爱创课堂前端培训

# NodeJS第一天

第1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21日

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

# NPM

NPM：NodeJS Package Manager NodeJS的模块管理器

NodeJS是模块化开发的。有两种模块。一种是自带的内置核心模块，一种是其它程序人员编写的第三方模块。

NPM是用来管理所有的第三方模块的。

比如 程序员A开发了一个模块a。程序员B开发了一个模块b。他俩分别处于南北半球。但是又正好需要对方开发出来的模块。此时如果他们都上传自己的模块到NPM。那么他们就可以搜索NPM上是否有自己需求的模块。就可以搜索得到。但是如果没有NPM，那么他们只能各自再开发一个对方的模块。就重复开发了。

官网：<https://www.npmjs.com/>

NPM已经集成在NodeJS的安装包中。

查看NodeJS版本：

|  |
| --- |
|  |

查看NPM版本：

|  |
| --- |
|  |

# 环境变量

windows系统中，有一个地方叫做环境变量。它可以监控所有的命令。只要你配置了环境变量。

它是一个配置项，配置的值是一个一个的字符串路径。比如：如果不配置环境变量，那么一个程序只可以在它的安装目录下调用。如果配置了环境变量：将该程序的所在目录字符串放到环境变量中，那么就可以在任意的地方去调用该程序。

举例：安装nodeJS时，我故意将addToPath选项取消。那么安装完毕之后，NodeJS不会将自身的所在路径加入到环境变量中。那么Node命令将只能够在安装目录下调用。

|  |
| --- |
| 1. 在外部调用失败  1. 切换到安装目录下  1. 调用成功 |

现在，我将NodeJS的安装目录 C:\Program Files\nodejs放入环境变量中

打开环境变量的方式： 右击桌面的计算机，选择属性 => 在新打开的面板中找到高级系统设置，弹出一个系统属性面板 => 选择“高级”选项卡（一般会默认） => 在该面板下选择环境变量按钮并点击 => 弹出一个新的面板 从系统变量中找到Path选项=> 如果是win7系统，则弹出的面板中所有的路径字符串都在一行，此时需要在每一个路径字符串之后，添加一个英文半角分号。并将刚刚复制的C:\Program Files\nodejs放在最后。保存即可。 如果是win10 则添加会弹出一个全新的面板。此时粘贴并保存即可。每一个面板都需要点击确定。

此时，打开一个新的命令面板。输入node -v测试是否配置成功。

# NPM的常用命令

## 3.1 下载模块

npm install moduleName1 moduleName2 ……

该命令用于下载第三方模块。下载规则：查看当前文件夹下是否具备node\_modules文件夹 如果有就把模块下载下来并放进去 如果没有node\_modules文件夹 此时会向上一级查找 如果找到就放入 如果没有再向更上一层找。直到最顶层。如果还没有找到，回到当前的下载目录中，创建一个新的node\_modules文件夹并放入。

## 3.2 指定版本

模块是具备版本号的。所以，你可能下载的并不是你想要的版本。因为默认下载最新版本。此时可以使用moduleName@x.x.x这样的方式去下载指定版本

demo:

|  |
| --- |
| 1. npm install jquery@1.7.2 |

## 3.3 查看版本

npm info moduleName

该命令用于查看模块信息

|  |
| --- |
|  |

## 3.4 package.json

这是一个文件。每一个NodeJS的第三方模块都应当具备该文件。该文件是当前模块的配置信息。比如：模块名称、模块描述、模块版本、模块的仓库、模块的启动命令等。

underscore.js模块的配置信息文件：package.json

|  |
| --- |
|  |

package.json内容：

|  |
| --- |
|  |

## 3.5 生成package.json

npm init 该命令用于生成package.json

生成过程：

|  |
| --- |
|  |

## 3.6 添加依赖到package.json

我们在开发之前，要生成一个package.json 因为我们开发的过程中要下载模块，如果没有package.json 则会报警告。如果有该文件，则会将下载的工具库添加到依赖项中。

|  |
| --- |
|  |

## 3.7 重新安装模块

如果有一天你的项目要从一台电脑上移动到另外一台电脑上。此时，我们应该复制并粘贴。但是这样的话，如果项目挺大，依赖的模块挺多。这样会花费较长的时间。于是我们就不需要移动模块文件。也就是不要node\_modules文件夹了。但是没关系，我们可以只带package.json文件过去。此时我们可以运行**npm install**命令。重新下载项目所依赖的模块。

# Express

这是一个服务器框架。

## 4.1 下载

npm install express

|  |
| --- |
|  |

## 4.2 搭建服务器

代码一共三行：

|  |
| --- |
| 1. // 既然要使用express创建服务器 那么就引入进来 2. var express = require("express"); 3. // 创建服务器 4. var app = express(); 5. // 监听端口号 6. app.listen(3000); |

访问：

|  |
| --- |
|  |

## 4.3 静态功能

express唯一内置的中间件 叫做“静态中间件”

它的作用：将某一个目录静态化

被静态化的目录中的内容就可以直接被前端访问。

使用方式:

|  |
| --- |
| 1. // 既然要使用express创建服务器 那么就引入进来 2. var express = require("express"); 3. // 创建服务器 4. var app = express(); 5. // Express唯一内置的中间件 static 该中间件用于静态化目录 6. app.use("/web/", express.static("web")); 7. // 监听端口号 8. app.listen(3000); |

现在将web目录下的内容都静态化了

app.use("/web/", express.static("web"));

以上这条代码中，第一个参数表示访问时的路径 第二个参数是被静态化的目录

如果没有第一个参数：app.use(express.static("web"));

|  |
| --- |
| 1. http://localhost:3000/index.html |

但是此时index.html其实是在服务器根目录下的web目录下。也就是前后端路径表示法不同。

为了统一，我们填写第一个参数：app.use("/web/", express.static("web"));

|  |
| --- |
| 1. http://localhost:3000/web/index.html |

## 4.4 获取GET请求提交的数据

匹配GET请求：app.get(pathName, handler);

demo：

|  |
| --- |
| 1. // 想要接收到GET请求数据 那么必须先匹配到该接口 2. app.get("/checkName", function(req, res) { 3. console.log(req.query); 4. }); |

当访问的时候：http://localhost:3000/checkName?username=123

|  |
| --- |
|  |

也就是说，只要使用Express了。那么就可以直接通过req.query得到GET请求提交的数据。

## 4.5 获取POST请求提交的数据

想要获取POST提交的数据，要借助插件：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}));

此时，我们就可以在req身上找到一个body属性。它就是请求体中的内容。

|  |
| --- |
| 1. // app.post用于处理POST请求 2. app.post("/login", function(req, res) { 3. console.log(req.body); 4. }); |

前端发送post请求：

|  |
| --- |
|  |

## 4.6 文件上传

在很多场景下，都需要将客户端的内容上传到服务器上。这叫做文件上传。

文件上传的准备

前端：使用表单上传时，要将method设置为post，要将enctype属性设置为multipart/form-data

后端：监听事件，处理。我们的处理方式是使用现成的模块。该模块叫做formidable。

下载formidable:

|  |
| --- |
| 1. npm install formidable |

引入：

|  |
| --- |
| 1. var formidable = require("formidable"); |

使用: 注，它不是express的模块，不需要app.use 而是直接在处理函数内 初始化

|  |
| --- |
| 1. app.post("/upload", function(req, res) { 2. // 初始化 3. var form = new formidable(); 4. // 设置上传临时存储路径 5. form.uploadDir = "./uploads"; // 该文件夹必须事先存在 6. // 开始解析req 因为前端提交上来的所有数据都在req身上 7. form.parse(req, function(err, fields, files) { 8. err: 错误信息 9. fields: 提交上来的文本内容 是一个对象 等价于req.query req.body 10. files: 提交上来的文件内容 是一个对象 属性名是前端的文件控件的name 属性值是该控件提交的文件的信息对象 11. }) 13. }) |

## 4.7 多文件上传

依旧使用formidable处理。

前端：只需要将input的multipe属性设置即可

后端：因为此时是有多个文件的，但是parse方法的第三个参数只能够得到最后一张图片的信息 所以要监听file事件

|  |
| --- |
| 1. form.on("file", function(key, value) { 2. console.log(key, value); 3. key是前端input控件的name属性 4. value是本次解析出来的图片的信息对象 5. }); |

重命名：定义一个数组 在file事件中将每一个图片的信息存储 并最终在parse方法中调用即可

|  |
| --- |
| 1. form.parse(req, function(err, fields, files) { 2. // 判定是否出错 3. if (err) { 4. // 报错 给前端一个响应 5. res.send({ 6. error: 1, 7. data: "解析失败" 8. }); 9. return; 10. } 11. try { 12. // 如果代码走到这里 说明没有进入err 说明没有错误 13. arr.forEach(function(value, key) { 14. // 获取原来的图片的名称 15. var user\_name = value.name; 16. // 获取临时文件的路径 17. var old\_name = value.path; 18. // 定义现在的名字 19. var new\_name = form.uploadDir + "/" + user\_name; 20. // 重命名 21. fs.renameSync(old\_name, new\_name); 22. }); 23. } catch(e) { 24. res.send({ 25. error: 2, 26. data: "代码执行失败" 27. }); 28. return; 29. } 30. // 当循环完毕 就已经修改了全部的图片的名称 31. res.send({ 32. error: 0, 33. data: "图片全部上传成功" 34. }); 35. }); |

## 4.8 路由

路由指的就是接口的匹配。

路由的匹配方式有两种

一种是app.verbs(); 比如：app.get() app.post() ……

|  |
| --- |
| 1. // 配置路由 2. app.post("/login", function(req, res) { 3. res.send("登录成功"); 4. }); |

还有另外一种：router 这是一个路由对象

获取该对象的方式:

|  |
| --- |
| 1. var router = express.Router(); |

该对象也可以调用get、post等方法。

|  |
| --- |
| 1. router.get(xxxx, handler); 2. router.post(xxxx, handler); |

最后，要将router挂载到app上。就可以让app一口气拥有所有router的配置项。

|  |
| --- |
| 1. var express = require("express"); 2. // 配置路由方式二 由router去调用 3. var router = express.Router(); 4. var login = require("./login"); 5. var regist = require("./regist"); 6. router.post("/login", login); 7. router.post("/regist", regist); 8. module.exports = router; |

引入，并挂载：

|  |
| --- |
| 1. // 引入外部的router对象 2. var router = require("./router"); 3. // 挂载到app上 4. app.use(router); |

## 4.9 Cookie

cookie是key=value;key1=value1的格式。

express自带操作cookie的方法。

res.cookie(key, value, option);

key: cookie的一个字段名

value: 对应的字段值

option: 配置对象

请求面板中的相关信息：

|  |
| --- |
|  |

Name表示cookie的一个字段的名称

Value表示cookie的一个字段的值

Domain 表示所属的域名

Path 表示所属的路径

Expires/Max-Age 最大存活时间 Expires是日期格式 Max-Age是毫秒值格式

Size 内容的大小

HTTP 布尔值 表示是否不允许浏览器端操作

以上是设置cookie。

获取Cookie要使用cookie-parser中间件。

下载: npm install cookie-parser

使用:

|  |
| --- |
| 1. // cookie-parser是用于将接收到的cookie字符串 转化成cookie对象 并安装在req身上 2. var cookie\_parser = require("cookie-parser"); 3. // 配置cookie 4. app.use(cookie\_parser()); |

访问时，请求会将cookie带上来，此时可以通过输出req.cookies得到cookie中的内容。

|  |
| --- |
|  |

## 4.10 EJS

这是一个模块。用于制作后端模板文件。

一个网站，可以允许登录，用户登录之后，看到的页面是大同小异。那么这些用户看到的页面真的是完整的静态页面吗？如果是真正的.html静态页，那么会累死前端程序员。因为有几个用户，就需要有几个静态页。于是，就出现了后端模板。只需要一个模板，把用户名的地方挖空，到时候，谁登录就填谁。

使用方式：

首先，所有的ejs模板文件的拓展名是.ejs

其次，所有的ejs模板文件一定要放在一个叫做views的文件夹内 该文件夹一定放在根目录下

语法：定义ejs的语法

|  |
| --- |
| 1. <%%> 开辟了一个真正的JS执行环境 2. <%=xxx%>向页面中输入内容 |

渲染：res.render(templatePath, dictionary);

templatePath: 模板所在的位置 注 只需要从views/之后开始书写即可

dictionary: 模板中的数据所存在的字典

|  |
| --- |
| 1. // 渲染模板 2. res.render("main.ejs", { 3. username: req.cookies.username 4. }); |

注意：一定要将EJS模块下载下来，但是无需引入。

## 4.11 Session

Session是后端的一块内存地址。用于存放重要的信息。比如用户名密码等。

Express用于操作Session的模块叫做express-session

下载: npm install express-session

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

使用:

|  |
| --- |
| 1. app.use(session({ 2. secret: "ofajwoifjewaoifjpoiewanmfpewaofewaofjewpofjpoewajfpoewjpodjewpof", 3. resave: false, 4. saveUninitialized: false 5. })) |

此时，当用户登录的时候，执行的代码是:

|  |
| --- |
| 1. req.session.username = req.body.username; 2. req.session.password = req.body.password; |

此时，响应头中会多出一条设置语句：

|  |
| --- |
|  |

这就是因为设置了session所以导致的cookie生成。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |