

NODE.JS

node.js是基于v8引擎运行js的环境，它不是一门语言，你可以把它理解为谷歌浏览器，它是工具

为什么把node称之为后台语言？

客户端向服务器端发送请求，服务器端需要编写相关的程序，把客户端请求的内容准备好，然后返回给客户端

我们可以使用java/php等语言编写这些程序，同样也可以使用js编写这些操作（js是全栈编程语言，它可以写后台的程序了）

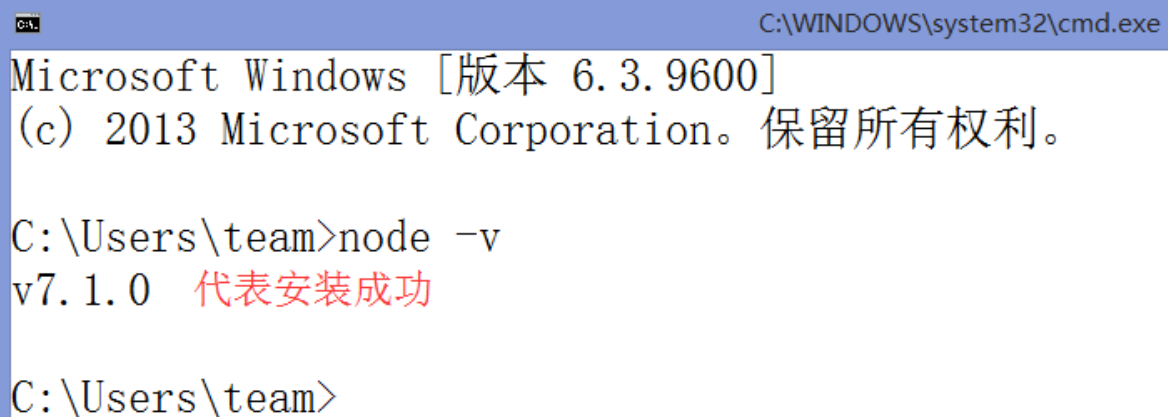
js代码写完后，我们要把它运行，此时我们在服务器上安装一个node工具，使用node可以把这些代码执行，从而让其具备相关的功能即可

安装node.js

<http://nodejs.cn/> 中文

<https://nodejs.org/en/> 英文

安装的时候基本上一路下一步即可，默认情况下，**node**安装成功后，会把相关的操作命令集成到系统的**dos**命令中（**MAC**是终端），以后我们可以在**DOS(终端)**命令中执行**node**的命令



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation。保留所有权利。

C:\Users\team>node -v
v7.1.0 代表安装成功

C:\Users\team>
```

node.js的基础知识

基于node环境使用js编写后台程序，相对于传统的后台语言具备了一些优势

- 基于V8引擎渲染解析代码（快）
- 基于事件驱动的非阻塞I/O操作
- 采用的是异步单线程开发
- npm包管理器，是全球最大的开源库生态系统：<https://www.npmjs.com/>
- 对于前端开发工程师来说，学习成本低，可以快速入手这门技术

NODE中的模块

node其实就是由很多模块拼起来的

- 内置模块：node环境本身自带的（类似于浏览器也会天生自带一些自己的方法）
- 自定义模块：开发者自己编写的
- 第三方模块：别人写好的模块，我们可以下载下来使用（类似于在客户端导入JQ）

我们需要使用的第三方模块，在npmjs.com中都可以获取到；而模块的下载安装统一使用npm包管理器完成；

NPM以及第三方模块

`npm install xxx -g` 把模块安装在全局环境下

`npm install xxx --global`

浏览器中的全局对象是`window`，NODE中的全局对象是`global`

`npm install xxx` 把模块安装在当前操作的项目目录下

`npm install xxx --save-dev` 不仅安装在当前下，并且把安装的信息记录在项目的`package.json`清单中，生成一条开发环境依赖项

`npm install xxx --save` 和上述操作类似，不过生成的是一条生产环境依赖项

`npm uninstall xxx -g/--save-dev/--save` 相当于安装来说，当前操作为卸载这些模块

`npm install jquery@1.11.3` 在安装这个模块的时候，指定了安装的版本号

...

1、安装在全局和安装在本地项目中的区别

安装在全局

```

C:\Users\team>npm install less --global
C:\Users\team\AppData\Roaming\npm\lessc ->
e_modules\less\bin\lessc
- assert-plus@1.0.0 node_modules\less\node_
plus
- assert-plus@1.0.0 node_modules\less\node_
lus
- assert-plus@1.0.0 node_modules\less\node_
us
- assert-plus@1.0.0 node_modules\less\node_
s
- assert-plus@1.0.0 node_modules\less\node_
us
安装在全局下的目录
C:\Users\team\AppData\Roaming\npm
-- less@2.7.2
-- request@2.82.0

```

安装在全局下，会在安装的全局目录中生成一个文件：

`lessc.cmd` -> 可以在DOS中执行命令的文件，此时我们就可以在DOS中执行 `lessc` 这个命令了

```

lessc.cmd
1 @IF EXIST "%~dp0\node.exe" (
2   "%~dp0\node.exe" "%~dp0\node_modules\less\bin\lessc" %*
3 ) ELSE ( 执行lessc命令，其实就是在node环境下，把指定的JS文件
4   @SETLOCAL中的代码给执行了（实现想要的功能）
5   @SET PATHEXT=%PATHEXT:;.JS;=%
6   node "%~dp0\node_modules\less\bin\lessc" %*
7 )

```

安装在全局环境下的模块可以使用命令来操作，但是只能使用命令操作，如果想把这个安装的模块导入到我们自己的JS代码中使用，则是不可以的

安装在项目中

```
npm install less
```

安装完成后在当前的项目目录下多了一个文件夹：`node_modules`，此文件夹中包含了我们安装的less模块

安装在本地项目中的模块无法使用命令来操作（默认情况下）；但是可以在当前项目的JS代码中，通过require把它导入进来，然后在js代码中调取模块中的方法，实现一些特殊的处理；

```
1. let lessc= require('less');  
2. lessc.render();
```

2、能否有办法即能使用命令也能导入到JS中？

真实项目开发的时候，我们很少安装在全局，因为安装在全局可能导致版本的冲突，一般我们都安装在本地项目中

在本地项目中配置模块的运行命令

```
npm init -y
```

在本地项目中生成一个 `package.json` 文件：项目的配置文件（命令中不加 `-y`，需要自己在执行的时候一个个的输入配置信息，加 `-y` 一切都走默认的信息，比较方便快捷）

```
1.  {
2.    "name": "BASE-INFO", //->项目名称
3.    "version": "1.0.0", //->项目版本
4.    "description": "", //->项目的描述
5.    "main": "index.js", //->项目的入口页面(首页面)
6.    "dependencies": { //->生产环境依赖模块清单
7.      "less": "^2.7.2"
8.    },
9.    "devDependencies": {}, //->开发环境依赖模块清单
10.   "scripts": { //->项目的命令脚本配置信息
11.     "test": "echo \"Error: no test specified\" && exit 1"
12.   },
13.   "keywords": [], //->关键词
14.   "author": "", //->作者
15.   "license": "ISC" //->监听模式
16. }
```

在生成的package.json文件中的 `scripts` 属性中，配置我们需要运行的命令

```
1. {
2.     ...
3.     "scripts": {
4.         // -> zxt是自己定义的属性名（随便起）
5.         // -> 属性值是我们即将要执行的命令（使用的就是less的命令）
6.         "zxt": "lessc LESS/index.less CSS/index.min.css -x"
7.     }
8.     ...
9. }
```

配置完成后，接下来在当前项目的DOS命令中执行：`npm run zxt`

```
E:\201708\WEEK9\BASE-INFO>npm run zxt
```

```
> BASE-INFO@1.0.0 zxt E:\201708\WEEK9\BASE-INFO
> lessc LESS/index.less CSS/index.min.css -x
```

相当于执行run zxt的时候，把zxt的属性值，在DOS命令中给执行了，而属性值就是把某个less编译成css的命令

3、生产环境和开发环境

开发环境：项目在本地开发的时候，所需要依赖的模块叫做开发依赖项

生产环境：项目开发完成部署到服务器上，所需要依赖的模块叫做生产环境依赖项

less模块，开发的时候需要依赖，项目部署后不需要依赖；开发的时候需要安装**less**模块，项目上线则不需要安装；

`npm install less --save-dev` 安装less模块并且把安装的信息存放在开发依赖项中

`npm install less --save` 安装less模块并且把安装的信息存放在生产依赖项中

为啥要设置依赖项？

项目如果是多人开发，我们使用git仓库来管理项目代码以及实现团队协作开发

A是其中的一个开发人员，开发这个项目需要用到很多模块，A在自己的电脑上已经把需要的模块都安装在本地项目中了（`node_modules`）

A在提交自己的代码到git仓库的时候，会忽略`node_modules`文件夹的提交：因为这个文件中的内容太大了（当前项目增加`.gitignore`文件）

B从git仓库下载代码，代码都有了，但是开发需要依赖的模块没有，项目无法运行，此时B也需要安装这些模块

1）找到A，手动记录一下需要的模块，然后B自己在本地一个个的安装（太low了）

2）此时体现出我们配置依赖项的好处了，A在他本地安装的时候，把安装的信息都记录到

`package.json`的`devDependencies/dependencies`这里面，虽然`node_modules`没有传递到git仓库中，但是`package.json`传递上去了，B下载完成后，在本地的`package.json`中可以看到需要依赖的模块信息，此时的B只需要执行：`npm install` 命令，就可以把当前项目需要依赖的模块自动的都安装上，我们把这个操作叫做 跑环境

项目上线也是同样的原理

4、安装指定版本的模块

查询当前模块的版本号

```
npm view webpack
```

 查看webpack的版本号

```
npm view webpack > version.webpack
```

 由于查看版本号的时候信息太多，命令窗口中可能放不下，此时我们把查看的信息放在version.webpack这个文件中（文件名可以自己定义，不一定叫做version.webpack）

```
npm install webpack@1.15.0 --save-dev
```

 安装指定版本的模块

NODE中的自定义模块

在拿JS写后台（基于NODE环境）的时候，我们在项目中创建某一个JS文件来实现后台的一些业务逻辑处理，此时新创建的JS被称为一个‘自定义模块’；

模块和模块之间是独立的，里面的方法不会产生冲突，当然也可以基于NODE提供的一些语法，实现模块与模块之间的调用

1、在当前模块中引入其它模块，从而调用其它模块中的方法

require

```
require('xxx')
```

导入某个模块，但是这样写是导入已经安装的第三方模块或者导入**NODE**内置的模块，这样写完后，它首先会到本地项目的**node_modules**中查找这个模块，如果有就是已经安装的第三方模块，如果没有继续查找**NODE**的内置模块，如果再没有就会报错

```
require('./xxx') 或者 require('../xxx') 或者
```

```
require('./stu/xxx') ...
```

这类模式都属于引入模块的时候指定了路径，此时他们的意思是导入自定义的模块（需要注意的是：路径地址一定要指定好）

/ 当前项目的根目录

./ 当前目录

../ 上一级目录

```
let temp = require('./TEST1')
```

我们导入模块后，可以使用一个变量来接收它的返回值

module.exports

`module`是`node`环境中天生自带的一个对象，用来进行`node`模块管理的

它里面有一个对象叫做`exports`，把模块中的部分方法导出，提供给其它的模块调取使用

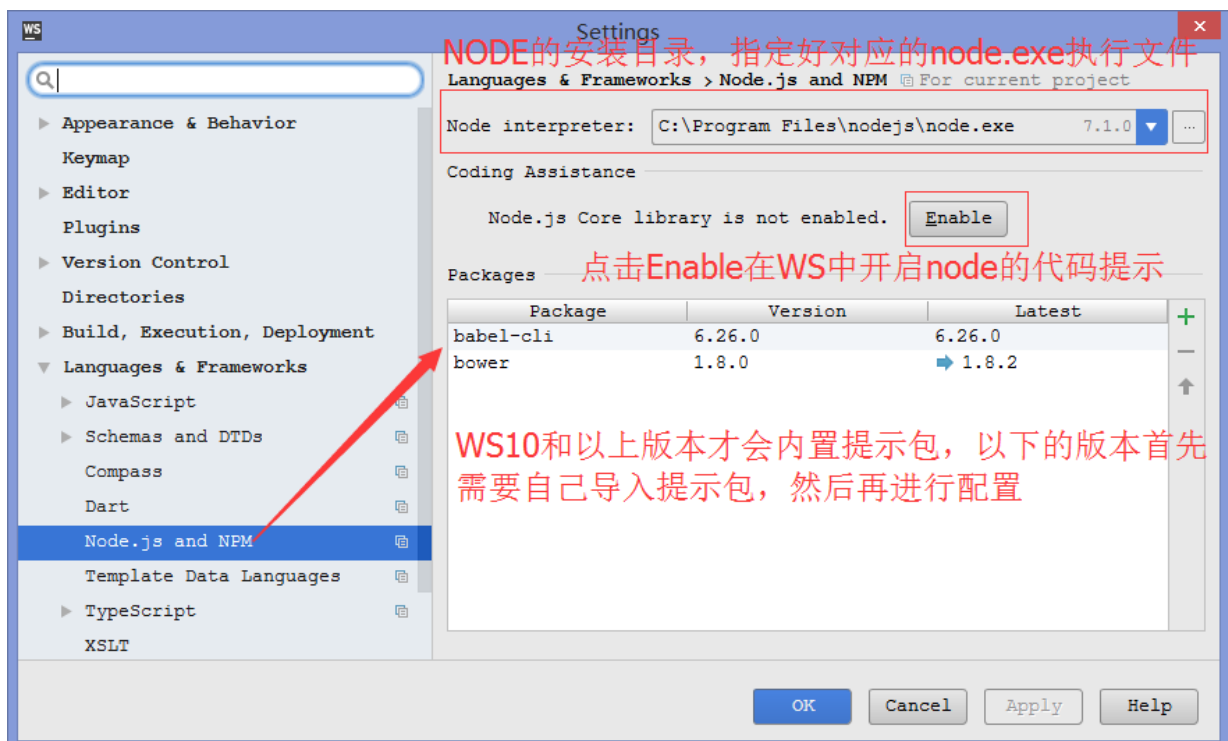
TEST1.js

```
1. let fn = ()=> {  
2.     console.log('TEST1');  
3. };  
4.  
5. let sum = (a, b)=> {  
6.     console.log(a + b);  
7. };  
8.  
9. // module.exports.sumFn = sum;  
10. // module.exports.fn = fn;  
11. module.exports = {  
12.     sum: sum,  
13.     fn: fn  
14. };
```

TEST2.js

```
1. let fn = ()=> {
2.     console.log('TEST2');
3. };
4.
5. //->需求：在TEST2模块中调用TEST1模块中的
    某一个方法
6. //TEMP存储的内容就是TEST1模块导出的对象
7. let temp = require('./TEST1');
8. temp.sum(1, 2);
```

2、在WS中配置语言提示包(基于NODE环境JS开发后程序需要的语言提示)



3、在NODE环境下执行js代码

webstorm

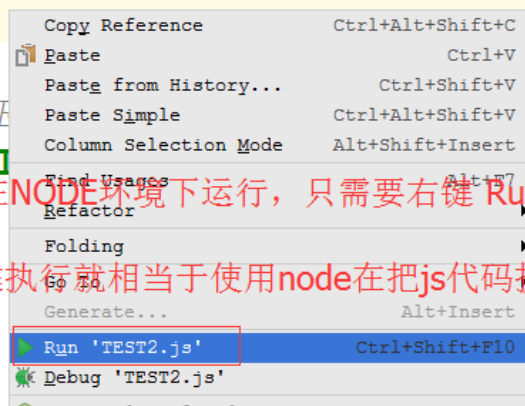
```
let fn = () => {
  console.log('TEST2');
};
```

// -> 需求: 在TEST2模块中调用TEST1

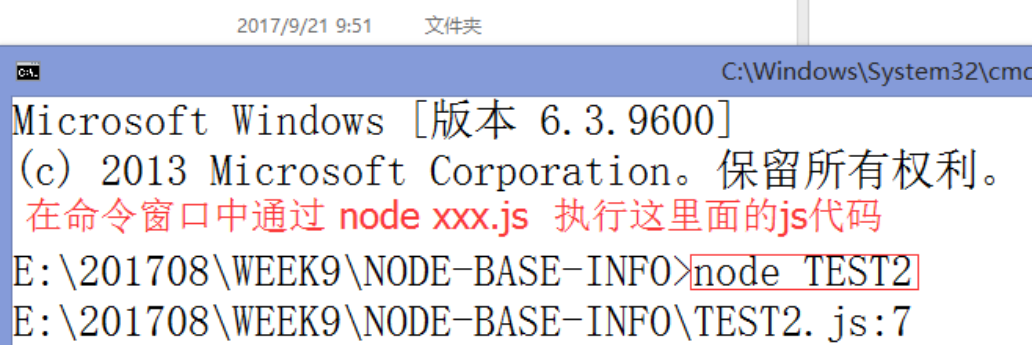
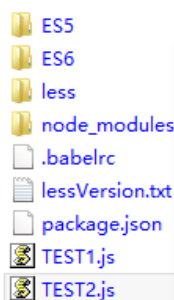
```
let temp = require('./TEST1');
temp.sum();
```

在WS中, 如果我们想把当前写好的JS代码在NODE环境下运行, 只需要右键 Run 一下即可

原因: ws中嵌入了node.exe执行文件, 右键执行就相当于使用node在把js代码执行



dos



ws中的terminal也相当于dos命令, 在那里面执行也可以

4、综合练习

有三个自定义模块: A、B、C

A中提供一个sum方法, 实现任意数求和

B中提供一个avg方法, 实现求一堆数的平均数 (先求和, 在求平均, 求和需要用到A中的sum)

C中准备一些数字, 调取B中的avg方法, 获取到需要的平均数

A

```
1. // module.exports = {
2. //     sum(){
3. //
4. //     }
5. // };
6.
7. // module.exports = {
8. //     sum: (...arg)=> {
9. //         //->使用拓展运算符接收传递给函
           数的实参集合:ARG是一个数组
10. //         return eval(arg.joi
           n('+'));
11. //     }
12. // };
13.
14. module.exports = {
15.     sum: (...arg)=> eval(arg.joi
           n('+'))
16. };
```

B


```
1. let temp = require('./A');
2. module.exports = {
3.     avg: (...arg)=> temp.sum(...arg)
   / arg.length
4. };
5.
6. // let ary = [12, 23, 13, 14, 25, 34,
   12];
7. // console.log(Math.max.apply(undefin
   ed, ary));
8. // console.log(Math.max(...ary));
   //=>原理等同于上面的操作
```

C

```
1. let temp = require('./B');
2. console.log(temp.avg(12, 23, 34, 4
   5));
```

NODE中的内置模块

当JS在NODE中运行的时候，NODE也会给JS提供一些内置的属性和方法，这些属性方法都存储在NODE的内置模块中

- http
- url
- fs
- ...

fs 内置模块

fs模块中提供了一些属性和方法供JS在服务器上进行I/O操作（input/output 输入/输出）：对文件的增删改查等处理

JS在客户端浏览器运行的时候，能否对客户端的本地文件进行I/O操作？

答案：不能，如果可以的话，对客户端的电脑会造成极大的安全隐患；但是有些操作是可以的，例如上传文件或者图片(input type='file')，但是需要用户手动去选择本地文件上传；

JS在服务器的NODE中运行的时候，能否对服务器上的文件进行I/O操作？

答案：能，FS模块中提供的方法就是让我们干这些事情的

```
1. let fs = require('fs');
2.
3. 1、读取文件中内容
4. fs.readFile 异步读取某一个文件中的内容
5. fs.readFileSync 同步读取某一个文件中的内容（相对于异步来说，同步操作是把内容读取完成后，才会执行后续的操作，而异步是读取过程中不管是否读取完成，都会去执行后续的操作）
6.
7. let con = fs.readFileSync([pathname],
  [encode]);
8.
9. fs.readFile([passname], (error, value)=> {
10.
11. });
12.
13. 2、向指定文件中写入内容
14. fs.writeFile([pathname],[content],[encode],[callback])
15. fs.writeFileSync([pathname],[content],[encode])
16. //=>我们的内容写入是覆盖式的：新写入的内容会把之前的内容都覆盖掉（之前的内容都没有了），如果不想覆盖，先把之前的内容获取，和最新的内容进行拼接，把拼接后的结果整体写入进去，类似于 xxx.innerHTML+=``
17.
```

3、读取文件夹中的文件目录

```
18: fs.readdir([pathname],(error,value)=>
    {
20:     //->当查找完成后执行回调函数，value存
        储的值就是找到的全部文件目录信息（数组）
21: });
22: fs.readdirSync([pathname]) 返回的结果是
        一个数组，包含当前目录下的文件列表信息
```

综合案例：自己编写一个模块，能够批量编译less文件，并且配置在项目的命令中

```
1. let less = require('less'),
2.     fs = require('fs');
3.
4. // -> 获取LESS文件夹中的文件, 筛选出所有后缀
   名是less的, 这些文件就是我们接下来要处理的
5. let fileAry = fs.readdirSync('./less');
6. fileAry = fileAry.filter((item, index) => {
7.     return /\.less/i.test(item);
8. });
9.
10. // -> 循环获取的less文件, 一个文件一个文件的
    进行处理
11. fileAry.forEach((item, index) => {
12.     // -> 读取循环这个LESS文件中的内容 (需要
        设置UTF-8, 只有这样获取的才是一个字符串, 否
        则是BUFFER数据)
13.     let con = fs.readFileSync(`./less/${item}`, 'utf-8');
14.
15.     // -> 把获取的内容编译为css
16.     less.render(con, {compress: true}, (error, value) => {
17.         // -> 把编译好的CSS写入到指定的文件
            中 (CSS目录下和之前LESS文件名相同的CSS文件
            下)
18.         let newFileName = item.replac
```

```
    e(/\.less/ig, '.min.css');
19.
20. fs.writeFileSync(`./css/${newFileName}
    e}`, value.css);
21.     });
22. });
23.
24. //=>调取LESS模块中的RENDER方法,实现把LES
    S代码编译成CSS; compress:true设置编译的时
    候压缩
25. // less.render('@W: 100px;          @H:
    100px;.box {width: @W;height: @H;
    background: red;}', {compress: true},
    (error, value)=> {
26. //      //->value:是一个对象,其中的css属
    性中存储的就是编译后的css代码(经过压缩的)
27. //      console.log(value.css);
28. // });
```

package.json

```
1. {  
2.   ...  
3.   "scripts": {  
4.     "less": "node lessRender.js"  
5.   },  
6.   ...  
7. }
```

url 内置模块

把一个url地址进行解析，解析出每一部分的内容

```
1. //=>url.parse([str],true/false)
2.
3. let url = require('url');
4. let str = 'http://www.zhufengpeixun.cn:80/stu/index.html?name=sh&age=18#video';
5.
6. console.log(url.parse(str,false));
7. /*
8.   Url {
9.     protocol: 'http:',  协议
10.    slashes: true,  是否有斜杠
11.    auth: null,  作者
12.    host: 'www.zhufengpeixun.cn:80',
        域名+端口
13.    port: '80',  端口号
14.    hostname: 'www.zhufengpeixun.cn',
        域名
15.    hash: '#video',  哈希
16.    search: '?name=sh&age=18',  问号传参
17.    query: 'name=sh&age=18',  问号传参
18.    pathname: '/stu/index.html',  请求资源的路径名称（前面最开始有一个斜杠）
19.    path: '/stu/index.html?name=sh&age=18',  路径+问号传参
20.    href: 'http://www.zhufengpeixun.cn:80/stu/index.html?name=sh&age=18#video'  原始字符串内容
```



```
21.  }
22.  */
23.
24.  console.log(url.parse(str,true));
25.  /*
26.    和写FALSE的时候对比，其它值一样，只是query
    的值不在是一个字符串而是一个对象，它默认会
    把问号传递参数的值以对象键值对的方式来存
    储，方便后期的操作（最常用）
27.    query: { name: 'sh', age: '18' }
28.  */
```

http内置模块

作为后台开发工程师，我们需要创建一个服务用来接收客户端的请求，并且把需要的数据内容准备好，最后返回给客户端，而HTTP内置模块中提供了做这些事情的属性和方法

```
1. let http = require('http'),
2.     url = require('url'),
3.     fs = require('fs');
4.
5. //->首先创建一个服务
6. let server = http.createServer((req,
7.     res)=> {
8.         /*
9.         * 如何向当前的服务发送请求?
10.        * 都是基于浏览器来完成，在客户端浏览器的
            地址栏中输入
11.        *   http://localhost:1990/....
            (访问本机服务)
12.        *   http://172.18.1.0:1990/...
            (局域网IP或者外网IP访问某台主机上的服务，真实项目中是通过域名访问的)
13.        */
14.
15.        //->回调函数在客户端请求的时候不仅被执行了，而且还传递了两个参数值：
16.        //req: request 对象,里面存储了客户端发送过来的全部请求信息，例如：req.url 存储的就是当前客户端请求资源的路径名称以及传递
```

的问号参数值

```
17.         //res: response 对象,里面提供了很多
            的方法可以让服务器端把内容返回给客户端, 例
            如: res.end([content]) 就是把指定的内容
            返回给客户端的浏览器
18.     });
19.
20.     //->还需要给创建的服务安排一个端口号(一台服
            务器上可以能有很多的服务,需要使用端口号来区
            分)
21.     server.listen(1990, ()=> {
22.         //->当服务创建成功,端口号也已经正常分
            配的时候,会触发此回调函数执行
23.         /*
24.          * Error: listen EACCES 0.0.0.0:8
            0
25.          * Error: listen EACCES ...:80
26.          * 类似于这样的报错信息都说明当前的80
            端口已经被其它的服务占用了,我们要不然结束其
            它正在使用这个端口号的服务,要不然自己换一个
            其它的端口号
27.          */
28.         console.log('server is success, l
            istening on 1990 port! ');
29.     });
```