爱创课堂前端培训

# NodeJS

第2天课堂笔记（本课程共6天）

班级：北京前端训练营19期

讲师：李兰波

日期：2019年1月23日

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

# 复习

NPM：NodeJS的模块管理系统

环境变量：当用户使用命令行，运行某一个命令时，如果该命令的路径不在环境变量中，那么用户必须切换到该目录下，再调用该命令。如果用户将该命令的路径配置在环境变量中，那么用户就可以在任意目录下调用该命令。

NPM常用命令：

npm install module1@x.x.x module2 module3…… 安装指定版本的模块和最新版本的模块 多个模块名称之间使用空格隔开

npm info module 查看该模块的信息

npm init 用于生成package.json文件

npm install 用于下载package.json中的所有依赖模块

Express：服务器框架

下载：npm install express

引入：var express = require("express");

使用：var app = express();

至此，服务器对象已经有了

监听端口号：app.listen(3000);

Express静态中间件：express.static 该中间件是Express唯一内置的中间件

使用：app.use(express.static(目录路径))；

至此，目录路径下的所有文件都被静态化了。也就是可以直接被前端通过HTTP请求访问到。但是，目录路径层级是不会出现在URL上的。

被静态化的目录: ./web

那么访问时的路径应该是: <http://localhost:3000/index.html> 不需要出现web层级

但是这样就会导致URL上的pathname部分与实际不同 所以配置的时候要限定路径

正确的使用方式: app.use("/web/", express.static("./web"));

获取GET请求提交的数据： req.query.xxx

获取POST请求的数据: req.body.xxx

但是需要插件的支持：body-parser

下载： npm install body-parser

引入： var body\_parser = require("body-parser");

使用： app.use(body\_parser.urlencoded({extended: false}));

注：这种方式只可以获取到普通POST请求。

文件上传：指的是浏览器端选择文件并提交，文件会发送到服务器上。

前端配置：method必须是POST enctype必须是multipart/form-data input[type=file]必须有name属性

后端配置：

匹配一个接口用于处理上传数据

使用formidable处理上传数据

下载： npm install formidable

引入： var formidable = require("formidable");

使用： 当前模块并不是Express的插件所以无需 app.use()

在路由内部 通过初始化formidable 得到对象 通过对象的parse方法处理

var form = new formidable();

form.uploadDir = "./uploads"; 该文件夹必须事先创建

处理多文件上传时需要的代码： 监听file事件 file事件是包含在parse事件中的

form.on("file", function(key, value) {

key: input的name属性

value: 本次解析完毕的文件信息对象

})

form.parse(req, function(err, fields, files) {

err: 解析过程中可能发生的异常

fields：文本区域的信息

files：文件区域的信息

})

路由：指的就是接口的路径

有两种配置方式：

第一种： app.verbs(pathname, handler);

第二种：

var router = express.Router();

router.verbs();

app.use(router);

Cookie的设置和读取：

设置cookie：本身Express就可以设置cookie

res.cookie(key, value, option);

key: 键值对的键名称

value: 键值对的值内容

option: 配置项

maxAge: cookie的存活周期 单位是毫秒值

domain: cookie的域名

path：cookie的路径名

httpOnly：是否只允许后端操作cookie

expires：cookie的存活周期 值是一个日期

读取cookie：通过中间件 cookie-parser

下载： npm install cookie-parser

引入： var cookie\_parser = require("cookie-parser");

使用： app.use(cookie\_parser());

至此，我们可以在app.use(cookie\_parser())之后的任意路由中 通过req.cookies获取cookie内容

Session的使用： 要使用express-session中间件

下载： npm install express-session

引入: var session = require("express-session");

使用： app.use(session({

secret: "随机字符串",

resave: false,

saveUninitialized: false

}))

EJS模板： 这是一个后端模板

前端发起一个登录请求，到达服务器，经过验证，通过验证，服务器要返回一个主页给用户，这个主页的内容要根据不同的用户渲染不同的用户名称。此时，开始后端模板的渲染。直到渲染完毕。至此，代码中的所有模板内容都没有了，再返回给前端。前端得到的就是完完整整的HTML文件内容。前端再渲染给用户看。

也就是，渲染过程发生在后端，也叫做：“后端渲染”。

配置：必须在app.js的同级 新建一个views文件夹 所有的模板文件都存储在views文件夹内 后缀名成是.ejs

语法：

<%%> 该语法开辟了一个真正的JS环境

<%=%> 该语法用于向页面中留下内容

后端调用：

res.render(path, dic);

path: 指的是模板文件的路径 该路径从views之后开始书写即可

dic：指的是渲染模板所需要的数据对象 也就是字典

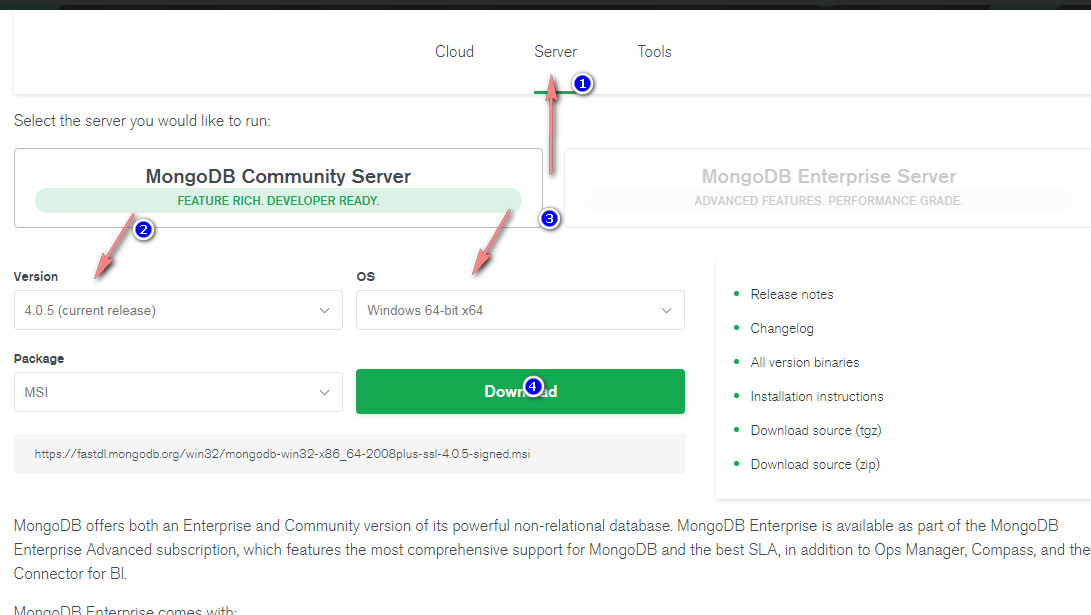
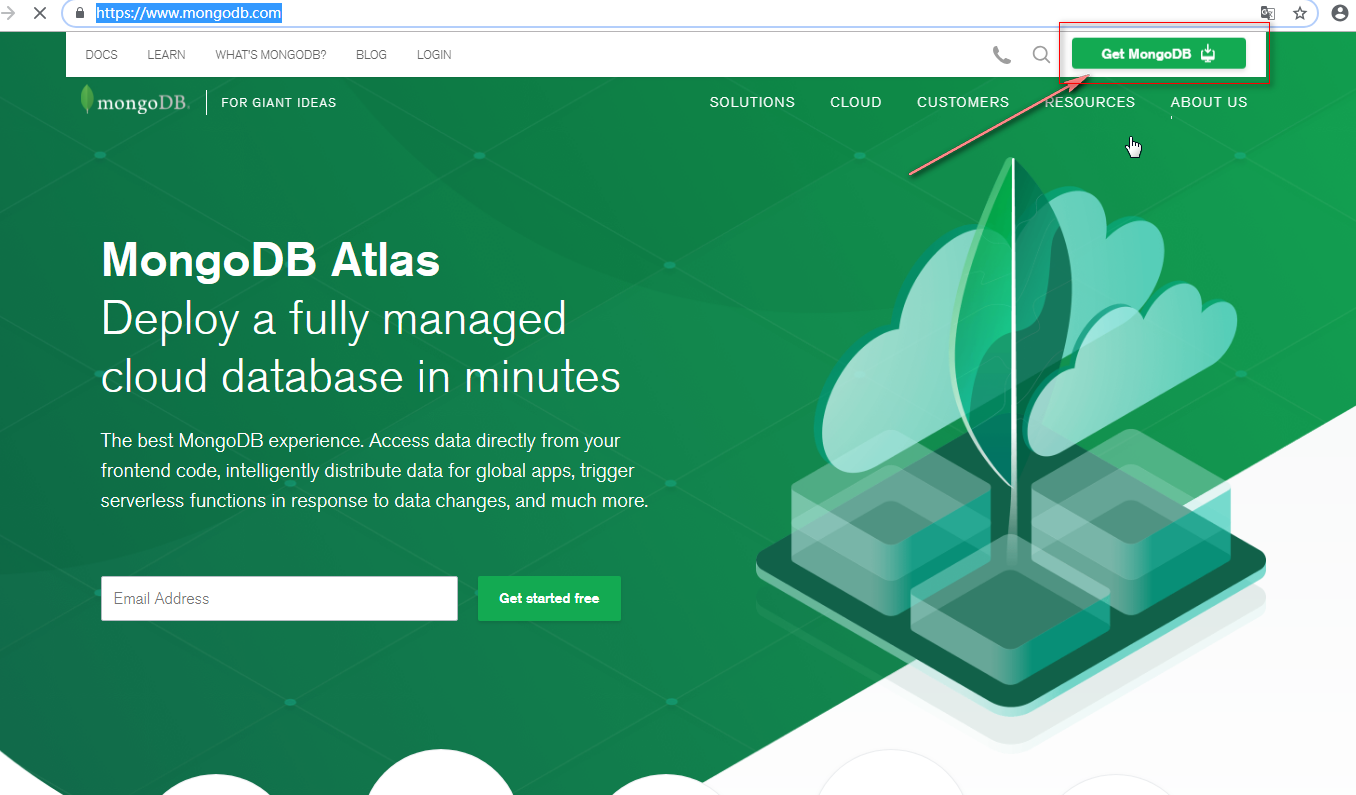
# MongoDB

这是一个数据库。所谓的数据库指的是存储数据的仓库。

## 1.1 官网

网址：<https://www.mongodb.com/>

下载：



## 1.2 概念介绍

关系型数据库：数据库的组成是由 数据库应用程序 => 数据库 => 表 => 字段

表中的每一条信息，都必然拥有表的所有字段。

MongoDB数据库： 这属于非关系型数据库 数据库组成是由 数据库应用程序 => 数据库文件夹 => 数据库 => 集合 => 字段

集合中的每一条信息的字段可以不同。

## 1.3 添加环境变量

当数据库安装完毕时，会附带一系列的命令。这些命令我们要经常用。但是又不想每一次都切换到该目录下去调用。所以我们就将它们的目录路径存放到环境变量中。

默认安装完毕的路径是:

|  |
| --- |
| 1. C:\Program Files\MongoDB |

但是它们的命令所在目录并不是MongoDB的安装目录，而是

|  |
| --- |
| 1. C:\Program Files\MongoDB\Server\3.2\bin |

所以我们要将第二个目录安装到环境变量中

## 1.4 常用命令

每一个应用程序都是要启动之后才可以访问。

### 1.4.1 启动数据库

MongoDB比较特殊，它可以在任意位置设置文件夹，并将文件夹部署成数据库文件夹。

第一步：创建数据库文件夹

在C盘根目录下创建一个ICKT\_19空文件夹

|  |
| --- |
|  |

目录自选，名称自定。

第二步：调用mongod命令 将它部署成一个mongodb数据库并开机

此时要打开命令面板。

|  |
| --- |
| 1. mongod --dbpath dbpath |

demo:

|  |
| --- |
|  |

此时，C:/ickt\_19就已经是一个数据库文件夹了。

之后，每一次都可以调用该命令 开机

### 1.4.2 连接数据库

注：连接数据库必须要保证数据库开机

命令： mongo

demo:想要调用mongo命令，但是无法在原本的命令面板中调用 所以再一个命令面板

|  |
| --- |
|  |

光标位置就是我们接下来输入命令 操作数据库的位置

如果没有开机则会报以下错误

|  |
| --- |
|  |

### 1.4.3 查看所有数据库

命令：show dbs

demo:

|  |
| --- |
|  |

### 1.4.4 切换/创建数据库

命令： use dbname

注：该创建方式并不会真的把数据库创建出来，而仅仅可以操作该数据库

demo:

|  |
| --- |
|  |

### 1.4.5 当前数据库

命令： db

该数据库就是当前的数据库，接下来的所有操作都是基于当前数据库进行的。

demo:

|  |
| --- |
|  |

### 1.4.6 当前数据库下的所有集合

命令： show collections

demo: 先切换到local数据库（自带的） 再调用命令 查看该数据库下有多少集合

|  |
| --- |
| 1. 有一个集合 startup\_log |

demo1: 再切换到一个并不存在的数据库中 再调用命令 查看集合

|  |
| --- |
| 1. 没有任何集合 |

### 1.4.7 插入数据

命令： db.collectionName.insert(data);

data: 被插入的数据 它是一个JSON格式的对象

demo:

|  |
| --- |
|  |

### 1.4.8 查询数据

命令： db.collectionName.find(query);

query: 表示条件 也是一个JSON格式的对象

demo: 查询所有 所有就表示没条件 所以不用传递参数

|  |
| --- |
|  |

demo: 查询age是24的数据

|  |
| --- |
|  |

demo: 查询age是23 并且name是王老五的

|  |
| --- |
|  |

### 1.4.9 删除数据

命令： db.collectionName.remove(query);

query: 表示删除条件 如果是无条件删除也必须传递一个空对象

demo: 删除所有

|  |
| --- |
|  |

### 1.4.10 修改数据

命令： db.collectionName.update(query, obj, option);

query: 筛选条件 表示要修改的是哪些数据

obj: 要修改成什么

option: 配置 可以通过配置决定修改一条还是修改多条

demo: 整体替换

|  |
| --- |
|  |

我们发现，将会只匹配一条，并将匹配到的整条数据，都修改为第二个参数。其余的字段都消失了。

demo1: 只匹配字段只修改字段 其余字段保留

|  |
| --- |
|  |

$set 是mongodb提供的一个修改器语法 用于修改指定的字段 或者添加字段

demo2: 修改多条

|  |
| --- |
|  |

第三个参数是一个配置对象，该对象的multi属性用于决定是单行匹配还是多行匹配。只要给定了一个可以转换为真的值 就会多行匹配。

### 1.4.11 limit

该方法用于限制 比如查询时的条目

demo: 原来有3条数据 现在限制1 表示只查询1条

|  |
| --- |
|  |

### 1.4.12 skip

该方法用于跳过几条

demo: 原来有3条数据，现在跳过一条 只查询两条

|  |
| --- |
|  |

# NodeJS连接MongoDB

服务器是服务器 数据库是数据库

这两个都是应用程序

现在我们要让一个应用程序去操作另外一个应用程序

NodeJS里有很多第三方模块，用于操作MongoDB数据库的有mongodb、mongoose模块

mongodb模块很简单 因为它就是在调用mongodb数据库的自身api

mongoose模块使用的是关系型数据库的思想去操作mongodb数据库

下载： npm install mongodb

demo:

|  |
| --- |
| 1. // 开始使用 2. var mongodb = require("mongodb"); 3. // 获取连接客户端 4. var MongoClient = mongodb.MongoClient; 5. // 定义连接字符串 6. var connectStr = "mongodb://localhost:27017"; 7. // 开始连接 8. MongoClient.connect(connectStr, {useNewUrlParser: true}, function(err, client) { 9. // 判断连接是否成功 10. if (err) { 11. // 连接失败 12. console.log("连接失败"); 13. console.log(err); 14. return; 15. } 16. // 连接成功 17. // 确定操作哪个数据库 18. var db = client.db("album"); 19. // 确定操作哪个集合 20. var coll = db.collection("users"); 21. // 通过coll调用find方法、调用update方法、调用remove方法等 22. coll.findOne({}, function(err, result) { 23. // 关闭数据库连接 24. client.close(); 25. // 判定是否出错 26. if (err) { 27. console.log("查询数据失败"); 28. console.log(err); 29. return; 30. } 31. // 成功 输出 32. console.log(result); 33. }); 34. }); |

# 如何查询mongodb模块的使用方式

<https://mongodb.github.io/node-mongodb-native/>

# 项目介绍

这是一个网站项目。可以登录、注册、修改个人信息、上传图片、创建相册、删除相册、查看相册内容、权限控制、裁剪图片。

## 4.1 布局

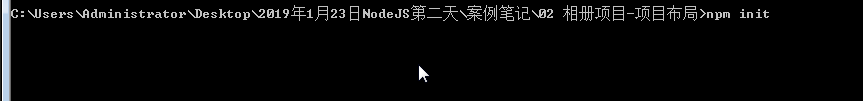
新建一个空文件夹，作为项目根目录。

|  |
| --- |
|  |

创建一个app.js 用于搭建服务器。

既然是一个NodeJS项目，那么必然少不了package.json

初始化



接下来是需要下载各种模块

|  |
| --- |
|  |

再接下来是分析项目网站结构  
web目录是用于盛放所有的前端文件

|  |
| --- |
|  |

再在web下创建一系列的前端文件夹比如 js文件夹、html文件夹、css等。

|  |
| --- |
|  |

再找到bootstrap的js与css 以及jquery、underscore的js放入各自的文件夹内

接下来是创建服务器：

创建服务器有这几步：1 生成 2 配置 3 路由 4 监听

生成代码

|  |
| --- |
| 1. // 引入express 2. var express = require("express"); 3. // 生成app 4. var app = express(); |

如果将配置代码与生成代码放在一起

|  |
| --- |
| 1. // 引入express 2. var express = require("express"); 3. // 生成app 4. var app = express(); 5. // 引入各种插件 6. var body\_parser = require("body-parser"); 7. var session = require("express-session"); 8. var cookie\_parser = require("cookie-parser"); 9. var formidable = require("formidable"); 10. // 配置各种插件 11. app.use(body\_parser.urlencoded({extended: false})); 12. app.use(cookie\_parser()); 13. app.use(session({ 14. secret: "sadfjoiewahfoieawhfoiewahfoieahwoifhewaoifh", 15. resave: false, 16. saveUninitialized: false 17. })); 18. app.use("/web/", express.static("./web")); |

两个代码掺杂在一起 不合适 于是就将配置代码分出去

所以要创建一个文件夹 叫做server

|  |
| --- |
|  |

与服务器相关的代码都放在这里

里面可以分两个文件夹或者更多 其中一个负责配置

|  |
| --- |
|  |

该文件夹下只有一个index.js 该文件就是放所有的配置代码的

|  |
| --- |
|  |

以后要修改配置，请找这个文件。所以此时，app.js中的代码就是：

|  |
| --- |
| 1. // 引入express 2. var express = require("express"); 3. // 生成app 4. var app = express(); 5. // 引入配置 6. var conf = require("./server/conf"); 7. // 配置 8. conf(app); |

同时，server中还需要一个文件夹，负责所有的路由名称叫做router

|  |
| --- |
|  |

## 4.2 首页

首页就是用户得到的第一个页面。我们当前项目作为首页的是登录及注册页面。

该页面拥有的功能：

|  |
| --- |
| 1. 登录： 2. 登录功能 3. 注册： 4. 用户名检测 5. 正则检测 使用的是策略模式 6. AJAX检测 7. 密码检测： 8. 检测两次密码是否一致 9. 注册功能 10. 当点击注册时，保证用户名和密码的检测都是通过 11. 所有的检测如果不通过要提示 需要模态框 12. 登录与注册表单之间的切换 |

使用的外部文件：

|  |
| --- |
| 1. bootstrap.min.js 方便布局 2. jquery.min.js 方便DOM操作 3. Strategy.js 方便策略的复用 4. Observer.js 方便模块之间的通信 5. Sea.js 模块化开发 |

## 4.3 封装数据库模块

在书写接口的时候，我们发现，所有的接口如果都挤在一个文件里，会不容易维护。所以，将接口拆分出去。于是就有了之前的server文件夹。

每一个接口是一个独立的文件。

这样如果有许多接口中都要连接数据库，如果每个接口都引入一次mongodb，书写一次连接过程，会很麻烦。于是就将连接数据库，操作数据库的相关代码，封装成一个模块。每一个接口文件引入该模块，调用该模块的方法即可。

于是就有了db文件夹

|  |
| --- |
|  |

内部只有一个index.js文件，封装了操作数据库的方法。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |