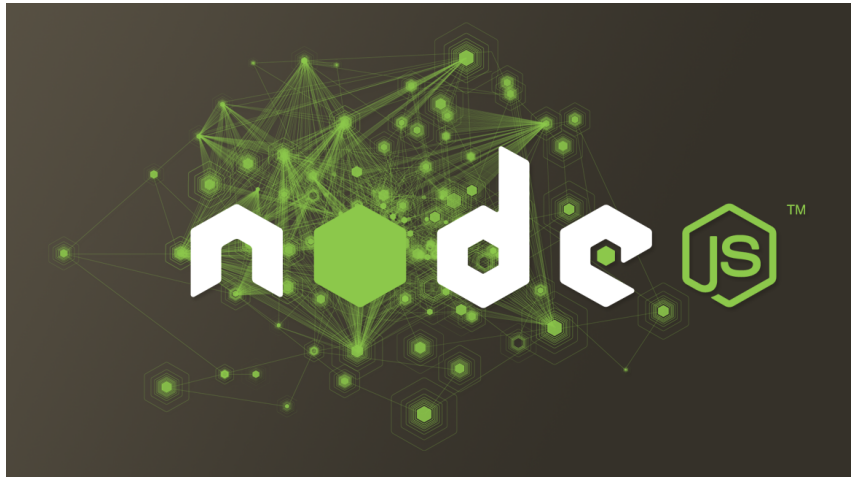


CS212 Webtechnologien: Node.js

Dominic Bosch

April 1, 2014

Introduction

Event-Driven Design

- Blocking & Non-Blocking I/O
- Asynchronous Events
- Time Awareness

Node.js Apps are the glue between Modules

- Built-in
- Node Packet Manager
- Your Project Modules

- Software Platform

- Software Platform
- Applications written in JavaScript

- Software Platform
- Applications written in JavaScript
- "Server-side"

- Software Platform
- Applications written in JavaScript
- "Server-side"
- Scalable Networking

- Software Platform
- Applications written in JavaScript
- "Server-side"
- Scalable Networking
- Asynchronous events, Non-blocking I/O

- Software Platform
- Applications written in JavaScript
- "Server-side"
- Scalable Networking
- Asynchronous events, Non-blocking I/O
- Built-in & Package Manager Modules

Blocking & Non-Blocking I/O Examples:

Blocking I/O:

```
var fs = require( 'fs' );
try {
  var data = fs.readFileSync( './cryptico.js', 'utf8' );
} catch ( err ) { /* Error Handler */ }
```

Blocking & Non-Blocking I/O Examples:

Blocking I/O:

```
var fs = require( 'fs' );
try {
  var data = fs.readFileSync( './cryptico.js', 'utf8' );
} catch ( err ) { /* Error Handler */ }
```

Non-Blocking I/O:

```
var fs = require( 'fs' );
var fAsyncCalled = function( err, data ) {
  if( err ){ /* Error Handler */ }
};
fs.readFile( './cryptico.js', 'utf8', fAsyncCalled );
```

Asynchronous Events

- Allow non-blocking, ...

Asynchronous Events

- Allow non-blocking, ...
- ... scalable application design

Asynchronous Events

- Allow non-blocking, ...
- ... scalable application design
- Time awareness is important

Asynchronous Events

- Allow non-blocking, ...
- ... scalable application design
- Time awareness is important
- Global variables almost extinguished

Well known for Browser (DOM) Events:

Onload Event Listener (Browser)

```
<!doctype html>
<html>
  <head></head>
  <body>
    <script>
      var triggerAlert = function () {
        window.alert( "Document loaded" );
      };
      window.addEventListener( "onload", triggerAlert );
    </script>
  </body>
</html>
```

Node.js Example:

Login Handler

```
var express = require( 'express' ), // Load Server Module
  app = express();

var handleLogin = function( req, resp ) {
  var body = '';
  req.on( 'data', function( data ) { body += data; } );
  req.on( 'end', function() {
     /* Process the request stored in body */
  });
}

app.post( '/login', handleLogin );
app.listen( 8111 ); // Start Listening on port 8111
```

Importance of time awareness:

*Declare functions **before** you **actually** assign them to an event that happens in the **future**!*

Why is this doing the wrong thing?

Print 0..99 in 3 seconds

```
for( var i = 0; i < 100; i++ ) {
  setTimeout( function() {
    console.log(i);
  }, 3000);
}
```

Solution:

Print 0..99 in 3 seconds

```
var fDelayed = function(id) { // Declare a function...
  return function() { // ( <-- Anonymous function )
    console.log(id);
  };
};
for( var i = 0; i < 100; i++ ) {
  // ... before you assign it to an event happening in the future:
  setTimeout( fDelayed(i), 3000);
}
```

Registers `i` by value and not by reference in the anonyomous function which is returned on the first function call of `fDelayed(i)`.

Real Life example:

Fetch User Objects from Database

```
function fetchUserObjectsAsync( arrNames, cbAnswer ){
  var semaphore = arrNames.length, // Number of objects to fetch
  objUsers = {},
  fGetUserAsync = function( name, cbLocal ){
    db.getUser(name, cbLocal);     // fork a token for the DB request
  },
  fJoin = function( uName ) {      // register username by value
    return function( err, obj ) { // Join the forked tokens
      --semaphore                  // Decrement the semaphore
      // Store the object for the user:
      if( !err && obj ) objUsers[ uName ] = obj;
      if( semaphore === 0 ){       // If all DB requests returned
        cbAnswer( null, objReplies ); // answer the overall request
      }
    }
  }; // <-- Until here only var declarations! Now the function code:
  for( var name in arrNames ){ // fetch object for each username
    fGetUserAsync( user, fJoin( name ));
  }
}
```

Node.js is all about writing, loading and glueing together powerful modules!

Ways to load modules:

- Built-in

Node.js is all about writing, loading and glueing together powerful modules!

Ways to load modules:

- Built-in
- Node Packet Manager

Node.js is all about writing, loading and glueing together powerful modules!

Ways to load modules:

- Built-in
- Node Packet Manager

Node.js is all about writing, loading and glueing together powerful modules!

Ways to load modules:

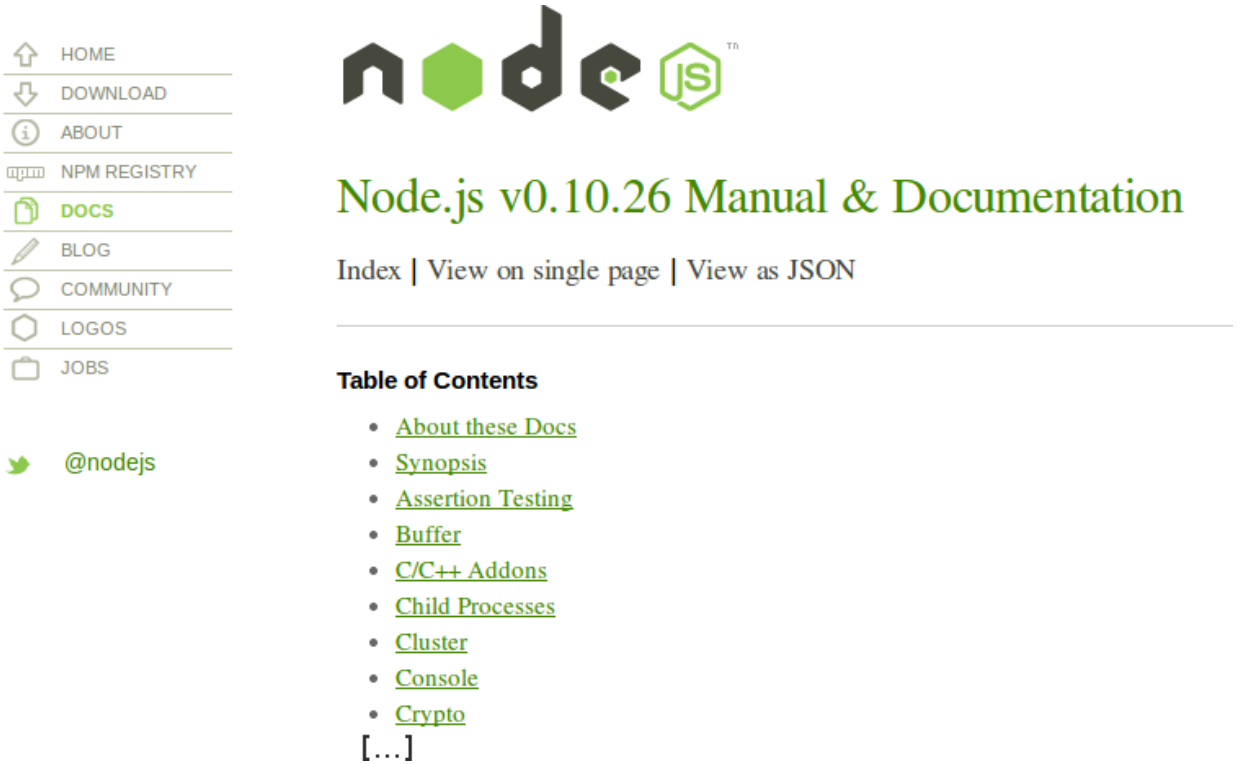
- Built-in
- Node Packet Manager
- Your Project Modules

$\Rightarrow$ **Modules are cached!**

Built-in Modules:

- Identify the module:

  `http://nodejs.org/api/`



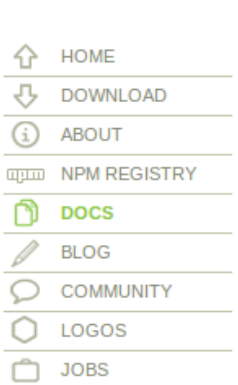
Built-in Modules:

- Identify the module:

  `http://nodejs.org/api/`

- Load the module:

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



Node Packaged Modules (NPM):

- Identify the module: `https://www.npmjs.org/`

Node Packaged Modules (NPM):

- Identify the module: `https://www.npmjs.org/`
- Install the module in one of the following ways:

Node Packaged Modules (NPM):

- Identify the module: `https://www.npmjs.org/`
- Install the module in one of the following ways:
  - Globally for the whole system:
    `npm install -g express`

Node Packaged Modules (NPM):

- Identify the module: `https://www.npmjs.org/`
- Install the module in one of the following ways:
  - Globally for the whole system:
    `npm install -g express`
  - Locally into the project folder:
    `npm install express`

Node Packaged Modules (NPM):

- Identify the module: `https://www.npmjs.org/`
- Install the module in one of the following ways:
  - Globally for the whole system:
    `npm install -g express`
  - Locally into the project folder:
    `npm install express`
  - **Through configuration file**

Node Packaged Modules (NPM):

- Identify the module: `https://www.npmjs.org/`
- Install the module in one of the following ways:
  - Globally for the whole system:
    `npm install -g express`
  - Locally into the project folder:
    `npm install express`
  - **Through configuration file**
- Load the module via:

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

Shortcomings of global and local installation:

```
npm install [-g] express
```

Shortcomings of global and local installation:

```
npm install [-g] express
```

- Installs latest version
  → *What if you need an older version?*

Shortcomings of global and local installation:

```
npm install [-g] express
```

- Installs latest version
  → *What if you need an older version?*
- Each module needs to be installed/removed seperately
  → *Portability of large projects?*

Shortcomings of global and local installation:

```
npm install [-g] express
```

- Installs latest version
  → *What if you need an older version?*
- Each module needs to be installed/removed seperately
  → *Portability of large projects?*
- Collaboration not as easy as it could be

Shortcomings of global and local installation:

```
npm install [-g] express
```

- Installs latest version
  → *What if you need an older version?*
- Each module needs to be installed/removed seperately
  → *Portability of large projects?*
- Collaboration not as easy as it could be

⇒ **Load NPM module through configuration file**

NPM management through Configuration File:

NPM management through Configuration File:

- Place package.json in project folder

package.json

```
{
  "name": "my-module",
  "author": "Dominic Bosch",
  "description": "My Module",
  "version": "0.1.0",
  "private": true,
  "repository": {},
  "dependencies": {
    "express": "3.4.8",
    "groc": "0.6.1"
  }
}
```

NPM management through Configuration File:

- Place package.json in project folder
- `npm install`

package.json

```
{
  "name": "my-module",
  "author": "Dominic Bosch",
  "description": "My Module",
  "version": "0.1.0",
  "private": true,
  "repository": {},
  "dependencies": {
    "express": "3.4.8",
    "groc": "0.6.1"
  }
}
```

- You will write more than one module in your project

- You will write more than one module in your project
- Attach to **`exports`** to be visible from outside

example-module.js

```
var fs = require('fs'),
  oe = require('./other-example');

exports.loadFile = function( path ) {
  oe.inform( path );
  try {
    return JSON.parse( fs.readFileSync( path ) );
  } catch (e) { /* [Error Handler] */ }
}
```

- You will write more than one module in your project
- Attach to **`exports`** to be visible from outside
- Load Modules:

```
var oe = require( './other-example' )
```

example-module.js

```
var fs = require('fs'),
  oe = require('./other-example');

exports.loadFile = function( path ) {
  oe.inform( path );
  try {
    return JSON.parse( fs.readFileSync( path ) );
  } catch (e) { /* [Error Handler] */ }
}
```

Thank You!

... and have fun developping your own Node.js modules!