MongoDB相关笔记

目 录

[第一章 数据库介绍、安装、使用 4](#_Toc2076)

[1. 数据库和文件的主要区别 4](#_Toc11731)

[2. NoSql 介绍 4](#_Toc7220)

[2.1. NoSQL 介绍 4](#_Toc14385)

[2.2. NoSQL 应用情况介绍 4](#_Toc24015)

[2.3. NoSQL 发展现状 5](#_Toc13163)

[3. 什么时候建议使用 NoSql 5](#_Toc1441)

[4. NoSql 和传统数据库简单对比。 5](#_Toc25230)

[5. NoSql 种类 6](#_Toc11022)

[5.1. 键值(key-value)存储数据库 6](#_Toc26088)

[5.2. 列存储数据库 6](#_Toc8633)

[5.3. 文档型数据库 6](#_Toc27869)

[5.4. 图形(Graph)数据库 7](#_Toc1175)

[6. MongoDb 介绍 7](#_Toc13093)

[7. MongoDb 安装 7](#_Toc25271)

[8. 使用 MongoDb 9](#_Toc3330)

[第二章 MongoDB 数据库创建删除、表创建删除、数据增删改查 10](#_Toc25363)

[1. 数据库使用 10](#_Toc8761)

[2. 创建数据库 10](#_Toc8235)

[3. 插入（增加）数据 11](#_Toc13784)

[4. 查找数据 11](#_Toc16698)

[5. 修改数据 14](#_Toc2308)

[6. 删除数据 15](#_Toc17985)

[第三章 MongoDB 索引 和 explain 的使用 15](#_Toc6506)

[1. 索引基础 15](#_Toc25918)

[2. 唯一索引 17](#_Toc16392)

[3. 索引的一些参数 18](#_Toc2631)

[4. 使用 explain 18](#_Toc7014)

[5. explain executionStats 查询具体的执行时间 19](#_Toc22887)

[第四章 Nodejs 操作 MongoDb 数据库 19](#_Toc18554)

[1. 在 Nodejs 中使用 Mongodb 19](#_Toc18538)

[2. Nodejs 创建服务 20](#_Toc18382)

[3. Nodejs 连接 MongoDb 数据库 20](#_Toc22522)

[4. Nodejs 查找 MongoDb 数据库集合 20](#_Toc6297)

[5. Nodejs 给 MongoDb 增加数据 21](#_Toc28352)

[6. Nodejs 修改 MongoDb 数据 21](#_Toc8689)

[7. Nodejs 删除 MongoDb 数据 22](#_Toc11872)

[第五章 Mongodb4.x 的使用以及 Mongodb 账户权限配置 22](#_Toc575)

[1. Mongodb4.x 的使用 22](#_Toc6540)

[2. Mongodb 账户权限配置（了解） 25](#_Toc24474)

[2.1. 创建超级管理用户 25](#_Toc23428)

[2.2. 修改 Mongodb 数据库配置文件 25](#_Toc26550)

[2.3. 重启mongodb 服务 25](#_Toc17597)

[2.4. 用超级管理员账户连接数据库 26](#_Toc25735)

[2.5. 给 eggcms 数据库创建一个用户 只能访问 eggcms 不能访问其他数据库 26](#_Toc28029)

[3. Mongodb 账户权限配置中常用的命令 26](#_Toc13004)

[4. Mongodb 数据库角色 26](#_Toc14007)

[4.1. 数据库用户角色 26](#_Toc4636)

[4.2. 数据库管理角色 27](#_Toc25197)

[4.3. 集群管理角色 27](#_Toc16938)

[4.4. 备份恢复角色 27](#_Toc2134)

[4.5. 所有数据库角色 27](#_Toc16217)

[4.6. 超级用户角色 28](#_Toc4265)

[4.7. 内部角色 28](#_Toc16609)

[5. 连接数据库的时候需要配置账户密码 28](#_Toc28208)

[第六章 关系型数据库表（集合）与表（集合）之间的几种关系 28](#_Toc27914)

[1. 一对一的关系 28](#_Toc23066)

[2. 一对多的关系 29](#_Toc7238)

[3. 多对多的关系 29](#_Toc10011)

[第七章 MongoDB 的高级查询 aggregate 聚合管道 30](#_Toc2621)

[1. MongoDB 聚合管道（Aggregation Pipeline） 30](#_Toc12003)

[2. MongoDB Aggregation 管道操作符与表达式 31](#_Toc3072)

[3. 模拟数据 32](#_Toc13296)

[4. $project 33](#_Toc23801)

[5. $match 33](#_Toc9027)

[6. $group 33](#_Toc24671)

[7. $sort 33](#_Toc15502)

[8. $limit 34](#_Toc25389)

[9. $skip 34](#_Toc7357)

[10. $lookup 表关联 34](#_Toc2800)

[第八章 mongoose 入门以及 mongoose 实现数据的增、删、改、查 34](#_Toc17953)

[1. mongoose 介绍 35](#_Toc16359)

[2. mongoose 的安装以及使用 35](#_Toc23159)

[2.1. 安装 35](#_Toc5454)

[2.2. 引入 mongoose 并连接数据库 35](#_Toc7844)

[2.3. 定义 Schema 35](#_Toc2246)

[2.4. 创建数据模型 35](#_Toc23777)

[2.5. 查找数据 36](#_Toc31290)

[2.6. 增加数据 36](#_Toc8612)

[2.7. 修改数据 36](#_Toc1970)

[2.8. 删除数据 37](#_Toc28822)

[2.9. 保存成功查找 37](#_Toc5358)

[2.10. 模块化 37](#_Toc14919)

[3. mongoose 模块化 38](#_Toc17610)

[第九章 Mongoose 预定义模式修饰符 Getters与 Setters 自定义修饰符 38](#_Toc24711)

[1. mongoose 预定义模式修饰符 38](#_Toc6831)

[2. mongoose 的安装以及使用 39](#_Toc18062)

[2.1. set方法 39](#_Toc27806)

[2.2. get方法（没用） 40](#_Toc4950)

[第十章 Mongoose 索引 41](#_Toc21154)

[1. Mongoose 索引 41](#_Toc16343)

[2. 索引基础 42](#_Toc26271)

[3. 唯一索引 43](#_Toc6144)

[4. 索引的一些参数 44](#_Toc32736)

[5. 使用 explain 44](#_Toc19622)

[6. explain executionStats 查询具体的执行时间 45](#_Toc12244)

[第十一章 Mongoose 内置 CURD 方法、扩展 Mongoose Model 的静态方法和实例方法 45](#_Toc26047)

[1. Mongoose 内置 CURD 45](#_Toc12748)

[2. 扩展 Mongoose CURD 方法 46](#_Toc8100)

[第十二章 Mongoose 数据校验 47](#_Toc19918)

[1. Mongoose 校验参数 47](#_Toc21295)

[2. Mongoose 自定义的验证器 48](#_Toc28325)

[第十三章 Mongoose 中使用 aggregate 聚合管道 49](#_Toc18408)

[1. MongoDB 聚合管道（Aggregation Pipeline） 49](#_Toc28125)

[2. Mongoose aggregate 多表关联查询 49](#_Toc2223)

[第十四章 Mongoose 中使用 populate 实现关联查询(不推荐) 50](#_Toc22134)

[1. Mongoose populate 官方文档 50](#_Toc1156)

[2. Mongoose populate 关联查询 50](#_Toc29194)

[2.1. 定义 ref(表示外键) 51](#_Toc19351)

[2.2. 关联查询 51](#_Toc32005)

[第十五章 MongoDb 数据库的导出导入 52](#_Toc299)

* 1. 数据库介绍、安装、使用
     1. 数据库和文件的主要区别

数据库有数据库表、行和列的概念，让我们存储操作数据更方便

数据库提供了非常方便的接口，可以让 nodejs、php java .net 很方便的实现增加修改删除功能。

* + 1. NoSql 介绍
       1. NoSQL 介绍

由于互联网的迅速发展，云计算与 Web2.0。这样大量的交互给数据库提出了更高的性能要求，传统的数据库（本文泛指 SQL 数据库），即关系数据库虽然具备良好的事物管理，但在处理大量数据的应用时很难在性能上满足设计要求。

NoSQL 就是主要为了解决当下大量高并发高要求的数据库应用需求，关系数据库具有严格的参照性，一致性，可用性，原子性，隔离性等特点，因此会产生一些例如表连接等操作，这样会大大降低系统的性能。而在当前很多应用场景下对性能的要求远远强于传统数据库关注的点，NoSQL 就是为了解决大规模数据与多样数据种类等问题，尤其是其中大数据的相关问题。 。

NoSQL(NoSQL = Not Only SQL )，意即“不仅仅是 SQL”，它指的是非关系型的数据库，是以 key-value形式存储，和传统的关系型数据库不一样，不一定遵循传统数据库的一些基本要求，比如说遵循 SQL 标准、ACID 属性、表结构等等。NoSQL 最早被提出是在 20 世纪 80 年代，在当时更多是强调的是与关系数据库区别对待，最近这些年被提及的更多是强调协助解决大数据等相关问题。NoSQL 在大数据时代有自己的意义。

* + - 1. NoSQL 应用情况介绍

国内的互联网蓬勃发展，不仅涌现出 BAT（百度，阿里巴巴，腾讯）之类的巨头，也带动了整个互联网行业的发展，大量的创业型公司如春笋般的涌出，在国家层面也提出了“互联网+”和“万众创业”的口号。更多传统的行业也开始拥抱互联网。但是无论是做所谓的生态平台还是传统业务的转型，涉及到的业务是多种多样的。这个时候企业架构师对于应用系统的核心——数据库管理 不仅有传统的 SQL 选项也有了NoSQL 这种适合特定场景需求的选项。

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。
   * + 1. NoSQL 发展现状

国外： Google 的 BigTable 和 Amazon 的 Dynamo 使用的就是 NoSQL 型数据库。

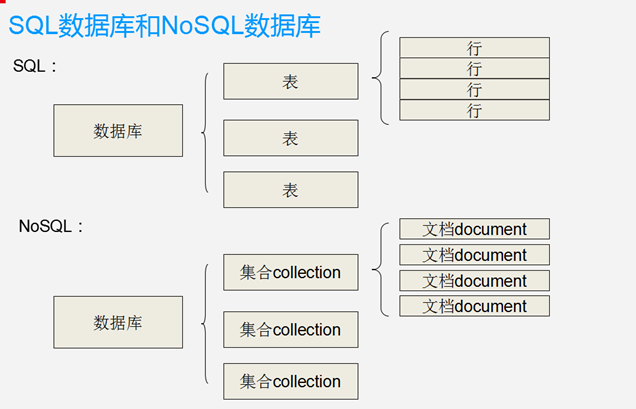
国内：百度、阿里、腾讯、新浪微博、视觉中国、优酷运营数据分析、飞信空间、豆瓣社区等...

* + 1. 什么时候建议使用 NoSql

1. 对数据库高并发读写的需求
2. 对海量数据的高效率存储和访问的需求
3. 对数据库的高可扩展性和高可用性的需求
   * 1. NoSql 和传统数据库简单对比。

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”。



* + 1. NoSql 种类

NoSQL数据库分为四大类

* + - 1. 键值(key-value)存储数据库

这一类数据库主要使用到哈希表，这个表中有一个特定的键和一个指针指向特定的数据。key/value模型对于IT系统来说的优势在于简单、易部署。但是如果DBA只对部分值进行查询或更新的时候，key/value就显得效率低下了。

如：Tokyo Cabinet/Tyrant，Redis，Voldemort，Oracle BDB。

* + - 1. 列存储数据库

这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列。这些列是有列家族来安排的。

如：Cassandra，HBase，Riak。

* + - 1. 文档型数据库

文档型数据库的灵感是来自于Lotus Notes办公软件的，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可以看做是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。

如：CouchDB，MongoDB，国内文档型数据库SequoiaDB，已开源。

* + - 1. 图形(Graph)数据库

图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，他是使用灵活的图形模型，并且能够扩展到多个服务器上。NoSQL数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。许多NoSQL数据库都有REST式的数据接口或者查询API。

如：Neo4J，InfoGrid，Infinite Graph。

* + 1. MongoDb 介绍

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bson 格式，因此可以存储比较复杂的数据类型。Mongo 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它的特点是高性能、易部署、易使用，存储数据非常方便。

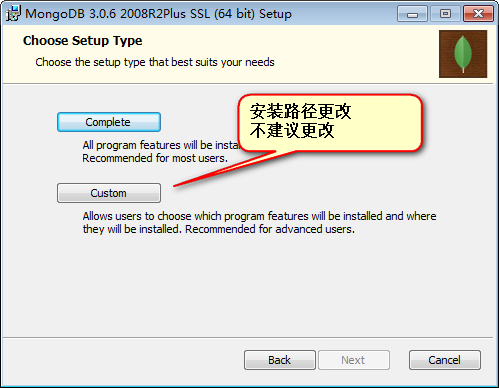
* + 1. MongoDb 安装

官网：https://www.mongodb.com/

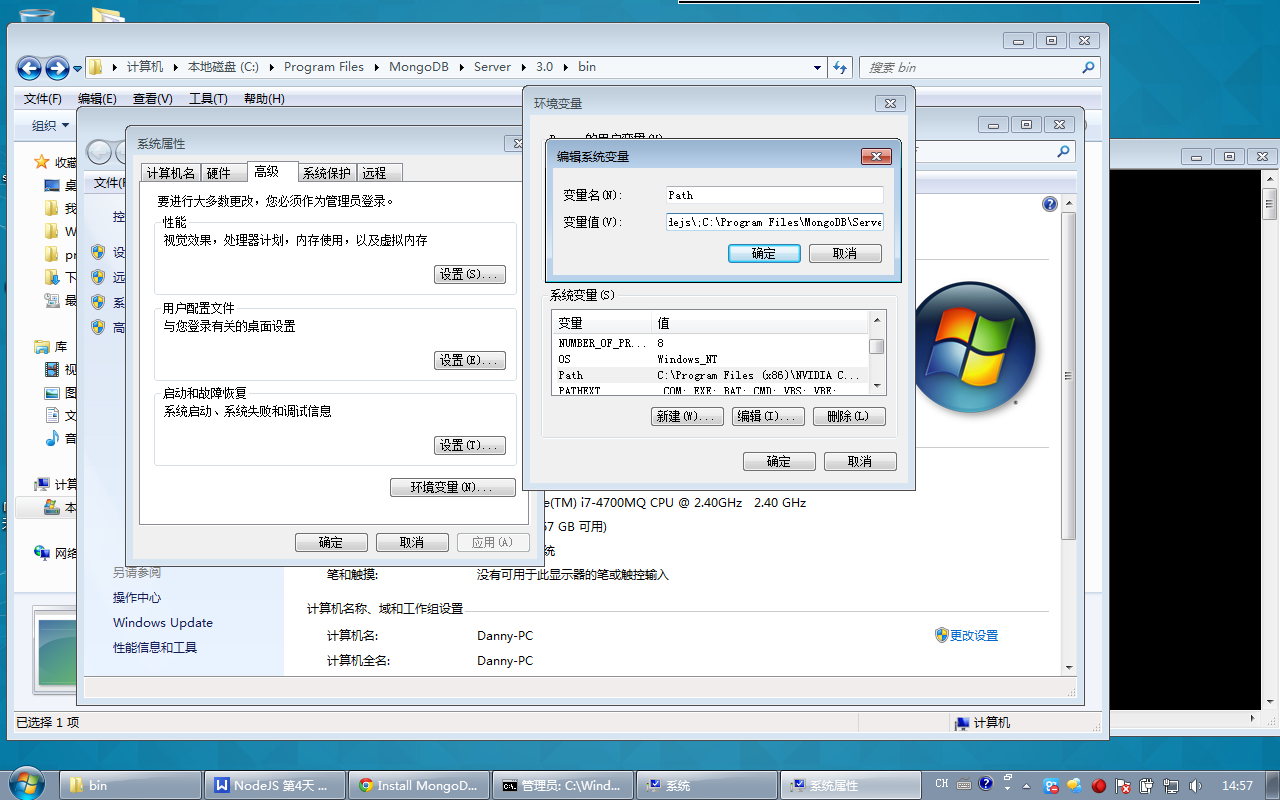
手册：https://docs.mongodb.org/manual/

注意：部分 win7 系统需要安装补丁，KB2731284。

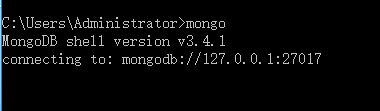
1. 双击 MongoDB 软件下一步下一步安装



1. 安装完成配置环境变量 C:\Program Files\MongoDB\Server\3.0\bin 加入到系统的path 环境变量中



1. 打开 cmd 输入 ：mongo命令看看是否成功。如果出来下图说明 mongodb配置成功。



* + 1. 使用 MongoDb

1. 新建一个存放数据库的文件夹。

**注意**：不能有中文和空格，建议不要放在 C 盘

1. 启动 MongoDb 服务

服务端：mongod 开启数据库服务 mongod --dbpath C:\mongodb

开启 MongoDb 服务命令：



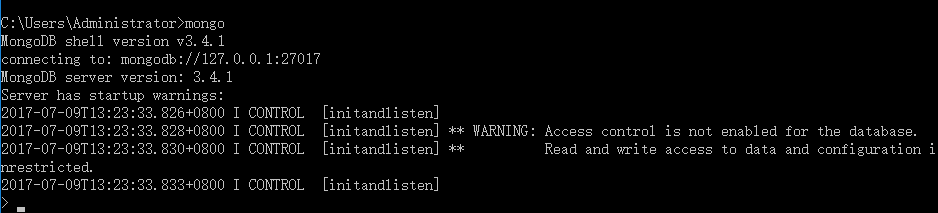
--dbpath 就是选择数据库文档所在的文件夹。

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。

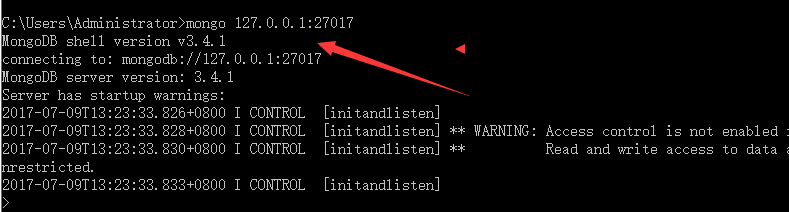
**注意**：一定要保持，开机这个 CMD 不能动了，不能关，不能 ctrl+c。 一旦这个 cmd 有问题了，数据库就自动关闭了。（可以在任务管理器-服务中自动开启，不需要手动打开该服务了）

1. 客户端输入 mongo 命令连接服务端

客户端：mongo（本地数据库）



客户端：mongo 使用数据库 ip 地址:端口号



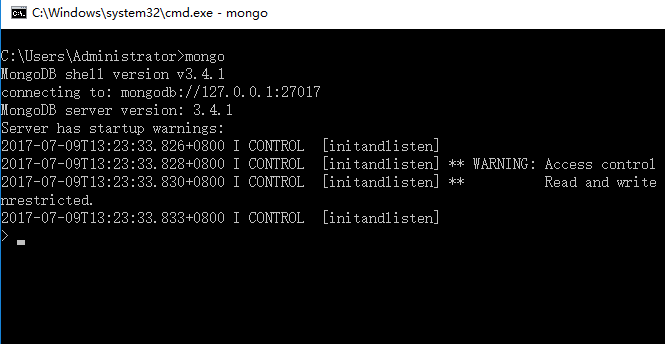
注意：mongo数据库的端口号默认为27017

* 1. MongoDB 数据库创建删除、表创建删除、数据增删改查
     1. 数据库使用

**开启 mongodb 服务**：要管理数据库，必须先开启服务，开启服务使用 mongod --dbpath c:\mongodb



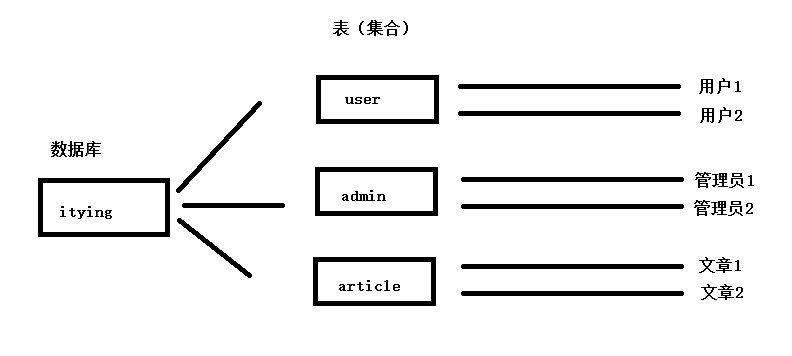
管理mongodb数据库：mongo (一定要在新的cmd中输入)



清屏：cls

查看所有数据库列表：show dbs

* + 1. 创建数据库



使用数据库、创建数据库：use student

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。

不需要专门创建集合，只需要写点语法插入数据就会创建集合：

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。

显示当前的数据集合（mysql 中叫表）：show collections

* + 1. 插入（增加）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。

db.表名.insert({"name":"zhangsan"})

* + 1. 查找数据

1. 查询所有记录

db.userInfo.find();

相当于：select\* from userInfo;

1. 查询去掉后的当前聚集集合中的某列的重复数据

db.userInfo.distinct("name");

会过滤掉 name 中的相同数据

相当于：select distict name from userInfo;

1. 查询 age = 22 的记录

db.userInfo.find({"age": 22});

相当于： select \* from userInfo where age = 22;

1. 查询 age > 22 的记录

db.userInfo.find({age: {$gt: 22}});

相当于：select \* from userInfo where age >22;

1. 查询 age < 22 的记录

db.userInfo.find({age: {$lt: 22}});

相当于：select \* from userInfo where age <22;

1. 查询 age >= 25 的记录

db.userInfo.find({age: {$gte: 25}});

相当于：select \* from userInfo where age >= 25;

1. 查询 age <= 25 的记录

db.userInfo.find({age: {$lte: 25}});

1. 查询 age >= 23 并且 age <= 26 注意书写格式

db.userInfo.find({age: {$gte: 23, $lte: 26}});

1. 查询 name 中包含 mongo 的数据 模糊查询用于搜索

db.userInfo.find({name: /mongo/});

//相当于%%

select \* from userInfo where name like ‘%mongo%’;

1. 查询 name 中以 mongo 开头的

db.userInfo.find({name: /^mongo/});

select \* from userInfo where name like ‘mongo%’;

1. 查询指定列 name、age 数据

db.userInfo.find({}, {name: 1, age: 1});

相当于：select name, age from userInfo;

当然 name 也可以用 true 或 false,当用 ture 的情况下河 name:1 效果一样，如果用 false 就是排除 name，显示 name 以外的列信息。

1. 查询指定列 name、age 数据, age > 25

db.userInfo.find({age: {$gt: 25}}, {name: 1, age: 1});

相当于：select name, age from userInfo where age >25;

1. 按照年龄排序 1 升序 -1 降序

升序：

db.userInfo.find().sort({age: 1});

降序：

db.userInfo.find().sort({age: -1});

1. 查询 name = zhangsan, age = 22 的数据

db.userInfo.find({name: 'zhangsan', age: 22});

相当于：select \* from userInfo where name = ‘zhangsan’ and age = ‘22’;

1. 查询前 5 条数据

db.userInfo.find().limit(5);

相当于：selecttop 5 \* from userInfo;

1. 查询 10 条以后的数据

db.userInfo.find().skip(10);

相当于：select \* from userInfo where id not in (selecttop 10 \* from userInfo);

1. 查询在 5-10 之间的数据

db.userInfo.find().limit(10).skip(5);

可用于分页，limit 是 pageSize，skip 是第几页\*pageSize

1. or 与 查询

db.userInfo.find({$or: [{age: 22}, {age: 25}]});

相当于：select \* from userInfo where age = 22 or age = 25;

1. findOne 查询第一条数据

db.userInfo.findOne();

相当于：select top 1 \* from userInfo;

db.userInfo.find().limit(1);

1. 查询某个结果集的记录条数 统计数量

db.userInfo.find({age: {$gte: 25}}).count();

相当于：select count(\*) from userInfo where age >= 20;

如果要返回限制之后的记录数量，要使用 count(true)或者 count(非 0)

db.users.find().skip(10).limit(5).count(true);

1. 查询某个字段是否存在的记录

db.userInfo.find({ age: { $exists:false } })

如果查询不存在则使用true

* + 1. 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：

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：

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

更改所有匹配项目："

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

注意：完整替换，不出现$set 关键字了

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

db.users.update({name: 'Lisi'}, {$inc: {age: 50}}, false, true);

相当于：update users set age = age + 50 where name = ‘Lisi’;

db.users.update({name: 'Lisi'}, {$inc: {age: 50}, $set: {name: 'hoho'}}, false, true);

相当于：update users set age = age + 50, name = ‘hoho’ where name = ‘Lisi’;

* + 1. 删除数据

db.collectionsNames.remove( { "borough": "Manhattan" } )

db.users.remove({age: 132});

db.restaurants.remove( { "borough": "Queens" }, { justOne: true } )

删除数据库（删除当前所在的数据库）：db.dropDatabase()

删除集合（删除指定集合collection）：db.collection.drop()

删除文档：db.collection.remove(<query>,<justOne>)

MongoDB 是 2.6 版本以后的，语法格式如下：

db.collection.remove(

<query>,

{

justOne: <boolean>,

writeConcern: <document>

}

)

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，如果不设置该参数，或使用默认值 false，则删除所有匹配条件的文档。

writeConcern：（可选）抛出异常的级别。

* 1. MongoDB 索引 和 explain 的使用
     1. 索引基础

索引是对数据库表中一列或多列的值进行排序的一种结构，可以让我们查询数据库变得**更快**。MongoDB 的索引几乎与传统的关系型数据库一模一样，这其中也包括一些基本的查询优化技巧。

**创建索引的命令：**

db.user.ensureIndex({ "username": 1 })

**获取当前集合的索引：**

db.user.getIndexes()

**删除索引的命令是：**

db.user.dropIndex({ "username": 1 })

在 MongoDB 中，我们同样可以创建**复合索引**，如：

数字 1 表示 username 键的索引按升序存储，

-1 表示 age 键的索引按照降序方式存储。

db.user.ensureIndex({ "username": 1, "age": -1 })

该索引被创建后，基于 username 和 age 的查询将会用到该索引，或者是基于 username 的查询也会用到该索引，**但是只是基于 age 的查询将不会用到该复合索引。因此可以说，如果想用到复合索引，必须在查询条件中包含复合索引中的前 N 个索引列**。

然而如果查询条件中的键值顺序和复合索引中的创建顺序不一致的话，MongoDB 可以智能的帮助我们调整该顺序，以便使复合索引可以为查询所用。如：

db.user.find({ "age": 30, "username": "stephen" })

对于上面示例中的查询条件，MongoDB 在检索之前将会动态的调整查询条件文档的顺序，以使该查询可以用到刚刚创建的复合索引。

对于上面创建的索引，MongoDB 都会根据索引的 keyname 和索引方向为新创建的索引自动分配一个索引名，**下面的命令可以在创建索引时为其指定索引名**，如：

db.user.ensureIndex({ "username": 1 }, { "name": "userindex" })

随着集合的增长，需要针对查询中大量的排序做索引。

如果没有对索引的键调用 sort，MongoDB 需要将所有数据提取到内存并排序。因此在做无索引排序时，如果数据量过大以致无法在内存中进行排序，此时 MongoDB 将会报错。

* + 1. 唯一索引

在缺省情况下创建的索引均不是唯一索引。

下面的示例将创建唯一索引，如：

db.user.ensureIndex({ "userid": 1 }, { "unique": true })

如果再次插入 userid 重复的文档时，MongoDB 将报错，以提示插入重复键，如：

db.user.insert({ "userid": 5 })

db.user.insert({ "userid": 5 })

E11000 duplicate key error index: user.user.$userid\_1 dup key: { : 5.0 }

如果插入的文档中不包含 userid 键，那么该文档中该键的值为 null，如果多次插入类似的文档，MongoDB 将会报出同样的错误，如：

db.user.insert({ "userid1": 5 })

db.user.insert({ "userid1": 5 })

E11000 duplicate key error index: user.user.$userid\_1 dup key: { : null }

如果在创建唯一索引时已经存在了重复项，我们可以通过下面的命令帮助我们在创建唯一索引时消除重复文档，仅保留发现的第一个文档，如：

先删除刚刚创建的唯一索引。

db.user.dropIndex({ "userid": 1 })

插入测试数据，以保证集合中有重复键存在。

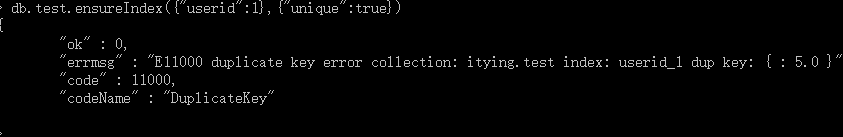
db.user.remove()

db.user.insert({ "userid": 5 })

db.user.insert({ "userid": 5 })

重新创建唯一索引

db.user.ensureIndex({ "userid": 1 },{ "unique": true })



我们同样可以创建复合唯一索引，即保证复合键值唯一即可。如：

db.user.ensureIndex({ "userid":1,"age": 1 },{ "unique": true })

* + 1. 索引的一些参数

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| background | Boolean | 创建索引过程会阻塞其他数据库操作，background可指定以后台方式创建索引，既增加"background"可选参数。"background"默认值为false |
| unique | Boolean | 建立的索引是否唯一。指定为true创建唯一索引。  默认值为false |
| name | string | 索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序生成一个索引名称 |
| dropDups | Boolean | 在建立唯一索引时是否删除重复记录，指定true创建唯一索引。  默认值为false |

如果在为已有数据的文档创建索引时，可以执行下面的命令，以使 MongoDB 在后台创建索引，这样的创建时就不会阻塞其他操作。但是相比而言，以阻塞方式创建索引，会使整个创建过程效率更高，但是在创建时 MongoDB 将无法接收其他的操作。

db.user.ensureIndex({ "username": 1 },{ "background": true })

* + 1. 使用 explain

explain 是非常有用的工具，会帮助你获得查询方面诸多有用的信息。只要对游标调用该方法，就可以得到查询细节。

explain 会返回一个文档，而不是游标本身。如：



explain 会返回查询使用的索引情况，耗时和扫描文档数的统计信息。

* + 1. explain executionStats 查询具体的执行时间

db.tablename.find().explain( "executionStats" )

关注输出的如下数值：explain.executionStats.executionTimeMillis

* 1. Nodejs 操作 MongoDb 数据库
     1. 在 Nodejs 中使用 Mongodb

前面的课程我们讲了用命令操作 MongoDB，这里我们看下如何用 nodejs 来操作数据库

需要引包：

npm install mongodb --save-dev

网址：

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

* + 1. Nodejs 创建服务

let express = require('express');

//数据库引用

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

let app = express();

//数据库连接的地址，最后的斜杠表示数据库名字

let shujukuURL = 'mongodb://localhost:27017/test';

app.listen(8020, () => {

console.log('listen server 8020');

});

* + 1. Nodejs 连接 MongoDb 数据库

app.get('/', (req, res) => {

//连接数据库，这是一个异步的操作

MongoClient.connect(dbURL, (err, db) => {

if (err) {

res.send('数据库连接失败')

throw err

}

res.send('连接成功')

res.end()

db.close()

})

})

* + 1. Nodejs 查找 MongoDb 数据库集合

app.get('/find', (req, res) => {

MongoClient.connect(dbURL, (err, db) => {

if (err) {

res.send('数据库查找失败')

throw err

}

const table = db.db('test')

const collection = table.collection('test')

collection.find({}).toArray((error, docs) => {

res.send(docs);

res.end();

db.close();

})

})

})

* + 1. Nodejs 给 MongoDb 增加数据

app.get('/add/:name', (req, res) => {

let param = req.params.name

//连接数据库，这是一个异步的操作

MongoClient.connect(dbURL, (err, db) => {

if (err) {

res.send('数据库连接失败')

throw err

}

const table = db.db('test')

const collection = table.collection('test')

collection.insertOne({

"name": param

}, (error, result) => {

let re = JSON.parse(result)

if (re.n === 1) {

res.send("插入成功。")

} else {

res.send("插入失败,error：" + error)

}

res.end()

db.close()

})

})

})

* + 1. Nodejs 修改 MongoDb 数据

app.get('/upd', (req, res) => {

let { name, age } = req.query

MongoClient.connect(dbURL, (err, db) => {

const table = db.db('test')

const collection = table.collection('test')

collection.updateOne({

"name": name

}, { $set: { "age": age } }, (error, result) => {

let re = JSON.parse(result);

if (re.n === 1) {

res.send("修改成功。");

} else {

res.send("修改失败,error：" + error);

}

res.end();

db.close();

})

})

})

* + 1. Nodejs 删除 MongoDb 数据

app.get('/del/:name', (req, res) => {

let param = req.params.name

//连接数据库，这是一个异步的操作

MongoClient.connect(dbURL, (err, db) => {

if (err) {

res.send('数据库连接失败')

throw err

}

const table = db.db('test')

const collection = table.collection('test')

collection.deleteOne({ "name": param }, (error, result) => {

let re = JSON.parse(result)

if (re.n === 1) {

res.send("删除成功。")

} else {

res.send("删除失败,error：" + error)

}

res.end()

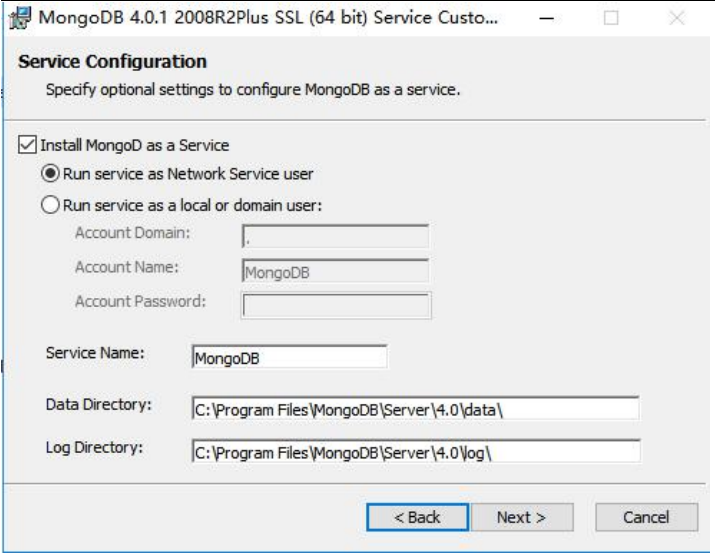
db.close()

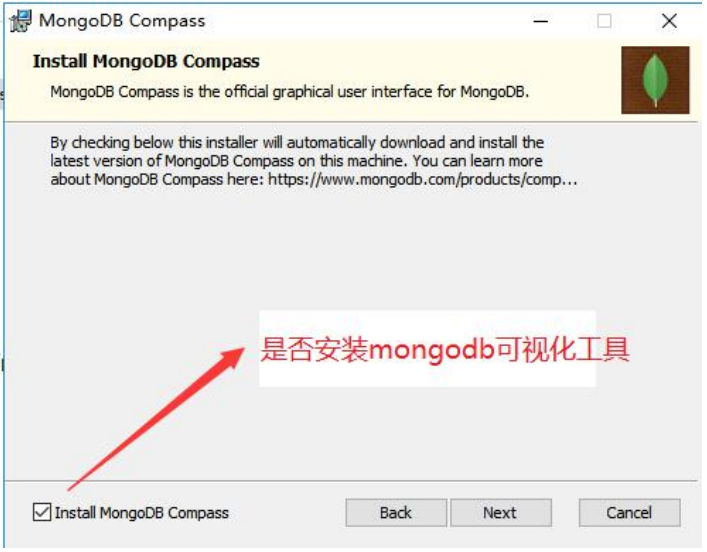
})

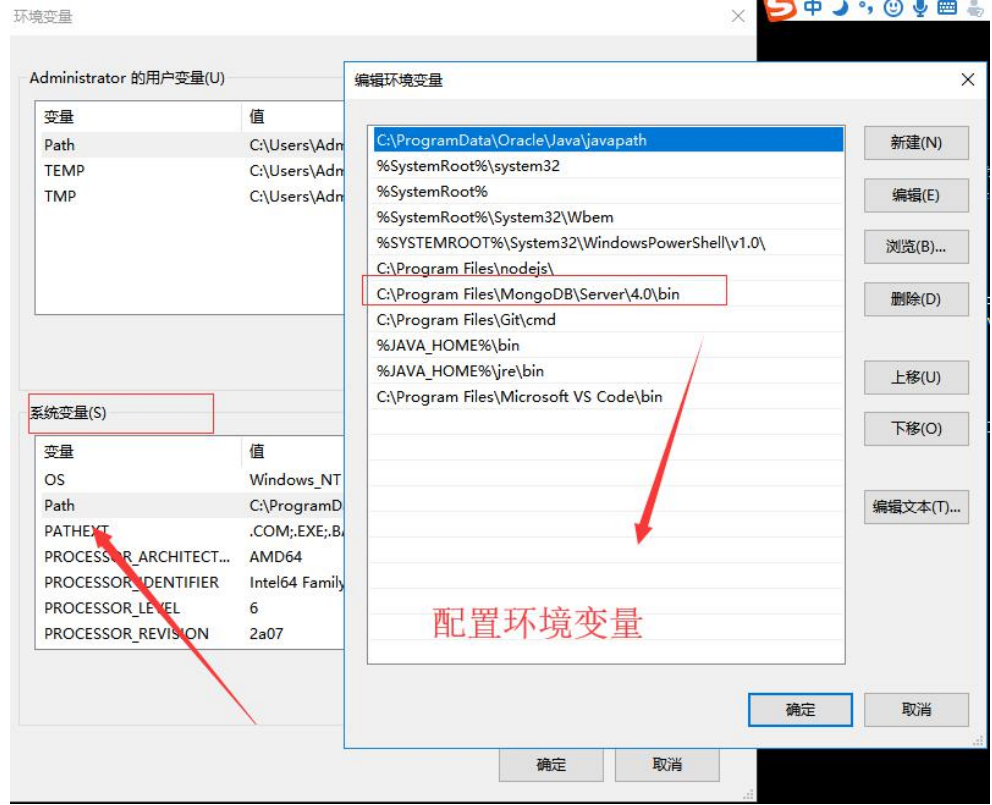
})

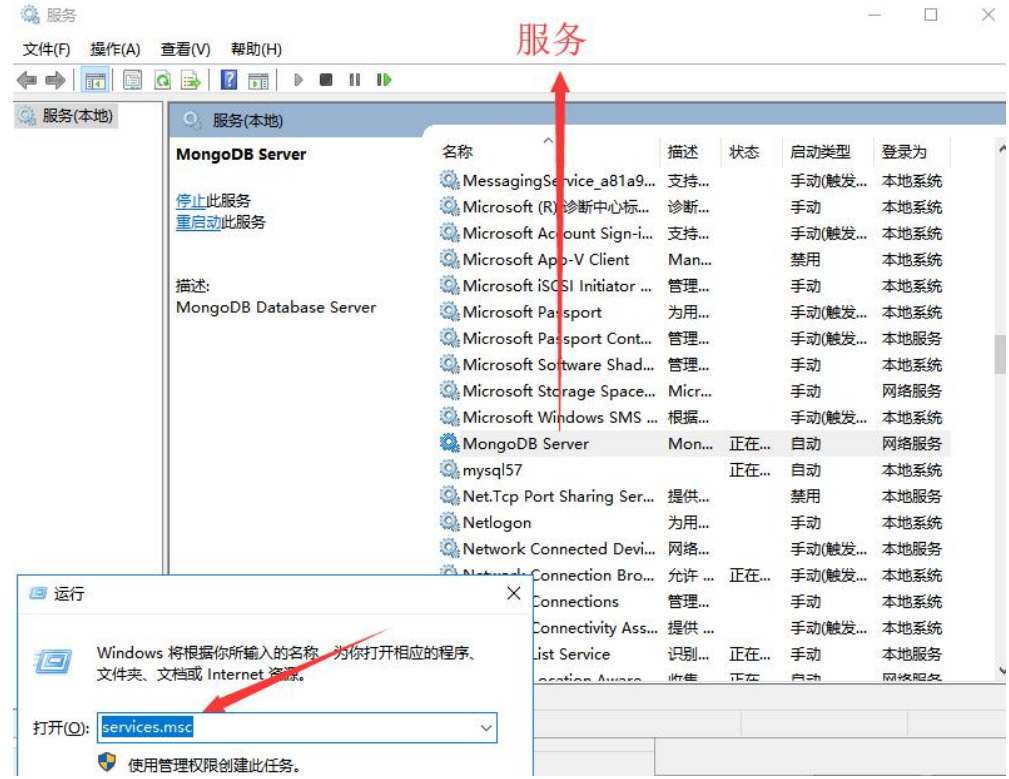
})

* 1. Mongodb4.x 的使用以及 Mongodb 账户权限配置
     1. Mongodb4.x 的使用











* + 1. Mongodb 账户权限配置（了解）
       1. 创建超级管理用户

use admin

db.createUser({

user: 'admin',

pwd: '123456',

roles: [{ role: 'root', db: 'admin' }]

})

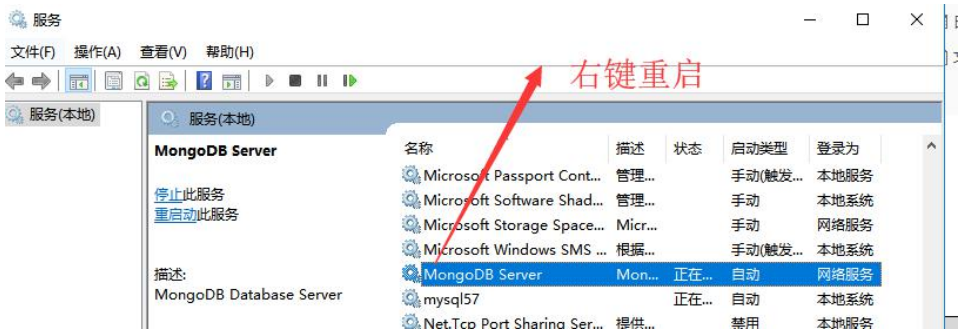
* + - 1. 修改 Mongodb 数据库配置文件

**路径：**C:\Program Files\MongoDB\Server\4.0\bin\mongod.cfg

**配置：**security:authorization: enabled

注：默认情况下不启用

* + - 1. 重启mongodb 服务



* + - 1. 用超级管理员账户连接数据库

mongo admin -u 用户名 -p 密码

mongo 192.168.1.200:27017/test -u user -p password

* + - 1. 给 eggcms 数据库创建一个用户 只能访问 eggcms 不能访问其他数据库

use eggcms

db.createUser({

user: "eggadmin",

pwd: "123456",

roles: [{ role: "dbOwner", db: "eggcms" }]

})

* + 1. Mongodb 账户权限配置中常用的命令

show users; #查看当前库下的用户

db.dropUser("eggadmin") #删除用户

db.updateUser( "admin",{pwd:"password"}); #修改用户密码

db.auth("admin","password"); #密码认证

* + 1. Mongodb 数据库角色
       1. 数据库用户角色

针对每一个数据库进行控制。

**read**：提供了读取所有非系统集合，以及系统集合中的system.indexes, system.js, system.namespaces

**readWrite**：包含了所有read权限，以及修改所有非系统集合的和系统集合中的system.js的权限。

* + - 1. 数据库管理角色

每一个数据库包含了下面的数据库管理角色。

**dbOwner**：该数据库的所有者，具有该数据库的全部权限。

**dbAdmin**：一些数据库对象的管理操作，但是没有数据库的读写权限。

**userAdmin**：为当前用户创建、修改用户和角色。拥有userAdmin权限的用户可以将该数据库的任意权限赋予任意的用户。

* + - 1. 集群管理角色

admin数据库包含了下面的角色，用户管理整个系统，而非单个数据库。这些权限包含了复制集和共享集群的管理函数。

**clusterAdmin**：提供了最大的集群管理功能。相当于clusterManager, clusterMonitor, and hostManager和dropDatabase的权限组合。

**clusterManager**：提供了集群和复制集管理和监控操作。拥有该权限的用户可以操作config和local数据库（即分片和复制功能）

**clusterMonitor**：仅仅监控集群和复制集。

**hostManager**：提供了监控和管理服务器的权限，包括shutdown节点，logrotate, repairDatabase等。

* + - 1. 备份恢复角色

admin数据库中包含了备份恢复数据的角色。

包括**backup**、**restore**等等。

* + - 1. 所有数据库角色

admin数据库提供了一个mongod实例中所有数据库的权限角色：

**readAnyDatabase**：具有read每一个数据库权限。但是不包括应用到集群中的数据库。

**readWriteAnyDatabase**：具有readWrite每一个数据库权限。但是不包括应用到集群中的数据库。

**userAdminAnyDatabase**：具有userAdmin每一个数据库权限，但是不包括应用到集群中的数据库。

**dbAdminAnyDatabase**：提供了dbAdmin每一个数据库权限，但是不包括应用到集群中的数据库。

* + - 1. 超级用户角色

**root**：dbadmin到admin数据库、useradmin到admin数据库以及UserAdminAnyDatabase。但它不具有备份恢复、直接操作system.\*集合的权限，但是拥有root权限的超级用户可以自己给自己赋予这些权限。

* + - 1. 内部角色

**\_\_system**

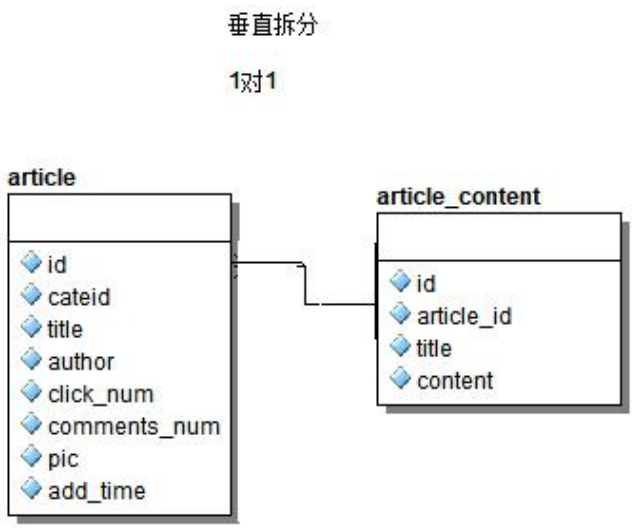
参考：https://www.cnblogs.com/zzw1787044/p/5773178.html

* + 1. 连接数据库的时候需要配置账户密码

const url = 'mongodb://admin:123456@localhost:27017/';

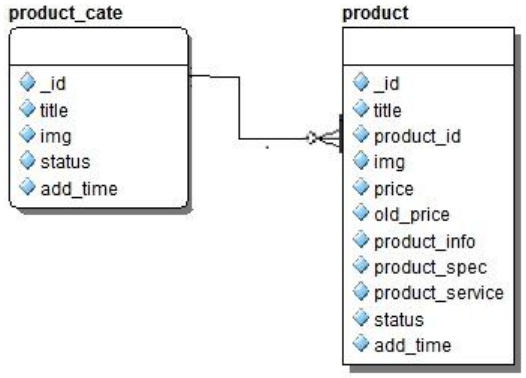
* 1. 关系型数据库表（集合）与表（集合）之间的几种关系
     1. 一对一的关系

例如：一个人对应一个唯一的身份证号，即为一对一的关系。



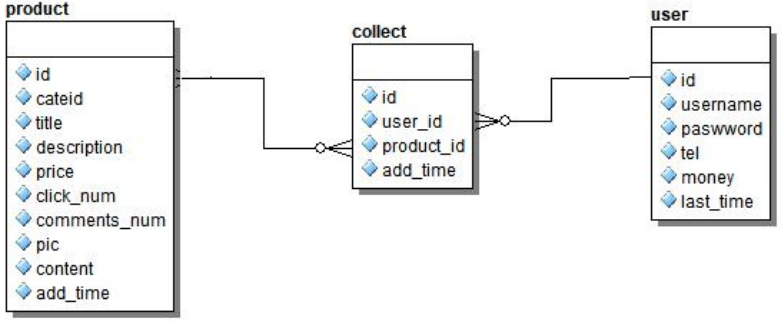
* + 1. 一对多的关系

例如：一个班级对应多名学生，一个学生只能属于一个班级，即为一对多关系。



* + 1. 多对多的关系

例如：一个学生可以选多门课程，而同一门课程可以被多个学生选修，彼此的对应关系即是多对多关系。



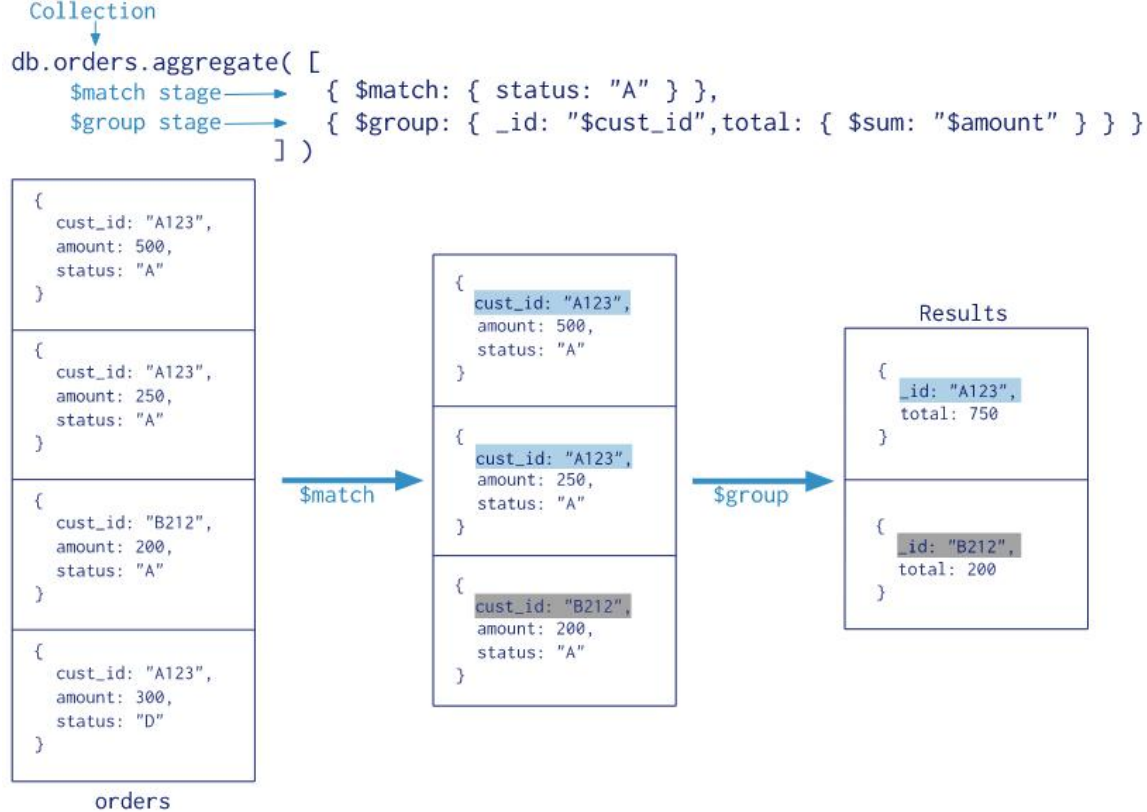
* 1. MongoDB 的高级查询 aggregate 聚合管道
     1. MongoDB 聚合管道（Aggregation Pipeline）

使用聚合管道可以对集合中的文档进行变换和组合。

实际项目：**表关联查询**、数据的统计。

MongoDB 中使用 **db.COLLECTION\_NAME.aggregate([{<stage>},...])** 方法

来构建和使用聚合管道。先看下官网给的实例，感受一下聚合管道的用法。



* + 1. MongoDB Aggregation 管道操作符与表达式

|  |  |
| --- | --- |
| **管道操作符** | **Description** |
| $project | 增加、删除、重命名字段 |
| $match | 条件匹配。只满足条件的文档才能进入下一阶段 |
| $limit | 限制结果的数量 |
| $skip | 跳过文档的数量 |
| $sort | 条件排序 |
| $group | 条件组合结果 统计 |
| $lookup | $lookup 操作符 用以引入其它集合的数据 （表关联查询） |
| $unwind | 将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值 |
| $geoNear | 输出接近某一地理位置的有序文档 |

SQL 和 NOSQL 对比：

|  |  |
| --- | --- |
| WHERE | $match |
| GROUP BY | $group |
| HAVING | $match |
| SELECT | $project |
| ORDER BY | $sort |
| LIMIT | $limit |
| SUM() | $sum |
| COUNT() | $sum |
| join | $lookup |

**管道表达式**：

管道操作符作为“键”，所对应的“值”叫做管道表达式。

例如{$match:{status:"A"}}，$match 称为管道操作符，而 status:"A"称为管道表达式，是管道操作符的操作数(Operand)。

每个管道表达式是一个文档结构，它是由**字段名、字段值、和一些表达式操作符**组成的。

MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

表达式：处理输入文档并输出。表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其它的文档。

|  |  |
| --- | --- |
| **常用表达式操作符** | **Description** |
| $addToSet | 将文档指定字段的值去重 |
| $max | 文档指定字段的最大值 |
| $min | 文档指定字段的最小值 |
| $sum | 文档指定字段求和 |
| $avg | 文档指定字段求平均 |
| $gt | 大于给定值 |
| $lt | 小于给定值 |
| $eq | 等于给定值 |
| $push | 在结果文档中插入值到一个数组中 |
| $first | 根据资源文档的排序获取第一个文档数据 |
| $last | 根据资源文档的排序获取最后一个文档数据 |

* + 1. 模拟数据

db.order.insert({"order\_id":"1","uid":10,"trade\_no":"111","all\_price":100,"all\_num":2})

db.order.insert({"order\_id":"2","uid":7,"trade\_no":"222","all\_price":90,"all\_num":2})

db.order.insert({"order\_id":"3","uid":9,"trade\_no":"333","all\_price":20,"all\_num":6})

db.order\_item.insert({"order\_id":"1","title":"商品鼠标 1","price":50,num:1})

db.order\_item.insert({"order\_id":"1","title":"商品键盘 2","price":50,num:1})

db.order\_item.insert({"order\_id":"1","title":"商品键盘 3","price":0,num:1})

db.order\_item.insert({"order\_id":"2","title":"牛奶","price":50,num:1})

db.order\_item.insert({"order\_id":"2","title":"酸奶","price":40,num:1})

db.order\_item.insert({"order\_id":"3","title":"矿泉水","price":2,num:5})

db.order\_item.insert({"order\_id":"3","title":"毛巾","price":10,num:1})

* + 1. $project

修改文档的结构，可以用来重命名、增加或删除文档中的字段。

查找指定字段。

要求查找 order 只返回文档中 trade\_no 和 all\_price 字段。

db.order.aggregate([

{ $project: { trade\_no: 1, all\_price: 1 } }

])

* + 1. $match

作用：用于过滤文档。用法类似于 find() 方法中的参数。

db.order.aggregate([

{ $match: { "all\_price": { $gte: 90 } } },

{ $project: { trade\_no: 1, all\_price: 1 } }

])

* + 1. $group

将集合中的文档进行分组，可用于统计结果。

统计每个订单的订单数量，按照订单号分组。

db.order\_item.aggregate([

{ $group: { \_id: "$order\_id", total: { $sum: "$num" } } }

])

* + 1. $sort

将集合中的文档进行排序。

db.order.aggregate([

{ $match: { "all\_price": { $gte: 90 } } },

{ $project: { trade\_no: 1, all\_price: 1 } },

{ $sort: { "all\_price": -1 } }

])

* + 1. $limit

db.order.aggregate([

{ $match: { "all\_price": { $gte: 90 } } },

{ $project: { trade\_no: 1, all\_price: 1 } },

{ $sort: { "all\_price": -1 } },

{ $limit: 1 }

])

* + 1. $skip

db.order.aggregate([

{ $match: { "all\_price": { $gte: 90 } } },

{ $project: { trade\_no: 1, all\_price: 1 } },

{ $sort: { "all\_price": -1 } },

{ $skip: 1 },

{ $limit: 1 }

])

* + 1. $lookup 表关联

db.order.aggregate([

{ $lookup: {

from: "order\_item",

localField: "order\_id",

foreignField: "order\_id",

as: "items"

} }

])

from：关联哪个表

localField：本地的属性

foreIgnField：关联表的属性

as：重命名

* 1. mongoose 入门以及 mongoose 实现数据的增、删、改、查
     1. mongoose 介绍

Mongoose 是在 node.js 异步环境下对 mongodb 进行便捷操作的对象模型工具。Mongoose是 NodeJS 的驱动，不能作为其他语言的驱动。

**Mongoose 有两个特点**

1. 通过关系型数据库的思想来设计非关系型数据库
2. 基于 mongodb 驱动，简化操作
   * 1. mongoose 的安装以及使用

官网：https://mongoosejs.com/

* + - 1. 安装

npm i mongoose--save

* + - 1. 引入 mongoose 并连接数据库

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/test');

如果有账户密码需要采用下面的连接方式：

mongoose.connect('mongodb://eggadmin:123456@localhost:27017/eggcms');

* + - 1. 定义 Schema

数据库中的 Schema，为数据库对象的集合。schema 是 mongoose 里会用到的一种数据模式，可以理解为表结构的定义；每个 schema 会映射到 mongodb 中的一个 collection，它不具备操作数据库的能力

var UserSchema = mongoose.Schema({

name: String,

age: Number,

status: 'number'

})

* + - 1. 创建数据模型

定义好了 Schema，接下就是生成 Model。model 是由 schema 生成的模型，可以对数据库的操作。

**注意**：mongoose.model 里面可以传入两个参数也可以传入三个参数

mongoose.model（参数 1:模型名称（首字母大写），参数 2:Schema）

mongoose.model（参数 1:模型名称（首字母大写），参数 2:Schema，参数 3:数据库集合名称）

**传入 2 个参数**：这个模型会和模型名称相同的复数的数据库建立连接。

如通过下面方法创建模型，那么这个模型将会操作 users 这个集合。

**传入 3 个参数**：模型默认操作第三个参数定义的集合名称

var User = mongoose.model('User', UserSchema);

* + - 1. 查找数据

User.find({}, function (err, docs) {

if (err) {

console.log(err);

return;

}

console.log(docs);

})

* + - 1. 增加数据

var u = new User({ //实例化模型 传入增加的数据

name: 'lisi2222',

age: 20,

status: true

})

u.save();

* + - 1. 修改数据

User.updateOne({ name: 'lisi2222' }, { name: '哈哈哈' }, function (err, res) {

if (err) {

console.log(err);

return;

}

console.log('成功')

});

* + - 1. 删除数据

User.deleteOne({ \_id: '5b72ada84e284f0acc8d318a' }, function (err) {

if (err) {

console.log(err);

return;

}

// deleted at most one tank document

console.log('成功');

});

* + - 1. 保存成功查找

var u = new User({

name: 'lisi2222333',

age: 20,

status: true //类型转换

})

u.save(function (err, docs) {

if (err) {

console.log(err);

return;

}

//console.log(docs);

User.find({}, function (err, docs) {

if (err) {

console.log(err);

return;

}

console.log(docs);

})

})

* + - 1. 模块化

mongoose.connect('mongodb://localhost:27017/test', {

useUnifiedTopology: true,

useNewUrlParser: true

}, (err) => {

if (err) {

console.error('数据库连接失败:', err)

} else {

console.log('数据库连接成功')

}

});

mongoose.connect(数据库地址，相关参数，回调)

useUnifiedTopology / useNewUrlParser这个属性会在url里识别验证用户所需的db,未升级前是不需要指定的,升级到一定要指定。

const TestSchema = mongoose.Schema({

name: String,

age: Number,

status: {

type: Number,

default: 0

}

})

可设置默认值；

在schema中没有的属性不会添加至数据库中。

* + 1. mongoose 模块化

见06\_module和07\_module.js。

* 1. Mongoose 预定义模式修饰符 Getters与 Setters 自定义修饰符
     1. mongoose 预定义模式修饰符

lowercase 转化为小写

uppercase 转化为大写

trim 去首尾空格

mongoose 提供的预定义模式修饰符，可以对我们增加的数据进行一些格式化。

var UserSchema = mongoose.Schema({

name: {

type: String,

trim: true

},

age: Number,

status: {

type: Number,

default: 1

}

})

* + 1. mongoose 的安装以及使用

除了 mongoose 内置的修饰符以外，我们还可以通过 set（建议使用） 修饰符在增加数据的

时候对数据进行格式化。也可以通过 get（不建议使用）在实例获取数据的时候（不是数据库的数据）对数据进行格式化。

* + - 1. set方法

var NewsSchema = mongoose.Schema({

title: "string",

author: String,

pic: String,

redirect: {

type: String,

set(url) {

if (!url) return url;

if (url.indexOf('http://') != 0 && url.indexOf('https://') != 0) {

url = 'http://' + url;

}

return url;

}

},

content: String,

status: {

type: Number,

default: 1

}

})

在获得数据后，存入数据库前，在schema中，对数据进行设置。

* + - 1. get方法（没用）

var NewsSchema = mongoose.Schema({

title: "string",

author: String,

pic: String,

redirect: {

type: String,

set(url) {

if (!url) return url;

if (url.indexOf('http://') != 0 && url.indexOf('https://') != 0) {

url = 'http://' + url;

}

return url;

},

get: function (url) {

if (!url) return url;

if (url.indexOf('http://') != 0 && url.indexOf('https://') != 0) {

url = 'http://' + url;

}

return url;

}

},

content: String,

status: {

type: Number,

default: 1

}

})



只有在写入模板后，调取有get方法的字段时，才会触发get方法，否则不触发；

在new模板时，先执行默认执行set方法，之后由于console.log触发get方法，已无法对param中的数据造成影响，将直接通过save方法存入数据库中。

get方法貌似没用，可能适用于动态变更schema的。

* 1. Mongoose 索引
     1. Mongoose 索引

索引是对数据库表中一列或多列的值进行排序的一种结构，可以让我们查询数据库变得更快。MongoDB 的索引几乎与传统的关系型数据库一模一样，这其中也包括一些基本的查询优化技巧。

mongoose 中除了以前创建索引的方式，我们也可以在定义 Schema 的时候指定创建索引。

var DeviceSchema = new mongoose.Schema({

sn: {

type: Number, // 唯一索引

unique: true

},

name: {

type: String, // 普通索引

index: true

}

});

* + 1. 索引基础

索引是对数据库表中一列或多列的值进行排序的一种结构，可以让我们查询数据库变得更快。MongoDB 的索引几乎与传统的关系型数据库一模一样，这其中也包括一些基本的查询优化技巧。

**创建索引的命令：**

db.user.ensureIndex({ "username": 1 })

**获取当前集合的索引：**

db.user.getIndexes()

**删除索引的命令是：**

db.user.dropIndex({ "username": 1 })

在 MongoDB 中，我们同样可以创建复合索引，如：

数字 1 表示 username 键的索引按升序存储，-1 表示 age 键的索引按照降序方式存储。

db.user.ensureIndex({ "username": 1, "age": -1 })

该索引被创建后，基于 username 和 age 的查询将会用到该索引，或者是基于 username的查询也会用到该索引，但是只是基于 age 的查询将不会用到该复合索引。因此可以说，如果想用到复合索引，必须在查询条件中包含复合索引中的前 N 个索引列。然而如果查询条件中的键值顺序和复合索引中的创建顺序不一致的话，MongoDB 可以智能的帮助我们调整该顺序，以便使复合索引可以为查询所用。如：

db.user.find({ "age": 30, "username": "stephen" })

对于上面示例中的查询条件，MongoDB 在检索之前将会动态的调整查询条件文档的顺序，以使该查询可以用到刚刚创建的复合索引。

对于上面创建的索引，MongoDB 都会根据索引的 keyname 和索引方向为新创建的索引自动分配一个索引名，下面的命令可以在创建索引时为其指定索引名，如：

db.user.ensureIndex({ "username": 1 }, { "name": "userindex" })

随着集合的增长，需要针对查询中大量的排序做索引。如果没有对索引的键调用 sort，MongoDB 需要将所有数据提取到内存并排序。因此在做无索引排序时，如果数据量过大以致无法在内存中进行排序，此时 MongoDB 将会报错。

* + 1. 唯一索引

在缺省情况下创建的索引均不是唯一索引。下面的示例将创建唯一索引，如：

db.user.ensureIndex({ "userid": 1 }, { "unique": true })

如果再次插入 userid 重复的文档时，MongoDB 将报错，以提示插入重复键，如：

db.user.insert({ "userid": 5 })

db.user.insert({ "userid": 5 })

E11000 duplicate key error index: user.user.$userid\_1 dup key: { : 5.0 }

如果插入的文档中不包含 userid 键，那么该文档中该键的值为 null，如果多次插入类似的文档，MongoDB 将会报出同样的错误，如：

db.user.insert({ "userid1": 5 })

db.user.insert({ "userid1": 5 })

E11000 duplicate key error index: user.user.$userid\_1 dup key: { : null }

如果在创建唯一索引时已经存在了重复项，我们可以通过下面的命令帮助我们在创建唯一索引时消除重复文档，仅保留发现的第一个文档，如：

先删除刚刚创建的唯一索引。

db.user.dropIndex({ "userid": 1 })

插入测试数据，以保证集合中有重复键存在。

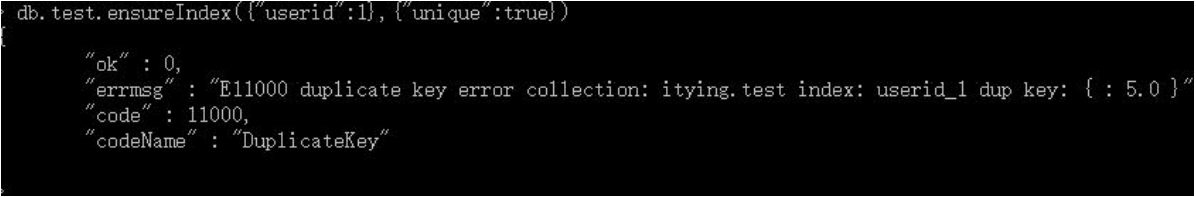
db.user.remove()

db.user.insert({ "userid": 5 })

db.user.insert({ "userid": 5 })

重新创建唯一索引

db.user.ensureIndex({ "userid": 1 }, { "unique": true })



我们同样可以创建复合唯一索引，即保证复合键值唯一即可。如：

db.user.ensureIndex({ "userid": 1, "age": 1 }, { "unique": true })

* + 1. 索引的一些参数



如果在为已有数据的文档创建索引时，可以执行下面的命令，以使 MongoDB 在后台创建索引，这样的创建时就不会阻塞其他操作。但是相比而言，以阻塞方式创建索引，会使整个创建过程效率更高，但是在创建时 MongoDB 将无法接收其他的操作。

db.user.ensureIndex({ "username": 1 }, { "background": true })

* + 1. 使用 explain

explain 是非常有用的工具，会帮助你获得查询方面诸多有用的信息。只要对游标调用该方法，就可以得到查询细节。explain 会返回一个文档，而不是游标本身。如：



explain 会返回查询使用的索引情况，耗时和扫描文档数的统计信息。

* + 1. explain executionStats 查询具体的执行时间

db.tablename.find().explain("executionStats")

关注输出的如下数值：explain.executionStats.executionTimeMillis

* 1. Mongoose 内置 CURD 方法、扩展 Mongoose Model 的静态方法和实例方法
     1. Mongoose 内置 CURD

https://mongoosejs.com/docs/queries.html

Model.deleteMany()

Model.deleteOne()

Model.find()

Model.findById()

Model.findByIdAndDelete()

Model.findByIdAndRemove()

Model.findByIdAndUpdate()

Model.findOne()

Model.findOneAndDelete()

Model.findOneAndRemove()

Model.findOneAndUpdate()

Model.replaceOne()

Model.updateMany()

Model.updateOne()

* + 1. 扩展 Mongoose CURD 方法

var mongoose = require('./db.js');

var UserSchema = mongoose.Schema({

name: {

type: String

},

age: Number,

status: {

type: Number,

default: 1

}

})

// 静态方法

UserSchema.statics.findByUid = function (uid, cb) {

this.find({

"\_id": uid

}, function (err, docs) {

cb(err, docs)

})

}

// 实例方法

UserSchema.methods.print = function () {

console.log('这是一个实例方法');

console.log(this);

};

module.exports = mongoose.model('User', UserSchema, 'user');

见11\_function.js

实例方法为实例化后的方法，基本上就是new后可打点访问的方法。

注：多使用**静态方法**，实例方法很少用。

* 1. Mongoose 数据校验
     1. Mongoose 校验参数

mongoose数据校验：用户通过mongoose给mongodb数据库增加数据的时候，对数据的合法性进行的验证；

mongoose里面定义Schema:字段类型，修饰符、默认参数 、数据校验都是为了数据库数据的一致性；

Schema，为数据库对象的集合,每个schema会映射到mongodb中的一个collection,定义Schema可以理解为表结构的定义。

required：表示这个数据必须传入

max：用于 Number 类型数据，最大值，**number类型**

min：用于 Number 类型数据，最小值，**number类型**

enum：枚举类型，要求数据必须满足枚举值 enum: ['0', '1', '2']，**string类型**

match：增加的数据必须符合 match（正则）的规则

maxlength：最大值

minlength：最小值

validate：自定义验证（如果验证器函数返回undefined或true值，则验证成功。如果返回false（除undefined）或引发错误，则验证失败）

var UserSchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

age: {

type: Number, // 是否必须的校验器

required: true, // 数字类型的最大值校验器

max: 120, // 数字类型的最小值校验器

min: 0

},

status: {

type: String, // 设置字符串的可选值

enum: ['0', '1', '2']

},

phone: {

type: Number,

match: /^\d{11}$/

},

desc: {

type: String,

maxlength: 20,

minlength: 10

}

});

* + 1. Mongoose 自定义的验证器

在缺省情况下创建的索引均不是唯一索引。下面的示例将创建唯一索引，如：

var UserSchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

age: {

type: Number, // 是否必须的校验器

required: true, // 数字类型的最大值校验器

max: 120, // 数字类型的最小值校验器

min: 0

},

status: {

type: String, // 设置字符串的可选值

enum: ['0', '1', '2']

},

phone: {

type: Number,

match: /^\d{11}$/

},

desc: {

type: String, // 自定义的验证器，如果通过验证返回 true，没有通过则返回 false

validate: function (desc) {

return desc.length >= 10;

}

}

});

* 1. Mongoose 中使用 aggregate 聚合管道
     1. MongoDB 聚合管道（Aggregation Pipeline）

见第七章。

* + 1. Mongoose aggregate 多表关联查询

见12\_aggregate1.js

筛选价格大于90的。

const OrderModal = require('./06\_module/order.js')

OrderModal.aggregate(

[

{ $lookup: {

from: "order\_item",

localField: "order\_id",

foreignField: "order\_id",

as: "items"

} },

{ $match: {

"all\_price": { $gte: 90 }

} }

], (err, res) => {

if (err) {

console.log(err)

return

}

console.log(JSON.parse(JSON.stringify(res)))

})

见12\_aggregate2.js

筛选商品为酸奶的。

const mongoose = require('./06\_module/db.js')

const OrderItmeModal = require('./06\_module/order\_item')

OrderItmeModal.aggregate([{

$lookup: {

from: "order",

localField: "order\_id",

foreignField: "order\_id",

as: "order\_info"

}

}, {

$match: {

// \_id: mongoose.Types.ObjectId("5b743da92c327f8d1b360546")

title: '酸奶'

}

}], (err, res) => {

if (err) {

console.log(err)

return

}

console.log(JSON.parse(JSON.stringify(res)))

})

mongoose中获取ObjectId mongoose.Types.ObjectId

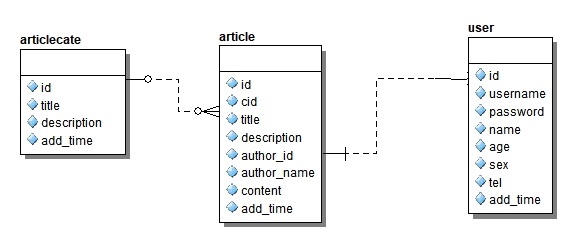
* 1. Mongoose 中使用 populate 实现关联查询(不推荐)
     1. Mongoose populate 官方文档

https://mongoosejs.com/docs/populate.html

相当于MongoDB的$lookup。

建议使用MongoDB的**aggregate**。

* + 1. Mongoose populate 关联查询



* + - 1. 定义 ref(表示外键)

var ArticleSchema = new Schema({

title: {

type: String,

unique: true

},

cid: {

type: Schema.Types.ObjectId,

ref: 'ArticleCate' //model 的名称

},

/\*分类 id\*/

author\_id: {

type: Schema.Types.ObjectId,

ref: 'User'

},

/\*用户的 id\*/

author\_name: {

type: String

},

descripton: String,

content: String

});

* + - 1. 关联查询

ArticleModel.find({}).populate('cid').populate('author\_id').exec(function (err, docs) {

console.log(docs)

})

* 1. MongoDb 数据库的导出导入

在 Mongodb 中我们使用 mongodump 命令来备份 MongoDB 数据。该命令可以导出所有数据到指定目录中。mongodump 命令可以通过参数指定导出的数据量级转存的服务器。使用mongorestore 命令来恢复备份的数据。

导出：

mongodump - h dbhost - d dbname - o dbdirectory

dbhost：IP地址及端口号，默认27017；

dbname：数据库名称；

dbdirectory：导出路径

导入：

mongorestore - h dbhost - d dbname path

dbhost：IP地址及端口号，默认27017；

path：导入路径