# Mongodb

1. **介绍**

Mongodb是一个基于分布式文件存储的数据库，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，最像关系型数据库的。支持对数据建立索引，由C++语言编写。

1. **NoSql数据库分类**

* **键值（key-value）存储数据库**

Redis

* **文档性数据库**

它同第一种键值存储相似。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格式存储，如JSON。文档型数据库可以看作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：Mongodb

* **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。如：HBase

* 图形（Graph）数据库

1. **Mongodb与关系型数据库对比**

|  |  |  |
| --- | --- | --- |
| **Sql术语概念** | **Mondodb术语概念** | **解释说明** |
| Database | Database | 数据库 |
| Table | Collection | 表/集合 |
| Row | Document | 行/文档 |
| Column | Field | 数据字段/域 |
| Index | Index | 索引 |
| Table joins |  | 表链接.mongodb不支持 |
| Primary key | Primary key | 主键 mongodb自动将\_id设为主键 |

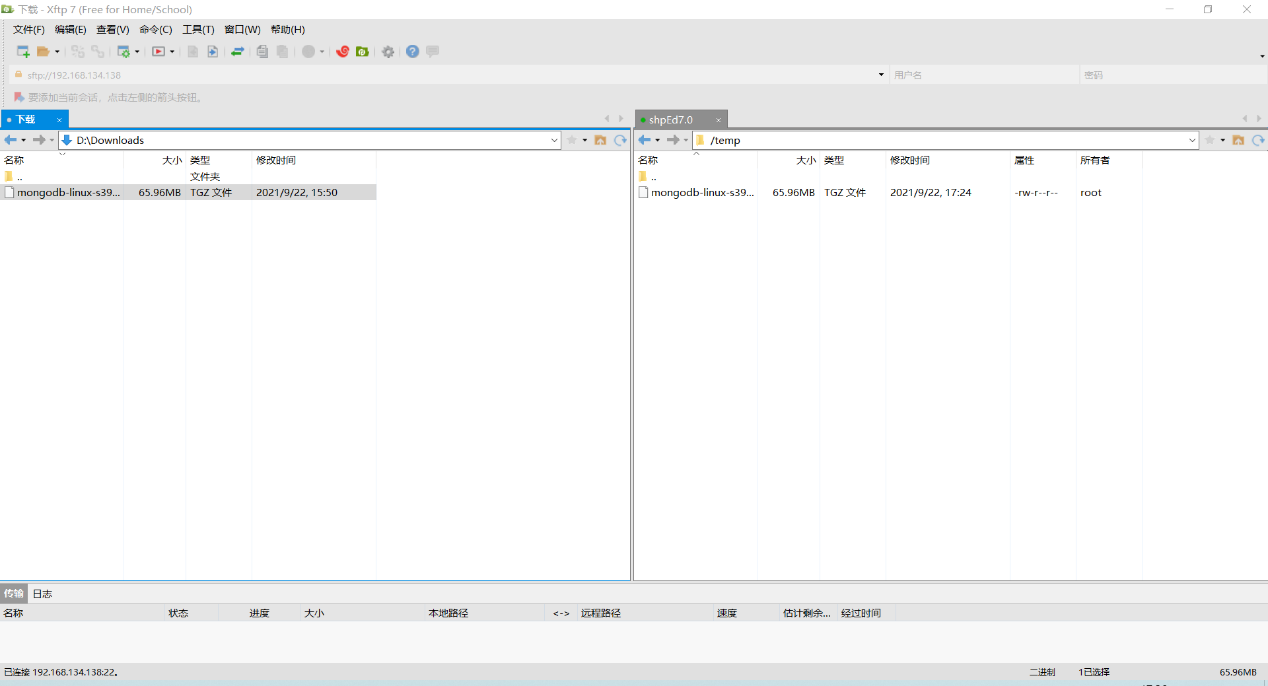
|  |  |
| --- | --- |
| **数据库服务和客户端** | |
| Mysql/Oracle | Mongod（服务端） |
| Mysql/sqlplus | Mongo（客户端） |

1. **Mongodb数据类型**

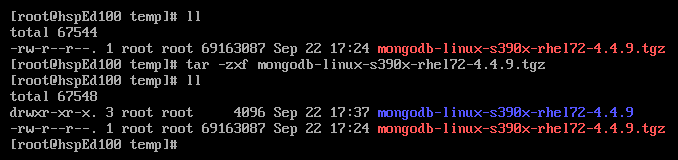
* **String：**
* **Integer：**
* **Boolean**
* **Double：**
* **Min/Max keys：**将一个值与BSON（二进制的JSON）元素的最低值和最高值相对比
* **Array：**
* **Timestamp：**时间戳
* **Object：**用于内嵌文档
* **Null：**
* **Symbol：**符号。该类型基本等同于字符串类型，不同的是，它一般用于采用特殊符号类型的语言
* **Date：**
* **Object ID：**对象ID，用于创建文档ID
* **Binary Data：**二进制数据。
* **Code：**代码类型。用于在文档中存库js代码
* **Regular expression：**正则表单式类型

1. **Mongodb下载安装**

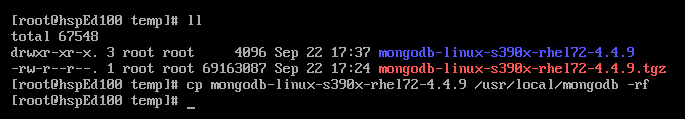
* **上传安装包到linux**

****

* **解压**

****

* **Copy解压后的文件到/usr/local**

****

* **创建数据库目录**

Mongodb的数据存储在data目录的db目录下，这个目录在安装过程中不会自动创建，需要手动创建data目录，并在data目录下创建db目录，data目录可以创在任何目录。



1. **启动与关闭**

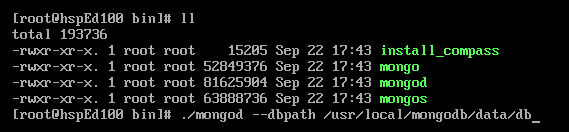
Mongodb的启动分为两种，无论哪种方式都需要执行bin目录中的mongod命令。启动时默认的查找数据库的路径为/data/db,如果数据库路径有变化，需要在该命令中通过—dbpath参数指定db目录的路径

* **启动参数**
* **--quiet：**安静输出
* **--port：**指定端口，默认27017
* **--bind：**绑定服务ip，若绑定127.0.0.1，则只能本机访问
* **--logpath：**指定日志文件
* **--dbpath：**指定db目录
* **--logappend：**使用追加的方式写日志
* **--fork：**守护进程的方式执行启动mongodb
* **--auth：**启用验证
* **--config：**指定配置文件路径
* **--joumal：**启用日志选项，mongodb的数据库操作将会写入到joumal文件夹的文件里
* **前置启动**

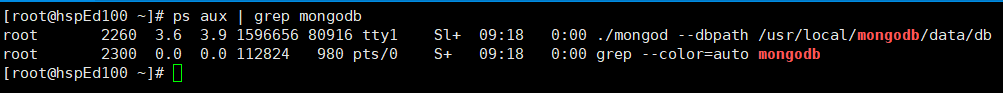
Mongodb默认启动为前置启动，前置启动就是mongodb启动会占用当前终端窗口

* **启动**

进入mongodb的bin目录，执行mongo命令。由于db目录不在linux的根目录，所以在执行该命令时需要通过—dbpath参数指定db路径；默认监听端口为27017



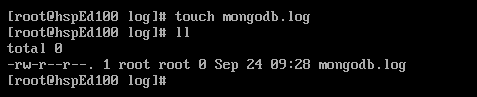




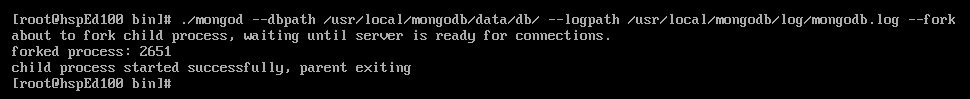
* **后置启动**

后置启动就是以守护进程的方式启动**mongodb**。需要在执行**mongodb**命令中添加—fork参数。需要注意的是，**--fork**参数需要配合着**—logpath**或者**—syslog**参数使用。**Mongodb**的日志文件可以在任何位置；以下在**mongodb**目录下创建**log**目录

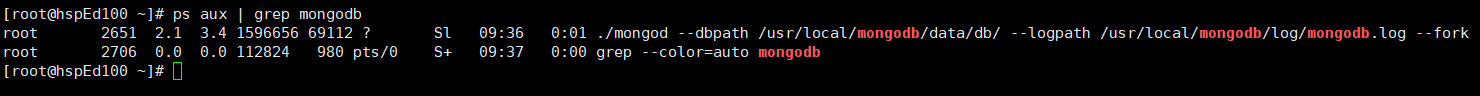
* **创建日志文件**



* **启动**



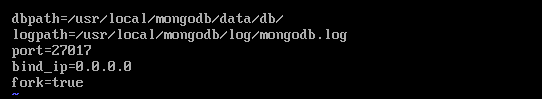
* **查看mongodb进程**



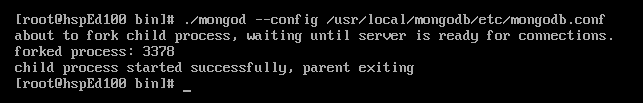
* **通过配置文件加载启动参数进行启动**

配置文件可以在任何目录下，配置文件的扩展名应为**.conf，**配置文件中使用**key=value**结构，在执行时通过**—confg**参数指定需要加载的配置文件；以下在**mongodb**目录中创建个**etc**目录并在该目录创建**mongodb.conf**文件

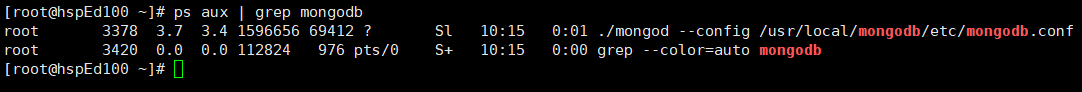
* **编辑配置文件**



* **启动**



* **查看进程**



* **关闭mongodb服务**
* **Ctrl + C关闭**

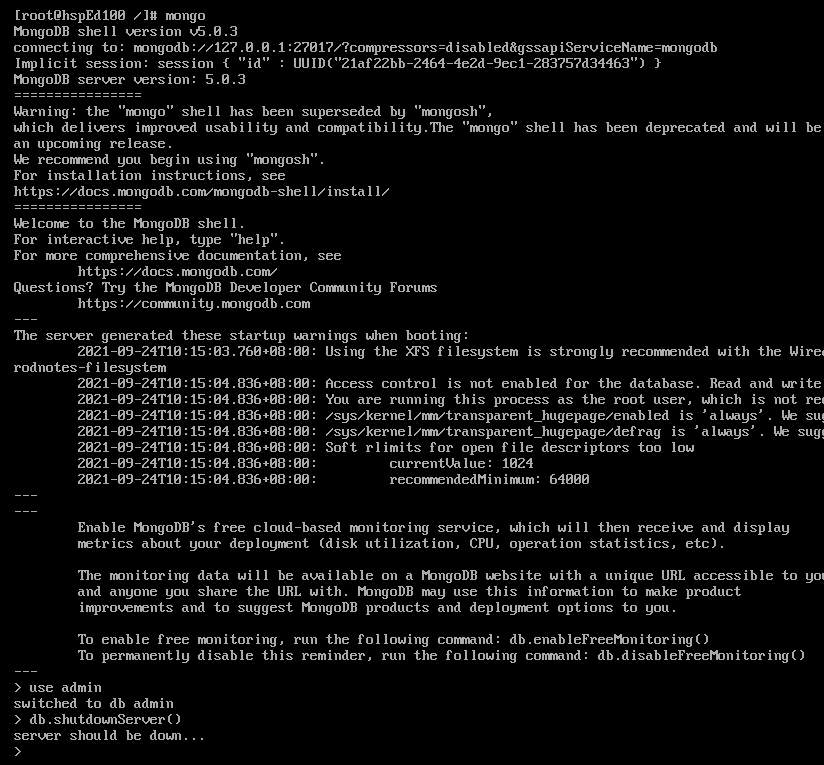
如果当前启动方式为前置启动，那么直接使用快捷键**ctrl+c**就可以关闭。这种方式关闭会等待当前进行中的操作完成，所以是安全的关闭方式

* **Kill命令关闭**

通过**kill**结束**mongodb**进程，然后删除**data/db**目录中的**mongodb.lock**文件，否则下次无法启动，此方法不建议使用，因为会造成数据损坏。

* **使用mongodb的函数关闭**
* **db.shutdownServer()**
* **db.runCommand(“shutdown”)**

如上两个方法都需要在**admin**库中执行，并且都是安全的关闭方式

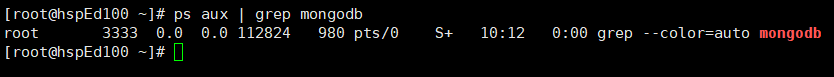


* **使用mongod命令关闭**

****

此关闭也是安全性的关闭方式

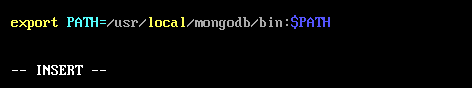
* **查看进程**

****

1. **配置环境变量**

为了可以在任何目录执行bin目录中的命令，我们可以将**bin**目录配置到环境变量中。修改**/ect/profile**文件，添加**export PATH=/usr/local/mongodb/bin:$PATH。**

* **配置环境变量**

****

* **重新加载配置文件**

****

* **测试是否配置成功**

****

1. **用户与权限**

* **用户权限列表**

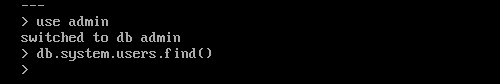
|  |  |
| --- | --- |
| **Read** | 允许读取指定数据库 |
| **readWrite** | 允许读写指定数据库 |
| **dbAdmin** | 允许在指定数据库中执行管理函数；如创建索引、删除 |
| **userAdmin** | 允许用户向system.users集合写入，可以找指定数据库里创建、删除和管理用户 |
| **clusterAdmin** | 只在admin数据库中可用，赋予用户所有分片和复制集相关函数的管理权限 |
| **readAnyDatabase** | 只在admin数据库中可用，赋予用户所有数据库的读权限 |
| **readWriteAnyDatabase** | 只在admin数据库中可用，赋予用户所有数据库的读写权限 |
| **userAdminAnyDatabase** | 只在admin数据库中可用，赋予用户所有数据库的userAdmin权限 |
| **dbAdminAnyDataBase** | 只在admin数据库中可用，赋予用户所有数据库的dbAdmin权限 |
| **root** | 只在admin数据库中可用，超级权限 |

* **创建DB管理用户**

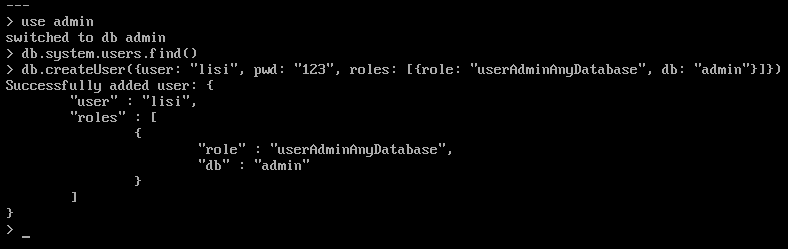
Mongodb有一个用户管理机制，简单描述为，有一个管理用户组，这个组的用户是专门为管理普通用户而设的，暂且称为管理员。

管理员通常没有数据库的读写权限，只有操作用户的权限，因此只需要赋予管理员userAdminAnyDatabase角色即可

* **查看admin库中的所有用户·**



* **创建用户**



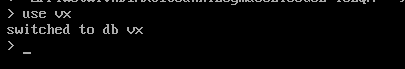
* **创建普通用户**

普通用户由管理员创建，通常需要指定操作某个数据库

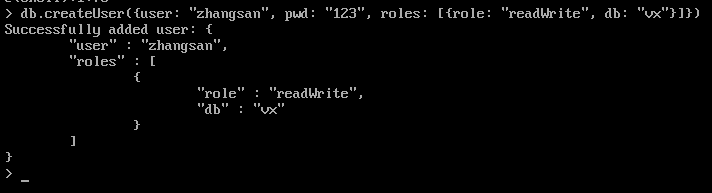
* **案例**

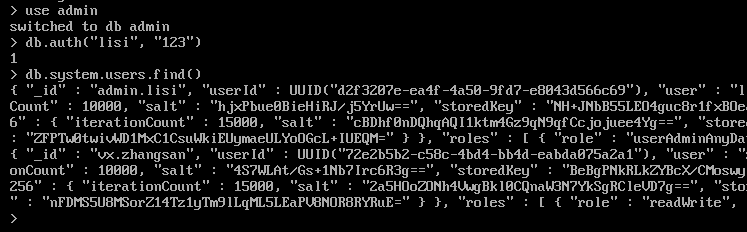
登录到管理员用户，创建一个**vx**数据库，并创建一个名为**zhangsan**的普通用户，指定数据库为**vx**

* **创建数据库**

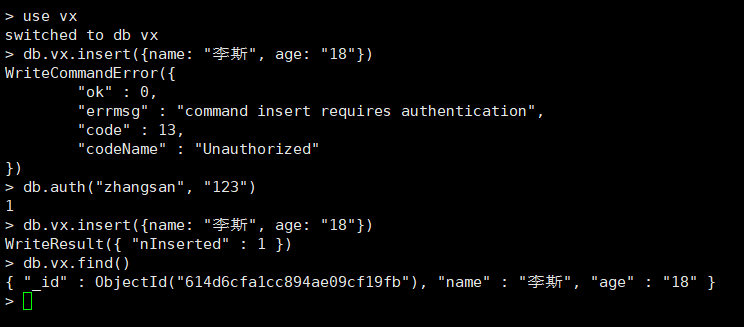
****

* **创建普通用户**

****

**·**

* **插入和查询数据**

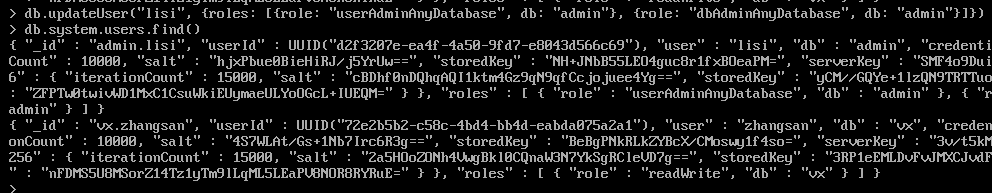
****

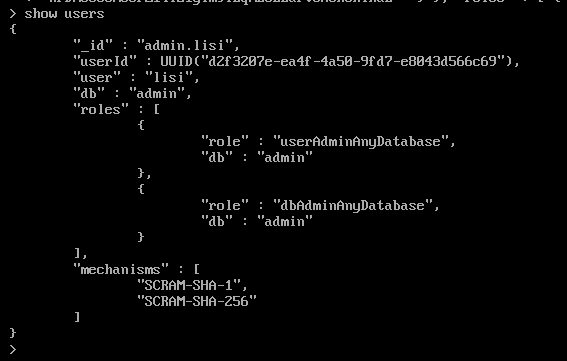
* **更新用户角色**

如果需要对已存在的用户角色做修改，那么可以使用**db.updateUser()**函数来更新用户角色。注意：该函数需要当前用户具有**userAdminAnyDatabase**或更高的权限。

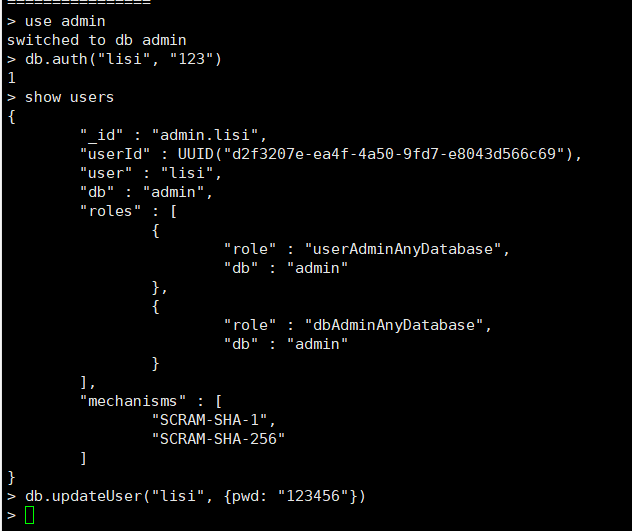
* **案例**

当前**lisi**用户具备**userAdminAnyDatabase**用户管理角色，为该用户添加一个**dbAdminAnyDatabase**的数据库管理角色。

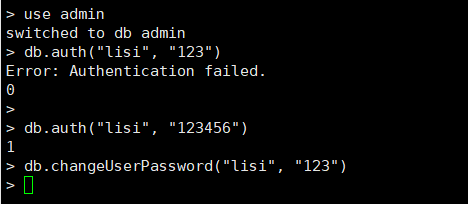




* **更新用户密码**
* **db.updateUser()函数更改**

****

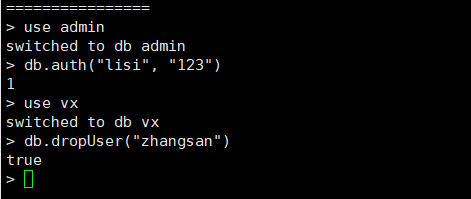
* **db.changeUserPassword()函数更改**

****

注意：更新某个用户的密码，首先要切换到当前用户所在的db库

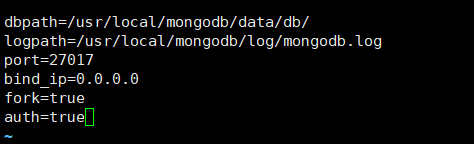
* **删除用户**

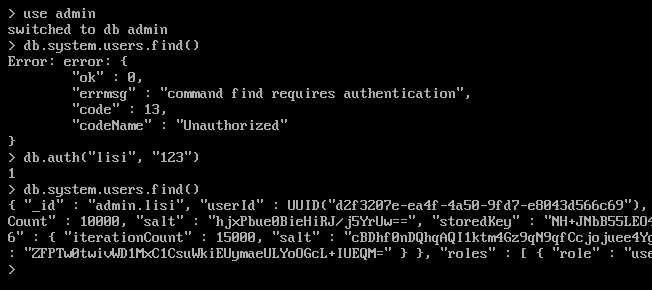
通过**db.dropUser()**函数可删除指定用户即可。需要具有**userAdminAnyDatabase**角色管理员权限才可以删除



1. **使用权限方式启动mongodb**

在默认的情况下**mongodb**是不开启用户认证的。如果我们添加用户，那么需要开启用户认证机制，通过修改**mongodb.conf**配置文件，将**auto**值设置为**true**即可

****

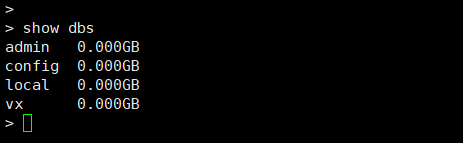
****

1. **数据库管理**

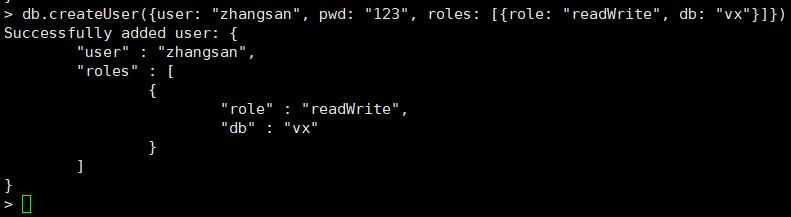
* **查看所有数据库**

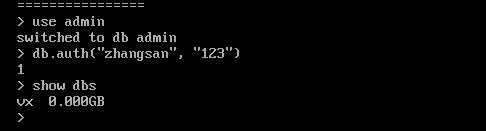
如果开启了用户认证，则需要先登录方可查看结果，否则不显示任何认证信息





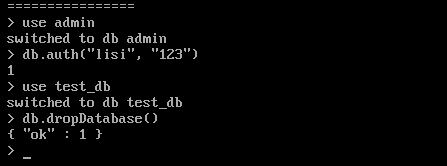
注意：**show dbs**只显示含有信息的或使用的数据库；如果使用的是普通用户登录的那么只能查询到该用户所拥有的数据库





* **删除数据库**

删除数据库使用**db.dropDatabase()**，在删除数据库之前，需要具备**dbAdminAnyDatabase**角色的管理员用户登录，然后切换到需要删除的数据库，执行**db.dropDatabase()**函数即可，删除成功后返回一个{ok：1}的字符串。



1. **集合**

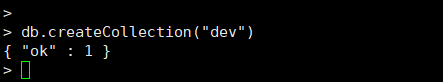
* **创建集合**

**db.createCollection(name, option);**

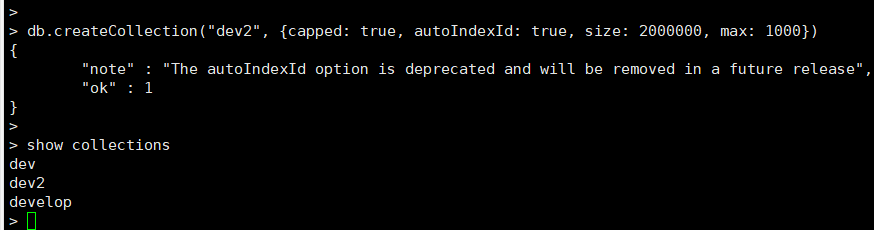
* **name:** 集合名
* **option：**可选参数
* **capped（布尔）：**可选，如果为**true**，则创建固定集合。固定集合是指有着固定大小的集合，当达到最大值时，它会自动覆盖最早的文档。当该值为**true**，必须指定**size**参数。
* **autoindexid（布尔）：**可选，如果为**true**，自动在**\_id**字段创建索引。默认为**false**。
* **size（数值）：**可选，为固定集合指定一个最大值，如果**capped**为**true**，也需要指定该字段
* **max（数值）：**可选，指定固定集合中包含文档的最大数量。

在插入文档时，**mongodb**首先检查固定集合的size字段，然后检查**max**字段

* **创建无参集合**



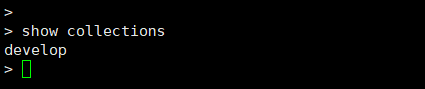
* **创建带参集合**

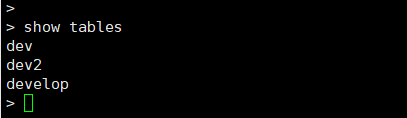


* **默认集合**

在**mongodb**中，也可以不用创建集合，当插入数据时，会自动创建集合，并且会使用数据库的名字作为集合的名称。

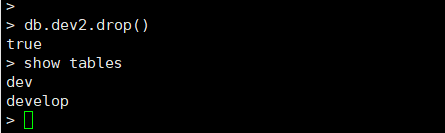
* **查看集合**

****

****

* **删除集合**

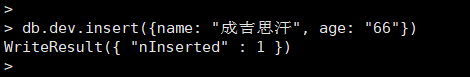
首先需要具备数据库管理员权限，并要切换当前集合所在的数据库。

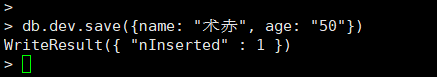


1. **文档**

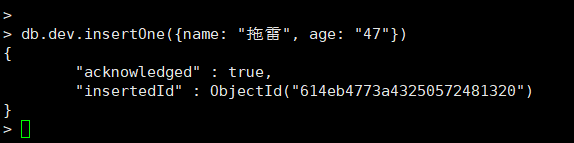
在**mongodb**中文档是指多个键及其相关联的值有序地放置在一起就是文档，指的就是数据。文档的数据结构JSON一样，所有存储在集合中的数据都是**BSON**格式。**BSON**是一种类似**JSON**的二进制形式存储格式，是**Binary** **JSON**的简称。

* **插入文档**
* **插入单个文档**

****

****

在**mongodb3.2**版本中，提供了**insertOne**（）插入文档

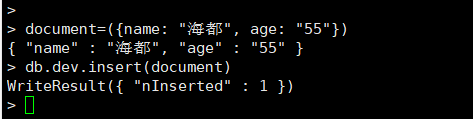
****

* **插入多个文档**

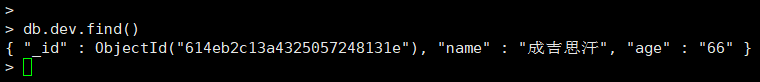
可使用**insert**、**save**、**insertMany**函数进行插入，其中**insertMany**是**3.2**以后版本中新增的一个方法。

**语法格式：**db.集合名.insert([{}, {}]);

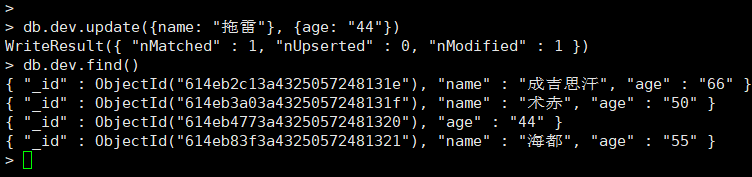
* **通过变量插入文档**

****

* **查看文档**

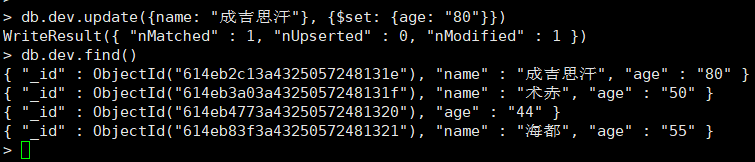
****

* **更新文档**
* **update函数**

****

第一个参数为查询条件；第二个参数要修改的内容；第三个参数可选；update函数更新会丢失数据项，取决于第二个参数的更新项；如果没有传递包含的项就会把原来的踢出，只保留当前传入的项；解决此问题可以使用$set操作符

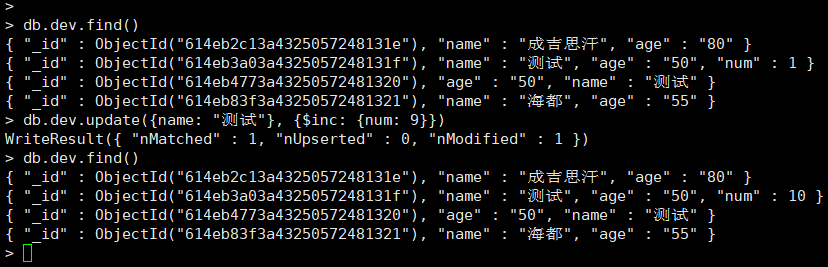
* **$set操作符**

****

注意：**$set**用来指定一个键并更新键值，若键不存在就创建

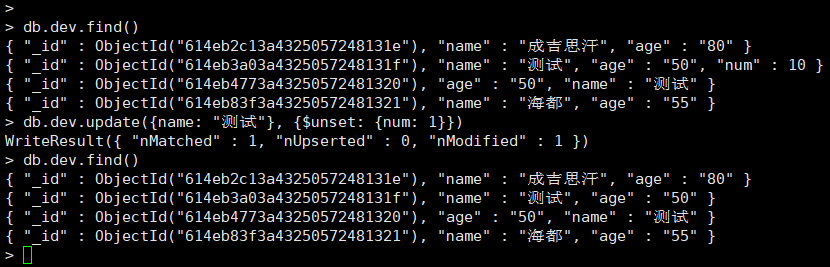
* **$inc操作符**

**$inc**可以对文档的某个值为数字型（只能为满足要求的数字）的键进行增减的操作。



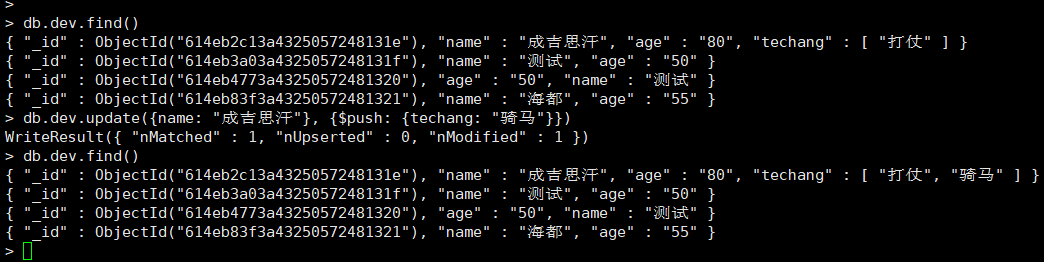
* **$unset操作符**

**$unset**主要是用来删除键



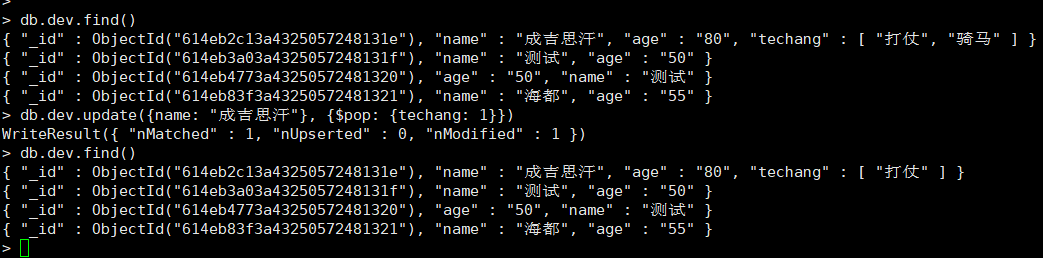
* **$push操作符**

**$push**向文档的某个数组类型的键添加一个元素，不过滤重复的数据，添加时键要存在，并且键值类型必须是数组，键不存在，则创建数组类型的键。



* **$pop操作符**

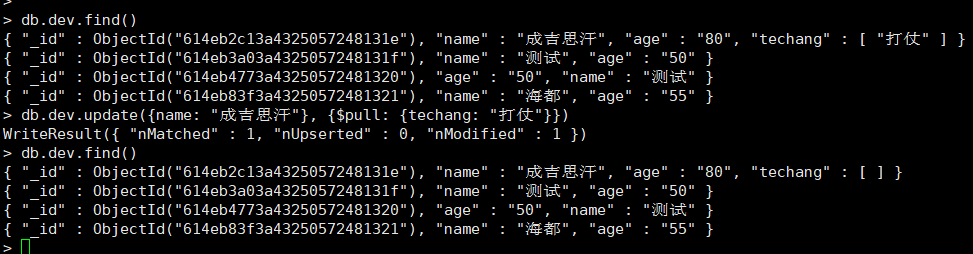
**$pop**和**$push**相对应，删除键为数组类型的最后一项



注意：如果techeng值为1表示从后删除，如果为-1表示从前删

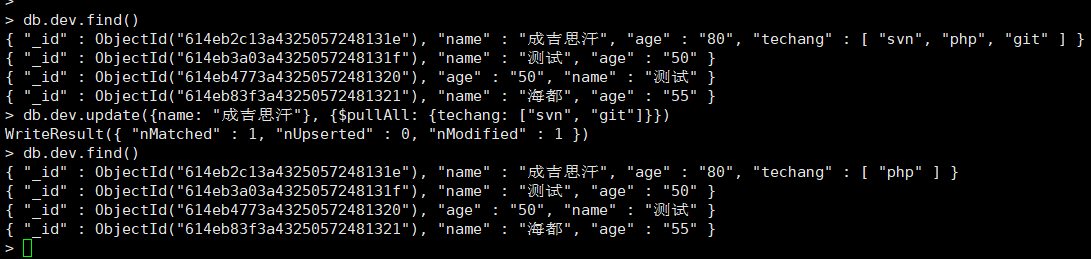
* **$pull操作符**

**$pull**删除满足条件的元素



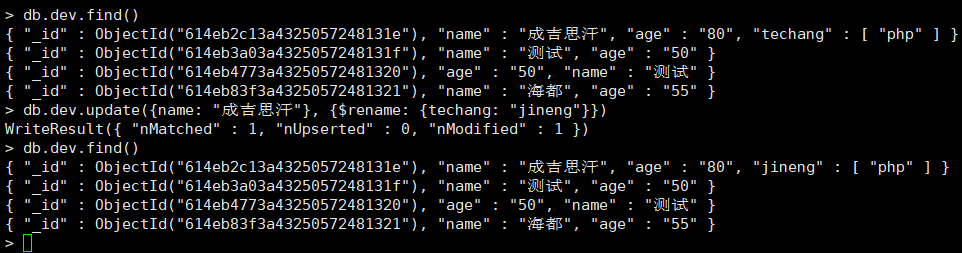
* **$pullAll操作符**

**$pullAll**从数组中删除满足条件的多个元素



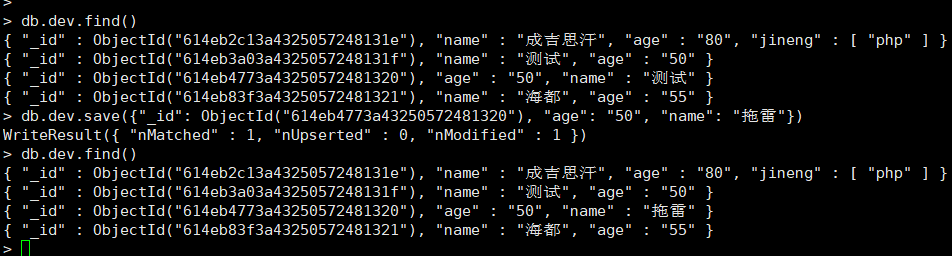
* **$rename操作符**

**$rename**对键进行重新命名

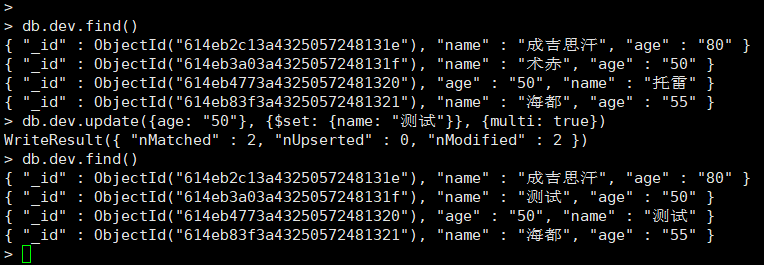
****

* **save（）函数更新文档**

**save函数**通过传入的文档替换已有文档



* **批量更新（multi）**

****

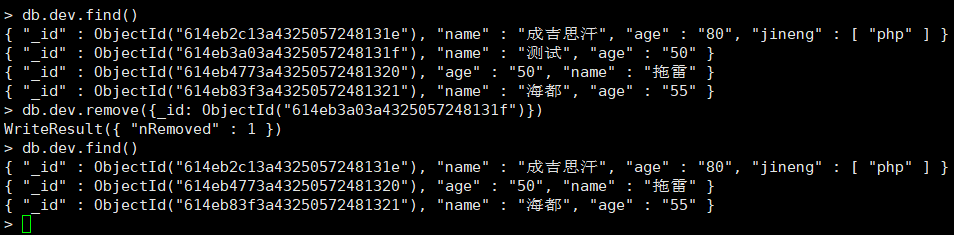
* **删除文档**
* **remove函数**

**remove**删除指定文档，第一参数表示条件，第二参数为可选参数

**remove**会删除所有满足条件的文档，我们可以在第二个参数给定**justOne**，表示只删除第一条；

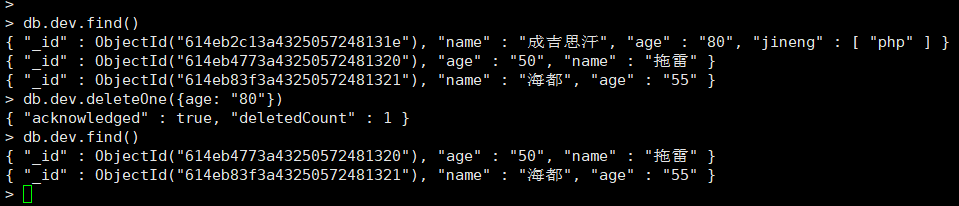
注意：当**justOne**为**1**或**2**时都会只删除第一项；remove函数不会真正释放空间，需要执行db.repairDatabase()来回收磁盘空间

语法：db.集合.remove(条件，2)



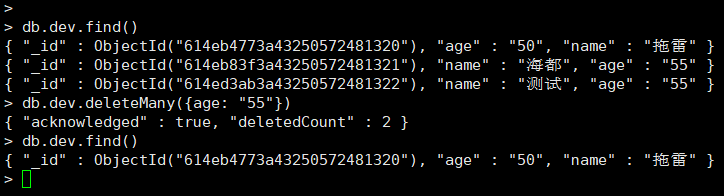
* **deleteOne函数**

此函数为官方推荐的删除文档方法。该方法只删除满足条件的第一条文档



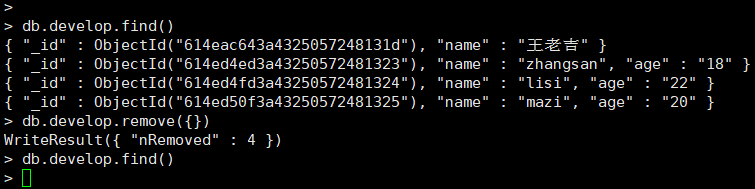
* **deleteMany函数**

此函数为官方推荐的删除文档方法。该方法删除满足条件的所有文档



* **删除集合中的所有文档**

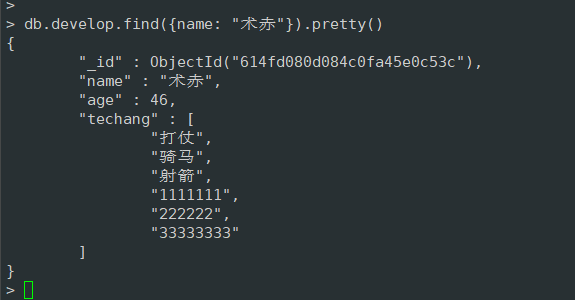
删除所有文档可以使用**remove**或**deleteMany**函数



* **查询文档**
* **find函数**

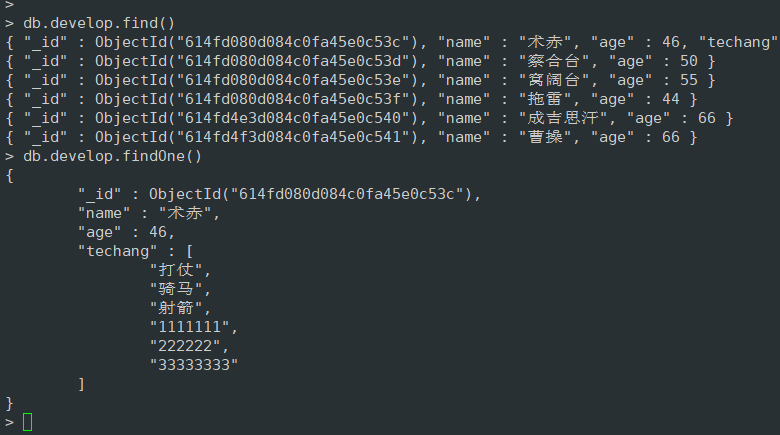
**语法格式：find**（{查询条件（可选）}， {指定投影的键（可选）}）

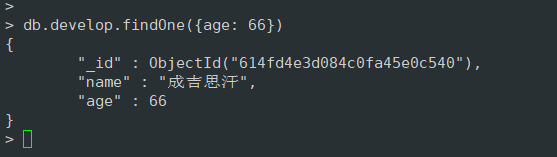
**pretty**函数格式化显示文档



* **findOne函数**

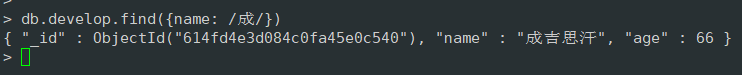
此函数只返回满足条件的第一条文档，如果未做投影操作该方法则自带格式化功能。



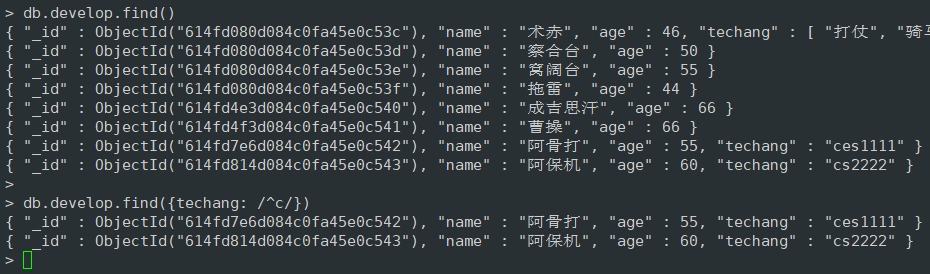


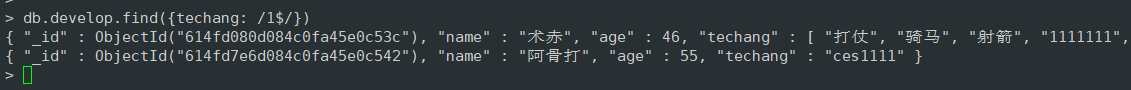
* **模糊查询**

在**mongodb**中，可以通过**//**或**^$**实现模糊查询，注意：使用模糊查询时，查询条件不能放到双引号或单引号中。



注意：//只支持键值为字符串类型

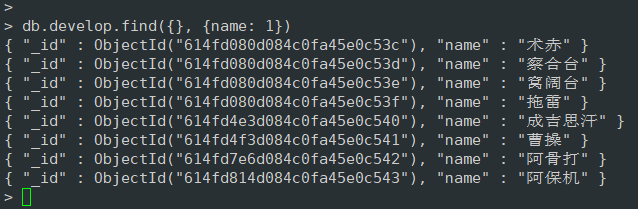


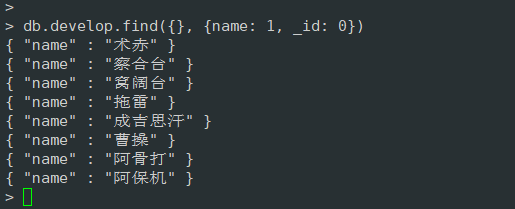


* **find函数投影操作**

**语法：find**（{条件}， {投影键名：0（不显示该列）1（显示该列）}）

**findOne**函数查询格式与以上格式相同





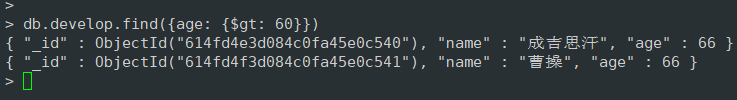
* **条件操作符**

条件操作符用于比较两个表达式并从mongodb中获取文档

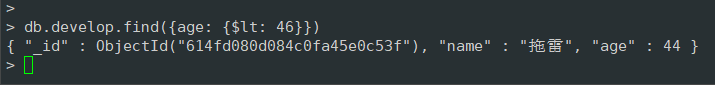
**语法：**find（{键：{操作符：条件}}）

* **$gt(>)大于操作符**

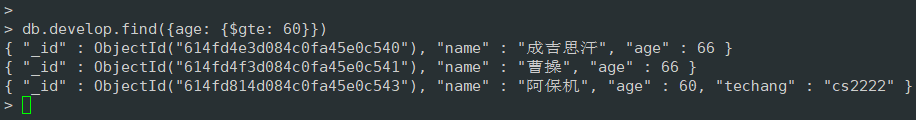
该操作符可以对数字和日期进行判断



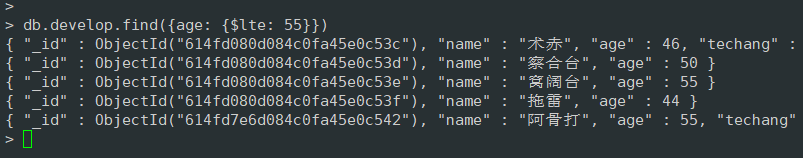
* **$lt(<)小于操作符**

****

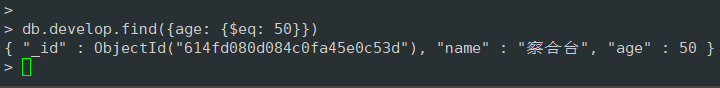
* **$gte(>=)大于等于操作符**

****

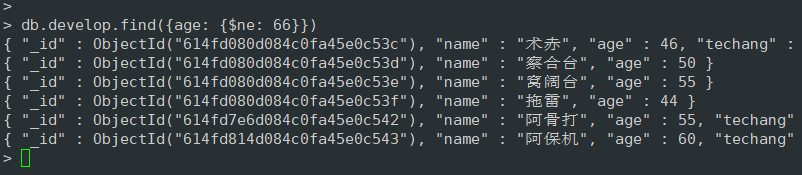
* **$lte(<=)小于等于操作符**

****

* **$eq(==)等于操作符**

****

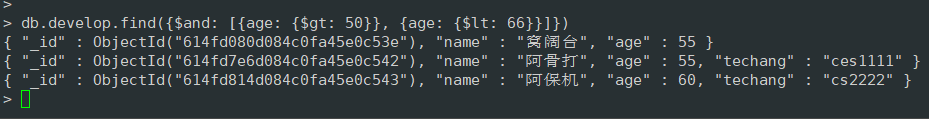
* **$ne(!=)不等于操作符**

****

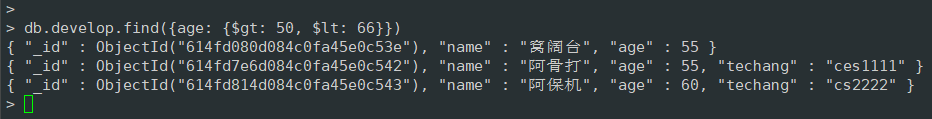
* **$and(和)操作符**

可以使用**$and**来表示多条件间的并且关系

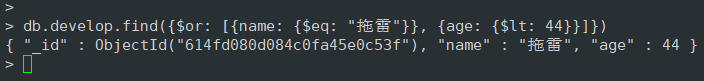
**语法：**find（{$and: [{条件1}, {条件2}]}）



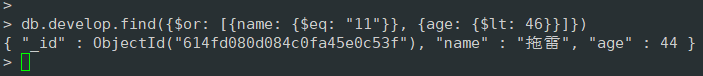
如果在查询中给定了多个查询条件，条件之间的关系默认为and关系。**如：**查询age大于50并且小于66的文档



* **$or(或)操作符**

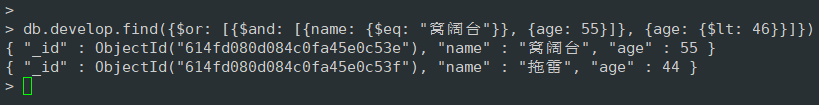
****

条件满足一个即可

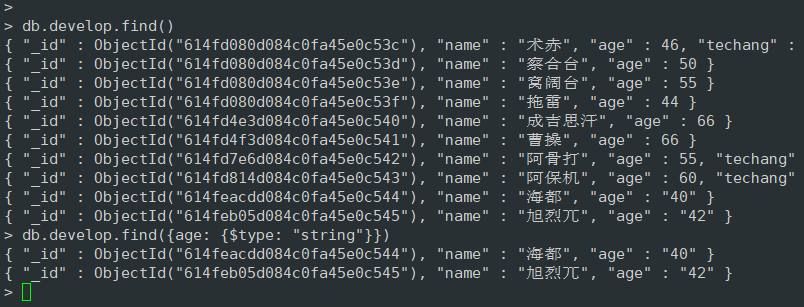
****

* **$and和$or联合使用**

查询人物name为“窝阔台“并且age等于55，或者age小于46的文档

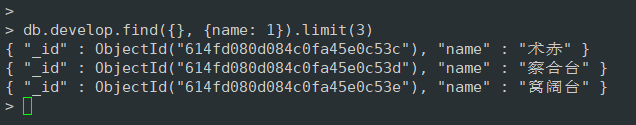


* **$type(键值的类型)操作符**

****

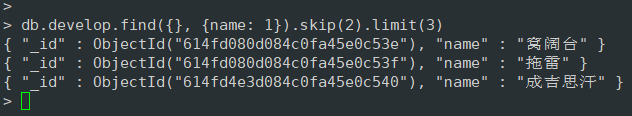
* **Limit函数**

读取指定条数文档



* **Skip函数**

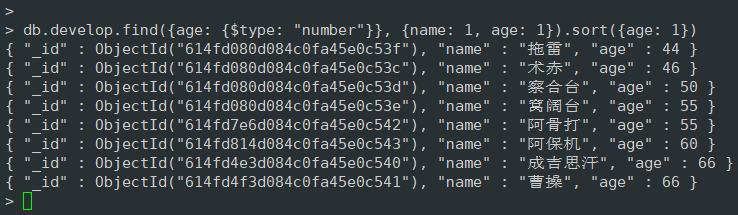
Skip函数跳过指定数量的数据



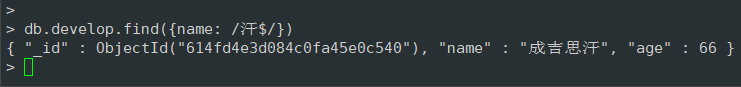
注意：我们可以使用**skip**函数和**limit**函数实现**mongodb**的分页查询，但是官方并不推荐此方式，因为会扫描全部文档然后在返回结果，效率过低。

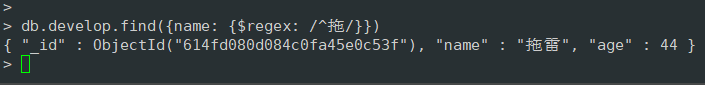
* **sort函数**

**语法：**sort({键：1(升序)-1(降序)})

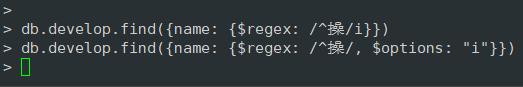


* **正则表达式查询**

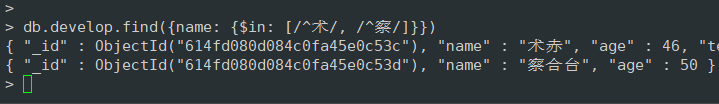
****

* **$regex**
* **$options可选参数**
* **i：**不区分大小写以匹配大小写的情况
* **m：**多行查找，如果内容里面不存在换行符号（\n）或者条件上没有（start/end），该选项没有任何效果
* **x：**设置x选项后，正则表达式中的非转义的空白字符将被忽略。需要$regex与$options语法
* **s：**允许点字符（即）匹配包括换行符在内的所有字符。

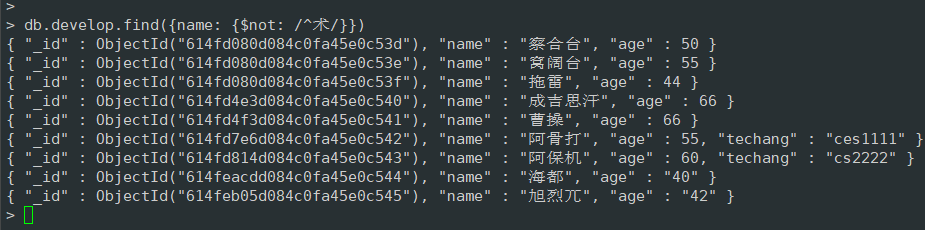
注意：可选参数可以组合使用

****

* **$in**

****

* **$not**

****

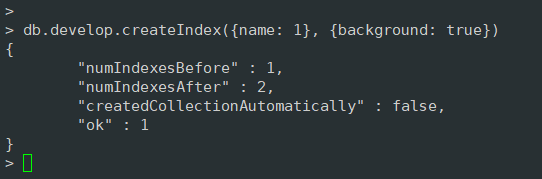
1. **创建索引**

在mongodb中会自动为文档中的**\_id**(文档的主键)键创建索引，与关系型数据库的主键索引类似。创建索引可以使用**createInde**函数来为其它的键创建索引。在创建索引时，需要指定排序规则。**1**按照升序规则；**-1**按照降序规则

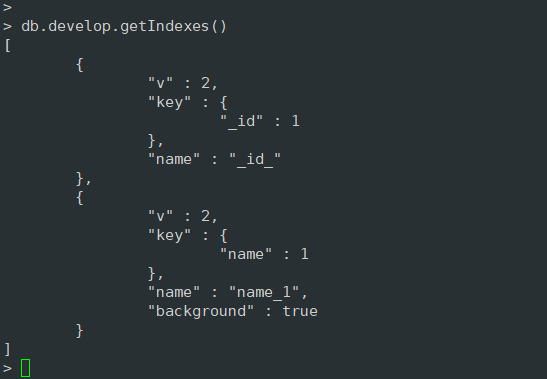
注意：创建索引时，需要具有**dbAdmin**或者**dbAdminAnyDatabase**角色用户

**语法：**createIndex({键：规则}, {可选参数})

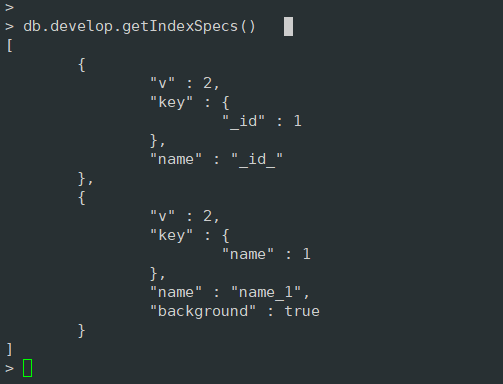
* **可选参数**
* **background（布尔-默认false）:** 后台创建索引，以便创建索引时不阻止其它数据库活动
* **unique（布尔-默认false）:** 创建唯一索引
* **name（string）:** 索引名称；如果未指定，会生成一个索引字段的名称和排序顺序串联
* **partialFilterExpression（document）:**
* **sparse（布尔-默认false）:** 对文档中不存在的字段数据不启用索引
* **expireAfterSeconds（integer）:** 指定索引的过期时间
* **storageEngine:** document类型允许用户配置索引的额存储引擎



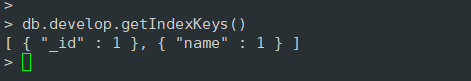
* **查看集合索引**
* **getIndexes函数**

****

* **getIndexSpecs函数**

****

* **查看索引键**
* **getIndexKeys函数**

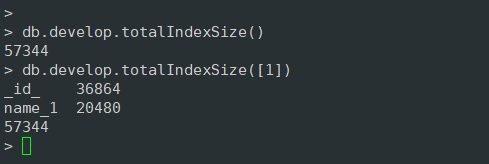
****

* **查看索引大小**

**totalIndexSize**函数来查看当前集合中的索引大小，单位为字节。

**语法：**db.col.totalIndexSize(detail(可选参数))

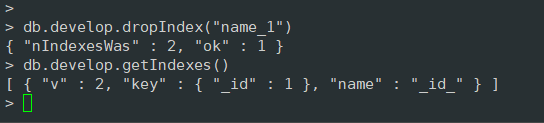
**参数说明：detail**为**0**或**false**外的任意数据，那么会显示该集合中每个索引的大小集合中索引的总大小。如果传入**0**或**false**则只显示该集合中所有索引的总大小，默认为**fasle**。



* **修改索引**

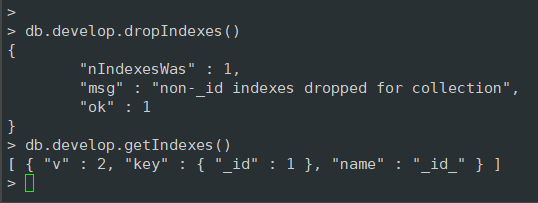
在mongodb中没有单独的修改索引方法，如果要修改某个索引，需要先删除旧的索引，再创建新的索引。

* **删除索引**
* **删除指定索引**

****

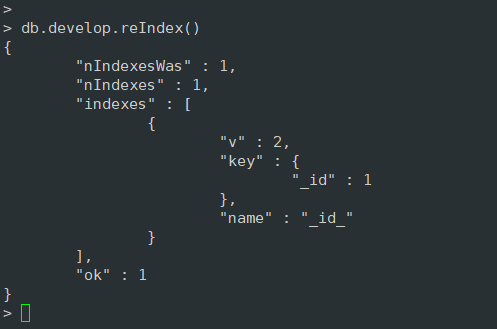
* **删除所有索引**

可以使用**dropIndexes**函数，\_id默认键的索引除外。



* **重建索引**

可以使用**reIndex**函数重建索引，重建索引可以价减少索引存储空间，减少索引碎片，优化索引查询效率。一般在数据大量变化后，会使用重建索引来提升索引性能。重建索引是删除原索引重新创建的过程。不建议反复使用。



* **单字段索引**

单字段索引就是在索引中只包含了一个键。查询时，可加速对字段的各种查询请求，时最常见的索引形式。默认创建的**\_id**索引也是这种类型。我们可以使用**createIndex（{键：排序规则}）**函数来创建单字段索引

* **交叉索引**

交叉索引就是为一个集合的多个字段分别创建索引，在查询的时候通过多个字段作为查询条件。

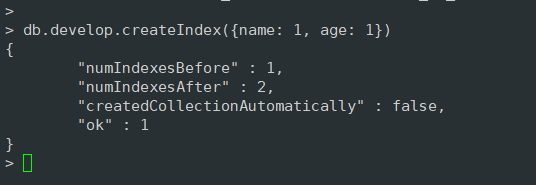
在查询文档时，在查询条件中包含一个交叉索引键或者在一次查询中使用多个交叉索引键作为查询条件都会触发交叉索引。

* **复合索引**

复合索引是**Single Filed Index**的升级版，它针对多个字段联合创建索引，先按第一字段排序，第一字段相同的文档按第二个字段排序，以此类推。

复合索引能满足的查询场景比单字段索引更丰富，不光能满足多个字段组合起来的查询，也能满足所有能匹配符合索引前缀的查询

**语法：**db.col.cerateIndex({键名：排序规则，键名：排序规则})

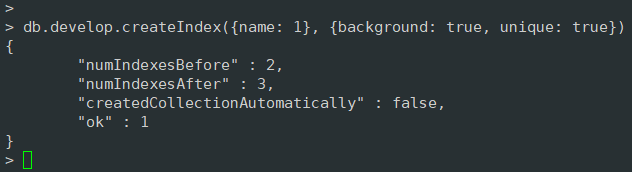


* **多key索引**

当索引字段为数组时，创建出的索引就为多key索引，多key索引会为数组的每个元素建立一条索引。

* **索引的额外属性**
* **唯一索引（unique index）**

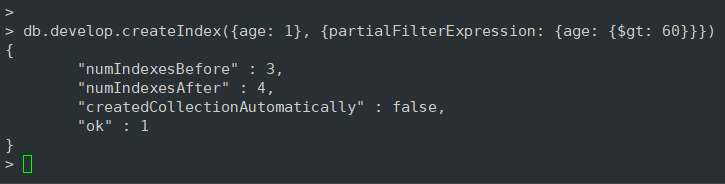
唯一索引会保证索引对应的键不会出现相同的值，比如\_id索引就是唯一索引，如果唯一索引所在的字段有重复的数据写入时，就会抛出异常。



* **部分索引**

部分索引就是针对符合某个特定条件的文档建立索引，**3.2**版本才支持该特性。**Mongodb**部分索引只为那些在一个集合中，满足指定的筛选条件的文档创建索引。由于部分索引是一个集合文档的子集，因此部分索引具有较低的存储需求，并降低了索引创建和维护的性能成本。部分索引通过指定过滤条件来创建，可以为**mongodb**支持所有索引类型使用部分索引。

**语法：**createIndex({键：排序规则}, {partialFilterExpression：{键：{匹配条件：条件值}}})



注意：部分索引只为集合中那些满足条件的文档创建索引。如果你指定的**partilFitlerExpression**和唯一约束、那么唯一性的约束只适用于满足筛选条件的文档。具有唯一约束的部分索引不会阻止不符合过滤条件的文档插入

* **覆盖索引查询**

1. **所有的查询字段是索引的一部分**
2. **所有的查询返回字段在同一个索引中**

由于所有出现在查询中的字段是索引的一部分，**mongodb**无需在整个数据文档中检查匹配查询条件和返回使用相同索引的查询结果

因为所有存在于**RAM**中，从索引中获取数据比通过扫描文档读取数据要快的多。

**如有以下索引：**

db.stu.createIndex({name: 1, age: 1})

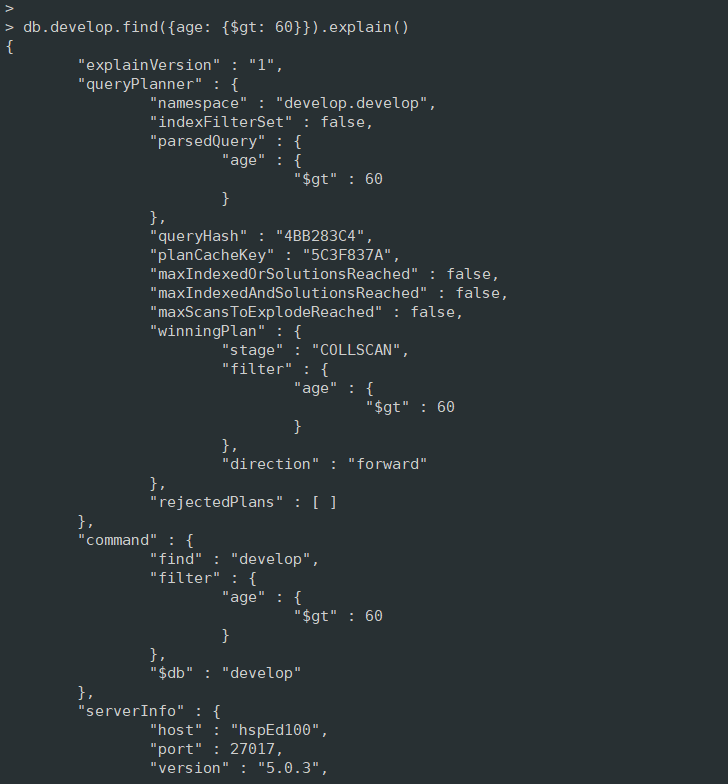
**执行以下查询，该索引会覆盖查询**

db.stu.find({name: “张三“}, {age: 1，\_id: 0})

对于上述查询，**mongodb**不会去数据库文件中查找。相反，它会从索引中提取数据，这是非常快的数据查询。

由于我们的索引中不包括**\_id**字段，**\_id**在查询中会默认返回，我们可以在**mongodb**的查询结果集中排除它。

* **查询计划**

****

* **使用索引注意事项**

索引虽然加快了查询速度，但也是有代价的，索引文件本身要消耗存储空间，同时索引会加重插入、删除、修改记录时的负担。另外，数据库在运行时也要消耗资源维护索引，因此索引并不是越多越好。建议集合文档超过2000条可以酌情考虑创建索引。

* **如何创建合适的索引**
* 为每个常用的查询结构建立合适的索引
* 建议使用复合索引，交叉索引效率较低
* **复合索引的字段排列顺序**

当复合索引内容包含匹配条件以及范围条件的时候，比如：包含用户名（匹配条件）以及年龄（范围条件），那么匹配条件应该放在范围条件之前。

* **查询时尽可能查询出索引字段**

有时候仅需要查询少量部分内容，而且这部分内容刚好都建立了索引，那么尽可能只查询出这些索引内容，需要用到的字段显式声明(\_id字段需要显示忽略！)。因为这些数据需要把原始数据文档从磁盘读入内容，造成一定的损耗。

* **对现有的数据大表建立索引的时候，采用后台运行方式**

在对数据集合建立索引的过程中，数据库会停止该集合的所有读写操作，因此如果建立索引的数据量大，建立过程慢的情况下，建议采用后台运行的方式，避免影响正常的业务流程

* **索引限制**
* **内存使用**

由于索引式存储在内存（**RAM**）中，所以应该确保该索引的大小不超过内存的限制。如果索引的大小大于内存的限制，mongodb就会删除一些索引，这将导致性能下降。

* **查询限制**

**索引不能被以下的查询使用：**

* 正则表达式（最左匹配除外）及非操作符，如**$nin、$not**等
* 算术运算符，如**$mod**

检测你的语句是否使用索引可以使用**explain**查看

* **最大限制**
* 集合中索引不能超过64个
* 索引名的长度不能超过128个字符
* 一个复合索引最多可以有31个字段

1. **聚合**

在**mongodb**中可以通过**aggregate**函数来完成一些聚合查询，**aggregate**主要用于处理诸如统计和平均值、求和等，并返回计算后的数据。

**语法：**aggregate({$group: {\_id: {“$分组键名”, “$分组键名”}}}, {条件筛选：{键：{运算条件：运算值}}})

* **常见的mongo的聚合与musql的查询做类比**

|  |  |
| --- | --- |
| **Mysql** | **Mongodb** |
| where | $match |
| group by | $group |
| having | $match |
| select | $project |
| order by | $sort |
| limit | $limit |
| sum | $sum |
| count() | $sum |
| join | $lookup(v3.2新增) |

* **$sum**
* **查询develop集合中一共有多少文档**

**Sql语句写法**：select count(\*) as count from develop

**Mongodb写法：**

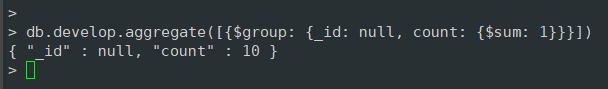
db.develop.aggregate([{$group: {\_id: null, count: {$sum: 1}}}])

说明：

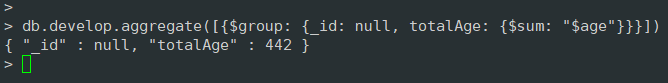
**$group:** 分组，代表聚合的分组条件

**\_id:** 分组的字段。相当于Sql分组语法group by column\_name,如果根据某字段的值分组，则定义为\_id: “$字段名”。所以此案例中的null代表一个固定的字面值null

**count**：返回结果字段名。可以自定义，类似mysql中的字段别名

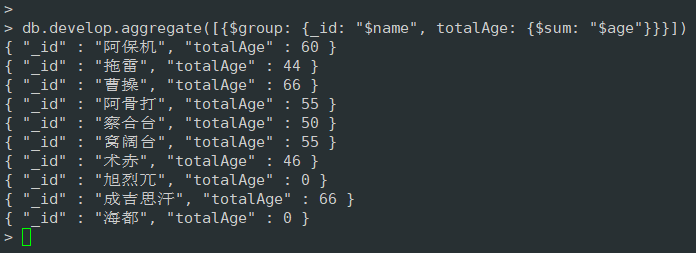
**$sum**: 求和表达式。相当于mysql中的sum  


* **查询develop集合中的所有age键值的总合**



**$age**：代表文档中的age字段的值

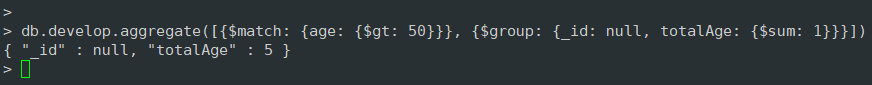
* **对每个name进行分组并计算每组中的age的总合**



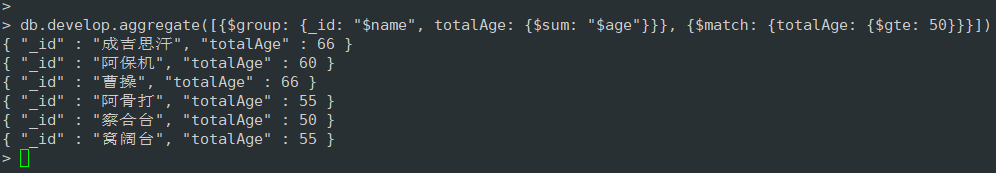
* **$match(条件筛选)**

相当于mysql中的where；如果$match写在$group前面等于先筛选再进行分组；如果写在后面等于先分组再对分组文档进行筛选

* **查询develop集合中有多少文档的age大于50**

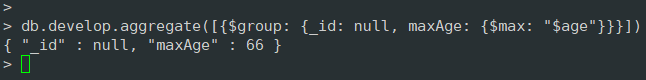
****

* **查询develop集合，根据name分组计算出每组的age总合，并过滤掉总合大于等于50的文档**

****

* **$max(最大值)**

查询develop集合中，age最大的文档



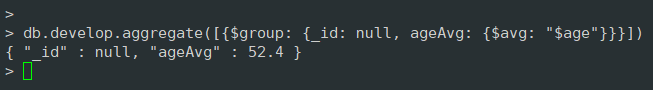
注意：获取的$max值类型必须为number

* **$min(最小值)**

与$max相对应

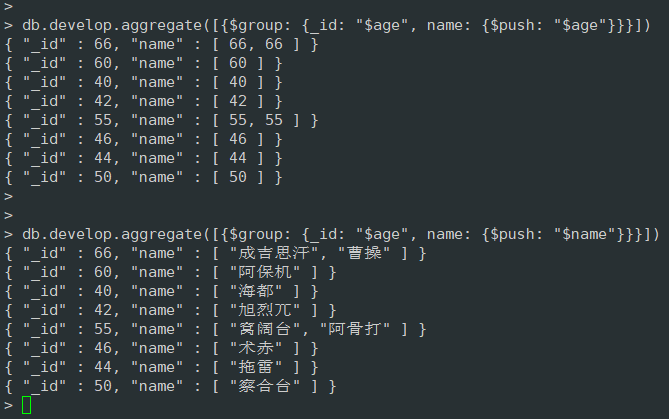
* **$avg(平均值)**

查询develop集合中age的平均值



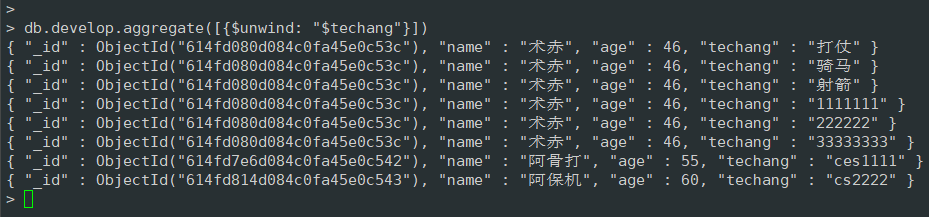
* **$push(统计结果返回数组)**

查询develop集合，按照age分组并返回它们的name，如果age相同则使用数组返回它们的name



* **$unwind(数组字段拆分)**

查询develop集合，将数组中的内容拆分显示



* **管道操作**

管道在Unix和linux中一般用于将当前命令的输出结果作为下一个命令的参数。

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个下一个管道处理。管道操作是可重复的。

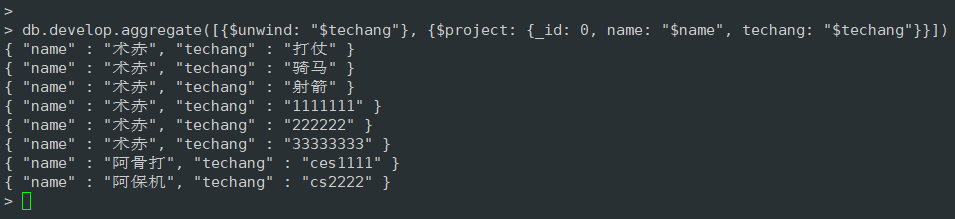
管道操作符是按照书写顺序依次执行的，每个操作符都会接受一连串的文档，对这些文档做一些类型转换，最后将转换后的文档作为结果传递给下一个操作符（对于最后一个管道操作符，是将结果返回给客户端），称为流式工作方式。

**管道操作符：**$match、$group、$sort、$limit、$skip、$unwind

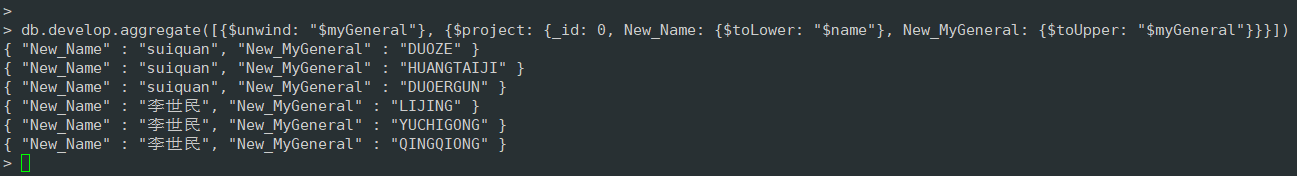
管道操作符，只能用于计算当前聚合管道的文档，不能处理其它文档

1. **$project聚合投影约束**

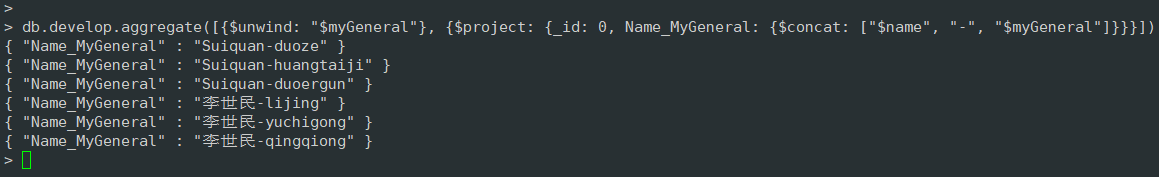
* **查询develop集合中，将数组中的内容拆分显示，并只显示name和techang键**

****

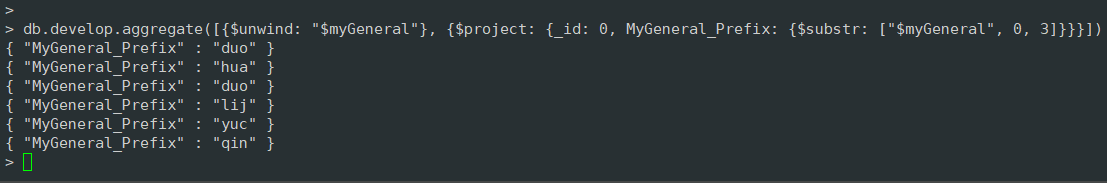
* **$project字符串处理**
* 查询develop集合，将数组中的内容拆分显示，将name的转为小写(**$toLower**)，命名为New\_Name; 将myGeneral值转为大写（**$toUpper**），命名为New\_MyGeneral



* 查询develop集合，将数组拆分显示，将name键和myGeneral键值拼接(**$concat**)为一个完整字符串并在Name\_MyGeneral字段中显示



* 查询develop集合，将数组拆分显示，并且只显示myGeneral前3个字符(**$substr**)

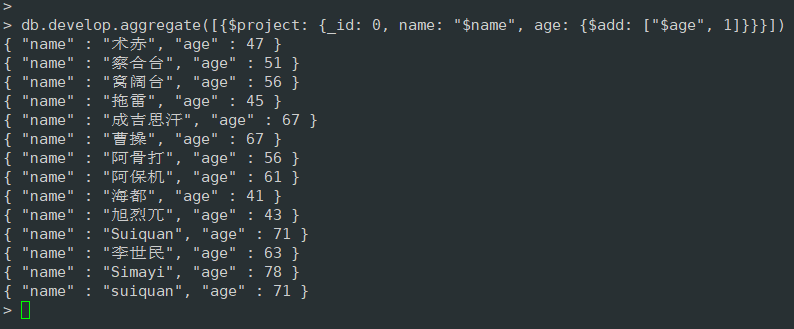


注意：**$substr**不能截取汉字，原因是$substr只能匹配**ASCLL**数据，对于中文要使用**$substrCP**

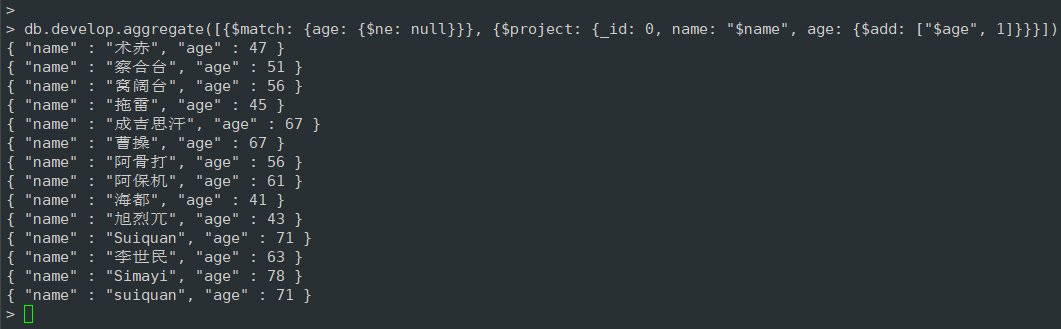
* **$project算术运算**

在$project中，可以通过mongodb的算术运算符进行对投影的约束。

* **查询develop集合，对age字段数据进行加1操作**

****

* **排除没有age的键的文档**

****