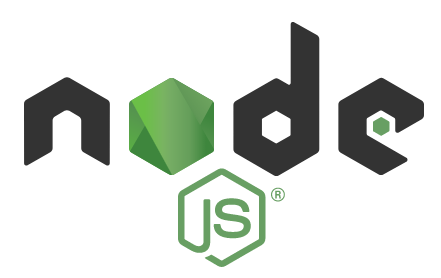
**Node.js进阶学习笔记**



Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。

Node.js 的包管理器 npm，是全球最大的开源库生态系统。

2018年4月8日

目录

[一、Node.js简介 4](#_Toc513129668)

[1. Node的诞生 4](#_Toc513129669)

[2. Node.js特点 4](#_Toc513129670)

[3. 适合什么开发 6](#_Toc513129671)

[4. Node.js无法挑战老牌3P 7](#_Toc513129672)

[二、Node.js安装 7](#_Toc513129673)

[三、网络操作 9](#_Toc513129674)

[1. HTTP模块 9](#_Toc513129675)

[2. URL模块 9](#_Toc513129676)

[3. POST请求 10](#_Toc513129677)

[4. 文件上传模块formidable 10](#_Toc513129678)

[四、文件操作 11](#_Toc513129679)

[五、第一阶段复习 11](#_Toc513129680)

[六、模块 12](#_Toc513129681)

[1. 什么是模块？ 12](#_Toc513129682)

[2. 模块的封装和使用 12](#_Toc513129683)

[2.1. 使用 12](#_Toc513129684)

[2.2. 封装 13](#_Toc513129685)

[3. NPM模块管理 14](#_Toc513129686)

[七、模板引擎 15](#_Toc513129687)

[1. EJS（Embedded JavaScript templates） 15](#_Toc513129688)

[2. Jade模板引擎 15](#_Toc513129689)

[3. SWIG模板引擎 15](#_Toc513129690)

[八、Express框架 16](#_Toc513129691)

[1. Express安装 16](#_Toc513129692)

[2. Express能力 16](#_Toc513129693)

[3. Express路由 17](#_Toc513129694)

[3.1. 路由方法 17](#_Toc513129695)

[3.2. 路由路径 18](#_Toc513129696)

[4. Express中间件 19](#_Toc513129697)

[4.1. app.get()、app.post() 19](#_Toc513129698)

[4.2. app.use() 20](#_Toc513129699)

[4.3. 模板引擎使用 21](#_Toc513129700)

[4.4. Get和Set请求 22](#_Toc513129701)

[九、MongoDB介绍 22](#_Toc513129702)

[5. MongoDB简介 22](#_Toc513129703)

[6. 为什么要用数据库 22](#_Toc513129704)

[7. 关系数据库存在的问题 23](#_Toc513129705)

[8. 非关系数据库（NoSQL） 23](#_Toc513129706)

[十、MongoDB安装 27](#_Toc513129707)

[十一、MongoDB使用 29](#_Toc513129708)

[1. 基本命令 29](#_Toc513129709)

[2. 插入数据 29](#_Toc513129710)

[3. 查找数据 30](#_Toc513129711)

[3.1. 基本查找 30](#_Toc513129712)

[3.2. 查找排序 30](#_Toc513129713)

[4. 修改数据 31](#_Toc513129714)

[5. 删除数据 31](#_Toc513129715)

[6. 分页显示数据 32](#_Toc513129716)

[十二、Cookie和Session 33](#_Toc513129717)

[1. Cookie 33](#_Toc513129718)

[2. Session 34](#_Toc513129719)

[2.1. Session工作原理 34](#_Toc513129720)

[2.2. Session和cookie的区别 34](#_Toc513129721)

[2.3. NodeJS的Session 34](#_Toc513129722)

[十三、加密 35](#_Toc513129723)

[十四、图片处理模块（GM） 35](#_Toc513129724)

[十五、中型案例-说说 36](#_Toc513129725)

# 一、Node.js简介

## Node的诞生

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.

## Node.js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。

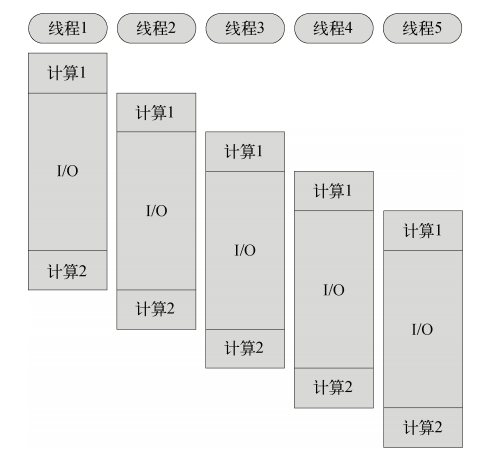
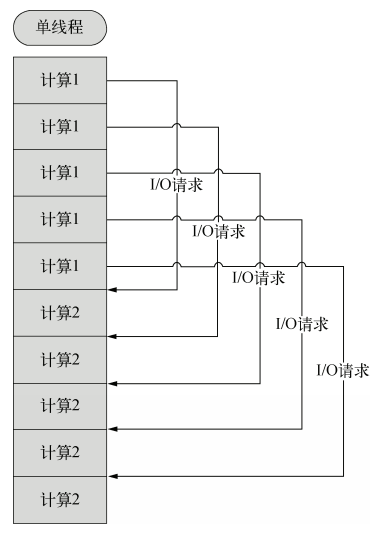
单线程

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。

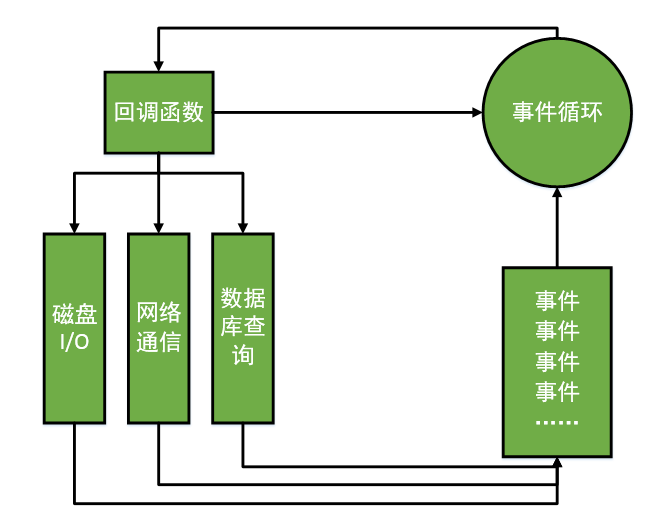
当某个I/O执行完毕时，将以事件的形式通知执行I/O操作的线程，线程执行这个事件的回调函数。为了处理异步I/O，线程必须有事件循环，不断的检查有没有未处理的事件，依次予以处理。

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。

## 适合什么开发

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）

## Node.js无法挑战老牌3P



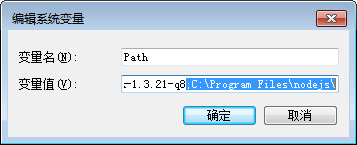
# 二、Node.js安装

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。

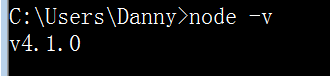
装完之后，我们在系统的环境变量里面，我们看一下：



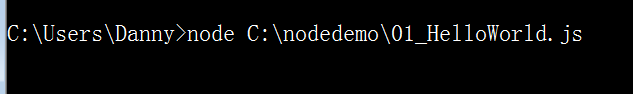
环境变量，就已经自动的填写进去了，就是我们node安装的目录。

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。



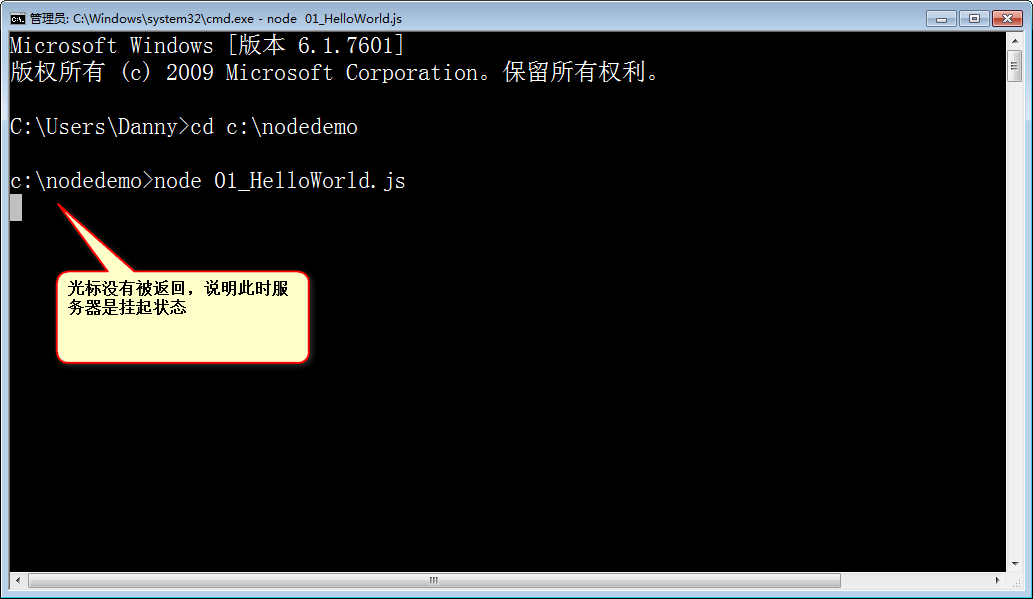
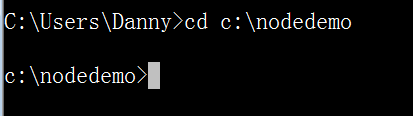
运行文件，就要用node命令来运行：



|  |
| --- |
| 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。

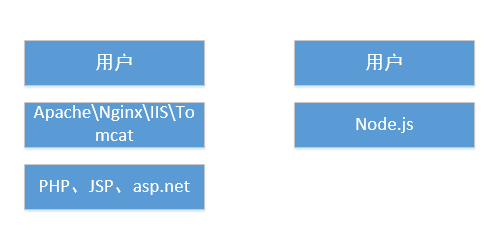
# 三、网络操作

## 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"}); |

## URL模块

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' } |



## 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"> |



## 文件上传模块formidable

# 四、文件操作

具体请看NODE文档，和基础笔记

# 五、第一阶段复习

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：

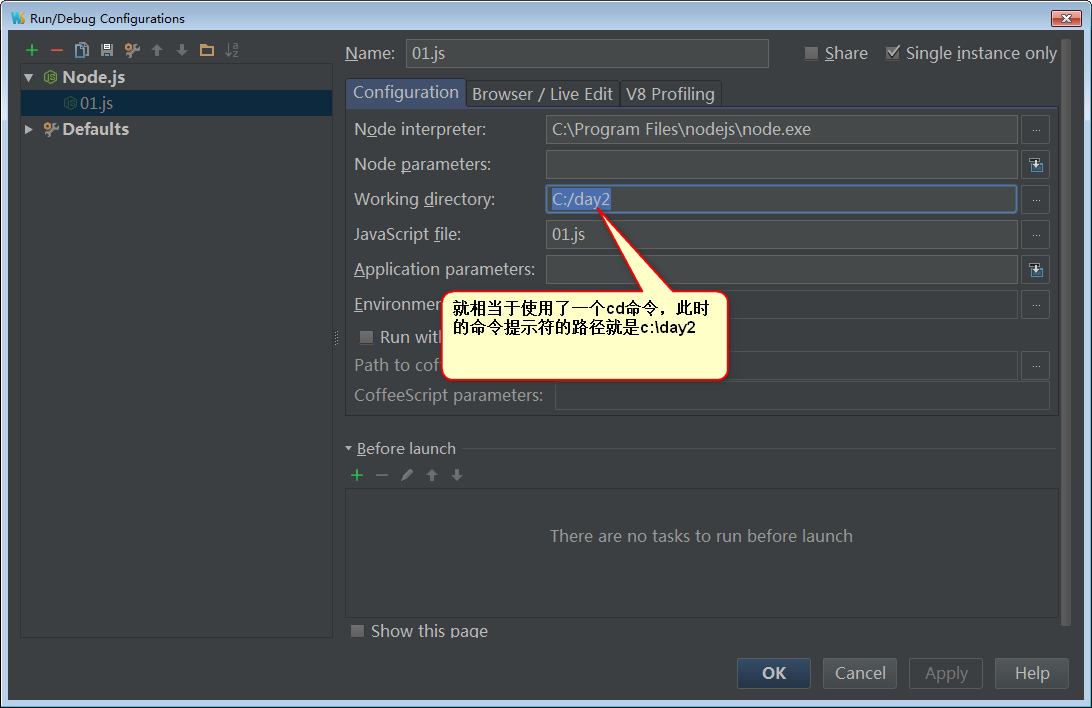
单线程、Non-blocking I/O、Event Driven。 实际上是一个特点。

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。（需要路由）



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"); |

# 六、模块

## 什么是模块？

* 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。

## 模块的封装和使用

### 使用

**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。

例如：

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); //你好 |

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。

**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. }); |

### 封装

* 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。

## NPM模块管理

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 –save //本地安装 2. npm install 模块名字 -g //全局安装 |

就可以安装。 模块名字全球唯一。

本地安装的时候，要注意，命令提示符的当前位置。它会下载到当前目录下的node\_module文件夹下

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>

# 七、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。

## EJS（Embedded JavaScript templates）

"E" 代表 "effective"，即【高效】。EJS 是一套简单的模板语言，帮你利用普通的 JavaScript 代码生成 HTML 页面。EJS 没有如何组织内容的教条；也没有再造一套迭代和控制流语法；有的只是普通的 JavaScript 代码而已。

https://ejs.bootcss.com/

## Jade模板引擎

jade是超高性能的node JavaScript模板引擎，有着非常强大的API和大量杰出的特性。它主要针对node的服务端。

http://jade-lang.com/

## SWIG模板引擎

swig 是node端的一个优秀简洁的模板引擎，类似Python模板引擎Jinja，目前不仅在node端较为通用，相对于jade、ejs优秀，而且在浏览器端也可以很好地运行。

中文说明文档：http://www.iqianduan.net/blog/how\_to\_use\_swig

http://www.swig.org/

# 八、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

● 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/>

## Express安装

安装Express框架，就是使用npm的命令。

|  |
| --- |
| npm install **--save** express |

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

## Express能力

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); |

## Express路由

路由是指如何定义应用的端点（URIs）以及如何响应客户端的请求。

路由是由一个 URI、HTTP 请求（GET、POST等）和若干个句柄组成，它的结构如下： app.METHOD(path, [callback...], callback)， app 是 express 对象的一个实例， METHOD 是一个 HTTP 请求方法， path 是服务器上的路径， callback 是当路由匹配时要执行的函数。

### 路由方法

#### Get方法

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. }); |

#### post方法

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. }); |

#### All方法

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。

### 路由路径

路由路径和请求方法一起定义了请求的端点，它可以是字符串、字符串模式或者正则表达式。

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

## Express中间件

Express 是一个自身功能极简，完全是由路由和中间件构成一个的 web 开发框架：从本质上来说，一个 Express 应用就是在调用各种中间件。

**中间件**（Middleware） 是一个函数，它可以访问请求对象（[request object](http://www.expressjs.com.cn/4x/api.html#req) (req)）, 响应对象（[response object](http://www.expressjs.com.cn/4x/api.html#res) (res)）, 和 web 应用中处于请求-响应循环流程中的中间件，一般被命名为 next 的变量。

中间件的功能包括：

* 执行任何代码。
* 修改请求和响应对象。
* 终结请求-响应循环。
* 调用堆栈中的下一个中间件。

使用可选则挂载路径，可在应用级别或路由级别装载中间件。另外，你还可以同时装在一系列中间件函数，从而在一个挂载点上创建一个子中间件栈。

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。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.get()、app.post()

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. }); |

### app.use()

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。

### 模板引擎使用

本练习中使用EJS作为模板引擎：

1. 引入EJS模板引擎的关键步骤：

//引入所需要的插件

var express = require('express');

var ejs = require('ejs');

//为express设置接受参数

var app = express();

//识别ejs代码

app.set('view engine','ejs');

1. 创建views文件夹存放ejs模板
2. 在js中渲染模板内容：

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');

### Get和Set请求

* GET请求的参数在URL中，在原生Node中，需要使用url模块来识别参数字符串。在Express中，不需要使用url模块了。可以直接使用req.query对象。
* POST请求在express中不能直接获得，必须使用body-parser模块。使用后，将可以用req.body得到参数。但是如果表单中含有文件上传，那么还是需要使用formidable模块。

# 九、MongoDB介绍

## MongoDB简介

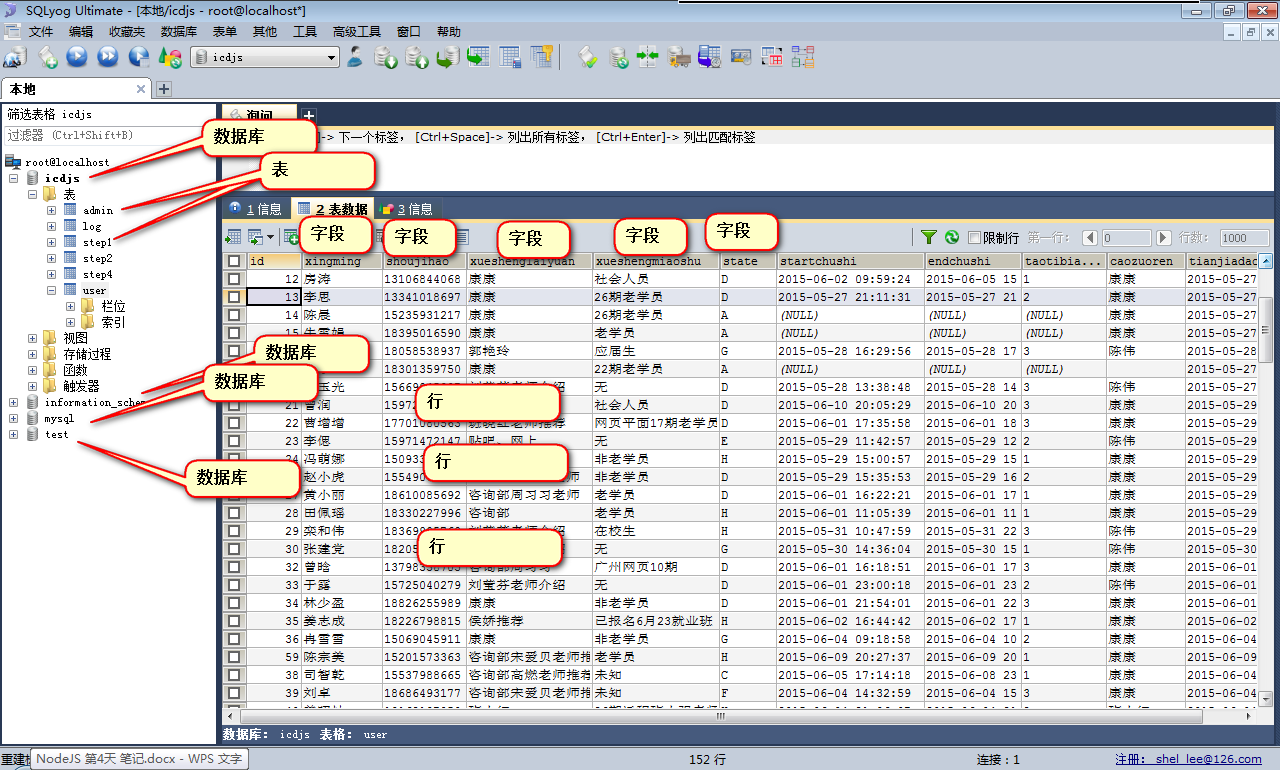
MongoDB（来自于英文单词“Humongous”，中文含义为“庞大”）是可以应用于各种规模的企业、各个行业以及各类应用程序的开源数据库。作为一个适用于敏捷开发的数据库，MongoDB的数据模式可以随着应用程序的发展而灵活地更新。

MongoDB是专为可扩展性，高性能和高可用性而设计的数据库。它可以从单服务器部署扩展到大型、复杂的多数据中心架构。利用内存计算的优势，MongoDB能够提供高性能的数据读写操作。 MongoDB的本地复制和自动故障转移功能使您的应用程序具有企业级的可靠性和操作灵活性。

## 为什么要用数据库

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：

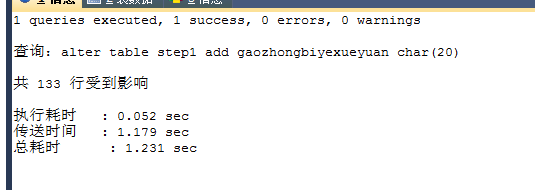
|  |
| --- |
| 1. SELECT \* FROM step1 WHERE xingbie = '女' AND nianling < 24 AND nianling >= 20 AND xianzaisuozaidi = '北京'; |

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。

## 关系数据库存在的问题

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。

* 字段不太灵活
* 数据不灵活

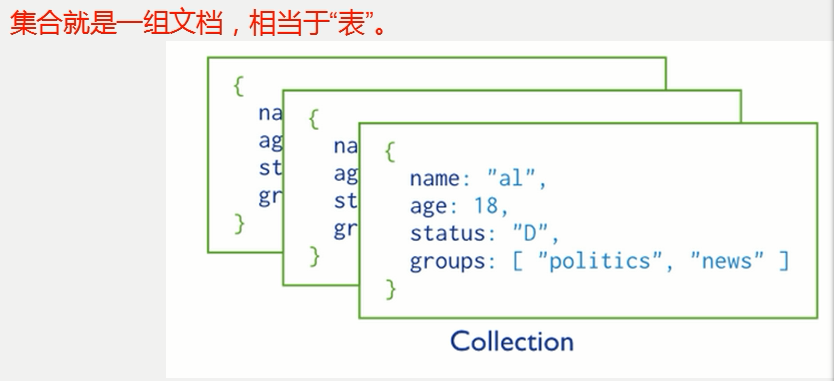
## 非关系数据库（NoSQL）

非关系数据库。**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”。**





NoSQL中，最小的 ‘数据条目’ 不是行 而是文档



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。



**总结：**

NoSQL数据库在以下的这几种情况下比较适用：

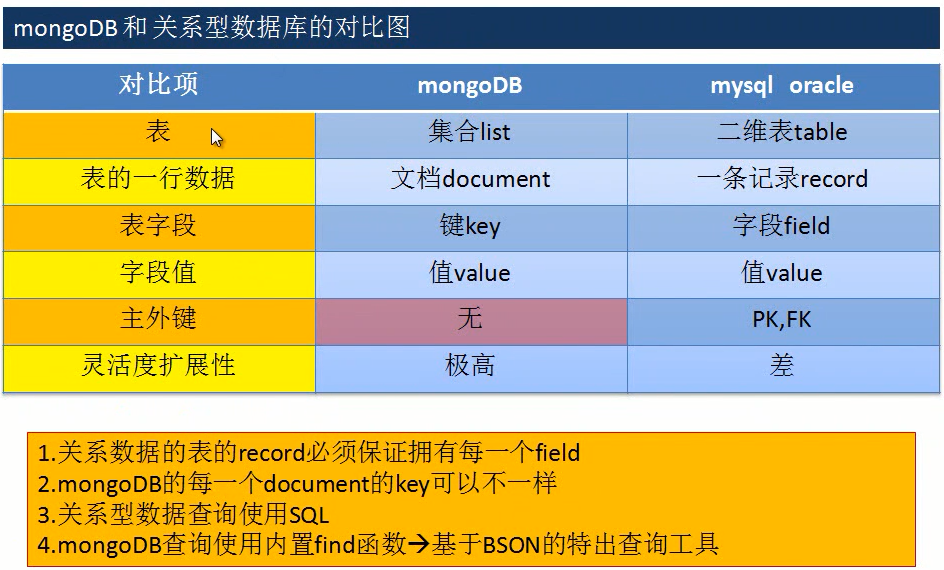
1. **、数据模型比较简单；**
2. **、需要灵活性更强的IT系统；**
3. **、对数据库性能要求较高；**
4. **、不需要高度的数据一致性；**

**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。

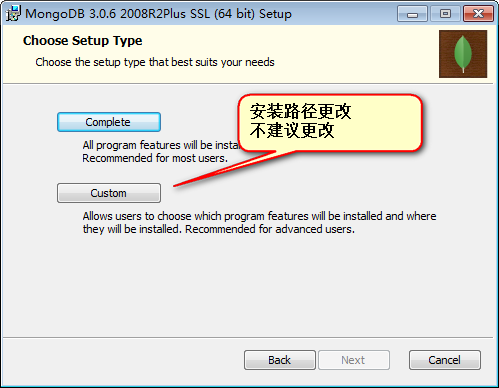


# 十、MongoDB安装

官网：<https://www.mongodb.com/>

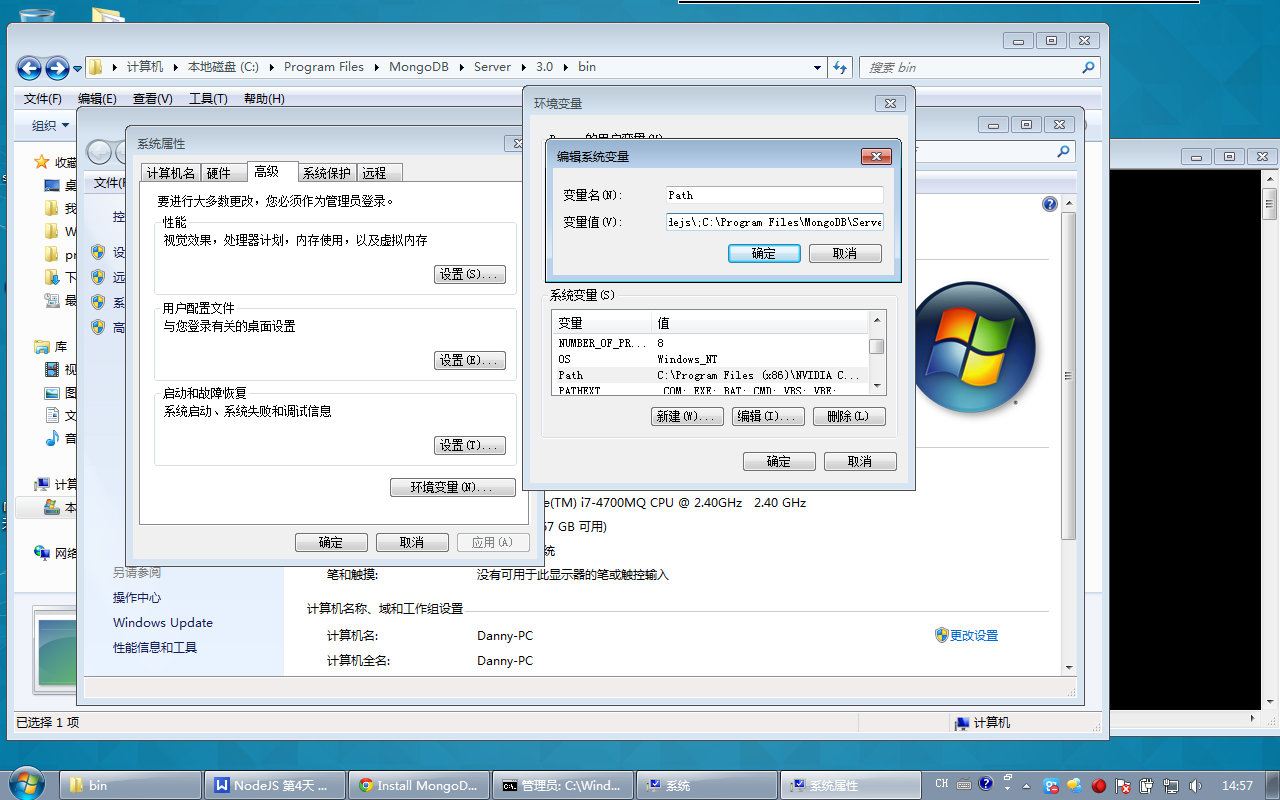
手册：<https://docs.mongodb.org/manual/>

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。

注意：如果安装安装报错，请把compass这个选项去掉！

# 十一、MongoDB使用

## 基本命令

* 查看所有数据库列表

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

* 使用数据库、创建数据库

|  |
| --- |
| 1. **use dbname** |

注意：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”}); |

* 删除数据库，删除当前所在的数据库

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

## 插入数据

数据库中不能直接插入数据，只能往集合（collections）里面插入数据

* 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"});** 2. 格式：db.collectionName.insert({“key ” : “value” , [,key1: value1]}) |

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：

* 批量插入数据

|  |
| --- |
| 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。**

## 查找数据

### 基本查找

* 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}]}); |

**pretty()格式化输出**

### 查找排序

在MongoDB中使用使用sort()方法对数据进行排序，sort()方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而-1是用于降序排列。

语法：

find().sort({KEY:1})

* 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 } **)** |

## 修改数据

MongoDB数据更新可以使用update()函数。

**db.collection.update( criteria, objNew, upsert, multi )**

update()函数接受以下四个参数：

* + criteria : update的查询条件，类似sql update查询内where后面的。
  + objNew : update的对象和一些更新的操作符（如$,$inc...）等，也可以理解为sql update查询内set后面的
  + upsert : 这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
  + multi : mongodb默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。

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}); |

## 删除数据

MongoDB remove()函数是用来移除集合中的数据。

MongoDB数据更新可以使用update()函数。在执行remove()函数前先执行find()命令来判断执行的条件是否正确，这是一个比较好的习惯。

remove() 方法的基本语法格式如下所示：

db.collection.remove(

<query>,

<justOne>

<writeConcern>

)

参数说明：

* query :（可选）删除的文档的条件。
* justOne : （可选）如果设为 true 或 1，则只删除一个文档。
* writeConcern :（可选）抛出异常的级别。

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 }** ) |

## 分页显示数据

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；

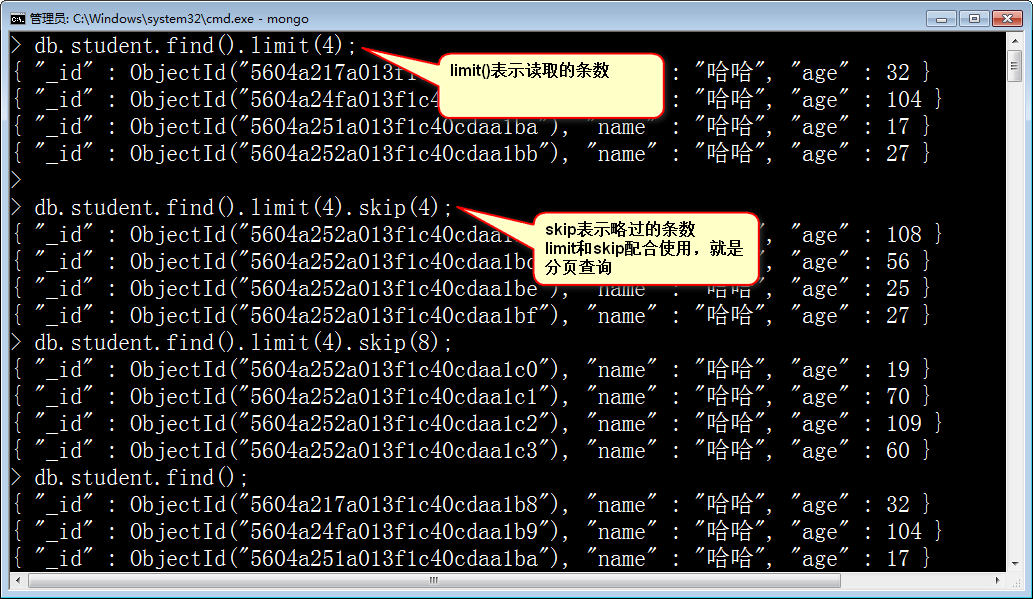
2） 正确的做法： 就是真的在数据库中，只读取这么多内容。

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。

## 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：

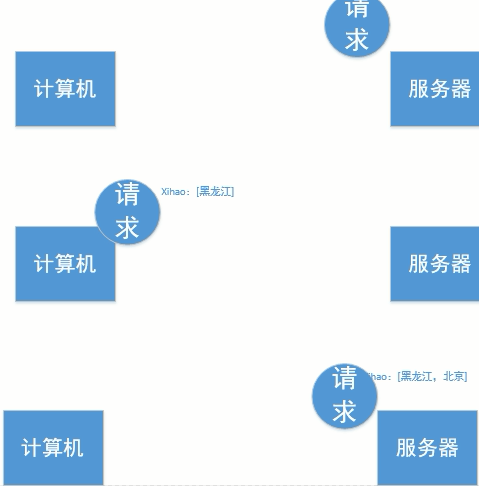
|  |
| --- |
| 1. db.members.createIndex( { "user\_id": 1 }, **{ unique: true }** ); |

# 十二、Cookie和Session

## 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。



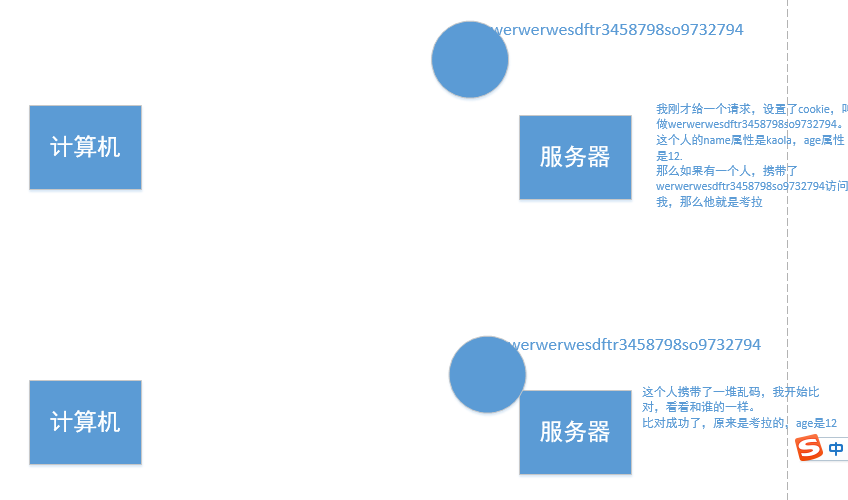
## Session

### Session工作原理

Session（会话），专门用于实现"登陆"，表示某一次用户的访问。

•会话是特殊的cookie。用户在登陆之后，服务器往客户端的cookie中发送一个字符串，但是此字符串是经过服务器加密的，服务器会缓存cookie文本和用户的信息；再次访问之后，如果这个加密字符串与服务器的缓存匹配，那么则判断这是同一个用户回来了，那么将保持登陆状态。

Session（会话）。 Session不是一个天生就有的技术，而是依赖cookie而存在的。



可知，session就是cookie 的一个变量

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。

### Session和cookie的区别

* Session下发的是乱码并且服务器自己存储对应信息，其对应数据量可以是无限大
* Cookie下发的是明码，不安全，大小有限

# 十三、加密

复习：

**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'); |

# 十四、图片处理模块（GM）

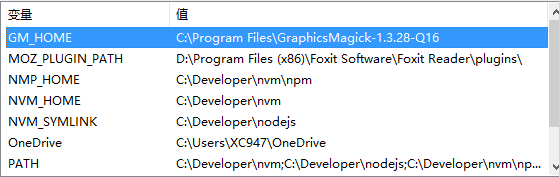
<http://www.graphicsmagick.org/>

GraphicsMagick号称图像处理领域的瑞士军刀。 短小精悍的代码却提供了一个鲁棒、高效的工具和库集合，来处理图像的读取、写入和操作，支持超过88种图像格式，包括重要的DPX、GIF、JPEG、JPEG-2000、PNG、PDF、PNM和TIFF。

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。

配置环境变量才能使用命令



* **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-gm模块：**

常用操作如下：

* node.js缩略图的制作：

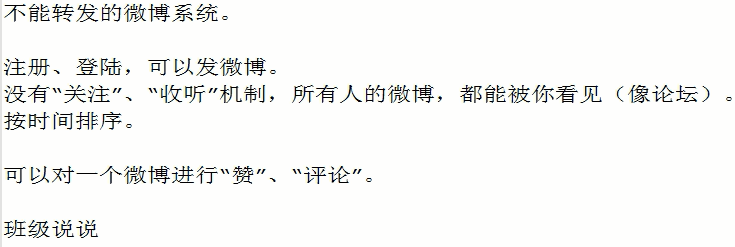
|  |
| --- |
| 1. var fs = require('fs'); 2. var gm = require('gm'); 3. gm('./danny.jpg') 4. .resize(50, 50,"!") 5. .write('./danny2.jpg', function (err) { 6. if (err) { 7. console.log(err); 8. } 9. }); |

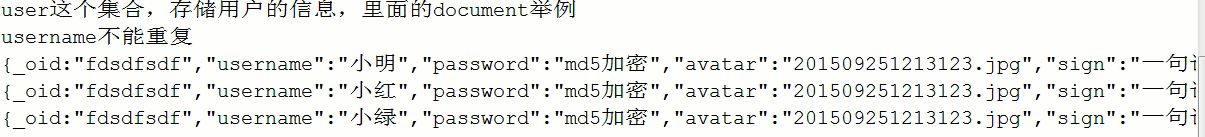
* node.js头像裁切：

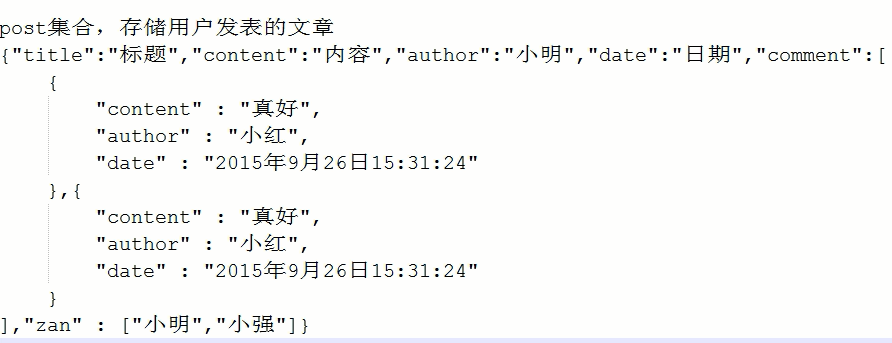
|  |
| --- |
| 1. gm("./danny.jpg")**.crop(141,96,152,181)**.write("./2.jpg",function(err){ 2. **//141 96 是宽高 。 152 181是坐标** 3. }); |

# 十五、中型案例-说说

需求说明：







# 十六、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。

## 名词解释：

* Schema： 一种以文件形式存储的数据库模型骨架，不具备数据库的操作能力
* Model： 由Schema编译而成的假想（fancy）构造器，具有抽象属性和行为。Model的每一个实例（instance）就是一个document。document可以保存到数据库和从数据库返回。
* Instance： 由Model创建的实例。

## Mongoose第一个应用

官网上的hello world：



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。

## 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. }); |

## 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); |

## 静态方法、类方法

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！

# 十七、Mongoose详细版

MongoDB是一个开源的NoSQL数据库，相比MySQL那样的关系型数据库，它更显得轻巧、灵活，非常适合在数据规模很大、事务性不强的场合下使用。同时它也是一个对象数据库，没有表、行等概念，也没有固定的模式和结构，所有的数据以文档的形式存储(文档，就是一个关联数组式的对象，它的内部由属性组成，一个属性对应的值可能是一个数、字符串、日期、数组，甚至是一个嵌套的文档。)，数据格式就是JSON。

Mongoose是MongoDB的一个对象模型工具，是基于node-mongodb-native开发的MongoDB nodejs驱动，可以在异步的环境下执行。同时它也是针对MongoDB操作的一个对象模型库，封装了MongoDB对文档的的一些增删改查等常用方法，让NodeJS操作Mongodb数据库变得更加灵活简单。

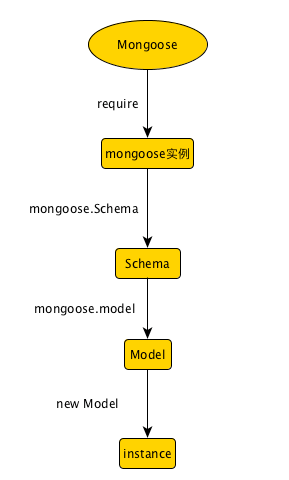
**传统数据库与mongodb和mongoose的差异比较：**

| **Oracle** | **MongoDB** | **Mongoose** |
| --- | --- | --- |
| 数据库实例(database instance) | MongoDB实例 | Mongoose |
| 模式(schema) | 数据库(database) | mongoose |
| 表(table) | 集合(collection) | 模板(Schema)+模型(Model) |
| 行(row) | 文档(document) | 实例(instance) |
| rowid | \_id | \_id |
| Join | DBRef | DBRef |

## 基本概念

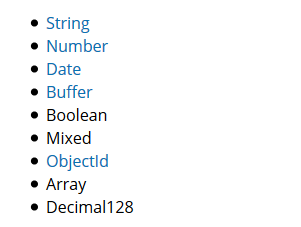
* **Schema**: 相当于一个数据库的模板. Model可以通过mongoose.model 集成其基本属性内容. 当然也可以选择不继承.
* **Model**: 基本文档数据的父类,通过集成Schema定义的基本方法和属性得到相关的内容.
* **instance**: 这就是实实在在的数据了. 通过 new Model()初始化得到.

其关系如下：



## 创建Schema（结构）

**Scheme支持的数据类型**



**创建Scheme，如下：**

var userSchema = new mongoose.Schema({

name: String,

email: String,

createdOn: Date

});

## 创建Model（模型）

* Mongoose.Model('User', userSchema);
* adminConnection.model( 'User', userSchema );

参数一为Model的名字，参数二为生成Model所需要的schema,Model就像是schema所编译而成的一样。

## Mongoose连接数据库方式

* **方式一**：

var dbURI = 'mongodb://localhost/mydatabase';

mongoose.connect(dbURI);

* **方式二**：

var dbURI = 'mongodb://localhost/myadmindatabase';

var adminConnection = mongoose.createConnection(dbURI);

如果需要声明端口号：

var dbURI = 'mongodb://localhost:27018/mydatabase';

如果需要定义用户名和密码：

var dbURI = 'mongodb://username:password@localhost/mydatabase';

可以这样传一个对象类型的参数：

var dbURI = 'mongodb://localhost/mydatabase';

var dbOptions = {'user':'db\_username','pass':'db\_password'};

mongoose.connect(dbURI, dbOptions);

## Model实例化

使用new来实例化一个实例。

var user = new User({ name: 'Simon' });

user就是模型User的一个实例。

## 自定义collections的名字

* 方式一，在创建schema的时候定义collection的名字：

var userSchema = new mongoose.Schema({

name: String,

email: {type: String, unique:true}

},

{

collection: 'myuserlist'

});

* 方式二，在创建Model的时候定义collection的名字：

mongoose.model( 'User', userSchema, 'myuserlist' );

## Model方法和实例方法

区别就是一个给Model添加方法（statics），

一个给实例添加方法（methods）。

## Mongoose的CURD操作

### 添加、保存操作

user.save(function (err) {

if (err) return handleError(err);

});

### 查找操作

当然也有另外一种方式能够直接查找数据库的，就是直接在查找方法中添加回调函数，使用方式为：

Model.find(conditions, [fields], [options], [callback])

例如：User.find({'name', 'simon holmes'}, function(err, user) {});

[options]中可以添加各种条件

### 更新操作

有三种方式来更新数据：

* update(conditions,update,options,callback);

该方法会匹配到所查找的内容进行更新，不会返回数据；

* findOneAndUpdate(conditions,update,options,callback);

该方法会根据查找去更新数据库，另外也会返回查找到的并未改变的数据；

* findByIdAndUpdate(conditions,update,options,callback);

该方法跟上面的findOneAndUpdate方法功能一样，不过他是根据ID来查找文档并更新的。

**三个方法都包含四个参数，参数的意思：**

conditions:查询条件

update:更新的数据对象，是一个包含键值对的对象

options:是一个声明操作类型的选项，这个参数在下面再详细介绍

callback:回调函数

对于options参数，在update方法中和findOneAndUpdate、findByIdAndUpdate两个方法中的可选设置是不同的；

* 在update方法中，options的可选设置为：

{

* safe:true|false, //声明是否返回错误信息，默认true
* upsert:false|true, //声明如果查询不到需要更新的数据项，是否需要新插入一条记录，默认false
* multi:false|true, //声明是否可以同时更新多条记录，默认false
* strict:true|false //声明更新的数据中是否可以包含在schema定义之外的字段数据，默认true

}

* 对于findOneAndUpdate、findByIdAndUpdate这两个方法，他们的options可选设置项为：

{

* new:true|false, //声明返回的数据时更新后的该是更新前的，如果为true则返回更新后的，默认true
* upsert:false|trure,
* sort:javascriptObject, //如果查询返回多个文档记录，则可以进行排序，在这里是根据传入的javascript object对象进行排序
* select:String //这里声明要返回的字段，值是一个字符串

}

### 删除操作

* remove();

remove方法有两种使用方式，一种是用在模型上，另一种是用在模型实例上

* 模型上是删除对应的数据，
* 模型实例是删除单个数据
* findOneAndRemove();
* findByIdAndRemove();

# 十八、Web Socket和Socket.IO框架