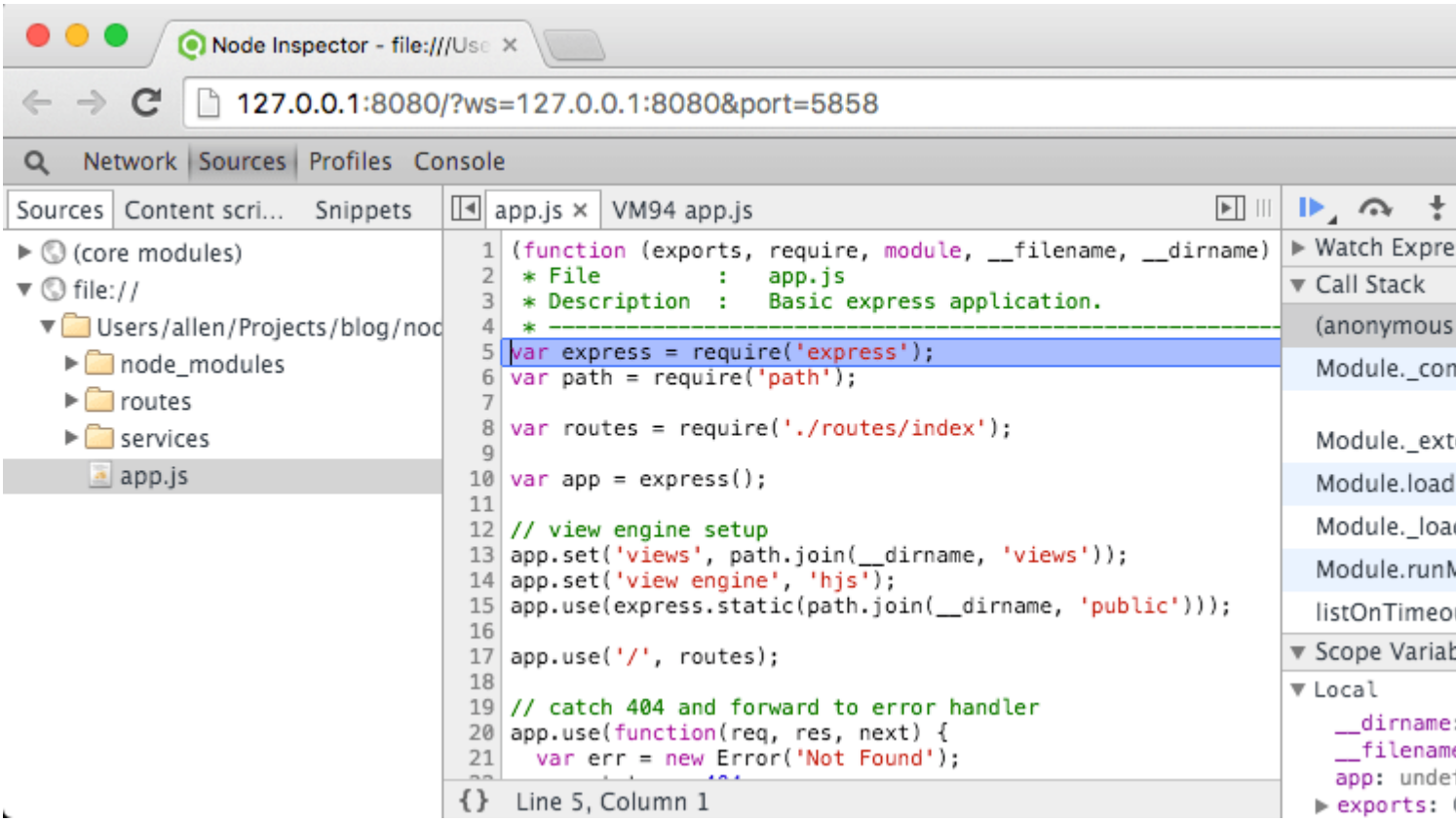
ステップ2 ノードインスペクタサーバをする

```
$ node-inspector
```

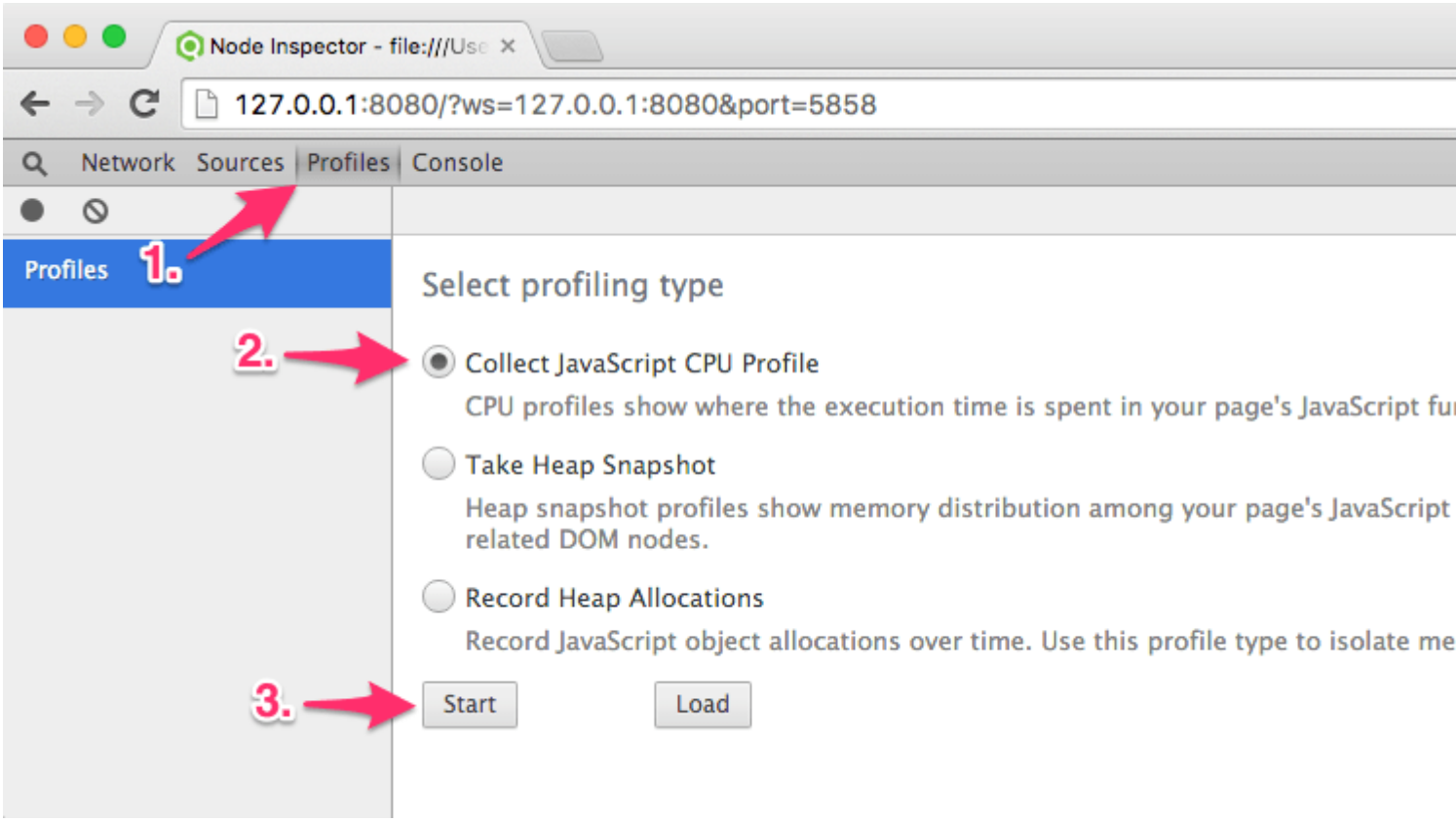
ステップ3 ノードアプリケーションのデバッグをする

```
$ node --debug-brk your/short/node/script.js
```

ステップ4 Chromeブラウザでhttp://127.0.0.1:8080/?port=5858をきます 。また、パネルにあるnodejsアプリケーションのソースコードとchrom-devツールのインタフェースがされます。また、アプリケーションのデバッグにデバッグブレークオプションをしたため、コードのはコードののでします。



ステップ**5** これは、プロファイリングタブにりえてアプリケーションのプロファイリングをするなです。のメソッドやフローのプロファイルをしたいは、そのコードがされるにコードがブレークポイントになっていることをしてください。



ステップ**6** CPUプロファイルまたはヒープダンプ/スナップショットまたはヒープりてをしたら、

をじウィンドウでしたり、ロ―カルドライブにしてでしたり、のプロファイルとすることができます。

このをして、プロファイルをむをることができます

- CPUプロファイルのみみ
- Chrome CPUプロファイラとヒ―ププロファイラ

オンラインでノ―ドのプロファイリングをするをむ https://riptutorial.com/ja/node-js/topic/9347/ノ―ドのプロファイリングをする

76: パスポート.js

き

パスポートは、ノードのなモジュールです。にえば、ユーザのアプリにするすべてのリクエストをします。 Passportは300のをサポートしているため、ログインをFacebook / Googleやのソーシャルネットワークとにすることができます。ここでは、ユーザとパスワードをして、ユーザのデータベースをしてユーザをするローカルのについてします。

Examples

passport.jsのLocalStrategyの

```
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){
});
```

オンラインでパスポート.jsをむ https://riptutorial.com/ja/node-js/topic/8812/パスポート-js

77: パスポートの

パスワードはにハッシュするがあります。 **NodeJS**をしてパスワードをするなは、 **bcrypt-nodejs**モジュールをすることです。

Examples

パスポートは、 `passport.initialize()`ミドルウェアをしてするがあります。ログインセッションをするには、 `passport.session()`ミドルウェアがです。

`passport.serialize()`および`passport.deserializeUser()`メソッドをするがあることにしてください。 **Passport**はセッションとのでユーザーインスタンスをシリアルおよびシリアルします

```
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);
```

ローカル

passport-localモジュールは、ローカルをするためにされます。

> このモジュールでは、Node.jsアプリケーションでユーザーとパスワードをしてすることができます。

ユーザーの

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

ユーザーのログイン

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

```
});
```

ルートの

```
// ...
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

パスポート - フェイスブックモジュールは、 **Facebook**をするためにされます。このでは、ユーザーがサインインにしない、ユーザーがされます。

```
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);
            });
```

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

ルートの

```
// ...
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);
```

なユーザー - パスワード

あなたのルート/ index.jsで

`user`は、userSchemaのモデルです

```
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

Googleではゴ―グルのためにnpmでできるシンプルなモジュ―ルをしています。passport - **google-oauth20**

のをえてみましょう。このでは、ルートディレクトリにpassport.jsとgoogle.jsファイルをつconfigというフォルダがされています。あなたのapp.jsにはがまれています

```
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);
```

//サ―バをするためののコ―ド、エラ―ハンドル

configフォルダのpassport.jsファイルには、のコ―ドがまれています

```
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();
};
```

じフォルダのgoogle.jsファイルにはのものがまれています

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

このでは、ユーザーがDBにない、ユーザーモデルのフィールドgoogleIdをしてローカルのためにDBにしいユーザーをします。

オンラインでパスポートのをむ https://riptutorial.com/ja/node-js/topic/7666/パスポートの

78: ハック

Examples

requireにしいをする

`require.extensions`することによって、 `require()`しいをできます。

XMLの

```
// 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)
            })
        })
    }
}
```

`hello.xml`のがの

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>
    <bar>baz</bar>
    <qux />
</foo>
```

あなたは`require()`をしてそれをんですることができます

```
require('./hello')((err, xml) {
    if (err)
        throw err;
    console.log(err);
})
```

`{ foo: { bar: [ 'baz' ], qux: [ '' ] } }`ます。

オンラインでハックをむ https://riptutorial.com/ja/node-js/topic/6645/ハック

79: パフォーマンスの

Examples

ノードによるクエリの

Nodeはシングルスレッドなので、のにはがです。

これは「ができました」というです。 jQueryをし、なモジュールをインストールすることをれないでください。

こののなロジック

1. クライアントはサーバーにをします。
2. サーバーはのノード・インスタンスでルーチンをし、するタスクIDとともにちにをします。
3. クライアントは、されたタスクIDののをするために、サーバーにチェックをしてします。

プロジェクトの

```
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');
```

data-processor.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"
  }
}
```

このはあなたになえをえることをしています。でするには、がです。

オンラインでパフォーマンスのをむ https://riptutorial.com/ja/node-js/topic/6325/パフォーマンスの

80: ファイルシステムI / O

Node.jsでは、I / Oなどのリソースはにされますが、 したがありますたとえば、 `fs.readFile`がし、するものが`fs.readFileSync`。 Nodeはシングルスレッドなので、 をするときはプロセスをブロックするのでがです。

プロセスがによってブロックされた、サイクルイベントループをむはされます。つまり、イベントやイベントハンドラをむのコードはされず、プログラムはのブロックがするまでしけます。

とのにながありますが、しくされるようにするがあります。

Examples

writeFileまたはwriteFileSyncをしたファイルへのきみ

```
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`とにします`fs.writeFile`が、それはメインスレッドので、ブロックをすると、コールバックをることはありません。ほとんどのnode.jsは、プログラムのにをじさせないバリアントをんでいます。

メインスレッドをブロックすることは、*node.js*ではいことです。は、デバッグまたはのオプションができないときにのみするがあります。

```
// 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);
}
```

ファイルからにみり

すべてのファイルにfilesystemモジュールをします。

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

エンコーディングき

このでは、ディレクトリ`/tmp`から`hello.txt`をみみます。このはバックグラウンドでし、またはにコールバックがします。

```
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);
});
```

エンコーディングなし

バイナリファイル`binary.txt`をのディレクトリから、バックグラウンドででみります。 'encoding' オプションはしないことにしてください。これにより、Node.jsがをにデコードすることができなくなります。

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

パス

な、スクリプトはののディレクトリでできることにしてください。のスクリプトとなファイルをするには、 `__dirname`または`__filename`します。

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

readdirまたは**readdirSync**をしてディレクトリのをする

```
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(' '));
});
```

バリアントは、メインスレッドをブロックする`readdirSync`としてでき、にコードのをします。ほとんどのは、パフォーマンスをさせるためにをしません。

```
let files;

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

ジェネレータの

```
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);
});
```

ファイルからしてみむ

どのファイルでも、filesystemモジュールがです

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

をむ

`fs.readFileSync`とにします`fs.readFile`が、それはメインスレッドので、ブロックをすると、コールバックをることはありません。ほとんどのnode.jsは、プログラムのにをじさせないバリアントをんでいます。

`encoding`オプションがされているはがされ、そうでないは`Buffer`がされます。

```
// 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);
}
```

unlinkまたはunlinkSyncをしてファイルをする

でファイルをする

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

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

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

にすることもできます*

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

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

*メソッドは、がするまでプロセスをブロックするのでけてください。

ストリームをしてファイルをバッファにみむ

`fs.readFile()`メソッドをすると、ファイルからコンテンツをみむことはすでにですが、なコールバックではなく、ストリームでデータをしたいことがあります。これにより、このデータをのにパイプしたり、わりににすべてのデータをしたりすることができます。

```
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!
});
```

ファイルまたはディレクトリのアクセスをする

`fs.access()`は、パスがするかどうか、およびユーザーがそのパスのファイルまたはディレクトリにするアクセスをします。 `fs.access`はをしません。エラーをさない、パスはし、ユーザーにはながえられます。

パーミッションモードは、 `fs`オブジェクトのプロパティとしてできます`fs.constants`

- `fs.constants.F_OK` - みり/きみ/を`fs.constants.F_OK`ますモードがされていないはこれがデフォルトです
- `fs.constants.R_OK` - みみがあります
- `fs.constants.W_OK` - きみをっています
- `fs.constants.X_OK` - を`fs.constants.F_OK`ますWindowsでは`fs.constants.F_OK`とじように`fs.constants.F_OK`します

に

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

に

`fs.access` 、バージョンがある`fs.accessSync`。 `fs.accessSync`を`fs.accessSync`は、try / catchブロックにむがあります。

```
// 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);
}
```

のディレクトリをまたはするのの

ノードののため、にディレクトリをまたはする

1. `fs.stat()`でそのをし、に
2. チェックのにじてまたはするか、

チェックのとのとのにフォルダがされた、 になるがあります。のメソッドは、 `fs.mkdir()`と`fs.mkdirSync()`を、そのコードが`EEXIST` すでにするのにをけしさせるエラーキャッチラッパーにラップします。エラーが`EPERM` pemission deniedのようなものであれば、ネイティブのようにエラーをげたりしたりします。

`fs.mkdir()`を`fs.mkdir()`バージョン

```
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

});
```

fs.mkdirSync()とのバ―ジョン

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

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

ファイルまたはディレクトリがするかどうかをする

に

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

に

ここでは、エラ―をするために`try/catch`ブロックにびしをラップするがあります。

```
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');
  }
}
```

ストリームをしたファイルのクローン

このプログラムは、ファイルシステムモジュールによってされる`createReadStream()`および`createWriteStream()`をして、みきなストリームをしてファイルをコピーするをしています。

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

パイプストリームによるファイルのコピー

このプログラムは、ストリームクラスによってされる`pipe()`をして、みみなストリームときみなストリームをしてファイルをコピーします。

```
// 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);
```

テキストファイルのをする

- なRegExp `replace(/email/gim, 'name')`というテキストファイル`index.txt` `name`に`email`という`name`

を`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!');
    })
})
```

テキストファイルのの

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
});
```

ノードアプリ

でファイルをみむ

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
});
```

ノードアプリ

オンラインでファイルシステムI / Oをむ https://riptutorial.com/ja/node-js/topic/489/ファイルシステムi---o

81: ファイルのアップロード

Examples

multerをしたのファイルアップロード

えている

- `uploads`のフォルダをしますでは`uploads`。
- multer `npm i -S multer`インストールします。

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>
```

をつファイルをアップロードするには、Node.js パスみみライブラリをできます

そのためには、 `server.js`ファイル`path`がです。

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

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

のでファイルをします。

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

によるアップロードをフィルタリングする

このでは、ファイルをアップロードしてののみをするをします。

たとえば、ののみです。 `var upload = multer({ storage : storage}).single('userFile');`して`var upload = multer({ storage : storage}).single('userFile');` fileFilter

```
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');
```

これで、 `png` 、 `jpg` 、 `gif`または`jpeg`のファイルのみをアップロードできます

なモジュールをする

モジュールをインストールしてドキュメントをむ

```
npm i formidable@latest
```

8080ポートのサーバーの

```
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);
```

オンラインでファイルのアップロードをむ https://riptutorial.com/ja/node-js/topic/4080/ファイルのアップロード

82: プッシュ

き

ですから、Webアプリケーションをしたいのであれば、Web /モバイルアプリケーションにPush.jsまたはSoneSignalフレームワークをすることをおめします。

プッシュは、Javascriptをしてしてするのです。にかなりしくされたNotification APIにより、Chrome、Safari、Firefox、IE 9+などののブラウザでユーザーのデスクトップにをできます。

あなたはSocket.ioといくつかのバックエンドフレームワークをするがあります、はこののためのユーザーエクスプレスです。

パラメーター

モジュール/フレームワーク	
node.js / express	Node.jsアプリケーションのためのシンプルなバックエンドフレームワーク、いやすくに
Socket.io	Socket.IOは、リアルタイムイベントベースのをにします。これは、すべてのプラットフォーム、ブラウザ、またはデバイスでし、とスピードにしくします。
Push.js	でものいデスクトップフレームワーク
OneSignal	Appleデバイスのプッシュをオフにするのフォーム
ファイアベース	Firebaseは、Googleのモバイルプラットフォームであり、のアプリをにしてビジネスをさせるのにちます。

Examples

Web

まず、 Push.jsモジュールをインストールするがあります。

```
$ npm install push.js --save
```

または、 CDNでフロントエンドアプリにインポートする

```
<script src="./push.min.js"></script> <!-- CDN link -->
```

あなたがそれをませたら、あなたはかなくてはなりません。これはなをしたいのようにえるです

```
Push.create('Hello World!')
```

はあなたのアプリでSocket.ioをするをっているとします。 Expressをしたバックエンドアプリケーションのコードをいくつかします

```
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' });

});
```

サーバーがすべてセットアップされたら、フロントエンドのものにすることができます。これでSocket.io CDNをimportし、このコードを*index.html*ファイルにするだけです。

```
<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>
```

そこには、、あなたのをすることができるはず、く、これはまた、のAndroidデバイスでし、uはしたいはFirebaseクラウドメッセージングを、あなたは、このモジュールでそれをすることができ、 ここでニックによってかれているのリンクですPush.jsの

これはAppleデバイスではしませんはそれらをすべてテストしませんでしたが、プッシュでOneSignalプラグインをチェックするようにしたいはしてください。

オンラインでプッシュをむ https://riptutorial.com/ja/node-js/topic/10892/プッシュ

83: ブルーバードの

Examples

ノードバックライブラリをにする

```
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)
})
```

の

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

フィルターの

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

の

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

コルーチンジェネレータ

```
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)
```

リソース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
})
```

シリーズでの

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

オンラインでブルーバードのをむ https://riptutorial.com/ja/node-js/topic/6728/ブルーバードの

84: フレームワークのないノードサーバー

Nodeには、にサーバーのとにつくのフレームワークがありますが、

Express もよくわれるフレームワーク

すべてをち、のフレームワークやモジュールにしないALL-IN-ONE UNITYフレームワーク。

しかし、にすべてのサイズにするサイズはないので、はのなしにのサーバーをするがあります。

もしがサーバーでアクセスした、 CORSがになるがあります。するためのコードがされています。

Examples

フレームワークレスのノードサーバー

```
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/');
```

CORSのをする

```
// 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);
```

オンラインでフレームワークのないノードサーバーをむ https://riptutorial.com/ja/node-js/topic/5910/フレームワークのないノードサーバー

85: プロジェクトの

き

nodejsプロジェクトのは、なみ、プロジェクトのアーキテクチャー、モジュールのをけます。また、モジュールのインスタンスメカニズムをするイベントベースのarc 'もあります。 MVCをつためには、クライアントサイドのコードがおそらくされブラウザにられ、なされているので、サーバとクライアントのソースコードをすることがです。そして、サーバーまたはバックエンドでCRUDをするためのAPIがされます

のプロジェクトでは、アプリケーションベースビューとミニディングライブラリとしてbrowserifyおよびvue.jsモジュールをしています。したがって、プロジェクトは、あなたがするmvcフレームワークにづいてかくされるがあります。たとえば、publicディレクトリにbuildディレクトリにはすべてのmvcコードがまれているがあります。あなたはあなたのためにこれをうタスクをつことができます。

Examples

MVCとAPIをえたなnodejsアプリケーション

- のきないは、ホスティングにされるにされるディレクトリとソースディレクトリのです。
- ソースディレクトリには、のにじてファイルまたはフォルダがあります。これには、とビジネスロジックのがまれています。これをconfigディレクトリにすることもできます。

```
|-- Config
       |-- config.json
       |-- appConfig
           |-- pets.config
           |-- payment.config
```

- 、サーバー/バックエンドとフロントエンドモジュールをするもなディレクトリです。 2つのディレクトリ*server*と*webapp*は、それぞれバックエンドとフロントエンドをしています。 *src* 。

  > がにかなっているかによって、サーバーやWebアプリケーションのなにじてなるでくことができます。それをあまりにもくするか、のプロジェクトのわりにあるようににしたくないことをしてください。

- サーバーディレクトリのにコントローラ、メインノードのファイルととなるApp.js / index.jsをつことができます。サーバーディレクトリ。 APIコントローラによってされるすべてのデータオブジェクトをする*dto* dirをつこともできます。

  ```
  |-- server
  ```

```
      |-- dto
          |-- pet.js
          |-- payment.js
      |-- controller
          |-- PetsController.js
          |-- PaymentController.js
      |-- App.js
```

- webappディレクトリは、 *public*と*mvc*の 2つのなにけられます。これは、したいビルドによってもをけます。たちはbrowserfiy webappのMVCのをし、にれて*mvc*ディレクトリからコンテンツをにえてしています。

  | - webapp | - | - mvc

- パブリックディレクトリには、すべてのリソース、、csssaasファイルもあります、そしてもなのはHTMLファイルがまれています。

```
|-- 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
```

- *mvc*ディレクトリには、 モデル 、 ビューコントローラ 、およびUIのとしてなそののユーティリティモジュールをむフロントエンドロジックがまれます。また、index.jsまたはshell.jsのいずれかがこのディレクトリのであるかもしれません。

```
  |-- mvc
      |-- controllers
          |-- Dashborad.js
          |-- Help.js
          |-- Login.js
      |-- utils
      |-- index.js
```

だから、にプロジェクトのがbelow.AndのようなMVCのスクリプトをし、 ディレクトリにする *browserify*のようななビルドタスクをていきます。このディレクトリを***express.usesatic 'public'*** APIをしてリソースとしてすることができます。

```
    |-- 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
```

オンラインでプロジェクトのをむ https://riptutorial.com/ja/node-js/topic/9935/プロジェクトの

86: プロミスとエラーファーストコールバックのをサポートするNode.jsライブラリの

き

や/をっているはいですが、モジュールをくときには、なコールバックスタイルのメソッドもサポートするプログラマーにとってはです。 2つのモジュール、または2つのセットをするのではなく、プログラマーがあなたのモジュールをするのではなく、あなたのモジュールはbluebirdのasCallbackまたはQのnodeifyをってのプログラミングメソッドをサポートできます。

Examples

Bluebirdをしたモジュールおよびするプログラムの

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);
  }

})();
```

オンラインでプロミスとエラーファーストコールバックのをサポートするNode.jsライブラリのをむ https://riptutorial.com/ja/node-js/topic/9874/プロミスとエラーファーストコールバックのをサポートするnode-jsライブラリの

87: マルチスレッド

き

Node.jsはシングルスレッドにされています。なのために、Nodeでするアプリケーションは1つのスレッドでされます。

しかし、Node.jsはマルチスレッドでします。I/Oなどはスレッドプールからされます。さらに、ノードアプリケーションのインスタンスはすべてのスレッドでされるため、マルチスレッドアプリケーションをするためにのインスタンスがされます。

イベントループをすることは、のスレッドをするとをするでです。

Examples

クラスタ

`cluster`モジュールをすると、じアプリケーションをすることができます。

クラスタリングは、なるインスタンスがじフローをち、いにしないにましいです。このシナリオでは、フォークとフォークまたはをできるマスターが1つあります。たちはしてき、RamとEvent Loopの1つのスペースをっています。

クラスタをすることは、ウェブサイト/APIにとってです。のスレッドにしないため、どのスレッドものにサービスをできます。 はできないので 、Cookieをするにはデータベースredisなどをします。スレッド。

```
// 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');
```

プロセス

プロセスは、なるとをってしてプロセスをしたいときにくです。クラスターのフォークのように、 `child_process`はスレッドでされますが、フォークとはなり、プロセスとするがあります。

コミュニケーションはにむので、とはメッセージをいてメッセージをできます。

../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');
```

../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?');

})
```

メッセージのには、 'error'、 'connected'、 'disconnect'などのくのイベントをくことができます。

プロセスのには、それにするのコストがあります。なりのデータをしたいとえています。

オンラインでマルチスレッドをむ https://riptutorial.com/ja/node-js/topic/10592/マルチスレッド

88: メタルミス

Examples

シンプルなブログをする

ノードとnpmがインストールされ、であるとすると、な`package.json`つプロジェクトフォルダをします。なをインストールします。

```
npm install --save-dev metalsmith metalsmith-in-place handlebars
```

プロジェクトフォルダのルートにをむ`build.js`という`build.js`ファイルをします。

```
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!');
  });
```

プロジェクトフォルダのルートに`src`というのフォルダをします。のをむ`src`に`index.html`をします。

```
---
title: My awesome blog
---
<h1>{{ title }}</h1>
```

`node build.js`をすると、 `src`すべてのファイルがされます。このコマンドをすると、ビルドフォルダにのの`index.html`がされます。

```
<h1>My awesome blog</h1>
```

オンラインでメタルミスをむ https://riptutorial.com/ja/node-js/topic/6111/メタルミス

89: モジュールのエクスポートと

Node.jsのすべてはにでわれますが、 `require()`はそれらのものの1つではありません。モジュールはにはしかロードするがないため、ブロックであり、しくするがあります。

モジュールは、ロードされたにキャッシュされます。のモジュールをするは、しいをするためにモジュールキャッシュのそのエントリをするがあります。つまり、たとえモジュールがモジュールキャッシュからクリアされても、モジュールはガベージコレクションされないため、でのにはがです。

Examples

モジュールのロードと

モジュールは`require()`によって "import"されるか、そうでなければ " `require()` "されます。たとえば、Node.jsにされている`http`モジュールをロードするには、のようにします。

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

ランタイムにされているモジュールのに、npmからインストールしたモジュールexpressなどをすることもできます。 `npm install express`システムにインストールしていたは、のようにくだけです。

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

また、アプリケーションのとしてでしたモジュールをめることもできます。この、のファイルとじディレクトリに`lib.js`というのファイルをめるには、のようにします。

```
const mylib = require('./lib');
```

をすることができ、 `.js`がきがれることにしてください。モジュールをロードすると、には、なファイルからされたメソッドとプロパティをむオブジェクトがみまれます。な

```
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);
```

hello-world.jsモジュールの

Nodeは、とをのファイルにする`module.exports`インターフェイスをします。これをうもなは、のにすように、1つのオブジェクトまたはのみをエクスポートすることです。

hello-world.js

```
module.exports = function(subject) {
    console.log('Hello ' + subject);
};
```

エクスポートをのオブジェクトにしたくない、とを`exports`オブジェクトのプロパティとしてエクスポートできます。の3つのはすべて、これをわずかになるでしています。

- hello-venus.jsのは々にわれ、 `module.exports`プロパティとしてされます
- hello-jupiter.jsは`module.exports`のプロパティのとしてされます
- hello-mars.jsは、 `exports`プロパティとしてされています。これは、いバージョンの `module.exports`

hello-venus.js

```
function hello(subject) {
    console.log('Venus says Hello ' + subject);
}

module.exports = {
    hello: hello
};
```

hello-jupiter.js

```
module.exports = {
    hello: function(subject) {
      console.log('Jupiter says hello ' + subject);
    },

    bye: function(subject) {
      console.log('Jupiter says goodbye ' + subject);
    }
};
```

hello-mars.js

```
exports.hello = function(subject) {
    console.log('Mars says Hello ' + subject);
};
```

ディレクトリをむモジュールのみみ

`hello`というのディレクトリがあり、のファイルがまれています。

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'
```

モジュールキャッシュの

に、じモジュールにして`require()`をすると、そのファイルにをえたとしても、にじモジュールがされることがあります。これは、モジュールがにロードされたときにモジュールがキャッシュされ、のモジュールのロードがキャッシュからロードされるためです。

このを`delete`には、キャッシュのエントリを`delete`があります。たとえば、モジュールをロードしたはのようになります。

```
var a = require('./a');
```

その、キャッシュエントリをすることができます

```
var rpath = require.resolve('./a.js');
delete require.cache[rpath];
```

そして、モジュールをする

```
var a = require('./a');
```

この`delete`は、ロードされたデータではなく、ロードされたモジュールへののみをするため、これはではおめできません。モジュールはガベージコレクションされていないため、このをにするとメモリがリークするがあります。

のモジュールをする

また、オブジェクトをしてし、そのオブジェクトにメソッドをにすることもできます。

```
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
}
```

これらのモジュールをするには、モジュールをどおりにするがあります。

```
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()
  })
}
```

すべてのモジュールが1のみされる

NodeJSは、めてなときにのみモジュールをします。さらになは、じObjectをするため、のにモジュールのコードをしません。また、ノードはrequireをしてロードされたモジュールをにキャッシュします。これにより、ファイルのみみがり、アプリケーションのがします。

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
```

node_modulesからのモジュールのロード

モジュールは、パスをせずに、 node_modulesというなディレクトリにくことによってdをrequireことができます。

たとえば、ファイルindex.jsからfooというモジュールをrequireするには、のディレクトリをできます。

```
index.js
 \- node_modules
  \- foo
   |- foo.js
   \- package.json
```

モジュールは、 package.jsonファイルとにディレクトリのにするがあります。 package.jsonファイルのmainフィールドは、モジュールのエントリポイントをすがあります。これは、ユーザーがrequire('your-module')ときにインポートされるファイルrequire('your-module')です。 mainデフォルトindex.jsされていない。あるいは、 requireびしへのパスをするだけで、モジュールにするファイルをすることができます require('your-module/path/to/file')。

モジュールは、 node_modulesディレクトリからファイルシステムまでのdをrequireこともできます。たちがのディレクトリをっているなら、

```
my-project
\- node_modules
 |- foo   // the foo module
  \- ...
 \- baz   // the baz module
  \- node_modules
   \- bar   // the bar module
```

require('foo')をってbarののファイルからモジュールfooをrequireすることができます。

nodeはファイルシステムのファイルにもいモジュールファイルののディレクトリ/ node_modulesからのみすることにしてください。ノードは、このでファイルシステムのルートまでのディレクトリをします。

npmレジストリまたはのnpmレジストリからしいモジュールをインストールするか、のものをる

ことができます。

モジュールとしてのフォルダ

モジュールは、じフォルダののjsファイルにできます。 *my_module*フォルダの

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');
```

このようなモジュールは、フォルダでされます

```
var split_module = require('./my_module');
```

なは、requireのから`./`またはフォルダへのパスのをして*node_modules*フォルダからモジュールをロードしようとします。

あるいは、じフォルダに`package.json`ファイルをのですることもできます。

```
{
    "name": "my_module",
    "main": "./your_main_entry_point.js"
}
```

このようにして、メインモジュールファイル "index"にをけるはありません。

オンラインでモジュールのエクスポートとをむ https://riptutorial.com/ja/node-js/topic/547/モジュールのエクスポートと

90: モジュールのロード

Examples

グローバルモード

グローバルモードでデフォルトディレクトリをしてNodeをインストールした、NPMは`/usr/local/lib/node_modules`パッケージをインストールします。シェルにのようにすると、NPMはディレクトリ`/usr/local/lib/node_modules/express`のsaxというのパッケージのバージョンを、ダウンロード、インストールします。

```
$ npm install -g express
```

フォルダになアクセスがあることをしてください。これらのモジュールは、そのマシンでされるすべてのノードプロセスでになります

ローカルモードインストール。 Npmは`/home/user/apps/my_app`ある、えば`node_modules`という`node_modules`しいフォルダをしてのフォルダにモジュールをダウンロードしてインストールします。しいフォルダは`node_modules /home/user/apps/my_app/node_modules`まだしない

モジュールのみみ

コードのモジュールをすると、ノードはにされたフォルダーの`node_module`フォルダーをなステートメントで`node_module`ます。モジュールがでなくコアモジュールではない、Nodeはの`node_modules`フォルダーでモジュールをしようとしますディレクトリ。たとえば、のをうと、Nodeは`./node_modules/myModule.js`ファイルをします。

```
var myModule = require('myModule.js');
```

ノードがファイルのにした、ノードは`../node_modules/myModule.js`というフォルダをします。すると、フォルダをし、ルートにするか、なモジュールがつかるまでしけます。

にじて`.js`をすることもできます。この、ノードで`.js`がされ、ファイルがされます。

フォルダモジュールのロード

フォルダのパスをすると、のようにモジュールをロードできます。

```
var myModule = require('./myModuleDir');
```

そうすると、Nodeはそのフォルダをします。ノードはこのフォルダがパッケージであるとし、パッケージをします。そのパッケージは`package.json`というのファイルでなければなりません。その

フォルダに`package.json`というのパッケージファイルがまれていない、パッケージエントリポイントは`index.js`のデフォルトを`./myModuleDir/index.js`ます`./myModuleDir/index.js`はパス`./myModuleDir/index.js`のファイルを`./myModuleDir/index.js`。

モジュールがいずれのフォルダにもつからないののは、グローバルモジュールのインストールフォルダです。

オンラインでモジュールのロードをむ https://riptutorial.com/ja/node-js/topic/7738/モジュールのロード

91: モンゴース

Examples

MongooseをってMongoDBにする

まず、Mongooseをのものとともにインストールします。

```
npm install mongoose
```

に、それをとしてserver.jsにしserver.js。

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
```

に、データベーススキーマとコレクションのをします。

```
var schemaName = new Schema({
    request: String,
    time: Number
}, {
    collection: 'collectionName'
});
```

モデルをし、データベースにします。

```
var Model = mongoose.model('Model', schemaName);
mongoose.connect('mongodb://localhost:27017/dbName');
```

に、MongoDBをし、 server.jsをしてserver.jsをしnode server.js

々はデータベースににしているかどうかをするために、々は、イベントをすることができopen 、errorからmongoose.connectionオブジェクトを。

```
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
  // we're connected!
});
```

MongooseとExpress.jsルートをってMongoDBにデータをする

セットアップ

まず、のパッケージをインストールします。

```
npm install express cors mongoose
```

コード

に、 `server.js`ファイルにをし、データベーススキーマとコレクションのをし、Express.jsサーバーをし、MongoDBにします。

```
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);
});
```

に、データのきみにするExpress.jsルートをします。

```
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)
        }
    })
})
```

ここでは、 `query`はHTTPリクエストの`<query>`パラメータになります。これはMongoDBにされます

```
var savedata = new Model({
    'request': query,
    //...
```

MongoDBへのきみにエラーがすると、コンソールにエラーメッセージがされます。すべてしたは、したデータがJSONでページにされます。

```
//...
}).save(function(err, result) {
    if (err) throw err;

    if(result) {
        res.json(result)
    }
})
//...
```

さて、あなたはMongoDBのをしてするが`server.js`してファイルを`node server.js`。

これをしてデータをするには、ブラウザののURLにアクセスしてください

```
http://localhost:8080/save/<query>
```

ここで`<query>`はしたいしいです。

```
http://localhost:8080/save/JavaScript%20is%20Awesome
```

JSONでの

```
{
    __v: 0,
    request: "JavaScript is Awesome",
    time: 1469411348,
    _id: "57957014b93bc8640f2c78c4"
}
```

MongooseとExpress.jsルートをしたMongoDBのデータの

セットアップ

まず、のパッケージをインストールします。

```
npm install express cors mongoose
```

コード

に、 `server.js`にをし、データベーススキーマとコレクションのをし、Express.jsサーバーをし、MongoDBにします。

```
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);
});
```

に、データのクエリにするExpress.jsルートをします。

```
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'
            }))
        }
    })
})
```

のドキュメントがモデルのコレクションにまれているとします。

```
{
        "_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
}
```

そして、は、 `"request"`キーのに`"JavaScript is Awesome"`をむすべてのをつけてすることです。

このためには、MongoDBをし、 `server.js`という`node server.js`をし`node server.js`。

これをしてデータをするには、ブラウザののURLにアクセスしてください

```
http://localhost:8080/find/<query>
```

`<query>`はクエリです。

```
http://localhost:8080/find/JavaScript%20is%20Awesome
```

```
[{
    _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
}]
```

Mongoose、Express.jsルートと**$ text**をして**MongoDB**でデータをする

セットアップ

まず、のパッケージをインストールします。

```
npm install express cors mongoose
```

コード

に、 `server.js`にをし、データベーススキーマとコレクションのをし、Express.jsサーバーをし、MongoDBにします。

```
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);
});
```

に、データのクエリにするExpress.jsルートをします。

```
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'
            }))
        }
    })
})
```

のドキュメントがモデルのコレクションにまれているとします。

```
{
        "_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
}
```

そして、は、 `"request"`キーので`"JavaScript"`だけをむすべてのをつけてすることです。

これをうには、まずコレクションの`"request"`テキストインデックスをします 。このために、`server.js`のコードをします。

```
schemaName.index({ request: 'text' });
```

そして、

```
    Model.find({
        'request': query
    }, function(err, result) {
```

をして

```
Model.find({
    $text: {
        $search: query
    }
}, function(err, result) {
```

ここでは、 `$text`および`$search` MongoDBをしてコレクション`collectionName`の、されたクエリからなくとも1つのをむすべてのドキュメントをします。

これをしてデ―タをするには、ブラウザののURLにアクセスしてください

```
http://localhost:8080/find/<query>
```

`<query>`はクエリです。

```
http://localhost:8080/find/JavaScript
```

```
[{
    _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
}]
```

モデルのインデックス。

MongoDBはインデックスをサポートしています。 Mongooseでは、これらのインデックスをスキーマでします。インデックスをするがあるは、スキーマレベルでインデックスをするがあります。

マングースコネクション

```
var strConnection = 'mongodb://localhost:27017/dbName';
var db = mongoose.createConnection(strConnection)
```

なスキーマの

```
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;
```

デフォルトでは、mongooseはモデルにされていない2つのしいフィールドをモデルにします。これらのフィールドはのとおりです。

_id

Mongooseは、Schemaコンストラクタにされない、スキーマにデフォルトで_idフィールドをりてます。りてられるは、MongoDBのデフォルトとするObjectIdです。スキーマに_idをしたくないは、このオプションをしてスキームをにすることができます。

```
var usersSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true
    }, {
        _id: false
```

```
});
```

__vまたはversionKey

versionKeyは、Mongooseがにしたときににされるプロパティです。このキーには、ドキュメントのリビジョンがまれます。このドキュメントプロパティのはです。

モデルでこのフィールドをににすることができます

```
var usersSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true
    }, {
    versionKey: false
});
```

インデックス

Mongooseがするにも、のをできます。

```
usersSchema.index({username: 1 });
usersSchema.index({email: 1 });
```

このような、たちのモデルには2つのインデックスがあります.1つはフィールドユーザーで、もう1つはメールフィールドです。しかし、インデックスをすることはです。

```
usersSchema.index({username: 1, email: 1 });
```

パフォーマンスのをインデックスする

デフォルトでは、すべてのensureIndexびしがしたとき、またはエラーがあったときに、にインデックスのensureIndexをにびして、モデルの 'index'イベントをします。

MongoDBでは、ensureIndexはバージョン3.0.0でされましたが、はcreateIndexのエイリアスです。

スキーマのautoIndexオプションをfalseにするか、config.autoIndexオプションをfalseにしてのグローバルにすることで、このをにすることをおめします。

```
usersSchema.set('autoIndex', false);
```

なモンゴース

Mongooseには`find()`でされたみみがいくつかまれています。

```
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
});
```

をってmongodbでデータをつける

セットアップ

まず、のパッケージをインストールします。

```
npm install express cors mongoose
```

コード

に、 server.jsにをし、データベーススキーマとコレクションのをし、Express.jsサーバーをし、MongoDBにします。

```
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!');
});
```

に、データのクエリにするExpress.jsルートをします。

```
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
})
```

のドキュメントがモデルのコレクションにまれているとします。

```
{
        "_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
}
```

そして、は、 `"request"`キーのに`"JavaScript is Awesome"`をむすべてのをつけてすることです。

このためには、MongoDBをし、 `server.js`という`node server.js`をし`node server.js`。

これをしてデータをするには、ブラウザののURLにアクセスしてください

```
http://localhost:8080/find/<query>
```

`<query>`はクエリです。

```
http://localhost:8080/find/JavaScript%20is%20Awesome
```

```
[{
    _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
}]
```

オンラインでモンゴースをむ https://riptutorial.com/ja/node-js/topic/3486/モンゴース

92: モンゴブにする

き

MongoDBは、フリーでオープンソースのクロスプラットフォームのドキュメントデータベースプログラムです。 NoSQLデータベースプログラムとしてされているMongoDBは、スキーマでJSONライクなをします。

はhttps://www.mongodb.com/をください。

- MongoClient.connect 'mongodb//127.0.0.127017 / crud'、functionerr、db{//ここにかする}};

Examples

Node.JSからmongoDBをするな

```
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はDBです。データベースにしないは、このびしでにされます。

mongoDBとコアNode.JSをするな

```
  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.");
    });
```

オンラインでモンゴブにするをむ https://riptutorial.com/ja/node-js/topic/6280/モンゴブにする

93: モンゴブ

- db。 コレクション .insertOne *document* 、 *optionsw*、wtimeout、j、serializeFuntions、forceServerObjectId、bypassDocumentValidation 、 コールバック
- db。 コレクション .insertMany *[documents]* 、 *optionsw*、wtimeout、j、serializeFuntions、forceServerObjectId、bypassDocumentValidation 、 コールバック
- db。 コレクション .find クエリ
- db。 コレクション .updateOne フィルタ 、 、 オプション*upsert*、w、wtimeout、j、bypassDocumentValidation 、 コールバック
- db。 コレクション .updateMany フィルタ 、 、 オプション*upsert*、w、wtimeout、j 、 コールバック
- db。 コレクション .deleteOne フィルタ 、 オプション*upsert*、w、wtimeout、j 、 コールバック
- db。 コレクション .deleteMany フィルタ 、 オプション*upsert*、w、wtimeout、j 、 コールバック

パラメーター

パラメータ	
	ドキュメントをすjavascriptオブジェクト
	ドキュメントの
クエリ	クエリをするオブジェクト
フィルタ	クエリをするオブジェクト
りし	がしたときにびされる
オプション	オプションオプションデフォルト*null*
w	オプションきみの
wtimeout	オプションきみのタイムアウトがします。 デフォルト*null*
j	オプションジャーナルみをするデフォルト*false*
アップサート	オプションのデフォルト*false*
マルチ	オプション 1つまたはすべてのドキュメントをするデフォルト*false*
serializeFunctions	オプションのオブジェクトのをシリアルするデフォルト*false*

パラメータ	
forceServerObjectId	オプションサーバにドライバのわりに_idをりてるようにするデフォルト*false*
bypassDocumentValidation	オプション MongoDB 3.2でドライバがスキーマをバイパスできるようにするデフォルト*false*

Examples

MongoDBにする

MongoDBにし、 'Connected'をします。をじます。

```
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 メソッド `Connect()`

MongoClient.connect *URL* 、 オプション 、 コールバック

	タイプ	
`url`		サーバのIP /ホスト、ポート、およびデータベースをする
`options`	オブジェクト	オプションオプションデフォルト*null*
`callback`		がしたときにびされる

`callback`は2つのをとります

- `err` エラー - エラーがした、 `err`がされます。
- `db` object - MongoDBインスタンス

をする

'myFirstDocument'というドキュメントをし、 **2**つのプロパティ、 `greetings`と`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
  });
});
```

コレクションメソッド`insertOne()`

db.collection コレクション .insertOne ドキュメント 、 オプション 、 コールバック

	タイプ	
`collection`		コレクションをする
`document`	オブジェクト	コレクションにされるドキュメント
`options`	オブジェクト	オプションオプションデフォルト*null*
`callback`		がしたときにびされる

`callback`は2つのをとります

- `err` エラー - エラーがした、 `err`がされます。
- `result` object - のをむオブジェクト

コレクションをむ

コレクション 'myCollection'のすべてのドキュメントをし、コンソールにします。

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

コレクションメソッド`find()`

db.collection *collection* .find

	タイプ	
`collection`		コレクションをする

ドキュメントをする

`{ greetings: 'Hellu' }`プロパティをつをつけて`{ greetings: 'Whut?' }`し`{ 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
        });
});
```

コレクションメソッド`updateOne()`

db.collection *collection* .updateOne フィルタ 、 、 オプション 、 コールバック

パラメータ	タイプ	
`filter`	オブジェクト	をします。
`update`	オブジェクト	するをします。
`options`	オブジェクト	オプションオプションデフォルト*null*
`callback`		がしたときにびされる

`callback`は2つのをとります

- `err` エラー - エラーがした、errがされます。
- `db` object - MongoDBインスタンス

ドキュメントをする

`{ greetings: 'Whut?' }`プロパティをつをします`{ 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
    });
});
```

コレクションメソッド`deleteOne()`

db.collection *collection* .deleteOne フィルタ 、 オプション 、 コールバック

パラメータ	タイプ	
`filter`	オブジェクト	をする
`options`	オブジェクト	オプションオプションデフォルト*null*
`callback`		がしたときにびされる

`callback`は2つのをとります

- `err` エラー - エラーがした、errがされます。
- `db` object - MongoDBインスタンス

のをする

「おれ」プロパティが「」にされているすべてのドキュメントをします。

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

コレクションメソッド`deleteMany()`

db.collection コレクション .deleteMany フィルタ 、 オプション 、 コールバック

パラメータ	タイプ	
`filter`		をする
`options`	オブジェクト	オプションオプションデフォルト*null*
`callback`		がしたときにびされる

`callback`は2つのをとります

- `err` エラー - エラーがした、errがされます。
- `db` object - MongoDBインスタンス

シンプルコネクト

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

シンプルな、を

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

オンラインでモンゴブをむ https://riptutorial.com/ja/node-js/topic/5002/モンゴブ
```

94: ヤーンパッケージマネージャー

き

Yarnはnpmとに Node.jsのパッケージマネージャです。 をしながら、とnpmにはいくつかのないがあります。

Examples

のインストール

このでは、OSにYarnをインストールするさまざまなについてします。

マックOS

```
brew update
brew install yarn
```

MacPorts

```
sudo port install yarn
```

あなたのPATHにをする

シェルプロファイル `.profile` 、 `.bashrc` 、 `.zshrc`などにをします。

```
export PATH="$PATH:`yarn global bin`"
```

Windows

インストーラ

まず、Node.jsがまだインストールされていないはインストールします。

Yarn Webサイトからインストーラを`.msi`としてダウンロードします 。

チョコレート

```
choco install yarn
```

Linux

Debian / Ubuntu

あなたのディストリビューションにNode.jsがインストールされていることをするか、をしてください

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

YarnPkgリポジトリをする

```
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
```

をインストールする

```
sudo apt-get update && sudo apt-get install yarn
```

CentOS / Fedora / RHEL

まだインストールされていないはNode.jsをインストールしてください

```
curl --silent --location https://rpm.nodesource.com/setup_6.x | bash -
```

をインストールする

```
sudo wget https://dl.yarnpkg.com/rpm/yarn.repo -O /etc/yum.repos.d/yarn.repo
sudo yum install yarn
```

アーチ

ヤーンをAURでりけます。

yaourtをった

```
yaourt -S yarn
```

Solus

```
sudo eopkg install yarn
```

すべてのディストリビューション

シェルプロファイル `.profile`、 `.bashrc`、 `.zshrc`などにをします。

```
export PATH="$PATH:`yarn global bin`"
```

のインストール

シェルスクリプト

```
curl -o- -L https://yarnpkg.com/install.sh | bash
```

インストールするバージョンをする

```
curl -o- -L https://yarnpkg.com/install.sh | bash -s -- --version [version]
```

タールボール

```
cd /opt
wget https://yarnpkg.com/latest.tar.gz
tar zvxf latest.tar.gz
```

Npm

すでにnpmがインストールされているは、にしてください

```
npm install -g yarn
```

ポストインストール

インストールされているバージョンのYarnをしてする

```
yarn --version
```

パッケージの

`yarn init`コマンドは、パッケージにするいくつかのをする`package.json`ファイルのをします。これは、 `npm init`コマンドにてい`npm init`。

パッケージをするしいディレクトリをしてナビゲートし、に`yarn init`する

```
mkdir my-package && cd my-package
yarn init
```

CLIでのにえてください

```
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.
```

これは、のような`package.json`ファイルをします

```
{
  "name": "my-package",
  "version": "1.0.0",
  "description": "A test package",
  "main": "index.js",
  "author": "StackOverflow Documentation",
  "license": "MIT"
}
```

ここでをしてみましょう。これのためのなは`yarn add [package-name]`

のコマンドをしてExpressJSをインストールします

```
yarn add express
```

これにより、 `dependencies`セクションが`package.json`にされ、ExpressJSがされます

```
"dependencies": {
    "express": "^4.15.2"
}
```

をってパッケージをインストールする

はnpmとじレジストリをします。つまり、npmでなすべてのパッケージはYarnでじです。

パッケージをインストールするには、 `yarn add package`し`yarn add package`。

パッケージののバージョンがなは、 `yarn add package@version`して`yarn add package@version`ことができます。

インストールするがあるバージョンにタグがいているは、 `yarn add package@tagyarn add package@tag`。

オンラインでヤーンパッケージマネージャーをむ https://riptutorial.com/ja/node-js/topic/9441/ヤーンパッケージマネージャー

95: ユニットテストフレームワーク

Examples

モカ

```
describe('Suite Name', function() {
  describe('#method()', function() {
    it('should run without an error', function() {
      expect([ 1, 2, 3 ].length).to.be.equal(3)
    })
  })
})
```

モカコールバック

```
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()
      })
    })
  })
})
```

モカPromise

```
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')
      })
    })
  })
})
```

モカ/

```
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)
    })
```

```
  })
})
```

オンラインでユニットテストフレームワークをむ https://riptutorial.com/ja/node-js/topic/6731/ユニットテストフレームワーク

96: ライブラリをせずにSTDINとSTDOUTのNode.jsコード

き

これはnode.jsのなプログラムで、ユーザーからのをけり、コンソールにします。

プロセスオブジェクトは、のNode.jsプロセスにするをし、そのプロセスをするグローバルです。グローバルなので、requireをわずにNode.jsアプリケーションがにできます。

Examples

プログラム

process.stdinプロパティは、stdinにする、またはstdinにけられたReadableストリームをします。

process.stdoutプロパティは、stdoutにする、またはstdoutにけられたWritableストリームをします。

```
process.stdin.resume()
console.log('Enter the data to be displayed ');
process.stdin.on('data', function(data) { process.stdout.write(data) })
```

オンラインでライブラリをせずにSTDINとSTDOUTのNode.jsコードをむ
https://riptutorial.com/ja/node-js/topic/8961/ライブラリをせずにstdinとstdoutのnode-jsコード

97: ループバック - RESTベースのコネクタ

き

レストベースのコネクタとそれにする。たちは、LoopbackはRESTベースのにさをしないことをっています

Examples

Webベースのコネクタの

```
//この□では、iTunesからの□□を□□します
{
 "□り"□ {
    "□□"□ "□□"、
  "コネクタ"□ "□□"、
  "デバッグ"□□、
  "options"□{
      "useQuerystring"□true、
   "タイムアウト"□10000、
    "headers"□{
         "accepts"□ "application / json"、
     "content-type"□ "application / json"
       }
     }、
  "オペレーション"□ [
       {
         "テンプレート"□ {
           "メソッド"□ "GET"、
     "url"□ "https://itunes.apple.com/search"、
     "query"□{
             「□□」□「{キーワード}」、
      "country"□ "{country = IN}"、
      "media"□ "{itemType = music}"、
      "limit"□ "{limit = 10}"、
      "explicit"□ "false"
          }
        }、
    "□□"□ {
          "□□"□[
            "キーワード"、
      "□"、
      "itemType"、
      "□□"
          ]
        }
      },
   {
    "テンプレート"□ {
          "メソッド"□ "GET"、
     "url"□ "https://itunes.apple.com/lookup"、
     "query"□{
             "やった}"
     }
```

```
    },
    "□□"□ {
            "findById"□[
              "id"
            ]
          }
        }
      ]
    }
}
```

オンラインでループバック - RESTベースのコネクタをむ https://riptutorial.com/ja/node-js/topic/9234/ループバック----restベースのコネクタ

98: ロダシュ

き

LodashはなJavaScriptユーティリティライブラリです。

Examples

コレクションをフィルタリングする

のコードスニペットは、lodashをしてオブジェクトのをフィルタリングするさまざまなをしています。

```
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"}];
```

オンラインでロダシュをむ https://riptutorial.com/ja/node-js/topic/9161/ロダシュ

99:

Examples

Node.Jsでをする

Node.jsには、/エラーをする3つのながあります。

1. **try** - **catch**ブロック
2. `callback`へののとしてのエラー
3. `emit`つEventEmitterをしてエラーイベントを

try-catchは、コードのからスローされたをするためにされます。びしまたはびしのびしなどがtry / catchをした、エラーをキャッチできます。びしのどれもtry-catchをっていない、プログラムがクラッシュします。
でtry-catchをしていて、asyncメソッドのコールバックからtry-catchでされないがスローされた。コールバックからをキャッチするには、 をすることをおめします。
それをよりよくする

```
// ** 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
```

コールバックは、イベントをにするため、Node.jsでにされます。ユーザーはコールバックをし、がしたときにいつかびすことができます。
のパターンは、コールバックがコールバック*err*、resultとしてびされ、がしたかしたかにじて、errおよびresultのいずれかのみがnullでないことです。

```
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よりなケースでは、コールバックをするわりに、がEventEmitterオブジェクトをすことができ、びしはエミッタのエラーイベントをリスンすることがされます。

```
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
});
```

ての

Node.jsはのプロセスでされるため、キャッチされないはアプリケーションのにするがあります。

かにをする

の々は、node.jsサ―バ―がエラ―をかにみませるようにしています。

- をかにする

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

これはいですが、それはしますが、

- なはのままであるため、エラ―をきこしたのにはちません。
- らかのでデ―タベ―スプ―ルがクロ―ズした、エラ―がにします。つまり、サ―バ―はですが、dbにしません。

にる

"uncaughtException"のは、サ―バをしてにすといでしょう。そこではすることがわかっています。はログにされ、アプリケ―ションはしますが、サ―バ―がしていることをするコンテナ―でされるため、サ―バ―ののへのがわれます。

- またはのCLIツ―ルをインスト―ルして、ノ―ドサ―バ―がにしていることをする

```
npm install forever -g
```

- にサ―バ―をする

```
forever start app.js
```

  はなぜそれがまり、たちがにするは、サ―バ―がにしたにプロセスがサ―バ―をびするということです。

- サ―バ―の

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

    // some logging mechanisam
    // ....

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

また、 クラスタとドメインでをするもありました。

ドメインはここではされています 。

エラーと

プロミスは、またはコールバックのコードとはなるでエラーをします。

```
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!
    });
```

では、のにげまれたエラーがされず、エラーがみまれ、エラーをすることがになるがあります。これは、 eslintのようなlintingツールをうか、に`catch`をつようにすることでできます。

このは、ノードプロセスをするために、ノード8でされなくなりました 。

オンラインでをむ https://riptutorial.com/ja/node-js/topic/2819/

100:

Examples

をする

1. なプロセス
2. デカップリング
3. テストライティング

なプロセス
ノードをする、はDIのにコードがなく、すべてのモジュールをにできるため、プロセスをにうことができます。

デカップリング
モジュールのがなくなり、メンテナンスがになります。

テストライティング
ハードコードされたは、それらをモジュールにして、モジュールのテストをくのがです。

オンラインでをむ https://riptutorial.com/ja/node-js/topic/7681/

101: とノードのプログラミング

Examples

asyncの

asyncパッケージは、コードのをします。

autoをすると、2つののをできます。

```
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);
});
```

このコードは、 `get_data` 、 `make_folder` 、 `write_file` 、および`email_link`をしいでびすだけで、にわれているがあります。 Asyncはをし、エラーがした `null` しくない`callback`のパラメータ、ののをします。

オンラインでとノードのプログラミングをむ https://riptutorial.com/ja/node-js/topic/8287/とノードのプログラミング

102: のオートリロード

Examples

nodemonをしたソースコードの

nodemonパッケージをすると、ソースコードのファイルをしたときににプログラムをリロードすることができます。

グローバルにnodemonをインストールする

`npm install -g nodemon` または`npm i -g nodemon`

ローカルにnodemonをインストールする

グローバルにインストールしたくない

`npm install --save-dev nodemon` または`npm i -D nodemon`

ノーデモンの

`nodemon entry.js` または`nodemon entry` でプログラムをする

これは、、 `node entry.js` または`node entry` のをきえ`node entry`。

nodemonのをnpmスクリプトとしてすることもできます。これは、パラメータをしてしないにです。

package.jsonを

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

こので、あなたはコンソールから`npm start`をうことができます。

Browsersync

Browsersyncは、ライブファイルのとブラウザのみみをにするツールです。それはNPMパッケージとしてです。

インストール

Browsersyncをインストールするには、まずNode.jsとNPMをインストールするがあります。は、Node.jsのインストールとにするSOのマニュアルをしてください。

プロジェクトをしたら、のコマンドでBrowsersyncをインストールすることができます

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

これにより、ローカルの`node_modules`ディレクトリにBrowsersyncがインストールされ、のにされます。

グローバルにインストールするは、 `-D`フラグのわりに`-g`フラグをします。

Windowsユーザー

WindowsにBrowsersyncをインストールするにがあるは、Visual Studioをインストールして、ビルドツールにアクセスしてBrowsersyncをインストールするがあります。に、しているVisual Studioのバージョンをのようにするがあります。

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

このコマンドは、Visual Studioの2013バージョンをします。

な

プロジェクトのJavaScriptファイルをするたびににサイトをリロードするには、のコマンドをします。

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

`myproject.dev`を、プロジェクトにアクセスするためにしているWebアドレスにきえ`myproject.dev`。 Browsersyncは、プロキシでサイトにアクセスするためにできるアドレスをします。

な

のコマンドラインインターフェイスのに、BrowsersyncはGrunt.jsおよびGulp.jsでもできます。

Grunt.js

Grunt.jsでのには、のようにインストールできるプラグインがです

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

に、このを`gruntfile.js`します

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

Gulp.js

BrowsersyncはCommonJSモジュールとしてするので、Gulp.jsプラグインはありません。のようなモジュールがです。

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

これで、 Browsersync APIをしてニーズにわせてできます。

API

Browsersync APIは、 https : //browsersync.io/docs/apiにあります。

オンラインでのオートリロードをむ https://riptutorial.com/ja/node-js/topic/1743/のオートリロード

103: プロセスによるファイルまたはコマンドの

- child_process.execcommand [、options] [、callback]
- child_process.execFilefile [、args] [、options] [、callback]
- child_process.forkmodulePath [、args] [、options]
- child_process.spawncommand [、args] [、options]
- child_process.execFileSyncfile [、args] [、options]
- child_process.execSyncコマンド[、options]
- child_process.spawnSynccommand [、args] [、options]

プロセスをう、すべてのメソッドは`ChildProcess`インスタンスをしますが、すべてのバージョンはされたもののをします。 Node.jsののとに、エラーがすると、それはスローされます。

Examples

しいプロセスをしてコマンドをする

バッファ `child_process.spawn()`されていないをとするしいプロセスをするにはえば、するのではなくにってをし、すぐにするようなするプロセス、 `child_process.spawn()`します。

このメソッドは、されたコマンドとのをしてしいプロセスをします。りは、 `ChildProcess`インスタンスで、 `stdout`および`stderr`プロパティをします。これらのストリームのが`stream.Readable`インスタンスです。

のコードは、コマンド`ls -lh /usr`とじです。

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

のコマンド

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

のようにくことができます

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

シェルをしてコマンドをする

バッファされたがなつまり、ストリームではないシェルでコマンドをするには、`child_process.exec`し`child_process.exec`。 たとえば、 `cat *.js file | wc -l`をするは、オプションなしの`cat *.js file | wc -l`はのようになります。

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

このは3つのパラメータをけります。

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

コマンドパラメータはであり、ですが、オプションオブジェクトとコールバックはともオプションです。オプションオブジェクトがされていない、 `exec`はデフォルトとしてをします。

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

オプションオブジェクトは`shell`パラメータもサポートしています。これは、デフォルトではUNIXでは`/bin/sh`、Windowsでは`cmd.exe`、プロセスのユーザIDをするための`uid`オプション、グループIDの`gid`オプションです。

コマンドがされたときにびされるコールバックは、3つの`(err, stdout, stderr)`びされます。コマンドがにされた、 `err`は`null`になり`null`。そうでないは`Error`インスタンスになります`err.code`はプロセスのコードで、 `err.signal`はするためにされたシグナルです。

`stdout`および`stderr`は、コマンドのです。オプションオブジェクトデフォルト `string` でされたエンコーディングでデコードされますが、それのは`Buffer`オブジェクトとしてされます。

また、 `execSync`というバージョンの`exec`もします。バージョンはコールバックをとらず、`ChildProcess`インスタンスのわりに`stdout`をします。バージョンでエラーがすると 、プログラムがスローされてします。これはのようになります。

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

ファイルをするプロセスの

ファイルなどのファイルをするは、 `child_process.execFile`し`child_process.execFile`。`child_process.exec`ようなシェルをするのではなく、しいプロセスをします。これはコマンドをするよりもややです。はのようにうことができます

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

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

`child_process.exec`とはなり、このは4つのパラメータをけれます.2のパラメータは、ファイルにするのです。

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

そうでなければ、オプションとコールバックは`child_process.exec`とじです。のバージョンのもじです

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

オンラインでプロセスによるファイルまたはコマンドのをむ https://riptutorial.com/ja/node-js/topic/2726/プロセスによるファイルまたはコマンドの

104: でのNode.jsアプリケーションのデプロイ

Examples

NODE_ENV = "production"をする

NODE_ENVでのデプロイメントはさまざまですが、 NODE_ENVにデプロイするのなは、 NODE_ENVというをし、そのを*"production"*にすることです。

ランタイムフラグ

アプリケーションでされているコードモジュールをむは、 NODE_ENVのをチェックできます。

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

NODE_ENVが*'production'*にされている、 *package.json*ファイルのすべてのdevDependenciesは、 npm installするとにされます。 --productionフラグでこれをすることもできます

```
npm install --production
```

NODE_ENVをするには、これらののいずれかをできます

1すべてのードのアプリケーションに**NODE_ENV**をする

Windows

```
set NODE_ENV=production
```

LinuxやそののUNIXベースのシステム

```
export NODE_ENV=production
```

これは、NODE_ENVこのがありますので、にのアプリケーションがし、のbashのセッションにNODE_ENVにしproduction。

2のアプリに**NODE_ENV**をする

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

これにより、のアプリだけに`NODE_ENV`がされます。これは、さまざまなでアプリをテストするときにちます。

3 `.env`ファイルをしてする

これはここでされているえをします 。よりしいについては、このをしてください。

に`.env`ファイルをし、にするbashスクリプトをいくつかします。

bashスクリプトのをけるため、 env-cmdパッケージをして、 `.env`ファイルでされたをみむことができます。

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

4 `cross-env`パッケージをする

このパッケージをすると、プラットフォームごとにをできます。

npmでインストールした、のように`package.json`のデプロイメントスクリプトにすることができます

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

プロセスマネージャーでアプリをする

プロセスマネージャーによってされるNodeJSアプリケーションをすることはいです。プロセスマネージャーは、アプリケーションをにかし、にし、ダウンタイムなしでロードし、をします。それらのでもなもの PM2のようなものにはロードバランサがみまれています。また、PM2をすると、アプリケーションのロギング、、およびクラスタリングをできます。

PM2プロセスマネージャ

PM2のインストール

```
npm install pm2 -g
```

ロードバランサがされたクラスタモードでプロセスをすると、プロセスでをできます。

`pm2 start app.js -i 0 --name "api"` - *i*は、するプロセスのをするためのもので、0のはプロセスはCPUのコアにづいています

ではのユーザーがいるが、PM2にはのポイントがである。したがって、pm2コマンドにはプレフィックスをけるがありますPM2 configの。それのは、それぞれのホームディレクトリにconfigをつすべてのユーザーにしてしいpm2プロセスをします。そしてそれはするでしょう。

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

PM2をした

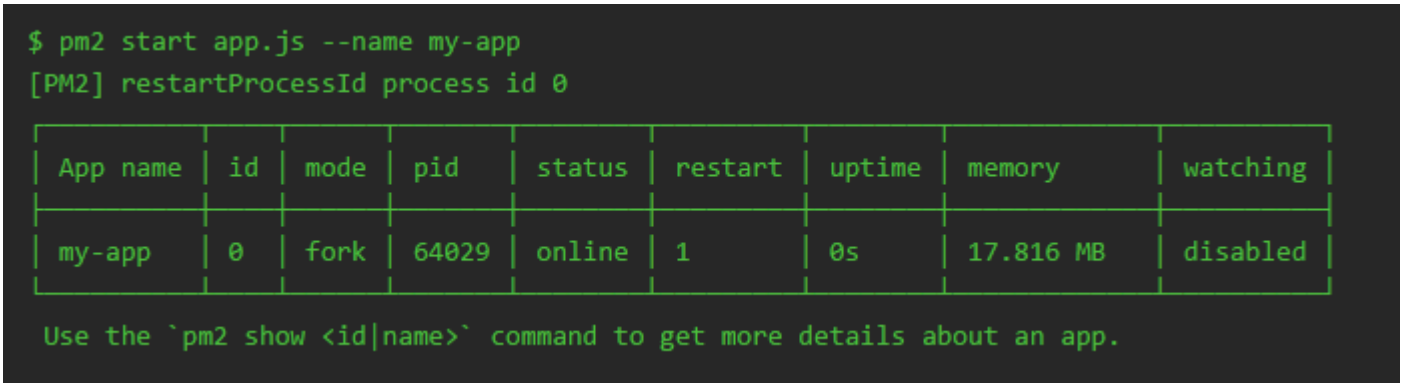
PM2は、 Node.jsアプリケーションのプロダクションプロセスマネージャです。アプリケーションをにかし、ダウンタイムなしでリロードすることができます。また、PM2をすると、アプリケーションのロギング、、およびクラスタリングをできます。

pm2グローバルにインストールします。

```
npm install -g pm2
```

に、 PM2.をしてnode.jsアプリケーションをしPM2.

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



のコマンドは、 PM2するにです。

のプロセスをすべてする

```
pm2 list
```

アプリをする

```
pm2 stop my-app
```

アプリをする

```
pm2 restart my-app
```

アプリにするをするには

```
pm2 show my-app
```

PM2のレジストリからアプリをするには

```
pm2 delete my-app
```

プロセスマネ―ジャをした

プロセスマネ―ジャは、、ノ―ドjアプリケ―ションをするためにでされます。プロセスマネ―ジャのなは、クラッシュしたにサ―バをし、リソ―スのをチェックし、のパフォ―マンスをし、するなどです。

ノ―ド･コミュニティ―によってられたなプロセス･マネ―ジャ―のは、に、pm2などです。

Forvever

`forever`は、されたスクリプトがにされるようにするためのコマンドラインインタ―フェイスツ―ルです。 `forever`のシンプルなインタ―フェイスは、 `Node.js`アプリケ―ションやスクリプトのなをするのにです。

`forever`プロセスをし、クラッシュしたはプロセスをします。

に`forever`にインスト―ルしてください。

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

アプリケ―ションをする

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

これはサ―バをし、プロセスのIDをえる0からまる。

アプリケ―ションをする

```
$ forever restart 0
```

ここで、 `0`はサ―バ―のIDです。

アプリケ―ションを

```
$ forever stop 0
```

restartとに、 `0`はサ―バ―のIDです。にえられたidのわりにプロセスIDやスクリプトをえることもできます。

そののコマンド https : //www.npmjs.com/package/forever

dev、qa、stagingなどのなるになるプロパティ/をする

なアプリケ―ションでは、なるでさせるときになるプロパティがになることがよくあります。NodeJsアプリケ―ションにをし、のプロパティファイルをロ―ドするためにノ―ドプロセスでじ

をすることで、これをできます。

なるに2つのプロパティファイルがあるとします。

- 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"
    }
}
```

アプリケーションののコードは、するそれぞれのプロパティファイルをエクスポートします。

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

々は、のようにアプリケーションにをえます

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

たちはじくらいとしてにそれのようなプロセス･マネージャをしている、

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

クラスタをする

Node.jsののインスタンスは、のスレッドでされます。マルチコアシステムをするには、をするNode.jsプロセスのクラスタをするがあることがあります。

```
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');

       }
```

オンラインでのNode.jsアプリケーションのデプロイをむ https://riptutorial.com/ja/node-js/topic/2975/でのnode-jsアプリケーションのデプロイ

105: なAPIデザインベストプラクティス

Examples

エラーすべてのリソースをする

どのようにエラーをし、それをコンソールにするのですか

った

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

もっとい

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

オンラインでなAPIデザインベストプラクティスをむ https://riptutorial.com/ja/node-js/topic/6490/なapiデザイン-ベストプラクティス

106:

Examples

へのアクセス

`process.env`プロパティは、ユーザーをむオブジェクトをします。

のようなオブジェクトをします。

```
{
  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'
```

`FOO`を`foobar`にすると、のようにアクセスできます。

```
process.env.FOO // 'foobar'
```

process.argvコマンドライン

process.argvは、コマンドラインをむです。のは`node`、2のはJavaScriptファイルのになります。のは、のコマンドラインです。

コード

すべてのコマンドラインの

`index.js`

```
var sum = 0;
for (i = 2; i < process.argv.length; i++) {
    sum += Number(process.argv[i]);
}

console.log(sum);
```

の

```
node index.js 2 5 6 7
```

は20なります

コードのな

ここでforループ`for (i = 2; i < process.argv.length; i++)`ループは2でまり`for (i = 2; i < process.argv.length; i++)`なぜなら、process.argvのの2つのはに `['path/to/node.exe', 'path/to/js/file', ...]`

process.argvのはに stringであるため、 `Number(process.argv[i])`する

dev、qa、stagingなどのなるになるプロパティ/をする

なアプリケーションでは、なるでさせるときになるプロパティがになることがよくあります。NodeJsアプリケーションにをし、のプロパティファイルをロードするためにノードプロセスでじをすることで、これをできます。

なるに2つのプロパティファイルがあるとします。

- 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"
        }
}
```

アプリケーションののコードは、するそれぞれのプロパティファイルをエクスポートします。

コードがenvironment.jsにあるとします

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

々は、のようにアプリケーションにをえます

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

たちはじくらいとしてにそれのようなプロセス･マネージャをしている、

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

ファイルのい

```
 var env= require("environment.js");
```

「プロパティファイル」からプロパティをロードする

- プロパティリーダーをインストールする

```
npm install properties-reader --save
```

- プロパティファイルをするディレクトリ**env**をします。

```
mkdir env
```

- **environments.js**をする

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

- サンプル**development.properties**プロパティーファイル

```
# Dev properties
[main]
# Application port to run the node server
app.port=8080
```

```
[database]
# Database connection to mysql
mysql.host=localhost
mysql.port=2500
...
```

- ロードされたプロパティの

```
var enviorment = require('./environments');
var PropertiesReader = require('properties-reader');
var properties = new PropertiesReader(enviorment);

var someVal = properties.get('main.app.port');
```

- サーバーの

```
npm start env=development
```

または

```
npm start env=production
```

オンラインでをむ https://riptutorial.com/ja/node-js/topic/2340/

107: するコールバック

Examples

コールバックをする

コールバックベース

```
db.notification.email.find({subject: 'promisify callback'}, (error, result) => {
   if (error) {
       console.log(error);
   }

   // normal code here
});
```

これはbluebirdのpromisifyAllメソッドをして、のコールバックベースのコードがかをします。bluebirdはオブジェクトのすべてのメソッドをしますが、これらのにづくメソッドにはAsyncがされています

```
let email = bluebird.promisifyAll(db.notification.email);

email.findAsync({subject: 'promisify callback'}).then(result => {

    // normal code here
})
.catch(console.error);
```

のだけがされるがあるは、そのをしてください

```
let find = bluebird.promisify(db.notification.email.find);

find({locationId: 168}).then(result => {

    // normal code here
});
.catch(console.error);
```

メソッドののオブジェクトが2のパラメータにされないと、することができないライブラリMassiveJSなどがあります。その、2のパラメータでpromisifiedするがあるメソッドののオブジェクトをし、コンテキストプロパティでそれをみます。

```
let find = bluebird.promisify(db.notification.email.find, { context: db.notification.email });

find({locationId: 168}).then(result => {

    // normal code here
});
.catch(console.error);
```

でコールバックをする

でコールバックをするがあることがあります。これは、コールバックがのエラーファーストにわない、またはするためにのロジックがなです。

fs.existspath、callbackの

```
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);
});
```

されたsetTimeout

```
function wait(ms) {
    return new Promise(function (resolve, reject) {
        setTimeout(resolve, ms)
    })
}
```

オンラインでするコールバックをむ https://riptutorial.com/ja/node-js/topic/2346/するコールバック

108: いコーディングスタイル

はこのスタイルのコーディングからにおめします。もしかがよりいをできればはこのテクニックをび、100kをえるユーザがっているアプリでにいています、おにごください。 TIA。

Examples

サインアップのためのプログラム

このでは、undertandibilityをさせるために**node.js**コードをなるモジュール/フォルダにするについてします。このをすると、コードをするのではなく、するファイルをできるため、のがコードをしやすくなります。なは、あなたがチームでいているときに、しいがでするときに、コードでになるようになることです。

index.js - このファイルはサーバーをします。

```
//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 - このファイルはすべてのにするパラメータをします。

```
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 - スキーマがされているモデルファイル

```
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 - このファイルには、ユーザーsignUpのがまれています

```
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 - これはuserControllerのルートです

```
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;
```

のはあまりにもきくえるかもしれませんが、node.jsのがなをしでもしてみると、これはでにつでしょう。

オンラインでいコーディングスタイルをむ https://riptutorial.com/ja/node-js/topic/6489/いコーディングスタイル

109: みまれた

- const readline = require 'readline'
- readline.close
- readline.pause
- readline.prompt[preserveCursor]
- readline.questionクエリ、コールバック
- readline.resume
- readline.setPromptプロンプト
- readline.writedata [、key]
- readline.clearLinestream、dir
- readline.clearScreenDownstream
- readline.createInterfaceoptions
- readline.cursorTostream、x、y
- readline.emitKeypressEventsstream [、interface]
- readline.moveCursorstream、dx、dy

Examples

のファイルみり

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

CLIをしたユーザーのプロンプト

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

オンラインでみまれたをむ https://riptutorial.com/ja/node-js/topic/1431/みまれた

110: /

き

Async / awaitは、コールバック コールバック･ヘル やプロミス･チェーン `.then().then().then()` にることなく、コードをきにくことをにするキーワードのセットです。

これは、 `await`キーワードをし`await` 、のまでのをし、 `async`キーワードをしてをすようなをすることによってします。

Async / awaitはデフォルトでnode.js 8からです。または、 `--harmony-async-await`フラグをして7からできます。

Examples

Try-Catchエラーによる

async / awaitのもれたの1つは、コードをするのとじように、なtry-catchコーディングスタイルがであることです。

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

Expressとpromise-mysqlのをにします。

```
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
  }
});
```

とIの

をいた

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

したがって、Async / Awaitがにをたすためには、のようになります。

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

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

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

したがって、キーワード`async`は、 `return new Promise((resolve, reject) => {...}`ますとています。

そして`then`コールバックであなたのをる`await`て`await`ます。

ここではそれをたににっていないかなりなgifをす

GIF

コールバックからの

はコールバックがあり、コールバックはOKでした

```
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.
  })
})
```

しかし、コールバックにはにながいくつかあったので、たちはすべてをしめました。

```
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
```

これはちょっとかったです。に、async / awaitがつかりました。これはまだフードのでをしています。

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

ちをつ

がもされない、タスクはをしawaitすることができます。

```
try{
    await User.findByIdAndUpdate(user._id, {
        $push: {
            tokens: token
        }
    }).exec()
}catch(e){
    handleError(e)
}
```

オンラインで/をむ https://riptutorial.com/ja/node-js/topic/6729/-

111: プログラミング

き

Nodeは、すべてがにするプログラミングです。に、のなといくつかのをつけることができます。

- doSomething[args]、function[argsCB]{/ *にかする* /};
- doSomething[args]、[argsCB]=> {/ *にかする* /};

Examples

コールバック

JavaScriptのコールバック

JavaScriptではコールバックがです。 はファーストクラスのであるため、JavaScriptではコールバックができます。

コールバック。

コールバックは、またはにすることができます。コールバックはよりなので、ここではシンクロナイゼーションコールバックのなをします。

```
// 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");
```

のコードのはのとおりです。

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

まず、のコードがどのようにされるかについてします。これは、コールバックのをすでにしていないにとっては、このをスキップすることがであるということをすでにしているにとっては、よりくのことです。にコードがされ、に`Before getSyncMessage call`をコンソールに`Before`

`getSyncMessage call`6がされます。に、 `getSyncMessage`の`cb`というのパラメータのとして、`getSyncMessage`をでする8がされます。されたのは、されたばかりの`cb`をする3の`getSyncMessage`であり、このびしは、されたの`message`というのparamに "Hello World"をります。は9にみ、 `Hello World!`をログにし`Hello World!`コンソールにします。その、は、 びしスタックをするプロセス また 、 を10、に4、に11にします。

なコールバックについてるべき

- にコールバックとしてするは、0、1、またはびすことができます。それはすべてにします。
- コールバックは、またはにびすことができ、によっては、およびにびすことができる。
- のとに、にパラメータをすははではありませんが、はです。たとえば、8では、パラメータ `message statement` 、 `msg`という`statement`けることができました。あるいは、あなたが`jellybean`ようななものになっているとします。コールバックにどのパラメータがられるのかをっていて、なでしいでできるようにするがあります。

コールバック。

JavaScriptにしてすべきことは、デフォルトではですが、にできるAPIブラウザ、Node.jsなどがありますはこちら 。

コールバックをけれるJavaScriptでのなもの

- イベント
- setTimeout
- setInterval
- フェッチAPI
-

また、ののいずれかをするは、コールバックをとるでラップすることができます。ただし、コールバックはコールバックになりますただし、コールバックをとるでをラッピングすることは、にするよりましいがあります。

そのようなをえられれば、のようなをすることができます。

```
// 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");
```

これはコンソールにのものをします

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

のは、「getSyncMessageびしのに」6のログにします。その、は8にり、パラメータcbコールバックをしてgetAsyncMessageをびします。 3がされ、のとしてコールバック、2のとして300というをつsetTimeoutがびされます。 setTimeoutはそのコールバックにがあってもそのコールバックをするので、で1000ミリでびすことができますが、タイムアウトをしてから1000ミリをするに、をしたにして4にりますに11をしてから1ポーズし、setTimeoutはコールバックをびします。このコールバックは3にgetAsyncMessagesます。この、 getAsyncMessagesコールバックはパラメータmessage「Hello World」でびされ、9。

Node.jsのコールバック

NodeJSはコールバックをち、はerrとdataとばれることがあるに2つのパラメータをにします。ファイルのテキストをむ。

```
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
    }
});
```

これは、1とばれるコールバックのです。

それはあなたがちょうどそれをしたりげたりしてもらかのでエラーをするのがよいです。あなたがげたりったり、げたりったりするようなことをして、ifでののをめるり、インデントをらすためにすることができるelseはありません。

err 、 dataをるのがかもしれませんが、コールバックがそのパターンをすることはにerrとはりませんが、ドキュメントをるのがです。

もう1つのコールバックのは、Expressライブラリexpress 4.xにあります。

```
// 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);
```

このは、びされるコールバックをしています。コールバックには、 `req`と`res`というのparamsという2つのオブジェクトがされています。これらのはリクエストとレスポンスにそれぞれし、リクエストをしてユーザーにするレスポンスをするをします。

ごのとおり、JavaScriptでとのコードをするためにコールバックをするさまざまながあり、JavaScriptをじてコールバックはにしています。

コード

のコードのはですかなぜですか

```
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");
```

これはかにられています `EBAD`。 `C`はログにされるときはです。

コンパイラーは、 `setTimeout`および`getDataFromDatabase` `setTimeout`しません。がするのは`E`です。コールバック *`setTimeout`*のは、されたタイムアウトにでされます。

1. `E`は`setTimeout`がありません
2. `B`タイムアウトは0ミリです
3. `A`は1000ミリのタイムアウトがされています
4. `D`は`A`にるように`D` 1000ミリするがある、データベースをしなければなりません。
5. データベースのデータがされたときにはであるため、 `C`はです。それは`A`にあるがあります。

エラー

キャッチしよう

エラーはにするがあります。プログラミングをしているは、 `try catch`できます。しかし、これはでしてもしません

```
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);
}
```

エラーは、コールバックでのみされます。

v0.8

イベントハンドラ

Node.JSののバージョンには、イベントハンドラがあります。

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

v0.8

ドメイン

ドメインでは、エラーはイベントエミッタをしてされます。これをすることで、すべてのエラー、タイマー、にドメインにされたコールバックメソッドのみがされます。エラーがすると、エラーイベントがされ、アプリケーションがクラッシュしないようにします。

```
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);
});
```

コールバック

あまりにもくのコールバックをコールバックのにネストすると、コールバック･ヘルピラミッド･ドゥームまたはブーメラン･エフェクトもします。に、ファイルをみむをします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");
    }
});
```

「コールバック」をける

2つのコールバックをネストすることをします。これはコードのをするのにち、もがになります。2つのコールバックをネストするがあるは、わりにイベントをしてみてください。

asyncとばれるライブラリもあり、npmでコールバックとそのをするのにちます。コールバックコードのがし、コールバックコードフローをまたはでできるようにするなど、コールバックコードフローをよりにできます。

ネイティブ

v6.0.0

プロミスはプログラミングのツールです。 JavaScriptではがらのためにられ`then`。には、「」と「み」の2つのながあります。が「」されると、「」にることはできません。これは、はしかこらないにはいことをします。「み」には「み」と「」の2つのがあります。 `new`キーワードをしてしいをし、をコンストラクタ`new Promise(function (resolve, reject) {})`することができます。

Promiseコンストラクタにされたは、、それぞれ`resolve`と`reject`というの1と2のパラメータをにけります。これらの2つのパラメータのはなものですが、されたを「み」または「」のいずれかに

れます。これらのいずれかがびされると、は「」から「み」になります。 `resolve`は、であることがい、のアクションがされ、アクションがエラーをこしたに`reject`がされたときにびされます。

のタイムアウトにはPromiseをすがあります。

```
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...");
```

コンソール

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

timeoutがびされると、Promiseコンストラクタにされたはなくされます。にsetTimeoutメソッドがされ、そのコールバックがの`ms`ミリ、この`ms=1000`するようにされます。 setTimeoutへのコールバックはまだされていないので、timeoutはびしスコープにをします。 `then`メソッドのは、Promiseがされたときにでびされるようにされます。ここに`catch`メソッドがあれば、それもされますが、が「」されたには/がされます。

スクリプトは 'waiting ...'をします。 1にsetTimeoutはをびすコールバックを "It was resolved"というでびします。そのは、 `then`メソッドのコールバックにされ、ユーザーにされます。

じで、コールバックをとするのsetTimeoutをラップすることができます。これは、のアクションをでむことができます。

のについては、JavaScriptのドキュメントのをごください。

オンラインでプログラミングをむ https://riptutorial.com/ja/node-js/topic/8813/プログラミング

クレジット

S. No		Contributors
1	Node.jsをいめる	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	angular.jsをむNode.jsexpress.jsサンプルコード	sigfried
3	async.js	David Knipe, devnull69, DrakaSAN, F. Kauder, jerry, lsampaio, Shriganesh Kolhe, Sky, walid
4	Browserfiyをしてブラウザで「」エラーをする	Big Dude
5	CLI	Ze Rubeus
6	CORSをしたNode.js	Buzut
7	ES6をしたNode.JS	Inanc Gumus, xam, ymz, zurfyx
8	Express.JSでajaxリクエストをルーティングする	RamenChef, SynapseTech
9	ExpressJSのルートコントローラ - サー	nomanbinhussein

	ビス	
10	ExpressをしたWebアプリケーション	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
11	HTMLやそののファイルをする	Himani Agrawal, RamenChef, user2314737
12	http	Ahmed Metwally
13	IISNodeをしてIISでNode.js Webアプリケーションをホストする	peteb
14	Koa Framework v2	David Xu
15	MSSQLの	damitj07
16	MySQLのプール	KlwntSingh
17	MySQLの	Aminadav, Andrés Encarnación, Florian Hämmerle, Ivan Schwarz, jdrydn, JohnnyCoder, Kapil Vats, KlwntSingh, Marek Skiba, Rafael Gadotti Bachovas, RamenChef, Simplans, Sorangwala Abbasali, surjikal
18	N-API	Parham Alvani
19	Node.js / Express.jsのMongoDBインテグレーション	William Carron
20	Node.js v6と	creyD, DominicValenciana, KlwntSingh
21	Node.jsアーキテクチャと	Ivan Hristov
22	Node.jsアプリケーションのセキュリティ	akinjide, devnull69, Florian Hämmerle, John Slegers, Mukesh Sharma, Pauly Garcia, Peter G, pranspach, RamenChef, Simplans

23	Node.jsアプリケーションのデバッグ	4444, Alister Norris, Ankur Anand, H. Pauwelyn, Matthew Shanley
24	Node.jsエラー	Karlen
25	Node.jsデザインの	Ankur Anand, pietrovismara
26	Node.jsでのAPIの	Mukesh Sharma
27	Node.jsでのPOSTリクエストの	Manas Jayanth
28	Node.JSでのWebSocketの	Rowan Harley
29	node.jsでのWindows	CJ Harries
30	node.jsでのモジュールのエクスポートとインポート	AndrewLeonardi, Bharat, commonSenseCode, James Billingham, Oliver, sharif.io, Shog9
31	Node.JSとMongoDB。	midnightsyntax, RamenChef, Satyam S
32	Node.jsのアンインストール	John Vincent Jardin, RamenChef, snuggles08, Trevor Clarke
33	Node.jsのインストール	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
34	Node.jsのパフォーマンス	Florian Hämmerle, Inanc Gumus
35	Node.JSのリモートデバッグ	Rick, VooVoo
36	Node.jsの	vintproykt
37	node.jsをサービスとしてする	Buzut
38	Node.jsをむECMAScript 2015	David Xu, Florian Hämmerle, Osama Bari

	ES6	
39	nodeJsとのArduinoの	sBanda
40	Nodejsの	Kelum Senanayake
41	NodeJSフレームワーク	dthree
42	NodeJsルーティング	parlad neupane
43	NodeJSガイド	Niroshan Ranapathi
44	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
45	nvm - ノードバージョンマネージャ	cyanbeam, guleria, John Vincent Jardin, Luis González, pranspach, Shog9, Tushar Gupta
46	OAuth 2.0	tyehia
47	OracleとNode.js	oliolioli
48	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
49	PostgreSQLの	Niroshan Ranapathi
50	RedisとNodeJS	evalsocket
51	Require	Philip Cornelius Glover
52	Sequelize.js	Fikra, Niroshan Ranapathi, xam

53	Socket.ioコミュニケーション	Forivin, N.J.Dawson
54	TCPソケット	B Thuy
55	Webをする	Houssem Yahiaoui
56	イベントエミッタ	DrakaSAN, Duly Kinsky, Florian Hämmerle, jamescostian, MindlessRanger, Mothman
57	イベントループ	Kelum Senanayake
58	カサンドラ	Vsevolod Goloviznin
59	クライアントとサーバーの	Zoltán Schmidt
60	クライアントにファイルストリームをする	Beshoy Hanna
61	クラスタモジュール	Benjamin, Florian Hämmerle, Kid Binary, MayorMonty, Mukesh Sharma, riyadhalnur, Vsevolod Goloviznin
62	グレースフルシャットダウン	RamenChef, Sathish
63	コールバックをける	tyehia
64	コマンドラインの	yrtimiD
65	コンソールとの	ScientiaEtVeritas
66	シンプルなRESTベースのCRUD API	Iceman
67	ストリームの	cyanbeam, Duly Kinsky, efeder, johni, KlwntSingh, Max, Ze Rubeus
68	ダウンタイムなしでNode.jsアプリケーションをする。	gentlejo
69	つぶやく	Naeem Shaikh, Waterscroll
70	データベース MongoDB with Mongoose	zurfyx

71	テンプレートフレームワーク	Aikon Mogwai
72	ノードjsのcsvパーサー	aisflat439
73	ノードJSのローカリゼーション	Osama Bari
74	ノードアプリケーションをにさせる	Alex Logan, Bearington, cyanbeam, Himani Agrawal, Mikhail, mscdex, optimus, pietrovismara, RamenChef, Sameer Srivastava, somebody, Taylor Swanson
75	ノードのプロファイリングをする	damitj07
76	パスポート.js	Red
77	パスポートの	Ankit Rana, Community, Léo Martin, M. A. Cordeiro, Rupali Pemare, shikhar bansal
78	ハック	signal
79	パフォーマンスの	Antenka, SteveLacy
80	ファイルシステムI / O	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
81	ファイルのアップロード	Aikon Mogwai, Iceman, Mikhail, walid
82	プッシュ	Mario Rozic
83	ブルーバードの	David Xu
84	フレームワークのないノードサーバー	Hasan A Yousef, Taylor Ackley
85	プロジェクトの	damitj07
86	プロミスとエラーファーストコールバックのをサポートするNode.jsライブラリの	Dave

87	マルチスレッド	arcs
88	メタルミス	RamenChef, vsjn3290ckjnaoij2jikndckjb
89	モジュールのエクスポートと	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
90	モジュールのロード	RamenChef, umesh
91	モンゴース	Alex Logan, manuerumx, Mikhail, Naeem Shaikh, Qiong Wu, Simplans, Will
92	モンゴブにする	FabianCook, Nainesh Raval, Shriganesh Kolhe
93	モンゴブ	cyanbeam, FabianCook, midnightsyntax
94	ヤーンパッケージマネージャー	Andrew Brooke, skiilaa
95	ユニットテストフレームワーク	David Xu, Florian Hämmerle, skiilaa
96	ライブラリをせずにSTDINとSTDOUTのNode.jsコード	Syam Pradeep
97	ループバック - RESTベースのコネクタ	Roopesh
98	ロダシュ	M1kstur
99		KlwntSingh, Nivesh, riyadhalnur, sBanda, sjmarshy, topheman
100		Niroshan Ranapathi
101	とノードのプログラミング	Craig Ayre, Veger
102	のオートリロード	ch4nd4n, Dean Rather, Jonas S, Joshua Kleveter, Nivesh, Sanketh Katta, zurfyx
103	プロセスによるファイルまたはコマンドの	guleria, hexacyanide, iSkore
104	でのNode.jsアプリケ	Apidcloud, Brett Jackson, Community, Cristian Boariu,

	ーションのデプロイ	duncanhall, Florian Hämmerle, guleria, haykam, KlwntSingh, Mad Scientist, MatthieuLemoine, Mukesh Sharma, raghu, sjmarshy, tverdohleb, tyehia
105	なAPIデザインベストプラクティス	fresh5447, nilakantha singh deo
106		Chris, Freddie Coleman, KlwntSingh, Louis Barranqueiro, Mikhail, sBanda
107	するコールバック	Clement JACOB, Michael Buen, Sanketh Katta
108	いコーディングスタイル	Ajitej Kaushik, RamenChef
109	みまれた	4444, Craig Ayre, Florian Hämmerle, peteb
110	/	Cami Rodriguez, Cody G., cyanbeam, Dave, David Xu, Dom Vinyard, m_callens, Manuel, nomanbinhussein, Toni Villena
111	プログラミング	Ala Eddine JEBALI, cyanbeam, Florian Hämmerle, H. Pauwelyn, John, Marek Skiba, Native Coder, omgimanerd, slowdeath007