## node.js简介

### 引入

Node.js不是一门语言

Node.js不是库、不是框架

Node.js是一个JavaScript 运行时环境

简单点来讲就是Nodejs可以解析和执行JavaScript代码

以前只有浏觉器可以可以解析执行JavaScript代码

也就是说现在的JavaScript 可以完全脱高浏览器来运行，一切都归功于：Node.js

### node.js(ECMAscript和node提供的一些api)

没有BOM、DOM

EcmaScript

在Node这个JavaScript执行环境中为JavaScript提供了一些服务器级别的操作API

例如文件读写、网络服务的构建、网络通信、http服务器、等处理。

### node.js的特性

Node.js uses an event-driven,non-blocking l/O model that makes it lightweight and efficient.

event-driven 事件驱动

non-blocking /O model 非阻塞IO模型（异步）

lightweight and efficient.轻量和高效

### npm

Node.js'package ecosystem,npm,is the largest ecosystem of open source libraries in the world.

npm是世界上最大的开源库生态系统

绝大多数JavaScript相关的包都存放在了npm上，这样做的目的是为了让开发人员更方便的去下载使用我。

### node.js能做什么

1、web服务器后台

2、命令行工具

例如：npm git(c语言)

3、

B/S编程模型

Browser-Server

back-end

任何服务端技术这种BS编程模型都是一样，和语言无关

Node只是作为我们学习BS编程模型的一个工具而已

## node.js环境安装

### 官网下载

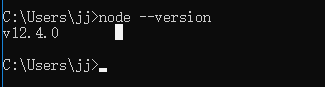
<https://nodejs.org/en/>

### 安装

傻瓜式的一撸安装 next

### 确实是否安装成功

打开命令行，输入 node –version 或者node -v



### 环境变量

## Node.js的helloworld

### 创建文件helloworld.js

|  |
| --- |
| var i = "helloworld";  console.log(i); |

### 在命令行输入命令

node helloworld.js



注：文件的名字不能是node.js

### 读取文件

浏览器中的Javascript是没有文件操作的能力的，但是Node中的Javascript具有文件操作的能力。

fs是file-system的简写，就是文件系统的意思在Node中如果想要进行文件操作，就必须引入fs这个核心模块 在fs这个核心模块中，就提供了所有的文件操作相关的API

例如：fs.readFile就是用来读取文件的

|  |
| --- |
| //1、使用require加载fs核心模块  var fs = require("fs");  //2、读取文件  //第一个参数是要要读的文件路径  //第二各参数是一个回调函数  //error  // 如果读取失败，error就是错误对象  // 如果读取成功，error就是null  //data  // 如果读取成功，data就是读取的数据  // 如果读取失败，error就是错误对象，data就是undifined  fs.readFile('../data/hello.txt',function(error,data){  //<Buffer 68 65 6c 6c 6f 20 77 6f 72 6c 64>  // 文件中存储的其实都是二进制数据e1  // 这里为什么看到的不是e和1呢？原因是二进制转为16进制了  // 但是无论是二进制还是16进制，人类都不认识  // 所以我们可以通过tostring方法把其转为我们能认识的字符  //console.log(data);  console.log(data.toString());  }); |

报错时：显示为一个对象。

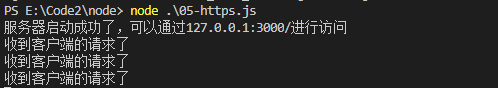
|  |
| --- |
| [Error: ENOENT: no such file or directory, open 'E:\Code2\data\hello1.txt'] {  errno: -4058,  code: 'ENOENT',  syscall: 'open',  path: 'E:\\Code2\\data\\hello1.txt'  } |

### 写文件

|  |
| --- |
| var fs = require('fs');  //第一个参数：文件的路径  //第二个参数:文件的内容  // error:  // 成功：文件写入成功。  // error 是null  // 失败:文件写入事变  // error就是错误对象  fs.writeFile('./data/你好.md','大家好，我是node.js',function(error){  console.log('文件写入成功');  }) |

### http(服务级别的helloworld)

|  |
| --- |
| //使用node非常轻松的构建一个web服务器，在node中转梦提供了一个核心模块：http/  //http这个模块的职责就是帮你创建编写服务器  //1.加载http 核心模块  var http = require('http');  //2.使用http.createServer()创建一个web服务器  var server = http.createServer();  //3、服务器的使用  // 提供服务:对数据的服务  //3.1注册request请求时间，当客户端请求过来，就会自动触发服务器的request请求时间  //然后执行第二个参数，回调处理  server.on('request',function(){  console.log('收到客户端的请求了');  })  //3.2绑定端口号，启动服务器  server.listen(3000,function(){  console.log('服务器启动成功了，可以通过127.0.0.1:3000/进行访问');  }); |



### 处理请求

|  |
| --- |
| //使用node非常轻松的构建一个web服务器，在node中转梦提供了一个核心模块：http/  //http这个模块的职责就是帮你创建编写服务器  //1.加载http 核心模块  var http = require('http');  //2.使用http.createServer()创建一个web服务器  var server = http.createServer();  //3、服务器的使用  // 提供服务:对数据的服务  //3.1注册request请求时间，当客户端请求过来，就会自动触发服务器的request请求时间  //然后执行第二个参数，回调处理  //request请求时间处理函数，需要接受两个和参数  // request 请求对象，请求对象可以获取客户端的请求的信息，例如请求路径  // response 响应对象，响应对象可以给客户端发送响应消息。  server.on('request',function(request,response){  //127.0.0.1:3000/a request.url = /a  //127.0.0.1:3000/abc request.url = /abc  console.log('收到客户端的请求了,请求路径是'+request.url);  //2、响应内容  //response 有个一个方法，wirite可以给客户端发送响应数据  //write可以使用多次，但是最后一定哟啊使用end来结束响应，否则客户端会一致等待  response.write('hello');  response.write('node.js');  //3、结束响应  response.end();  })  //3.2绑定端口号，启动服务器  server.listen(3000,function(){  console.log('服务器启动成功了，可以通过127.0.0.1:3000/进行访问');  }); |

### http进阶

//由于我们的服务器的能力还非常的若，无论是什么请求，都只能响应hellonode.js.

//思考，我们希望当请求不同的路径的时候响应不同的结果

//例如： / index

// /login 登录

// /register 注册

// /haha 哈哈

|  |
| --- |
| //由于我们的服务器的能力还非常的若，无论是什么请求，都只能响应hellonode.js.  //思考，我们希望当请求不同的路径的时候响应不同的结果  //例如： / index  // /login 登录  // /register 注册  // /haha 哈哈  //1.加载http 核心模块  var http = require('http');  //2.使用http.createServer()创建一个web服务器  var server = http.createServer();  //3、服务器的使用  /// /favicon.ico 浏览器会自动访问该请求，请求网站的图标  server.on('request',function(req,res){  console.log('收到客户端的请求了,请求路径是'+req.url);  //根据不同的请求路径发送不同的响应结果  //1、获取请求的路径  //req.url 获取到的是端口号之后的那一部分路径，也就是所有的url都是以/开头的  //2、处理请求  var url = req.url;  // if(url =='/'){  // res.end('index page');  // } else if(url=='/login'){  // res.end('login page');  // } else {  // res.end('404 not found');  // }  //响应内容只能是字符串,数字，对象，布尔值，都不行  if(url==='/products'){  var product= [  {  name:'苹果X',  price:8888  },  {  name:'香蕉',  price:8777  },  {  name:'橘子',  price:4444  }  ]  }  res.end(JSON.stringify(product));  })  //3.2绑定端口号，启动服务器  server.listen(80,function(){  console.log('服务器启动成功了，可以访问了');  }); |

## node中的js

### EcmaScript

没有dom 和bom

### 核心模块

Node为JavaScript提供了很多服务级别的API，这些Api大多数都被包装到了一个核心模块中的。例如文件操作的fs核心模块，http服务构建的http模块，path路径操作模块，os操作系统信息的模块

如果想要使用模块，必须将该模块引入。

|  |
| --- |
| var os = require('os');  //获取当前机器的CPU信息  console.log(os.cpus());  //memory内存  console.log(os.totalmem); |

### 用户自定义模块

a模块：

|  |
| --- |
| //require 是一个方法,它的作用就是用来加载模块的  //在node中，模块又是三种：  // 具名的核心模块，例如fs http  // 用户自己编写的文件模块  // 注：相对路径必须加./  var foo = 'aa';  console.log('a start');  require('./b.js');  console.log('a end');  console.log(foo)  function add(x,y){  return x+y;  } |

b模块

|  |
| --- |
| var foo = 'aa';  console.log(add(3+2)); |

执行a.js结果

a start

b.js 文件被加载执行了

a end

1、在node中没有全局作用域

在Node中,没有全局作用域,只有模块作用域。外部访问不到内部，内部也访问不到外部

2、引用的省略

例如：require(‘./b’)；

3、相对路径不能省略，如果省略，会将内容当做核心模块。

4、模块作用域之间的通信，模块之间默认是封闭的。

有时候,我们加载文件模块的目的不是为了简简单单的执行里面的代码,更重要是为了使用里面方法。

### 加载和导出

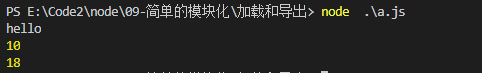
a.js

|  |
| --- |
| //1、require 方法的两个作用：  // 1、加载文件模块并执行里面的代码  // 2、拿到被夹在文件模块导出的接口对象，每个文件模块都提供一乐一个对象：exports  // exports 默认是一个空对象,目的是把所有被外部访问的成员，挂载到这歌export对象中  var bExports = require('./b.js');  console.log(bExports.foo);  console.log(bExports.add(1,9));  console.log(bExports.age); |

b.js

|  |
| --- |
| var foo = 'bbb';  var age = 18;  exports.age = age;  exports.foo = 'hello';  exports.add = function(x, y){  return x + y;  } |

结果



### 响应内容content-type

|  |
| --- |
| var http = require('http');  var server = http.createServer();  server.on('request',function(req,res){  console.log("hello");  var url = req.url;  if(url ==='/plain'){  res.setHeader('Content-Type','text/plain;charset=utf-8');  res.end('hello 世界')  } else if (url ==='/html'){  res.setHeader('Content-Type','text/html;charset=utf-8');  res.end("<font color='red' >hello 哈哈</font>");  }  });  server.listen(3000,function(){  console.log("server is on");  }) |

## 请求对象和响应对象

## 实现Apache服务器的功能

咱们以前使用过Apache服务器软件,这个软件默认有一个www目录,所有存放在www目录中的

127.0.0.1:80/a.txt

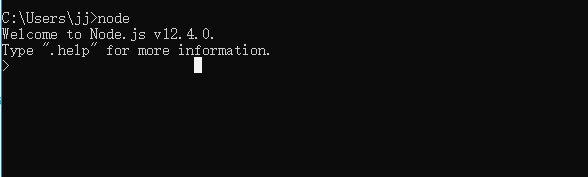
127.0.0.1:80/index.txt

127.0.0.1:80/apple/login.html

|  |
| --- |
| //app accpliction 应用程序  var http = require('http');  var fs = require('fs');  var template = require('art-template');  var url = require('url');  var comments = [  {  name:'张三',  message:'1',  dateTime:'2019-08-05',  },  {  name:'张三',  message:'2',  dateTime:'2019-08-05',  },  {  name:'张三',  message:'3',  dateTime:'2019-08-05',  },  {  name:'张三',  message:'4',  dateTime:'2019-08-05',  }  ]  http.createServer(function(res,rsp) {  var parseObj= url.parse(res.url,true);  var pathname = parseObj.pathname;  console.log(url);  if(pathname==='/'){  fs.readFile('./views/index.html',function(err,data){  if(err){  return rsp.end('404 not Found');  }  var htmlStr = template.render(data.toString(),{  comments:comments  });  rsp.end(htmlStr);  }  )} else if(pathname==='/post') {  fs.readFile('./views/post.html',function(err,data){  if(err){  return rsp.end('404 111 Found');  }  rsp.end(data);  })  } else if(pathname==='/pinglun') {  //使用url模块的parse方法把请求路径中的查询字符串解析成一个对象。  //1、获取髹提交的数据parseObj.query  //2、将当前时间日期添加到数据对象中，然后存储到数组中  //3.让用户重定向跳转到首页  //当用户重新请求/的时候，数据中的数据发生改变  var comment = parseObj.query;  comment.dateTime = '2017-08-05';  comments.push(comment);  //如何通过服务器让客户端重定向  //1、状态吗设置为302 临时重定向 statusCode  //2、在响应投中通过Location告诉客户端往哪儿重定向  //如果客户端收到服务器的响应码是302，就会自动去响应头中找Location  rsp.statusCode = 302,  rsp.setHeader('Location','/');  rsp.end();  }else if(pathname.indexOf('/public/')===0) {  //public/css/main.css  //public/js/main.js  //public/lib/jquery.js  //统一处理，如果以/public开头的，则认为是要从public中获取某个资源  fs.readFile('.'+url,function(err,data){  if(err){  return rsp.end('404 not found');  }  rsp.end(data);  });  console.log(url);  } else {  fs.readFile('.\views\404.html',function(err,data){  console.log(err);  if(err){  return rsp.end('404 Not Fund')  }  rsp.end(data);  });  }  }).listen(3000,function(){  console.log('running')  }); |

### 进node的控制台

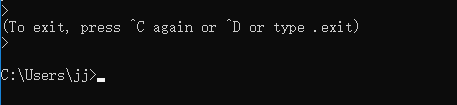
在cmd 中 输入node，进入node控制台



不用加载模块，可以直接使用node中模块



退出控制台ctrl+C+C



## 模块系统

### CommonJS模块规范

在Node中的JavaScript还有一个很重要的概念:模块系统。

1、模块作用域

2、使用require方法用来加载模块

3、使用exports接口对象用来导出模块中的成员

## require加载模块

### require模块

语法：var 自定义变量名称 = require(‘模块’);

作用：1、执行被加载模块中的代码

2、得到被加载模块中的exports导出接口对象

### require加载规则

1、优先从缓存加载

2、核心模块

3、路径形式的文件模块

4第三方模块

### 优先从缓存加载

require('./a.js');

require('./b.js');

在a.js中已经加载了b.js，将会优先从缓存中加载，这样做的目的是为了避免重复加载

### 模块标识符

require(‘模块标识符’)

1、非路径形式的模块表示

路径形式的模块：

./ : 当前

../ :上一级目录，不可省略

/xxx 几乎不用

d:/xx/a.js 绝对路径基本不用

例如：require(‘/foo.js’);

2、核心模块

核心模块文件已经被编译到了二进制文件中了,我们只需要按照名字来加载就可以了

require('fs')

require('http') ;

3、第三方模块

第三方模块必须通过npm下载下来，使用的时候通过require(‘包名’)的方式进行加载才能使用。既不是核心模块。也不是路径形式的模块。

1、先找到当前文件所处目录中的node\_modules目录

2、node\_modules/art-template

3、node\_modules/art-template/package.json文件

4、node\_modules/art-template/package.json文件中的main属性

5、main属性中就记录了art-template的入口模块 然后加载使用这个第三方包，实际上最终加载的还是文件

6、如果package.json文件不存在或者main指定的入口模块是也没有

7、则node会自动找该目录下的index.js也就是说index.js会作为一个默认备选项

如果以上所有任何一个条件都不成立,则会进入上一级目录中的node-modules目录查找。如果上一级还没有,则继续往上上一级查，直到该磁盘的根目录。

注意:我们一个项目有且只有一个node-modules,放在项目根目录中,这样的话项目中所有的子目录中的代码都会被加载，不会出现有多个nodemodules

### exports加载模块

Node中是模块作用域,默认文件中所有的成员只在当前文件模块有效，对于希望可以被其它模块访问的成员,我们就需要把这些公开的成员都挂载到exports接口对象中就可以了

1、导出多个成员(必须在对象中):

|  |
| --- |
| exports.a = 'hello';//字符串  exports.b = 123;//数字  exports.c = function(){//函数  console('ccc');  };  exports.d = {//对象  foo:'var'  }; |

2导出单个成员(拿到的就是:函数、字符串)

|  |
| --- |
| module.exports ='hello';  //也可以这样导出多个成员  module.exports = function(x,y){  return x+y;  }  //该方法也可以导出多个成员  module.exports = {  add:function(){  return x+y  },  str:'hello'  } |

原理：exports和module.exports的一个引用

内部代码 var exports = module.exports

exports.foo = ‘aaa’,//两者等价

module.exports.foo = ‘aaa’,

如果重新给exports赋值，exports会重新指向一个新的对象。

1、无法导出赋值之后的值

exports.foo = ‘aa’ //将会导出该值

exports = {}//对foo重新赋值

exprots.foo = ‘bar’;//无法导出该值

module.exports.b = ‘456’; //最终会导出该值

2、重新给module.exports 赋值，最终会导出该值

|  |
| --- |
| //1、与exports断开连接，不同引用  module.exports = {  foo:'var'  }  //2、重新恢复引用  exports = module.exports;  //3、导出对象  exports = 'hello'; |

## path路径操作模块

path.basename

,获取一个路径的文件名(默认包含扩展名)

path.dirname

。获取一个路径中的目录部分

path.extname

。获取一个路径中的扩展名部分

path.parse

。把一个路径转为对象

. root根路径

? dir目录

- base包含后缀名的文件名

ext后名

. name不包含后缀名的文件名

path.join

,当你需要进行路径拼接的时候,推荐使用这个方法

path.isAbsolute判断一个路径是否是绝对路径

### node中的非模块成员

在每个模块中,除了require 、exports等模块相关API之外,还有两个特殊的成员

-dirname可以用来获取当前文件模块所属目录的绝对路径，不包含文件名

\_f1lename可以用来获取当前文件的绝对路径

console.log(\_\_dirname) E:\Code2\node\day06

console.log(\_\_filename) E:\Code2\node\day06\app.js

### 路径问题

在文件操作中,使用相对路径是不可靠的,因为在Node中文件操作的路径被设计为相对于执行node命令所处的路径(不是bug,人家这样设计是有使用场景) 。

所了为了解决这个问题,很简单,只需要把相对路径变为绝对路径就可以了。

在拼接路径的过程中,为了避免手动拼接带来的一些低级错误,所以推荐多使用: path.join()辅助拼接。所以为了尽量避免刚才所描述这个问题,大家以后在文件操作中使用的相对路径都统一转换为动态的绝对路径。

|  |
| --- |
| var fs = require('fs');  var path = require('path')  //这里的路径是相对于执行node命令的位置  //node的路径，相对路径设计的就是相对于执行node命令所处的路径  //这里可以使用\_\_dirname或者\_\_filename  fs.readFile(path.join(\_\_dirname+'/a.txt'),function(err,data){  if(err){  throw err  }  console.log(data);  }) |

注：模块中的路径标识和这里的路径没关系,不受影响(相对于文件模块)

## npm(node package manager)

### npm

npm的第二层含义就是一个命令行工具,只要你安装了 node就已经安装了npm。

npm也有版本这个概念。可以通过在命令行中输入npm –version

npm升级：npm install –global npm

### 常用命令

npm init

npm init-y可以跳过向导,快速生成

npm instal

一次性把dependencies选项中的依赖项全部安装

npm i

nprm install 包名

只下载

npm i包名

npm install--save 包名

下载并且保存依赖项(package.json文件中的dependencles选项)

npmi-s包名

npm uninstall 包名

只删除,如果有依赖项会依然保存

npm un 包名

npm uninstall--save 包名

删除的同时也会把依赖信息也去除

npm un-S包名

npm help 查看使用帮助

### 创建项目

这个文件可以通过npm init 初始化的方式生成出来

package name: (npm) npm-demo

version: (1.0.0) 0.0.1

description: 这是一个项目

entry point: (index.js) main.js

test command:

git repository:

keywords:

author: libin

license: (ISC)

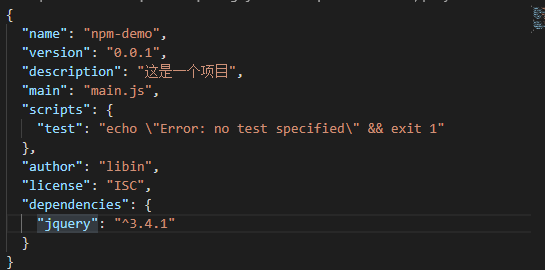
### package.json

作用：保存第三方包的信息。

我们建议每一个项目都要有一个package-json文件(包描述文件,就像产品的说明书一样) ,给人踏实的感觉。

1、npm install --save jquery:下载模块，会在packjson中生成依赖，建议在执行npm是都加上—save

2、如果node module中的依赖包都被删除，使用nom install会将package.json中的依赖全部下载。



### 解决npm被墙问题

npm存储包文件的服务器在国外,有时候会被墙,速度很慢,所以我们需要解决这个问题。

http://npm.tabao.org)润淘宝的开发团队把npm在国内做了一备份。

1、安装淘宝的cnpm

npm install –-global cnpm

–-global 安装到全局

接下来你安装包的时候把之前的npm替换成cnpm。

例如：cnpm install jquery 会从走国内的cnpm

2、如果不想使用cnom从淘宝的服务器下载

npm install jquery –registry = <https://registry.npm.taobao.org>

只要经过了上面命令的配置,则你以后所有的npm install都会默认通过淘宝的服务器来下载。

但是每一次手动这样加参效很麻烦,所我们可以把这个选项加入配置文件中:

npm config set registry https://registry.npm.taobao.org

查看配置信息 npm config list

## Express

### Express

原生的http在某些方面表现不足以应对我们的开发需求,所以我们就需要使用框架来加快我们的开发效率,框架的目的就是提高效率,让我们的代码更高度統一。

在Node中,有很多Web开发框架,我们这里以学习express为主。

官网：<http://expressjs.com/>

Express

### helloword

|  |
| --- |
| var express = require('express');  //1、创建app  var app = express();  app.get('/',function(req,res){  //1、原生的语法还可以使用  // res.write('hello');  // res.write('world');  // res.end();  })  app.get('/index',function(req,res){  //1、原生的语法还可以使用  // res.write('hello');  // res.write('world');  // res.end();  })  app.listen(3000,function(err,data){  console.log('express app is running');  }) |

### express的路径问题

|  |
| --- |
| //加载模块,./不能省略,加载后直接进行调用  require('./data/foo.js')('hello');  var fs = require('fs');  //文件操作中的相对路径可以省略./，模块无法省略./  //所有的文件操作都是异步的  //data/a.tx:同级目录下去寻找  ///data/a.txt :去当前磁盘根目录的data/a.txt去寻找  fs.readFile('data/a.txt',function(err,data){  if(err){  return console.log('读取失败')  }  //后执行  console.log(data.toString());  })  //1、先执行  //模块加载中的./不能省略  //./data/foo.js 同级目录去寻找  ///data/foo.js 磁盘的根目录去寻找  require('./data/foo.js')('msg'); |

### node.js修改后自动重启服务器

我们这里可以使用一个第三方命名航工具: nodemon来帮我们解决频繁修改代码重启服务器问题。nodemon是一个基于Node.js开发的一个第三方命令行工具,我们使用的时候需要独立安装。

注:在任意目录执行该命令都可以，也就是说,所有需要--global来安装的包都可以在任意目录执行

安装nodemon :npm install –global nodemon

使用：nodemon app.js

测试：nodemon –-version

只要是通过nodemon app.js启动的服务,则它会监视你的文件变化,当文件发生变化的时候, 自动帮你重启服务器。

### 开放静态资源

|  |
| --- |
| var express = require('express');  //1、创建app  var app = express();  //当以/public/开头的时候，去./public/目录中找对应的资源  //ap.user('/public',express.static('./public/'))  //1、当没有第一个参数的时候，直接访问该资源,例如/index.html，不需要加前缀  app.use(express.static('./public/'));  //2、访问必须加/public/index.html  app.use('/public/',express.static('./public/'));  //3、访问必须加/a/public/index.html  app.use('/a/',express.static('./public/'));  app.get('/',function(req,res){  res .send('dfds11f11');  })  app.listen(3000,function(err,data){  console.log('express app is running');  }); |

### express中配置art-template

1、安装：

npm install --save art-template express-art-template

2、配置

app.engine('html',require('express-art-template'));

例如：

|  |
| --- |
| var express = require('express');  var app = express();  //1、配置使用art-temlplate模板引擎  //第一个参数，表示当渲染一.art结尾的文件的时候使用art-template模板引擎  //express-art-template是专门用来在Express中把art-template整合到Express 中  // 虽然外面这里下需要记载art-template但是也必须安装  // 原因就在于express-art-template依赖了 art-templatel  app.engine('html',require('express-art-template'));  //2Express,为Response相应对象提供了一个方法: render  //render方法默认是不可以使用,但是如果配置了模板引擎就可以使用了  //格式:res.render('html的模板名',{模板数据})  // 第一个参数不能写路径,默认会去项目中的views目录查找该模板文件  // 也就是说Express有一个约定:开发人员把所有的视图文件都放到views目录中  //3、修改默认的views目录  //app.set('views','render函数的默认路径');  app.get('/',function(req,res){  res.render('404.html');  });  app.get('/admin',function(req,res){  //默认去views取寻找  res.render('admin/index.html',{  title:'管理系统'  });  });  app.use('/public/',express.static('./public/'))  app.get('/',function(req,res){  res.send('/page');  });  app.get('/post',function(req,res){  res.send('/post page');  });  app.listen(3000,function(){  console.log('runing');  }); |

### express中获取post表单数据

在Express中没有内置获取表单POST请求体的API,这里我们需要使用一个第三方包body-parser

安装：npm install –save body -parser

|  |
| --- |
| var express = require('express');  //1、导包  var bodyParser = require('body-parser');  var app = express();  //2、配置body-parser  // 只要加入这个配置,则在req请求对象上会多出来一个属性: body  // 也就是说你就可以直接通过req.body来获取表单POST请求体数据了  app.use(bodyParser.urlencoded('body-parser'));  //1、配置使用art-temlplate模板引擎  //第一个参数，表示当渲染一.art结尾的文件的时候使用art-template模板引擎  //express-art-template是专门用来在Express中把art-template整合到Express 中  // 虽然外面这里下需要记载art-template但是也必须安装  // 原因就在于express-art-template依赖了 art-templatel  app.engine('html',require('express-art-template'));  //2Express,为Response相应对象提供了一个方法: render  //render方法默认是不可以使用,但是如果配置了模板引擎就可以使用了  //格式:res.render('html的模板名',{模板数据})  // 第一个参数不能写路径,默认会去项目中的views目录查找该模板文件  // 也就是说Express有一个约定:开发人员把所有的视图文件都放到views目录中  //3、修改默认的views目录  //app.set('views','render函数的默认路径');  app.get('/',function(req,res){  res.render('404.html');  });  app.get('/admin',function(req,res){  //默认去views取寻找  res.render('admin/index.html',{  title:'管理系统'  });  });  app.use('/public/',express.static('./public/'))  app.get('/',function(req,res){  res.send('/page');  });  //post请求  app.post('/post',function(req,res){  console.log('收到表单post请求');  //1、获取表单post请求体的数据，req.query只能获取get的请求数据  //2、处理  //3、发送响应  });  app.listen(3000,function(){  console.log('runing');  }); |

## -基本的curd

## MongoDB

## node中操作MongoDB

### 1、使用官方的mongodb包来操作

### 使用第三方mongoos包操作mongodb

官网:https://www.mongoose.com/

1、初始化项目

npm init -y

2、下载mongooose

npm i mongoose

3、连接数据库

|  |
| --- |
| const mongoose = require('mongoose');  //连接mongdb数据库  mongoose.connect('mongodb://localhost:27017/test', {useNewUrlParser: true});  //创建一个模型，设计数据库  const Cat = mongoose.model('Cat', { name: String });  const kitty = new Cat({ name: 'Zildjian' });  kitty.save().then(() => console.log('meow')); |

## node操作mysql

### 安装

安装：npm i mysql

### 配置

|  |
| --- |
| var mysql = require('mysql');  var connection = mysql.createConnection({  host : 'localhost',  user : 'me',  password : 'secret',  database : 'my\_db'  });    connection.connect();    connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {  if (error) throw error;  console.log('The solution is: ', results[0].solution);  });    connection.end(); |

### hello world

|  |
| --- |
| var mysql = require('mysql');  //1、创建连接数据库  var connection = mysql.createConnection({  host : 'localhost',  user : 'root',  password : '123456',  database : 'car'  });  //2、连接数据库  connection.connect();  //3、执行数据库操作  connection.query('select userpwd from user where id=1 ', function (error, results, fields) {  if (error) throw error;  console.log('The solution is: ', results);  });    connection.end(); |

### promise函数

|  |
| --- |
| var fs = require('fs');  //在EcmaScript 6 中新增了一个 API Promise, Promise是一个构造函数  //1、创建promise容器，primise不是异步，里面的函数是异步  var promise = new Promise(function(resolve,reject){  fs.readFile('./data/a.txt','utf8',function(err,data){  if(err){  //承诺容器中的任务失败,将容器的pending状态变为，rejected  reject(data)  }else {  //承诺容器中的任务成功了，将容器的pending状态变为，resolve  resolve(data)  }  });  });  var promise2 = new Promise(function(resolve,reject){  fs.readFile('./data/b.txt','utf8',function(err,data){  if(err){  //承诺容器中的任务失败,将容器的pending状态变为，rejected  reject(data)  }else {  //承诺容器中的任务成功了，将容器的pending状态变为，resolve  resolve(data)  }  });  });  var promise3 = new Promise(function(resolve,reject){  fs.readFile('./data/c.txt','utf8',function(err,data){  if(err){  //承诺容器中的任务失败,将容器的pending状态变为，rejected  reject(data)  }else {  //承诺容器中的任务成功了，将容器的pending状态变为，resolve  resolve(data)  }  });  });  //当promise成功后，然后then做指定的操作  //then方法接收的function就是容器中的resolve函数  //第一个函数为成功后resolve函数  //第二个函数为成功后reject函数  promise.then(function(data){  return promise2;  },function(err){  console.log(err);  }).then(function(data){  //此处的结果是promise2的promise  console.log(data);  return promise3;  }).then(function(data){  //此处的结果是promise3的promise  console.log(data);  console.log('end')  }); |

### 封装promise

|  |
| --- |
| var fs = require('fs');  function pReadFile(filePath){  return promise3 = new Promise(function(resolve,reject){  fs.readFile(filePath,'utf8',function(err,data){  if(err){  reject(data)    }else {  resolve(data)  }  });  });  }  pReadFile('./data/a.txt').  then(function(data){  return pReadFile('./data/b.txt');  }).then(function(data){  return pReadFile('./data/c.txt');  }).then(function(data){  console.log(data);  }); |

## 创建项目

### 初始化项目

npm init -y 初始化

git init 初始化git

npm i art-template express-art-template 安装express

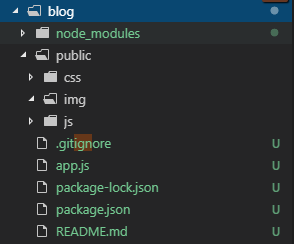
创建README.md 和.gitignore 文件

npm install bootstrap@3

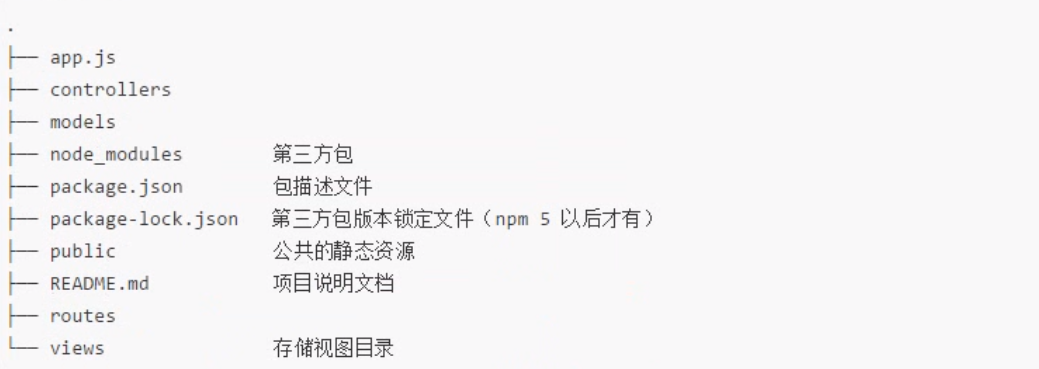
npm install jquery

npm i express mongoose

获取表单数据：npm install --save body -parser



### 项目结构



### 路由设计

