Node.js入门和企业级开发实战

第2天课堂笔记（本课程共8天）

# 目录

[目录 2](#_Toc25559)

[一、复习 3](#_Toc7870)

[二、模块 4](#_Toc23543)

[三、post请求 7](#_Toc9453)

[四、模板引擎 8](#_Toc10459)

[先说EJS 8](#_Toc7828)

# 一、复习

复习：

Node.js开发服务器，数据、路由。本地关心的效果，交互；

Node.js实际上是极客开发出的一个小玩具，不是银弹。有着别人不具备的怪异特点：

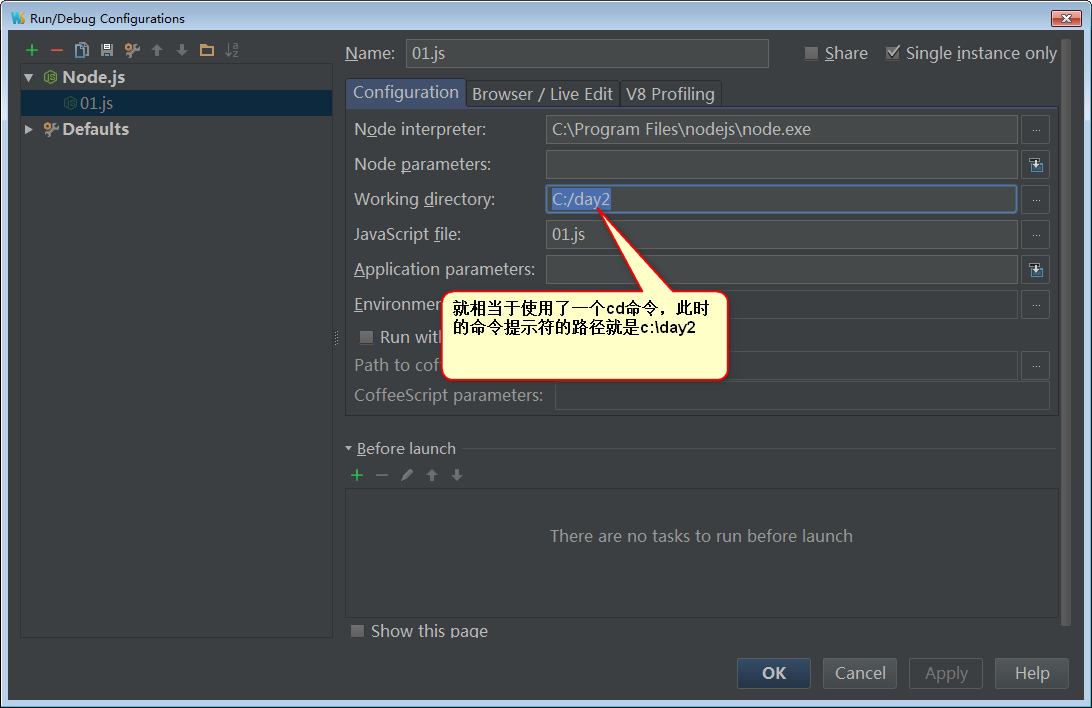
单线程、Non-blocking I/O、Event Driven。 实际上是一个特点。

首先，Node不为每个用户开辟一个线程，所以非常极端的选择了单线程。单线程，要照顾所有的用户，那么就必须有非阻塞I/O，否则一个人的I/O就把别人、自己都阻塞了。一旦有非阻塞I/O，一个人如果I/O去了，就会放弃CPU的使用权，换成另一个人使用CPU（或者执行此人后面的语句）。所以CPU的利用率100%。第一个人I/O结束了，就要用事件来通知线程，执行回调函数。此时必须有事件环，就有一个排队调度机制。Node中有超过半数的C++代码，在搭建事件环。

Node.js和别的老牌3P不一样：

1） 没有自己的语法，使用V8引擎，所以就是JS。V8引擎解析JS的，效率非常高，并且V8中很多东西都是异步的。Node就是将V8中的一些功能自己没有重写（别人做了，自己就站在巨人肩膀上），移植到了服务器上。

2） 没有web容器，就是安装配置完成之后，没有一个根目录。



命令提示符所在路径太重要了，因为程序中的所有相对路径”./”，都是相对这个命令提示符路径的，而不是相对于js文件自己。

系统中，80端口，就是默认http端口。所以当没有端口号的时候，就是80端口。

|  |
| --- |
| 1. server.listen(80,"127.0.0.1"); |

# 二、模块

● 在Node.js中，以模块为单位划分所有功能，并且提供了一个完整的模块加载机制，这时的我们可以将应用程序划分为各个不同的部分。

不可能用一个js文件去写全部的业务。肯定要有MVC。

● **狭义**的说，每一个JavaScript文件都是一个模块；而多个JavaScript文件之间可以相互require，他们共同实现了一个功能，他们整体对外，又称为一个**广义**上的模块。

● **Node.js中，一个JavaScript文件中定义的变量、函数，都只在这个文件内部有效**。当需要从此JS文件外部引用这些变量、函数时，必须使用exports对象进行暴露。使用者要用require()命令引用这个JS文件。

foo.js文件中的代码：

|  |
| --- |
| 1. var msg = "你好"; 2. exports.msg = msg; |

msg这个变量，是一个js文件内部才有作用域的变量。

**如果别人想用这个变量，那么就要用exports进行暴露**。

使用者：

|  |
| --- |
| 1. var **foo** = require("./test/foo.js"); 2. console.log(foo.msg); |

使用者用foo来接收exports对象，也就是说，这里的foo变量，就是文件中的exports变量。

● 一个JavaScript文件，可以向外exports无数个变量、函数。但是require的时候，仅仅需要require这个JS文件一次。使用的它的变量、函数的时候，用点语法即可。所以，无形之中，增加了一个顶层命名空间。

js文件中，可以用exports暴露很多东西，比如函数、变量。

|  |
| --- |
| 1. var msg = "你好"; 2. var info = "呵呵"; 3. function showInfo(){ 4. console.log(info); 5. } 6. exports.msg = msg; 7. exports.info = info; 8. exports.showInfo = showInfo; |

在使用者中，只需要require一次。

|  |
| --- |
| 1. var foo = require("./test/foo.js"); |

相当于增加了顶层变量。所有的函数、变量都要从这个顶层变量走：

|  |
| --- |
| 1. console.log(foo.msg); 2. console.log(foo.info); 3. foo.showInfo(); |

Node中，js文件和js文件，就是被一个个exports和require构建成为网状的。

不是靠html文件统一在一起的。

● 可以将一个JavaScript文件中，描述一个类。用

module.export = 构造函数名;

的方式向外暴露一个类。

也就是说，js文件和js文件之间有两种合作的模式：

1） 某一个js文件中，提供了函数，供别人使用。 只需要暴露函数就行了； exports.msg=msg;

2） 某一个js文件，描述了一个类。 module.exports = People;

● 如果在require命令中，这么写:

|  |
| --- |
| 1. var foo = require("foo.js"); //没有写./， 所以不是一个相对路径。是一个特殊的路径 |

那么Node.js将该文件视为node\_modules目录下的一个文件

● node\_modules文件夹并不一定在同级目录里面，在任何直接祖先级目录中，都可以。甚至可以放到NODE\_PATH环境变量的文件夹中。这样做的好处稍后你将知道：分享项目的时候，不需要带着modules一起给别人。

● 我们可以使用文件夹来管理模块，比如

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

那么Node.js将会去寻找node\_modules目录下的bar文件夹中的index.js去执行。

每一个模块文件夹中，推荐都写一个package.json文件，这个文件的名字不能改。node将自动读取里面的配置。有一个main项，就是入口文件：

|  |
| --- |
| 1. { 2. "name": "kaoladebar", 3. "version": "1.0.1", 4. **"main" : "app.js"** 5. } |

package.json文件，要放到模块文件夹的根目录去。

我们刚才学习了，模块就是一些功能的封装，所以一些成熟的、经常使用的功能，都有人封装成为了模块。并且放到了社区中，供人免费下载。

这个伟大的社区，叫做npm。 也是一个工具名字 node package management

<https://www.npmjs.com/>

去社区搜索需求，然后点进去，看api。

如果要配置一个模块，那么直接在cmd使用

|  |
| --- |
| 1. npm install 模块名字 |

就可以安装。 模块名字全球唯一。

安装的时候，要注意，命令提示符的所在位置。

1.我们的依赖包，可能在随时更新，我们永远想保持更新，或者某持某一个版本；

2.项目越来越大的时候，给别人看的时候，没有必要再次共享我们引用的第三方模块。

我们可以用package.json来管理依赖。

在cmd中，使用npm init可以初始化一个package.json文件，用回答问题的方式生成一个新的package.json文件。

使用

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

将能安装所有依赖。

npm也有文档，这是package.json的介绍：

<https://docs.npmjs.com/files/package.json>

**require()别的js文件的时候，将执行那个js文件。**

注意：

**require()中的路径，是从当前这个js文件出发，找到别人。而fs是从命令提示符找到别人。**

所以，**桌面上有一个a.js， test文件夹中有b.js、c.js、1.txt**

a要引用b：

|  |
| --- |
| 1. var b = require(“./test/b.js”); |

b要引用c：

|  |
| --- |
| 1. var b = require(“./c.js”); |

但是，fs等其他的模块用到路径的时候，都是相对于cmd命令光标所在位置。

所以，在b.js中想读1.txt文件，推荐用绝对路径：

|  |
| --- |
| 1. fs.readFile(**\_\_dirname** + "/1.txt",function(err,data){ 2. if(err) { throw err; } 3. console.log(data.toString()); 4. }); |

# 三、post请求

|  |
| --- |
| 1. var alldata = ""; 2. //下面是post请求接收的一个公式 3. //node为了追求极致，它是一个小段一个小段接收的。 4. //接受了一小段，可能就给别人去服务了。防止一个过大的表单阻塞了整个进程 5. req.addListener("data",function(chunk){ 6. alldata += chunk; 7. }); 8. //全部传输完毕 9. req.addListener("end",function(){ 10. console.log(alldata.toString()); 11. res.end("success"); 12. }); |

原生写POST处理，比较复杂，要写两个监听。文件上传业务比较难写。

所以，用第三方模块。formidable。

只要涉及文件上传，那么form标签要加一个属性：

|  |
| --- |
| 1. <form action="http://127.0.0.1/dopost" method="post" enctype="multipart/form-data"> |



# 四、模板引擎

|  |
| --- |
| 1. <a href="<%= url %>"><img src="<%= imageURL %>" alt=""></a> |

数据绑定，就成为一个完整的html字符串了。

前台的模板，我们现在要学习的是后台的模板。

后台模板，著名的有两个，第一个叫做ejs； 第二个叫做jade。

是npm第三方包。

## 先说EJS

Embedded JavaScript templates

后台模板引擎

|  |
| --- |
| 1. <ul> 2. **<% for(var i = 0 ; i < news.length ; i++){ %>** 3. <li>**<%= news[i] %>**</li> 4. **<% } %>** 5. </ul> |

|  |
| --- |
| 1. var dictionary = { 2. a:6, 3. news : ["1期班太牛逼了","高薪就业","哈哈哈哈哈"] 4. }; |