

# Node.js入门和企业级项目开发

第3天课堂笔记

班级：全栈前端训练营1120班

讲师：邵山欢

日期：2017年3月26日

邵山欢老师

微博：@邵山欢

QQ : 179427026

E-mail : ssh@iqianduan.cn

爱前端官网 ：[www.iqianduan.cn](http://www.iqianduan.cn)

# 目录

[Node.js入门和企业级项目开发 1](#_Toc26030)

[目录 2](#_Toc11058)

[一、复习 3](#_Toc23439)

[1.1 call和apply 3](#_Toc9897)

[1.2 node.js的串讲复习 7](#_Toc21501)

[二、MVC分层开发 10](#_Toc30344)

[2.1 基本原理 10](#_Toc19458)

[2.2 点餐系统MVC开发 11](#_Toc7735)

[2.3 因数计算器系统MVC开发 15](#_Toc20959)

[三、MongoDB芒果数据库 17](#_Toc4481)

[3.1 NoSQL简介 17](#_Toc30334)

[3.2 MongoDB的安装 18](#_Toc201)

[3.3 数据库的基本操作 19](#_Toc1874)

[四、NodeJS和MongoDB的连接 22](#_Toc20460)

[4.1 基本连接 22](#_Toc15403)

[4.2 集合和文档的概念 23](#_Toc21819)

[4.3 增 24](#_Toc8402)

[4.4 删 26](#_Toc10541)

[4.5 查 26](#_Toc105)

[4.6 改 28](#_Toc21692)

# 一、复习

## 1.1 call和apply

call和apply是用来执行上下文运行函数的。

我们写了一个函数，比如

function fun(){

}

此时如果运行它，可以直接加圆括号运算符：

fun()

此时函数的上下文就是window对象。所谓的上下文就是函数中出现的this是谁。

有很多时候，我们需要让函数指定上下文运行，此时就要使用call或者apply，它们两个功能完全相同。

比如：

|  |
| --- |
| //这是一个函数，功能是让上下文对象的a属性变为100  //这个函数到底给谁的a属性变为了100，此时不知道，要看函数调用的时候指定的上下文是谁。  **function** fun**(){**  **this.**a **=** 100**;**  **}**  //小明对象  **var** xiaoming **=** **{**  a **:** 8**,**  b **:** 9  **}**  //小红对象  **var** xiaohong **=** **{**  a **:** 10**,**  b **:** 3  **}**  //运行fun函数，同时执行xiaoming对象是fun函数的上下文  fun**.**call**(**xiaoming**);**  console**.**log**(**xiaoming**);**  console**.**log**(**xiaohong**);** |



我们把

fun.call(xiaoming);

改为

fun.apply(xiaoming);

此时我们可以认为call和apply功能是一样的。

区别在哪儿呢？在于一个函数如果需要传递参数，此时又需要指定上下文，call和apply就有明显区别的：

|  |
| --- |
| //这是一个函数，功能是让上下文对象的a属性变为传入的两个参数的和  //这个函数到底给谁的a属性变了，此时不知道，要看函数调用的时候指定的上下文是谁。  **function** fun**(**m**,**n**){**  **this.**a **=** m **+** n**;**  **}**  //小明对象  **var** xiaoming **=** **{**  a **:** 8**,**  b **:** 9  **}**  //小红对象  **var** xiaohong **=** **{**  a **:** 10**,**  b **:** 3  **}**  fun**.**call**(**xiaoming**,**2**,**3**);**  //fun.apply(xiaoming,[2,3]);  console**.**log**(**xiaoming**);**  console**.**log**(**xiaohong**);** |



我们发现，call()的时候需要用逗号一个一个罗列要传入的参数。apply需要传入数组。

也就是说：

|  |
| --- |
| fun.call(xiaoming,2,3); |

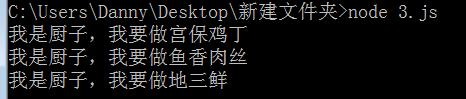
等价于：

|  |
| --- |
| fun.apply(xiaoming,[2,3]); |

此时call和apply你感觉就是语法的区别，但实际上在用的时候特别灵活。

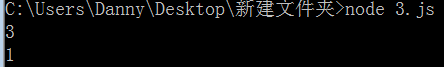
比如，我们现在有一个厨子函数，可以将用户传入的不定量参数都输出出来：

|  |
| --- |
| **function** chushi**(){**  **for(var** i **=** 0 **;** i **<** arguments**.**length **;** i**++){**  console**.**log**(**"我是厨子，我要做" **+** arguments**[**i**]);**  **}**  **}**  chushi**(**"宫保鸡丁"**,**"鱼香肉丝"**,**"地三鲜"**);** |



此时我们引入一个新的服务员函数，服务员的函数的目的就是原封不动的将自己的参数传给厨子函数。此时你觉得直接fuwuyuan(arguments)就行么？此时不对，来看一个准备知识：

|  |
| --- |
| **function** chushi**(){**  console**.**log**(**arguments**.**length**); //这里输出几？1。因为服务员调用chushi的时候是将服务员自己的所有参数当做一个整体传入了chushi。**  **}**  //服务员函数，这个函数的功能就是原封不动的把自己的参数传入厨师  **function** fuwuyuan**(){**  console**.**log**(**arguments**.**length**); //这里输出几？3**  chushi**(**arguments**);**  **}**  fuwuyuan**(**"宫保鸡丁"**,**"鱼香肉丝"**,**"地三鲜"**);** |

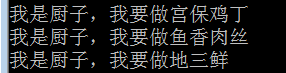


所以在服务员函数里面，应该这样调用chuzi函数：

|  |
| --- |
| chuzi(null,arguments) |

完整的代码：

|  |
| --- |
| **function** chushi**(){**  **for(var** i **=** 0 **;** i **<** arguments**.**length **;** i**++){**  console**.**log**(**"我是厨子，我要做" **+** arguments**[**i**]);**  **}**  **}**  //服务员函数，这个函数的功能就是原封不动的把自己的参数传入厨师  //是一个代理函数，proxy，也可以叫做委托函数，delegate  **function** fuwuyuan**(){**  chushi**.**apply**(null,**arguments**); //因为chuzi里面根本就没有出现this这个词儿，所以就不需要指定上下文。设置为null就可以了。**  **}**  fuwuyuan**(**"宫保鸡丁"**,**"鱼香肉丝"**,**"地三鲜"**);** |



实际应用，比如sum和average两个函数的关系，就是chuzi和fuwuyuan的关系：

//计算实参列表所有数字的和

function sum(){

for (var i = 0 , \_sum = 0; i < arguments.length; i++) {

\_sum += arguments[i];

}

return \_sum;

}

//计算实参列表所有数字的平均数，一定要先计算和

function average(){

**var \_sum = sum.apply(null,arguments);**

return \_sum / arguments.length;

}

console.log(average(4,4,4,5,5,5));

常用的寻找数组中最大值，可以借助Math对象的max方法，但是一定注意是散着传入的，用逗号直接隔开传入。

console.log(Math.max(4,5,6,7,22,23,4,2,1));

此时我们要寻找这个数组中的最大值：

var arr = [23,4,45,3,23,42,34,4];

console.log( **Math.max.apply(null,arr)** );

此时我们要把数组“打散”传入Math.max函数，此时就要借助apply方法！因为apply是用来打散数组的。

## 1.2 node.js的串讲复习

学习任何知识前两天特别重要，你必须准确把握这个东西是什么。React、Redux、Flux、webpack、jsx等等。

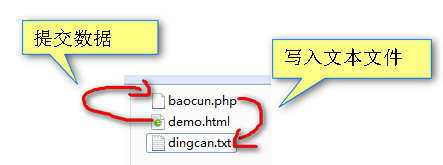
node.js是用来做后台的，是用来替代PHP的！因为PHP太吃硬件，所以现在很多公司把业务迁移到了nodejs上。

后台和前台是到底是干嘛的？用PHP来进行演示。

前端开发工程师最常用的语言就是HTML+CSS，实现一个界面，俗称切图：



但是界面没有功能的，点击提交按钮之后，你的数据是没有人知道的。所以我们就要书写PHP程序



baocun.php

<?php

//接受用户的请求数据

$shoujihao = $\_POST["shoujihao"];

$cai = $\_POST["cai"];

//打开要写入的文件，a表示在原有内容下面追加

$myfile = fopen("dingcan.txt", "a");

//要写入的文字

$txt = "【手机号】{$shoujihao}\r\n【点的菜】{$cai}\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n";

//写入，往$myfile文件中写入$txt文本

fwrite($myfile, $txt);

//关闭文件

fclose($myfile);

?>

前端要负责精准的提交Ajax请求，把数据带给后台PHP程序，在工作中，放心，后台一定不会让你写的。除非是Nodejs。

<script type="text/javascript">

$("#btn").click(function(){

var shoujihao = $("input[name=shoujihao]").val();

var cai = $("input[name=cai]").val();

$.post("baocun.php",{

shoujihao : shoujihao ,

cai : cai

},function(data){

if(data == 1){

alert("成功");

}else{

alert("失败");

}

})

});

</script>

NodeJS的目的和PHP是一样的，都是负责持久化数据的！前端的哲学是美，后端的哲学是数据。

点餐系统是用PHP写的，如果并发大了，立刻崩溃。所以NodeJS诞生了，解决了大数据时代下高并发的问题。

**NodeJS没有阿帕奇，它连一个最最基本的路由都需要些程序。比如我们访问demo.html，就是想看demo.html物理页面。但是NodeJS需要用fs.readFile()一下，然后res.end()呈递一下。**

太慢！！！

所以Express应运而生！

工作中没有公司用原生Node，一定都是Express，所以我们直接来用Express！

$ npm init

$ cnpm install express --save

此时express可以非常方便的跑一个服务器：

var express = require("express");

var app = express();

app.get("/",function(req,res){

res.send("adfadsf ");

});

app.listen(3000);

运行：

$ node app.js

我们改变demo.html文件，让Ajax提交到/baocun去，改变app.js：

var express = require("express");

var app = express();

var formidable = require("formidable");

var fs = require("fs");

////当有人用get请求访问/的时候做的事情

app.get("/",function(req,res){

res.send("首页");

});

//当有人用post请求访问/baocun的时候做的事情

app.post("/baocun",function(req,res){

var form = new formidable.IncomingForm();

form.parse(req, function(err, fields) {

console.log(fields);

//写入文件,writeFile()表示覆盖文件，appendFile表示追加文件。

fs.appendFile("./public/dingcan.txt",JSON.stringify(fields),function(err){

if(err){

res.send("-1");

}else{

res.send("1");

}

});

});

});

//将public文件夹静态化出来，此时里面的文件将自动拥有路由

app.use(express.static("public"));

app.listen(3000);

# 二、MVC分层开发

## 2.1 基本原理

MVC全名是el VModiew Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种将业务逻辑、数据、界面显示分离的组织代码的方式。方便了我们的产品迭代、测试，比如我们要改业务逻辑后者改变UI界面，我们将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式：

Model（模型）表示应用程序核心（比如数据库记录列表）。

View（视图）显示数据（显示数据库记录）。

Controller（控制器）处理输入（写入数据库记录）。

**Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。**

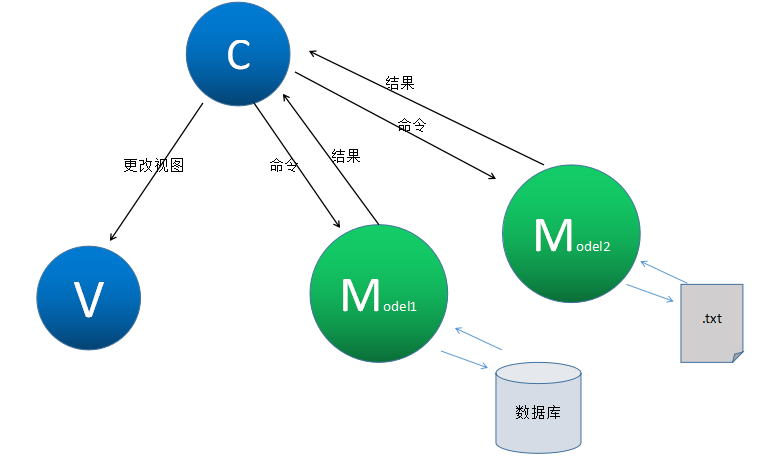
　　通常模型对象负责在数据库中存取数据。最底层、最原子、最脏、最累的活儿要交给Model层完成。

**View（视图）是应用程序中处理数据显示的部分。**

　　通常视图是依据控制器汇总的模型数据创建的。模型数据由控制器来汇总。

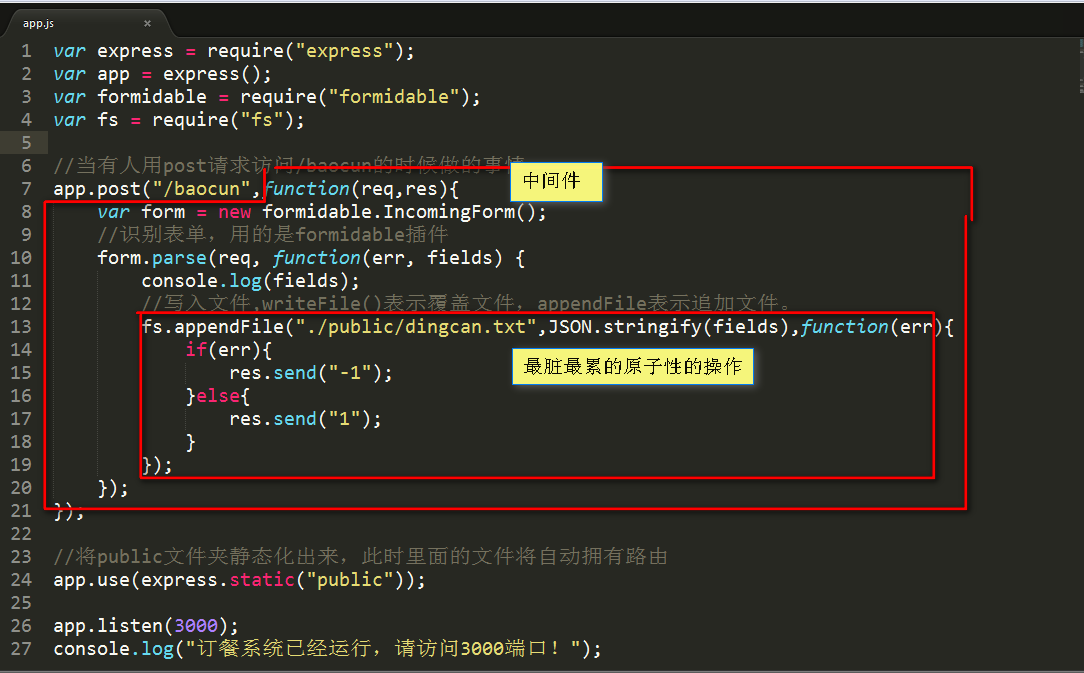
**Controller（控制器）是应用程序中处理用户交互的部分。**

　　通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。模型发回的数据，控制器要汇总、分析交由视图呈递。



## 2.2 点餐系统MVC开发

我们用今天复习时候写的一个订餐系统来举例，app.js涵盖了所有的逻辑：



分锅！分而治之。

所以我们将中间件，挪到controller中，我们创建一个controllers文件夹，写一个mainctrl.js文件：

var formidable = require("formidable");

var fs = require("fs");

//向外暴露一个中间件

**exports.baocun = function(req,res){**

var form = new formidable.IncomingForm();

//识别表单，用的是formidable插件

form.parse(req, function(err, fields) {

console.log(fields);

//写入文件,writeFile()表示覆盖文件，appendFile表示追加文件。

fs.appendFile("./public/dingcan.txt",JSON.stringify(fields),function(err){

if(err){

res.send("-1");

}else{

res.send("1");

}

});

});

**}**

此时代码管理增强了，感觉app.js没有那么重了：

var express = require("express");

var app = express();

**var mainctrl = require("./controllers/mainctrl.js");**

//当有人用post请求访问/baocun的时候做的事情

**app.post("/baocun",mainctrl.baocun);**

//将public文件夹静态化出来，此时里面的文件将自动拥有路由

app.use(express.static("public"));

app.listen(3000);

console.log("订餐系统已经运行，请访问3000端口！");

我们可以将模型层创建出来，负责最基本的底层业务，通常是文件操作和数据库读取。我们下午学习MongoDB，明天上午9:00上课，一上来学习Mongoose框架，Mongoose框架直接就帮你创建了一个个模型。

项目结构：

|  |
| --- |
| ┝ app.js  ┝ public  ┝ ┝ index.html  ┝ models  │ ┝ file.js  ┝ controllers  │ ┝ mainctrl.js |

models/file.js文件：

var fileurl = "./public/dingcan.txt";

var fs = require("fs");

exports.save = function(data,callback){

//追加写入这个文件

fs.appendFile(fileurl,data,callback);

}

更改之后的controllers/mainctrl.js文件：

var formidable = require("formidable");

var file = require("../models/file.js");

//向外暴露一个中间件，这个是Ajax接口

exports.baocun = function(req,res){

var form = new formidable.IncomingForm();

//识别表单，用的是formidable插件

form.parse(req, function(err, fields) {

file.save(JSON.stringify(fields),function(err){

if(err){

res.send("-1");

}else{

res.send("1");

}

});

});

}

我们现在要增加一个业务，我们现在想把所有定单的人的电话，单独保存，我们修改了file.js：

var fs = require("fs");

var baseurl = "./data";

exports.save = function(shoujihao,data,callback){

//写入这个文件

fs.writeFile(baseurl + "/" + shoujihao + ".txt",data,callback)

}

此时要再次给file.js增加一个函数，能够罗列所有的文件名。

var fs = require("fs");

var baseurl = "./data";

//保存

exports.save = function(shoujihao,data,callback){

//写入这个文件

fs.writeFile(baseurl + "/" + shoujihao + ".txt",data,callback)

}

//这个函数的功能就是从文件夹中获得所有文件名

var getAllFilesName = exports.getAllFilesName = function(){

fs.readdir(baseurl,function(err,filenameArray){

console.log(filenameArray);

});

}

getAllFilesName();

红色部分就是我们创建的一个函数，它好用么？我们可以对这个model进行单元测试。

单元测试就是单独测试某一个js文件好用不好用，排除其他文件的影响。

$ node models/files.js

至此我们学习过的fs模块的常用API如下：

fs.writeFile() 写文件

fs.appendFile() 追加文件

fs.readFile() 读文件

fs.readdir() 读取文件夹中的所有文件名

fs.rename() 改名

调用这个函数：

//暴露中间件

exports.showAlldingcan = function(req,res){

//命令file模型去读取文件夹中的所有手机号，呈递视图

//调用file模型的函数，让模型去做最脏最累的活

file.getAllFilesName(function(arr){

//arr就是文件夹中所有文件夹名

res.render("alldingdan.ejs",{

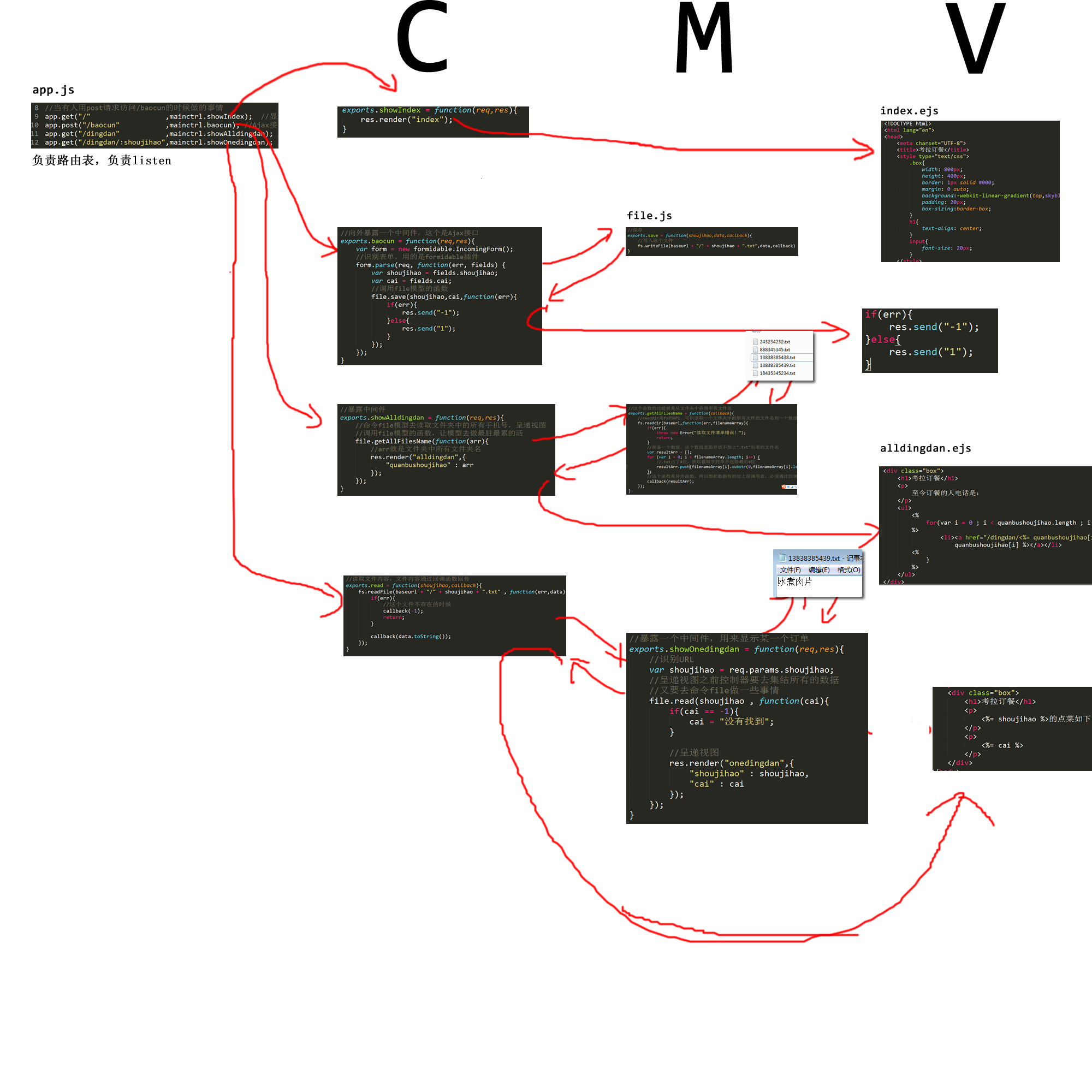
"quanbushoujihao" : arr

});

});

}

整个项目的MVC分层：



## 2.3 因数计算器系统MVC开发

让用户查询一个数字的因数，NodeJS不擅长计算，容易堵塞进程。所以策略是，算一个保存一个，当用户查询的时候，我们先去寻找之前有没有别人计算过这个数字，如果有，就直接写答案。

我们研究一下模块的哲学到底是什么？模块看不见全局，只能看见自己的业务（井底之蛙）。

每个模块不需要了解自己的功能如何贡献给项目，但是只需要知道让自己鲁棒就行了。

我们感觉需要两个模块：

文件file模块，可以写文件，读取文件

数学math模块，可以计算因数

看代码即可，笔记不写东西了。

项目结构：

C:.

│ app.js

│ package.json

│

├─controllers

│ mainctrl.js

│

├─data

│ 123.txt

│ 2342.txt

│ 234234.txt

│ 2344.txt

│

├─models

│ file.js

│ math.js

│

├─public

│ ├─css

│ │ bootstrap.min.css

│ │

│ ├─fonts

│ │ glyphicons-halflings-regular.eot

│ │ glyphicons-halflings-regular.svg

│ │ glyphicons-halflings-regular.ttf

│ │ glyphicons-halflings-regular.woff

│ │ glyphicons-halflings-regular.woff2

│ │

│ └─js

│ bootstrap.js

│ bootstrap.min.js

│ jquery.min.js

│ npm.js

│

└─views

index.ejs

result.ejs

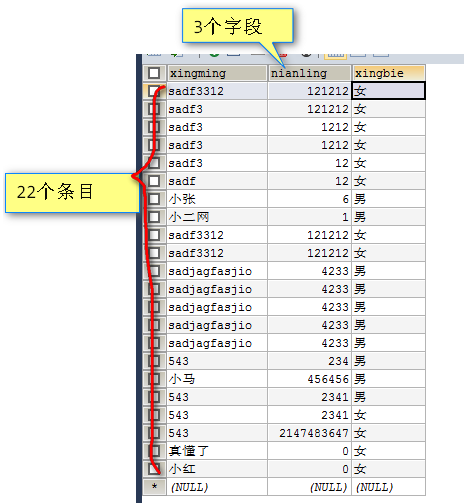
# 三、MongoDB芒果数据库

## 3.1 NoSQL简介

SQL就是Structor Query Language 结构化查询语言。

MySQL、Oracle、SQL Server数据库都是SQL数据库，在大数据时代有些场景使用他们显得太笨重。

**SQL有一个非常大的缺点，就是限制字段：比如现在数据库中已经有10万条数据了（在今天10万条都不算大数据），如果想从下一条数据开始增加一些字段，此时之前的10万条数据都需要一同被更改。SQL中，表的字段是不能自由更改的，不能某一个条目有一些字段，另外的条目的没有。**



**SQL也有优点，能够轻松执行复杂查找。**比如寻找所有大于18岁的且是女生的人的妈妈的年龄，**这里涉及了主从查找，SQL的主从查找非常强**。但是在大数据时代，这种寻找有多少呢？试想，你面对10086、10010的详单，你会有多少复杂查询的几率呢？基本上都是限制日期查询。

**总结一下：在大数据时代，SQL的优势在衰减，而缺点在放大。**

NoSQL应运而生，NoSQL就是Not Only SQL，不仅仅是SQL。**非关系型数据库**。

**NoSQL中没有限制字段，只有条目。每个条目可以自由设置字段。**

我们学习最经典的MongoDB数据库，它是NoSQL。

## 3.2 MongoDB的安装

非常简单，简单的令人发指。

芒果数据库，官网https://www.mongodb.com/



下载可以选择zip格式或者msi格式，msi格式就是下一步、下一步安装。我们讲解zip格式。

|  |  |
| --- | --- |
|  | 解压缩这个文件夹，千万不要有中文路径 |
|  | 解压到c:\Program Files文件夹中 |
|  | 解压之后找到你的bin文件夹，再次确认没有中文路径。 |
|  | 将这个bin文件夹设置为系统的环境变量。 |
|  | 至此你已经成功安装了，在系统任何盘符下能够输入mongo命令不会报“mongo不是可以执行的命令”错误。 |

输入mongo之后，提示你没有安装 KB2731284补丁，此时就要安装发给你的补丁文件。

如果不提示你，就不需要安装：



## 3.3 数据库的基本操作

首先需要使用mongod来“开机”，表示打开数据库。用mongod来开机。

要找一个地方存储数据库。在C盘建立mydatabase，最好在C盘，因为有的系统有安全限制。

开机命令：

$ mongod --dbpath c:\mydatabase

--dbpath表示数据库的路径。

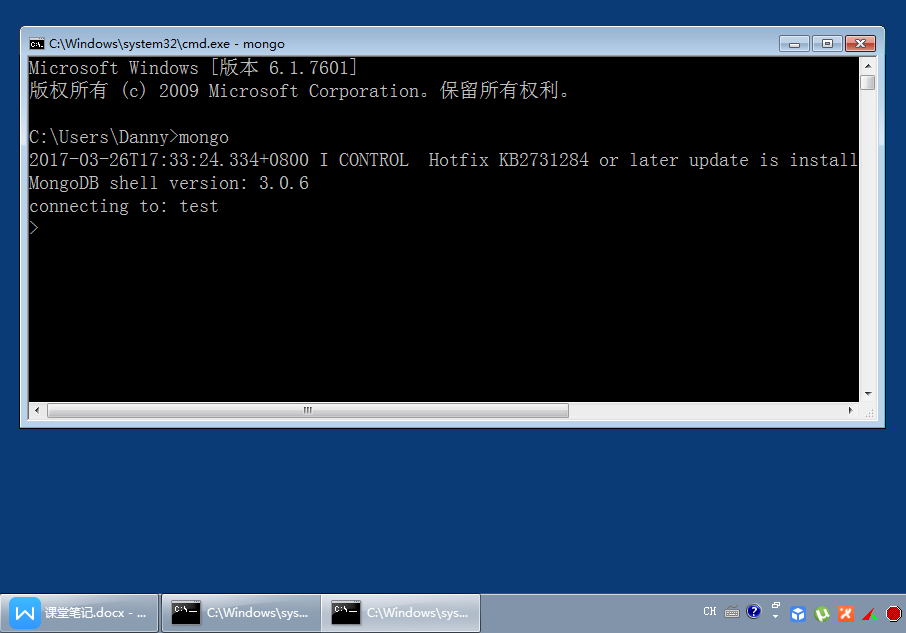
看见这个提示，表示开机正确：



此时光标是挂起状态，没有返回。

此时注意，**这个CMD窗口不能关闭，一关闭CMD窗口，数据库就关闭了**。

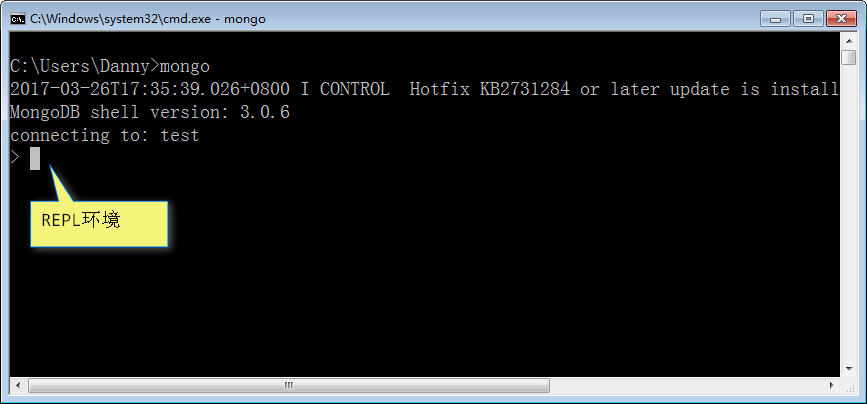
再打开一个新的CMD窗口



输入mongo来管理数据库，一旦输入mongo按回车

$ mongo

此时就能进入mongo的REPL环境（Read-Eval-Print-Loop，读一句、执行一句、显示一句）。



在REPL环境中：

> use student

这行语句表示使用student数据库，此时没有student数据库，MongoDB会帮我们自动创建。

**注意>不需要输入，笔记中的>，一律表示REPL环境！**



此时我们试着插入一条数据：

> db.banji1120.insert({"xingming":"小明","age":12});



banji1120称作“集合”，可以当做“表”。

此时我们想查看这条数据：

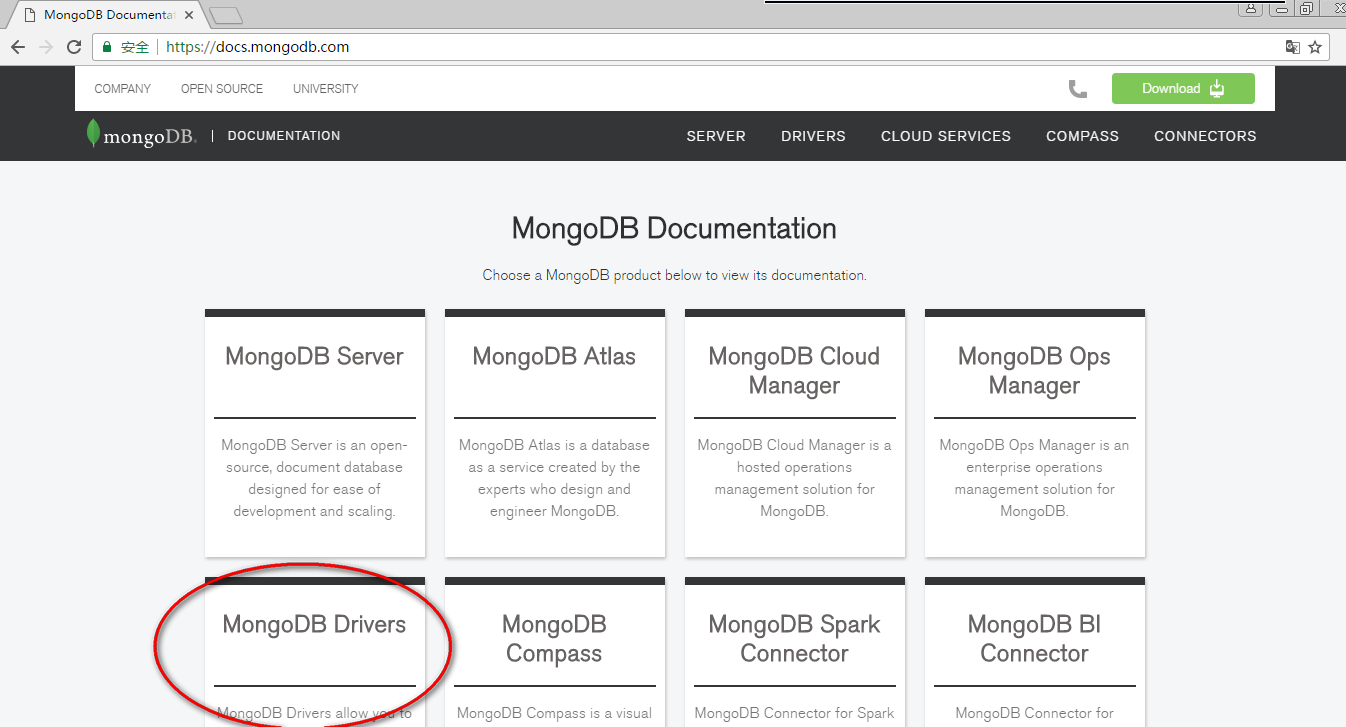
> db.banji1120.find({});



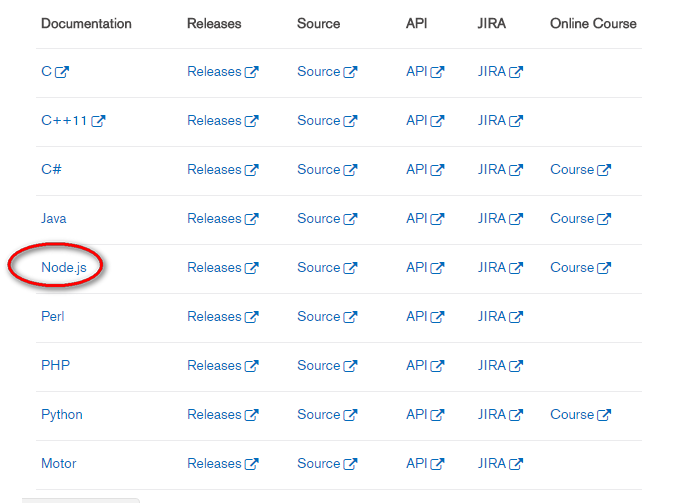
# 四、NodeJS和MongoDB的连接

## 4.1 基本连接

我们无意在REPL环境学习太多语句，我们直接学习NodeJS和MongoDB的连接，让我们的js程序操作MongoDB。



选择node.js：



API：http://mongodb.github.io/node-mongodb-native/2.2/

进入day3文件夹，安装依赖

$ cnpm install --save mongodb

我们写一个01.js案例，就是非常简单的连接数据库

/\*【注意】这个案例不需要背诵！是从

\* http://mongodb.github.io/node-mongodb-native/2.2/quick-start/quick-start/

\* 抄的

\*/

//引入mongodb

var MongoClient = require('mongodb').MongoClient;

//mongodb的端口号和数据库名字，如果数据库不存在会被创建

var dburl = 'mongodb://localhost:27017/xuesheng';

//链接数据库

MongoClient.connect(dburl, function(err, db) {

if(err){

console.log("错误！数据库没有链接成功！");

console.log("你mongod了么？请使用mongod --dbpath 先开机！");

return;

}

console.log("数据库连接成功");

//对数据库的所有操作，写这里！

//对数据库的所有操作，写这里！

//对数据库的所有操作，写这里！

//对数据库的所有操作，写这里！

//对数据库的所有操作，写这里！

});

注意node 01.js之前，一定要记得

$ mongod --dbpath c:\mydatabase

不能关闭CMD！！

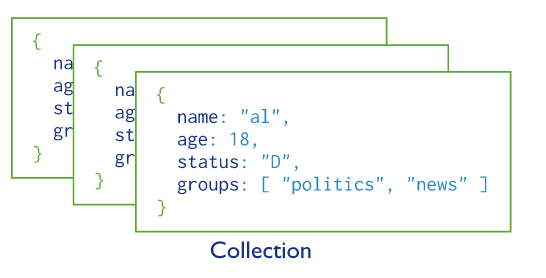
$ node 01.js



## 4.2 集合和文档的概念

数据库由集合组成，集合就是JSON的集合，每一条JSON叫做一个文档。

集合：



文档：



MongoDB用JSON来存储数据。

在REPL环境中

> show dbs

显示所有数据库列表

> use xuesheng

使用学生数据库

> show collections

显示当前数据库的所有集合列表

> db.banji1120.find()

查询banji1120中所有的文档

增删改查的API：http://mongodb.github.io/node-mongodb-native/2.2/tutorials/crud/

## 4.3 增

02.js案例，增加：

/\*【注意】这个案例不需要背诵！是从

\* http://mongodb.github.io/node-mongodb-native/2.2/quick-start/quick-start/

\* 抄的

\*/

//引入mongodb

var MongoClient = require('mongodb').MongoClient;

//mongodb的端口号和数据库名字，如果数据库不存在会被创建

var dburl = 'mongodb://localhost:27017/xuesheng';

//链接数据库

MongoClient.connect(dburl, function(err, db) {

if(err){

console.log("错误！数据库没有链接成功！");

console.log("你mongod了么？请使用mongod --dbpath 先开机！");

return;

}

console.log("数据库连接成功");

//插入数据的标准写法：

**db.collection("banji1120").insertOne(**

{

"xingming": "刘备",

"nianling": 12,

"xingbie" : "男"

},

function(err,r){

//回调函数

if(err){

console.log("插入数据失败！");

return;

}

//r是所有的数据库变动信息，常用的r.insertedCount表示插入的条目数量

console.log("成功插入了" + r.insertedCount + "条数据");

//关闭数据库

db.close();

}

**)**

});

插入多条：

//插入数据的标准写法：

**db.collection("banji1120").insertMany(**

**[**

{

"xingming": "刘备",

"nianling": 12,

"xingbie" : "男"

},

{

"xingming": "关羽",

"nianling": 15,

"xingbie" : "男"

},

{

"xingming": "大乔",

"nianling": 10,

"xingbie" : "女"

}

**],**

function(err,r){

//回调函数

if(err){

console.log("插入数据失败！");

return;

}

//r是所有的数据库变动信息，常用的r.insertedCount表示插入的条目数量

console.log("成功插入了" + r.insertedCount + "条数据");

//关闭数据库

db.close();

}

**);**

## 4.4 删

03.js案例，删除所有姓名是刘备的人：

//删除数据的标准写法：

db.collection("banji1120").deleteMany(

{

"xingming": "刘备"

},

function(err,r){

//回调函数

if(err){

console.log("删除数据失败！");

return;

}

//r是所有的数据库变动信息，常用的r.insertedCount表示插入的条目数量

console.log("成功删除了" + r.deletedCount + "条数据");

//关闭数据库

db.close();

}

);

## 4.5 查

04.js案例，注意，外壳我没有粘贴在笔记上面，自己看04.js案例：

//查询数据的标准写法：

db.collection("banji1120").**find({"xingming": "刘备"}).toArray(**function(err,results){

//回调函数

if(err){

console.log("查询数据失败！");

return;

}

console.log(results);

//关闭数据库

db.close();

});

且逻辑：寻找姓名是刘备且年龄是12的人，逗号表示且：

db.collection("banji1120").find(**{"xingming": "刘备" , "nianling" : 12}**).toArray(function(err,results){

//回调函数

if(err){

console.log("查询数据失败！");

return;

}

console.log(results);

//关闭数据库

db.close();

});

大于逻辑：寻找年龄大于5岁的人，$gt表示大于，$lt表示小于：

db.collection("banji1120").**find({"nianling" : {$gt : 5}})**.toArray(function(err,results){

//回调函数

if(err){

console.log("查询数据失败！");

return;

}

console.log(results);

//关闭数据库

db.close();

});

混合逻辑：寻找年龄大于5岁且是男的人：

db.collection("banji1120").find(

**{"nianling" : {$gt : 5} , "xingbie" : "男"}**

).toArray(function(err,results){

//回调函数

if(err){

console.log("查询数据失败！");

return;

}

console.log(results);

//关闭数据库

db.close();

});

或逻辑： 寻找年龄大于13岁的人，或年龄大于5岁的女人。

//查询数据的标准写法：

db.collection("banji1120").find(

**{$or : [{"nianling" : {$gt : 13} , "xingbie" : "男"} , {"nianling" : {$gt : 5} , "xingbie" : "女"}]}**

).toArray(function(err,results){

//回调函数

if(err){

console.log("查询数据失败！");

return;

}

console.log(results);

//关闭数据库

db.close();

});

## 4.6 改

改有两种，第一增量改；第二彻底改。

先说彻底改，工作中一点用都没有！注意，举动危险！！下面的语句是错误的，会将刘备这个数据彻底变为nianling:45，其他的属性丢失！！

//改变数据

db.collection("banji1120").updateOne(

{"xingming":"刘备"},{"nianling":45}

,function(err,r){

//回调函数

if(err){

console.log("查询数据失败！");

return;

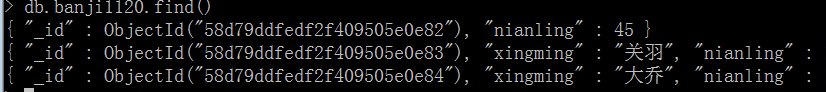
}

console.log("成功");

//关闭数据库

db.close();

});



05.js案例，正确的增量改变的方法：

//改变数据

db.collection("banji1120").updateOne(

**{"xingming":"刘备"},{$set : {"nianling":45}}**

,function(err,r){

//回调函数

if(err){

console.log("查询数据失败！");

return;

}

console.log("成功");

//关闭数据库

db.close();

});