

Node.js入门和企业级项目开发

第7天课堂笔记（本课程共8天）

前端与移动开发学院

http://web.itcast.cn

# 目录

[目录 2](#_Toc7215)

[一、复习 3](#_Toc28269)

[二、索引index 5](#_Toc13934)

[二、Mongoose 6](#_Toc24043)

[2.1 数据库连接 7](#_Toc30184)

[2.2 定义模型 7](#_Toc7107)

# 一、复习

ejs模式：

|  |
| --- |
| 1. //写服务 2. app.get("/allstudent",function(req,res,next){ 3. db.find("students",{},function(err,result){ 4. //寻找完毕之后，result就是一个数组，说实话服务已经成功了 5. //但是，必须界面必须要可视化，所以要有模板引擎呈递 6. res.render("allstudent",{ 7. "result" : result; 8. }); 9. //如果不用模板引擎呈递，可以直接输出JSON，前台用Ajax或者Angular调用 10. //var obj = {"result":result}; 11. //res.json(obj); 12. }); 13. }); 14. <% for(var i = 0 ; i < result.length ; i++) {%> 15. <div class="grid"> 16. <p class="title"><%=result[i].name%></p> 17. <p class="xuehao"><%=result[i].xuehao%></p> 18. <p class="sex"><%=result[i].sex%></p> 19. </div> 20. <%}%> |

Ajax模式：

|  |
| --- |
| 1. //写服务 2. app.get("/allstudent",function(req,res,next){ 3. db.find("students",{},function(err,result){ 4. //如果不用模板引擎呈递，可以直接输出JSON，前台用Ajax或者Angular调用 5. var obj = {"result":result}; 6. res.json(obj); 7. }); 8. }); 9. ====================================== 10. <script type="text/template" id="tem"> 11. //这里是一个underscore模板 12. <div class="grid"> 13. <p class="title">{{=name}}</p> 14. <p class="xuehao">{{=xuehao}}</p> 15. <p class="sex">{{=sex}}</p> 16. </div> 17. </script> 18. 前端页面： 19. jQuery片段： 20. //得到模板html 21. var templateString = $("#tem").html(); 22. //ajax请求 23. $.get("/allsutdent",function(result){ 24. //这个result就是一个json对象 25. //要放到页面上，所以为了不字符串拼接，可以用模板引擎underscore 26. for(var i = 0 ; i < result.result.length ; i++){ 27. //组装模板 28. var compiled = \_.compiled(templateString,{ 29. "name" : result.result[i].name; 30. "sex" : result.result[i].sex; 31. "xuehao" : result.result[i].xuehao; 32. }); 33. //上DOM 34. $(".liebiao").append($(compiled)); 35. } 36. }); |

# 二、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 }** ); |

# 二、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。

## 2.1 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. }); |

## 2.2 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！

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |