[一、Node.js简介 2](#_Toc6228)

[1.1 简介 2](#_Toc31527)

[1.2 特点 2](#_Toc4764)

[1.3 适合开发什么？ 4](#_Toc28897)

[1.4 Node.js无法挑战老牌3P 5](#_Toc31442)

[二、Node.js安装 5](#_Toc30380)

[三、HTTP模块 8](#_Toc24139)

[四、模块 9](#_Toc23899)

[五、post请求 13](#_Toc10254)

[六、模板引擎 13](#_Toc29497)

[先说EJS 13](#_Toc18204)

[七、Express框架 14](#_Toc17848)

[八、路由 16](#_Toc19401)

[九、中间件 17](#_Toc8670)

[十、传统数据库技术回顾 19](#_Toc23800)

[十一、NoSQL 21](#_Toc20752)

[十二、MongoDB安装 22](#_Toc11332)

[十三、MongoDB数据库使用 24](#_Toc15647)

[13.1 插入数据 25](#_Toc2136)

[13.2 查找数据 26](#_Toc15809)

[13.3 修改数据 26](#_Toc26362)

[13.4 删除数据 27](#_Toc8930)

[十四、数据库分页 27](#_Toc19618)

[十五、Cookie和Session 28](#_Toc27247)

[15.1 Cookie 28](#_Toc25998)

[15.2 Session 30](#_Toc17803)

[15.3 session 31](#_Toc1205)

[十六、加密 31](#_Toc28745)

[十七、图片处理 32](#_Toc7939)

[十八、索引index 33](#_Toc14437)

[十九、Mongoose 34](#_Toc19906)

[19.1 数据库连接 35](#_Toc12942)

[19.2 定义模型 35](#_Toc31019)

[二十、web Socket和Socket.IO框架 36](#_Toc23364)

[复习 39](#_Toc5215)

# 一、Node.js简介

## 1.1 简介

V8引擎本身就是用于Chrome浏览器的JS解释部分，但是Ryan Dahl这哥们，鬼才般的，把这个V8搬到了服务器上，用于做服务器的软件。

Node.js是一个专注于实现高性能Web服务器优化的专家，几经探索，几经挫折后，遇到V8而诞生的项目。

Node.js是一个让JavaScript运行在服务器端的开发平台，它让JavaScript的触角伸到了服务器端，可以与PHP、JSP、Python、Ruby平起平坐。

但Node似乎有点不同：

● **Node.js不是一种独立的语言**，与PHP、JSP、Python、Perl、Ruby的“既是语言，也是平台”不同，Node.js的**使用JavaScript进行编程**，运行在JavaScript引擎上（V8）。

● 与PHP、JSP等相比（PHP、JSP、.net都需要运行在服务器程序上，Apache、Naginx、Tomcat、IIS。

），**Node.js跳过了Apache、Naginx、IIS等HTTP服务器，它自己不用建设在任何服务器软件之上**。Node.js的许多设计理念与经典架构（LAMP = Linux + Apache + MySQL + PHP）有着很大的不同，可以提供强大的伸缩能力。一会儿我们就将看到，Node.js没有web容器。

Node.js自身哲学，是花最小的硬件成本，追求更高的并发，更高的处理性能。

官网：<https://nodejs.org/en/>

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

## 1.2 特点

所谓的特点，就是Node.js是如何解决服务器高性能瓶颈问题的。

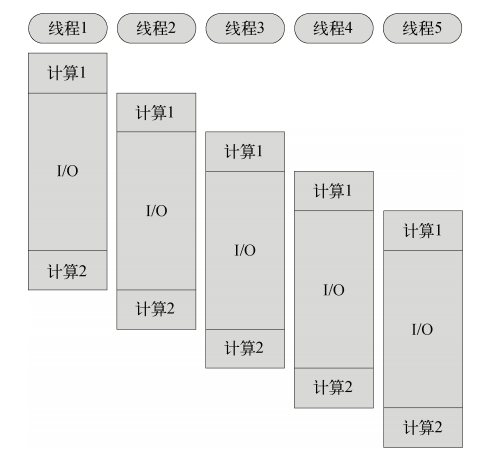
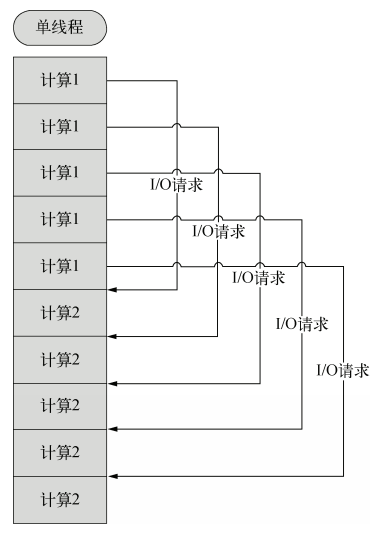
单线程

在Java、PHP或者.net等服务器端语言中，会为每一个客户端连接创建一个新的线程。而每个线程需要耗费大约2MB内存。也就是说，理论上，一个8GB内存的服务器可以同时连接的最大用户数为4000个左右。要让Web应用程序支持更多的用户，就需要增加服务器的数量，而Web应用程序的硬件成本当然就上升了。

Node.js不为每个客户连接创建一个新的线程，而仅仅使用一个线程。当有用户连接了，就触发一个内部事件，通过非阻塞I/O、事件驱动机制，让Node.js程序宏观上也是并行的。使用Node.js，一个8GB内存的服务器，可以同时处理超过4万用户的连接。

另外，带线程的带来的好处，还有操作系统完全不再有线程创建、销毁的时间开销。

坏处，就是一个用户造成了线程的崩溃，整个服务都崩溃了，其他人也崩溃了。

多线程、单线程的一个对比。

也就是说，单线程也能造成宏观上的“并发”。

非阻塞I/O non-blocking I/O

例如，当在访问数据库取得数据的时候，需要一段时间。在传统的单线程处理机制中，在执行了访问数据库代码之后，整个线程都将暂停下来，等待数据库返回结果，才能执行后面的代码。**也就是说，I/O阻塞了代码的执行，极大地降低了程序的执行效率。**

由于Node.js中采用了非阻塞型I/O机制，因此在执行了访问数据库的代码之后，将立即转而执行其后面的代码，把数据库返回结果的处理代码放在回调函数中，从而提高了程序的执行效率。

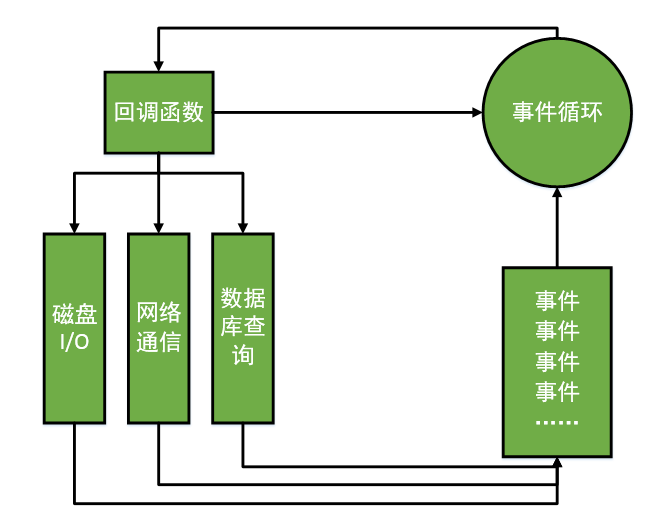
当某个I/O执行完毕时，将以事件的形式通知执行I/O操作的线程，线程执行这个事件的回调函数。为了处理异步I/O，线程必须有事件循环，不断的检查有没有未处理的事件，依次予以处理。

阻塞模式下，一个线程只能处理一项任务，要想提高吞吐量必须通过多线程。**而非阻塞模式下，一个线程永远在执行计算操作，这个线程的CPU核心利用率永远是100%。**所以，这是一种特别有哲理的解决方案：**与其人多，但是好多人闲着；还不如一个人玩命，往死里干活儿。**

事件驱动event-driven

在Node中，客户端请求建立连接，提交数据等行为，会触发相应的事件。在Node中，在一个时刻，只能执行一个事件回调函数，但是在执行一个事件回调函数的中途，可以转而处理其他事件（比如，又有新用户连接了），然后返回继续执行原事件的回调函数，这种处理机制，称为“事件环”机制。

Node.js底层是C++（V8也是C++写的）。**底层代码中，近半数都用于事件队列、回调函数队列的构建。**用事件驱动来完成服务器的任务调度，这是鬼才才能想到的。针尖上的舞蹈，用一个线程，担负起了处理非常多的任务的使命。



**单线程**，单线程的好处，减少了内存开销，操作系统的内存换页。

如果某一个事情，进入了，但是被I/O阻塞了，所以这个线程就阻塞了。

**非阻塞I/O**， 不会傻等I/O语句结束，而会执行后面的语句。

非阻塞就能解决问题了么？比如执行着小红的业务，执行过程中，小刚的I/O回调完成了，此时怎么办？？

**事件机制，事件环**，不管是新用户的请求，还是老用户的I/O完成，都将以事件方式加入事件环，等待调度。

说是三个特点，实际上是一个特点，离开谁都不行，都玩儿不转了。

Node.js很像抠门的餐厅老板，只聘请1个服务员，服务很多人。结果，比很多服务员效率还高。

Node.js中所有的I/O都是异步的，回调函数，套回调函数。

## 1.3 适合开发什么？

Node.js适合用来开发什么样的应用程序呢？

善于I/O，不善于计算。因为Node.js最擅长的就是任务调度，如果你的业务有很多的CPU计算，实际上也相当于这个计算阻塞了这个单线程，就不适合Node开发。

**当应用程序需要处理大量并发的I/O，而在向客户端发出响应之前，应用程序内部并不需要进行非常复杂的处理的时候，Node.js非常适合。Node.js也非常适合与web socket配合，开发长连接的实时交互应用程序。**

比如：

● 用户表单收集

● 考试系统

● 聊天室

● 图文直播

● 提供JSON的API（为前台Angular使用）

## 1.4 Node.js无法挑战老牌3P



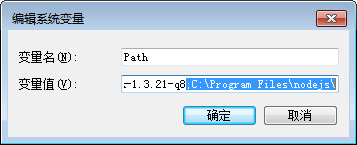
# **二、Node.js安装**

Node.js和Java非常像，跨平台的。不管是Linux还是windows编程是完全一致的（有一些不一样，比如路径的表述）。Linux版本的Node.js环境和windows环境是不一样的，但是编程语言一样。很像Java虚拟机。

我们学习的是最新版，2015年9月19日最新版本4.1.0。

安装包的大小只有10M不到，真的就是一个小玩具。

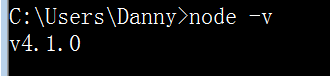
装完之后，我们在系统的环境变量里面，我们看一下：



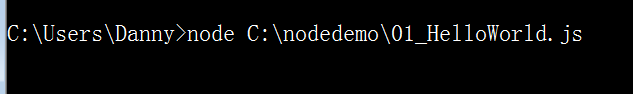
环境变量，就已经自动的填写进去了，就是我们node安装的目录。

什么叫做环境变量？就是在系统的任何目录下，都能运行c:\program files\nodejs里面的程序。

在cmd中，输入node -v就能够查看nodejs的版本。你会发现，我们现在的盘符，不在安装目录下，但是也能够运行，这就是因为有系统环境变量。系统的环境变量已经有了c:\program files\nodejs了，所以，这个文件夹中的node.exe就能够在任何盘符运行。



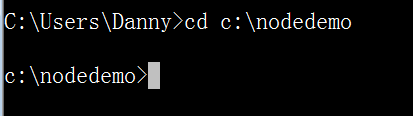
运行文件，就要用node命令来运行：

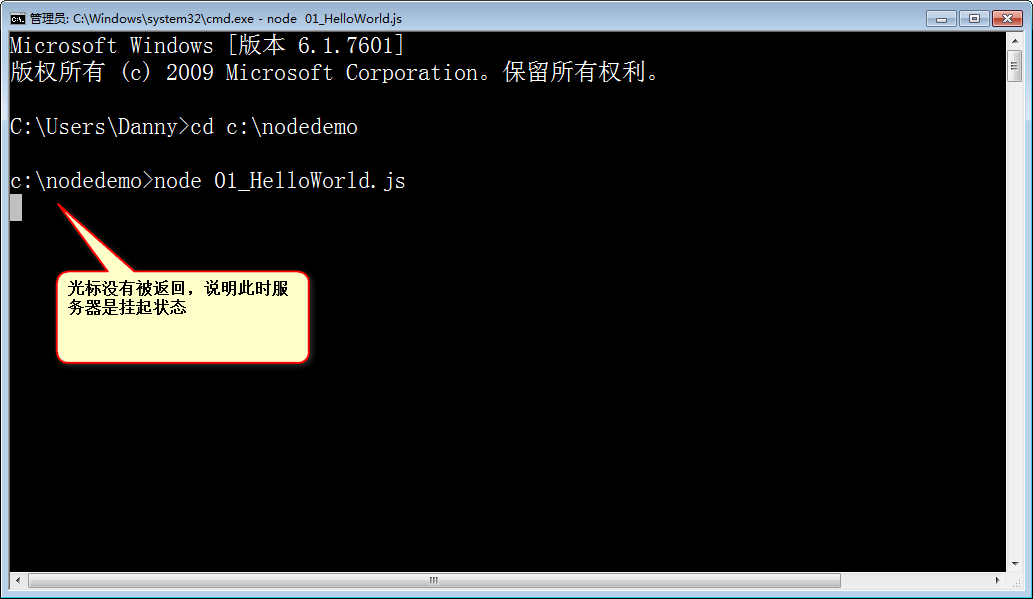


|  |
| --- |
| 1. node 路径名 |

推荐，不要使用完整的路径名，而是通过先进入案例文件夹，然后node相对地址

cd命令，就是change directory的缩写，表示更改当前目录。





Node.js是服务器的程序，写的js语句，都将运行在服务器上。返回给客户的，都是已经处理好的纯html。

|  |
| --- |
| 1. //require表示引包，引包就是引用自己的一个特殊功能 2. var http = require("http"); 3. //创建服务器，参数是一个回调函数，表示如果有请求进来，要做什么 4. var server = http.createServer(function(req,res){ 5. //req表示请求，request; res表示响应，response 6. //设置HTTP头部，状态码是200，文件类型是html，字符集是utf8 7. res.writeHead(200,{"Content-type":"text/html;charset=UTF-8"}); 8. res.end("哈哈哈哈，我买了一个iPhone" + (1+2+3) + "s"); 9. }); 10. //运行服务器，监听3000端口（端口号可以任改） 11. server.listen(3000,"127.0.0.1"); |

如果想修改程序，必须中断当前运行的服务器，重新node一次，刷新，才行。

**ctrl+c，就可以打断挂起的服务器程序。**此时按上箭头，能够快速调用最近的node命令。

你会发现，我们本地写一个js，打死都不能直接拖入浏览器运行，但是有了node，我们任何一个js文件，都可以通过node来运行。**也就是说，node就是一个js的执行环境。**

我们现在，要跑起来一个服务器，这个服务器的脚本，要以.js存储。是一个js文件。用node命令运行这个js文件罢了。

Node.js没有根目录的概念，因为它根本没有任何的web容器！

让node.js提供一个静态服务，都非常难！

也就是说，node.js中，如果看见一个网址是

|  |
| --- |
| 1. 127.0.0.1:3000/fang |

别再去想，一定有一个文件夹，叫做fang了。可能/fang的物理文件，是同目录的test.html

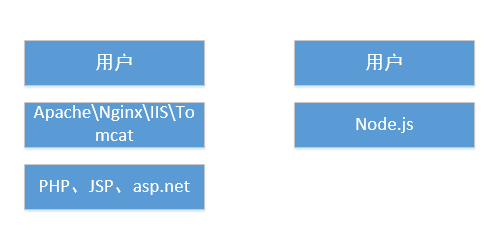
URL和真实物理文件，是没有关系的。URL是通过了Node的顶层路由设计，呈递某一个静态文件的。

# 三、HTTP模块

Node.js中，将很多的功能，划分为了一个个mudule，大陆的书翻译为模块；台湾的书，翻译为模组。

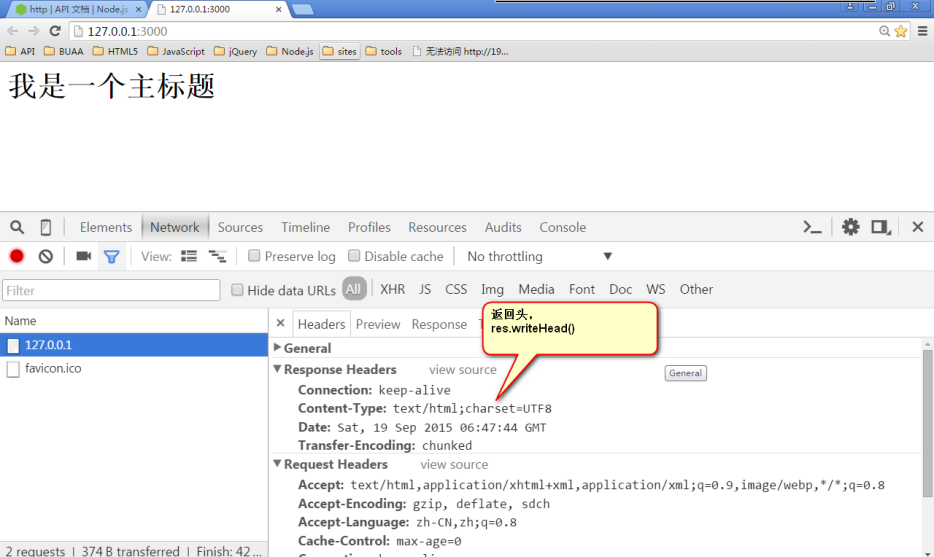
这是因为，有一些程序需要使用fs功能（文件读取功能），有一些不用的，所以为了效率，你用啥，你就require啥。

|  |
| --- |
| 1. //这个案例简单讲解http模块 2. //引用模块 3. var http = require("http"); 4. //创建一个服务器，回调函数表示接收到请求之后做的事情 5. var **server** = http.**createServer**(function(req,res){ 6. //req参数表示请求，res表示响应 7. console.log("服务器接收到了请求" + req.url); 8. res.end(); 9. }); 10. //监听端口 11. **server.listen**(3000,"127.0.0.1"); |



设置一个响应头：

|  |
| --- |
| 1. res.writeHead(200,{"Content-Type":"text/plain;charset=UTF8"}); |



我们现在来看一下req里面能够使用的东西。

最关键的就是req.url属性，表示用户的请求URL地址。所有的路由设计，都是通过req.url来实现的。

我们比较关心的不是拿到URL，而是识别这个URL。

识别URL，用到两个新模块，第一个就是url模块，第二个就是querystring模块

字符串查询，用querystring处理

|  |
| --- |
| 1. querystring.parse('foo=bar&baz=qux&baz=quux&corge') 2. // returns 3. { foo: 'bar', baz: ['qux', 'quux'], corge: '' } 4. // Suppose gbkDecodeURIComponent function already exists, 5. // it can decode `gbk` encoding string 6. querystring.parse('w=%D6%D0%CE%C4&foo=bar', null, null, 7. { decodeURIComponent: gbkDecodeURIComponent }) 8. // returns 9. { w: '中文', foo: 'bar' } |



# 四、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. }); |

# 五、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"> |



# 六、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. }; |

# 七、Express框架

Express框架是后台的Node框架，所以和jQuery、zepto、yui、bootstrap都不一个东西。

Express在后台的受欢迎的程度，和jQuery一样，就是企业的事实上的标准。

● 原生Node开发，会发现有很多问题。比如：

■ 呈递静态页面很不方便，需要处理每个HTTP请求，还要考虑304问题

■ 路由处理代码不直观清晰，需要写很多正则表达式和字符串函数

■ 不能集中精力写业务，要考虑很多其他的东西

我们自己可以把第一天的作业，就是那个静态文件服务给封装成为模块。封装的越多，就自己做出了类似Express的东西。

● EXPRESS的哲学是在你的想法和服务器之间充当薄薄的一层。这并不意味着他不够健壮，或者没有足够的有用特性，而是尽量少干预你，让你充分表达自己的思想，同时提供一些有用的东西。

英语官网：http://expressjs.com/

中文官网：<http://www.expressjs.com.cn/>

整体感知，Express框架。

安装Express框架，就是使用npm的命令。

|  |
| --- |
| 1. npm install **--save** express |

--save参数，表示自动修改package.json文件，自动添加依赖项。

路由能力：

|  |
| --- |
| 1. var express = require("express"); 2. var app = express(); 3. app.get("/",function(req,res){ 4. res.send("你好"); 5. }); 6. app.get("/haha",function(req,res){ 7. res.send("这是haha页面，哈哈哈哈哈哈"); 8. }); 9. app.get(/^\/student\/([\d]{10})$/,function(req,res){ 10. res.send("学生信息，学号" + req.params[0]); 11. }); 12. app.get("/teacher/:gonghao",function(req,res){ 13. res.send("老师信息，工号" + req.params.gonghao); 14. }); 15. app.listen(3000); |

静态文件伺服能力：

|  |
| --- |
| 1. app.use(express.static("./public")); |

模板引擎：

|  |
| --- |
| 1. var express = require("express"); 2. var app = express(); 3. app.set("view engine","ejs"); 4. app.get("/",function(req,res){ 5. res.render("haha",{ 6. "news" : ["我是小新闻啊","我也是啊","哈哈哈哈"] 7. }); 8. }); 9. app.listen(3000); |

我们学习的是Express4.X，和Express3.X差别非常大。

# 八、路由

当用get请求访问一个网址的时候，做什么事情：

|  |
| --- |
| 1. app.get("网址",function(req,res){ 3. }); |

当用post访问一个网址的时候，做什么事情：

|  |
| --- |
| 1. app.post("网址",function(req,res){ 3. }); |

如果想处理这个网址的任何method的请求，那么写all

|  |
| --- |
| 1. app.all("/",function(){ 3. }); |

这里的网址，不分大小写，也就是说，你路由是

|  |
| --- |
| 1. app.get("/AAb",function(req,res){ 2. res.send("你好"); 3. }); |

实际上小写的访问也行。

所有的GET参数，? 后面的都已经被忽略。 锚点#也被忽略

你路由到/a ， 实际/a?id=2&sex=nan 也能被处理。

正则表达式可以被使用。正则表达式中，未知部分用圆括号分组，然后可以用req.params[0]、[1]得到。

req.params类数组对象。

|  |
| --- |
| 1. app.get(**/^\/student\/([\d]{10})$/**,function(req,res){ 2. res.send("学生信息，学号" + **req.params[0]**); 3. }); |

冒号是更推荐的写法。

|  |
| --- |
| 1. app.get("/student/:id",function(req,res){ 2. var id = req.params["id"]; 3. var reg= /^[\d]{6}$/; //正则验证 4. if(reg.test(id)){ 5. res.send(id); 6. }else{ 7. res.send("请检查格式"); 8. } 9. }); |

表单可以自己提交到自己上。

|  |
| --- |
| 1. app.get("/",function(req,res){ 2. res.render("form"); 3. }); 4. app.post("/",function(req,res){ 5. //将数据添加进入数据库 6. res.send("成功"); 7. }); |

适合进行 RESTful路由设计。简单说，就是一个路径，但是http method不同，对这个页面的使用也不同。

/student/345345

get 读取学生信息

add 添加学生信息

delete 删除学生新

# 九、中间件

如果我的的get、post回调函数中，没有next参数，那么就匹配上第一个路由，就不会往下匹配了。

如果想往下匹配的话，那么需要写**next()**

|  |
| --- |
| 1. app.get("/",function(req,res,**next**){ 2. console.log("1"); 3. **next();** 4. }); 5. app.get("/",function(req,res){ 6. console.log("2"); 7. }); |

下面两个路由，感觉没有关系：

|  |
| --- |
| 1. app.get("**/:username/:id**",function(req,res){ 2. console.log("1"); 3. res.send("用户信息" + req.params.username); 4. }); 5. app.get("**/admin/login**",function(req,res){ 6. console.log("2"); 7. res.send("管理员登录"); 8. }); |

但是实际上冲突了，因为admin可以当做用户名 login可以当做id。

解决方法1：交换位置。 也就是说，express中所有的路由（中间件）的顺序至关重要。

匹配上第一个，就不会往下匹配了。 具体的往上写，抽象的往下写。

|  |
| --- |
| 1. app.get("/admin/login",function(req,res){ 2. console.log("2"); 3. res.send("管理员登录"); 4. }); 5. app.get("/:username/:id",function(req,res){ 6. console.log("1"); 7. res.send("用户信息" + req.params.username); 8. }); |

解决方法2：

|  |
| --- |
| 1. app.get("/:username/:id",function(req,res,**next**){ 2. var username = req.params.username; 3. //检索数据库，如果username不存在，那么next() 4. if(检索数据库){ 5. console.log("1"); 6. res.send("用户信息"); 7. }else{ 8. **next();** 9. } 10. }); 11. app.get("/admin/login",function(req,res){ 12. console.log("2"); 13. res.send("管理员登录"); 14. }); |

路由get、post这些东西，就是中间件，中间件讲究顺序，匹配上第一个之后，就不会往后匹配了。next函数才能够继续往后匹配。

app.use()也是一个中间件。与get、post不同的是，他的网址不是精确匹配的。而是能够有小文件夹拓展的。

比如网址： http://127.0.0.1:3000/admin/aa/bb/cc/dd

|  |
| --- |
| 1. app.use("/admin",function(req,res){ 2. res.write(req.originalUrl + "\n"); // /admin/aa/bb/cc/dd 3. res.write(req.baseUrl + "\n"); // /admin 4. res.write(req.path + "\n"); // /aa/bb/cc/dd 5. res.end("你好"); 6. }); |

如果写一个/

|  |
| --- |
| 1. //当你不写路径的时候，实际上就相当于"/"，就是所有网址 2. app.use(function(req,res,next){ 3. console.log(new Date()); 4. next(); 5. }); |

app.use()就给了我们增加一些特定功能的便利场所。

实际上app.use()的东西，基本上都从第三方能得到。

● 大多数情况下，渲染内容用res.render()，将会根据views中的模板文件进行渲染。如果不想使用views文件夹，想自己设置文件夹名字，那么app.set("views","aaaa");

● 如果想写一个快速测试页，当然可以使用res.send()。这个函数将根据内容，自动帮我们设置了Content-Type头部和200状态码。send()只能用一次，和end一样。和end不一样在哪里？能够自动设置MIME类型。

● 如果想使用不同的状态码，可以：

res.status(404).send('Sorry, we cannot find that!');

● 如果想使用不同的Content-Type，可以：

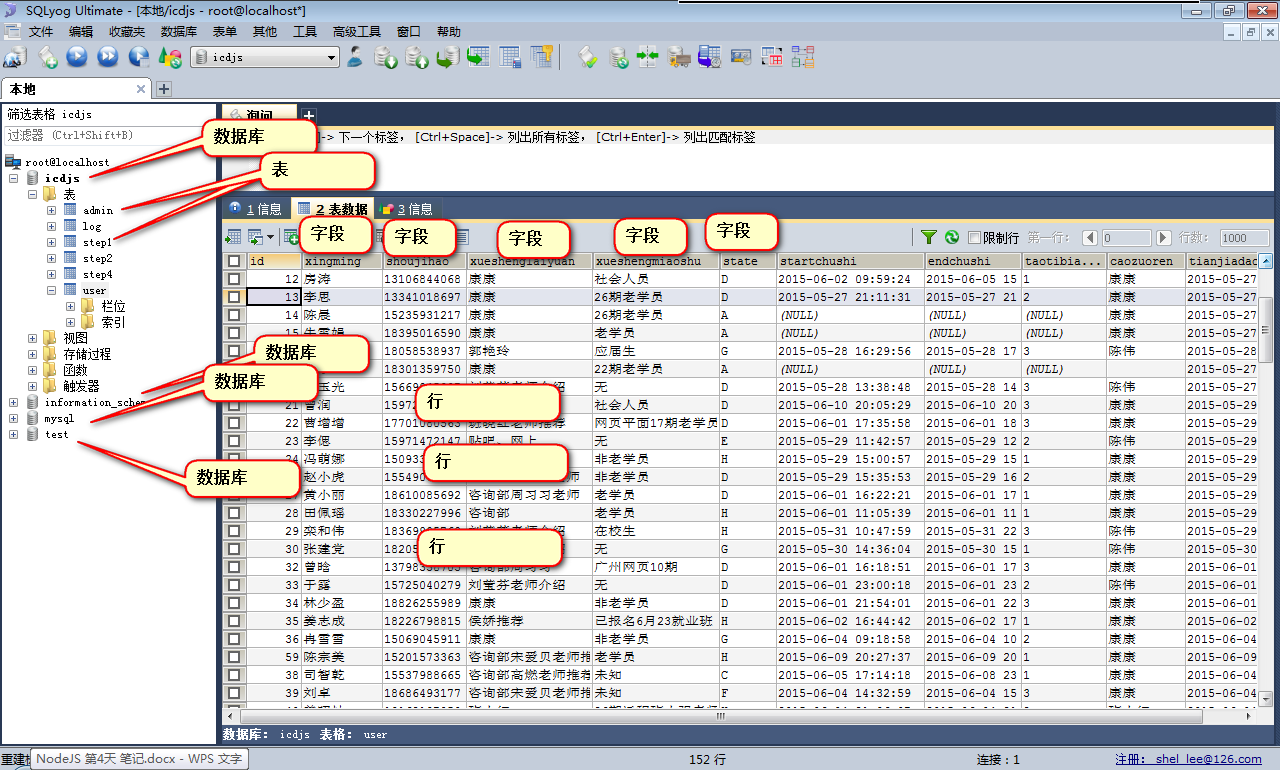
res.set('Content-Type', 'text/html');

# 十、传统数据库技术回顾

数据库就是存储数据的，那么存储数据就用txt就行了啊，为什么要有数据库？

理由之1： 数据库有行、列的概念，数据有关系，数据不是散的。

老牌数据库，比如MySQL、SQL Server、Oracle、Access。这些数据库，我们管他们叫做结构型数据库。为什么？因为每个表中，都有明确的字段，每行记录，都有这些字段。不能有的行有，有的行没有。



理由二：数据库能够提供非常方便的接口，让增删改查操作变得简单

我们的老牌数据库，都无一例外的使用SQL语言，管理数据库。

SQL就是structure query language。

比如，查询所有女生： SELECT \* FROM step1 WHERE xingbie = '女';

再比如，查询所有女生，并且年龄20~24之间，且在北京：

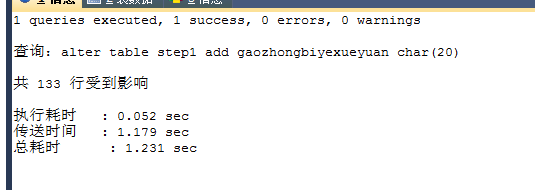
|  |
| --- |
| 1. SELECT \* FROM step1 WHERE xingbie = '女' AND nianling < 24 AND nianling >= 20 AND xianzaisuozaidi = '北京'; |

理由之三：数据库不能自己玩儿，要给向PHP、.net、jsp等语言提供接口。

用php这些语言，能够向数据库之中增删改查。

老牌数据库，都是结构型数据库，现在出了什么问题？

比如，我们现在想往一个已经有1000条数据的数据库中增加一个字段“高中信息”。



之前已经存在的数据，实际上不需要增加这个字段。因为这些用户已经填写完毕表单了，不需要再手机高中信息了。我们的意图就是在今后注册的用户，需要填写高中信息。但是，我们刚才说了，所谓的字段，是表的一个结构。所有的行都必须拥有，不能有的行有这个字段，有的行没有这个字段。

可想而知，大数据时代，数据库中有100万条数据都算少的。我们如果要动字段，时间太长。

**所以，字段这个东西，太不灵活。**

**数据不灵活。一个字段，需要是同样类型的数据。**不能一行记录是文本，一行记录是数字。

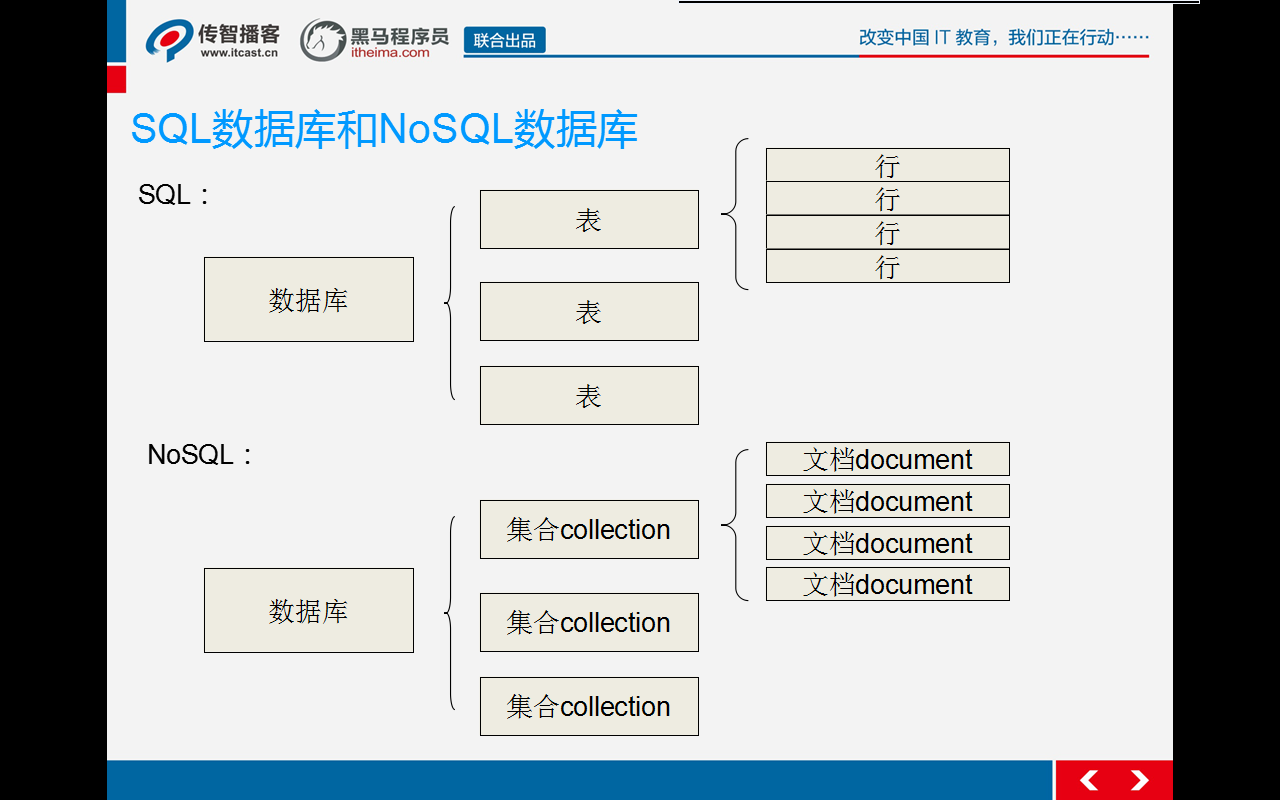
非结构型数据库NoSQL应运而生。

NoSQL是个怪胎，无法挑战老牌数据库，但是在大数据时代有自己的意义。

# 十一、NoSQL

非结构型数据库。**没有行、列的概念。用JSON来存储数据。**

**集合就相当于“表”，文档就相当于“行”。**



文档就是JSON，上下文语境中，也是JavaScript范畴，所以我们的数据库也是JS范畴的东西，JS全栈。



因此，我们总结NoSQL数据库在以下的这几种情况下比较适用：1、数据模型比较简单；2、需要灵活性更强的IT系统；3、对数据库性能要求较高；4、不需要高度的数据一致性；5、对于给定key，比较容易映射复杂值的环境。

我们看，有些系统，特别需要筛选。比如，筛选出所有女生大于20岁的。那么SQL型数据库，非常擅长！因为它有行、列的概念。

但是，有些系统，真的不需要进行那么多的筛选，比如站内信。站内信只需要存储就好了。不需要筛选。那么NoSQL的。

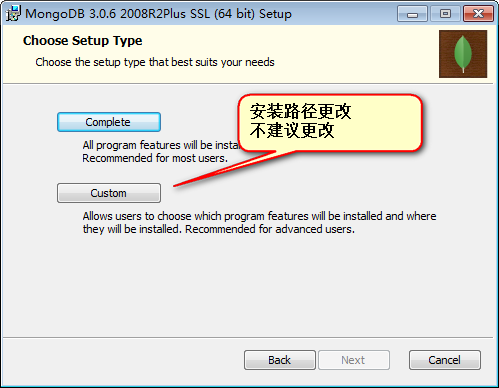
NoSQL不是银弹，没有资格挑战老牌数据库，还是特定情况下，是适合的。

# 十二、MongoDB安装

官网：<https://www.mongodb.com/>

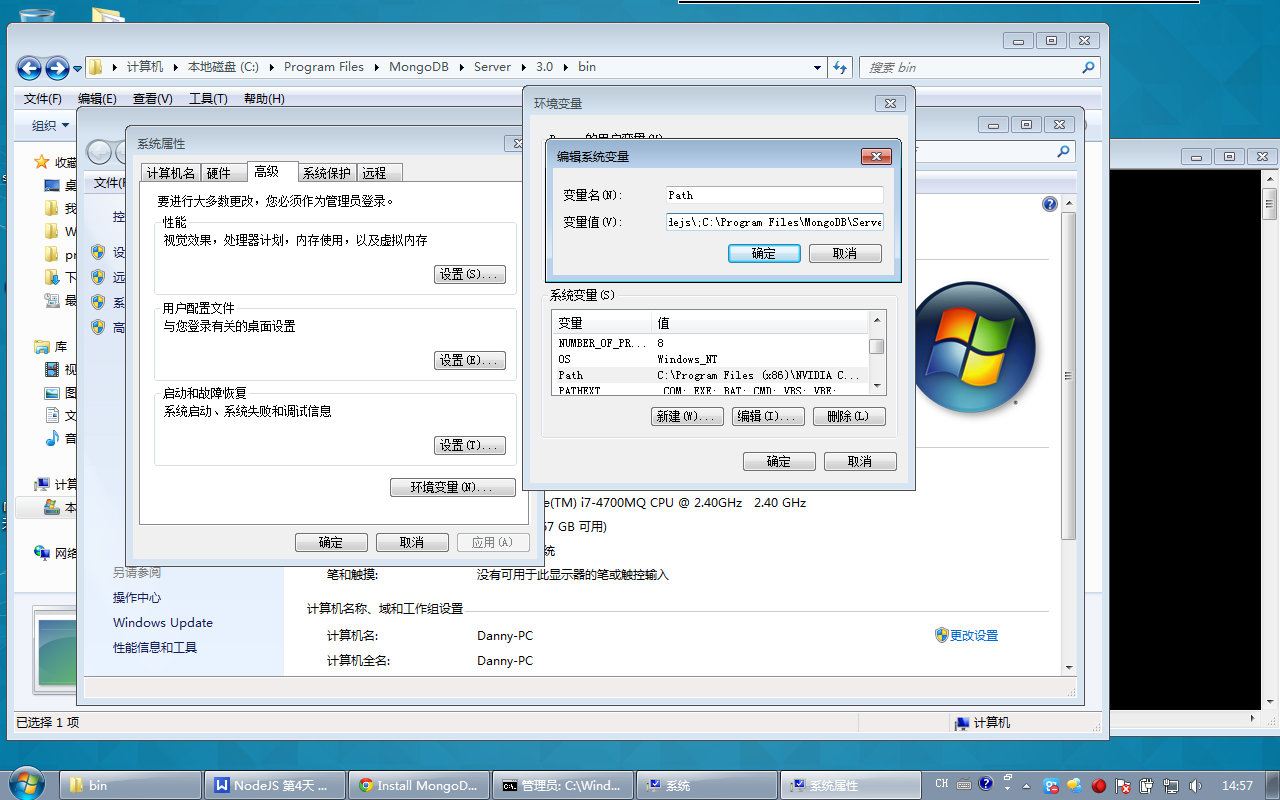
手册：<https://docs.mongodb.org/manual/>

win7系统需要安装补丁，KB2731284。



此时，我们看一下装好的文件夹：

C:\Program Files\MongoDB\Server\3.0\bin 加入到系统的path环境变量中



那么我们就能在系统的任何盘符，使用mongo命令了：

mongo 使用数据库

mongod 开机

mongoimport 导入数据

开机命令：



--dbpath就是选择数据库文档所在的文件夹。

也就是说，mongoDB中，真的有物理文件，对应一个个数据库。U盘可以拷走。

一定要保持，开机这个CMD不能动了，不能关，不能ctrl+c。 一旦这个cmd有问题了，数据库就自动关闭了。

所以，**应该再开一个cmd**。输入



那么，运行环境就是mongo语法了。

列出所有数据库：

|  |
| --- |
| 1. show dbs |

使用某个数据库

|  |
| --- |
| 1. use 数据库名字 |

如果想新建数据库，也是use。use一个不存在的，就是新建。

查看当前所在数据库

|  |
| --- |
| 1. db |

插入数据：



student就是所谓的集合。集合中存储着很多json。

student是第一次使用，集合将自动创建。

# 十三、MongoDB数据库使用

要管理数据库，必须先开机，开机使用**mongod --dbpath c:\mongo**

管理数据库：mongo (一定要在新的cmd中输入)



清屏：

|  |
| --- |
| 1. cls |

查看所有数据库列表

|  |
| --- |
| 1. show dbs |

使用数据库、创建数据库

|  |
| --- |
| 1. use itcast |

如果真的想把这个数据库创建成功，那么必须插入一个数据。

数据库中不能直接插入数据，只能往集合(collections)中插入数据。不需要创建集合，只需要写点语法：

|  |
| --- |
| 1. db.student.insert({“name”:”xiaoming”}); |

db.student 系统发现student是一个陌生的集合名字，所以就自动创建了集合。

删除数据库，删除当前所在的数据库

|  |
| --- |
| 1. db.dropDatabase(); |

## 13.1 插入数据

插入数据，随着数据的插入，数据库创建成功了，集合也创建成功了。

|  |
| --- |
| 1. **db.student.insert({"name":"xiaoming"});** |

我们不可能一条一条的insert。所以，我们希望用sublime在外部写好数据库的形式，然后导入数据库：

|  |
| --- |
| 1. mongoimport **--db test** **--collection restaurants** **--drop** **--file primer-dataset.json** |

**-db test 想往哪个数据库里面导入**

**--collection restaurants 想往哪个集合中导入**

**--drop 把集合清空**

**--file primer-dataset.json 哪个文件**

**这样，我们就能用sublime创建一个json文件，然后用mongoimport命令导入，这样学习数据库非常方便。**

## 13.2 查找数据

查找数据，用find。find中没有参数，那么将列出这个集合的所有文档：

|  |
| --- |
| 1. db.restaurants.find() |

精确匹配：

|  |
| --- |
| 1. db.student.find({"score**.**shuxue":70}); |

多个条件：

|  |
| --- |
| 1. db.student.find({"score.shuxue":70 , "age":12}) |

大于条件：

|  |
| --- |
| 1. db.student.find({"score.yuwen":**{$gt:50}**}); |

或者。寻找所有年龄是9岁，或者11岁的学生

|  |
| --- |
| 1. db.student.find({**$or**:[{"age":9},{"age":11}]}); |

查找完毕之后，打点调用sort，表示升降排序。

|  |
| --- |
| 1. db.restaurants.find().**sort(** { "borough": 1, "address.zipcode": 1 } **)** |

## 13.3 修改数据

修改里面还有查询条件。你要该谁，要告诉mongo。

查找名字叫做小明的，把年龄更改为16岁：

|  |
| --- |
| 1. db.student.update({"name":"小明"},{$set:{"age":16}}); |

查找数学成绩是70，把年龄更改为33岁：

|  |
| --- |
| 1. db.student.update({"score.shuxue":70},{$set:{"age":33}}); |

更改所有匹配项目："

By default, the update() method updates a single document. To update multiple documents, use the multi option in the update() method.

|  |
| --- |
| 1. db.student.update({"sex":"男"},{$set:{"age":33}},{multi: true}); |

完整替换，不出现$set关键字了：

|  |
| --- |
| 1. db.student.update({"name":"小明"},{"name":"大明","age":16}); |

## 13.4 删除数据

|  |
| --- |
| 1. db.restaurants.remove( { "borough": "Manhattan" } ) |

By default, the remove() method removes all documents that match the remove condition. Use the justOne option to limit the remove operation to only one of the matching documents.

|  |
| --- |
| 1. db.restaurants.remove( { "borough": "Queens" }, **{ justOne: true }** ) |

# 十四、数据库分页

分页，想想我们的百度百家Ajax案例，当时调用了百度的JSON，有一个参数叫做page=3，生成的JSON不一样。

这个就是分页，就是我们想寻找所有的新闻，但是是位于第3页的新闻。那么有两种做法：

1） 错误的做法： 就是讲所有的result都读取到数组，然后进行数据操作，进行分页；

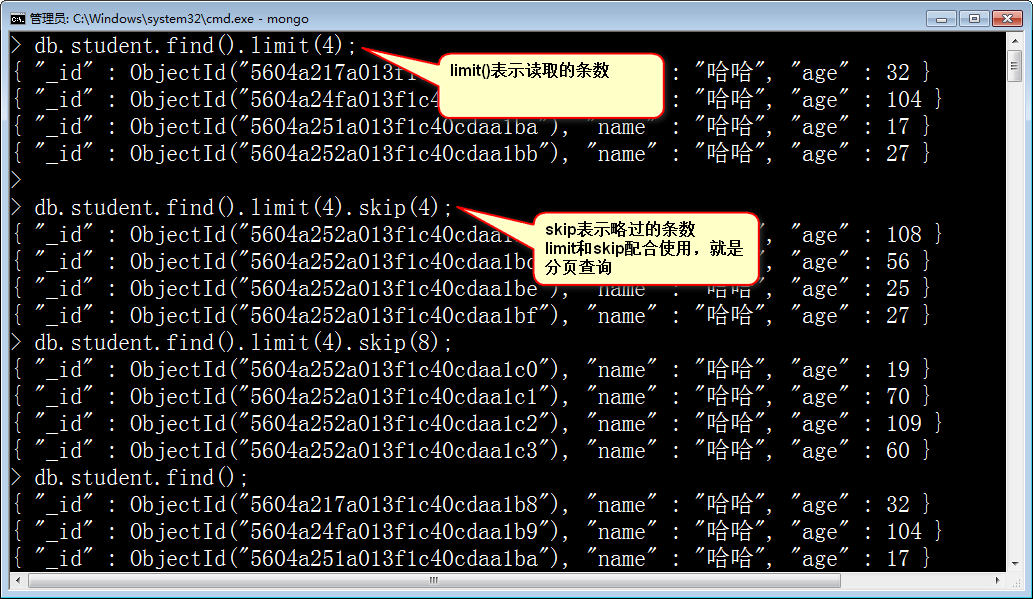
2） 正确的做法： 就是真的在数据库中，只读取这么多内容。

错误的，我们试图每次都读取全部数据，但是这样开销很大。

|  |
| --- |
| 1. var a = []; 2. db.find("student",{},function(err,result){ 3. for(var i = 10 \* page ; i < 10 \* (page + 1) ; i++){ 4. a.push(result[i]); 5. } 6. res.send(a); 7. }); |

所以，mongodb提供了傻傻的两个函数。

limit() skip()



加入，第一页是page=0。每页10条，所以当前页的查询语句

|  |
| --- |
| 1. db.student.find({}).limit(10).skip(page\*10) |

数据总数怎么得到？

shell中

|  |
| --- |
| 1. db.student.stats().count; |

ejs页面中，如果我们想使用underscore的模板，就会有模板冲突的问题。

underscore用的：

|  |
| --- |
| 1. <script type="text/template" id="moban"> 2. <div class="list-group"> 3. <a href="#" class="list-group-item active"> 4. <h4 class="list-group-item-heading">**<%= xingming %>**</h4> 5. <p class="list-group-item-text">**<%= liuyan %>**</p> 6. </a> 7. </div> 8. </script> |

ejs以为是自己的模板。所以报错，提示你没有船体xingming参数。

解决方法就是underscore源码。在源码中搜索<%即可。

# 十五、Cookie和Session

## 15.1 Cookie

● HTTP是无状态协议。简单地说，当你浏览了一个页面，然后转到同一个网站的另一个页面，服务器无法认识到，这是同一个浏览器在访问同一个网站。每一次的访问，都是没有任何关系的。

那么世界就乱套了，比如我上一次访问，登陆了，下一次访问，又让我登陆，不存在登陆这事儿了。

● Cookie是一个简单到爆的想法：当访问一个页面的时候，服务器在下行HTTP报文中，命令浏览器存储一个字符串；浏览器再访问同一个域的时候，将把这个字符串携带到上行HTTP请求中。

第一次访问一个服务器，不可能携带cookie。 必须是服务器得到这次请求，在下行响应报头中，携带cookie信息，此后每一次浏览器往这个服务器发出的请求，都会携带这个cookie。

特点

● cookie是不加密的，用户可以自由看到；

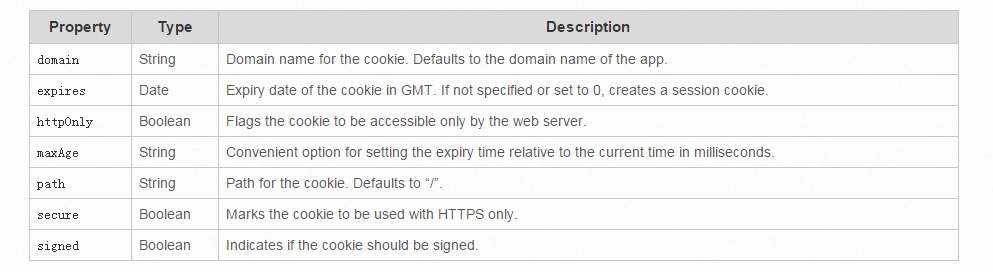
● 用户可以删除cookie，或者禁用它

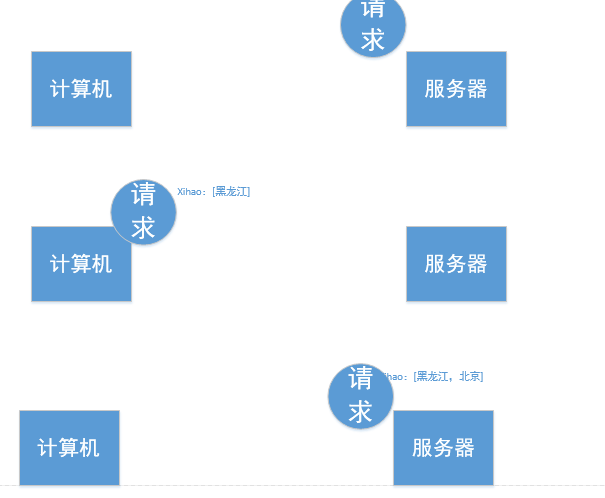
● cookie可以被篡改

● cookie可以用于攻击

● cookie存储量很小。未来实际上要被localStorage替代，但是后者IE9兼容。

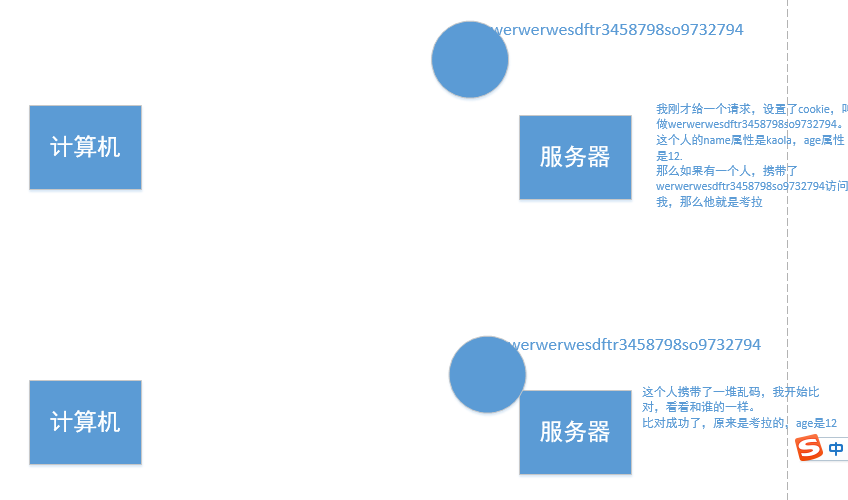
express中的cookie，你肯定能想到。 res负责设置cookie， req负责识别cookie。





## 15.2 Session

会话。 Session不是一个天生就有的技术，而是依赖cookie。



session依赖cookie，当一个浏览器禁用cookie的时候，登陆效果消失； 或者用户清除了cookie，登陆也消失。

session比cookie不一样在哪里呢？ session下发的是乱码，并且服务器自己缓存一些东西，下次浏览器的请求带着乱码上来，此时与缓存进行比较，看看是谁。

所以，一个乱码，可以对应无限大的数据。

任何语言中，session的使用，是“机理透明”的。他是帮你设置cookie的，但是足够方便，让你感觉不到这事儿和cookie有关。

## 15.3 session

|  |
| --- |
| 1. var session = require("express-session"); 2. app.use(session({ 3. secret: 'keyboard cat', 4. resave: false, 5. saveUninitialized: true 6. })) 7. app.get("/",function(req,res){ 8. if(**req.session.login == "1"**){ 9. res.send("欢迎" + **req.session.username**); 10. }else{ 11. res.send("没有成功登陆"); 12. } 13. }); 14. app.get("/login",function(req,res){ 15. **req.session.login = "1"**; //设置这个session 16. req.session.username = "考拉"; 17. res.send("你已经成功登陆"); 18. }); |

# 十六、加密

**永远不要用明码写密码**。CSDN今年泄露用户密码了，并且泄露的明码。

黑客拿到的用户的密码的加密信息，所以也没用。因为他无法翻译成为明码。

MD5加密是函数型加密。就是每次加密的结果一定相同，没有随机位。

特点：

● 不管加密的文字，多长多短，永远都是32位英语字母、数字混合。

● 哪怕只改一个字，密文都会大变。

● MD5没有反函数破解的可能，网上的破解工具，都是通过字典的模式，通过大量列出明-密对应的字典，找到明码。两次加密网上也有对应的字典。所以我们不要直接用一层md5，这样对黑客来说和明码是一样。

MD5常用于作为版本校验。可以比对两个软件、文件是否完全一致。

node中自带了一个模块，叫做crypto模块，负责加密。

首先创建hash，然后update和digest：

|  |
| --- |
| 1. var md5 = crypto.createHash('md5'); 2. var password = md5.update(fields.password).digest('base64'); |
|  |

加密使用的是MD5加密。

1 C4CA4238A0B923820DCC509A6F75849B

2 C81E728D9D4C2F636F067F89CC14862C

我爱北京天安没和陈伟：

97AFBA2EE71DE6A78B61149A391A378F

我爱北京天安没与陈伟：

CCB721C1E26067BBF4BB49DFDBFA0C9E

我爱北京天安没和陈猥：

C3D110D2441EF9323D3F1E6082E708F8

不管你加密多大的东西，哪怕10M文字，都会加密为32位的字符串，就是密码。并且神奇的，数学上能够保证，哪怕你更改1个文字，都能大变。所以MD5也能用于比对版本。

MD5是数学上，不能破解的。 不能反向破解。

也就是说，C4CA4238A0B923820DCC509A6F75849B 没有一个函数，能够翻译成为1的。

但是，有的人做数据库，就是把1~999999所有数字都用MD5加密了，然后进行了列表，所以有破解的可能。

# 十七、图片处理

<http://www.graphicsmagick.org/>

GraphicsMagick is the swiss army knife of image processing. 瑞士军刀。

只要服务器需要处理图片，那么这个服务器就要安装graphicsmagick软件。免费的。

装完之后，可视化工具一点用都没有，从桌面上删除。我们要把安装目录设置为环境变量。

控制台CMD命令：

|  |
| --- |
| 1. //格式转换 2. gm convert a.bmp a.jpg |

|  |
| --- |
| 1. //更改当前目录下\*.jpg的尺寸大小，并保存于目录.thumb里面 2. gm mogrify -resize 320x200 danny.jpg |

nodejs要使用graphicsmagick，需要npm装一个gm的包。

node.js命令

node.js缩略图的制作：

|  |
| --- |
| 1. var fs = require('fs'); 2. var gm = require('gm'); 3. gm('./danny.jpg') 4. .resize(50, 50,"!") 5. .write('./danny2.jpg', function (err) { 6. if (err) { 7. console.log(err); 8. } 9. }); |

node.js头像裁切：

|  |
| --- |
| 1. gm("./danny.jpg")**.crop(141,96,152,181)**.write("./2.jpg",function(err){ 2. **//141 96 是宽高 。 152 181是坐标** 3. }); |

# 十八、索引index

数据库中，根据一个字段的值，来寻找一个文档，是很常见的操作。比如根据学号来找一个学生。

这个学号，是唯一的，只要有学号，就能唯一确认一个学生的文档。学号这个属性，就非常适合建立索引，这样一来，查找学生就变得简单了。

这个语句，能够查看检索的过程：

|  |
| --- |
| 1. db.student.find({"name":"user888"}); |

学生的姓名是唯一的，为了快速的进行检索，所以就把name属性建立成为“索引”。

|  |
| --- |
| 1. db.student.createIndex({"name":1}); |

这样，今后通过name寻找student文档的时候，速度非常快。因为能够快速的从索引表中，找到这个文档。

缺点就是插入每条数据的时候，时间变慢了，效率低了。但是换回来的就是寻找的速度快了。

索引这个属性，所有的文档都不能相同：

|  |
| --- |
| 1. db.members.createIndex( { "user\_id": 1 }, **{ unique: true }** ); |

# 十九、Mongoose

是一个将JavaScript对象与数据库产生关系的一个框架，object related model。操作对象，就是操作数据库了；对象产生了，同时也持久化了。

这个思路是Java三大框架SSH中Hibernate框架的思路。彻底改变了人们使用数据库的方式。

<http://mongoosejs.com/>

mongoose就是:



官网上的hello world：

|  |
| --- |
| 1. //引包，**并不需要引用mongodb这个包** 2. var mongoose = require('mongoose'); 3. //链接数据库,**haha是数据库名字** 4. mongoose.connect('mongodb://localhost**/haha**'); 5. //创建了一个模型。猫的模型。所有的猫，都有名字，是字符串。“类”。 6. var Cat = mongoose.model('Cat', { name: String }); 7. //实例化一只猫 8. var kitty = new Cat({ name: 'Zildjian' }); 9. //调用这只猫的save方法，保存这只猫 10. kitty.save(function (err) { 11. console.log('喵喵喵'); 12. }); 13. var tom = new Cat({"name":"汤姆"}); 14. tom.save(function(){ 15. console.log('喵喵喵'); 16. }); |

上面的代码中，没有一个语句是明显的操作数据库，感觉都在创建类、实例化类、调用类的方法。都在操作对象，但是数据库同步被持久了。

mongoose的哲学，就是让你用操作对象的方式操作数据库。

创建一个模型

|  |
| --- |
| 1. mongoose.model("Cat",{"name" : String , "age" : Integer}); |

就可以被实例化

|  |
| --- |
| 1. var kitty = new Cat({ name: 'Zildjian' }); |

然后这个实例就可以被save。

中文博客： <https://cnodejs.org/topic/51ff720b44e76d216afe34d9>

mongoose首先要想明白一件事儿，所有的操作都不是对数据库进行的。所有的操作都是对类、实例进行的。但是数据库的持久化自动完成了。

## 19.1 数据库连接

公式：

|  |
| --- |
| 1. var mongoose = require('mongoose'); 2. //创建数据库连接 3. var db = mongoose.**createConnection**('mongodb://127.0.0.1:27017/haha'); 4. //监听open事件 5. db.once('open', function (callback) { 6. console.log("数据库成功连接"); 7. }); |

## 19.2 定义模型

**创造schema → 定义一些schema的静态方法 → 创造模型**

创造schema用什么语句？ new mongoose.schema({});

创造模型用什么语句？ db.model(“Student”,schema名字);

|  |
| --- |
| //创建了一个schema结构。  var studentSchema = new mongoose.Schema({  name : {type : String},  age : {type : Number},  sex : {type : String}  });  //创建静态方法  studentSchema.statics.zhaoren = function(name, callback) {  this.model('Student').find({name: name}, callback); //this.model('Student')指的是当前这个类  };  //创建修改的静态方法  studentSchema.statics.xiugai = function(conditions,update,options,callback){  this.model("Student").update(conditions, update, options, callback);  }  //创建了一个模型，就是学生模型，就是学生类。  //类是基于schema创建的。  var studentModel = db.model('Student', studentSchema); |

解释一下，什么是静态方法，什么是类方法：

|  |
| --- |
| 1. //类 2. function Student(){ 4. } 5. //实例化一个学生 6. var xiaoming = new Student(); 7. //实例方法，因为这个sleep方法的执行者是类的实例 8. xiaoming.sleep(); 9. //静态方法（类方法），这个方法的执行者是这个类，不是这个类的实例。 10. Student.findAllBuJiGe(); |

前台界面：不操作数据库，只操作类！

# 二十、web Socket和Socket.IO框架

HTTP无法轻松实现实时应用：

● HTTP协议是无状态的，服务器只会响应来自客户端的请求，但是它与客户端之间不具备持续连接。

● 我们可以非常轻松的捕获浏览器上发生的事件（比如用户点击了盒子），这个事件可以轻松产生与服务器的数据交互（比如Ajax）。但是，反过来却是不可能的：服务器端发生了一个事件，服务器无法将这个事件的信息实时**主动**通知它的客户端。只有在客户端查询服务器的当前状态的时候，所发生事件的信息才会从服务器传递到客户端。

但是，确实聊天室确实存在。

方法：

● 长轮询：客户端每隔很短的时间，都会对服务器发出请求，查看是否有新的消息，只要轮询速度足够快，例如1秒，就能给人造成交互是实时进行的印象。这种做法是无奈之举，实际上对服务器、客户端双方都造成了大量的性能浪费。

● 长连接：客户端只请求一次，但是服务器会将连接保持，不会返回结果（想象一下我们没有写res.end()时，浏览器一直转小菊花）。服务器有了新数据，就将数据发回来，又有了新数据，就将数据发回来，而一直保持挂起状态。这种做法的也造成了大量的性能浪费。

WebSocket协议能够让浏览器和服务器全双工实时通信，互相的，服务器也能主动通知客户端了。

● WebSocket的原理非常的简单：利用HTTP请求产生握手，HTTP头部中含有WebSocket协议的请求，所以握手之后，二者转用TCP协议进行交流（QQ的协议）。现在的浏览器和服务器之间，就是QQ和QQ服务器的关系了。

所以WebSocket协议，需要浏览器支持，更需要服务器支持。

● 支持WebSocket协议的浏览器有：Chrome 4、火狐4、IE10、Safari5

● 支持WebSocket协议的服务器有：Node 0、Apach7.0.2、Nginx1.3

Node.js上需要写一些程序，来处理TCP请求。

● Node.js从诞生之日起，就支持WebSocket协议。不过，从底层一步一步搭建一个Socket服务器很费劲（想象一下Node写一个静态文件服务都那么费劲）。所以，有大神帮我们写了一个库Socket.IO。

● Socket.IO是业界良心，新手福音。它屏蔽了所有底层细节，让顶层调用非常简单。并且还为不支持WebSocket协议的浏览器，提供了长轮询的透明模拟机制。

● Node的单线程、非阻塞I/O、事件驱动机制，使它非常适合Socket服务器。

网址：<http://socket.io/>

先要npm下载这个库

|  |
| --- |
| 1. npm install socket.io |

写原生的JS，搭建一个服务器，server创建好之后，创建一个io对象

|  |
| --- |
| 1. var http = require("http"); 2. var server = http.createServer(function(req,res){ 3. res.end("你好"); 4. }); 5. **var io = require('socket.io')(server);**   **//监听连接事件**  **io.on("connection",function(socket){**  **console.log("1个客户端连接了");**  **})**   1. server.listen(3000,"127.0.0.1"); |

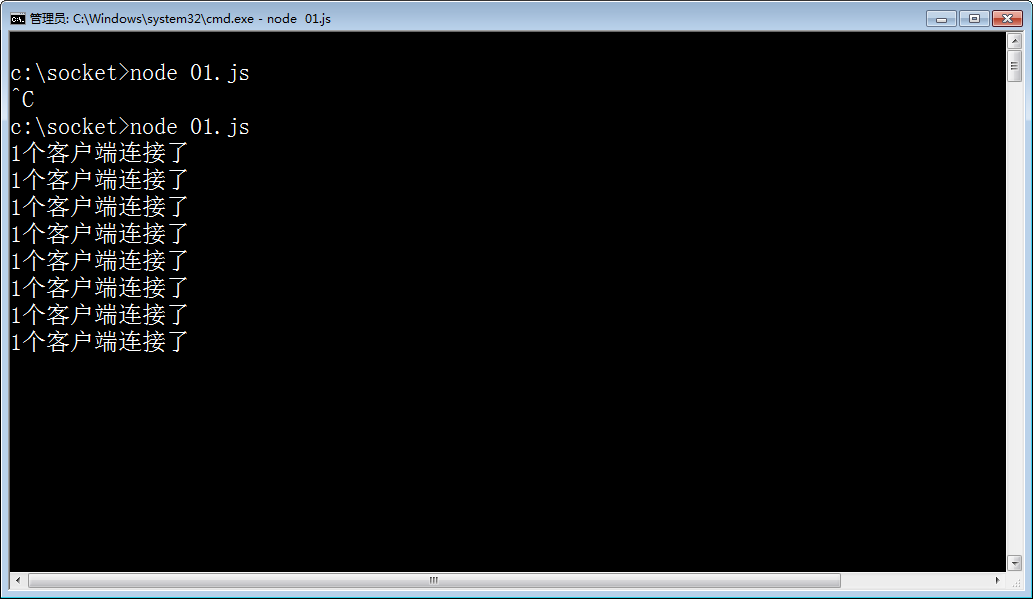
写完这句话之后，你就会发现，http://127.0.0.1:3000/socket.io/socket.io.js 就是一个js文件的地址了。

现在需要制作一个index页面，这个页面中，必须引用秘密js文件。调用io函数，取得socket对象。

|  |
| --- |
| 1. <!DOCTYPE html> 2. <html lang="en"> 3. <head> 4. <meta charset="UTF-8"> 5. <title>Document</title> 6. </head> 7. <body> 8. <h1>我是index页面，我引用了秘密script文件</h1> 9. <script type="text/javascript" src="/socket.io/socket.io.js"></script> 10. <script type="text/javascript"> 11. var socket = io(); 12. </script> 13. </body> 14. </html> |

此时，在服务器上，app.js中就要书写静态文件呈递程序，能够呈递静态页面。

|  |
| --- |
| 1. var server = http.createServer(function(req,res){ 2. if(req.url == "/"){ 3. //显示首页 4. fs.readFile("./index.html",function(err,data){ 5. res.end(data); 6. }); 7. } 8. }); |



至此，服务器和客户端都有socket对象了。服务器的socket对象：

服务器端的：

|  |
| --- |
| 1. var io = require('socket.io')(server); 2. //监听连接事件 3. io.on("connection",function(**socket**){ 4. console.log("1个客户端连接了"); 5. socket.on("tiwen",function(msg){ 6. console.log("本服务器得到了一个提问" + msg); 7. **socket**.emit("huida","吃了"); 8. }); 9. }); |

每一个连接上来的用户，都有一个socket。 由于我们的emit语句，是socket.emit()发出的，所以指的是向这个客户端发出语句。

广播，就是给所有当前连接的用户发送信息：

|  |
| --- |
| 1. //创建一个io对象 2. var io = require('socket.io')(server); 3. //监听连接事件 4. io.on("connection",function(socket){ 5. console.log("1个客户端连接了"); 6. socket.on("tiwen",function(msg){ 7. console.log("本服务器得到了一个提问" + msg); 8. io.emit("huida","吃了"); 9. }); 10. }); |

# 复习

Node自己复习的时候，顺序：

Node.js特点：单线程、异步I/O（非阻塞I/O）、事件驱动（事件环）

适合的程序：就是没有太多的计算，I/O比较多的业务。举例：留言本、考试系统、说说、图片裁切服务器

Node.js原生： http、fs、path、url。 静态服务、简单路由、GET、POST请求。

模块：formidable、gm、express

Express：中间件、MVC建站、模板引擎ejs、静态服务、简单路由、GET、POST请求、MD5加密、图片上传。

服务器的一些概念：Cookie、Session

持久化NoSQL： 非关系型数据库，Not Only SQL。

特点：没有schema，没有行和列。用文档（JSON）来存储。

MongoDB：安装、开启、导入数据、Shell管理数据库、Mongo Vue、Node.js做CRUD（增删改查）、DAO层的封装、索引、操作符$set $lt $gt $push $pull

Mongoose： ODM，不用直接操作数据库，操作对象，这个对象自动持久。

Defining your schema 定义文档结构

schema定义的时候，支持的类型：

The permitted SchemaTypes are

String、Number、Date、Buffer、Boolean、Mixed、ObjectId、Array

转换为对象：

|  |
| --- |
| 1. mongoose.model(modelName, schema): |

定义对象（实例）方法：

|  |
| --- |
| 1. // define a schema 2. var animalSchema = new Schema({ name: String, type: String }); 3. // assign a function to the "methods" object of our animalSchema 4. animalSchema.**methods**.findSimilarTypes = function (cb) { 5. return this.model('Animal').find({ type: this.type }, cb); 6. } |

虚拟属性：

|  |
| --- |
| 1. // define a schema 2. var personSchema = new Schema({ 3. name: { 4. first: String, 5. last: String 6. } 7. }); 8. // compile our model 9. var Person = mongoose.model('Person', personSchema); 10. // create a document 11. var bad = new Person({ 12. name: { first: 'Walter', last: 'White' } 13. }); |