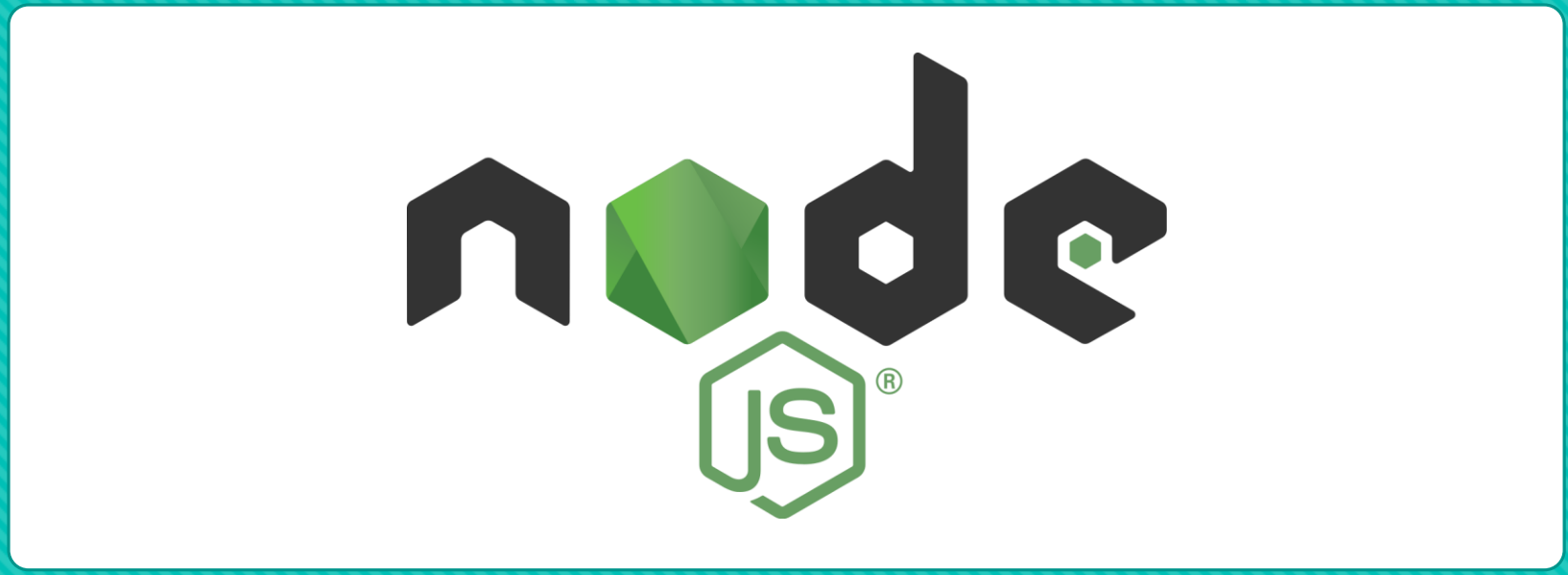


for all code => https://github.com/ipskm/example-node-tutorial

NODEJS Tutorial

Tutorial javaScript using nodejs

From Starch to Beginner

NodeJS Tutorial

Starch Chapter

BY
Phongsakorn Suttama

Introduction

What is Node.js ?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Why Node.js?

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

- Sends the task to the computer's file system.
- Waits while the file system opens and reads the file.
- Returns the content to the client.
- Ready to handle the next request.

Why Node.js?

Here is how Node.js handles a file request:

- Sends the task to the computer's file system.
- Ready to handle the next request.
- When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

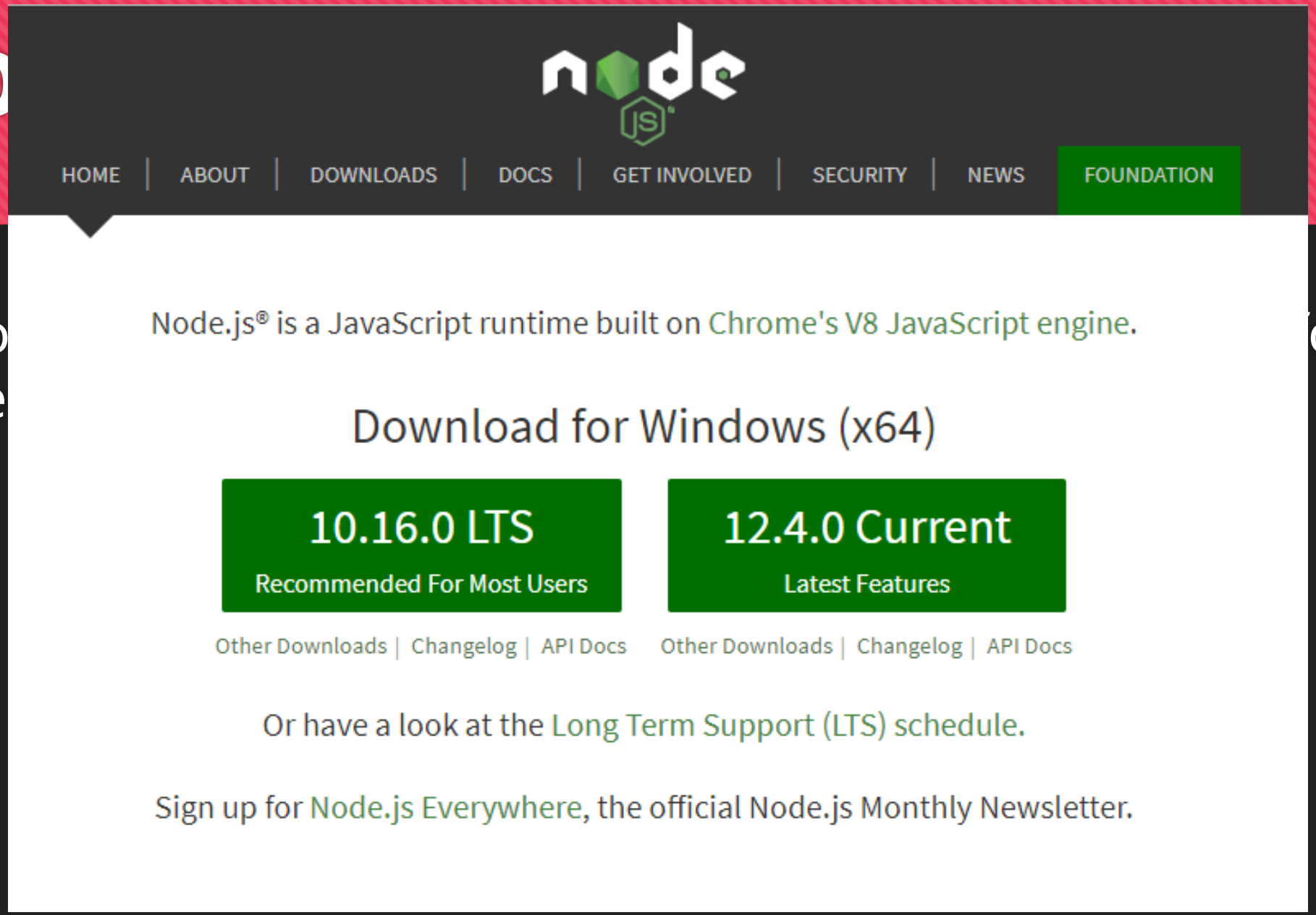
What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Get Started

Do

The o or Node



let's try to display "Hello World" in a web browser.

Create a Node.js file named "myfirst.js", and add the following code:

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

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

Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write node myfirst.js and hit enter:

```
C:\Users\Your Username\Your Directory>node myfirst.js
```

Now, your computer works as a server!

If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Start your internet browser, and type in the address: http://localhost:8080

Node.js Modules

What is a Module in Node.js?

- Consider modules to be the same as JavaScript libraries.
- A set of functions you want to include in your application.

Built-in Modules

- Node.js has a set of built-in modules which you can use without any further installation.
- Look at Built-in Modules Reference for a complete list of modules.

Include Modules

To include a module, use the require() function with the name of the module:

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

Now your application has access to the HTTP module, and is able to create a server:

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

Create Your Own Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

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

Use the exports keyword to make properties and methods available outside the module file , Save the code above in a file called "myfirstmodule.js"

Include Your Own Module

```
var http = require('http');
var dt = require('./myfirstmodule');

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

Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.

The Built-in HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, use the require() method:

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

Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client. Use the `createServer()` method to create an HTTP server:

```
var http = require('http');
//create a server object:

http.createServer(function (req, res) {

    res.write('Hello World!'); //write a response to the client

    res.end(); //end the response

}).listen(8080); //the server object listens on port 8080
```

The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on port 8080.Save the code above in a file called "demo_http.js", and initiate the file

Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

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

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

The first argument of the `res.writeHead()` method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Read the Query String

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).

This object has a property called "url" which holds the part of the url that comes after the domain name:

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

Save the code above in a file called "demo_http_url.js" and initiate the file

Split the Query String

There are built-in modules to easily split the query string into readable parts, such as the URL module.

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

var url = require('url');
http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

var q = url.parse(req.url, true).query;

var txt = q.year + " " + q.month;

res.end(txt);

}).listen(8080); Save the code above in a file called "demo_querystring.js" and initiate the file:
```

http://localhost:8080/?year=2019&month=July

The Built-in URL Module

The URL module splits up a web address into readable parts.

To include the URL module, use the require() method:

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

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

CODE

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

Node.js File Server

Now we know how to parse the query string, and in the previous chapter we learned how to make Node.js behave as a file server. Let us combine the two, and serve the file requested by the client.

Create two html files and save them in the same folder as your node.js files.

summer.html

```
<!DOCTYPE html>
<html>
<body>
    <h1>Summer</h1>
    <p>I love the sun!</p>
</body>
</html>
```

winter.html

```
<!DOCTYPE html>
<html>
<body>
    <h1>Winter</h1>
    <p>I love the snow!</p>
</body>
</html>
```

Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error:

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

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

Node.js NPM

What is NPM?

- NPM is a package manager for Node.js packages, or modules if you like.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js

What is a Package?

- A package in Node.js contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project.

Download a Package

Downloading a package is very easy. Open the command line interface and tell NPM to download the package you want. In this example I want to download a package called "upper-case":

C:\Users*Your Name*>npm install upper-case

NOTE : Using --save to install package into your project directory

Now you have downloaded and installed your first package!

NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

Using a Package

Once the package is installed, it is ready to use.

Include the "upper-case" package the same way you include any other module:

```
var uc = require('upper-case');
```

Create a Node.js file that will convert the output "Hello World!" into upper-case letters:

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

Save the code above in a file called "demo_uppercase.js", and initiate the file:

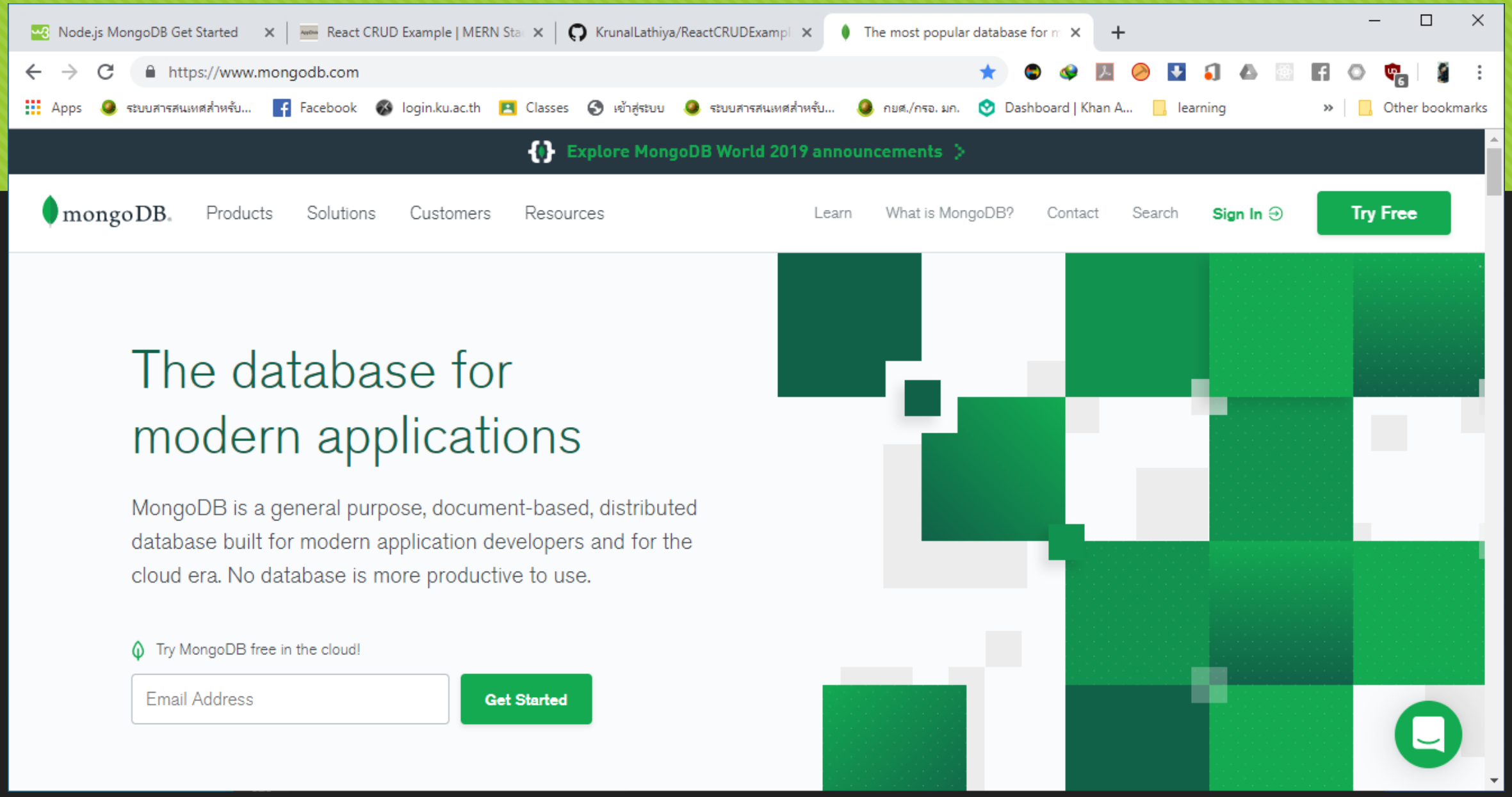
END Starch Chapter

NodeJS Tutorial

Beginner Chapter

BY
Phongsakorn Suttama

Node.js
MongoDB


Node.js MongoDB Get Started
React CRUD Example | MERN Sta
KrunalLathiya/ReactCRUDExampl
The most popular database for m
https://www.mongodb.com
Apps
ระบบสารสนเทศสำหรับ...
Facebook
login.ku.ac.th
Classes
เข้าสู่ระบบ
ระบบสารสนเทศสำหรับ...
กยศ./กรอ. มก.
Dashboard | Khan A...
learning
Other bookmarks
Explore MongoDB World 2019 announcements
mongoDB.
Products
Solutions
Customers
Resources
Learn
What is MongoDB?
Contact
Search
Sign In
Try Free
The database for modern applications
MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era. No database is more productive to use.
Try MongoDB free in the cloud!
Email Address
Get Started

Install MongoDB Driver

Let us try to access a MongoDB database with Node.js.

To download and install the official MongoDB driver, open the Command Terminal and execute the following:

C:\Users\Your Name> npm install mongodb

Now you have downloaded and installed a mongodb database driver.

Node.js can use this module to manipulate MongoDB databases:

```
var mongo = require('mongodb');
```

Creating a Database

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.
- MongoDB will create the database if it does not exist, and make a connection to it.

CODE

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    console.log("Database created!");
    db.close();
});
```

Save the code above in a file called "demo_create_mongo_db.js" and run the file:

Creating a Collection

To create a collection in MongoDB, use the createCollection() method:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.createCollection("customers", function(err, res) {
        if (err) throw err;
        console.log("Collection created!");
        db.close();
    });
});
```

Insert Into Collection

- To insert a record, or document as it is called in MongoDB, into a collection, we use the insertOne() method.
- A document in MongoDB is the same as a record in MySQL
- The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.

CODE

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var myobj = { name: "Company Inc", address: "Highway 37" };
    dbo.collection("customers").insertOne(myobj, function(err, res) {
        if (err) throw err;
        console.log("1 document inserted");
        db.close();
    });
});
```

Insert Multiple Documents

- To insert multiple documents into a collection in MongoDB, we use the insertMany() method.
- The first parameter of the insertMany() method is an array of objects, containing the data you want to insert.
- It also takes a callback function where you can work with any errors, or the result of the insertion:

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
if (err) throw err;
var dbo = db.db("mydb");
```

Code cont.

```
var myobj = [
{ name: 'John', address: 'Highway 71'},
{ name: 'Peter', address: 'Lowstreet 4'},
{ name: 'Amy', address: 'Apple st 652'},
{ name: 'Hannah', address: 'Mountain 21'},
{ name: 'Michael', address: 'Valley 345'},
{ name: 'Sandy', address: 'Ocean blvd 2'},
{ name: 'Betty', address: 'Green Grass 1'},
{ name: 'Richard', address: 'Sky st 331'},
{ name: 'Susan', address: 'One way 98'},
{ name: 'Vicky', address: 'Yellow Garden 2'},
{ name: 'Ben', address: 'Park Lane 38'},
{ name: 'William', address: 'Central st 954'},
{ name: 'Chuck', address: 'Main Road 989'},
{ name: 'Viola', address: 'Sideway 1633'}
];
```

Code cont.

```
dbo.collection("customers").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("Number of documents inserted: " +
res.insertedCount);
    db.close();
});
});
```

The _id Field

- If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.
- In the example above no _id field was specified, and as you can see from the result object, MongoDB assigned a unique _id for each document.
- If you do specify the _id field, the value must be unique for each document:

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
      if (err) throw err;
      var dbo = db.db("mydb");
      var myobj = [
            { _id: 154, name: 'Chocolate Heaven’},
            { _id: 155, name: 'Tasty Lemon’},
            { _id: 156, name: 'Vanilla Dream’}
      ];
      dbo.collection("products").insertMany(myobj, function(err, res) {
            if (err) throw err;
            console.log(res);
            db.close();
      });
});
```

Node.js MongoDB Find

- In MongoDB we use the **find** and **findOne** methods to find data in a collection.
- Just like the **SELECT** statement is used to find data in a table in a MySQL database.

Find One

- To select data from a collection in MongoDB, we can use the findOne() method.
- The findOne() method returns the first occurrence in the selection.
- The first parameter of the findOne() method is a query object. In this example we use an empty query object, which selects all documents in a collection (but returns only the first document).

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.collection("customers").findOne({}, function(err, result) {
        if (err) throw err;
        console.log(result.name);
        db.close();
    });
});
```

Find All

- To select data from a table in MongoDB, we can also use the find() method.
- The find() method returns all occurrences in the selection.
- The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.
- No parameters in the find() method gives you the same result as SELECT * in MySQL.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.collection("customers").find({}).toArray(function(err, result) {
    if (err) throw err;
        console.log(result);
        db.close();
    });
});
```

Find Some

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

    MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.collection("customers").find({}, { projection: { _id: 0, name: 1, address: 1 }
    }).toArray(function(err, result) {
        if (err) throw err;
        console.log(result);
    db.close();
    });
});
```

Filter the Result

- When finding documents in a collection, you can filter the result by using a query object.

- The first argument of the find() method is a query object, and is used to limit the search.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var query = { address: "Park Lane 38" };
    dbo.collection("customers").find(query).toArray(function(err, result) {
        if (err) throw err;
        console.log(result);
        db.close();
    });
});
```

Filter With Regular Expressions

- You can write regular expressions to find exactly what you are searching for.
- Regular expressions can only be used to query strings.
- To find only the documents where the "address" field starts with the letter "S", use the regular expression /^S/:

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var query = { address: /^S/ };
    dbo.collection("customers").find(query).toArray(function(err, result) {
        if (err) throw err;
        console.log(result);
        db.close();
    });
});
```

Sort the Result

- Use the sort() method to sort the result in ascending or descending order.

- The sort() method takes one parameter, an object defining the sorting order.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var mysort = { name: 1 };
    dbo.collection("customers").find().sort(mysort).toArray(function(err, result) {
        if (err) throw err;
        console.log(result);
        db.close();
    });
});
```

Sort Descending

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var mysort = { name: -1 };
    dbo.collection("customers").find().sort(mysort).toArray(function(err, result) {
        if (err) throw err;
        console.log(result);
        db.close();
    });
});
```

Delete Document

- To delete a record, or document as it is called in MongoDB, we use the deleteOne() method.
- The first parameter of the deleteOne() method is a query object defining which document to delete.
- Note: If the query finds more than one document, only the first occurrence is deleted.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var myquery = { address: 'Mountain 21' };
    dbo.collection("customers").deleteOne(myquery, function(err, obj) {
    if (err) throw err;
        console.log("1 document deleted");
        db.close();
    });
});
```

Delete Many

- To delete more than one document, use the deleteMany() method.

- The first parameter of the deleteMany() method is a query object defining which documents to delete.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var myquery = { address: /^O/ };
    dbo.collection("customers").deleteMany(myquery, function(err, obj) {
        if (err) throw err;
        console.log(obj.result.n + " document(s) deleted");
        db.close();
    });
});
```

Drop Collection

- You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

- The drop() method takes a callback function containing the error object and the result parameter which returns true if the collection was dropped successfully, otherwise it returns false.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.collection("customers").drop(function(err, delOK) {
        if (err) throw err;
        if (delOK) console.log("Collection deleted");
    db.close();
    });
});
```

db.dropCollection

- You can also use the dropCollection() method to delete a table (collection).
- The dropCollection() method takes two parameters: the name of the collection and a callback function.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.dropCollection("customers", function(err, delOK) {
        if (err) throw err;
        if (delOK) console.log("Collection deleted");
        db.close();
    });
});
```

Update Document

- You can update a record, or document as it is called in MongoDB, by using the updateOne() method.
- The first parameter of the updateOne() method is a query object defining which document to update.
- Note: If the query finds more than one record, only the first occurrence is updated.
- The second parameter is an object defining the new values of the document.

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var myquery = { address: "Valley 345" };
    var newvalues = { $set: {name: "Mickey", address: "Canyon 123" } };
    dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {
        if (err) throw err;
        console.log("1 document updated");
        db.close();
    });
});
```

Update Many Documents

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";
MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    var myquery = { address: /^S/ };
    var newvalues = {$set: {name: "Minnie"} };
    dbo.collection("customers").updateMany(myquery, newvalues, function(err, res) {
        if (err) throw err;
        console.log(res.result.nModified + " document(s) updated");
        db.close();
    });
});
```

Limit the Result

- To limit the result in MongoDB, we use the limit() method.
- The limit() method takes one parameter, a number defining how many documents to return.
- Consider you have a "customers" collection:

code

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.collection("customers").find().limit(5).toArray(function(err, result) {
        if (err) throw err;
        console.log(result);
    db.close();
    });
});
```

Join Collections

- MongoDB is not a relational database, but you can perform a left outer join by using the $lookup stage.

- The $lookup stage lets you specify which collection you want to join with the current collection, and which fields that should match.

- Consider you have a "orders" collection and a "products" collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("mydb");
    dbo.collection('orders').aggregate([
        { $lookup:
            {
                from: 'products’,
                localField: 'product_id’,
                foreignField: '_id’,
                as: 'orderdetails’
            }
        }
    ]).toArray(function(err, res) {
    if (err) throw err;
    console.log(JSON.stringify(res));
    db.close();
    });
});
```

END