

# Node.js入门和企业级项目开发

第2天课堂笔记

班级：全栈前端训练营1120班

讲师：邵山欢

日期：丁酉年癸卯月庚戌日

邵山欢老师

微博：@邵山欢

QQ : 179427026

E-mail : ssh@iqianduan.cn

爱前端官网 ：[www.iqianduan.cn](http://www.iqianduan.cn)

# 目录

[Node.js入门和企业级项目开发 1](#_Toc23321)

[目录 2](#_Toc23241)

[一、 上节课的复习 3](#_Toc30725)

[二、模块 6](#_Toc16218)

[2.1 内置模块 6](#_Toc28778)

[2.2 自定义模块 7](#_Toc23425)

[2.2.1 require谁就会运行谁 7](#_Toc9499)

[2.2.2 require的js文件中如果有异步函数，将不会等待 7](#_Toc22175)

[2.2.3 连续require 8](#_Toc19019)

[2.2.4 循环引用 9](#_Toc32018)

[2.2.5 文件夹层次 9](#_Toc2059)

[2.3 js文件天生是隔离作用域的 10](#_Toc22309)

[2.4 使用exports.xx = xx的语法暴露 11](#_Toc19445)

[2.5 使用module.exports暴露 13](#_Toc28834)

[2.6 神奇的node\_modules文件夹 14](#_Toc15602)

[三、npm的世界 17](#_Toc7656)

[3.1 npm的使用 17](#_Toc2691)

[3.2 使用package.json管理依赖 19](#_Toc12972)

[3.3 全局安装 22](#_Toc23282)

[3.4 淘宝镜像 22](#_Toc32721)

[四、GET请求和POST请求 23](#_Toc30796)

[4.1 GET请求 23](#_Toc22784)

[4.2 POST请求 24](#_Toc6695)

[4.3 formidable模块 24](#_Toc10203)

[五、Express.js框架 26](#_Toc23153)

[5.1 基本使用 26](#_Toc523)

[5.2 静态路由 27](#_Toc5597)

[5.3 使用某一个页面 28](#_Toc8310)

[5.4 模板引擎 29](#_Toc15538)

# 一、 上节课的复习

请介绍Node.js，它是什么？

Node.js是一个JavaScript的runtime，运行环境，实际上就是利用Chrome V8引擎，将它移植到了服务器上，用它开发服务器程序，可以提供HTTP服务。

Node.js的主要特点是什么？

Single Thread，单线程； Non-blocking I/O非阻塞I/O；Event Driven事件驱动。

这三个特点是相辅相成的，是必须的选择。

|  |
| --- |
| Node为了在低硬件服务器条件下高并发，所以就减少内存消耗，剑走偏锋，选择了单线程  ↓  必须要非阻塞I/O，因为你只有一个线程，你必须当A用户去I/O的时候，处理B业务的事情，B业务去I/O你要去处理C的事情……A、B、C都有回调函数。  ↓  为了让A、B、C不乱套，每个人都进行合理调度，谁先来处理谁。你不能让B一直等待，处理C、D、E……所以Node使用了一个机制叫做事件环，采用事件驱动来调度事件。 |

哪条语句是正确的，为什么？

A语句：

var data = fs.readFile("./test.txt");

B语句：

fs.readFile("./test.txt",function(err,data){

});

因为fs模块的readFile方法是异步方法，事实上Node中的fs模块、mongodb模块基本上都是异步方法。一定要记住（当做规矩记住），**异步函数不能通过return返回，不能通过等号来接受数据。必须通过回调函数传实参的模式来传输数据。**

说说http模块，他有什么功能？有哪些方法？req和res各有什么属性和方法？

http模块用来提供HTTP服务的。最常用的：

http.createServer(function(req,res){

});

创建了一个服务器。

req对象封装了HTTP上行请求的所有信息，常用属性req.url、req.connection.remoteAddress、req.type等等。

res对象是服务器应该给出的下行响应，常用方法是req.write()、req.end()、req.setHeader()等等。

说说NodeJS的顶层路由设计

Apache和NodeJS的模式完全不一样。

Apache天生有静态资源服务，但是NodeJS不行。NodeJS必须使用路由清单给出明确的路由才行：

var pathname = url.parse(req.url).pathname;

if(pathname == "/music" ){

res.end("音乐频道");

}else if(pathname == "/news"){

res.end("新闻频道");

}else if(/^\/student\/[\d]{6}$/.test(pathname)){

var xuehao = /^\/student\/([\d]{6})$/.exec(pathname)[1];

res.end("学生频道，学号是" + xuehao);

}else{

res.end("错误的频道号码");

}

说说常用的CMD命令

CMD命令在Windows下很弱，将来我们学习Linux你会发现相当强大。

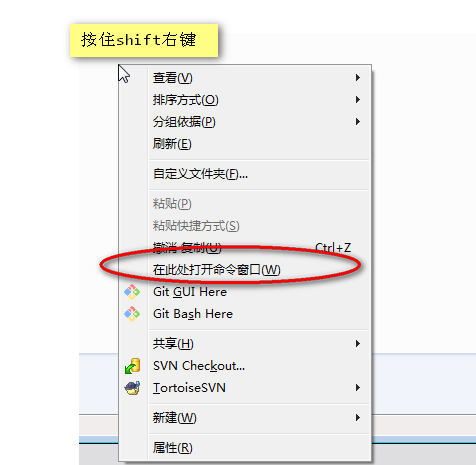
$ cls

清除屏幕内容

$ cd c:\node\_study

切换文件夹，cd是change directory

今天学习一个小tip，windows中的资源管理器中，任何位置按住shift+右键，此时有：



$ mkdir abc

创建文件夹abc

$ dir

列出当前目录的所有文件

$ node 1.js

执行node命令。

当CMD挂起的时候，使用ctrl+c来打断它。

# 二、模块

先区分两个词：

模块我们叫做module，指的是一系列有关系的js程序的集合。



后面我们学习MVC中的M指的是model，模型，模型是实现最基本的算法的程序单位。



## 2.1 内置模块

NodeJS中内置了很多模块，可以直接用require来进行引用。国际惯例是，你接受的名字最好和模块的名字一样。

var http = require("http");

var qs = require("requrystring");

内置模块的引用使用require函数，require表示引用的意思。

* 引用是无条件的，不需要提前再引入什么。
* 引用是无路径的，在任何的目录下，都是通过require("http")来引用内置http模块，而不需要require("../http");

内置模块是Node.js天生就有的，Node.js手册就是按照模块来区分的：



我们后面还会学习一些模块，注意记忆里面的常用方法和属性。比如

url.parse();

querystring.parse();

path.extname();

## 2.2 自定义模块

每一个js文件就是一个模块。**Node.js使用CMD（通用模块定义）规范**。后面我们将知道webpack、seajs也是用的是CMD规范，而Angular、requirejs等等他们使用的是**AMD（异步模块定义）**。

具体的区别我们后面的课程介绍，但是今天你必须去了解Node.js模块的一些特性，而这些特性今后在学习webpack、seajs的时候都是通用的。

### 2.2.1 require谁就会运行谁

结构：

┝ 01.js

┝ test.js

01.js：

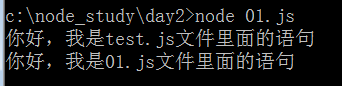
**require("./test.js");**

console.log("你好，我是01.js文件里面的语句");

test.js:

console.log("你好，我是test.js文件里面的语句");

运行01.js:



结论：**require()谁，就会运行谁。**

注意小细节，

require("**./**test.js");

**这里的./坚决不能省略！**为什么不能省略，一会儿你将知道，如果你省略了，表示读取node\_modules文件夹中的内容了。

### 2.2.2 require的js文件中如果有异步函数，将不会等待

结构：

┝ 01.js

┝ test.js

01.js：

require("./test.js");

console.log("你好，我是01.js文件里面的语句");

test.js:

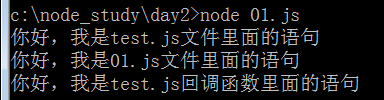
var fs = require("fs");

console.log("你好，我是test.js文件里面的语句");

fs.readFile("./pic.jpg",function(err,data){

console.log("你好，我是test.js回调函数里面的语句");

});



结论：**require的文件中如果有异步语句，此时nodejs不会死等它结束，会返回执行主文件中的程序，如果文件读取完毕，执行回调函数。**

### 2.2.3 连续require

结构：

┝ 01.js

┝ test1.js

┝ test2.js

01.js：

**require("./test1.js");**

console.log("你好，我是01.js文件里面的语句");

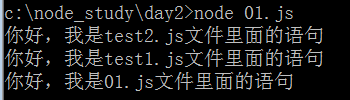
test1.js：

**require("./test2.js");**

console.log("你好，我是test1.js文件里面的语句");

test2.js:

console.log("你好，我是test2.js文件里面的语句");



### 2.2.4 循环引用

如果A引用B，B引用A会发生什么？

先说答案：nodejs很智能，如果B重新引用了A，此时如同没有引用一样，会帮你自动抑制。

结构：

┝ 01.js

┝ test1.js

01.js:

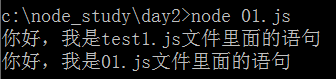
require("./test1.js");

console.log("你好，我是01.js文件里面的语句");

test1.js：

require("./01.js");

console.log("你好，我是test1.js文件里面的语句");



node.js会帮你自动抑制循环引用。

### 2.2.5 文件夹层次

结构：

┝ 01.js

┝ test

│ ┝ test2.js

│ ┝ d

│ │ ┝ test1.js

01.js：

require("./test/d/test1.js");

console.log("你好，我是01.js文件里面的语句");

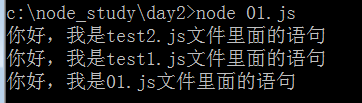
test1.js：

require("../test2.js");

console.log("你好，我是test1.js文件里面的语句");

test2.js：

console.log("你好，我是test2.js文件里面的语句");



## 2.3 js文件天生是隔离作用域的

结构：

┝ 01.js

┝ test1.js

01.js:

require("./test.js");

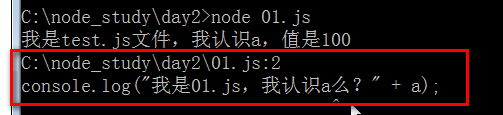
console.log("我是01.js，我认识a么？" + a);

test1.js:

var a = 100;

console.log("我是test.js文件，我认识a，值是" + a);

程序报错：



结论：**在Node.js中，每个js文件是单独的作用域**。和DOM浏览器开发不一样，浏览器中，var a此时a自动成为了window对象的属性，此时js文件和js文件可以共享作用域。但是Node.js中，每个js文件是铜墙铁壁，天生是隔离的作用域的。

## 2.4 使用exports.xx = xx的语法暴露

结构：

┝ 01.js

┝ test1.js

01.js:

**var test =** require("./test.js");

console.log("我是01.js，我认识a么？" + **test.**a);

console.log("我是01.js，我认识b么？" + **test.**b);

console.log("我是01.js，我认识c么？" + **test.**c);

var result = **test.**zonghe();

console.log("总和是" + result);

test1.js:

var a = 100;

var b = 200;

var c = 300;

**exports.**a = a;

**exports.**b = b;

**exports.**c = c;

function zonghe(){

return a + b + c;

}

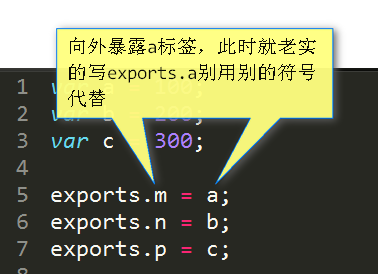
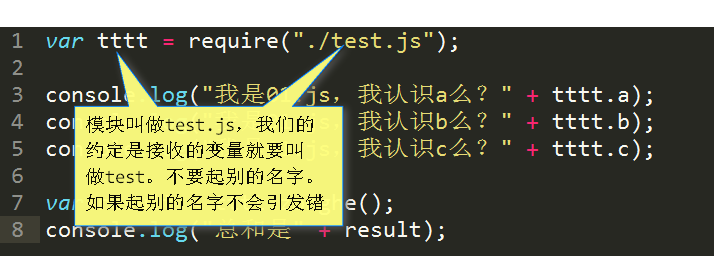
**exports.**zonghe = zonghe;

我们如果想要把变量或者函数向外暴露，此时可以使用exports.\*\* = \*\*向外暴露。

此时引用这个模块的文件将用等号来接收，等号左边的变量将自动成为exports对象。

上面的例子中，var test这个变量就“接收”了exports对象。

严禁玩儿花的：



这种暴露非常好用，通常用于类似的功能向外暴露，比如数学函数：

math-tool.js：

//求和函数，可以计算所有实参列表的和

function sum(){

var result = 0;

for (var i = 0; i < arguments.length; i++) {

result += arguments[i];

}

return result;

}

//平均数函数

function average(){

var \_sum = sum.apply(null,arguments);

return \_sum / arguments.length;

}

//平方函数

function pingfang(a){

return a \* a;

}

//最大值函数

function zuidazhi(){

return Math.max.apply(null,arguments);

}

exports.sum = sum;

exports.average = average;

exports.pingfang = pingfang;

exports.zuidazhi = zuidazhi;

此时外部很好用：

01.js：

var **mt** = require("./math-tool.js");

console.log(**mt.**sum(66,77,88,99));

console.log(**mt.**average(66,77,88,99));

console.log(**mt.**pingfang(66));

console.log(**mt.**zuidazhi(66,77,88,99));

## 2.5 使用module.exports暴露

刚才我们一个js文件中向外暴露了多个东西，比如sum、average、pingfang等等。如果一个js文件中，仅仅向外暴露一个东西，此时用module.exports不方便，比如我们要向外暴露一个类。

结构：

┝ 02.js

┝ People.js

02.js:

var People = require("./People.js");

var xiaoming = new **People.People**("小明","男",12);

xiaoming.sayHello();

People.js:

function People(name,sex,age){

this.name = name;

this.sex = sex;

this.age = age;

}

People.prototype.sayHello = function(){

console.log("你好我是" + this.name + "，我今年" + this.age + "岁了" + "，我是" + this.sex + "的");

}

exports.People = People;



调用函数的时候，要使用People.People()写法，非常的不方便。

原因是什么？因为js文件中你要暴露很多东西。

如果js文件中仅仅暴露一个东西，此时可以用module.exports = xx的写法。

01.js文件改为：

var People = require("./People.js");

var xiaoming = **new People**("小明","男",12);

xiaoming.sayHello();

People.js文件中暴露语法改变了：

function People(name,sex,age){

this.name = name;

this.sex = sex;

this.age = age;

}

People.prototype.sayHello = function(){

console.log("你好我是" + this.name + "，我今年" + this.age + "岁了" + "，我是" + this.sex + "的");

}

module.exports = People;

此时module.exports等号右边的东西，将自动传给02.js中var接受的那个变量。

**用多了，就熟悉了。记住，如果一个js文件要向外暴露多个变量、函数，此时exports.xx = xx;如果仅仅暴露一个东西，通常是构造函数，此时语法为module.exports = Xx;**

## 2.6 神奇的node\_modules文件夹

结构：

┝ 03.js

┝ **node\_modules**

│ ┝ test.js

如果我们写一个引用，没有写./：

03.js文件：

require("test.js");

**此时表示引用node\_modules文件夹中的东西**。

**换句话说，当你引用node\_modules文件夹里面内容的时候，就不需要写./了。**

并且现在更为神奇的是，node\_modules文件夹里面的模块，在引用的时候不需要考虑路径。你只需要确保node\_modules文件夹需要引用模块的js文件里路径的任何祖先路径中。这句话很绕，我们给你解释：

结构：

┝ 03.js

┝ haha

│ ┝ haha.js

┝ **node\_modules**

│ ┝ test.js

03.js：

require("./haha/haha.js");

console.log("我是03.js");

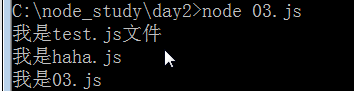
haha.js:

require("test.js"); //注意，这里很嚣张，没有指名具体路径！但是test.js真的就能被找到！

console.log("我是haha.js");

test.js：

console.log("我是test.js文件");



不仅如此，比如我们现在haha.js文件，它不是要引入test.js么？此时node\_modules文件夹出现在任何haha.js的祖先目录中都可以。

下面四个目录中出现了node\_modules文件夹，此时将自动识别test.js文件：

C:\node\_study\day2\haha

C:\node\_study\day2\

C:\node\_study\

C:\

但是

c:\3000soft 文件夹不行，因为不是祖先路径！！

|  |
| --- |
| 有一个路径更神奇，叫做系统环境路径。在：  c:\Users\你的电脑名字\AppData\Roaming\npm\node\_modules  此时在这个文件夹中放入的模块，将在整个计算机中被识别，无论是C盘还是D盘。这个路径是全局路径。 |

当你省略文件名的时候，自动识别index.js文件：

结构：

┝ 03.js

┝ haha

│ ┝ haha.js

┝ **node\_modules**

│ ┝ test

│ │ ┝ index.js

03.js:

require("./haha/haha.js");

console.log("我是03.js");

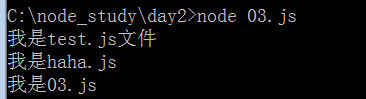
haha.js:

**require("test"); //实际上引入的是node\_modules文件夹中的test文件夹中的index.js文件**

console.log("我是haha.js");

index.js:

console.log("我是test.js文件");



做对这个题目，就说明你已经完全会了。

下列的四种require写法，实际引入的是谁：

require("./a.js")

require("a.js")

require("a")

require("./a")

回答：

require("./a.js") 引入的是当前文件夹中的a.js文件

require("a.js") 引入的是node\_modules文件夹中的a.js文件

require("a") 引入的是node\_modules文件夹中的a文件夹中的index.js文件

require("./a") 引入的是当前文件夹中a文件夹中的index.js文件。

# 三、npm的世界

我们刚才封装了一个数学函数math-tool很好用，此时node开发者也发现了这个事情，**如果让全球开发者贡献自己的实用模块那该多好，让大家不要重复造轮子**。

说一下什么是模块，一个math-tool就是一个模块，但是如果5、6个js文件都是服务于数学方面的，他们整体又成为一个大模块。比如：

│ 01.js

└─ math-tool

┝ base.js

┝ circle.js

┝ index.js

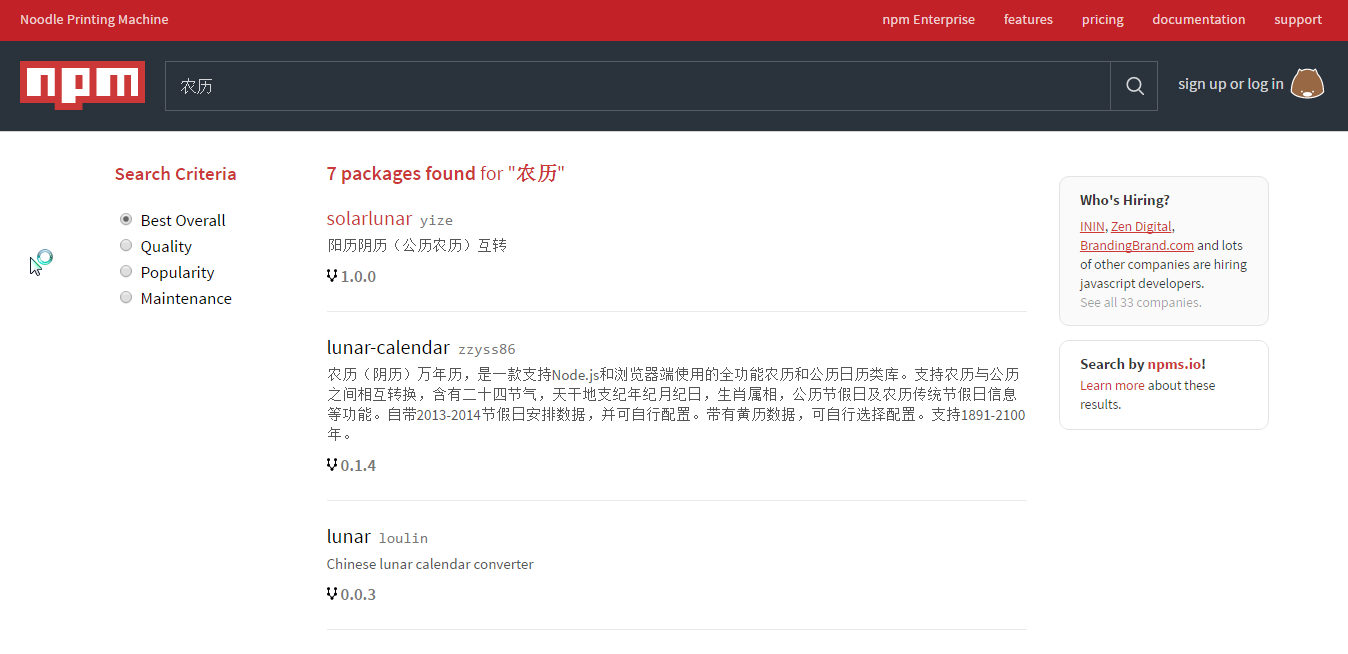
math-tool整体就是一个大模块。

所以node.js主导了一个社区，叫做npm(node package management，node包管理器)。

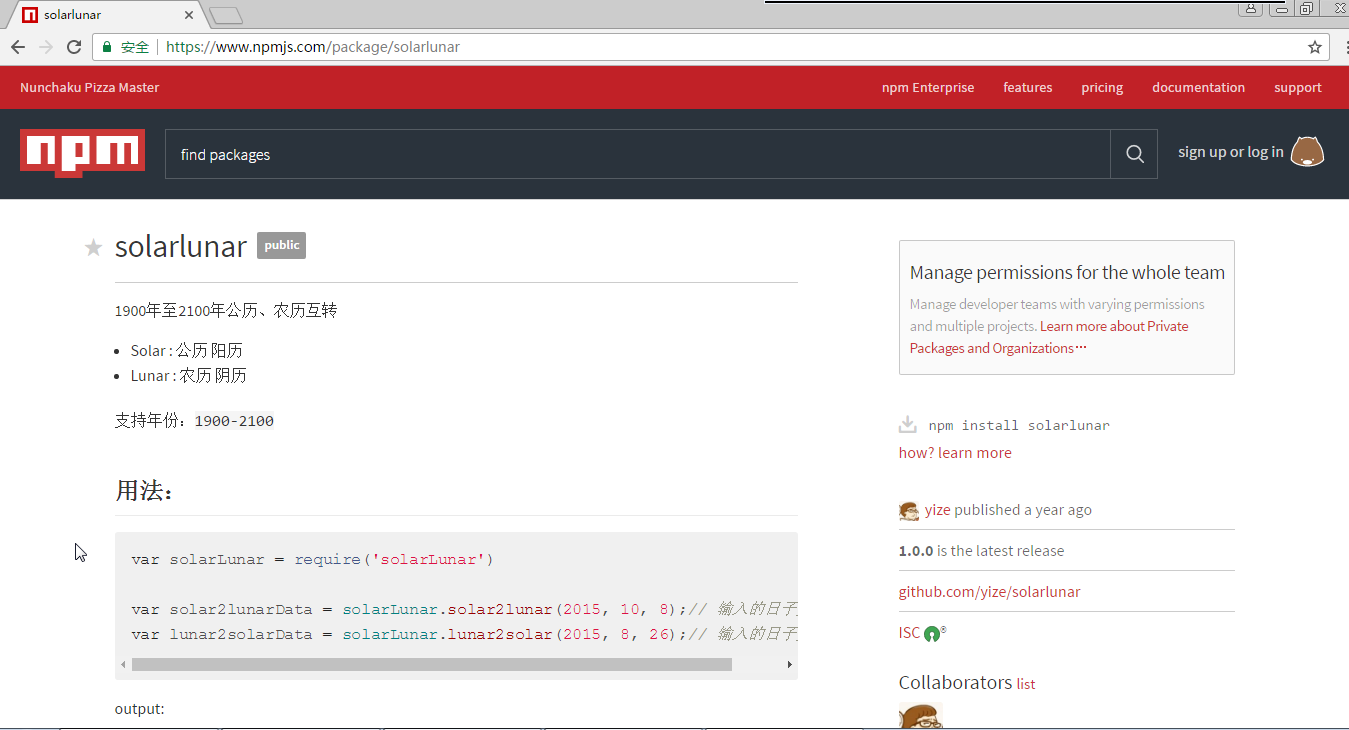
## 3.1 npm的使用

官网：https://www.npmjs.com/

我们可以去npm上寻找我们要的东西，**比如现在有需求，把公历变为农历**。第一时间就要想到，我不要重新造轮子，要去npm社区找找看，在npmjs.com上搜索“农历”：



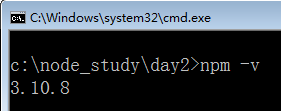
选择一个靠谱的，可以点击进入详情，可以看见API：



直接使用npm命令即可下载。npm是随着node安装的，我们现在可以直接在CMD中使用npm命令。

$ npm -v

查看npm版本。



此时我们看上了一个农历-公历互相转换的npm包：

https://www.npmjs.com/package/solarlunar

此时，可以使用：

$ npm install solarlunar

注意一定要联网环境，此时solarlunar这个模块就会自动的被下载到你项目的node\_modules文件夹中。

非常的神奇！！此时通过查看API（就是上面的网址），我们可以写我们的业务了：

var **solarLunar** = **require('solarLunar');**

var jieguo = **solarLunar**.solar2lunar(2017, 3, 24);

console.log(jieguo);

再比如要把数字转换成财务用的中文大写数字：

$ npm install chinese-finance-number

自己看API：https://www.npmjs.com/package/chinese-finance-number，即可使用：

var solarLunar = require('solarLunar');

**var cfn = require('chinese-finance-number')**

var jieguo = solarLunar.solar2lunar(2017, 3, 24);

console.log(jieguo);

**var jieguo2 = cfn(12345.67);**

**console.log(jieguo2);**

## 3.2 使用package.json管理依赖

很明显，我们现在的项目03.js依赖了两个别人的、从npm上下载的模块。分别是：chinese-finance-number和solarLunar。显然的，node\_modules文件夹是不能删除的。如果删除了这个文件夹，此时将会报错：



比如我们现在用U盘拷贝项目给比人，此时完全没有必要去拷贝node\_modules文件夹，因为这里面的东西在npm线上啊！npm很稳定，你可以随时下载安装。

npm有一个创造性的举动，可以让开发者声明自己的项目的全部依赖。我可以告诉别人这个项目依赖什么。

在项目目录下，创建一个文件，叫做package.json：

{

"dependencies" : {

"solarLunar" : "latest",

"chinese-finance-number" : "latest"

}

}

dependencies是英语依赖的意思。



此时我们在项目根目录（就是package.json这一层目录），在CMD输入：

$ npm install

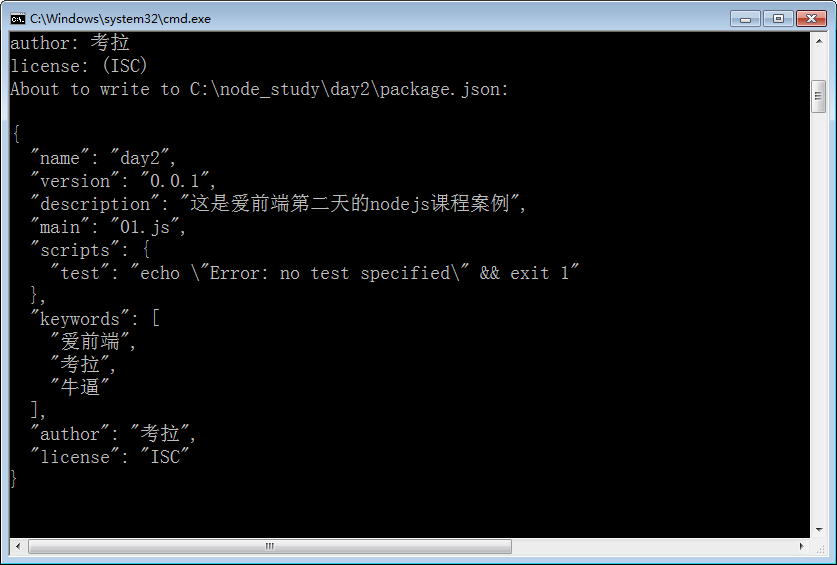
**此时npm软件将会自动安装package.json里面的所有声明的依赖项**。

所以package.json文件非常关键！就是项目的身份证。事实上，里面还有很多配置项。

我们可以用

$ npm init

**此时npm将会有一个表单，引导你去创建一个较为完整的package.json文件**：



此时我们仅仅需要回答一些题目，默认的内容在圆括号中，如果用默认的就直接按回车。

帮我们生成：

{

"name": "day2", //项目名字

"version": "0.0.1", //版本

"description": "这是爱前端第二天的nodejs课程案例", //描述

"main": "01.js", //入口文件

"scripts": { //测试命令，后面使用webpack、gulp、grunt等工程化软件的时候很好用

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [ //关键词，方便其他人在npm上搜索你，当然你得发布

"爱前端",

"考拉",

"牛逼"

],

"author": "考拉", //作者

"license": "ISC" //版权协议

}

**但是dependencies没有生成，你自己补充一项即可**：

{

"name": "day2",

"version": "0.0.1",

"description": "这是爱前端第二天的nodejs课程案例",

"main": "01.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [

"爱前端",

"考拉",

"牛逼"

],

"author": "考拉",

"license": "ISC",

"dependencies" : {

"solarLunar" : "latest",

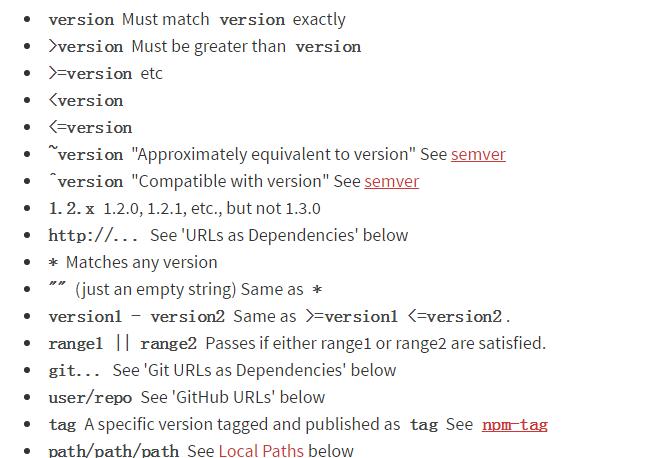
"chinese-finance-number" : "latest"

}

}

版本有很多符号可以表示你要依赖的版本是什么。

我们看看npm的手册：https://docs.npmjs.com/files/package.json



常用的举例：

"dependencies" : {

"aa" : "1.0.0", //必须明确限制为1.0.0版本

"bb" : "~1.0.0" , //大约近似等于1.0.0版本，如果有这个版本，一定使用这个版本。如果没有找最近的。

"cc" : "^1.0.0" , //寻找和1.0.0版本兼容的版本，如果有1.0.0就用，如果没有就找近似的。和~实测是一样的。

"dd" : "^1.0.0" , //寻找和1.0.0版本兼容的版本，如果有1.0.0就用，如果没有就找近似的。和~实测是一样的。

"ee" : "1.2.x",

"ff" : "\*", //匹配任何版本，实测和latest效果一样，会寻找最新版本

"gg" : "laste" //寻找最新版本

}

如果我们安装一个依赖的时候，想它自动进入到package.json中，此时使用--save参数即可。

$ npm install solarLunar **--save**

或者

$ npm install --save **solarLunar**

此时依赖将被安装，并且会自动将solarLunar写入package.json文件中的dependencies对象中。

如果想要限制版本：

$ npm install solarLunar#^1.0.0 --save

至此我们已经学习了很多npm命令：

$ npm init //帮我们创建一个package.json文件的，项目开发的第一件事儿就是这个。

$ npm install //根据package.json文件，安装全部的项目依赖。

$ npm install solarLunar //指名道姓安装某一个

$ npm install solarLunar --save //指名道姓安装某一个，并且把它写入package.json依赖项中。

## 3.3 全局安装

$ npm install 包名字 -g

此时这个包（模块）将被安装在全局，就是上午说的：

C:\Users\Danny\AppData\Roaming\npm

**一些CLI（命令行程序）、工程化的东西将被安装在全局。随着学习的深入，我们将遇见**。

## 3.4 淘宝镜像

淘宝为了方便中国程序员对抗GFW，此时发明了cnpm。你可以用此代替官方版本(只读)，同步频率目前为 10分钟 一次以保证尽量与官方服务同步。

如何设置淘宝呢？一句话：

$ npm install -g cnpm --registry=https://registry.npm.taobao.org

今后所有的npm活动，都使用cnpm代替。感谢马爸爸！

最后说一句，有很多包非常大，自己也依赖别人，不用怕，他们的依赖也写在了他们自己的package.json里面。npm、cnpm能够深入读取每个人的package.json，将他们的依赖都下载下来。

npm是个怎么样的世界？就是你站在巨人的肩膀上，他也站在别人的肩膀上……。

# 四、GET请求和POST请求

## 4.1 GET请求

GET请求传参数通过URL，POST请求传参数通过HTTP上行报文。GET请求不安全，方便分享网址；POST请求安全，不方便分享网址，内容信息无限长。

get请求的参数在URL中

http://127.0.0.1/addStudent?id=10001&name=小明&sex=男

nodejs程序如果获得get请求，实际上就是如何解析URL。昨天已经讲解过了。

就是：

url.parse(req.url,true).query

安装一个小型服务器，因为只有服务器环境才能发出Ajax请求。

$ cnpm install serve-static --save

$ cnpm install finalhandler --save

GET请求的识别非常的简单，就是URL的识别，使用的是url.parse(req.url,true).query。

var finalhandler = require('finalhandler')

var http = require('http')

var serveStatic = require('serve-static');

var url = require("url");

var fs = require("fs");

// 配置静态资源服务器，将public文件夹静态化出来

var serve = serveStatic('public', {'index': ['index.html', 'index.htm']})

// Create server

var server = http.createServer(function onRequest (req, res) {

//路由

var pathname = url.parse(req.url).pathname;

if(pathname == "/addStudent"){

//拿到GET请求参数，就是拿到地址栏中的东西。

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

// console.log(queryJSON.name);

// console.log(queryJSON.age);

// console.log(queryJSON.sex);

//准备要写入文本文件的内容

var data = "姓名：" + queryJSON.name + "\r\n";

data += "年龄：" + queryJSON.age + "\r\n";

data += "性别:" + queryJSON.sex + "\r\n";

//我们写文件

fs.writeFile("./student\_data/" + queryJSON.name + ".txt",data,function(err){

if(err){

res.end("cuowu");

}else{

res.end("chenggong");

}

});

return;

}

//使用静态资源

serve(req, res, finalhandler(req, res));

});

// Listen

server.listen(3000,"127.0.0.1");

console.log("服务器已经运行在3000端口");

## 4.2 POST请求

node.js识别post请求的参数有一个标准的语法，就是写两个on参数监听data事件和end事件。

**// 定义了一个content变量，用于暂存请求体的信息**

var content = "";

**// post请求经常会很长，此时会分段，网上有文章大约800k左右是一个HTTP报文段**

req.on('data', function(chunk){

console.log("收到了一个段落")

//将段落合并

content += chunk;

});

**// 当所有数据发送完毕之后，此时将会触发end事件**

req.on('end', function(){

content = querystring.parse(content);

res.end("chenggong");

});

我们马上用插件替代这种写法，所以不需要详细了解。

## 4.3 formidable模块

npm上可以下载一个formidable的模块，用来处理post请求。甚至可以处理图片、zip文件等的上传。

**API文档：https://npm.taobao.org/package/formidable**



安装依赖：

$ cnpm install --save formidable

然后通过看API，来学习使用它，这些代码可以随时通过API文档的来复制粘贴，不要背诵！

//创建一个表单的实例，这是formidable的API

//**请看https://npm.taobao.org/package/formidable**

var form = new formidable.IncomingForm();

//设置上传的文件存放在哪里

form.uploadDir = "./uploads";

//处理表单

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

//fields 表示普通控件

//files 表示文件控件

//验证是否有wenjian这个控件，为了鲁棒考虑的

if(!files.wenjian){

return;

}

//验证传输的文件有名字，为了鲁棒考虑的

if(!files.wenjian.name){

return;

}

//拓展名

var extname = path.extname(files.wenjian.name);

//改名，因为formidable天生传输的文件是没有拓展名的

//fs的rename方法用来改名。files.wenjian.path就是它默认的路径+文件名，改名为默认的路径+文件名+拓展名

fs.rename(files.wenjian.path , files.wenjian.path + extname , function(){

res.end("上传成功！");

});

});

# 五、Express.js框架

传统node.js缺点：

路由不方便制作，尤其是正则表达式路由

静态资源服务不方便

页面呈递不方便，没有模板引擎

官网：http://www.expressjs.com.cn/



## 5.1 基本使用

$ cnpm install express --save

然后就可以写基本的demo了，基本上都是固定动作，没有什么好解释的：

var express = require("express");

var app = express();

**//路由清单，术语叫做“中间件”**

app.get("/",function(req,res){

res.send("我是首页");

});

app.get("/music",function(req,res){

res.send("音乐频道");

});

app.get("/news",function(req,res){

res.send("新闻频道");

});

app.get("/student/:xuehao",function(req,res){

res.send("学生频道，学号" + req.params.xuehao);

});

**//监听**

app.listen(3000);

看一下中间件的特写：

app.get("/news",function(req,res){

res.send("新闻频道");

});

表示当你用get请求访问/news的时候做的事情。此时输出使用res.send()而不是res.end()。好处是自动变为utf-8编码。

并且express是自动使用pathname与/news进行比较的，也就是说会自动过滤quertring、hash等等。

我们可以用:来引导一个变量，此时特别方便做一个匹配正则模式的路由：

app.get("/student/**:xuehao**",function(req,res){

res.send("学生频道，学号" + **req.params.xuehao**);

});

更复杂的可以验证学号的正确性：

app.get("/student/**:xuehao**",function(req,res){

if(**/^[\d]{6}$/.test(req.params.xuehao)**){

res.send("学生频道，学号" + req.params.xuehao);

}else{

res.send("学号不正确，应该是6位数字！");

}

});

## 5.2 静态路由

一个中间件解决问题：

app.use(**"/jingtai"**,express.static("**public**"));

use表示使用中间件，刚才的get表示在get请求下使用中间价。use是无论什么请求都会使用。

此时我们放置在物理文件夹**public**文件夹中的luhan.png图片，将被路由到：

http://127.0.0.1:3000/**jingtai/**luhan.png

## 5.3 使用某一个页面

比如当我们访问/的时候，我们就想要呈递一个public/haha.html页面，这是一个经常的事情。

此时不需要那个逻辑了：

fs.readFile("...",function(){

....

)};

而是：

app.get("/",function(req,res){

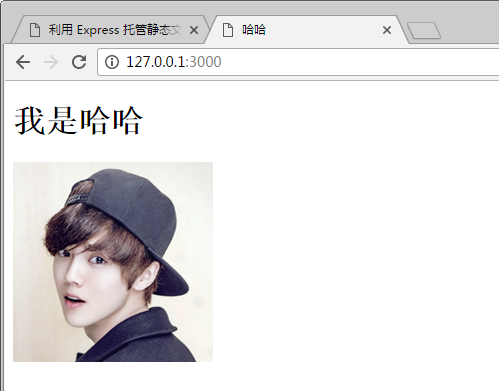
//res常用的函数是send和sendFile。sendFile表示发送一个页面给用户。

//注意，必须使用绝对路径当做参数

//\_\_dirname表示当前这个js文件的绝对路径，

res.sendFile(\_\_dirname + "/public/haha.html");

});



## 5.4 模板引擎

express可以像php一样使用后台语言模板，此时最好用的模板引擎叫做ejs模板引擎。

全程：Embedded JavaScript templates， 嵌入式JS模板。

API ： https://npm.taobao.org/package/ejs

$ cnpm install ejs --save

我们写一个08.js案例，是一个标准express程序：

var express = require("express");

var app = express();

//设置默认的模板引擎，此时express将帮你引入ejs，所以你一定要提前

//cnpm install ejs --save

**app.set("view engine","ejs");**

app.get("/",function(req,res){

**res.render("shouye",{**

**"dongxi" : "八个手机",**

**"qian" : 1000 \* 8**

**});**

});

app.listen(3000);

**此时当用户访问/的时候，会自动使用views文件夹中的shouye.ejs文件当做模板。字典就是后面传入的json。**

views文件夹中的shouye.ejs文件：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

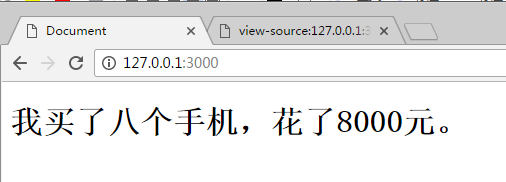
<body>

<h1>我买了<%= dongxi %>，花了<%= qian %>元。</h1>

</body>

</html>

呈递的时候：



模板是服务器填充的，访问者休想看见源代码！！

模板引擎很灵活，比如：

<select name="" id="">

<%

for(var i = 1949 ; i <= 2017 ; i++){

%>

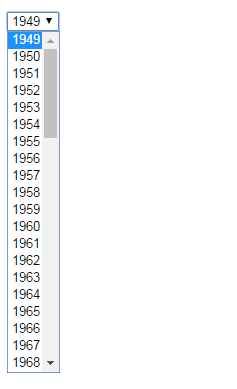
<option value="<%= i %>"><%= i %></option>

<%

}

%>

</select>



很方便，任何JS中能够用的数据结构都可以往后台传：

app.get("/",function(req,res){

res.render("shouye",{

"dongxi" : "八个手机",

"qian" : 1000 \* 8 ,

"ouxiang" : [

{"xingming" : "王俊凯" , "nianling" : 16},

{"xingming" : "王宝强" , "nianling" : 26},

{"xingming" : "宋小宝" , "nianling" : 36}

]

});

});

<h4>我的偶像：</h4>

<%

for(var i = 0 ; i < **ouxiang**.length ; i++){

%>

<p><b><%= **ouxiang**[i].xingming %></b> <%= **ouxiang**[i].nianling %>岁</p>

<%

}

%>