LUO JIN 2020-05-17 (SUN)

**MongDB**

[第一部分 3](#_Toc40806181)

[1, 数据库概述和环境搭建 3](#_Toc40806182)

[1.1 为什么要使用数据库 3](#_Toc40806183)

[1.2 什么数据库 3](#_Toc40806184)

[1.3 下载安装 3](#_Toc40806185)

[1.4 MongoDB 可视化软件: compass 3](#_Toc40806186)

[1.5 数据库 3](#_Toc40806187)

[1.6 MongoDB第三方模块 3](#_Toc40806188)

[1.7 启动mongoDB 3](#_Toc40806189)

[1.8 数据库连接 4](#_Toc40806190)

[1.9 创建数据库 4](#_Toc40806191)

[3. MongoDB 增删查改操作 4](#_Toc40806192)

[3.1 创建集合 4](#_Toc40806193)

[3.2 创建文档 - 向集合插入数据 4](#_Toc40806194)

[3.2.1 方法一： 创建集合实例 4](#_Toc40806195)

[3.3 mongoDB 数据库导入数据 5](#_Toc40806196)

[3.4 查询文档 6](#_Toc40806197)

[3.5 删除文档 7](#_Toc40806198)

[3.6 更新文档 7](#_Toc40806199)

[3.6 mongoose验证 7](#_Toc40806200)

[3.7 集合关联（实现） 9](#_Toc40806201)

[3.8 案例： 用户的增删改查 10](#_Toc40806202)

[第二部分 11](#_Toc40806203)

[1, 模板引擎得基础概念 11](#_Toc40806204)

[1.1 模板引擎 11](#_Toc40806205)

[1.2 art-template模板引擎 11](#_Toc40806206)

[1.3 案例 11](#_Toc40806207)

[2. 模板引擎语法 12](#_Toc40806208)

[2.1 模板语法 12](#_Toc40806209)

[2.2 输出 12](#_Toc40806210)

[2.3 原文输出 12](#_Toc40806211)

[2.4 条件判断 13](#_Toc40806212)

[2.5 循环 13](#_Toc40806213)

[2.6 子模版 14](#_Toc40806214)

[2.7 模板继承 15](#_Toc40806215)

[2.8 模板继承示例 15](#_Toc40806216)

[2.9 模板配置 16](#_Toc40806217)

[3， 案例 17](#_Toc40806218)

[3.1 案例介绍 – 学生档案管理 17](#_Toc40806219)

[3.2 项目制作流程 17](#_Toc40806220)

[3.3 第三方模块 router 17](#_Toc40806221)

[3.4 第三方模块 serve-static 18](#_Toc40806222)

[3.5. 添加学生信息功能步骤分析 18](#_Toc40806224)

[3.6 学生信息列表页面分析 19](#_Toc40806225)

# 第一部分

# 1, 数据库概述和环境搭建

## 1.1 为什么要使用数据库

动态网站的数据存储 <http://www.czxy.com/artical?id=1> id不一样页面不一样

持久存储客户端通过表单手机的用户信息

数据库软件本身可以对数据进行搞笑的管理

## 1.2 什么数据库

数据库就是存储数据，可以将数据进行有序的分门别类的存储。他是独立于语言之外的软件，通过API操作 **Node.js** 🡨--（**数据库提供的API**，**数据库反馈操作结果**）**---🡪 数据库**

常见的数据库软件：mysql(php), mongoDB, oracle

## 1.3 下载安装

## 1.4 MongoDB 可视化软件: compass

## 1.5 数据库

在一个数据库软件中，可以包含多个数据仓库，在每个数据仓库中可以包含多个数据ihe，每个数据集合可以包含多个文档（具体的数据）



## 1.6 MongoDB第三方模块

使用node.js 操作MongoDB数据库需要依赖node,js的第三方包mongoose

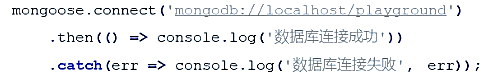
Npm install mongoose

## 1.7 启动mongoDB

命令行工具：net start mongoDB / net stop mongodb

## 1.8 数据库连接

Connect



## 1.9 创建数据库

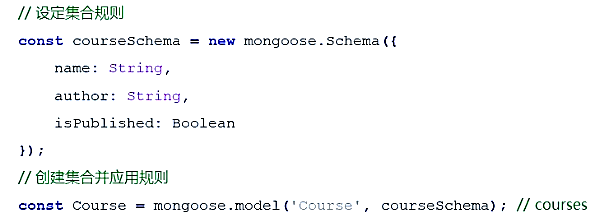
不用显示的去创建数据库，如果正在使用的不存在，会自动帮你创建

# 3. MongoDB 增删查改操作

## 3.1 创建集合

1）创建集合规则

2）创建集合 mongoose.Schema构造函数的实例=创建集合



Model方法中的参数：

- 第一个参数：‘集合名称首字母大写‘，- 他自己在数据库创建的是小写而且是复数

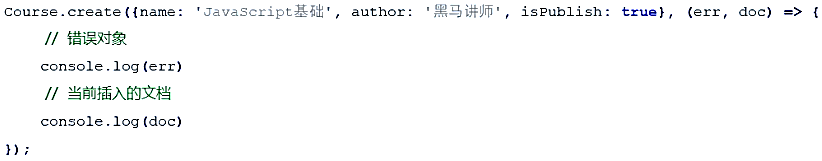
- 第二个参数： 集合的规则

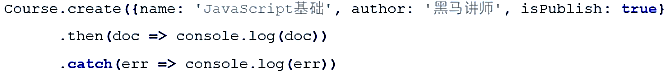
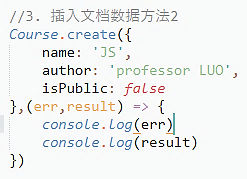
## 3.2 创建文档 - 向集合插入数据

### 3.2.1 方法一： 创建集合实例

1. 创建集合实例
2. 调用实例对象下的sava方法保存到数据库中

 3.2.2 方法二：构造函数的create方法



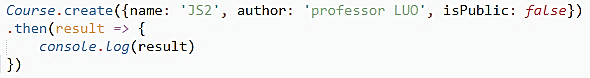


Create也返回 Promise对象

参数一：对象

参数二：回调函数

在数据库中所有操作都是异



## 3.3 mongoDB 数据库导入数据

命令：mongoimport -d 数据库名称 -c 集合名称 -file 要导入的数据

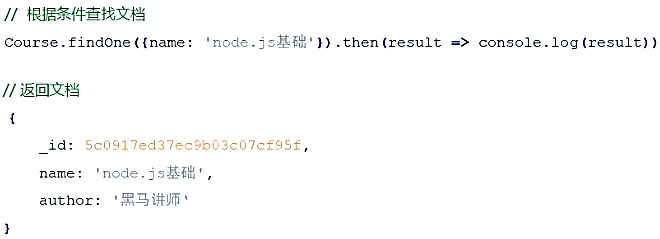
但是我们不能使用现在，因为命令行回去计算机查找mongimport的可执行文件，**需要手动将命令添加到系统环境变量**中

mongoimport -d playground -c users --file ./user.json

## 3.4 查询文档

1） Find() **方法也返回promise对象**





**//3, 查询用户集合中的所有文档**

//User.**find().**then(result => console.log(result));

//3.1 通过id字段查找

//User.find({\_id: '5c09f267aeb04b22f8460968'}).then(result => console.log(result));

**//4，返回一个，默认返回第一条**

//User.**findOne().**then(result => console.log(result));

//4.1 可以加条件

User.**findOne({name: '李四'}).**then(result => console.log(result));

1. 匹配大于，小于

User.**find({age:** **{$gt: 20, $lt: 50}}).**then(result => console.log(result))

1. 匹配包含 **$in 网站搜索时，在后台查询的时候用**

User.find({hobbies**:{$in:['敲代码']}**}).then(result => console.log(result))

1. 选择要查询的字段

User.find().**select**('name email').then(result => console.log(result));

1. 根据年龄进行升序排列

User.find().**sort(**'age').then(result => console.log(result));

1. 根据年龄进行降序排列

User.find().sort('-age').then(result => console.log(result));

1. Skip 跳过多少条数据，limit限制查询数量 = 翻页使用；跳过前两个文档，只显示3个数

User.find().**skip(2).limit(3).**then(result => console.log(result));

## 3.5 删除文档

1) 删除单个，也返回promise对象，通过then; 查找到一条文档并删除，返回删除的文档，

User.**findOneAndDelete({\_id: '5c09f2d9aeb04b22f846096b'}).**then(result => console.log(result));

2) 删除多个,不传全部删除，要小心, 参数也是有条件的

User.**deleteMany({})**.then(result => console.log(result));

## 3.6 更新文档

1) 更新文档

User.updateOne({查询条件}，{要修改的值}).then(result => console.log(result));

User.**updateOne({name:'李四'},{name: '李狗蛋'}).**then(result=>console.log(result));

2) 更新多个

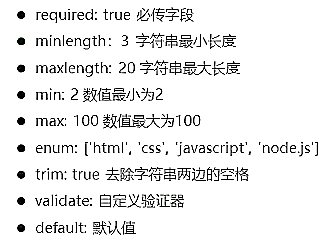
User.updateMany({查询条件}，{要修改的值}).then(result => console.log(result));

User.**updateMany({},{age: 56}).**then(result=>console.log(result));

## mongoose验证

在创建集合规则时，可以设置当前字段的验证规则，验证失败就则插入失败

* Required: true 必传字段
* Min/max: 针对数值这样的字段类型



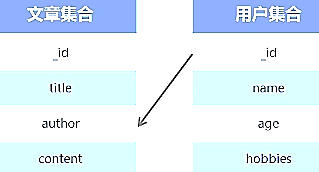


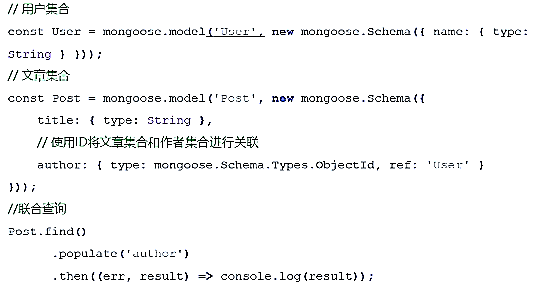


## 集合关联（实现）

通常**不同集合的数据之间是具有关系的**，例如文章信息和用户信息存在不同的集合中，但文章时某个用户发表的，要查询文章的所有信息包括发表用户，就需要用到集合关联

* 使用id对集合进行关联：查找当前发布文章的作者信息





* 1, 创建用户集合规则 2，创建文章集合规则/数据库中创建首字母大写的用户集合和文章集合
* 在文章集合的作者中，关联用户集合 type: mongoose.Schema.Types.ObjectId, ref: 'User'
* 3，创建用户
* 4, 创建一篇文章
* 5, 查询作者信息 Post.find().populate('author').then(result=>console.log(result))



## 3.8 案例： 用户的增删改查

1）搭建网站服务器，实现客户端与服务器端的通信

2）连接数据库，创建用户集合，向集合中插入文档

3）当用户访问/list时，将所有的用户信息查询出来

4）将用户访问/add时，呈现表单页面，并实现添加用户信息功能

6）当用户访问/modify时，呈现修改页面，并实现用户信息修改功能

1. 当用户访问/delete时，实现用户删除功能

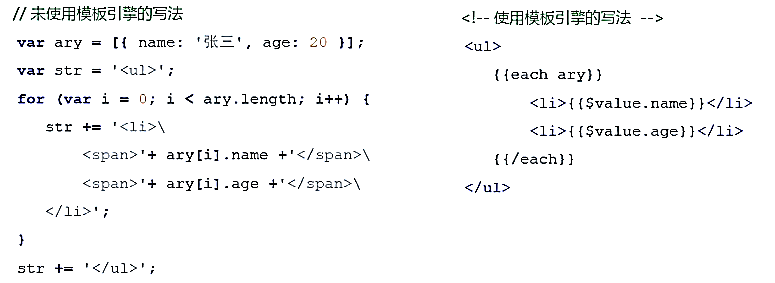
# 第二部分

# 1, 模板引擎得基础概念

## 1.1 模板引擎

第三方模块。

让开发者以更加友好得方式拼接字符串，是项目代码更加清晰，更加易于维护



## 1.2 art-template模板引擎

1）种类很多，这个是由腾讯公司出品，目前运行最快的，公司使用最多的。

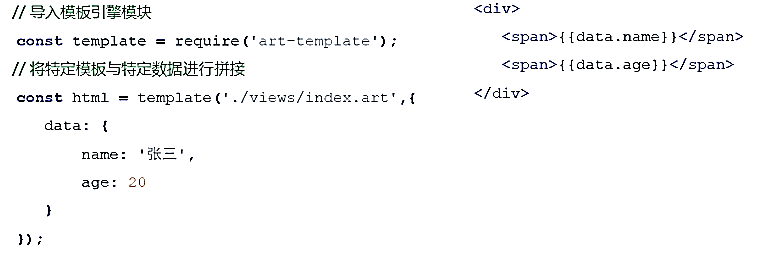
2）下载： npm install art-template

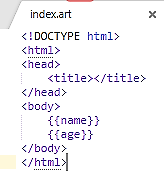
3) 使用const template = require(‘art-template’) 引入

4）调用它，告诉模板引擎要拼接的数据和模板在哪 const html = template(“模板路径”，数据)；

## 1.3 案例

1） 后缀 .art





# 2. 模板引擎语法

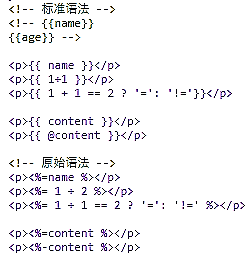
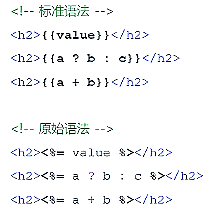
## 2.1 模板语法

1） art-template 同时支持两种模板语法：**标准语法，原始语法**

2）**标准语法**：容易就阅读 **{{数据}}**

**原始语法**：强大的裸机处理能力 **<%=数据 %>**

## 2.2 输出



## 2.3 原文输出

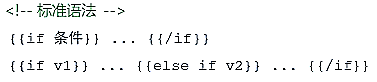
1）如果数据中携带HTML标签，默认模板引擎不会解析标签，会将其转义后输出

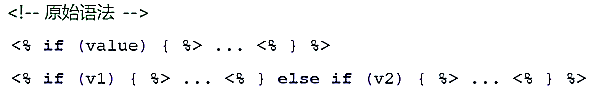
2）标准语法：{{ @数据 }}

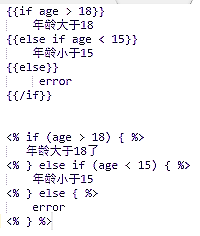
3）原始语法：<%-数据 %>

## 2.4 条件判断

1） 在模板中可以根据条件决定展是那块HTML代码





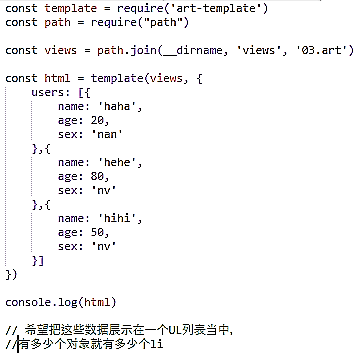
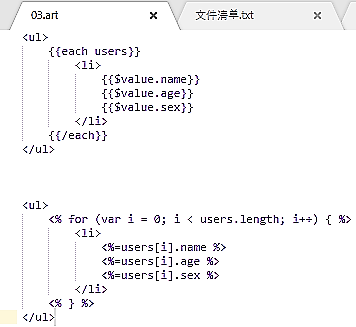
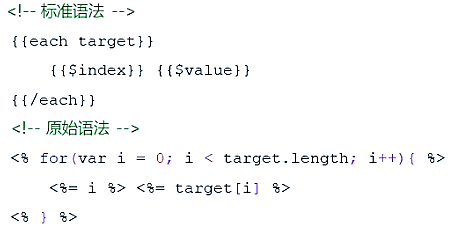


## 2.5 循环

1）从数据库中查询数据，会返回一个数组，包含多个对象，如何展示在页面中？

2）标准语法：{{ each 数据}} {{/each}}

3) 原始语法： <% for() {%> <%}%>



## 2.6 子模版

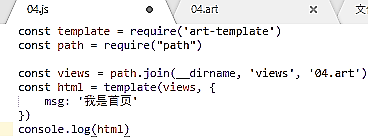
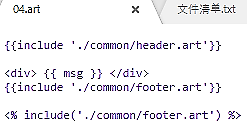
1）使用子模版可以净网站公共区块（头部，底部）抽离到单独的文件中去

2）标准语法：{{include ‘子模版的路径‘}}

3）原始语法：<%include(‘子模版的路径‘) %>

4) {{include ‘./header.art’}}

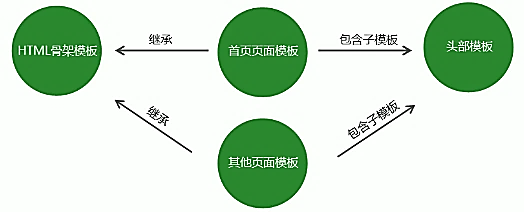
5) <%include(‘./header.art’) %>

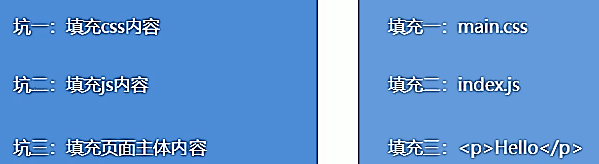


## 2.7 模板继承

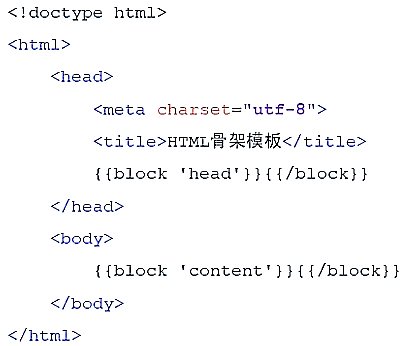
1) html骨架在每个页面也属于公共部分

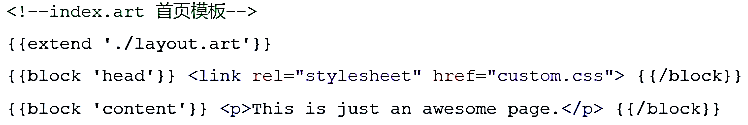
2）模板继承可以将网站HTML股价抽离到单独的文件中，其他页面模板可以继承骨架文件





## 2.8 模板继承示例





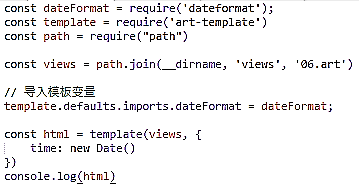
## 2.9 模板配置

1） 向模板中导入变量 template.defaults.imports.变量名 = 变量值

- 从数据库中查时间，是原始的，要处理

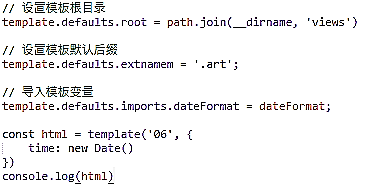
- 日期处理的第三方模块

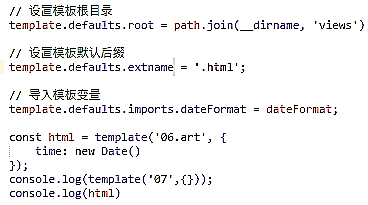
- **{{dateFormat(time, 'yyyy-mm-dd')}}**



2）设置模板根目录 template.defaults.root = path.join(\_\_dirname, 'views')

3）设置模板默认后缀





# 3， 案例

## 3.1 案例介绍 – 学生档案管理

- 目标：模板引擎应用，强化node.js项目制作流程

- 知识点：http请求响应，数据库，模板引擎，静态资源访问

- 后期优化代码

## 3.2 项目制作流程

1）建立项目文件夹并生成描述文件

2) 创建数据库服务器实现客户端和服务器端通信

3）连接数据库并根据需求设计学院信息表

4）创建路由并实现页面模板呈递

- npm install router

- 下载模板引擎 npm i art-template

5）实现静态资源访问 – css, js, 图片文件等

6）实现学生信息添加功能

7）实现学生信息展示功能

## 3.3 第三方模块 router

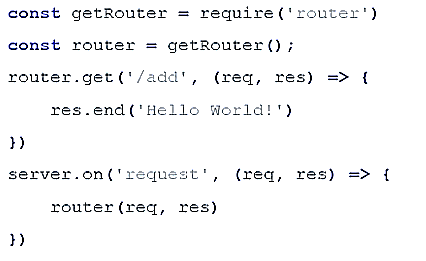
- **功能**：实现路由

- **使用步骤：**1，获取路由对象

2，调用路由对象提供的方法创建路由

3，启用路由，是路由生效

-代码： **npm install router**

、

## 3.4 第三方模块 serve-static

- 功能：实现静态资源访问功能

- 使用步骤：

1，引入serve-static 模块获取创建静态资源服务功能的方法

2，调用方法创建静态资源服务并指定静态资源服务目录，当参数传递给他

3，启用静态资源服务功能：调用它

* 代码：npm install serve-static

# 

## 3.5. 添加学生信息功能步骤分析

1）需要向服务器发请求的post，在模板的表单(action,method)中执行请求地址与请求方式

2）为每一个表单添加name属性

3）添加实现信息功能路由，处理请求

4）接收客户端请求来的学生信息

5）将学生信息太牛加到数据库

6）将页面重定向到学生信息页面

## 3.6 学生信息列表页面分析

1）将数据库中多有的学生信息查询出来

2）通过模板引擎将学生信息和HTML模板进行拼接

3）将拼接好的模板响应给客户端

- npm install dateformat 处理时间

model: 数据库相关代码

public： 静态资源

route: 路由

views: 模板

app.js: 入口文件，主文件





