爱创课堂前端培训

# NodeJs

第1天课堂笔记（本课程共5天）

班级：北京前端训练营25期

讲师：彭帅伟

日期：2019年9月2日

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

目录

[NodeJs 1](#_Toc31638)

[一、 Express 3](#_Toc23418)

[1.1 介绍 3](#_Toc5827)

[1.2 官网 3](#_Toc27320)

[1.3 下载 3](#_Toc22877)

[1.4 使用 3](#_Toc6670)

[1.5 静态功能 4](#_Toc3467)

[1.6 处理get请求 4](#_Toc16405)

[1.7 Router 5](#_Toc14436)

[1.8 Express处理post请求 5](#_Toc15073)

[二、 Express中设置Cookie 6](#_Toc24480)

[2.1 介绍 6](#_Toc24947)

[2.2 Express中获取cookie内容 7](#_Toc28090)

[2.3 设置session 8](#_Toc11729)

[2.4 Token 9](#_Toc12341)

[三、 EJS 10](#_Toc19651)

[3.1 下载： 10](#_Toc2267)

[3.2 使用 10](#_Toc26753)

[四、 文件上传 11](#_Toc29235)

[4.1 单文件上传 11](#_Toc28125)

[4.1.1 使用 11](#_Toc14475)

[4.2 多文件上传 11](#_Toc18348)

[五、 nodemon自动重启工具 12](#_Toc5177)

[5.1 为什么要使用 12](#_Toc10869)

[5.2 如何使用 13](#_Toc30469)

[5.3下载 13](#_Toc6841)

[六、MongoDB数据库 13](#_Toc11745)

[6.1 SQL数据库 13](#_Toc27211)

[6.2 NoSql数据库 13](#_Toc16725)

[6.3 安装数据库 13](#_Toc3472)

[6.4 安装过程 14](#_Toc4678)

[6.5 环境变量 14](#_Toc28579)

[七、打开数据库 15](#_Toc24067)

[7.1 开启数据库 15](#_Toc25773)

[7.2 连接数据库 15](#_Toc27587)

# Express

## 介绍

基于 [Node.js](https://nodejs.org/en/) 平台，快速、开放、极简的 Web 开发框架

用于快速搭建后台服务器，快速解析post请求，快速处理静态功能……

## 官网

<http://www.expressjs.com.cn/>



## 下载

npm install express

|  |
| --- |
|  |

## 1.4 使用

|  |
| --- |
| 1. // 引入express 2. var express = require('express'); 3. // 创建应用程序 4. var app = express(); 5. // 监听端口号 6. app.listen(3000); |

## 1.5 静态功能

我们可以利用express的唯一中间件 将所需要访问的目录静态化

|  |
| --- |
| 1. // 引入express 2. var express = require('express'); 3. // 创建应用程序 4. var app = express(); 5. // 静态化web目录 可以利用express 的一个唯一中间件 6. app.use('/ickt/', express.static('./web')) 7. // 监听端口号 8. app.listen(3000); |

## 1.6 处理get请求

在Express中处理请求有两种方式：

第一种可以通过app.verbs处理请求，第二种是通过Router对象处理请求

第一种方式：

app.verbs(path, callback)

path: 匹配的规则

callback: 回调函数

req: 请求对象

res: 响应对象

next: 放行函数

注：verbs是一个是动词的复数形式 ，代表 get、post、put等等

demo:

|  |
| --- |
|  |

app.js:

|  |
| --- |
| 1. // 处理get请求 2. app.get('/ickt', function(req, res, next) { 3. // 获取数据 通过req.query 4. console.log(req.query); 5. }) |

req.query是express给原生Node请求对象添加的一个属性。

所携带的数据在req.query中

结果：

|  |
| --- |
|  |

在Express中允许对一个接口，添加多个处理函数配置

demo:

|  |
| --- |
| 1. // 处理get请求 2. app.get('/ickt', function(req, res, next) { 3. console.log('这里是第一个处理函数'); 4. // 执行next方法 5. next(); 6. }) 7. // 处理ickt的另外一个函数 8. app.get('/ickt', function(req, res, next) { 9. console.log('这里是第二个处理函数'); 10. }) |

结果：

|  |
| --- |
|  |

## 1.7 Router

Router是Express自带的一个构造函数，用于定义路由对象

利用该对象也可以处理请求

demo:

|  |
| --- |
| 1. // // 引入express 2. var express = require('express'); 3. // 定义路由对象 4. var Router = express.Router; 5. // 获取路由对象q 6. var router = new Router(); 7. // 处理逻辑交给router 8. router.get('/bbb', function() { 9. console.log('bbb') 10. }) 11. // 暴露功能 12. module.exports = router; |

此时， 所有的处理函数，都交给router来处理， 与app无关

但是只能处理，简单的post请求，无法处理携带的文件内容

## 1.8 Express处理post请求

处理请求的方式可以通过app.post或者是router.post

但是处理post请求需要借助一个中间件: body-parser

可以将post请求正文中的内容解析为对象并挂载在req.body中

获取post请求内容的方式: req.body

引入：

|  |
| --- |
| 1. // 引入body-parser 2. var body\_parser = require('body-parser'); |

配置：

|  |
| --- |
| 1. // 配置body\_parse 2. app.use(body\_parser.urlencoded({ extended: false })); |

处理：通过req.body获取请求内容

|  |
| --- |
| 1. // 处理post请求 2. router.post('/postData', function(req, res) { 3. console.log(req.body); 4. }) |

输出结果：

|  |
| --- |
|  |

# Express中设置Cookie

## 2.1 介绍

Cookie是HTTP协议中请求头中的一个字段

由服务器设置，浏览器保存

特点：当浏览器发送请求的时候，会找到对应的域名下的cookie文件并将内容携带到服务器中

登录过程：

当用户通过表单或者是AJAX提交登录的时候， 发送请求到服务器，服务器经过验证处理之后，将用户的信息设置在响应头的set-cookie中，并将内容返回，浏览器接收响应，会根据HTTP协议中的规定，解析set-cookit中的内容，生成一个cookit文件并存储，之后浏览器每发送的一个请求都会将cookit中的内容携带到服务器中，服务器根据cookit中的内容验证用户是否登录

在登录之前：

|  |
| --- |
|  |

在登录之后：响应头中出现set-cookie字段

|  |
| --- |
|  |

之后，不管浏览器发送任何请求都会将cookit字段携带到服务器中：

demo:

|  |
| --- |
|  |

请求头中：

|  |
| --- |
|  |

## 2.2 Express中获取cookie内容

当需要获取cookie中内容的时候

第一步: 借助一个中间件: cookie-parser

下载:

|  |
| --- |
| 1. npm install cookie-parser |

第二步：配置cookie-parser

|  |
| --- |
| 1. // 配置cookie-parser 2. app.use(cookie\_parser()); |

code:

|  |
| --- |
| 1. // 获取coolie 2. router.get('/getCookie', function(req, res) { 3. console.log(req.cookies); 4. }) |

结果：

|  |
| --- |
|  |

## 2.3 设置session

当用户登录成功之后，我们可以通过req.session来设置或者是获取内容

当获取的时候需要借助: express-session中间件

demo:

|  |
| --- |
| 1. // 引入express-session 2. var express\_session = require('express-session'); 3. // 配置session 4. app.use(express\_session({ 5. secret: 'ajsdlaoisdlanlksdnaslkdj', 6. resave: false, 7. saveUninitialized: false 8. })) |

router.js:

|  |
| --- |
| 1. // 处理post请求 2. router.post('/login', function(req, res) { 3. // 获取用户名 4. var username = req.body.username; 5. // 获取密码 6. var password = req.body.password; 7. console.log(username); 8. console.log(password); 9. // 设置session 10. req.session.username = username; 11. req.session.password = password; 12. // 返回信息 13. res.send('登录成功'); 14. }) 15. // 获取coolie 16. router.get('/getSession', function(req, res) { 17. // console.log(req.session); 18. // 获取用户名 19. var username = req.session.username; 20. var password = req.session.password; 21. console.log(username, password); 22. }) |

## 2.4 Token

含义：凭证、令牌

生成：由后端生成

存储：存储在前端的cookie中或者本地存储中

格式：头部，数据，签名

作用：验证用户身份

流程机制：

客户端使用用户名跟密码发送登录请求

服务端收到请求，去验证用户名与密码

验证成功后，服务端会生成一个 Token字符串，再把这个 Token字符串发送给客户端，服务器端不会保留该Token字符串。

客户端收到 Token字符串，可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里

客户端每次向服务端请求资源的时候需要带着服务端签发的 Token字符串

服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，响应本次请求，如果验证失败则服务器可以拒绝

特点：

服务器无状态 因为服务器只负责解密 而不负责存储

把所有状态信息都附加在 Token 上，服务器就可以不保存。但是服务端仍然需要认证 Token 有效。

只要服务端能确认是自己签发的 Token，而且其信息未被改动过，那就可以认为 Token 有效。“签名”就是做这个的。

Token 是在服务端产生的，如果前端使用用户名/密码向服务端请求认证，服务端认证成功，那么在服务端会返回 Token 给前端。

前端可以在每次请求的时候带上 Token 证明自己的合法地位。如果这个 Token 在服务端持久化（比如存入数据库），那它就是一个永久的身份令牌。

JWT标准：

因为这个过程的验证并不是HTTP协议中规定的方式，而是自定义的。所以很可能一个公司就有一个公司的使用方式。（程序员们从一个公司到了另一个公司，还要再学习另一套使用方式）。所以，就出现了标准。

标准的 Token 有三个部分：

header（头部）

payload（数据）

iss：Issuer，发行者

sub：Subject，主题

aud：Audience，观众

exp：Expiration time，过期时间

nbf：Not before

iat：Issued at，发行时间

jti：JWT ID

signature（签名）

header

payload

secret

# EJS

ejs是后台服务器的模板，天生可以与Express搭配，不需要引入，但是要下载

## 3.1 下载：

|  |
| --- |
| 1. npm install ejs |

## 3.2 使用

1 要创建一个views文件夹

2 所有的文件都要放在views目录下

3 文件的后缀名称必须是.ejs

4 通过res.render方法渲染模板

该方法的第一个参数是以views文件目录中的文件为准

第二个参数是一个字典对象

5 <%= %>是ejs模板标准语法，在<%%>之间开辟的是一个js环境

6 <%= %>第一个<% 与 =号之间不能有空格，但是后面是可以有空格的

views文件夹：

|  |
| --- |
|  |

code:

|  |
| --- |
| 1. // 返回一个新的页面 2. res.render('main.ejs', { 3. username: 'wanglaowu' 4. }) 5. }) |

# 文件上传

## 4.1 单文件上传

需要下载： formidable插件

formidable不是Express的插件，任何服务器都可以使用的

### 4.1.1 使用

1 引入formidable模块文件

2 初始化formidable实例

3 设置上传文件的路径 通过fd.uploadDir

4 解析req对象

fd.parse(req, handler)

req: 请求对象

handler: 处理函数

err: 异常参数

data: 前端提交的数据

files: 提交的文件

demo:

|  |
| --- |
| 1. // 引入formidable 2. var Formidable = require('formidable'); // 1 3. // 处理post请求 4. router.post('/postData', function(req, res) { 5. // 初始化formidable 6. var fd = new Formidable(); // 2 7. // 设置上传的路径 8. fd.uploadDir = './uploads'; // 3 9. // 解析req 10. fd.parse(req, function(err, data, files) { // 4 11. // console.log(err); 12. // console.log(data); 13. // console.log(files); 14. }) 15. }) |

## 4.2 多文件上传

当我们需要上传多个文件的时候，如果使用单文件上传的方式，在回调函数中处理的是最后一个文件

所以此时要监听一个事件: file事件

该事件会在每一张文件上传之后触发一次

demo:

|  |
| --- |
| 1. // 处理post请求 2. router.post('/postData', function(req, res) { 3. // 初始化formidable 4. var fd = new Formidable(); 5. // 设置上传的路径 6. fd.uploadDir = './uploads'; 7. // 定义一个空数组， 用于存储上传的文件 8. var fileArr = []; 9. // parse方法用于解析req对象， 分别可以解析文件数据和文件数据 10. // 当需要上传多个问价的事件要监听file事件 11. fd.on('file', function(key, value) { 12. console.log(value); 13. // 没上传一个文件数据就存储起来 14. fileArr.push(value); 15. console.log(fileArr); 16. }) 17. // 解析 18. fd.parse(req, function() { 19. // console.log('上传完毕了'); 20. try { 21. // 循环处理fileArr 22. for (var i = 0; i < fileArr.length; i++) { 23. // 获取原名称 24. var oldName = fileArr[i].path; 25. // 获取新名称 26. var newName = fileArr[i].name; 27. // 改名 28. fs.renameSync(oldName, newName); 29. } 30. } catch (e) { 31. if (e) { 32. res.send({ 33. error: 1, 34. data: '更改名称失败' 35. }) 36. return; 37. } 38. } 39. // 说明没有问题 40. res.send('更换名称成功'); 41. }) 42. }) |

# nodemon自动重启工具

## 5.1 为什么要使用

在编写调试Node.js项目，修改代码后，需要频繁的手动close掉，然后再重新启动，非常繁琐。现在，我们可以使用nodemon这个工具，它的作用是监听代码文件的变动，当代码改变之后，自动重启。

## 5.2 如何使用

例如：[nodemon](https://link.jianshu.com/?t=https://github.com/remy/nodemon" \t "https://www.jianshu.com/p/_blank) app.js

## 5.3下载

npm install -g nodemon

安装到全局

npm install -g nodemon

安装完毕之后可以在：

C:\Users\Administrator\AppData\Roaming\npm 找到指令

# 六、MongoDB数据库

数据库：存储数据的仓库。

世界上有各种各样的数据库应用程序。

大致可分为两类：

SQL数据库： mySql、oracle、sqlserver等

NoSql数据库：Mongodb数据库等

## 6.1 SQL数据库

结构化查询语言：structure query language

它的结构组成：应用程序 => 数据库 => 表 => 数据

使用SQL数据库要学习SQL语言。

## 6.2 NoSql数据库

它的结构组成：应用程序 => 数据库 => 集合 => 文档。

SQL与NOSQL的最大区别在于NoSQL的集合中的文档，它的结构可以不固定。

## 6.3 安装数据库

官网：<https://www.mongodb.com/>

获取安装包：

第一步点击try free

然后按照步骤下载：

|  |
| --- |
|  |

下载之后的安装包：

|  |
| --- |
|  |

## 6.4 安装过程

略

安装完毕之后的路径：

C:\Program Files

找到：

|  |
| --- |
|  |

## 6.5 环境变量

它是系统中的配置项

用于扩大命令的调用范围

如何找到环境变量并设置

1：通过右键点击计算机 选择属性

2：点击高级系统设置

3：找到环境变量

4：找到系统变量

5：找到path字段 点击边界

6：在编辑系统变量中 首先要找到命令所在目录

7: 在一堆字符串的后面，用英文半角符号隔开 将命令所在的目录路径粘贴上去

此时，在窗口命令中任何调用目录命令下执行 mongo命令 不再出现 “不是内部或者是外部命令”

# 七、打开数据库

## 7.1 开启数据库

通过mongd指令开启数据库

此时，要找到c:/ 创建data的文件夹在里面还要创建db文件夹

当创建完成之后，再次执行mongd指令 之后就会开启数据库

demo:

|  |
| --- |
|  |

## 7.2 连接数据库

通过mongo指令建立连接

连接之后：

|  |
| --- |
|  |

如果想要连接其它数据库要通过另一个指令：

例如：

mongod --dbpath c:/ickt25

当建立连接完毕之后之前的空文件夹现在将变成以下：

|  |
| --- |
|  |

在该文件中的内容不要修改

如果在开启一个数据库之后还需要开启另一个数据库:

输入指令 mongod --port 27018

当需要连接端口是27018数据库的时候：

输入指令 mongo --port 端口号

关闭数据库：

通过ctrl + C

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |