MongoDB 教程

0 课程内容

第一次课

1. MongoDB 适用场景
2. MongoDB 的基本原理
3. MongoDB 集合 CURD
4. MongoDB 聚合操作

第二次课

5. MongoDB 副本集集群原理和实践
6. MongoDB 分片集群原理和实践
7.通过程序操作 mongodb

资源：

英文官方参考文档：https://docs.mongodb.com/manual/reference/

中文社区参考文档：https://docs.mongoing.com/

中文社区：https://mongoing.com/

博客参考：https://mongoing.com/guoyuanwei

中文社区公众号：https://mp.weixin.qq.com/s/Wuzh47jsBh5QonBrZxUnjg 《WiredTiger 存储引擎系列》

1 MongoDB 简述

1.1 MongoDB 的历史

关系型的数据库已经出现了近 40 年，并且在很长一段时间里，一直是数据库领域当之无愧的王者，例如 SQL Server、MySQL、Oracle 等，目前在数据库领域中仍处于主导地位。但随着信息时代数据量的增大以及 Web2.0 的数据结构复杂化，关系型数据库的一些缺陷也逐渐显现出来，主要包括以下几项。

（1） 大数据处理能力差。关系型数据库被设计为单机运行，在处理海量数据方面代价高昂，甚至

无法承担重任。

（2） 程序产出效率低。我们在开发程序使用关系型数据库会发现， 更多的精力被用在了建立关系型数据库表的数据结构与开发语言数据结构的映射上。使用关系型数据库时，为了实现系统中某个实体的存储查询操作，我们首先需要设计表的结构和字段以及数据类型。于是无论是创建、删除还是更新，我们要涉及的操作增加了许多。

（3） 数据结构变动困难。互联网项目时刻都在发展和变动，改变存储单元的结构是常事，但是生产环境中关系型数据库要增加或减少一个字段无疑是非常严肃、 重要并且是容易生意外的事情。

为了解决这些存在的问题，一个更好的数据库方案 NoSQL 数据库应运而生。 所谓 NoSQL ，并不是指没有 SQL ，而是指“Not Only SQL ”，即非传统关系型数据库。这类数库的主要特点包括非关系型、水平可扩展、分布式与开源；另外它还具有模式自由、 最终一致性。NoSQL 常用的存储模式有 key-value 存储、文档存储、列存储、 图形存储、 XML 存储等， MongoDB 正是文档数据库的典型代表。

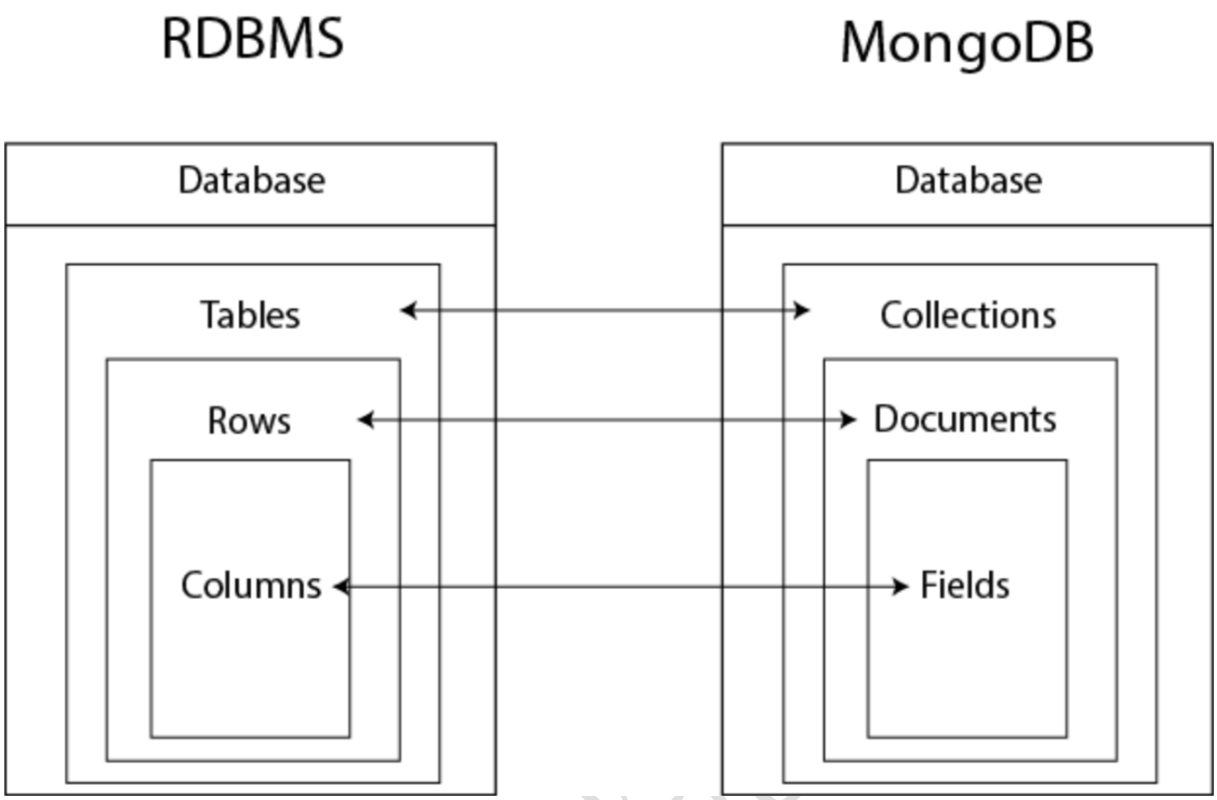
MongoDB 可以在不受任何表格 schema 模式的约束下工作。数据以类似 JSON 的格式存储，并且可以包含不同类型的数据结构。例如，在同一集合 collection 中，我们可以拥有以下两个文档 document：

```
{
    id: '4',
    name: 'Mark',
    age: '21',
    addresses : [
        { street: '123 Church St', city: 'Miami', cc: 'USA' },
        { street: '123 Mary Av', city: 'Los Angeles', cc: 'USA' }
    ]
}

{
    id: '15',
    name: 'Robin',
    department: 'New Business',
    example: 'robin@example.com'
}
```

1.2 数据如何存储在 MongoDB 中

与传统的 RDBMS 关系型数据库不同，MongoDB 并没有表 Table，行 row 和列 column 的概念。它将数据存储在集合 collections，文档 documents 和字段 fields 中。下图说明了与 RDBMS 类比的结构之间的关系：



1.3 MongoDB 数据库定位

- 原则上 Oracle 和 MySQL 能做的事情，MongoDB 都能做（包括 ACID 事务)
- 优点：横向扩展能力，数据量或并发量增加时候架构可以自动扩展
- 优点：灵活模型，适合迭代开发，数据模型多变场景
- 优点：JSON 数据结构，适合微服务/REST API

1.4 版本变迁

0.x:	2008	起步阶段
1.x:	2010	支持复制集和分片集
2.x	2012	更丰富的数据库功能
3.x	2014	WiredTiger 和周边生态环境
4.x:	2018	分布式事务支持
5.x	2021	在线重新分片

1.5 基于功能选择 MongoDB

特性	MongoDB	关系型数据库
亿级以上数据量	轻松支持	要努力一下，分库分表 Mysql 分库分表 mycat
灵活表结构	轻松支持	Entity Key /value 表，关联查询比较痛苦
高并发读	轻松支持	需要优化
高并发写	轻松支持	需要优化
跨地区集群	轻松支持	需要定制方案
分片集群	轻松支持	需要中间件
地理位置查询	比较完整的地理位置	PG 还可以，其他数据库略麻烦
聚合计算	功能很强大	使用 Group By 等，能力有限
异构数据	轻松支持	使用 EKV 属性表
大宽表	轻松支持	性能受限

大宽表从字面意义上讲就是字段比较多的数据库表

1.6 MongoDB 适用场景

1. 移动应用（App、小程序）
2. 电商
3. 内容管理
4. 物联网
5. Saas 应用
6. 主机分流
7. 关系型迁移

移动应用

场景特点	MongoDB 选型考量
基于 REST API /JSON 快速迭代，数据结构变化频繁 地理位置功能 爆发增长可能性 高可用	文档模型可以支持不同的结构 原生地理位置功能 横向扩展能力支撑爆发增长 复制集机制快速提供高可用 哈啰单车/Keep(自由运动场)

电商商品信息

场景特点	MongoDB 选型考量
商品信息包罗万象 商品的属性不同品类差异很大 数据库模式设计困难	文档模型可以集成不同商品属性 可变模型适合迭代 京东商城/小红书

内容管理

场景特点	MongoDB 选型考量
内容数据多样，文本，图片，视频 扩展困难，数据量爆发增长 **Vivo 评论**	JSON 结构可以支持非结构化数据 分片架构可以解决扩展问题-Adobe AEM / Sitecore

物联网

场景特点	MongoDB 选型考量
传感器的数据结构往往是半结构化 传感器数量很大，采集频繁 数据量很容易增长到数亿到百亿	JSON 结构可以支持半结构化数据 使用分片能力支撑海量数据 JSON 数据更加容易和其他系统通过 REST API 进行集成 华为/Bosch /Mindsphere

SaaS 应用

场景特点	MongoDB 选型考量
多租户模式，需要服务很多客户 需求多变，迭代压力大 数据量增长快	无模式数据库，适合快速迭代 水平扩展能力可以支撑大量用户增长 ADP /Teambition

主机分流

场景特点	MongoDB 选型考量
金融行业传统采用 IBM 或者小机 传统瀑布开发模式流程长成本高 结构不易改变，难于适应新需求 根据某银行的统计，99%的数据库操作为读流量 基于 MIPS 付费，读流量成本高	使用实时同步机制，将数据同步出来到 MongoDB 使用 MongoDB 的高性能查询能力来支撑业务的读操作 相比于关系模型数据库，更加容易迁入数据并构建成 JSON 模型进行 API 服务 中国银行/Barclays

关系型数据库替换

场景特点	MongoDB 选型考量
基于 Oracle /MySQL/ sQLServer 的历史应用 数据量增长或者使用者变多以后性能无 法支持新的业务需求 分库分表需要应用配合，复杂且坑多 结构死板，增加新需求复杂困难	高性能高并发的数据库性能 无需应用分库分表，集群自动解决扩容问题 动态模型适合快速开发 头条/网易/百度/东航/中国银行

其他场景

还有其他的场景

MongoDB 适用于以下场景，只要有一项需求满足就可以考虑匹配越多，就约可以考虑使用 MongoDB。

（1）网站数据

MongoDB 非常适合实时地插入、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。如果你正考虑搭建一个网站，可以考虑使用 MongoDB，你会发现它非常适用于迭代更新快、需求变更多、以对象数据为主的网站应用。{"name"："joe"， "age"： 18}

（2）缓存

由于 MongoDB 是内存+硬盘型数据库，性能很高，MongoDB 也适合作为信息基础设施的缓存层。在系统重启后，由 MongoDB 搭建的持久化缓存可以避免下层的数据过载。

（3）大尺寸、低价值的数据

使用传统的关系型数据库存储一些数据会超级麻烦，首先要创建表，再设计数据表结构，进行数据清理，得到有用的数据，按格式存入表中；而 MongoDB 可以随意构建一个 Json 格式的文档就能把它先保存起来，留着以后处理。

（4）高伸缩性的场景

如果网站的数据量非常大，很快就会超过一台服务器能够承受的范围，那么 MongoDB 可以胜任网站对数据库的需求，MongoDB 可以轻松地自动分片到数十甚至数百台服务器。

（5）用于对象及 JSON 数据的存储

MongoDB 的 BSON 数据格式非常适合文档格式化的存储及查询。

（6）不需要事务及复杂 join 支持

（7）新应用，需求会变，数据模型不确定，想快速迭代开发

（8）要应对 2000~3000 以上的的读写 QPS（或更高）

（9）要存储 TB 甚至 PB 级别数据

（10）应用发展迅速，要快速水平扩展

（11）有大量的地理位置查询，文本查询。

从目前阿里云 MongoDB 云数据库上的用户看，MongoDB 的应用已经渗透到各个领域，比如游戏、物流、电商、内容管理、社交、物联网、视频直播等，以下是几个实际的应用案例。

- 游戏场景，使用 MongoDB 存储游戏**用户信息，用户的装备、积分等直接以内嵌文档的**形式存储，方便查询、更新
- 物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌

数组的形式来存储，一次查询就能将订单所有的变更读取出来。

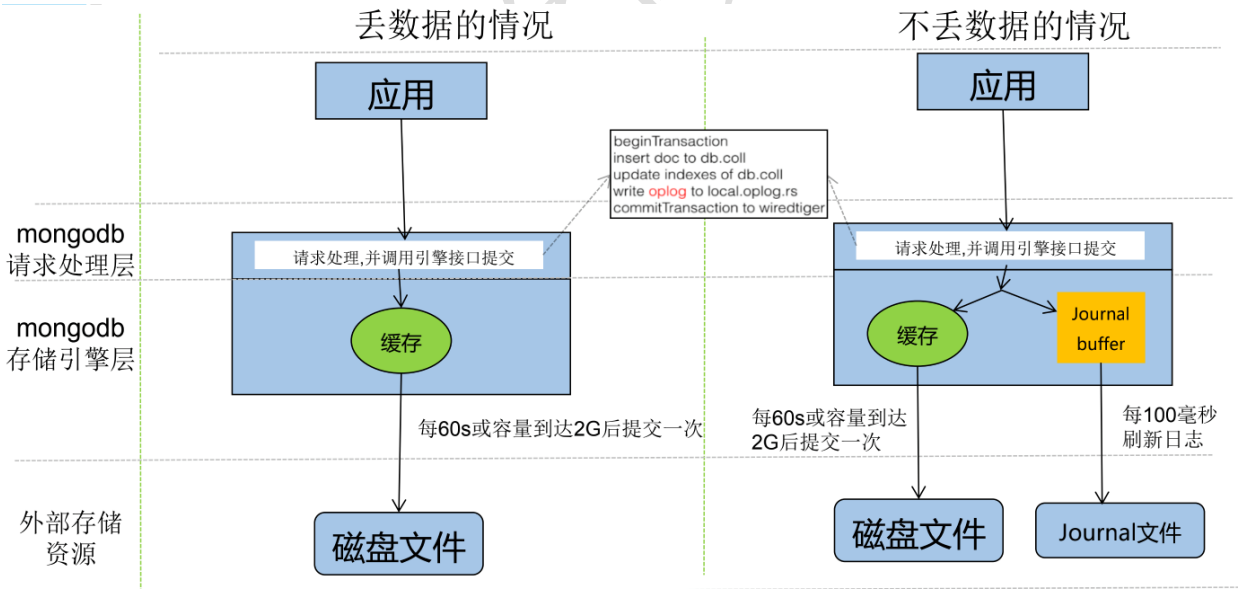
- 社交场景，使用 MongoDB 存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能
- 物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析
- 视频直播，使用 MongoDB 存储用户信息、礼物信息等

……

2 MongoDB 原理

MongoDB 存取读写速度快，甚至可以用来当作缓存数据库。但是在使用过程中会发现 MongoDB 服务非常占内存，几乎是服务器有多少内存就会占用多少内存。为什么会出现这种情况呢？我们要从 MongoDB 的读写工作流程和对内存的使用方式说起 。

MongoDB 在存取工作流程上有一个非常酷的设计决策，MongoDB 的所有数据实际上是存放在硬盘的，然后把部分或者全部要操作的数据通过内存映射存储引擎映射到内存中。 即是：**虚拟内存+持久化的存储方式。**



如果是读操作，直接从内存中取数据，如果是写操作，就会修改内存中对应的数据，然后就不需要管了。操作系统的虚拟内存管理器会定时把数据刷新保存到硬盘中。 内存中的数据什么时候写到硬盘中，则是操作系统的事情了。

MongoDB 的存取工作流程区别于一般硬盘数据库在于两点：

- 读：一般硬盘数据库在需要数据时才去硬盘中读取请求数据， MongoDB 则是尽可能地放入内

存中。

- 写：一般硬盘数据库在有数据需要修改时会马上写入刷新到硬盘，MongoDB 只是修改内存中的数据就不管了，写入的数据会排队等待操作系统的定时刷新保存到硬盘。

MongoDB 的设计思路有两个好处：

（1）将什么时候调用 IO 操作写入硬盘这样的内存管理工作交给操作系统的虚拟内存管理器来完成，大大简化了 MongoDB 的工作。

（2）把随机的写操作转换成顺序的写操作，顺其自然地写入，而不是有数据修改就调 IO 操作去写入，这样减少了 IO 操作，避免了零碎的硬盘操作，大幅度提升性能。

但是这样的设计思路也有坏处：

（1）如果 MongoDB 在内存中修改了数据，在数据刷新到硬盘之前，停电了或者系统岩机了，就会丢失数据了。

针对这样的问题，MongoDB 设计了 Journal 模式，Journal 是服务器意外岩机的情况下，将数据库操作进行重演的日志。如果打开 Journal ，默认情况下 MongoDB 100 毫秒（这是在数据文件和 Journal 文件处于同磁盘卷上的情况，而如果数据文件和 Journal 文件不在同磁盘卷上时，默认刷新输出时间是 30 毫秒）往 Journal 文件中 flush 次数据，那么即使断电也只会丢失 100ms 的数据，这对大多数应用来说都可以容忍了。从版 1.9.2+, MongoDB 默认打开 Journal 功能，以确保数据安全。而且 Journal 的刷新时间是可以改变的，使用--journalCommitInterval 命令修改，范围是 2~300ms 值越低，刷新输出频率越高，数据安全度也就越高，但磁盘性能上的开销也更高。MongoDB 存取工作流程的实现关键在于通过内存映射存储引擎把数据映射到内存中。

2.1 存储原理- WiredTiger 引擎

MongoDB 目前支持的 MMAP，MMAPV1，WiredTiger 以及 In-Memory 存储引擎。主要关注 WiredTiger

WiredTiger（3.2 以上版本默认）（B+树）

- 采用**文档型锁**，吞吐量相对高许多
- 相比 MMAPV1 存储索引时 WiredTiger 使用**前缀压缩**,更节省对内存空间的损耗
- 提供**压缩算法**,可以大大降低对硬盘资源的消耗,节省约 60%以上的硬盘资源；
- 支持 snappy（默认）和 zlib 两种压缩模式
- 默认每分钟一次 checkpoint，及数据持久化

官方讲解：https://docs.mongoing.com/cun-chu/cun-chu-yin-qing/wiredtiger-storage-engine《WiredTiger

存储引擎》

详情强烈参考：https://mp.weixin.qq.com/s/Wuzh47jsBh5QonBrZxUnjg 《WiredTiger 存储引擎系列》

扫码关注 Mongoing 中文社区公众号，里面有大量 mongodb 相关的技术文档。

内存使用

通过 WiredTiger，MongoDB 可以利用 WiredTiger 内部缓存和文件系统缓存。

从 MongoDB 3.4 开始，默认的 WiredTiger 内部缓存大小是以下两者中的较大者： 50％（RAM-1 GB）或 256 MB。

如果是 2G 内存，50%*（2-1）=500M

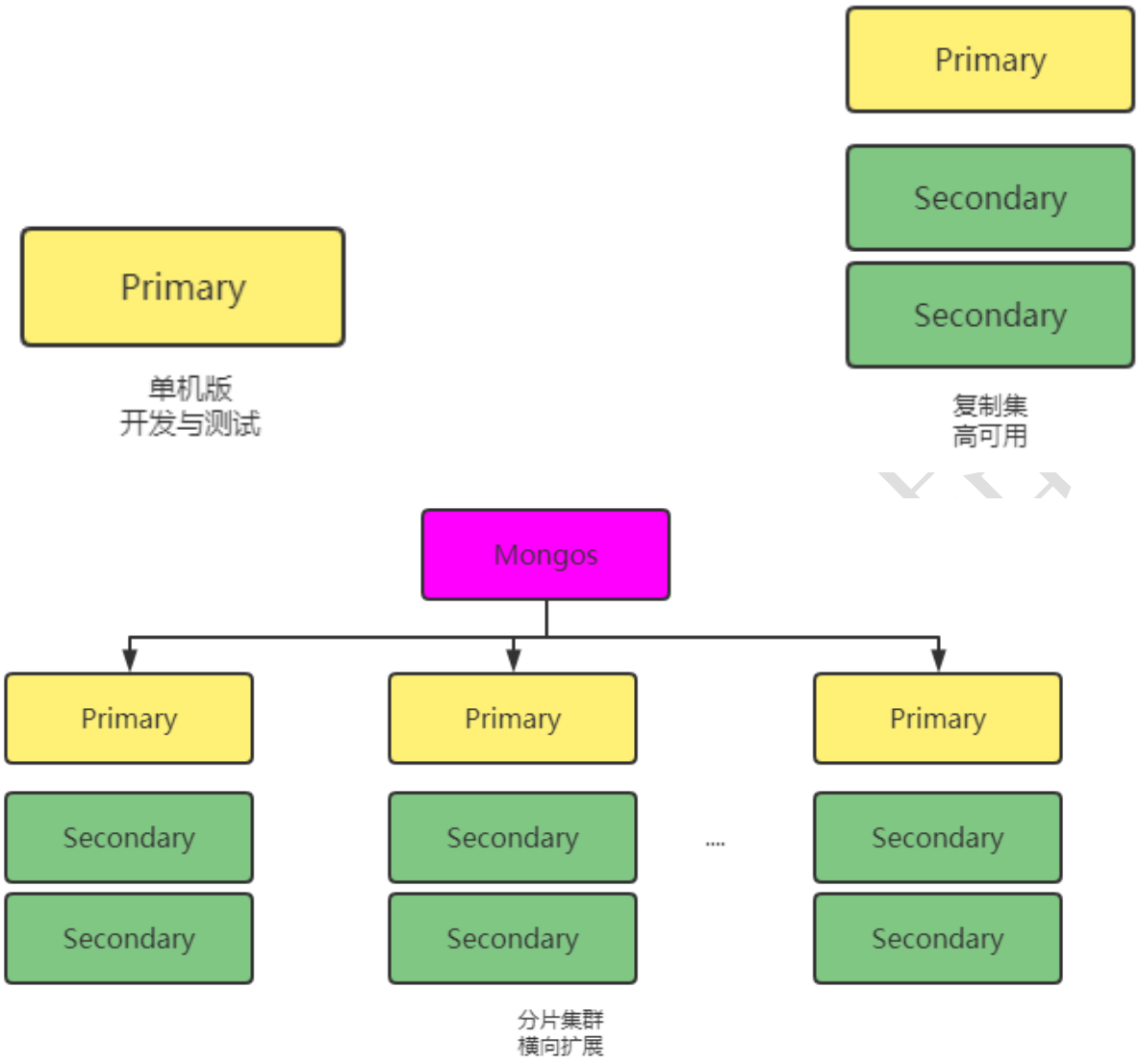
4G 内存，50%*（4-1）=1.5G

例如，在总共有 4GB RAM 的系统上，WiredTiger 缓存将使用 1.5GB RAM（0.5 *（4 GB-1 GB）= 1.5 GB）。*相反，总内存为 1.25 GB 的系统将为 WiredTiger 缓存分配 256 MB，因为这是总 RAM 的一半以上减去一 GB（0.5* （1.25 GB-1 GB）= 128 MB

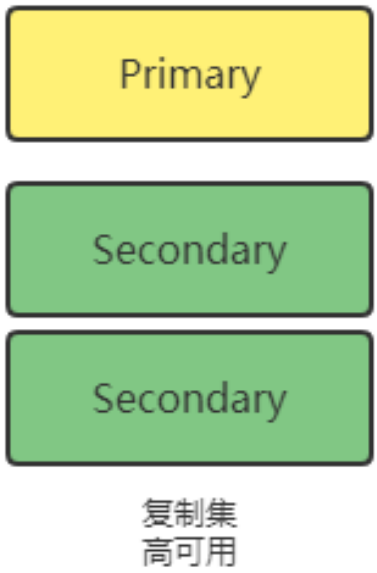
小型应用，还是用 mysql 节省服务器的资源。

2.2 单机使用

主要用于开发测试。


Primary
Secondary
Secondary
Primary
单机版
开发与测试
复制集
高可用
Mongos
Primary
Primary
Primary
Secondary
Secondary
Secondary
....
Secondary
Secondary
Secondary
分片集群
横向扩展

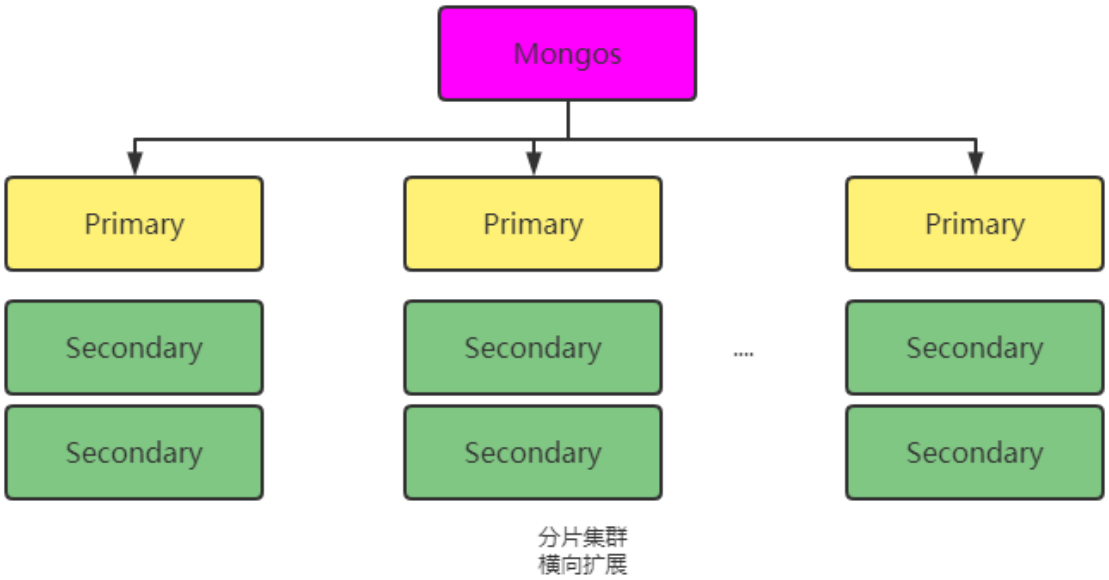
2.3 复制集集群



官方统计百分之七八十的线上项目采用复制集的方式进行部署。

2.4 分片集群

10%左右采用分片集群的方式部署。



3 安装 MongoDB

对应文档：

https://docs.mongodb.com/manual/introduction/

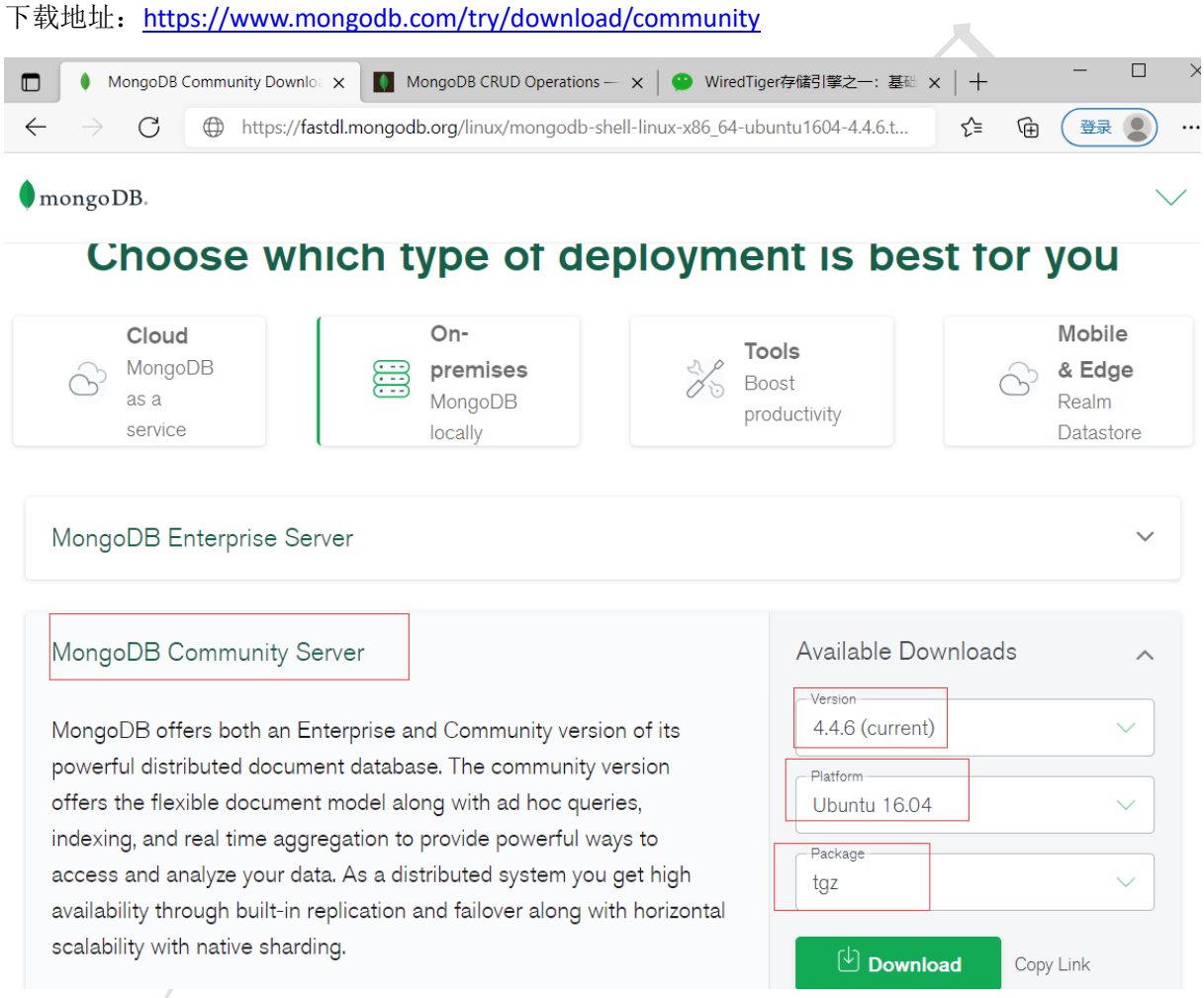
https://docs.mongoing.com/mongodb-crud-operations/insert-documents

3.1 下载和安装

安装：https://docs.mongodb.com/manual/administration/install-community/

MongoDB 提供了 linux 各发行版本 64 位的安装包，你可以在官网下载安装包。

下载地址：https://www.mongodb.com/try/download/community



我这里选择的 Ubuntu 16.04 版本

https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu1604-4.4.6.tgz

下载完安装包，并解压 **tgz**（以下演示的是 64 位 Linux 上的安装） 。

```
# wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu1604-4.4.6.tgz --no-check-certificate
# tar xvf mongodb-linux-x86_64-ubuntu1604-4.4.6.tgz      # 解压 mongodb 安装包
# mv mongodb-linux-x86_64-ubuntu1604-4.4.6 /usr/local/mongodb  # 移动到安装目录
```

```
# vim /etc/profile   # 添加环境变量
在/etc/profile 最后一行，添加 export PATH=/usr/local/mongodb/bin:$PATH
# source /etc/profile
```

```
export PATH=/usr/local/mongodb/bin:$PATH
```

3.2 对应命令和工具说明

软件模块	描述
mongod	MongoDB 数据库软件
mongo	MongoDB 命令行工具，管理 MongoDB 数据库
mongos	MongoDB 路由进程，分片环境下使用

3.3 启动 MongoDB 服务

3.3.1 创建数据库目录

MongoDB 的数据存储在 data 目录的 db 目录下，但是这个目录在安装过程不会自动创建，所以你需要手动创建 data 目录，并在 data 目录中创建 db 目录。

以下实例中我们将 data 目录创建于根目录下(/)。

注意：/data/db（需要自己手动创建） 是 MongoDB 默认的启动的数据库路径(--dbpath)，也可以自己指定路径，比如我放在/home/lqf/data/db。

```
mkdir -p /home/lqf/data/db
```

3.3.2 运行 MongoDB 服务

你可以在命令行中执行 mongo 安装目录中的 bin 目录执行 mongod 命令来启动 mongdb 服务。

注意：如果你的数据库目录不是/data/db，可以通过 --dbpath 来指定。

默认端口为：27017

```
$ mongod --dbpath=/home/lqf/data/db --logpath /home/lqf/data/db/mongod.log  --bind_ip=0.0.0.0  --fork
mongod: /usr/local/lib/libcurl.so.4: no version information available (required by mongod)
2019-09-17T12:17:12.295+0800 I STORAGE  [main] Max cache overflow file size custom option: 0
2019-09-17T12:17:12.298+0800 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] MongoDB starting : pid=1733 port=27017 dbpath=/home/wangbojing/share/mongodb/db 64-bit host=ubuntu
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] db version v4.4.6
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] git version: 5776e3cbf9e7afe86e6b29e22520ffb6766e95d4
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1f 6 Jan 2014
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] allocator: tcmalloc
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] modules: none
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] build environment:
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten]     distmod: ubuntu1404
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten]     distarch: x86_64
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten]     target_arch: x86_64
2019-09-17T12:17:12.302+0800 I CONTROL  [initandlisten] options: { net: { bindIp: "0.0.0.0" }, storage: { dbPath: "/home/wangbojing/share/mongodb/db" } }
2019-09-17T12:17:12.302+0800 I STORAGE  [initandlisten] Detected data files in /home/wangbojing/share/mongodb/db created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.
2019-09-17T12:17:12.302+0800 I STORAGE  [initandlisten]
2019-09-17T12:17:12.303+0800 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2019-09-17T12:17:12.303+0800 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2019-09-17T12:17:12.303+0800 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=1455M,cache_overflow=(file_max=0M),session_max=20000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress),


wangbojing@ubuntu:~/share/mongodb$
wangbojing@ubuntu:~/share/mongodb$ 2019-09-17T12:17:13.149+0800 I STORAGE  [initandlisten] WiredTiger message [1568693833:149588][1733:0x7f605ea00a80], txn-recover: Main recovery loop: starting at 3/6016 to 4/256
```

```
2019-09-17T12:17:13.229+0800 I STORAGE  [initandlisten] WiredTiger message [1568693833:22913
4][1733:0x7f605ea00a80], txn-recover: Recovering log 3 through 4
```

确认启动：可以使用 **lsof -i:27017** 查看端口监听情况

```
lqf@ubuntu:~$ lsof -i:27017
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
mongod  75146  lqf   13u  IPv4 240765      0t0  TCP *:27017 (LISTEN)
```

如果想在后台运行，启动时只需添加 –fork 参数即可

可以在日志路径后面添加--logappend，防止日志被删除。

3.4 MongoDB 后台管理 Shell

如果你需要进入 MongoDB 后台管理，你需要先打开 mongodb 装目录的下的 bin 目录，然后执行 mongo 命令文件。MongoDB Shell 是 MongoDB 自带的交互式 Javascript shell,用来对 MongoDB 进行操作和管理的交互式环境。当你进入 mongoDB 后台后，它默认会链接到 test 文档（数据库）：

```
$ mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
2019-09-17T12:17:43.452+0800 I NETWORK  [listener] connection accepted from 127.0.0.1:52348
#1 (1 connection now open)
2019-09-17T12:17:43.453+0800 I NETWORK  [conn1] received client metadata from 127.0.0.1:5234
8 conn1: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client
", version: "4.4.6" }, os: { type: "Linux", name: "Ubuntu", architecture: "x86_64", version:
 "14.04" } }
Implicit session: session { "id" : UUID("db2a376e-77e9-4ebb-b0c9-1b111f07fc43") }
MongoDB server version: 4.4.6
Welcome to the MongoDB shell.
For interactive help, type "help".
```

由于它是一个 JavaScript shell，您可以运行一些简单的算术运算:

```
> 2+2
4
> 3+6
9
```

现在让我们插入一些简单的数据，并对插入的数据进行检索：

```
> db.zvoice.insert({x:10})
```

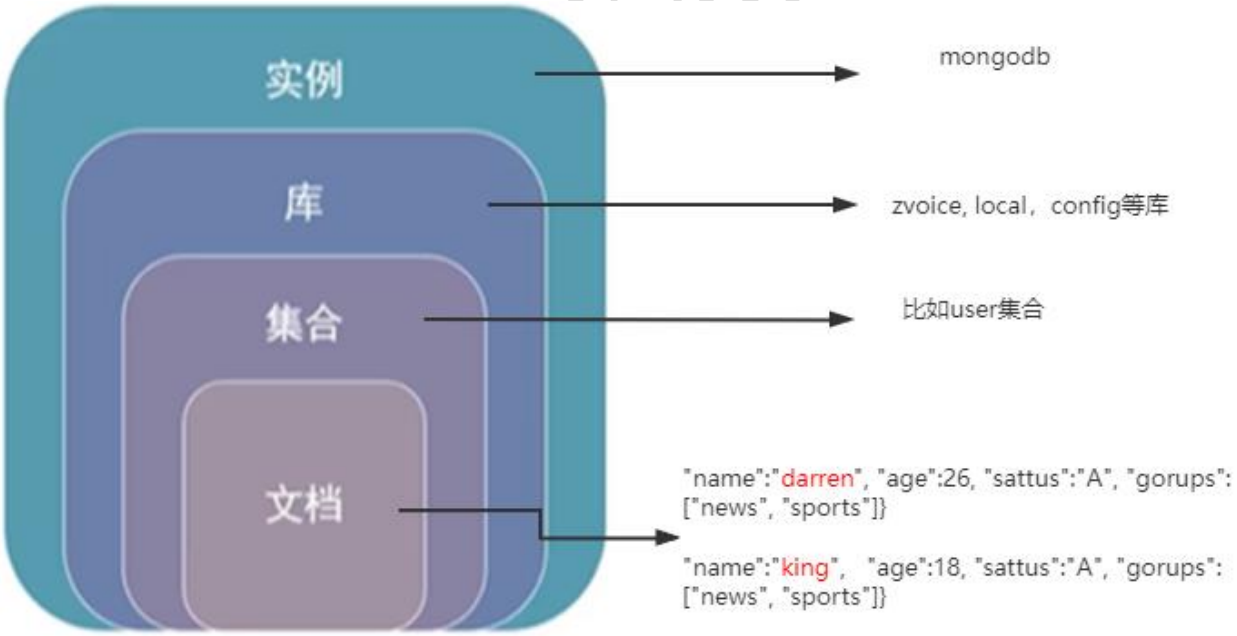
```
2019-09-17T12:20:04.154+0800 I STORAGE  [conn1] createCollection: test.zvoice with generated
 UUID: 32aee592-2475-4751-b38f-e24cb3750df6
WriteResult({ "nInserted" : 1 })
>
> db.zvoice.find()
{ "_id" : ObjectId("5d805ef415db2f66a53467a3"), "x" : 10 }
>
```

第一个命令将数字 10 插入到 zvoice 集合的 x 字段中。

```
> db.zerovoice.find()
{ "_id" : ObjectId("5d805ef415db2f66a53467a3"), "x" : 10 }
>
```

4 MongoDB 概念解析

4.1 层级结构



实例：系统上运行的进程及节点集，一个实例可以有多个库，默认端口 27017。

库：多个集合组成数据库，每个数据库都是独立的，有自己的用户、权限信息，独立的存储文件集合。

集合：即是一组文档的集合，集合内的文档结构可以不同。

文档：MongoDB 的最小数据单元，其基本概念为：多个键值对有序组合在一起的数据单元。示例如下所示：

db.user.insertOne({"name":"darren", "age":26, "sattus":"A", "gorups":["news", "sports"]})

db.user.insertOne({"name":"king", "age":18, "sattus":"A", "gorups":["news", "sports"]})

4.2 与关系型数据库的对比

可执行文件名对比

文件名	MongoDB	MySQL
服务	mongod	mysqld
客户端	mongo	mysql

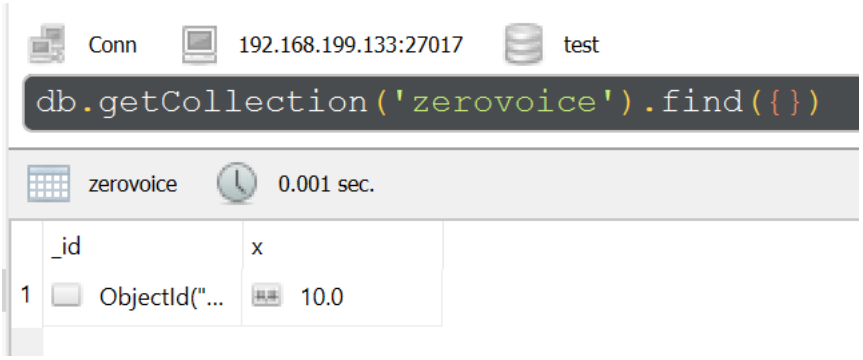
概念对比

MongoDB 术语/概念	SQL 术语/概念	解释/说明
database	database	数据库
collection	table	集合/数据库表
document	row	文档/数据记录行
field	column	域/数据字段
index	index	索引
embedding&linkding	table joins	MongoDB 用内嵌文档方式/表关联
primary key	primary key	MongoDB 自动将_id 字段设置为主键/主键
shard	partition	分片/分区
sharding key	partition key	分区键

4.3 基本概念

不管我们学习什么数据库都应该学习其中的基础概念，在 mongodb 中基本的概念是文档、集合、数据库，下面我们挨个介绍。下表将帮助您更容易理解 Mongo 中的一些概念：

通过下图实例，我们也可以更直观的了解 Mongo 中的一些概念：



数据库

一个 mongodb 中可以建立多个数据库。MongoDB 的默认数据库为"db"，该数据库存储在 data 目录中。MongoDB 的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。

"show dbs" 命令可以显示所有数据的列表。

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
>
```

执行 **"db"** 命令可以显示当前数据库对象或集合。

```
$ ./mongo
> db
test
>
```

运行"use"命令，可以连接到一个指定的数据库。

```
> use local
```

```
switched to db local
> db
local
>
```

以上实例命令中，"local" 是你要链接的数据库。

在下一个章节我们将详细讲解 MongoDB 中命令的使用。

数据库也通过名字来标识。**数据库名可以是满足以下条件的任意 UTF-8 字符串**。

- 不能是空字符串（""）。
- 不得含有' '（空格）、.、$、/、\和\0 (空字符)。
- 应全部小写。
- 最多 64 字节。

有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

- **admin**： 从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
- **local**: 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
- **config**: 当 Mongo 用于分片设置时，config 数据库在内部使用，用于保存分片的相关信息。

文档(Document)

文档是一组键值(key-value)对(即 BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。一个简单的文档例子如下：

```
{"site":"www.0voice.com", "name":"零声学院"}
```

需要注意的是：

- 文档中的键/值对是有序的。
- 文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个

嵌入的文档)。

- MongoDB 区分类型和大小写。
- MongoDB 的文档不能有重复的键。
- 文档的键是字符串。除了少数例外情况，键可以使用任意 UTF-8 字符。
- 文档对应于许多编程语言中的本机数据类型
- 嵌入式文档和数组减少了对连接的需求
- 动态模式支持流畅的多态性

文档键命名规范：

- 键不能含有\0 (空字符)。这个字符用来表示键的结尾。
- .和$有特别的意义，只有在特定环境下才能使用。
- 以下划线"_"开头的键是保留的(不是严格要求的)。

集合

集合就是 MongoDB 文档组，类似于 RDBMS（关系数据库管理系统：Relational Database Management System)中的表格。

集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

比如，我们可以将以下不同数据结构的文档插入到集合中：

```
{"site":"www.baidu.com"}
{"site":"www.google.com","name":"Google"}
{"site":"www.0voice.com","name":"零声学院","num":5}
```

当第一个文档插入时，集合就会被创建。

合法的集合名

- 集合名不能是空字符串""。
- 集合名不能含有\0 字符（空字符)，这个字符表示集合名的结尾。
- 集合名不能以"system."开头，这是为系统集合保留的前缀。
- 用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含，这是因为某些系统生成的集合中包含该字符。除非你要访问这种系统创建的集合，否则千万不要在名字里出现$。

如下实例：

```
db.col.findOne()
```

固定集合 capped collections

Capped collections 就是固定大小的 collection。它有很高的性能以及队列过期的特性(过期按照插入的顺序). 有点和 "RRD" 概念类似。Capped collections 是高性能自动的维护对象的插入顺序。它非常适合类似记录日志的功能和标准的 collection 不同，你必须要显式的创建一个 capped collection，指定一个 collection 的大小，单位是字节。collection 的数据存储空间值提前分配的。Capped collections 可以按照文档的插入顺序保存到集合中，而且这些文档在磁盘上存放位置也是按照插入顺序来保存的，所以当我们更新 Capped collections 中文档的时候，更新后的文档不可以超过之前文档的大小，这样话就可以确保所有文档在磁盘上的位置一直保持不变。由于 Capped collection 是按照文档的插入顺序而不是使用索引确定插入位置，这样的话可以提高增添数据的效率。MongoDB 的操作日志文件 oplog.rs 就是利用 Capped Collection 来实现的。要注意的是指定的存储大小包含了数据库的头信息。

```
db.createCollection("mycoll", {capped:true, size:100000})
```

- 在 capped collection 中，你能添加新的对象。
- 能进行更新，然而，对象不会增加存储空间。如果增加，更新就会失败 。
- 使用 Capped Collection 不能删除一个文档，可以使用 drop() 方法删除 collection 所有的行。
- 删除之后，你必须显式的重新创建这个 collection。
- 在 32bit 机器中，capped collection 最大存储为 1e9($1X10^9$)个字节。

固定集合可以声明 collection 的容量大小，其行为类似于循环队列。数据插入时，新文档会被插入到队列的末尾，如果队列已经被占满，那么最老的文档会被之后插入的文档覆盖。

固定集合特性：固定集合很像环形队列，如果空间不足，最早的文档就会被删除，为新的文档腾出空间。一般来说，固定集合适用于任何想要自动淘汰过期属性的场景。

固定集合应用场景

比如日志文件，聊天记录，通话信息记录等只需保留最近某段时间内的应用场景，都会使用到 MongoDB 的固定集合。

```
test1
> db.createCollection("mycoll", {capped:true, size:1000, max:2})
{ "ok" : 1 }
> db.mycoll.insert({"name":"darren1"})
WriteResult({ "nInserted" : 1 })
> db.mycoll.insert({"name":"darren2"})
WriteResult({ "nInserted" : 1 })
> db.mycoll.find()
{ "_id" : ObjectId("60d722afdc26487e82c4e916"), "name" : "darren1" }
{ "_id" : ObjectId("60d722b7dc26487e82c4e917"), "name" : "darren2" }
> db.mycoll.insert({"name":"darren3"})
WriteResult({ "nInserted" : 1 })
> db.mycoll.find()
{ "_id" : ObjectId("60d722b7dc26487e82c4e917"), "name" : "darren2" }
{ "_id" : ObjectId("60d722c6dc26487e82c4e918"), "name" : "darren3" }
>
```

固定集合的优点

1. 写入速度提升。固定集合中的数据被顺序写入磁盘上的固定空间，所以，不会因为其他集合的一些随机性的写操作而“中断”，其写入速度非常快（不建立索引，性能更好）。
2. 固定集合会自动覆盖掉最老的文档，因此不需要再配置额外的工作来进行旧文档删除。设置 Job 进行旧文档的定时删除容易形成性能的压力毛刺。

固定集合非常适合用于记录日志等场景。

固定集合的创建

不同于普通集合，固定集合必须在使用前显式创建。

例如，创建固定集合 coll_testcapped，大小限制为 1024 个字节。

db.createCollection("coll_testcapped",{capped:true,size:1024});

注意事项

（1） 固定集合创建之后就不可以改变，只能将其删除重建。

（2） 普通集合可以使用 convertToCapped 转换固定集合，但是固定集合不可以转换为普通集合。

（3） 创建固定集合，为固定集合指定文档数量限制时（指参数 max），必须同时指定固定集合的大小（指参数 size）。不管先达到哪一个限制，之后插入的新文档都会把最老的文档移除集合。

（4） 使用 convertToCapped 命令将普通集合转换固定集合时，既有的索引会丢失，需要手动创建。并且，此转换命令没有限制文档数量的参数（即没有 max 的参数选项）。

（5） 不可以对 固定集合 进行分片。

（6） 对固定集合中的文档可以进行更新(update)操作，但更新不能导致文档的 Size 增长或缩小，否则更新失败。

假如集合中有一个 key,其 value 对应的数据长度为 100 个字节，如果要更新这个 key 对应的 value，

更新后的值也必须为 100 个字节，大于 100 个字节不可以，小于 100 个字节也不可以。

报错信息为：Cannot change the size of a document in a capped collection : XXXX（XXXX 代表某个数据字） !=XXXX。

（7） 不可以对固定集合执行删除文档操作，但可以删除整个集合。

删除文档时，报错信息为：cannot remove from a capped collection：XXXX

（8） 还有一定需要注意，对集合估算 size 时，不要依据集合的 storageSize ，而是依据集合的 size。storageSize 是 wiredTiger 存储引擎采用高压缩算法压缩后的。

元数据

元数据是一个预留空间，在对数据库或应用程序结构执行修改时，其内容可以由数据库自动更新。元数据是系统中各类数据描述的集合，是执行详细的数据收集与数据分析的主要途径。

元数据最重要的作用是作为一个分析阶段的工具，任何字典最重要的操作是查询，在结构化分析中，元数据的作用是给每一个结点加上定义与说明。换句话说，数据流图上所有节点的定义和解释的集合就是元数据，而且在元数据中建立严密一致的定义有助于提高需求分析人员和用户沟通的效率。

数据库的信息是存储在集合中。它们使用了系统的命名空间：

```
dbname.system.*
```

在 MongoDB 数据库中名字空间 <dbname>.system.* 是包含多种系统信息的特殊集合(Collection)，如下:

集合命名空间	描述
dbname.system.namespaces	列出所有名字空间。
dbname.system.indexes	列出所有索引。
dbname.system.profile	包含数据库概要(profile)信息。
dbname.system.users	列出所有可访问数据库的用户。
dbname.local.sources	包含复制对端（slave）的服务器信息和状态。

对于修改系统集合中的对象有如下限制。在{{system.indexes}}插入数据，可以创建索引。但除此之外该表信息是不可变的(特殊的 drop index 命令将自动更新相关信息)。{{system.users}}是可修改的。{{system.profile}}是可删除的。

4.4 MongoDB 数据类型

常用数据类型

下表为 MongoDB 中常用的几种数据类型， json 基础上的扩展。

数据类型	描述
String	字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。
Integer	整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。
Boolean	布尔值。用于存储布尔值（真/假）。
Double	双精度浮点值。用于存储浮点值。
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	用于将数组或列表或多个值存储为一个键。
Timestamp	时间戳。记录文档修改或添加的具体时间。
Object	用于内嵌文档。
Null	用于创建空值。
Symbol	符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。

Date	日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。
Object ID	对象 ID。用于创建文档的 ID。
Binary Data	二进制数据。用于存储二进制数据。
Code	代码类型。用于在文档中存储 JavaScript 代码。
Regular expression	正则表达式类型。用于存储正则表达式。

下面说明下几种重要的数据类型。

ObjectId（不自己指定，默认生成）（自己去生成对应的 ID）

ObjectId 类似唯一主键，可以很快的去生成和排序，包含 12 bytes，含义是：

- 前 4 个字节表示创建 **unix** 时间戳,格林尼治时间 **UTC** 时间，比北京时间晚了 8 个小时
- 接下来的 3 个字节是机器标识码
- 紧接的两个字节由进程 id 组成 PID
- 最后三个字节是随机数（1 秒最多 1000 个不重复的数据，高并发 1000 的写入 qps 不行的）

MongoDB 中存储的文档必须有一个 _id 键。这个键的值可以是任何类型的，默认是个 ObjectId 对象，由于 ObjectId 中保存了创建的时间戳，所以你不需要为你的文档保存时间戳字段，你可以通过 getTimestamp 函数来获取文档的创建时间:

```
> var newObject = ObjectId()
> newObject.getTimestamp()
ISODate("2019-09-17T04:42:12Z")
```

ObjectId 转为字符串

```
> newObject.str
5d80642415db2f66a53467a4
```

```
zerovoice  0.000GB
> var newObject = ObjectId()
> var newObject = ObjectId("60d5dfbd3531f8ada472907b")
> newObject.getTimestamp()
ISODate("2021-06-25T13:53:01Z")
> var newObject = ObjectId()
> newObject.getTimestamp()
ISODate("2021-06-25T14:02:07Z")
> var newObject = ObjectId("60d5dfbd3531f8ada472907b")
> newObject.str
60d5dfbd3531f8ada472907b
>
```

字符串

BSON 字符串都是 UTF-8 编码。

时间戳

BSON 有一个特殊的时间戳类型用于 MongoDB 内部使用，与普通的 日期 类型不相关。 时间戳值是一个 64 位的值。其中：

- 前 32 位是一个 time_t 值（与 Unix 新纪元相差的秒数）
- 后 32 位是在某秒中操作的一个递增的序数

在单个 mongod 实例中，时间戳值通常是唯一的。

在复制集中， oplog 有一个 ts 字段。这个字段中的值使用 BSON 时间戳表示了操作时间。

BSON 时间戳类型主要用于 MongoDB 内部使用。在大多数情况下的应用开发中，你可以使用 BSON 日期类型。

日期

表示当前距离 Unix 新纪元（1970 年 1 月 1 日）的毫秒数。日期类型是有符号的, 负数表示 1970 年之前的日期。

```
> var date1 = new Date()     //格林尼治时间
> date1
ISODate("2019-09-17T04:47:39.919Z")
> typeof mydate1
object
> var mydate2 = ISODate() //格林尼治时间
> mydate2
ISODate("2019-09-17T04:48:21.854Z")
> typeof mydate2
```

```
object
```

这样创建的时间是日期类型，可以使用 JS 中的 Date 类型的方法。

返回一个时间类型的字符串：

```
> var date1str = date1.toString()
> date1str
Tue Sep 17 2019 12:47:39 GMT+0800 (CST)
> typeof date1str
string
```

或者

```
> Date()
Tue Sep 17 2019 12:50:16 GMT+0800 (CST)
```

5 MongoDB 数据库操作

操作命令	作用
use DATABASE_NAME	如果数据库不存在，则创建数据库，否则切换到指定数据库。
show dbs	查看所有数据库
db	显示当前数据库
db.dropDatabase()	删除当前数据库

系统数据库有：

- admin：存储用户信息
- local： 存储本地数据
- config：存储分片信息

更多命令参考：https://docs.mongodb.com/manual/reference/method/js-database/

MongoDB 创建数据库

MongoDB 创建数据库的语法格式如下：

```
use DATABASE_NAME
```

如果数据库不存在，则创建数据库，否则切换到指定数据库。

以下实例我们创建了数据库 zvoice：

```
> use zvoice
switched to db zvoice
> db
zvoice
```

如果你想查看所有数据库，可以使用 **show dbs** 命令：

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
>
```

可以看到，我们刚创建的数据库 zvoice 并不在数据库的列表中， 要显示它，我们需要向 zvoice 数据库插入一些数据。

```
> db.zvoice.insert({"name":"零声学院"})
WriteResult({ "nInserted" : 1 })
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
zvoice  0.000GB
```

MongoDB 中默认的数据库为 test，如果你没有创建新的数据库，集合将存放在 test 数据库中。

MongoDB 删除数据库

MongoDB 删除数据库的语法格式如下：

```
db.dropDatabase()
```

删除当前数据库，默认为 test，你可以使用 db 命令查看当前数据库名。

以下实例我们删除了数据库 zvoice。

首先，查看所有数据库：

```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
zvoice  0.000GB
```

接下来我们切换到数据库 zvoice：

```
> use zvoice
switched to db zvoice
>
```

执行删除命令：

```
> use zvoice
switched to db zvoice
> db.dropDatabase()
{ "dropped" : "zvoice", "ok" : 1 }
```

最后，我们再通过 show dbs 命令数据库是否删除成功：

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

6 MongoDB 集合操作

操作命令	作用
db.createCollection	创建集合
show collections	查看当前库有多少集合
show tables	查看当前库有多少集合
db.collection.drop()	删除集合
db.collection1.renameCollection(" collection2")	将集合名 collection1 命名为 collection2.

更多参考：https://docs.mongodb.com/manual/reference/method/db.createCollection

MongoDB 创建集合

本章节我们为大家介绍如何使用 MongoDB 来创建集合。MongoDB 中使用 **createCollection()** 方法来创建集合。

语法格式：

```
db.createCollection(name, options)
```

参数说明：

- name: 要创建的集合名称
- options: 可选参数, 指定有关内存大小及索引的选项

options 可以是如下参数：

字段	类型	描述
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 时，必须指定 size 参数。**
size	数值	（可选）为固定集合指定一个最大值，以千字节计（KB）。 **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。

在 test 数据库中创建 zvoice 集合：

```
> use zvoice
switched to db test
> db.createCollection("zvoice")
{ "ok" : 1 }
> db.createCollection("zvoice1")
{ "ok" : 1 }
```

如果要查看已有集合，可以使用 `show collections` 或 `show tables` 命令：

```
> show collections
zvoice
zvoice1
```

下面是带有几个关键参数的 createCollection() 的用法：

创建固定集合 mycol，整个集合空间大小 6142800 KB，文档最大个数为 10000 个。

```
> db.createCollection("col_capped", {capped: true, size: 6142800, max: 10000})
显示
{
        "ok" : 1
}
> show tables
```

在 MongoDB 中，你不需要创建集合。当你插入一些文档时，MongoDB 会自动创建集合。

```
> db.col2.insert({"name": "零声学院"})
WriteResult({ "nInserted" : 1 })
>
> show tables
zvoice
zvoice1
col2
col_capped...
```

MongoDB 删除集合

本章节我们为大家介绍如何使用 MongoDB 来删除集合。MongoDB 中使用 drop() 方法来删除集合。

语法格式：

```
db.collection.drop()
```

参数说明：

- 无

返回值

- 如果成功删除选定集合，则 drop() 方法返回 true，否则返回 false。

在数据库 mydb 中，我们可以先通过 **show collections** 命令查看已存在的集合：

```
> use zvoice
switched to db zvoice
> show tables
zvoice
zvoice1
col2
col_capped
>
```

接着删除集合 col2 :

```
> db.col2.drop()
true
```

通过 show tables 再次查看数据库 zvoice 中的集合：

```
> show tables
zvoice
zvoice1
Col_capped
```

从结果中可以看出 col2 集合已被删除。

MongoDB 重命名集合

通过 db.collection1.renameCollection(" collection2")重命名集合名。

```
> show collections
    zvoice
    zvoice1
    col_capped
> db.zvoice.renameCollection("darren")
      { "ok" : 1 }
> show collections
    col_capped
    darren
    zvoice1
```

7 MongoDB 文档操作

对应的函数说明：https://docs.mongodb.com/manual/reference/method/js-collection/

英文：https://docs.mongodb.com/manual/crud/

中文：https://docs.mongoing.com/mongodb-crud-operations

7.1 文档增删改查

命令操作	作用
db.collection.insert()	db.collection.insert()将单个文档或多个文档插入到集合中
db.collection.insertOne()	插入文档，3.2 版中的新功能
db.collection.insertMany()	插入多个文档,3.2 版中的新功能
db.collection.update	更新或替换与指定过滤器匹配的单个文档，或更新与指定过滤器匹配的所有文档。 默认情况下，db.collection.update()方法更新单个文档。要更新多个文档，请使用 multi 选项。
db.collection.updateOne(<filter>, <update>, <options>)	即使多个文档可能与指定的过滤器匹配，最多更新与指定的过滤器匹配的单个文档。 *3.2 版中的新功能*
db.collection.updateMany(<filter>, <update>, <options>)	更新所有与指定过滤器匹配的文档。 *3.2 版中的新功能*
db.collection.replaceOne(<filter>,	即使多个文档可能与指定过滤器匹配，也最多替换一个

<update>, <options>)	与指定过滤器匹配的文档。
db.collection.remove()	删除单个文档或与指定过滤器匹配的所有文档
db.collection.deleteOne()	即使多个文档可能与指定过滤器匹配，也最多删除一个与指定过滤器匹配的文档。 *3.2 版中的新功能*
db.collection.deleteMany()	删除所有与指定过滤器匹配的文档。 *3.2 版中的新功能*
db.collection.find(query, projection)	查询文档
db.collection.findOne()	

7.1.1 MongoDB 插入文档

本章节中我们将向大家介绍如何将数据插入到 MongoDB 的集合中。文档的数据结构和 JSON 基本一样。所有存储在集合中的数据都是 BSON 格式。BSON 是一种类似 JSON 的二进制形式的存储格式，是 Binary JSON 的简称。MongoDB 使用 insert() 或 save() 方法向集合中插入文档，语法如下：

db.collection.insertOne() 将单个文档插入到集合中。

db.collection.insertMany()将多个文件插入集合中。

db.collection.insert()将单个文档或多个文档插入到集合中。

以下文档可以存储在 MongoDB 的 zvoice 数据库 的 col 集合中：

```
> db.col.insert({title:'MongoDB 教程', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100})
这个是结果：WriteResult({ "nInserted" : 1 })
```

以上实例中 col 是我们的集合名，如果该集合不在该数据库中， MongoDB 会自动创建该集合并插入文档。

查看已插入文档：

```
> db.col.find()
显示
{ "_id" : ObjectId("5d80715e019abe974dac5164"), "title" : "MongoDB 教程", "description" : "MongoDB 是 DB", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
```

我们也可以将数据定义为一个变量，如下所示：

```
> document=({title:'MongoDB 教程', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voi
ce.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100})
{
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是 DB",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
> db.col.insert(document)
WriteResult({ "nInserted" : 1 })
```

练习

插入单个文档

db.col.insert({title:'MongoDB 教程 1', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100})

插入多个文档

db.col.insert([{title:'MongoDB 教程 2', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100}, {title:'MongoDB 教程 3', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100}

])

查找文档

db.col.find({ title: "MongoDB 教程 1" })

插入单个文档

db.col.insertOne({title:'MongoDB 教程 4', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100})

插入多个文档

db.col.insertMany([{title:'MongoDB 教程 5', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100}, {title:'MongoDB 教程 6', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100}])

7.1.2 MongoDB 更新文档

此页面使用以下 mongo shell 方法：

- db.collection.updateOne(<filter>, <update>, <options>)
- db.collection.updateMany(<filter>, <update>, <options>)
- db.collection.replaceOne(<filter>, <update>, <options>)

此页面上的示例使用库存收集。 要创建和/或填充清单集合，请运行以下命令：

此页上的示例使用 **inventory** 集合。要创建和/或填充 **inventory** 集合，请运行以下操作：

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },
   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },
   { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status:
"A" }
] );
```

更新集合中的文档

为了更新文档，MongoDB 提供了更新操作符（例如$set）来修改字段值。

要使用更新运算符，请将以下形式的更新文档传递给更新方法：

```
{
        <update operator>: { <field1>: <value1>, ... },
        <update operator>: { <field2>: <value2>, ... },
        ...
}
```

如果字段不存在，则某些更新操作符（例如$set）将创建该字段。 有关详细信息，请参见各个更新操作员参考。

更新单个文档 updateOne

下面的示例在 inventory 集合上使用 db.collection.updateOne()方法更新项目等于“ paper”的第一个文档：

```
db.inventory.updateOne(
    { item: "paper" },
    {
        $set: { "size.uom": "cm", status: "P" },
        $currentDate: { lastModified: true }
    }
)
```

```
db.inventory.updateOne(
    { item: "paper" },
    {
        $set: { "size.uom": "cm", status: "P" },
        $currentDate: { lastModified: true }
    }
)
```

查询条件

修改操作

更新操作：

- 使用$set 运算符将 size.uom 字段的值更新为“ cm”，将状态字段的值更新为“ P”，
- 使用$currentDate 运算符将 lastModified 字段的值更新为当前日期。 如果 lastModified 字段不存在，则$currentDate 将创建该字段。 有关详细信息，请参见$currentDate。

更新多个文档 updateMany

3.2 版中的新功能

以下示例在清单集合上使用 db.collection.updateMany()方法来更新数量小于 50 的所有文档：

```
db.inventory.updateOne(
      { "qty": { $lt: 50 } },
      {
          $set: { "size.uom": "in", status: "P" },
          $currentDate: { lastModified: true }
      }
  )
```

```
  db.inventory.updateMany(
      { "qty": { $lt: 50 } },
      {
          $set: { "size.uom": "in", status: "P" },
          $currentDate: { lastModified: true }
      }
  )
```

更新操作：

- 使用$set 运算符将 size.uom 字段的值更新为“ in”，将状态字段的值更新为“ P”.
- 使用 $currentDate 运算符将 lastModified 字段的值更新为当前日期。如果 lastModified 字段不存在，则$currentDate 将创建该字段。有关详细信息，请参见 $currentDate 。

更换文档 replaceOne

要替换_id 字段以外的**文档的全部内容**，

请将一个全新的文档作为第二个参数传递给 db.collection.replaceOne()。

当替换一个文档时，替换文档必须只包含字段/值对;即不包括更新操作符表达式。

替换文档可以具有与原始文档不同的字段。在替换文档中，由于_id 字段是不可变的，因此可以省略_id 字段。但是，如果您确实包含_id 字段，则它必须与当前值具有相同的值。

下面的示例替换了 inventory 集合中的第一个文件，其中项为"paper":

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "darren", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
)
```

7.1.3 MongoDB 查询文档

MongoDB 查询文档使用 find() 方法。find() 方法以非结构化的方式来显示所有文档。

MongoDB 查询数据的语法格式如下：

```
db.collection.find(query, projection)
```

- **query** ： 可选，使用查询操作符指定查询条件
- **projection** ： 可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可（默认省略）。

```
>
> db.inventory.find({item:"paper"}, {_id:0,item:1, instock:1})
{ "item" : "paper", "instock" : [ { "warehouse" : "A", "qty" : 60 }, {
>
```

如果你需要以易读的方式来读取数据，可以使用 pretty() 方法。

db.demo.find() 返回数据时，每一条会占取一行，不便于阅读；而使用 db.find().pretty()方法时显示美观，便于阅读。

语法格式如下：

```
>db.col.find().pretty()
```

pretty() 方法以格式化的方式来显示所有文档。

以下实例我们查询了集合 col 中的数据：

```
> db.col.find().pretty()
{
        "_id" : ObjectId("5d807497019abe974dac5167"),
        "title" : "MongoDB",
        "description" : "MongoDB 是 DB",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

除了 find() 方法之外，还有一个 findOne() 方法，它只返回一个文档。

这个页面提供了使用 mongo shell 中的 db.collection.find()方法的查询操作示例。此页上的示例使用 inventory 集合。要填充 inventory 集合，请运行以下操作：

```
db.inventory.remove({})

db.inventory.insertMany([
{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

选择集合中的所有文档

要选择集合中的所有文档，请将空文档作为查询过滤器参数传递给 **find** 方法。 查询过滤器参数确定选择条件：

```
db.inventory.find( {} )
```

此操作对应于以下 SQL 语句：

```
SELECT * FROM inventory
```

有关该方法的语法的更多信息，请参见 find()。

指定平等条件

要指定相等条件，请在查询筛选文档使用**<field>：<value>**表达式：

```
{ <field1>: <value1>, ... }
```

下面的示例从 inventory 中选择状态等于**" D"**的所有文档：

```
db.inventory.find( { status: "D" } )
```

此操作对应于以下 SQL 语句：

```
SELECT * FROM inventory WHERE status = "D"
```

使用查询运算符指定条件 or

查询过滤器文档可以使用查询运算符以以下形式指定条件：

```
{ <field1>: { <operator1>: <value1> }, ... }
```

下面的例子从状态等于**" A"**或**" D"**的 **inventory** 集合中检索所有文档:

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

Note：尽管可以使用$or 操作符表示此查询，但在对同一字段执行相等性检查时，请使用 $in 操作符而不是$or 操作符。

该操作对应于以下 SQL 语句：

```
SELECT * FROM inventory WHERE status in ("A", "D")
```

有关 MongoDB 查询运算符的完整列表，请参阅查询和投影运算符文档。

指定和条件 and

复合查询可以为集合文档中的多个字段指定条件。逻辑和连词隐式地连接复合查询的子句，以便查询在集合中选择符合所有条件的文档。 **有$符号都是 mongod 的关键字**

下面的示例 inventory 状态为**"A"**且数量小于($lt) 30 的库存集合中的所有文档:

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

该操作对应于以下 SQL 语句:

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

有关其他 MongoDB 比较运算符，请参阅比较运算符 。

指定或条件

使用$or 操作符，可以指定一个复合查询，用逻辑 **OR** 连词连接每个子句，以便查询在集合中选择至少匹配一个条件的文档。

下面的示例 **retrieve** 状态为**“A”**或 **qty** 小于($lt)30 的集合中的所有文档:

```
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```

该操作对应于以下 SQL 语句:

```
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

[success] Note

使用比较运算符的查询需要使用括号类型。

指定和以及或条件

在下面的例子中，复合查询文档选择状态为**“A”**且 **qty** 小于($lt) 30 或 **item** 以字符 **p** 开头的集合中的所有文档:

```
db.inventory.find( {
    status: "A",
```

```
        $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
    })
```

该操作对应于以下 SQL 语句:

```
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

[success] Note

MongoDB 支持正则表达式$regex 查询来执行字符串模式匹配。

MongoDB 投影查询

mongodb 投影意义是只选择需要的数据，而不是选择整个一个文档的数据。如果一个文档有 5 个字段，只需要显示 3 个，只从中选择 3 个字段。

MongoDB 的 find()方法，解释了 MongoDB 中查询文档接收的第二个可选的参数是要检索的字段列表。在 MongoDB 中，当执行 find()方法，那么它会显示一个文档的所有字段。要限制这一点，需要设置字段列表值为 1 或 0。**1 是用来显示字段，而 0 被用来隐藏字段。**

语法

find()方法的基本语法如下：

```
db.COLLECTION_NAME.find({},{KEY:1})
```

范例 1：

```
> db.inventory.find( { status: "D" })
{ "_id" : ObjectId("6239752216df9538c3ee1a87"), "item" : "planner", "qty" : 75,
"size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
> db.inventory.find( { status: "D" }, {_id:0})
{ "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" },
"status" : "D" }
> db.inventory.find( { status: "D" }, {_id:0, item:1})
{ "item" : "planner" }
> db.inventory.find( { status: "D" }, {_id:0, item:0})
{ "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
> db.inventory.find( { status: "D" }, {_id:0, item:1})
{ "item" : "planner" }
```

附加查询教程

有关其他查询示例，请参见：

- Query on Embedded/Nested Documents
- Query an Array

- Query an Array of Embedded Documents
- Project Fields to Return from Query
- Query for Null or Missing Fields

7.1.4 MongoDB 删除文档

在前面的几个章节中我们已经学习了 MongoDB 中如何为集合添加数据和更新数据。在本章节中我们将继续学习 MongoDB 集合的删除。MongoDB remove()函数是用来移除集合中的数据。MongoDB 数据更新可以使用 update()函数。在执行 remove()函数前先执行 find()命令来判断执行的条件是否正确，这是一个比较好的习惯。

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。

以下文档我们执行两次插入操作：

```
> db.col.insert({title: 'MongoDB 教程',
    description: 'MongoDB 是一个 Nosql 数据库',
    by: '零声学院',
    url: 'http://www.0voice.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
})
```

使用 find() 函数查询数据：

```
> db.col.find()
显示
```

```
    { "_id" : ObjectId("56066169ade2f21f36b03137"), "title" : "MongoDB 教程", "description" :
 "MongoDB 是一个 Nosql 数据库", "by" : "零声学院", "url" : "http://www.0voice.com", "tags" :
[ "mongodb", "database", "NoSQL" ], "likes" : 100 }

    { "_id" : ObjectId("5606616dade2f21f36b03138"), "title" : "MongoDB 教程", "description" :
 "MongoDB 是一个 Nosql 数据库", "by" : "零声学院", "url" : "http://www. zvoice.com", "tags" :
[ "mongodb", "database", "NoSQL" ], "likes" : 100 }
```

接下来我们移除 title 为 'MongoDB 教程' 的文档：

```
    > db.col.remove({'title':'MongoDB 教程'})
    WriteResult({ "nRemoved" : 2 })           # 删除了两条数据
    > db.col.find()
    ……                                      # 没有数据
```

如果你只想删除第一条找到的记录可以设置 justOne 为 1，如下所示：

```
    >db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

如果你想删除所有数据，可以使用以下方式（类似常规 SQL 的 truncate 命令）：

```
    >db.col.remove({})
    >db.col.find()
    >
```

此页面使用以下 mongo shell 方法

- db.collection.deleteMany()
- db.collection.deleteOne()

此页面上的示例使用 inventory 收集。 要填充 inventory 收集，请运行以下命令：

```
db.inventory.deleteMany({})

db.inventory.insertMany( [
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status:
"A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status:
"P" },
```

```
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status:
"D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },
status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },
status: "A" },
] );
```

删除所有文档（注意和删除集合的区别）

要删除集合中的所有文档，请将空的 filter 文档{}传递给 db.collection.deleteMany() 方法。

以下示例从 **inventory** 收集中删除所有文档：

```
db.inventory.deleteMany({})
```

该方法返回具有操作状态的文档。 有关更多信息和示例，请参见 deleteMany().

删除所有符合条件的文档

您可以指定标准或过滤器，以标识要删除的文档。 filter 使用与读取操作相同的语法。

要指定相等条件，请在查询过滤器文档:中使用**<field>：<value>**表达式：

```
{ <field1>: <value1>, ... }
```

查询过滤器文档可以使用查询操作符 以以下形式指定条件:

```
{ <field1>: { <operator1>: <value1> }, ... }
```

要删除所有符合删除条件的文档，请将 filter 参数传递给 deleteMany()方法。

以下示例从状态字段等于**“ A”**的 **inventory** 集合中删除所有文档：

```
db.inventory.deleteMany({ status : "A" })
```

该方法返回具有操作状态的文档。 有关更多信息和示例，请参见 deleteMany().

仅删除一个符合条件的文档

要删除最多一个与指定过滤器匹配的文档(即使多个文档可以与指定过滤器匹配)，请使用 db.collection.deleteOne()方法。

下面的示例删除状态为**“ D”**的第一个文档：

```
db.inventory.deleteOne( { status: "D" } )
```

删除行为

索引

即使从集合中删除所有文档，删除操作也不会删除索引。

原子性

MongoDB 中的所有写操作都是**单个文档级别的原子操作**。 有关 MongoDB 和原子性的更多信息，请参见原子性和事务。

Mongodb 不是集合锁，是文档锁，对应 mysql 的行锁。

7.1.5 MongoDB 条件语句查询与 RDBMS(Relational Database Management System)

Where 语句比较

如果你熟悉常规的 SQL 数据，通过下表可以更好的理解 MongoDB 的条件语句查询：

操作	格式	范例	RDBMS 中的类似语句
等于	{<key>:<value>}	db.col.find({"by":"零声学院"}).pretty()	where by = '零声学院'
小于	{<key>:{$lt:<value>}}	db.col.find({"likes":{$lt:50}}).pretty()	where likes < 50
小于或等于	{<key>:{$lte:<value>}}	db.col.find({"likes":{$lte:50}}).pretty()	where likes <= 50
大于	{<key>:{$gt:<value>}}	db.col.find({"likes":{$gt:50}}).pretty()	where likes > 50
大于或等于	{<key>:{$gte:<value>}}	db.col.find({"likes":{$gte:50}}).pretty()	where likes >= 50
不等于	{<key>:{$ne:<value>}}	db.col.find({"likes":{$ne:50}}).pretty()	where likes != 50

great little equal

db.col.insertMany([{title:'MongoDB 教程 5', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100}, {title:'MongoDB 教程 6', description: 'MongoDB 是 DB', by: '零声学院', url: 'www.0voice.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100}])

MongoDB AND 条件

MongoDB 的 find() 方法可以传入多个键(key)，每个键(key)以逗号隔开，即常规 SQL 的 AND 条件。语法格式如下：

```
> db.col.find({key1:value1, key2:value2}).pretty()
```

以下实例通过 **by** 和 **title** 键来查询 **零声学院** 中 **MongoDB 教程** 的数据

```
> db.col.find({"by":"零声学院", "title":"MongoDB 教程"}).pretty()
{
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "零声学院",
        "url" : "http://www.0voice.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

以上实例中类似于 WHERE 语句：**WHERE by='零声学院' AND title='MongoDB 教程'**

再比如：

db.col.find({"by":"零声学院", "_id":ObjectId("60d43ef2f9599cfcf4985d93")}).pretty()

MongoDB OR 条件

MongoDB OR 条件语句使用了关键字 **$or**,语法格式如下：

```
>db.col.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

以下实例中，我们演示了查询键 **by** 值为 零声学院 或键 **title** 值为 **MongoDB 教程** 的文档。

```
>db.col.find({$or:[{"by":"零声学院"},{"title": "MongoDB 教程"}]}).pretty()
{
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "零声学院",
        "url" : "http://www.0voice.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
>
```

MongoDB AND 和 OR 联合使用

以下实例演示了 AND 和 OR 联合使用，类似常规 SQL 语句为： 'where likes>50 AND (by = '零声学院' OR title = 'MongoDB 教程')'

```
>db.col.find({"likes": {$gt:50}, $or: [{"by": "零声学院"},{"title": "MongoDB 教程"}]}).pretty
()
{
```

```
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "零声学院",
        "url" : "http://www.0voice.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

7.2 MongoDB 条件操作符

条件操作符用于比较两个表达式并从 mongoDB 集合中获取数据。在本章节中，我们将讨论如何在 MongoDB 中使用条件操作符。MongoDB 中条件操作符有：

- (>) 大于 - $gt
- (<) 小于 - $lt
- (>=) 大于等于 - $gte
- (<=) 小于等于 - $lte

我们使用的数据库名称为"zvoice" 我们的集合名称为"col"，以下为我们插入的数据。

为了方便测试，我们可以先使用以下命令清空集合 "col" 的数据：

```
db.col.remove({})
```

插入以下数据

```
> db.col.insert({title: 'c++教程', description:'c++是最好的语言', by: '零声学院', url: 'www.0voice.com', tags:['cpp'] , likes: 150})
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'c langage', description:'c 语言很不错', by: '零声学院', url: 'www.0voice.com', tags:['cpp'], likes: 100})
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'java', description:'java good', by: '零声学院', url: 'www.0voice.com', tags:['cpp'], likes: 80})
WriteResult({ "nInserted" : 1 })
```

```
> db.col.insert({title: 'php', description:'script langage', by: '零声学院', url: 'www.0voic
e.com', tags:['cpp'], likes: 90})
WriteResult({ "nInserted" : 1 })
```

使用 find()命令查看数据：

```
> db.col.find().pretty()
{
        "_id" : ObjectId("5d807c11019abe974dac516a"),
        "title" : "c++教程",
        "description" : "c++是最好的语言",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 150
}
{
        "_id" : ObjectId("5d807c6e019abe974dac516b"),
        "title" : "c langage",
        "description" : "c 语言很不错",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 100
}
{
        "_id" : ObjectId("5d807c9b019abe974dac516c"),
        "title" : "java",
        "description" : "java good",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
```

```
        "likes" : 80
}
{
        "_id" : ObjectId("5d807cb0019abe974dac516d"),
        "title" : "php",
        "description" : "script langage",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 90
}
```

MongoDB (>) 大于操作符 - $gt

如果你想获取 "col" 集合中 "likes" 大于 100 的数据，你可以使用以下命令：

```
> db.col.find({likes:{$gt:100}})
```

类似于 SQL 语句：

```
Select * from col where likes > 100;
```

输出结果：

```
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程", "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 150 }
```

MongoDB（>=）大于等于操作符 - $gte

如果你想获取"col"集合中 "likes" 大于等于 100 的数据，你可以使用以下命令：

```
> db.col.find({likes:{$gte:100}})
```

类似于 SQL 语句：

```
Select * from col where likes >=100;
```

输出结果：

```
> db.col.find({likes : {$gte : 100}})
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程", "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 150 }
{ "_id" : ObjectId("5d807c6e019abe974dac516b"), "title" : "c langage", "description" : "c 语言很不错", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 100 }
>
```

MongoDB (<) 小于操作符 - $lt

如果你想获取"col"集合中 "likes" 小于 150 的数据，你可以使用以下命令：

```
> db.col.find({likes:{$lt:100}})
```

类似于 SQL 语句：

```
Select * from col where likes < 100;
```

输出结果：

```
> db.col.find({likes : {$lt : 100}})
{ "_id" : ObjectId("5d807c9b019abe974dac516c"), "title" : "java", "description" : "java good", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 80 }
{ "_id" : ObjectId("5d807cb0019abe974dac516d"), "title" : "php", "description" : "script langage", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 90 }
```

MongoDB (<=) 小于等于操作符 - $lte

如果你想获取"col"集合中 "likes" 小于等于 150 的数据，你可以使用以下命令：

```
> db.col.find({likes:{$lte:100}})
```

类似于 SQL 语句：

```
Select * from col where likes <= 100;
```

输出结果：

```
> db.col.find({likes : {$lte : 100}})
{ "_id" : ObjectId("5d807c6e019abe974dac516b"), "title" : "c langage", "description" : "c 语言很不错", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 100 }
{ "_id" : ObjectId("5d807c9b019abe974dac516c"), "title" : "java", "description" : "java good ", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 80 }
{ "_id" : ObjectId("5d807cb0019abe974dac516d"), "title" : "php", "description" : "script lan gage", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 90 }
```

MongoDB 使用 (<) 和 (>) 查询 - $lt 和 $gt

如果你想获取"col"集合中 "likes" 大于 100，小于 200 的数据，你可以使用以下命令：

```
> db.col.find({likes:{$gt:100, $lt:200}})
```

类似于 SQL 语句：

```
Select * from col where likes>100 AND  likes<200;
```

输出结果：

```
> db.col.find({likes : {$lt :200, $gt : 100}})
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程", "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 150 }
>
```

7.3 MongoDB $type 操作符

类型匹配用。

在本章节中，我们将继续讨论 MongoDB 中条件操作符 $type。$type 操作符是基于 BSON 类型来检索集合中匹配的数据类型，并返回结果。MongoDB 中可以使用的类型如下表所示：

类型	数字	备注
double	1	
string	2	
object	3	
array	4	
binary data	5	
undefined	6	已废弃。
object id	7	
boolean	8	
date	9	
null	10	
Regular Expression	11	
JavaScript	13	
Symbol	14	
JavaScript (with scope)	15	
32-bit integer	16	NumberInt()
Timestamp	17	
64-bit integer	18	NumberLong()
Min key	255	Query with -1.
Max key	127	

我们使用的数据库名称为"zvoice" 我们的集合名称为"col"，以下为我们插入的数据。

简单的集合"col"：

```
> db.col.insert({title: 1234, description:'c++是最好的语言', by: '零声学院', url: 'www.0voice.com', tags:['cpp'] , likes: 150})
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'c++教程', description:'c++是最好的语言', by: '零声学院', url: 'www.0voice.com', tags:['cpp'] , likes: 150})
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'c langage', description:'c 语言很不错', by: '零声学院', url: 'www.0voice.com', tags:['cpp'], likes: 100})
```

```
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'java', description:'java good', by: '零声学院', url: 'www.0voice.com
', tags:['cpp'], likes: 80})
WriteResult({ "nInserted" : 1 })
> db.col.insert({title: 'php', description:'script langage', by: '零声学院', url: 'www.0voic
e.com', tags:['cpp'], likes: 90})
WriteResult({ "nInserted" : 1 })
```

使用 find()命令查看数据：

```
> db.col.find().pretty()
{
        "_id" : ObjectId("5d807c11019abe974dac516a"),
        "title" : "c++教程",
        "description" : "c++是最好的语言",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ]
}
{
        "_id" : ObjectId("5d807c6e019abe974dac516b"),
        "title" : "c langage",
        "description" : "c 语言很不错",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 100
}
{
        "_id" : ObjectId("5d807c9b019abe974dac516c"),
        "title" : "java",
        "description" : "java good",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 80
}
```

```
{
        "_id" : ObjectId("5d807cb0019abe974dac516d"),
        "title" : "php",
        "description" : "script langage",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 90
}
>
```

MongoDB 操作符 - $type 实例

如果想获取 "col" 集合中 title 为 String 的数据，你可以使用以下命令：

```
> db.col.find({"title": {$type: 2}})
```

或

```
> db.col.find({"title": {$type: 'string'}})
```

输出结果为：

```
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程", "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 150 }
{ "_id" : ObjectId("5d807c6e019abe974dac516b"), "title" : "c langage", "description" : "c 语言很不错", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 100 }
{ "_id" : ObjectId("5d807c9b019abe974dac516c"), "title" : "java", "description" : "java good ", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 80 }
{ "_id" : ObjectId("5d807cb0019abe974dac516d"), "title" : "php", "description" : "script lan gage", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 90 }
```

> db.col.find({"title": {$type: 1}})

{ "_id" : ObjectId("62397d3c16df9538c3ee1a8c"), "title" : 1234, "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : ["cpp"], "likes" : 150 }

> db.col.insert({title: NumberInt(5678), description:'c++ 是 最 好 的 语 言 ', by: ' 零 声 学 院 ', url:

'www.0voice.com', tags:['cpp'] , likes: 150})

> db.col.find({"title": {$type: 16}})

{ "_id" : ObjectId("62397dae16df9538c3ee1a8d"), "title" : 5678, "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : ["cpp"], "likes" : 150 }

7.4 MongoDB Limit， Skip， Sort

MongoDB Limit() 方法

如果你需要在 MongoDB 中读取指定数量的数据记录，可以使用 MongoDB 的 Limit 方法，limit()方法接受一个数字参数，该参数指定从 MongoDB 中读取的记录条数。

limit()方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

集合 col 中的数据如下：

```
> db.col.find().limit(2)
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程", "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 150 }
{ "_id" : ObjectId("5d807c6e019abe974dac516b"), "title" : "c langage", "description" : "c 语言很不错", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 100 }
```

以下实例为显示查询文档中的记录：

```
> db.col.find({},{"title":1}).limit(5)
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程" }
{ "_id" : ObjectId("5d807c6e019abe974dac516b"), "title" : "c langage" }
{ "_id" : ObjectId("5d807c9b019abe974dac516c"), "title" : "java" }
{ "_id" : ObjectId("5d807cb0019abe974dac516d"), "title" : "php" }
```

注：如果你们没有指定 limit()方法中的参数则显示集合中的所有数据。

MongoDB Skip() 方法

我们除了可以使用 limit()方法来读取指定数量的数据外，还可以使用 skip()方法来跳过指定数量的数据，skip 方法同样接受一个数字参数作为跳过的记录条数。

skip() 方法脚本语法格式如下：

```
> db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

以下实例只会显示第二条文档数据

```
> db.col.find({},{"title":1}).limit(5).skip(2)
{ "_id" : ObjectId("5d807c9b019abe974dac516c"), "title" : "java" }
{ "_id" : ObjectId("5d807cb0019abe974dac516d"), "title" : "php" }
```

注:skip()方法默认参数为 0 。

MongoDB sort() 方法

在 MongoDB 中使用 sort() 方法对数据进行排序，sort() 方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 :

- 1 为升序排列
- -1 是用于降序排列。

sort()方法基本语法如下所示：

```
> db.COLLECTION_NAME.find().sort({KEY:1})
```

col 集合中的数据如下：

```
> db.col.find()
{ "_id" : ObjectId("5d807c11019abe974dac516a"), "title" : "c++教程", "description" : "c++是最好的语言", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 150 }
{ "_id" : ObjectId("5d807c6e019abe974dac516b"), "title" : "c langage", "description" : "c 语言很不错", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 100 }
{ "_id" : ObjectId("5d807c9b019abe974dac516c"), "title" : "java", "description" : "java good", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 80 }
{ "_id" : ObjectId("5d807cb0019abe974dac516d"), "title" : "php", "description" : "script langage", "by" : "零声学院", "url" : "www.0voice.com", "tags" : [ "cpp" ], "likes" : 90 }
```

以下实例演示了 col 集合中的数据按字段 likes 的降序排列：

```
> db.col.find({},{"title":1, "likes":1, "_id":0}).sort({"likes":1})
{ "title" : "java", "likes" : 80 }
{ "title" : "php", "likes" : 90 }
{ "title" : "c langage", "likes" : 100 }
```

```
{ "title" : "c++教程", "likes" : 150 }
```

7.5 MongoDB 索引创建、查询、删除

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB 在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，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

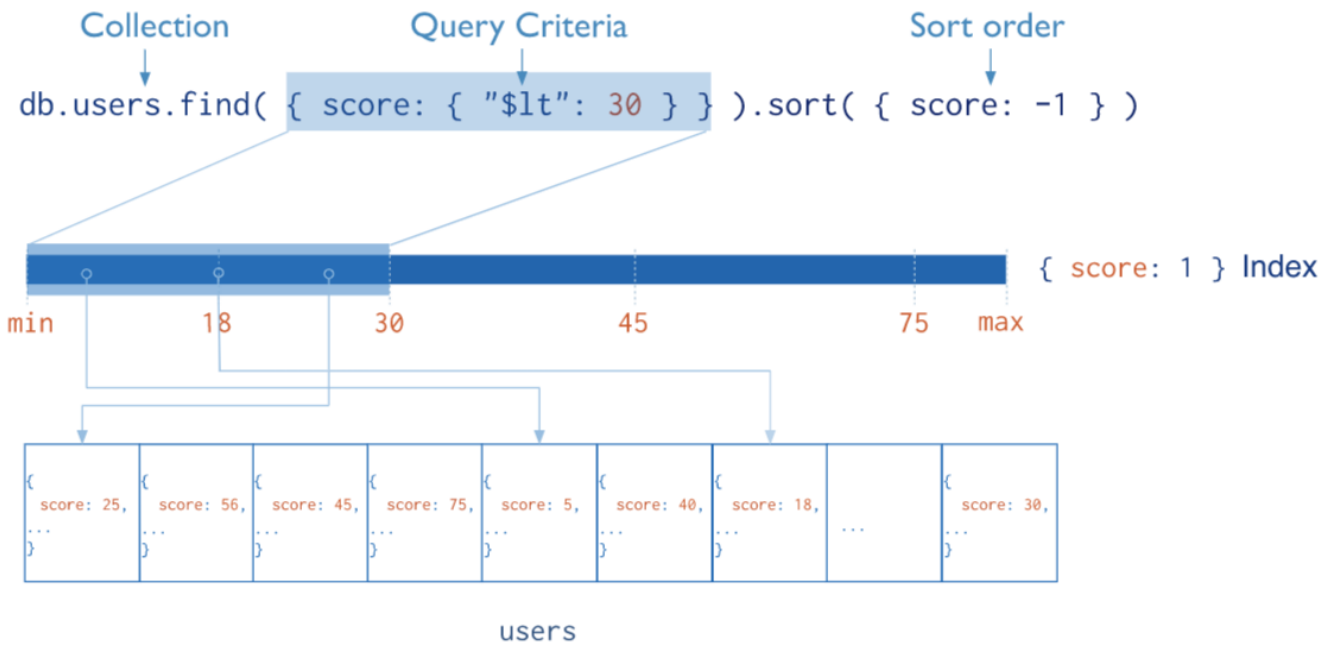
官方文档：https://docs.mongoing.com/indexes

索引原理参考：https://mongoing.com/archives/2789 《MongoDB 索引原理》

索引支持在 MongoDB 中有效地执行查询。如果没有索引，**MongoDB 必须执行集合扫描，即扫描集合中的每个文档**，以选择那些与查询语句匹配的文档。如果一个查询存在适当的索引，MongoDB 可以使用该索引来限制它必须检查的文档数量。

索引是特殊的数据结构，它以一种易于遍历的形式存储集合数据集的一小部分。索引存储一个或一组特定字段的值，按字段的值排序。索引项的排序支持有效的相等匹配和基于范围的查询操作。此外，MongoDB 可以通过使用索引中的排序返回排序后的结果。

下图说明了使用索引选择和排序匹配文档的查询：



使用索引选择并返回排序结果的查询图。 索引按升序存储“分数”值。 MongoDB 可以按升序或降序遍历索引以返回排序的结果。

基本上，MongoDB 中的索引与其他数据库系统中的索引类似。MongoDB 在集合级别定义索引，并支持在 MongoDB 集合中文档的任何字段或子字段上的索引。

1、创建索引

db.col.createIndex({"title":1})

2、查看集合索引

db.col.getIndexes()

3、查看集合索引大小

db.col.totalIndexSize() 返回当前集合所有的索引所占用的空间大小

4、删除集合所有索引

db.col.dropIndexes()

5、删除集合指定索引

db.col.dropIndex("索引名称")

createIndex() 创建索引

MongoDB 使用 createIndex() 方法来创建索引。

注意在 3.0.0 版本前创建索引方法为 db.collection.ensureIndex()，之后的版本使用了 db.collection.createIndex() 方法，ensureIndex() 还能用，但只是 createIndex() 的别名。

createIndex()方法基本语法格式如下所示：

```
> db.collection.createIndex(keys, options)
```

语法中 Key 值为你要创建的索引字段，**1 为指定按升序创建索引**，如果你想按降序来创建索引指定为 -1 即可。

```
> db.col.createIndex({"title":1})
> db.col.createIndex({"title":1}, {unique:true})
```

createIndex() 方法中你也可以设置使用多个字段创建索引（关系型数据库中称作复合索引）。

```
> db.col.createIndex({"title":1,"description":-1})
```

```
>
```

createIndex() 接收可选参数，可选参数列表如下：

Parameter	Type	Description
background	Boolean	建索引过程会阻塞其它数据库操作，background 可指定以后台方式创建索引，即增加 "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顺序生成一个索引名称。
dropDups	Boolean	**3.0+版本已废弃。**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**.
sparse	Boolean	对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为 true 的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 **false**.
expireAfterSeconds	integer	指定一个以秒为单位的数值，完成 TTL 设定，设定集合的生存时间。
v	index version	索引的版本号。默认的索引版本取决于 mongod 创建索引时运行的版本。
weights	document	索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。
default_language	string	对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语

language_overr ide	string	对于文本索引，该参数指定了包含在文档中的字段名， 语言覆盖默认的 language，默认值为 language.

实例

在后台创建索引：

```
db.values.createIndex({open: 1, close: 1}, {background: true})
```

通过在创建索引时加 background:true 的选项，让创建工作在后台执行

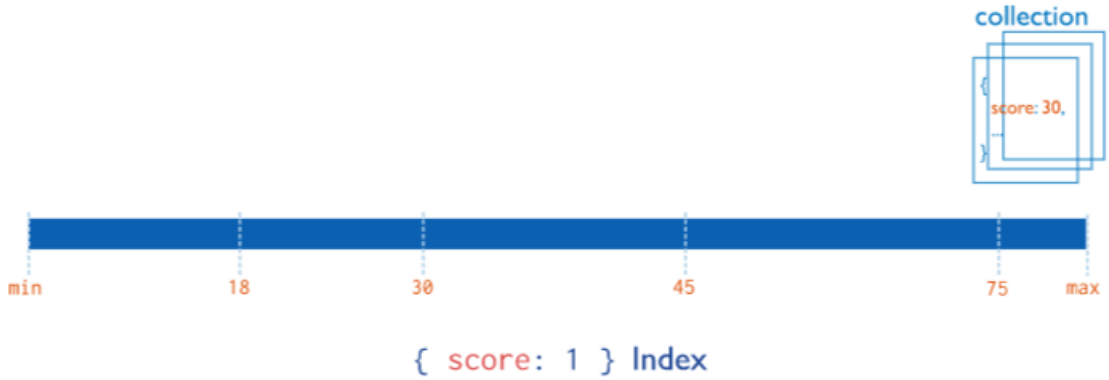
默认的 _id index

Mongodb 在 collection 创建时会默认建立一个基于_id 的唯一性索引作为 document 的 primary key，这个 index 无法被删除。

Mongodb 支持多种方式创建索引，具体创建方式见官方文档 https://docs.mongodb.com/manual/indexes/#create-an-index

单个字段单字段索引

单字段索引是 Mongodb 最简单的索引类型，不同于 MySQL，MongoDB 的索引是有顺序 的，支持升序或者降序。



但是对于单字段索引来说，索引的顺序无关紧要，因为 MongoDB 支持任意顺序遍历单字段索引。

在此创建一个 records collection：

```
{
  "_id": ObjectId("570c04a4ad233577f97dc459"),
  "score": 1034,
  "location": { state: "NY", city: "New York" }
}
db.records.insert({
  "score": 1034,
  "location": { state: "NY", city: "New York" }
})
```

然后创建一个 单字段索引：

```
db.records.createIndex( { score: 1 } )
```

上面的语句在 collection 的 score field 上创建了一个 升序索引，这个索引支持以下查询：

```
db.records.find( { score: 2 } )
db.records.find( { score: { $gt: 10 } } )
```

可以使用 MongoDB 的 explain 来对以上两个查询进行分析：

```
db.records.find({score:2}).explain('executionStats')
```

显示："stage" : "IXSCAN",

```
db.records.find({"location.state":"NY"}).explain('executionStats')
```

显示："stage" : "COLLSCAN",

MongoDB 查询性能分析—— explain 操作返回结果详解 - 学习园-IT 技术交流分享平台 (xuexiyuan.cn) https://xuexiyuan.cn/article/detail/179.html

嵌入字段的单个索引

此外 MongoDB 还支持对嵌入字段进行索引创建：

```
db.records.createIndex( { "location.state": 1 } )
```

上面的 embedded index 支持以下查询：

```
db.records.find( { "location.state": "CA" } )
db.records.find( { "location.city": "Albany", "location.state": "NY" } )
```

单个索引上排序

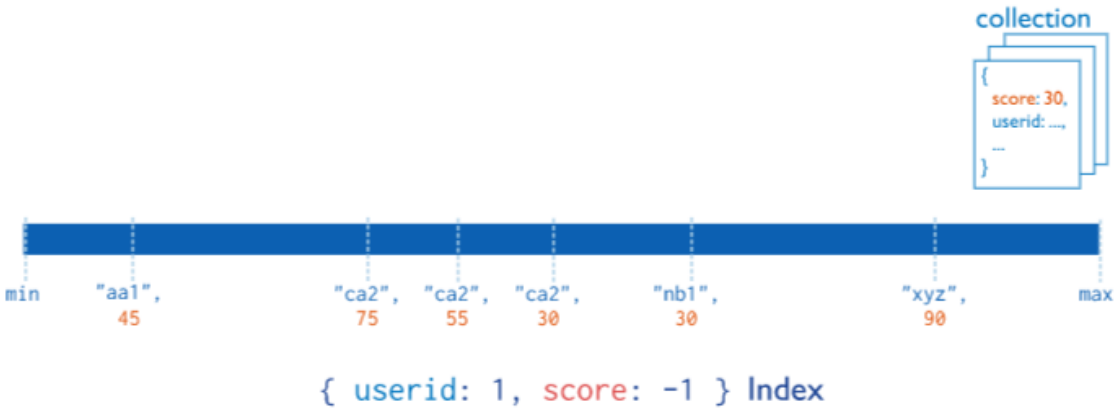
对于单索引来说，由于 MongoDB index 本身支持顺序查找，所以对于单索引来说

```
db.records.find().sort( { score: 1 } )
db.records.find().sort( { score: -1 } )
db.records.find({score:{$lte:100}}).sort( { score: -1 } )
```

这些查询语句都是满足使用 index 的。

复合索引

Mongodb 支持对多个字段建立索引，称之为复合索引。复合索引 中 field 的顺序对索引的性能有至关重要的影响，比如索引 {userid:1, score:-1} 首先根据 userid 排序，然后再在每个 userid 中根据 score 排序。



创建复合索引

在此创建一个 products collection：

```
db.products.insert({
 "item": "Banana",
 "category": ["food", "produce", "grocery"],
 "location": "4th Street Store",
 "stock": 4,
 "type": "cases"
})

db.products.insert({
 "item": "Apple",
 "category": ["food", "produce", "grocery"],
```

```
 "location": "4th Street Store",
 "stock": 8,
 "type": "cases"
})
```

然后创建一个 复合索引：

db.products.createIndex({ "item": 1, "stock": 1 })

这个 index 引用的 document 首先会根据 item 排序,然后在 每个 item 中,又会根据 stock 排序，以下语句都满足该索引：

db.products.find({ item: "Banana" })

db.products.find({ item: "Banana", stock: { $gt: 5 } })

条件 {item: "Banana"} 满足是因为这个 query 满足 prefix 原则。

```
    "item": 1, "stock": 1
```

索引：

```
  "item": 1
  "item": 1, "stock": 1
```

使用复合索引需要满足 prefix 原则

Index prefix 是指 index fields 的左前缀子集，考虑以下索引：

```
{ "item": 1, "location": 1, "stock": 1 }
```

这个索引包含以下 index prefix：

```
{ item: 1 }
{ item: 1, location: 1 }
{ "item": 1, "location": 1, "stock": 1 }
```

所以只要语句满足 index prefix 原则都是可以支持使用 复合索引 的：

db.products.find({ item: "Banana" })

db.products.find({ item: "Banana",location:"4th Street Store"})

db.products.find({ item: "Banana",location:"4th Street Store",stock:4})

相反如果不满足 index prefix 则无法使用索引，比如以下 field 的查询：

the **location** field

the **stock** field

the **location** and **stock** fields

由于 index prefix 的存在，如果一个 collection 既有 **{a:1, b:1}** 索引 ，也有 **{a:1}** 索引，如果二者没有稀疏或者唯一性的要求，单索引是可以移除的。

Sort on 复合索引

前文说过 single index 的 sort 顺序无关紧要，但是 复合索引 则完全不同。

考虑有如下场景：

```
db.events.find().sort( { username: 1, date: -1 } )
```

events collection 有一个上面的查询，首先结果根据 username 进行 ascending 排序，然后再对结果进行 date descending 排序，或者是下面的查询：

```
db.events.find().sort( { username: -1, date: 1 } )
```

根据 username 进行 descending 排序，然后再对 date 进行 ascending 排序，索引：

```
db.events.createIndex( { "username" : 1, "date" : -1 } )
```

可以支持这两种查询，但是下面的查询不支持：

```
db.events.find().sort( { username: 1, date: 1 })
```

也就是说 sort 的顺序必须要和创建索引的顺序是一致的，一致的意思是不一定非要一样，总结起来大致如下

	{ "username" : 1, "date" : -1 }	**{ "username" : 1, "date" : 1 }**
sort({ username: 1, date: -1 })	支持	不支持
sort({ username: -1, date: 1 })	支持	不支持
sort({ username: 1, date: 1 })	不支持	支持
sort({ username: -1, date: -1 })	不支持	支持

即排序的顺序必须要和索引一致，逆序之后一致也可以，下表清晰的列出了 复合索引 满足的 query 语句：

query	index
db.data.find().sort({ a: 1 })	{ a: 1 }
db.data.find().sort({ a: -1 })	{ a: 1 }
db.data.find().sort({ a: 1, b: 1 })	{ a: 1, b: 1 }
db.data.find().sort({ a: -1, b: -1 })	{ a: 1, b: 1 }
db.data.find().sort({ a: 1, b: 1, c: 1 })	{ a: 1, b: 1, c: 1 }
db.data.find({ a: { $gt: 4 } }).sort({ a: 1, b: 1 })	{ a: 1, b: 1 }

非 index prefix 的排序

考虑索引｛ a: 1, b: 1, c: 1, d: 1 ｝，即使排序的 field 不满足 index prefix 也是可以的，但前提条件是**排序 field 之前的 index field 必须是等值条件**，

	Example	Index Prefix
r1	db.data.find({ a: 5 }).sort({ b: 1, c: 1 })	{ a: 1 , b: 1, c: 1 }
r2	db.data.find({ b: 3, a: 4 }).sort({ c: 1 })	{ a: 1, b: 1, c: 1 }
r3	db.data.find({ a: 5, b: { $lt: 3} }).sort({ b: 1 })	{ a: 1, b: 1 }

上面表格 r1 的排序 field 是 b 和 c，a 是 index field 而且在 b 和 c 之前，可以使用索引；r3 的排序中 b 是范围查询，但是 b 之前的 a 用的也是等值条件，也就是只要排序 field 之前的 field 满足等值条件即可，其它的 field 可以任意条件。

哈希索引

哈希索引(hashed Indexes)就是将 field 的值进行 hash 计算后作为索引，其强大之处在于实现 O(1)查找，当然用哈希索引最主要的功能也就是实现定值查找， 对于经常需要排序或查询范围查询的集合不要使用哈希索引。

1. 添加测试数据

```
db.userinfos.insertMany([
    {_id:1, name: "张三", age: 23,level:10, ename: { firstname: "san", lastname: "zhang"}, roles: ["vip","gen" ]},
    {_id:2, name: "李四", age: 24,level:20, ename: { firstname: "si", lastname: "li"}, roles:[ "vip" ]},
    {_id:3, name: "王五", age: 25,level:30, ename: { firstname: "wu", lastname: "wang"}, roles:
```

["gen","vip"]},

{_id:4, name: "赵六", age: 26,level:40, ename: { firstname: "liu", lastname: "zhao"}, roles: ["gen"] },

{_id:5, name: "田七", age: 27, ename: { firstname: "qi", lastname: "tian"}, address:'北京' },

{_id:6, name: "周八", age: 28,roles:["gen"], address:'上海' }

]);

2. 索引测试

//建立索引

db.userinfos.createIndex({"name":"hashed"})

//测试索引

db.userinfos.find({"name":"张三"}).explain()

//删除索引

正确删除：db.userinfos.dropIndex({ "name": "hashed" })

错误删除：db.userinfos.dropIndex({ "name": 1 }) 这里的 1 是代表的是升序，不是 enable

如何建立正确索引

前文基本覆盖了日常使用 MongoDB 所需要的主要索引知识，但是如何才建立正确的索引？

使用 explain 分析查询语句

MongoDB 默认提供了类似 MySQL explain 的语句来分析查询语句的来对我们正确建立索引提供帮助，在建立索引时我们需要对照 explain 对各种查询条件进行分析。

理解 field 顺序对索引的影响

索引的真正作用是帮助我们限制数据的选择范围，比如 复合索引 多个 feild 的顺序如何决定，应该首选可以最大化的缩小数据查找范围的 field，这样如果第一个 field 可以迅速缩小数据的查找范围，那么后续的 feild 匹配的行就会变少很多。考虑语句：

{'online_time': {'$lte': present}, 'offline_time': {'$gt': present}, 'online': 1, 'orientation': 'quality', 'id': {'$gt': max_id}}

考虑如下索引

	索引	nscanded
r1	{start_time:1, end_time: 1, origin: 1, id: 1, orientation: 1}	12959
r2	{start_time:1, end_time: 1, origin: 1, orientation: 1, id: 1}	2700

由于 field id 和 orientation 的顺序不同会导致需要扫描的 documents 数量差异巨大，说明二者对对数据的限制范围差别很大，优先考虑能够最大化限制数据范围的索引顺序。

监控慢查询

始终对生成环境产生的慢查询进行第一时间分析，提早发现问题并解决。

db profiling

MongoDB 支持对 DB 的请求进行 profiling，目前支持 3 种级别的 profiling。

- 0： 不开启 profiling
- 1： 将处理时间超过某个阈值(默认 100ms)的请求都记录到 DB 下的 system.profile 集合 （类似于 mysql、redis 的 slowlog）
- 2： 将所有的请求都记录到 DB 下的 system.profile 集合（生产环境慎用）

通常，生产环境建议使用 1 级别的 profiling，并根据自身需求配置合理的阈值，用于监测慢请求的情况，并及时的做索引优化。

如果能在集合创建的时候就能『根据业务查询需求决定应该创建哪些索引』，当然是最佳的选择；但由于业务需求多变，要根据实际情况不断的进行优化。索引并不是越多越好，集合的索引太多，会影响写入、更新的性能，每次写入都需要更新所有索引的数据；所以你 system.profile 里的慢请求可能是索引建立的不够导致，也可能是索引过多导致。

getIndexes()查询索引

语法：

db.collection.getIndexes()

实例：

db.values.getIndexes()

显示：

```
[
    {
        "v" : 2,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_"
    },
    {
        "v" : 2,
        "key" : {
            "open" : 1,
            "close" : 1
        },
        "name" : "open_1_close_1",
        "background" : true
    }
]
```

dropIndexe()查询索引

通过使用 dropIndex 方法在 MongoDB 中删除索引。

删除索引：

```
db.values.dropIndexes({open:1, close:1})
```

删除_id 索引以外的所有索引

```
db.values.dropIndex()
```

8 MongoDB 聚合

8.1 概念

MongoDB 聚合框架（Aggregation Framework）是一个计算框架，它可以：

- 作用在一个或几个集合上；
- 对集合中的数据进行的一系列运算；
- 将这些数据转化为期望的形式；

从效果而言，聚合框架相当于 SQL 查询中的：

- GROUP BY
- LEFT OUTER JOIN
- AS 等

管道（Pipeline）和步骤（Stage）

整个聚合运算过程称为管道（Pipeline），它是由多个步骤（Stage）组成的，每个管道：

- 接受一系列文档（原始数据）；
- 每个步骤对这些文档进行一系列运算；
- 结果文档输出给下一个步骤；

聚合运算的基本格式

```
pipeline = [$stage1, $stage2, ...$stageN];
db.<Collection>.aggregtae(
    pipeline,
    {options}
);
```

聚合运算常见 步骤（Stage）

步骤	作用	SQL 等价运算符

$match	过滤	WHERE
$project	投影	AS
$sort	排序	ORDER BY
$group	分组	GROUP BY
$skip/$limit	结果限制	SKIP/LIMIT
$lookup	左外连接	LEFT OUTER JOIN
$unwind	将数组类型的字段进行拆分	

聚合运算的常见使用场景

- 聚合查询可以用于 OLAP 和 OLTP。例如：

OLAP **（On-Line Analytical Processing，联机分析处理）**	**OLTP** **（On-Line Transaction Processing，联机事务处理）**
分析一段时间内的销售总额、均值； 计算一段时间内的净利润； 分析购买人群的年龄分布； 分析学生成绩分布； 统计员工绩效；	计算

8.2 表达式操作符

表达式	描述	实例
$sum	计算总和。	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	计算平均值	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])

$min	获取集合中所有文档对应值得最小值。	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	获取集合中所有文档对应值得最大值。	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	在结果文档中插入值到一个数组中。	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	在结果文档中插入值到一个数组中，但不创建副本。	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	根据资源文档的排序获取第一个文档数据。	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	根据资源文档的排序获取最后一个文档数据	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

8.3 aggregate() 方法

MongoDB 中聚合的方法使用 aggregate()。aggregate() 方法的基本语法格式如下所示：

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

集合中的数据如下：

```
> db.col.find().pretty()
{
        "_id" : ObjectId("5d807c11019abe974dac516a"),
        "title" : "c++教程",
        "description" : "c++是最好的语言",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 150
}
{
        "_id" : ObjectId("5d807c6e019abe974dac516b"),
        "title" : "c langage",
        "description" : "c 语言很不错",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 100
}
{
        "_id" : ObjectId("5d807c9b019abe974dac516c"),
        "title" : "java",
        "description" : "java good",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 80
}
{
        "_id" : ObjectId("5d807cb0019abe974dac516d"),
        "title" : "php",
```

```
        "description" : "script langage",
        "by" : "零声学院",
        "url" : "www.0voice.com",
        "tags" : [
                "cpp"
        ],
        "likes" : 90
}
>
```

现在我们通过以上集合计算每个作者所写的文章数，使用 aggregate()计算结果如下：

```
> db.col.aggregate([{$group: {_id: "$likes", num: {$sum:1}}}])
{ "_id" : 90, "num" : 1 }
{ "_id" : 80, "num" : 1 }
{ "_id" : 100, "num" : 1 }
{ "_id" : 150, "num" : 1 }
>
```

以上实例类似 sql 语句：

```
select likes, count(*) from col group by likes
```

8.4 管道的概念

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。这里我们介绍一下聚合框架中常用的几个操作：

- $project：修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。
- $match：用于过滤数据，只输出符合条件的文档。$match 使用 MongoDB 的标准查询操作。
- $limit：用来限制 MongoDB 聚合管道返回的文档数。
- $skip：在聚合管道中跳过指定数量的文档，并返回余下的文档。
- $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。
- $group：将集合中的文档分组，可用于统计结果。

- $sort：将输入文档排序后输出。
- $geoNear：输出接近某一地理位置的有序文档。

1、$project 实例

```
> db.col.aggregate({$project:{title:1}})
```

这样的话结果中就只还有_id,tilte 和 author 三个字段了，默认情况下_id 字段是被包含的，如果要想不包含_id 话可以这样:

```
> db.col.aggregate({$project:{title:1, likes:1, _id:0}});
```

2.$match 实例

```
> db.col.aggregate([{$match: {likes:{$gt:80, $lte:150}}}, {$group: {_id:0, num: {$sum:1}}}])
{ "_id" : 0, "num" : 3 }
```

$match 用于获取分数大于 70 小于或等于 90 记录，然后将符合条件的记录送到下一阶段$group 管道操作符进行处理。

3.$skip 实例

```
db.col.aggregate([{$match: {likes:{$gt:80, $lte:150}}}, {$group: {_id:0, likes: {$sum:
1}}}]).skip(1)
```

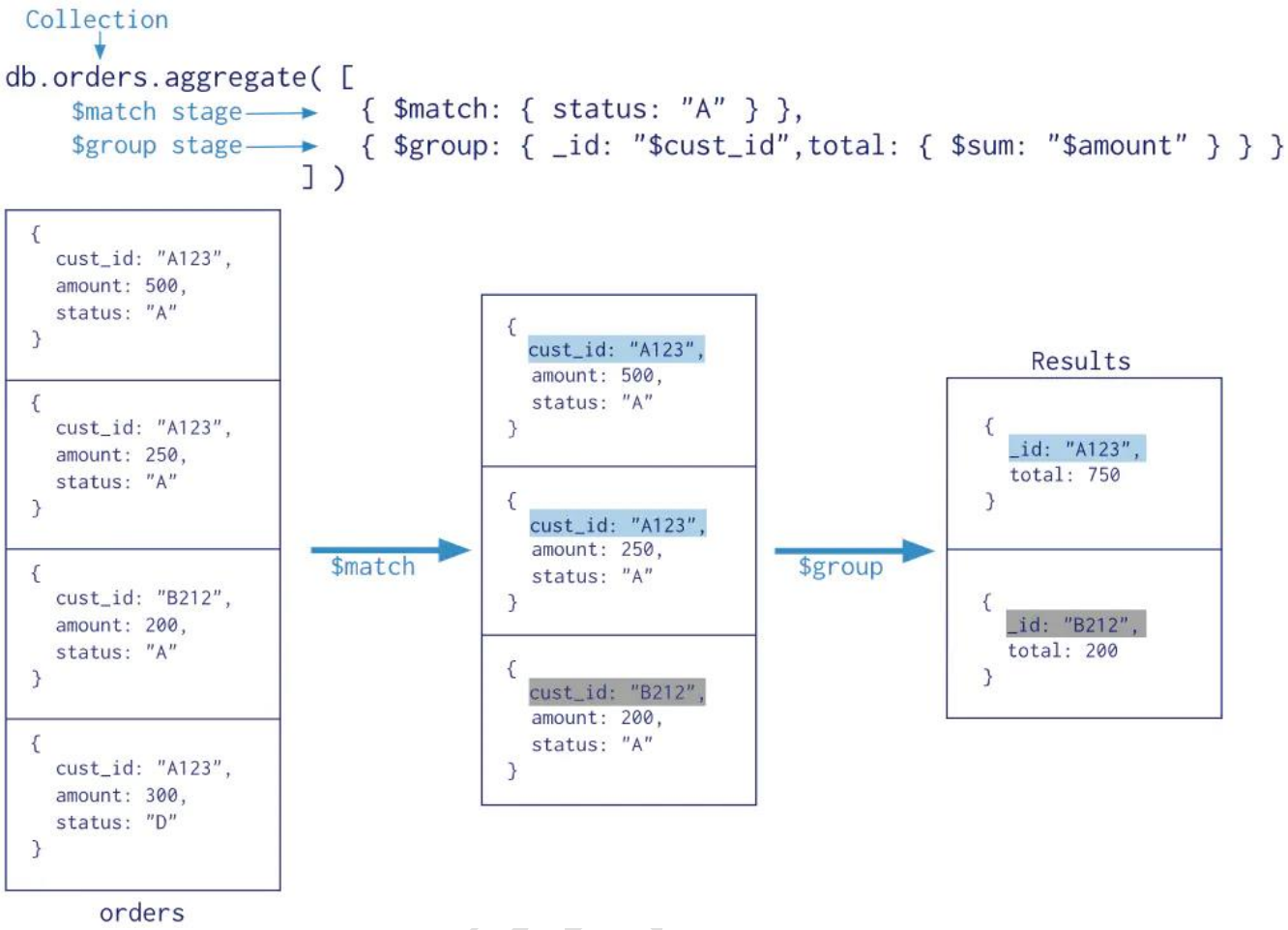
经过$skip 管道操作符处理后，前五个文档被"过滤"掉。

9 MongoDB 聚合运算范例

范例 1-官方参考

MongoDB 的聚合框架基于数据处理管道的概念。文档进入一个多级管道，将文档转换为聚合结果。比如文档的投影，过滤，排序，分组，等等。此外，管道阶段还可以使用操作符来执行任务，例如计算平均值或连接字符串等等。

下图为一个简单的聚合例子(此图来自 mongodb 的官网)



官网聚合管道案例

如上图所示：先使用**$match** 构建筛选出 status 等于 A 的数据，然后使用**$group** 构建分组数据，以 cust_id 进行分组，使用 $sum 进行分组的求和操作。

聚合管道的限制

文档大小限制

聚合的返回单个文档不可超过 16M,但是聚合的过程中单个文档可以超过 16M.

内存的限制

聚合阶段**默认**情况下可以**使用 100M 的内存**，超过则报错。如果想处理需要超过 100M 内存的数据，则需要将 **allowDiskUse** 设置成 **true**,让其可以写入临时文件。但是在$graphLookup 阶段，内存还

是限制到 100M 以内，即使设置了 allowDiskUse=true，在此管道阶段会失效，但是如果以其他的管道阶段还是会生效的。**当 allowDiskUser=false**,内存超出发生异常。

聚合管道阶段

- $match 用于过滤数据，用于聚合阶段的输入
- $order 用指定的键，对文档进行排序
- $limit 用于限制多少个文档作为输入
- $skip 跳过多少个文档
- $project 投影字段，可以理解为查询多少个字段，类似为 select a,b,c 中的 a,b,c
- $group 进行分组操作,其中_id 字段用于指定需要分组的字段。
- $count 返回这个聚合管道阶段的文档的数量

更多管道阶段，请点击 https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#aggregation-pipeline-operator-reference。

准备数据

```
db.persons.insertMany([
        {userId : '001',age : 24,salary : 5000,dept : '部门一'},
        {userId : '002',age : 25,salary : 7000,dept : '部门二'},
        {userId : '003',age : 23,salary : 8000,dept : '部门一'},
        {userId : '004',age : 26,salary : 1000,dept : '部门三'},
        {userId : '005',age : 27,salary : 2000,dept : '部门二'},
        {userId : '006',age : 22,salary : 7000,dept : '部门一'},
        {userId : '007',age : 25,salary : 6000,dept : '部门三'},
        {userId : '008',age : 26,salary : 4000,dept : '部门三'},
        {userId : '009',age : 28,salary : 9000,dept : '部门二'}
])
```

1 使用 $project 投影出需要的字段

- 排除 _id 字段
- 返回 age 字段
- 产生一个新字段 newAge,它的值为原 age 字段的值 加 1

```
db.persons.aggregate([
    {$project: {_id : 0,age : 1,newAge : {$add : ['$age',1]}}}
])
```

2 使用 $match 进行数据的过滤

和普通的查询条件是一样的。

```
db.persons.aggregate([
      {$match : {age : {$gt : 22}} }
])
```

3 使用 $sort 进行排序

```
db.persons.aggregate([
      {$match : {age : {$gt : 22}} },
      {$sort : {age : 1}}
])
```

4 使用 $limit 和 $skip 进行限制数据和过滤数据

```
db.persons.aggregate([
      {$match : {age : {$gt : 22}} },
      {$sort : {age : -1}},
      {$limit : 6},
      {$skip : 3}
])
```

5 使用 $group 进行分组操作

```
db.persons.aggregate([
      {$group : {_id : "$dept",count : {$sum : 1}}}
]);
```

简单的练习

需求：获取 6 个年龄大于 22 周岁的用户，其中如果薪水如果小于 1000，直接将薪水上调到 4000，前面一步做好后，需要排除年龄最大的一个，求出每个部门，相同年龄的员工的平均薪水，并得到薪水最高的三个人。

思路：1、投影出**年龄(age),部门(dept),薪水(salary)**字段

2、查出年龄大于 22 周岁的员工

3、以年龄倒序进行排序

4、限制返回 7 条数据，并跳过一条数据

5、以部门年龄进行分组，并求出平均分

6、以上一步的平均分在进行倒序排序

7、然后再返回 3 条数据

代码如下：

```
db.persons.aggregate([
    { $project : {age : 1,dept : 1,oldSalary : "$salary",salary : {
        $switch : {
            branches : [
                { case : { $lte : ["$salary",1000] }, then : 4000}
            ],
            default : '$salary'
        }
    }} },
    { $match : {age : {$gt : 22}} },
    { $sort : {age : -1}},
    { $limit : 7},
    { $skip : 1 },
    { $group : { _id : {dept : "$dept",age : "$age"},pers : {$sum : 1} ,
deptAvgSalary : { $avg : "$salary"} } },
    { $sort : {deptAvgSalary : -1}},
    { $limit : 3}
]);
```

运行效果

```
{ "_id" : { "dept" : "部门二", "age" : 25 }, "pers" : 1, "deptAvgSalary" : 7000 }
{ "_id" : { "dept" : "部门三", "age" : 25 }, "pers" : 1, "deptAvgSalary" : 6000 }
{ "_id" : { "dept" : "部门一", "age" : 24 }, "pers" : 1, "deptAvgSalary" : 5000 }
```

范例 2-求和

数据

```
db.news_clf.insertMany([
{
    "_id" : "2020-02-01",
    "website_clf" : [
        {
            "source" : "猎云网",
            "sum_num" : 3880,
            "day_num" : 11,
            "time_clf" : {
                "1" : 0,
                "2" : 0,
                "3" : 0,
                "4" : 0,
                "5" : 0,
                "6" : 0,
                "7" : 0,
                "8" : 1,
                "9" : 0,
                "10" : 0,
                "11" : 3,
                "12" : 0,
                "13" : 0,
```

```
                    "14" : 0,
                    "15" : 2,
                    "16" : 2,
                    "17" : 1,
                    "18" : 1,
                    "19" : 0,
                    "20" : 0,
                    "21" : 0,
                    "22" : 0,
                    "23" : 1,
                    "24" : 0
                }
            },
            {
                "source" : "钛媒体",
                "sum_num" : 1086,
                "day_num" : 14,
                "time_clf" : {
                    "1" : 0,
                    "2" : 0,
                    "3" : 0,
                    "4" : 0,
                    "5" : 0,
                    "6" : 0,
                    "7" : 0,
                    "8" : 1,
                    "9" : 2,
                    "10" : 1,
                    "11" : 0,
                    "12" : 1,
                    "13" : 0,
                    "14" : 1,
                    "15" : 1,
                    "16" : 0,
                    "17" : 0,
                    "18" : 1,
                    "19" : 1,
                    "20" : 4,
                    "21" : 0,
                    "22" : 1,
                    "23" : 0,
                    "24" : 0
                }
            }
        ]
},
{
        "_id" : "2020-02-02",
        "website_clf" : [
            {
                "source" : "猎云网",
                "sum_num" : 3895,
                "day_num" : 15,
                "time_clf" : {
                    "1" : 0,
                    "2" : 0,
                    "3" : 0,
                    "4" : 0,
                    "5" : 0,
                    "6" : 0,
                    "7" : 0,
```

```
                "8" : 0,
                "9" : 1,
                "10" : 0,
                "11" : 0,
                "12" : 3,
                "13" : 2,
                "14" : 1,
                "15" : 4,
                "16" : 0,
                "17" : 2,
                "18" : 0,
                "19" : 0,
                "20" : 1,
                "21" : 0,
                "22" : 0,
                "23" : 0,
                "24" : 1
            }
        },
        {
            "source" : "钛媒体",
            "sum_num" : 1101,
            "day_num" : 15,
            "time_clf" : {
                "1" : 0,
                "2" : 0,
                "3" : 0,
                "4" : 0,
                "5" : 0,
                "6" : 0,
                "7" : 0,
                "8" : 0,
                "9" : 1,
                "10" : 1,
                "11" : 2,
                "12" : 0,
                "13" : 2,
                "14" : 1,
                "15" : 0,
                "16" : 0,
                "17" : 1,
                "18" : 4,
                "19" : 0,
                "20" : 1,
                "21" : 2,
                "22" : 0,
                "23" : 0,
                "24" : 0
            }
        }
    ]
}])
```


以 source 为类别统计 day_num 和数量（match'匹配对应的 id，website 是数组所以要拆分，group'分组结果）

```
db.getCollection('news_clf').aggregate(

    [{'$match':{"_id":{'$gte':"2020-02-01",'$lte':"2020-02-02"}}},
```

```
{'$unwind':'$website_clf'},
{'$group':{"_id":"$website_clf.source",'sumFaceAmnt':{'$sum':'$website_clf.day_num'}}}]
)
```

结果

```
{ "_id" : "猎云网", "sumFaceAmnt" : 26 }
{ "_id" : "钛媒体", "sumFaceAmnt" : 29 }
```

范例 3-求和

集合中的数据如下：

```
db.author.insertMany([
{
    title: 'MongoDB Overview',
    description: 'MongoDB is no sql database',
    by_user: 'zvoice',
    url: 'http://www.runoob.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
},
{
    title: 'NoSQL Overview',
    description: 'No sql database is very fast',
    by_user: 'zvoice',
    url: 'http://www.runoob.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 10
},
{
    title: 'Neo4j Overview',
    description: 'Neo4j is no sql database',
    by_user: 'Neo4j',
```

```
    url: 'http://www.neo4j.com',
    tags: ['neo4j', 'database', 'NoSQL'],
    likes: 750
}])
```

计算每个作者所写的文章数

现在我们通过以上集合计算每个作者所写的文章数，使用 aggregate()计算结果如下：

```
db.author.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
```

查询结果如下所示：

```
{ "_id" : "Neo4j", "num_tutorial" : 1 }
{ "_id" : "zvoice", "num_tutorial" : 2 }
```

统计每个作者被 like 的总和，计算表达式

```
db.author.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
```

查询结果如下所示

```
{ "_id" : "Neo4j", "num_tutorial" : 750 }
{ "_id" : "zvoice", "num_tutorial" : 110 }
```

范例 4-统计业绩

测试集合 sales 的数据如下

```
db.sales.insertMany([
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : ISODate("2014-01-01T08:00:00Z") },
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" : ISODate("2014-02-03T09:00:00Z") },
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 5, "date" : ISODate("2014-02-03T09:05:00Z") },
{ "_id" : 4, "item" : "abc", "price" : 10, "quantity" : 10, "date" : ISODate("2014-02-15T08:00:00Z") },
{ "_id" : 5, "item" : "xyz", "price" : 5, "quantity" : 10, "date" : ISODate("2014-02-15T09:05:00Z") }
])
```

需要完成的目标是，**基于日期分组**，统计每天的销售额，聚合公式为：

```
db.sales.aggregate(
```

```
[
  {
    $group:
      {
        _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
        totalAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } },
        count: { $sum: 1 }
      }
  }
]
)
```

查询结果是：

```
{ "_id" : { "day" : 46, "year" : 2014 }, "totalAmount" : 150, "count" : 2 }
{ "_id" : { "day" : 34, "year" : 2014 }, "totalAmount" : 45, "count" : 2 }
{ "_id" : { "day" : 1, "year" : 2014 }, "totalAmount" : 20, "count" : 1 }
```

上面的，可以看出$group，我们都使用了_id,使用了分组，那么如果，我们的需求不需要分组，应该怎么办呢？

例如。**我们现在要统计 sales 集合中一共卖出了多少件商品**。

如果直接去掉 group 阶段的_id,如下:

```
db.sales.aggregate(
  [
    {
      $group:
        {
          totalAmount: { $sum: "$quantity" }
        }
    }
  ]
)
```

则报错：

```
{
	"message" : "a group specification must include an _id",
	"ok" : 0,
	"code" : 15955,
	"codeName" : "Location15955",
	"name" : "MongoError"
}
```

我们还是需要添加上_id,但是可以添加个常量，及时根据常量分组,可以为 _id：“0” 可以是 _id：“a”, _id：“b”, 还可以使_id：“x”, _id：“y” 等等。

例如：

```
 db.sales.aggregate(
    [
      {
         $group:
           {
             _id : "Total"
             totalAmount: { $sum: "$quantity" }
           }
      }
    ]
)
```

查询结果为：

```
{
	"_id" : "Total",
	"totalAmount" : 28
}
```

范例 5-projection

project 阶段

假设存在一个 students 集合，其数据结构如下：

```
db.students.insertMany([
{ "_id": 1, "quizzes": [ 10, 6, 7 ], "labs": [ 5, 8 ], "final": 80, "midterm": 75 },
{ "_id": 2, "quizzes": [ 9, 10 ], "labs": [ 8, 8 ], "final": 95, "midterm": 80 },
{ "_id": 3, "quizzes": [ 4, 5, 5 ], "labs": [ 6, 5 ], "final": 78, "midterm": 70 }
])
```

现在的需求是统计每个学生的 平常的测验分数总和、实验分数总和、期末期中分数总和。

```
db.students.aggregate([
    {
       $project: {
          quizTotal: { $sum: "$quizzes"},
          labTotal: { $sum: "$labs" },
          examTotal: { $sum: [ "$final", "$midterm" ] }
       }
    }
])
```

其查询输出结果如下：

```
{ "_id" : 1, "quizTotal" : 23, "labTotal" : 13, "examTotal" : 155 }
{ "_id" : 2, "quizTotal" : 19, "labTotal" : 16, "examTotal" : 175 }
{ "_id" : 3, "quizTotal" : 14, "labTotal" : 11, "examTotal" : 148 }
```

计算所有的总分

```
db.students.aggregate([
    {
```

```
        $project: {
            quizTotal: { $sum: "$quizzes"},
            labTotal: { $sum: "$labs" },
            examTotal: { $sum: [ "$final", "$midterm"] },
            total: { $sum: [ "$final", "$midterm" , { $sum: "$labs" }, {$sum: "$quizzes"}] },
        }
    }
])
```

结果

```
{ "_id" : 1, "quizTotal" : 23, "labTotal" : 13, "examTotal" : 155, "total" : 191 }
{ "_id" : 2, "quizTotal" : 19, "labTotal" : 16, "examTotal" : 175, "total" : 210 }
{ "_id" : 3, "quizTotal" : 14, "labTotal" : 11, "examTotal" : 148, "total" : 173 }
```

10 参考链接

英文官方参考文档：https://docs.mongodb.com/manual/reference/

中文社区参考文档：https://docs.mongoing.com/

中文社区：https://mongoing.com/

博客参考：https://mongoing.com/guoyuanwei

中文社区公众号：https://mp.weixin.qq.com/s/Wuzh47jsBh5QonBrZxUnjg 《WiredTiger 存储引擎系列》

mongodb 手册 https://docs.mongodb.com/manual/introduction/

英文：https://docs.mongodb.com/manual/crud/

中文：https://docs.mongoing.com/mongodb-crud-operations

索引原理参考：https://mongoing.com/archives/2789

MongoDB 查询性能分析 explain 操作返回结果详解 https://xuexiyuan.cn/article/detail/179.html