

# MongoDB 学习笔记

(<http://www.runoob.com/mongodb/mongodb-window-install.html>)



## 目录

NoSQL 简介 .....	1
关系型数据库遵循 ACID 规则 .....	1
分布式系统.....	2
分布式计算的优点.....	2
分布式计算的缺点.....	3
什么是 NoSQL? .....	3
RDBMS vs NoSQL.....	3
CAP 定理 (CAP theorem) .....	4
NoSQL 的优点/缺点 .....	5
BASE.....	5
ACID vs BASE .....	5
NoSQL 数据库分类.....	6
MongoDB 简介 .....	6
什么是 MongoDB ?.....	6
主要特点.....	7
MongoDB 下载.....	8
MongoDB 工具.....	8
监控.....	8
GUI .....	8
window 平台安装 MongoDB .....	9
MongoDB 下载.....	9
命令行下运行 MongoDB 服务器 .....	10
将 MongoDB 服务器作为 Windows 服务运行 .....	10
MongoDB 后台管理 Shell .....	11
Linux 平台安装 MongoDB .....	11
命令行中运行 MongoDB 服务.....	12
MongoDB 后台管理 Shell .....	12
MongoDB 概念解析.....	13
数据库.....	13
文档.....	15
集合.....	17
合法的集合名.....	17
capped collections.....	17
元数据.....	18
MongoDB 数据类型 .....	18
MongoDB 数据模型.....	19
MongoDB 设计模式的一些考虑 .....	19
例子.....	19
MongoDB 创建数据库.....	21
use 命令 .....	21
语法: .....	21
示例: .....	21

MongoDB 删除数据库 .....	22
dropDatabase() 方法 .....	22
语法: .....	22
示例: .....	22
MongoDB 创建集合 .....	22
createCollection() 方法 .....	23
语法: .....	23
例子: .....	23
MongoDB 删除集合 .....	24
drop() 方法 .....	24
语法: .....	24
示例: .....	24
MongoDB 插入文档 .....	25
insert() 方法 .....	25
语法: .....	25
例子: .....	25
示例: .....	26
MongoDB 查询文档 .....	26
find() 方法 .....	26
语法: .....	27
pretty() 方法 .....	27
语法: .....	27
MongoDB 与 RDBMS Where 语句比较 .....	27
MongoDB AND 条件 .....	28
实例 .....	28
MongoDB OR 条件 .....	29
实例 .....	29
AND 和 OR 联合使用 .....	30
MongoDB 更新文档 .....	30
update() 方法 .....	31
实例 .....	31
save() 方法 .....	32
实例 .....	32
MongoDB 删除文档 .....	33
remove() 方法 .....	33
语法: .....	33
例子 .....	34
删除只有一个 .....	34
删除所有文件 .....	34
MongoDB 条件操作符 .....	34
描述 .....	34
MongoDB (>) 大于操作符 - \$gt .....	36
MongoDB (>=) 大于等于操作符 - \$gte .....	36
MongoDB (<) 小于操作符 - \$lt .....	37

MongoDB (<=) 小于操作符 - \$lte.....	37
MongoDB 使用 (<) 和 (>) 查询 - \$lt 和 \$gt.....	38
MongoDB 投影.....	38
find() 方法 .....	38
语法: .....	39
例子.....	39
MongoDB Limit/限制记录 .....	39
Limit() 方法 .....	39
语法: .....	39
示例.....	39
MongoDB Skip() 方法 .....	40
语法: .....	40
示例: .....	40
MongoDB 排序文档.....	40
sort() 方法 .....	40
语法: .....	41
例子.....	41
MongoDB 索引.....	41
ensureIndex() 方法.....	41
语法: .....	41
MongoDB 聚合.....	42
aggregate() 方法.....	43
语法: .....	43
例子: .....	43
管道的概念.....	45
管道操作符实例.....	45
MongoDB 复制（副本集） .....	46
为什么要复制? .....	46
MongoDB 复制原理.....	47
副本集特征.....	47
MongoDB 副本集设置.....	48
实例.....	48
副本集添加成员.....	48
语法.....	48
实例.....	48
MongoDB 分片.....	49
分片.....	49
为什么使用分片.....	49
MongoDB 分片.....	49
分片实例.....	50
MongoDB 备份(mongodump)与恢复(mongorestore).....	52
MongoDB 数据备份 .....	52
语法.....	52
实例.....	52

MongoDB 数据恢复 .....	53
语法 .....	53
MongoDB 监控 .....	54
mongostat 命令 .....	54
mongotop 命令 .....	55
Python 操作 MongoDB（PyMongo 模块的使用） .....	57
MongoDb 随笔，PyMongo 简单使用 .....	59
安装 MongoDb .....	59
安装 PyMongo .....	60
PyMongo 简单使用 .....	60

# NoSQL 简介

NoSQL(NoSQL = Not Only SQL)，意即"不仅仅是 SQL"。

在现代的计算系统上每天网络上都会产生庞大的数据量。

这些数据有很大一部分是由关系数据库管理系统（RDBMSs）来处理。1970 年 E.F.Codd's 提出的关系模型的论文 "A relational model of data for large shared data banks"，这使得数据建模和应用程序编程更加简单。

通过应用实践证明，关系模型是非常适合于客户服务器编程，远远超出预期的利益，今天它是结构化数据存储在网络和商务应用的主导技术。

NoSQL 是一项全新的数据库革命性运动，早期就有人提出，发展至 2009 年趋势越发高涨。

NoSQL 的拥护者们提倡运用非关系型的数据存储，相对于铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。

---

## 关系型数据库遵循 ACID 规则

事务在英文中是 transaction，和现实世界中的交易很类似，它有如下四个特性：

### 1、A (Atomicity) 原子性

原子性很容易理解，也就是说事务里的所有操作要么全部做完，要么都不做，事务成功的条件是事务里的所有操作都成功，只要有一个操作失败，整个事务就失败，需要回滚。

比如银行转账，从 A 账户转 100 元至 B 账户，分为两个步骤：1) 从 A 账户取 100 元；2) 存入 100 元至 B 账户。这两步要么一起完成，要么一起不完成，如果只完成第一步，第二步失败，钱会莫名其妙少了 100 元。

### 2、C (Consistency) 一致性

一致性也比较容易理解，也就是说数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束。

例如现有完整性约束  $a+b=10$ ，如果一个事务改变了  $a$ ，那么必须得改变  $b$ ，使得事务结束后依然满足  $a+b=10$ ，否则事务失败。

### 3、I (Isolation) 独立性

所谓的独立性是指并发的事务之间不会互相影响，如果一个事务要访问的数据正在被另外一个事务修改，只要另外一个事务未提交，它所访问的数据就不受未提交事务的影响。

比如现有有个交易是从 A 账户转 100 元至 B 账户，在这个交易还未完成的情况下，如果此时 B 查询自己的账户，是看不到新增加的 100 元的。

#### **4、D (Durability) 持久性**

持久性是指一旦事务提交后，它所做的修改将会永久的保存在数据库上，即使出现宕机也不会丢失。

---

## **分布式系统**

分布式系统（distributed system）由多台计算机和通信的软件组件通过计算机网络连接（本地网络或广域网）组成。

分布式系统是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。

因此，网络和分布式系统之间的区别更多的在于高层软件（特别是操作系统），而不是硬件。

分布式系统可以应用在在不同的平台上如：Pc、工作站、局域网和广域网上等。

---

## **分布式计算的优点**

### **可靠性（容错）：**

分布式计算系统中的一个重要的优点是可靠性。一台服务器的系统崩溃并不影响到其余的服务器。

### **可扩展性：**

在分布式计算系统可以根据需要增加更多的机器。

### **资源共享：**

共享数据是必不可少的应用，如银行，预订系统。

### **灵活性：**

由于该系统是非常灵活的，它很容易安装，实施和调试新的服务。

### **更快的速度：**

分布式计算系统可以有多个计算机的计算能力，使得它比其他系统有更快的处理速度。

### **开放系统：**

由于它是开放的系统，本地或者远程都可以访问到该服务。

### **更高的性能：**

相较于集中式计算机网络集群可以提供更高的性能（及更好的性价比）。



---

# 分布式计算的缺点

**故障排除：**：

故障排除和诊断问题。

**软件：**

更少的软件支持是分布式计算系统的主要缺点。

**网络：**

网络基础设施的问题，包括：传输问题，高负载，信息丢失等。

**安全性：**

开发系统的特性让分布式计算系统存在着数据的安全性和共享的风险等问题。

---

## 什么是 NoSQL？

NoSQL，指的是非关系型的数据库。NoSQL 有时也称作 Not Only SQL 的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。

NoSQL 用于超大规模数据的存储。（例如谷歌或 Facebook 每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

---

## RDBMS vs NoSQL

### RDBMS

- 高度组织化结构化数据
- 结构化查询语言（SQL）（SQL）
- 数据和关系都存储在单独的表中。
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

### NoSQL

- 代表着不仅仅是 SQL
- 没有声明性查询语言
- 没有预定义的模式
- 键 - 值对存储，列存储，文档存储，图形数据库

- 最终一致性，而非 ACID 属性
- 非结构化和不可预知的数据
- CAP 定理
- 高性能，高可用性和可伸缩性

## CAP 定理（CAP theorem）

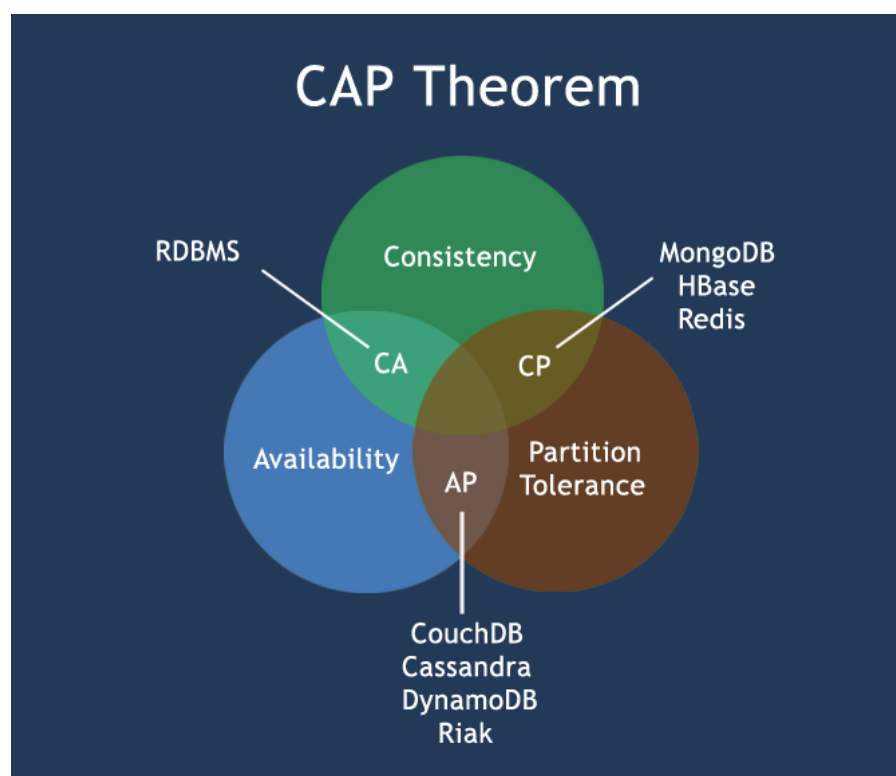
在计算机科学中, CAP 定理（CAP theorem），又被称作 布鲁尔定理（Brewer's theorem），它指出对于一个分布式计算系统来说，不可能同时满足以下三点：

- **一致性(Consistency)** (所有节点在同一时间具有相同的数据)
- **可用性(Availability)** (保证每个请求不管成功或者失败都有响应)
- **分隔容忍(Partition tolerance)** (系统中任意信息的丢失或失败不会影响系统的继续运作)

CAP 理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

- **CA** - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。
- **CP** - 满足一致性，分区容忍必的系统，通常性能不是特别高。
- **AP** - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。



---

# NoSQL 的优点/缺点

优点:

- - 高可扩展性
- - 分布式计算
- - 低成本
- - 架构的灵活性，半结构化数据
- - 没有复杂的关系

缺点:

- - 没有标准化
- - 有限的查询功能（到目前为止）
- - 最终一致是不直观的程序

---

## BASE

BASE: Basically Available, Soft-state, Eventually Consistent。 由 Eric Brewer 定义。

CAP 理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

BASE 是 NoSQL 数据库通常对可用性及一致性的弱要求原则：

- Basically Available --基本可用
- Soft-state --软状态/柔性事务。 "Soft state" 可以理解为"无连接"的，而 "Hard state" 是"面向连接"的
- Eventual Consistency --最终一致性 最终一致性， 也是是 ACID 的最终目的。

---

## ACID vs BASE

ACID	BASE
原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)

持久性 (Durable)

## NoSQL 数据库分类

类型	部分代表	特点
列存储	Hbase Cassandra Hypertable	顾名思义，是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一系列或者某几列的查询有非常大的 IO 优势。
文档存储	MongoDB CouchDB	文档存储一般用类似 json 的格式存储，存储的内容是文档型的。这样也就有机会对某些字段建立索引，实现关系数据库的某些功能。
key-value 存储	Tokyo Cabinet / Tyrant Berkeley DB MemcacheDB Redis	可以通过 key 快速查询到其 value。一般来说，存储不管 value 的格式，照单全收。（Redis 包含了其他功能）
图存储	Neo4J FlockDB	图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。
对象存储	db4o Versant	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。
xml 数据库	Berkeley DB XML BaseX	高效的存储 XML 数据，并支持 XML 的内部查询语法，比如 XQuery, Xpath。

## MongoDB 简介

### 什么是 MongoDB ?

MongoDB 是由 C++ 语言编写的，是一个基于分布式文件存储的开源数据库系统。

在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```



## 主要特点

- MongoDB 提供了一个面向文档存储，操作起来比较简单和容易。
- 你可以在 MongoDB 记录中设置任何属性的索引（如：FirstName="Sameer",Address="8 Gandhi Road"）来实现更快的排序。
- 你可以通过本地或者网络创建数据镜像，这使得 MongoDB 有更强的扩展性。
- 如果负载的增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上这就是所谓的分片。
- Mongo 支持丰富的查询表达式。查询指令使用 JSON 形式的标记，可轻易查询文档中内嵌的对象及数组。
- MongoDB 使用 update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段。
- MongoDB 中的 Map/reduce 主要是用来对数据进行批量处理和聚合操作。
- Map 和 Reduce。Map 函数调用 emit(key,value)遍历集合中所有的记录，将 key 与 value 传给 Reduce 函数进行处理。
- Map 函数和 Reduce 函数是使用 Javascript 编写的，并可以通过 db.runCommand 或 mapreduce 命令来执行 MapReduce 操作。
- GridFS 是 MongoDB 中的一个内置功能，可以用于存放大量小文件。
- MongoDB 允许在服务端执行脚本，可以用 Javascript 编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。
- MongoDB 支持各种编程语言:RUBY, PYTHON, JAVA, C++, PHP, C#等多种语言。
- MongoDB 安装简单。

# MongoDB 下载

你可以在 [mongodb](http://www.mongodb.org/downloads) 官网下载该安装包，地址为：<http://www.mongodb.org/downloads>。MongoDB 支持以

下平台：

- OS X 32-bit
  - OS X 64-bit
  - Linux 32-bit
  - Linux 64-bit
  - Windows 32-bit
  - Windows 64-bit
  - Solaris i86pc
  - Solaris 64
- 

## MongoDB 工具

有几种可用于 MongoDB 的管理工具。

### 监控

MongoDB 提供了网络和系统监控工具 **Munin**，它作为一个插件应用于 MongoDB 中。

**Gangila** 是 MongoDB 高性能的系统监视的工具，它作为一个插件应用于 MongoDB 中。

基于图形界面的开源工具 **Cacti**，用于查看 CPU 负载，网络带宽利用率，它也提供了一个应用于监控 MongoDB 的插件。

### GUI

- **Fang of Mongo** – 网页式，由 Django 和 jQuery 所构成。
- **Futon4Mongo** – 一个 CouchDB Futon web 的 mongodb 山寨版。
- **Mongo3** – Ruby 写成。
- **MongoHub** – 适用于 OSX 的应用程序。

- Opricot – 一个基于浏览器的 MongoDB 控制台，由 PHP 撰写而成。
- Database Master — Windows 的 mongodb 管理工具
- RockMongo — 最好的 PHP 语言的 MongoDB 管理工具，轻量级，支持多国语言。

## mongodb 的限制

单文档最大 16MB

单数据库最大 2GB

# window 平台安装 MongoDB

## MongoDB 下载

MongoDB 提供了可用于 32 位和 64 位系统的预编译二进制包，你可以从 MongoDB 官网下载安装，MongoDB

预编译二进制包下载地址：<http://www.mongodb.org/downloads>

### 创建数据目录

MongoDB 将数据目录存储在 db 目录下。但是这个数据目录不会主动创建，我们在安装完成后需要创建它。

请注意，数据目录应该放在根目录下（如： C:\ 或者 D:\ 等 ）。

在本教程中，我们已经在 C： 盘 安装了 mongodb，现在让我们创建一个 data 的目录然后在 data 目录里创建 db 目录。

```
c:\>cd c:\

c:\>mkdir data

c:\>cd data

c:\data>mkdir db

c:\data>cd db

c:\data\db>
```

你也可以通过 window 的资源管理器中创建这些目录，而不一定通过命令行。

# 命令行下运行 MongoDB 服务器

为了从命令提示符下运行 MongoDB 服务器，你必须从 MongoDB 目录的 bin 目录中执行 mongod.exe 文件。

```
mongod.exe --dbpath c:\data\db
```

备注：对于 win32 系统，用如下命令：

```
mongod.exe --dbpath "C:\Users\Administrator\Desktop\MongoDB_data" --storageEngine=mmapv1
```

见 [mongodb 学习资料/MongoDB 学习笔记](#)：28663 cannot start sever

如果执行成功，会输出如下信息：

```
2015-09-25T15:54:09.212+0800 I CONTROL Hotfix KB2731284 or later update is not
installed, will zero-out data files
2015-09-25T15:54:09.229+0800 I JOURNAL [initandlisten] journal dir=c:\data\db\j
ournal
2015-09-25T15:54:09.237+0800 I JOURNAL [initandlisten] recover : no journal fil
es present, no recovery needed
2015-09-25T15:54:09.290+0800 I JOURNAL [durability] Durability thread started
2015-09-25T15:54:09.294+0800 I CONTROL [initandlisten] MongoDB starting : pid=2
488 port=27017 dbpath=c:\data\db 64-bit host=WIN-1VONBJOCE88
2015-09-25T15:54:09.296+0800 I CONTROL [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2015-09-25T15:54:09.298+0800 I CONTROL [initandlisten] db version v3.0.6
.....
```

## 将 MongoDB 服务器作为 Windows 服务运行

请注意，你必须有管理权限才能运行下面的命令。执行以下命令将 MongoDB 服务器作为 Windows 服务运行：

```
mongod.exe --bind_ip yourIPadress --logpath "C:\data\dbConf\mongodb.log"
--logappend --dbpath "C:\data\db" --port yourPortNumber --serviceName
"YourServiceName" --serviceDisplayName "YourServiceName" --install
```

下表为 mongod 启动的参数说明：

参数	描述
--bind_ip	绑定服务 IP，若绑定 127.0.0.1，则只能本机访问，不指定默认本地所有 IP
--logpath	定 MongoDB 日志文件，注意是指定文件不是目录
--logappend	使用追加的方式写日志
--dbpath	指定数据库路径
--port	指定服务端口号，默认端口 27017
--serviceName	指定服务名称



<code>--serviceName</code>	指定服务名称，有多个 <code>mongodb</code> 服务时执行。
<code>--install</code>	指定作为一个 Windows 服务安装。

备注：对于 win32 系统，用如下命令：

`mongod.exe --dbpath "C:\Users\Administrator\Desktop\MongoDB_data" --storageEngine=mmapv1`

见 `mongodb 学习资料/MongoDB 学习笔记：28663 cannot start sever`

## 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.exe`

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: 3.0.6
connecting to: test
.....
```

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
>
```

`db` 命令用于查看当前操作的文档（数据库）：

```
> db
test
>
```

## Linux 平台安装 MongoDB

MongoDB 提供了 linux 平台上 32 位和 64 位的安装包，你可以在官网下载安装包。

下载地址：<http://www.mongodb.org/downloads>

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

```
curl -O https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.0.6.tgz # 下载
tar -zxvf mongodb-linux-x86_64-3.0.6.tgz # 解压
```

```
mv mongodb-linux-x86_64-3.0.6/ /usr/local/mongodb
# 将解压包
拷贝到指定目录
```

MongoDB 的可执行文件位于 bin 目录下，所以可以将其添加到 PATH 路径中：

```
export PATH=<mongodb-install-directory>/bin:$PATH
```

<mongodb-install-directory> 为你 MongoDB 的安装路径。如本文的 /usr/local/mongodb。

## 命令行中运行 MongoDB 服务

你可以再命令行中执行 mongo 安装目录中的 bin 目录执行 mongod 命令来启动 mongodb 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。

```
$ ./mongod
2015-09-25T16:39:50.549+0800 I JOURNAL [initandlisten] journal dir=/data/db/journal
2015-09-25T16:39:50.550+0800 I JOURNAL [initandlisten] recover : no journal files present, no recovery needed
2015-09-25T16:39:50.869+0800 I JOURNAL [initandlisten] preallocateIsFaster=true 3.16
2015-09-25T16:39:51.206+0800 I JOURNAL [initandlisten] preallocateIsFaster=true 3.52
2015-09-25T16:39:52.775+0800 I JOURNAL [initandlisten] preallocateIsFaster=true 7.7
```

## 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）：

```
$ cd /usr/local/mongodb/bin
$ ./mongo
MongoDB shell version: 3.0.6
connecting to: test
Welcome to the MongoDB shell.
.....
```

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

# MongoDB 概念解析

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：

SQL 术语/概念	MongoDB 术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB 不支持
primary key	primary key	主键,MongoDB 自动将 <code>_id</code> 字段设置为主键

通过下图实例，我们也可以更直观的的了解 `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。

```
$ ./mongo
MongoDB shell version: 3.0.6
connecting to: test
> show dbs
local  0.078GB
test   0.078GB
>
```

执行 "db" 命令可以显示当前数据库对象或集合。

```
$ ./mongo
MongoDB shell version: 3.0.6
connecting to: test
> 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

文档是一个键值(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.runoob.com", "name":"菜鸟教程"}
```

下表列出了 RDBMS 与 MongoDB 对应的术语：

RDBMS	MongoDB
数据库	数据库
表格	集合
行	文档
列	字段
表联合	嵌入文档
主键	主键 (MongoDB 提供了 key 为 _id )
数据库服务和客户端	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

需要注意的是：

1. 文档中的键/值对是有序的。
2. 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）。
3. **MongoDB 区分类型和大小写。**
4. **MongoDB 的文档不能有重复的键。**
5. 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)。

示例文档

下面给出的示例显示了一个博客网站,这简直是一个逗号分隔的键值对文档结构。

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials yiibai',
  url: 'http://www.yiibai.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

\_id 是一个 12 字节的十六进制数,保证每一份文件的唯一性。您可以提供\_id 同时插入文档。如果没有提供,那么 MongoDB 的每个文档提供了一个独特的 ID。这 12 个字节,前 4 个字节为当前时间戳,未来 3 个字节的机器 ID,接下来的 2 个字节的进程 id MongoDB 的服务器及剩余 3 个字节是简单的增量值。

# 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.runoob.com", "name": "菜鸟教程", "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 \$。

## 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。

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。

- 数据库不允许进行删除。使用 `drop()` 方法删除 `collection` 所有的行。
- 注意：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( 1X109)个字节。

## 元数据

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。

## MongoDB 数据类型

下表为 MongoDB 中常用的几种数据类型。

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 <b>BSON</b> （二进制的 <b>JSON</b> ）元素的最低值和最高值相对比。
Arrays	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 <b>UNIX</b> 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 <b>Date</b> 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 <b>JavaScript</b> 代码。
Regular expression	正则表达式类型。用于存储正则表达式。

# MongoDB 数据模型

在 MongoDB 中的数据有灵活的模式。在相同集合中文档并不需要有相同的一组字段或结构的公共字段的集合，文档可容纳不同类型的数据。

## MongoDB 设计模式的一些考虑

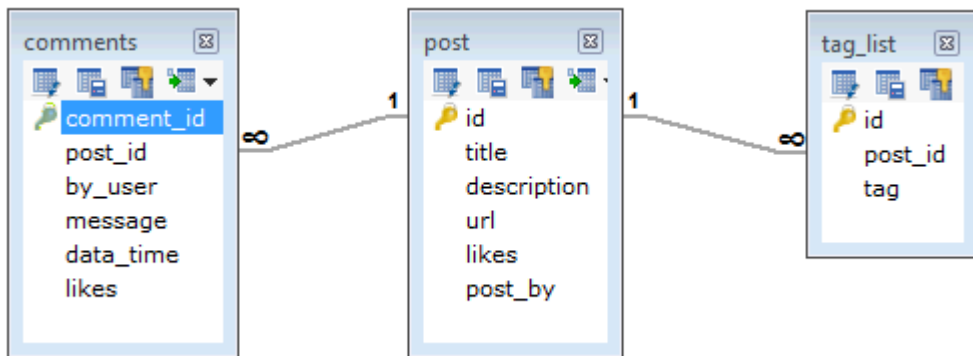
- 可根据用户要求设计架构。
- 合并对象为一个文件，如果要将它们放在一起。否则分开它们（但确保不需要连接）。
- 重复数据（有限），因为磁盘空间便宜（相比计算时间）。
- 不需要连接写入，而是读。
- 优化架构是最常见的用例。
- 在模式上做复杂的聚集。

## 例子

假设一个客户端需要一个数据库设计，设计一个博客网站，来看看 **RDBMS** 和 **MongoDB** 架构设计之间的差异。网站有以下要求。

- 每一个文章内容都有独特的标题，描述和网址。
- 每一个文章内容可以有一个或多个标签。
- 每一个文章内容都有其出版商总数喜欢的名称。
- 每一个文章内容有评论以及名字，消息，时间和喜欢的用户。
- 对于每个文章，可以是零个或多个评论。

上述要求在 RDBMS 模式设计，将有至少三个表。



*Yibai.com*

在 MongoDB 模式设计就文章一个集合，并具有以下结构：

```

{
  _id: POST_ID
  title: TITLE_OF_POST,
  description: POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
    {
      user: 'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    },
    {
      user: 'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    }
  ]
}
  
```

```
}
```

因此，尽管 RDBMS 要显示数据，需要加入三个表，而在 MongoDB 数据只是从一个集合。

# MongoDB 创建数据库

## use 命令

**MongoDB use DATABASE\_NAME 用于创建数据库。**该命令将创建一个新的数据库，如果它不存在，否则将返回现有的数据库。

## 语法:

use DATABASE 语句的基本语法如下:

```
use DATABASE_NAME
```

## 示例:

如果想创建一个数据库名称 **<mydb>**，那么 **use DATABASE** 语句如下:

```
>use mydb  
switched to db mydb
```

**要检查当前选择的数据库使用命令 db**

```
>db  
mydb
```

**如果想检查数据库列表，使用命令 show dbs.**

```
>show dbs  
local      0.78125GB  
test       0.23012GB
```

创建的数据库 **mydb** 列表中是不存在的。要显示的数据库，需要把它插入至少一个文件。

```
>db.movie.insert({"name":"tutorials yiibai"})  
>show dbs  
local      0.78125GB  
mydb       0.23012GB  
test       0.23012GB
```

在 MongoDB 默认数据库测试。如果没有创建任何数据库，然后集合将被存储在测试数据库。

上面的 **movie** 就是集合名称

# MongoDB 删除数据库

## dropDatabase() 方法

**MongoDB db.dropDatabase()** 命令是用来删除一个现有的数据库。

语法:

dropDatabase() 命令的基本语法如下:

```
db.dropDatabase()
```

这将删除选定的数据库。如果还没有选择任何数据库，然后它会删除默认的 'test' 数据库

示例:

首先，检查列表数据库通过使用命令 show dbs

```
>show dbs
local      0.78125GB
mydb       0.23012GB
test       0.23012GB
>
```

如果想删除新数据库 <mydb>，那么 dropDatabase() 命令如下:

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

现在检查的数据库列表

```
>show dbs
local      0.78125GB
test       0.23012GB
>
```

创建删除可以结合 **mongoVUE** 可视化软件监控，在数据库存放的路径下，数据库文件也会被删除。

# MongoDB 创建集合

## createCollection() 方法

**MongoDB db.createCollection(name, options)** 是用来创建集合。

语法:

基本的 createCollection() 命令语法如下:

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

在命令中, name 是要创建的集合的名称. Options 是一个文件, 用于指定配置的集合

参数	类型	描述
Name	String	要创建的集合名称
Options	Document	(可选) 指定有关内存大小和索引选项

选项参数是可选的, 所以只需要到指定的集合名称。以下是可以使用的选项列表:

字段	类型	描述
capped	Boolean	(可选) 如果为 true, 则启用封顶集合。封顶集合是固定大小的集合, 会自动覆盖最早的条目, 当它达到其最大大小。如果指定 true, 则需要也指定尺寸参数。
autoIndexID	Boolean	(可选) 如果为 true, 自动创建索引_id 字段的默认值是 false。
size	number	(可选) 指定最大大小字节封顶集合。如果封顶如果是 true, 那么你还需要指定这个字段。
max	number	(可选) 指定封顶集合允许在文件的最大数量。

当插入文档, MongoDB 第一检查大小字段封顶集合, 然后它会检查最大的字段中。

例子:

createCollection() 方法不使用选项的基本语法如下:

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

可以检查通过使用创建的集合命令 **show collections**

```
>show collections
mycollection
```

```
system.indexes
```

下面的例子显示了几个重要的选项 `createCollection()` 方法的语法:

```
>db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )
{ "ok" : 1 }
>
```

在 MongoDB 中, 不需要创建集合。当插入一些文件 MongoDB 自动创建的集合。

```
>db.yiibai.insert({"name" : "yiibai"})
>show collections
mycol
mycollection
system.indexes
yiibai
>
```

## MongoDB 删除集合

### drop() 方法

**MongoDB 的 `db.collection.drop()` 是用来从数据库中删除一个集合。**

语法:

drop() 命令的基本语法如下

```
db.COLLECTION_NAME.drop()
```

示例:

首先, 检查可用的集合在数据库 **mydb**

```
>use mydb
switched to db mydb
>show collections
mycol
mycollection
system.indexes
yiibai
>
```

现在删除集合名称为 **mycollection**

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

再次检查到数据库中的集合列表

```
>show collections  
mycol  
system.indexes  
yiibai  
>
```

drop() 方法将返回 **true**，如果选择成功收集被丢弃，否则将返回 **false**

# MongoDB 插入文档

## insert() 方法

要插入数据到 **MongoDB** 集合，需要使用 **MongoDB** 的 **insert()** 或 **save()** 方法。

## 语法

insert() 命令的基本语法如下：

```
>db.COLLECTION_NAME.insert(document)
```

## 例子

```
>db.mycol.insert({  
  _id: ObjectId(7df78ad8902c),  
  title: 'MongoDB Overview',  
  description: 'MongoDB is no sql database',  
  by: 'tutorials yiibai',  
  url: 'http://www.yiibai.com',  
  tags: ['mongodb', 'database', 'NoSQL'],  
  likes: 100  
})
```

这里 mycol 是集合的名称，如前面的教程中创建。如果集合在数据库中不存在，那么 MongoDB 将创建此集合，然后把它插入文档。

插入文档中，如果我们不指定\_id 参数，然后 MongoDB 本文档分配一个独特的 ObjectId。

\_id 是 12 个字节的十六进制数，唯一一个集合中的每个文档。12 个字节被划分如下：

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)
```

要插入单个查询的多个文档，可以传递一个数组 `insert()` 命令的文件。

## 示例

```
>db.post.insert([
{
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials yiibai',
  url: 'http://www.yiibai.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
},
{
  title: 'NoSQL Database',
  description: 'NoSQL database doesn't have tables',
  by: 'tutorials yiibai',
  url: 'http://www.yiibai.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 20,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2013,11,10,2,35),
      like: 0
    }
  ]
}
])
```

要插入文件，也可以使用 `db.post.save(document)`。如果不指定 `_id` 在文档中，然后将其 `save()` 方法和 `insert()` 方法工作一样。如果指定 `_id`，它会替换整个数据文件，其中包含 `_id` 指定 `save()` 方法。

# MongoDB 查询文档

## find() 方法

要从 MongoDB 查询集合数据，需要使用 MongoDB 的 `find()` 方法。



## 语法

基本的 find() 方法语法如下

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

find() 方法将在非结构化的方式显示所有的文件。

## pretty() 方法

结果显示在一个格式化的方式，可以使用 pretty() 方法。

语法:

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

例子

```
>db.mycol.find().pretty()
{
  "_id": ObjectId("7df78ad8902c"),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials yiibai",
  "url": "http://www.yiibai.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
>
```

除了 find() 方法外，还有一个 findOne() 法，返回一个文件。

# MongoDB 与 RDBMS 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":"菜鸟教程"}).pretty()	where by = '菜鸟教程'
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

操作	格式	范例	RDBMS 中的类似语句
不等于	{<key>:{\$ne:<value>}}	db.col.find({"likes":{\$ne:50}}).pretty()	where likes != 50

## 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，及常规 SQL 的 AND 条件。**

语法格式如下：

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

## 实例

以下实例通过 **by** 和 **title** 键来查询 菜鸟教程 中 MongoDB 教程 的数据

```
> db.col.find({"by":"菜鸟教程", "title":"MongoDB 教程"}).pretty()

{
  "_id" : ObjectId("56063f17ade2f21f36b03133"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

以上实例中类似于 WHERE 语句: **WHERE by='菜鸟教程' AND title='MongoDB 教程'**

## 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()
```

### 实例

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

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

```
    ],  
    "likes" : 100  
  }  
>
```

## 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 = '菜鸟教程' OR title = 'MongoDB 教程')**

```
>db.col.find({"likes": {$gt:50}, $or: [{"by": "菜鸟教程"}, {"title": "MongoDB 教程"}]}).pretty()  
  
{  
  "_id" : ObjectId("56063f17ade2f21f36b03133"),  
  "title" : "MongoDB 教程",  
  "description" : "MongoDB 是一个 Nosql 数据库",  
  "by" : "菜鸟教程",  
  "url" : "http://www.runoob.com",  
  "tags" : [  
    "mongodb",  
    "database",  
    "NoSQL"  
  ],  
  "likes" : 100  
}
```

## MongoDB 更新文档

**MongoDB 使用 update() 和 save() 方法来更新集合中的文档。接下来让我们详细来看下两**

个函数的应用及其区别。

## update() 方法

update() 方法用于更新已存在的文档。语法格式如下：

```
db.collection.update(  
    <query>,  
    <update>,  
    {  
        upsert: <boolean>,  
        multi: <boolean>,  
        writeConcern: <document>  
    }  
)
```

参数说明：

- **query** : update 的查询条件，类似 sql update 查询内 where 后面的。
- **update** : update 的对象和一些更新的操作符（如\$,\$inc...）等，也可以理解为 sql update 查询内 set 后面的
- **upsert** : 可选，这个参数的意思是，如果不存在 update 的记录，是否插入 objNew，true 为插入，默认是 false，不插入。
- **multi** : 可选，**mongodb** 默认是 **false**，只更新找到的第一条记录，如果这个参数为 **true**，就把按条件查出来多条记录全部更新。
- **writeConcern** : 可选，抛出异常的级别。

## 实例

我们在集合 col 中插入如下数据：

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

接着我们通过 update() 方法来更新标题(title)：

```
>db.col.update({'title': 'MongoDB 教程'},{$set: {'title': 'MongoDB'}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }) #  
输出信息
```

```
> db.col.find().pretty()
{
  "_id" : ObjectId("56064f89ade2f21f36b03136"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
>
```

可以看到标题(title)由原来的 "MongoDB 教程" 更新为了 "MongoDB"。

以上语句只会修改第一条发现的文档，如果你要修改多条相同的文档，则需要设置 **multi** 参数为 **true**。

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}},
{multi:true})
```

## save() 方法

save() 方法通过传入的文档来替换已有文档。语法格式如下：

```
db.collection.save(
  <document>,
  {
    writeConcern: <document>
  }
)
```

参数说明：

- **document**：文档数据。
- **writeConcern**：可选，抛出异常的级别。

## 实例

以下实例中我们替换了 \_id 为 56064f89ade2f21f36b03136 的文档数据：

```
>db.col.save({
```

```

    "_id" : ObjectId("56064f89ade2f21f36b03136"),
    "title" : "MongoDB",
    "description" : "MongoDB 是一个 Nosql 数据库",
    "by" : "Runoob",
    "url" : "http://www.runoob.com",
    "tags" : [
        "mongodb",
        "NoSQL"
    ],
    "likes" : 110
})

```

替换成功后，我们可以通过 `find()` 命令来查看替换后的数据

```

>db.col.find().pretty()
{
  "_id" : ObjectId("56064f89ade2f21f36b03136"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "Runoob",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "NoSQL"
  ],
  "likes" : 110
}
>

```

## MongoDB 删除文档

### remove() 方法

**MongoDB 的 `remove()` 方法用于从集合中删除文档。**`remove()` 方法接受两个参数。第一个是删除 `criteria`，第二是 `justOne` 标志：

1. **deletion criteria**：（可选）删除标准，根据文件将被删除。
2. **justOne**：（可选）如果设置为 **true** 或 **1**，然后只删除一个文件。

语法：

基本语法 `remove()` 方法如下

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

## 例子

考虑以下数据 mycol 集合。

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"
}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Yiibai Overview"
}
```

下面的例子将删除所有的文件，其标题是 'MongoDB Overview'

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Yiibai Overview"
}
>
```

## 删除只有一个

如果有多个记录且要删除的只有第一条记录，那么设置 remove()方法中 justOne 参数

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

## 删除所有文件

如果不指定删除条件，然后 MongoDB 将从集合中删除整个文件。这相当于 [SQL](#) 的 truncate 命令。

```
>db.mycol.remove()
>db.mycol.find()
>
```

# MongoDB 条件操作符

## 描述

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"runoob" 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: 'PHP 教程',
  description: 'PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['php'],
  likes: 200
})
>db.col.insert({title: 'Java 教程',
  description: 'Java 是由 Sun Microsystems 公司于1995年5月推出的高级程序设计语言。',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['java'],
  likes: 150
})
>db.col.insert({title: 'MongoDB 教程',
  description: 'MongoDB 是一个 Nosql 数据库',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['mongodb'],
  likes: 100
})
```

使用 find() 命令查看数据:

```
> db.col.find()
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
```

```
"by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由 Sun Microsystems 公司于 1995 年 5 月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

## 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：

```
> db.col.find({"likes" : {$gt : 100}})
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由 Sun Microsystems 公司于 1995 年 5 月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "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("56066542ade2f21f36b0313a"), "title" : "PHP 教程",
  "description" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。",
  "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php"
    ], "likes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程",
  "description" : "Java 是由 Sun Microsystems 公司于 1995 年 5 月推出的高级
  程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tag
  s" : [ "java" ], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教
  程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "u
  rl" : "http://www.runoob.com", "tags" : [ "mongodb" ], "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 : 150}})
```

类似于 SQL 语句：

```
Select * from col where likes < 150;
```

输出结果：

```
> db.col.find({likes : {$lt : 150}})
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教
  程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "u
  rl" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100
  }
```

## MongoDB (<=) 小于操作符 - \$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 : 150}})
```

类似于 SQL 语句：

```
Select * from col where likes <= 150;
```

输出结果：

```
> db.col.find({likes : {$lte : 150}})
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由 Sun Microsystems 公司于 1995 年 5 月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程", "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

## 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 : {$lt :200, $gt : 100}})
```

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("56066549ade2f21f36b0313b"), "title" : "Java 教程", "description" : "Java 是由 Sun Microsystems 公司于 1995 年 5 月推出的高级程序设计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "java" ], "likes" : 150 }
>
```

## MongoDB 投影

mongodb 投影意思是只选择必要的数据而不是选择一个文件的数据的整个。如果一个文档有 5 个字段, 需要显示只有 3 个, 然后选择其中只有 3 个字段。

### find() 方法

**MongoDB 的 find()方法, 在 [MongoDB 查询](#) 文档解释接受第二个可选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所有字段。要限制这一点, 需要设置的字段列表值 1 或 0。 1 用来显示字段而 0 是用来隐藏字段。**

## 语法:

find()方法具有投影基本语法如下

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

## 例子

考虑集合 myycol 具有以下的数据库

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"
}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"
}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Yiibai Overview"
}
```

下面的例子将显示文件的标题而文件的几点质疑。

```
>db.mycol.find({}, {"title":1, _id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Yiibai Overview"}
>
```

请注意\_id 字段始终显示在执行 find()方法, 如果不想这个字段, 那么需要将其设置为 0

# MongoDB Limit/限制记录

## Limit() 方法

要限制 MongoDB 中的记录, 需要使用 limit() 方法。limit() 方法接受一个数字型的参数, 这是要显示的文档数。

## 语法:

limit() 方法的基本语法如下

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

## 示例

考虑集合 myycol 具有以下的数据库

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"
}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Yiibai Overview"
}
```

下面的例子将显示只有 2 个文档，当执行文档查询。

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

如果不指定数量 `limit()` 方法的参数，它会显示从集合中的所有文件。

## MongoDB Skip() 方法

除了 `limit()` 方法，还有一个方法 `skip()` 也接受数字类型的参数，并使用跳过的文档数。

语法:

`skip()`方法基本语法如下

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

示例:

下面的例子将只显示第二个文档。

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

请注意，`skip()`方法的默认值是 0

# MongoDB 排序文档

## sort() 方法

要在 MongoDB 中的文档进行排序，需要使用 `sort()`方法。**`sort()` 方法接受一个文档，其中包含的字段列表连同他们的排序顺序。要指定排序顺序 1 和-1。 1 用于升序排列，而-1 用于降序。**

## 语法:

sort() 方法的基本语法如下

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

## 例子

考虑集合 myycol 具有以下的数据

下面的例子将显示按标题降序排序的文件。

```
>db.mycol.find({}, {"title":1, _id:0}).sort({"title":-1})
{"title":"Tutorials Yiibai Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

请注意，如果不指定排序优先，然后 sort() 方法将文档显示在升序排列。

# MongoDB 索引

索引支持的解析度的查询效率。如果没有索引，[MongoDB](#) 必须扫描每一个文档的集合，要选择那些文档相匹配的查询语句。这种扫描的效率非常低，会要求 mongod 做大数据量的处理。

索引是一种特殊的数据结构，存储设置在一个易于遍历形式的数据的一小部分。索引存储一个特定的字段或一组字段的值，在索引中指定的值的字段排列的。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

## ensureIndex() 方法

要创建一个索引，需要使用 MongoDB 的 ensureIndex()方法。

## 语法:

ensureIndex() 方法的基本语法如下

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

这里关键是要在其中创建索引，1 是按升序排列的字段名称。要创建降序索引，需要使用-1。

```
>db.mycol.ensureIndex({"title":1})
```

在 ensureIndex()方法，可以通过多个字段多个字段上创建索引。

```
>db.mycol.ensureIndex({"title":1,"description":-1})>
```

ensureIndex() 方法也可以接受的选项列表（可选），其下面给出的列表：

参数	类型	描述
background	Boolean	在后台建立索引，以便建立索引并不能阻止其他数据库活动。指定 true 建立在后台。默认值是 false.
unique	Boolean	创建唯一索引，以便收集不会接受插入索引键或键匹配现有的值存储在索引文档。指定创建唯一索引。默认值是 false.
name	string	索引的名称。如果未指定，MongoDB 中都生成一个索引名索引字段的名称和排序顺序串联.
dropDups	Boolean	创建一个唯一索引的字段，可能有重复。MongoDB 的索引只有第一次出现的一个键，从集合中删除的所有文件包含该键的后续出现的。指定创建唯一索引。默认值是 false.
sparse	Boolean	如果为 true，指数只引用文档指定的字段。这些索引使用更少的空间，但在某些情况下，特别是各种不同的表现。默认值是 false.
expireAfterSeconds	integer	指定一个值，以秒为 TTL 控制多久 MongoDB 的文档保留在此集合.
v	index version	索引版本号。默认的索引版本取决于 mongod 运行的版本在创建索引时.
weights	document	权重是从 1 到 99999 范围内的数，表示该字段的意义，相对于其他的索引字段分数.
default_language	string	对于文本索引时，决定停止词和词干分析器和标记生成规则列表的语言。默认值是 english.
language_override	string	对于文本索引时，指定的名称在文档中包含覆盖默认的语言，语言字段中。默认值是语言.

## MongoDB 聚合

聚合操作过程中的数据记录和计算结果返回。**聚合操作分组值从多个文档，并可以执行各种操作，分组数据返回单个结果。**在 SQL COUNT (\*) 和 group by 相当于 MongoDB 的聚集。**MongoDB 中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。**有点类似 sql 语句中的 count(\*)。



## aggregate() 方法

对于在 MongoDB 中聚集，应该使用 aggregate() 方法。

### 语法:

aggregate() 方法的基本语法如下

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

### 例子:

在集合中，有以下的数据库:

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by_user: 'yiibai yiibai',
  url: 'http://www.yiibai.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
},
{
  _id: ObjectId(7df78ad8902d)
  title: 'NoSQL Overview',
  description: 'No sql database is very fast',
  by_user: 'yiibai yiibai',
  url: 'http://www.yiibai.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 10
},
{
  _id: ObjectId(7df78ad8902e)
  title: 'Neo4j Overview',
  description: 'Neo4j is no sql database',
  by_user: 'Neo4j',
  url: 'http://www.neo4j.com',
  tags: ['neo4j', 'database', 'NoSQL'],
  likes: 750
},
```

现在从上面的集合，如果想显示一个列表，有很多用户写的教程，那么使用 aggregate() 方法，如下所示:

```

> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}}])
{
  "result" : [
    {
      "_id" : "yiibai yiibai",
      "num_tutorial" : 2
    },
    {
      "_id" : "Neo4j",
      "num_tutorial" : 1
    }
  ],
  "ok" : 1
}
>

```

上述使用的情况相当于 SQL 查询 `select by_user, count(*) from mycol group by by_user`

在上面的例子中，我们通过字段 `by_user` 字段对数据进行分组，并计算 `by_user` 字段相同值的总和。

下表展示了一些聚合的表达式：

表达式	描述	实例
\$sum	计算总和。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$sum : "\$likes"}}}}])</code>
\$avg	计算平均值	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$avg : "\$likes"}}}}])</code>
\$min	获取集合中所有文档对应值得最小值。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$min : "\$likes"}}}}])</code>
\$max	获取集合中所有文档对应值得最大值。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", num_tutorial : {\$max : "\$likes"}}}}])</code>
\$push	在结果文档中插入值到一个数组中。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", url : {\$push: "\$url"}}}}])</code>
\$addToSet	在结果文档中插入值到一个数组中，但不创建副本。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", url : {\$addToSet : "\$url"}}}}])</code>
\$first	根据资源文档的排序获取第一个文档数据。	<code>db.mycol.aggregate([{\$group : {_id : "\$by_user", first_url : {\$first : "\$url"}}}}])</code>

\$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"}}}])
--------	---------------------	--

## 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。

## 管道操作符实例

### 1、\$project 实例

```
db.article.aggregate(
  { $project : {
    title : 1 ,
    author : 1 ,
  }}
);
```

这样的话结果中就只剩下 `_id`, `title` 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` 可以这样：

```
db.article.aggregate(
  { $project : {
```

```
    _id : 0 ,
    title : 1 ,
    author : 1
  } } );
```

## 2.\$match 实例

```
db.articles.aggregate( [
    { $match : { score : { $gt : 70, $lte : 90 } } },
    { $group: { _id: null, count: { $sum: 1 } } }
] );
```

\$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.article.aggregate(
    { $skip : 5 } );
```

经过\$skip 管道操作符处理后，前五个文档被"过滤"掉。

# MongoDB 复制（副本集）

MongoDB 复制是将数据同步在多个服务器的过程。

**复制提供了数据的冗余备份**，并在多个服务器上存储数据副本，提高了数据的可用性， 并可以保证数据的安全性。

**复制还允许您从硬件故障和服务中断中恢复数据。**

## 为什么要复制？

- 保障数据的安全性
- 数据高可用性 (24\*7)
- 灾难恢复
- 无需停机维护（如备份，重建索引，压缩）
- 分布式读取数据

## MongoDB 复制原理

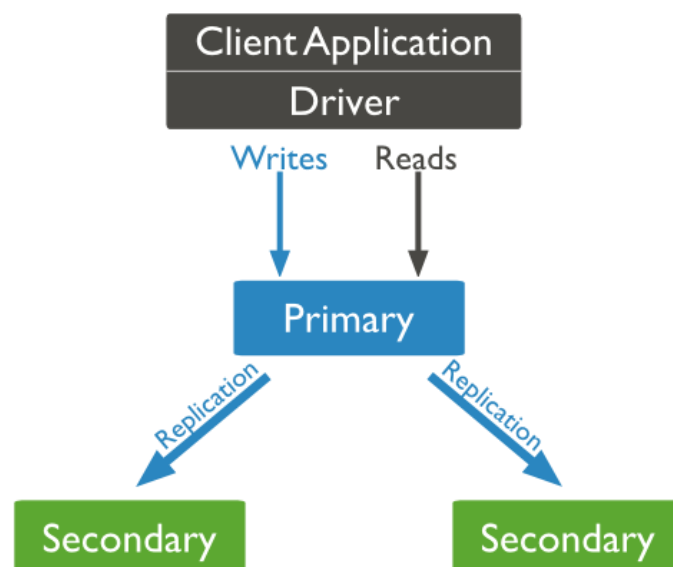
**mongodb 的复制至少需要两个节点。其中一个为主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据。**

mongodb 各个节点常见的搭配方式为：一主一从、一主多从。

主节点记录在其上的所有操作 **oplog**，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。

1. 副本集是一组中的两个或多个节点（一般至少 3 个节点是必需的）。
2. 在副本中设置一个节点是主节点和剩余节点都是次要的。
3. 从主要到次要节点的所有数据复制。
4. 自动故障转移或维修的时候，选初级建立，并选出新的主节点。
5. 失败的节点恢复后，再加入副本集和作品作为辅助节点。

MongoDB 复制结构图如下所示：



以上结构图总，客户端总主节点读取数据，在客户端写入数据到主节点是，主节点与从节点进行数据交互保障数据的一致性。

## 副本集特征

- N 个节点的集群
- 任何节点可作为主节点
- 所有写入操作都在主节点上

- 自动故障转移
- 自动恢复

## MongoDB 副本集设置

在本教程中我们使用同一个 MongoDB 来做 MongoDB 主从的实验， 操作步骤如下：

1、关闭正在运行的 MongoDB 服务器。

现在我们通过指定 `--replSet` 选项来启动 mongoDB。`--replSet` 基本语法格式如下：

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```

### 实例

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

以上实例会启动一个名为 `rs0` 的 MongoDB 实例，其端口号为 27017。

启动后打开命令提示框并连接上 mongoDB 服务。

在 Mongo 客户端使用命令 `rs.initiate()` 来启动一个新的副本集。

我们可以使用 `rs.conf()` 来查看副本集的配置

查看副本集姿态使用 `rs.status()` 命令

### 副本集添加成员

添加副本集的成员，我们需要使用多条服务器来启动 mongo 服务。进入 Mongo 客户端，并使用 `rs.add()` 方法来添加副本集的成员。

### 语法

`rs.add()` 命令基本语法格式如下：

```
>rs.add(HOST_NAME:PORT)
```

### 实例

假设你已经启动了一个名为 `mongod1.net`，端口号为 27017 的 Mongo 服务。 在客户端命令窗口使用 `rs.add()` 命令将其添加到副本集中， 命令如下所示：

```
>rs.add("mongod1.net:27017")
```

MongoDB 中你只能通过主节点将 Mongo 服务添加到副本集中，判断当前运行的 Mongo 服务是否为主节点可以使用命令 `db.isMaster()`。

MongoDB 的副本集与我们常见的主从有所不同，主从在主机宕机后所有服务将停止，而副本集在主机宕机后，副本会接管主节点成为主节点，不会出现宕机的情况。

## MongoDB 分片

### 分片

在 MongoDB 里面存在另一种集群，就是分片技术，可以满足 MongoDB 数据量大量增长的需求。

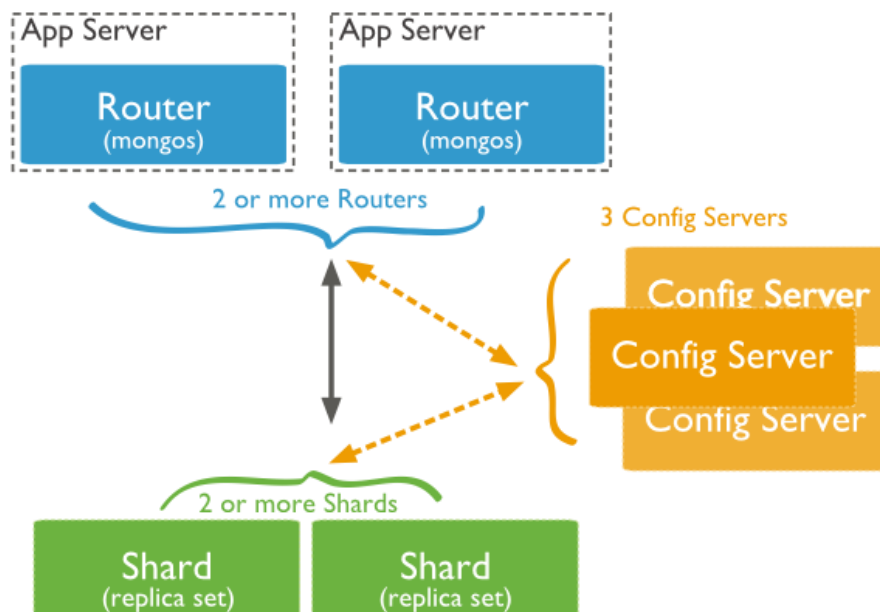
当 MongoDB 存储海量的数据时，一台机器可能不足以存储数据也足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。

### 为什么使用分片

- 复制所有的写入操作到主节点
- 延迟的敏感数据会在主节点查询
- 单个副本集限制在 12 个节点
- 当请求量巨大时会出现内存不足。
- 本地磁盘不足
- 垂直扩展价格昂贵

### MongoDB 分片

下图展示了在 MongoDB 中使用分片集群结构分布：



上图中主要有如下所述三个主要组件：

- **Shard:**  
用于存储实际的数据块，实际生产环境中一个 **shard server** 角色可由几台机器组一个 **relica set** 承担，防止主机单点故障
- **Config Server:**  
**mongod** 实例，存储了整个 **ClusterMetadata**，其中包括 **chunk** 信息。
- **Query Routers:**  
前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。
  - 碎片: 碎片被用来存储数据。它们提供了高可用性和数据的一致性。在生产环境中，每个碎片是一个单独的副本集。
  - 配置服务器: 配置服务器集群的元数据存储。该数据包含集群的数据碎片的映射。查询路由器使用这个元数据，操作具体的碎片。在生产环境中，有整整 3 分片集群配置服务器。
  - 查询路由: 查询路由基本上是 **Mongos** 实例，客户端应用程序界面和直接操作相应的碎片。查询路由过程目标操作的碎片，然后将结果返回到客户端。分片集群可以包含多个查询路由来划分客户端请求负载。客户端发送请求到一个查询路由。一般分片集群有许多查询路由。

## 分片实例

分片结构端口分布如下：

Shard Server 1: 27020



```
Shard Server 2: 27021
Shard Server 3: 27022
Shard Server 4: 27023
Config Server : 27100
Route Process: 40000
```

### 步骤一：启动 Shard Server

```
[root@100 /]# mkdir -p /www/mongoDB/shard/s0
[root@100 /]# mkdir -p /www/mongoDB/shard/s1
[root@100 /]# mkdir -p /www/mongoDB/shard/s2
[root@100 /]# mkdir -p /www/mongoDB/shard/s3
[root@100 /]# mkdir -p /www/mongoDB/shard/log
[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27020 --dbpath=/www/mongoDB/shard/s0 --logpath=/www/mongoDB/shard/log/s0.log --logappend --fork
....
[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27023 --dbpath=/www/mongoDB/shard/s3 --logpath=/www/mongoDB/shard/log/s3.log --logappend --fork
```

### 步骤二：启动 Config Server

```
[root@100 /]# mkdir -p /www/mongoDB/shard/config
[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27100 --dbpath=/www/mongoDB/shard/config --logpath=/www/mongoDB/shard/log/config.log --logappend --fork
```

**注意：**这里我们完全可以像启动普通 mongodb 服务一样启动，不需要添加—shardsvr 和 configsvr 参数。因为这两个参数的作用就是改变启动端口的，所以我们自行指定了端口就可以。

### 步骤三：启动 Route Process

```
/usr/local/mongoDB/bin/mongos --port 40000 --configdb localhost:27100 --fork --logpath=/www/mongoDB/shard/log/route.log --chunkSize 500
```

mongos 启动参数中，chunkSize 这一项是用来指定 chunk 的大小的，单位是 MB，默认大小为 200MB。

### 步骤四：配置 Sharding

接下来，我们使用 MongoDB Shell 登录到 mongos，添加 Shard 节点

```
[root@100 shard]# /usr/local/mongoDB/bin/mongo admin --port 40000
MongoDB shell version: 2.0.7
connecting to: 127.0.0.1:40000/admin
mongos> db.runCommand({ addshard:"localhost:27020" })
{ "shardAdded" : "shard0000", "ok" : 1 }
.....
mongos> db.runCommand({ addshard:"localhost:27029" })
```

```
{ "shardAdded" : "shard0009", "ok" : 1 }
mongos> db.runCommand({ enablesharding:"test" }) #设置分片存储的数据库
{ "ok" : 1 }
mongos> db.runCommand({ shardcollection: "test.log", key: { id:1,time:1}})
{ "collectionsharded" : "test.log", "ok" : 1 }
```

步骤五： 程序代码内无需太大更改，直接按照连接普通的 mongo 数据库那样，将数据库连接接入接口 40000

# MongoDB 备份(mongodump)与恢复(mongorestore)

## MongoDB 数据备份

在 MongoDB 中我们使用 mongodump 命令来备份 MongoDB 数据。该命令可以导出所有数据到指定目录中。

**mongodump 命令可以通过参数指定导出的数据量级转存的服务器。**

## 语法

mongodump 命令脚本语法如下：

```
>mongodump -h dbhost -d dbname -o dbdirectory
```

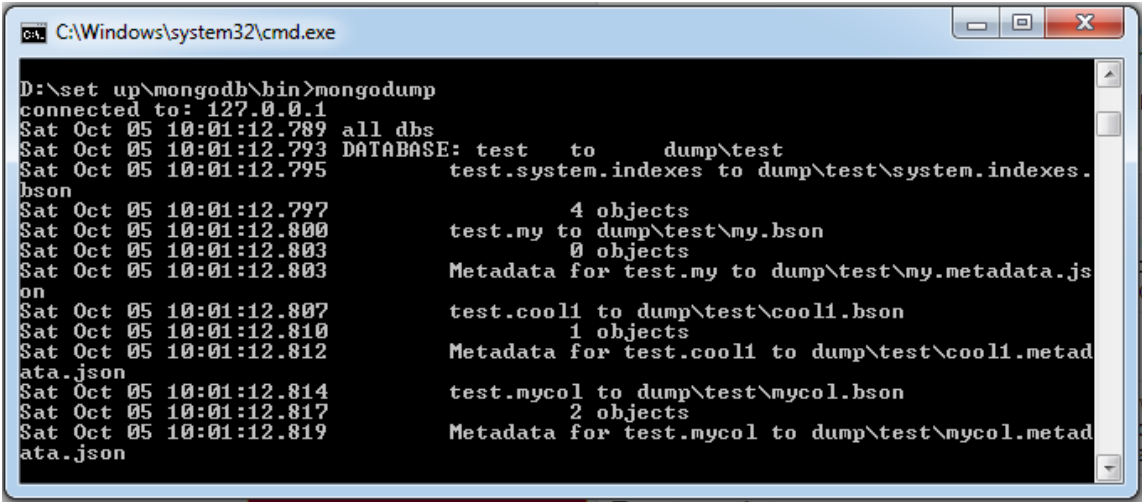
- **-h:**  
**MongoDB 所在服务器地址，例如：127.0.0.1，当然也可以指定端口号：127.0.0.1:27017**
- **-d:**  
**需要备份的数据库实例，例如：test**
- **-o:**  
**备份的数据存放位置，例如：c:\data\dump，当然该目录需要提前建立，在备份完成后，系统自动在 dump 目录下建立一个 test 目录，这个目录里面存放该数据库实例的备份数据。**

## 实例

在本地使用 27017 启动你的 mongod 服务。打开命令提示符窗口，进入 MongoDB 安装目录的 bin 目录输入命令 mongodump:

```
>mongodump
```

执行以上命令后，客户端会连接到 ip 为 127.0.0.1 端口号为 27017 的 MongoDB 服务上，并备份所有数据到 bin/dump/ 目录中。命令输出结果如下：



```
C:\Windows\system32\cmd.exe
D:\set up\mongodb\bin>mongodump
connected to: 127.0.0.1
Sat Oct 05 10:01:12.789 all dbs
Sat Oct 05 10:01:12.793 DATABASE: test to dump\test
Sat Oct 05 10:01:12.795 test.system.indexes to dump\test\system.indexes.
bson
Sat Oct 05 10:01:12.797 4 objects
Sat Oct 05 10:01:12.800 test.my to dump\test\my.bson
Sat Oct 05 10:01:12.803 0 objects
Sat Oct 05 10:01:12.803 Metadata for test.my to dump\test\my.metadata.js
on
Sat Oct 05 10:01:12.807 test.cool1 to dump\test\cool1.bson
Sat Oct 05 10:01:12.810 1 objects
Sat Oct 05 10:01:12.812 Metadata for test.cool1 to dump\test\cool1.metad
ata.json
Sat Oct 05 10:01:12.814 test.mycol to dump\test\mycol.bson
Sat Oct 05 10:01:12.817 2 objects
Sat Oct 05 10:01:12.819 Metadata for test.mycol to dump\test\mycol.metad
ata.json
```

mongodump 命令可选参数列表如下所示：

语法	描述	实例
mongodump --host HOST_NAME --port PORT_NUMBER	该命令将备份所有 MongoDB 数据	mongodump --host w3cschool.cc --port 27017
mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY		mongodump --dbpath /data/db/ --out /data/backup/
mongodump --collection COLLECTION --db DB_NAME	该命令将备份指定数据库的集合。	mongodump --collection mycol --db test

## MongoDB 数据恢复

**mongodb 使用 mongorerstore 命令来恢复备份的数据。**

### 语法

mongorestore 命令脚本语法如下：

```
>mongorestore -h dbhost -d dbname --directoryperdb dbdirectory
```

- **-h:**  
**MongoDB 所在服务器地址**
- **-d:**  
**需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如 test2**
- **--directoryperdb:**

备份数据所在位置，例如：c:\data\dump\test，这里为什么要多加一个 test，而不是备份时候的 dump，读者自己查看提示吧！

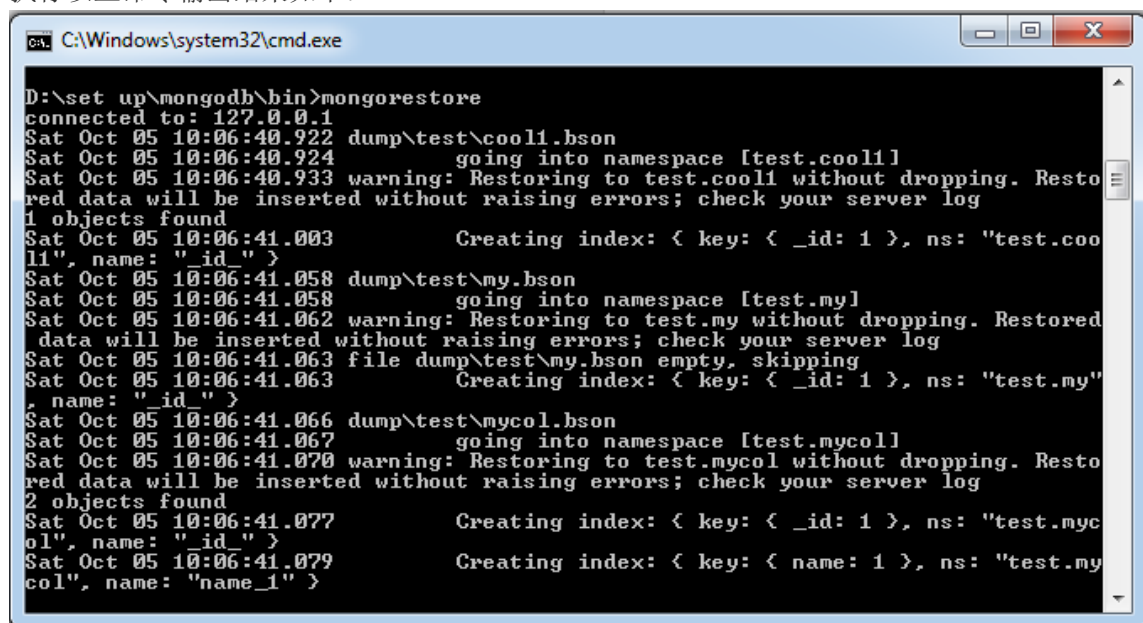
- **--drop:**

恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

接下来我们执行以下命令：

```
>mongorestore
```

执行以上命令输出结果如下：



```
C:\Windows\system32\cmd.exe
D:\set up\mongodb\bin>mongorestore
connected to: 127.0.0.1
Sat Oct 05 10:06:40.922 dump\test\cool1.bson
Sat Oct 05 10:06:40.924 going into namespace [test.cool1]
Sat Oct 05 10:06:40.933 warning: Restoring to test.cool1 without dropping. Restored data will be inserted without raising errors; check your server log
1 objects found
Sat Oct 05 10:06:41.003 Creating index: < key: < _id: 1 >, ns: "test.cool1", name: "_id" >
Sat Oct 05 10:06:41.058 dump\test\my.bson
Sat Oct 05 10:06:41.058 going into namespace [test.my]
Sat Oct 05 10:06:41.062 warning: Restoring to test.my without dropping. Restored data will be inserted without raising errors; check your server log
Sat Oct 05 10:06:41.063 file dump\test\my.bson empty, skipping
Sat Oct 05 10:06:41.063 Creating index: < key: < _id: 1 >, ns: "test.my", name: "_id" >
Sat Oct 05 10:06:41.066 dump\test\mycol.bson
Sat Oct 05 10:06:41.067 going into namespace [test.mycoll]
Sat Oct 05 10:06:41.070 warning: Restoring to test.mycoll without dropping. Restored data will be inserted without raising errors; check your server log
2 objects found
Sat Oct 05 10:06:41.077 Creating index: < key: < _id: 1 >, ns: "test.mycoll", name: "_id" >
Sat Oct 05 10:06:41.079 Creating index: < key: < name: 1 >, ns: "test.mycoll", name: "name_1" >
```

## MongoDB 监控

在你已经安装部署并允许 MongoDB 服务后，你必须要了解 MongoDB 的运行情况，并查看 MongoDB 的性能。这样在大流量得情况下可以很好的应对并保证 MongoDB 正常运作。

MongoDB 中提供了 mongostat 和 mongotop 两个命令来监控 MongoDB 的运行情况。

### mongostat 命令

**mongostat** 是 **mongodb** 自带的状态检测工具，在命令行下使用。它会间隔固定时间获取 **mongodb** 的当前运行状态，并输出。如果你发现数据库突然变慢或者有其他问题的话，你第一手的操作就考虑采用 **mongostat** 来查看 **mongo** 的状态。



```
C:\Windows\system32\cmd.exe - mongotop
local.system.users      0ms      0ms      0ms
local.system.replset    0ms      0ms      0ms
local.startup_log       0ms      0ms      0ms
ns      total      read      write
2013-10-06T13:53:28
test.system.users      0ms      0ms      0ms
local.system.users     0ms      0ms      0ms
local.system.replset   0ms      0ms      0ms
local.startup_log      0ms      0ms      0ms
ns      total      read      write
2013-10-06T13:53:29
test.system.users      0ms      0ms      0ms
local.system.users     0ms      0ms      0ms
local.system.replset   0ms      0ms      0ms
local.startup_log      0ms      0ms      0ms
ns      total      read      write
2013-10-06T13:53:30
test.system.users      0ms      0ms      0ms
local.system.users     0ms      0ms      0ms
local.system.replset   0ms      0ms      0ms
local.startup_log      0ms      0ms      0ms
```

带参数实例

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop 10
```

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop
connected to: 127.0.0.1
ns      total      read      write
2013-03-29T04:12:17
local.system.replset    0ms      0ms      0ms
local.system.namespaces 0ms      0ms      0ms
local.system.indexes    0ms      0ms      0ms
admin.system.indexes    0ms      0ms      0ms
admin.                  0ms      0ms      0ms
CpsCommodityInfo.system.namespaces 0ms      0ms      0ms
CpsCommodityInfo.system.js 0ms      0ms      0ms
ns      total      read      write
2013-03-29T04:12:18
local.system.replset    0ms      0ms      0ms
local.system.namespaces 0ms      0ms      0ms
local.system.indexes    0ms      0ms      0ms
admin.system.indexes    0ms      0ms      0ms
admin.                  0ms      0ms      0ms
CpsCommodityInfo.system.namespaces 0ms      0ms      0ms
CpsCommodityInfo.system.js 0ms      0ms      0ms
```

后面的 10 是<sleepime>参数，可以不使用，等待的时间长度，以秒为单位，mongotop 等待调用之间。通过的默认 mongotop 返回数据的每一秒。

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop --locks
```

报告每个数据库的锁的使用中，使用 mongotop - 锁，这将产生以下输出：

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop --locks
connected to: 127.0.0.1

      db      total      read      write
2013-03-29T04:21:59
      local      0ms      0ms      0ms
      admin      0ms      0ms      0ms
      CpsCommodityInfo      0ms      0ms      0ms
      .      0ms      0ms      0ms

      db      total      read      write
2013-03-29T04:22:00
      local      0ms      0ms      0ms
      admin      0ms      0ms      0ms
      CpsCommodityInfo      0ms      0ms      0ms
      .      0ms      0ms      0ms

      db      total      read      write
2013-03-29T04:22:01
```

输出结果字段说明：

- **ns:**  
包含数据库命名空间，后者结合了数据库名称和集合。
- **db:**  
包含数据库的名称。名为 . 的数据库针对全局锁定，而非特定数据库。
- **total:**  
mongod 花费的时间工作在这个命名空间提供总额。
- **read:**  
提供了大量的时间，这 mongod 花费在执行读操作，在此命名空间。
- **write:**  
提供这个命名空间进行写操作，这 mongod 花了大量的时间。

## Python 操作 MongoDB (PyMongo 模块的使用)

PyMongo 模块是 Python 对 MongoDB 操作的接口包，代码主要实现对 MongoDB 的几种操作：增删改查以及排序等功能，网上有很多资料，本文仅充当学习笔记。

```

#!/usr/bin/env python
#coding:utf-8
# Author:      --<qingfengkuyu>
# Purpose: MongoDB 的使用
# Created: 2014/4/14
#32 位的版本最多只能存储 2.5GB 的数据（NoSQLFan：最大文件尺寸为 2G，生产环境推荐 64 位）

import pymongo
import datetime
import random

#创建连接
conn = pymongo.MongoClient('10.11.1.70', 27017)
#连接数据库
db = conn.study
# study 为数据库的名字
#db = conn['study']

#打印所有聚集名称，连接聚集
print u'所有聚集:', db.collection_names()
posts = db.post
# post 为集合的名字
#posts = db['post']
print posts

#插入记录
new_post = {"AccountID":22, "UserName":"libing", 'date':datetime.datetime.now()}
new_posts = [{"AccountID":22, "UserName":"liuw", 'date':datetime.datetime.now()},
              {"AccountID":23, "UserName":"urling", 'date':datetime.datetime.now()}]

#每条记录插入时间都不一样

posts.insert(new_post)
#posts.insert(new_posts)#批量插入多条数据

#删除记录
print u'删除指定记录:\n', posts.find_one({"AccountID":22, "UserName":"libing"})
posts.remove({"AccountID":22, "UserName":"libing"})

#修改聚集内的记录
posts.update({"UserName":"urling"}, {"$set":{"AccountID":random.randint(20, 50)}})

#查询记录，统计记录数量
print u'记录总计为: ', posts.count(), posts.find().count()
print u'查询单条记录:\n', posts.find_one()

```



```

print posts.find_one({"UserName":"liuw"})

#查询所有记录
print u' 查询多条记录:'
#for item in posts.find():#查询全部记录
#for item in posts.find({"UserName":"urling"}):#查询指定记录
#for item in posts.find().sort("UserName"):#查询结果根据 UserName 排序，默认为升序
#for item in posts.find().sort("UserName", pymongo.ASCENDING):#查询结果根据 UserName 排序，
ASCENDING 为升序, DESCENDING 为降序
for item in posts.find().sort([("UserName", pymongo.ASCENDING), ('date', pymongo.DESCENDING)]):#查
询结果根据多列排序
    print item

#高级查询
print " _____ "
print '''collection.find({"age":{"$gt":"10"}})'''
print " _____ "
for data in collection.find({"age":{"$gt":"10"}}).sort("age"):
    print data

#查看查询语句的性能
#posts.create_index([("UserName", pymongo.ASCENDING), ("date", pymongo.DESCENDING)])#加索引
print
posts.find().sort([("UserName", pymongo.ASCENDING), ('date', pymongo.DESCENDING)]).explain()["cur
sor"]#未加索引用 BasicCursor 查询记录
print
posts.find().sort([("UserName", pymongo.ASCENDING), ('date', pymongo.DESCENDING)]).explain()["nsc
anned"]#查询语句执行时查询的记录数

```

# MongoDb 随笔，PyMongo 简单使用

## 安装 MongoDb

[MongoDb](#) 下载对应的系统版本的可执行文件

本人系统环境:rhel-server-6.2-x86\_64

解压缩包 tar zxvf mongodb-linux-x86\_64-rhel62-3.0.2.tgz

可以查看目录下的 README，了解各个可执行文件的作用。

简单启动命令 mkdir db; mongo --dbpath=./db

mongo --help 可以获取更多帮助。

## 安装 PyMongo

安装命令: `pip install pymongo`

更多关于 pip 的应用可参考 [Python](#) 下 `pip pydoc 2to3` 等工具

## PyMongo 简单使用

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import pymongo
import datetime

def get_db():
    # 建立连接
    client = pymongo.MongoClient(host="10.244.25.180", port=27017)
    db = client['example']
    #或者 db = client.example
    return db

def get_collection(db):
    # 选择集合 (mongo 中 collection 和 database 都是延时创建的)
    coll = db['informations']
    print db.collection_names()
    return coll

def insert_one_doc(db):
    # 插入一个 document
    coll = db['informations']
    information = {"name": "quyang", "age": "25"}
    information_id = coll.insert(information)
    print information_id

def insert_multi_docs(db):
    # 批量插入 documents, 插入一个数组
    coll = db['informations']
    information = [{"name": "xiaoming", "age": "25"}, {"name": "xiaoqiang", "age":
"24"}]
    information_id = coll.insert(information)
    print information_id
```

```

def get_one_doc(db):
    # 有就返回一个，没有就返回 None
    coll = db['informations']

    print coll.find_one() # 返回第一条记录
    print coll.find_one({"name": "quyang"})
    print coll.find_one({"name": "none"})

def get_one_by_id(db):
    # 通过 objectId 来查找一个 doc
    coll = db['informations']
    obj = coll.find_one()
    obj_id = obj["_id"]
    print "_id 为 ObjectId 类型, obj_id:" + str(obj_id)

    print coll.find_one({"_id": obj_id})
    # 需要注意这里的 obj_id 是一个对象，不是一个 str，使用 str 类型作为 _id 的值无法找到记录
    print "_id 为 str 类型 "
    print coll.find_one({"_id": str(obj_id)})
    # 可以通过 ObjectId 方法把 str 转成 ObjectId 类型
    from bson.objectid import ObjectId

    print "_id 转换成 ObjectId 类型"
    print coll.find_one({"_id": ObjectId(str(obj_id))})

def get_many_docs(db):
    # mongo 中提供了过滤查找的方法，可以通过各种条件筛选来获取数据集，还可以对数据进行计数，排序等处理

    coll = db['informations']

    #ASCENDING = 1 升序;DESCENDING = -1 降序;default is ASCENDING
    for item in coll.find().sort("age", pymongo.DESCENDING):
        print item

    count = coll.count()
    print "集合中所有数据 %s 个" % int(count)

    #条件查询
    count = coll.find({"name": "quyang"}).count()
    print "quyang: %s"%count

def clear_all_datas(db):

```

```

#清空一个集合中的所有数据
db["informations"].remove()

if __name__ == '__main__':
    db = get_db()
    my_collection = get_collection(db)
    post = {"author": "Mike", "text": "My first blog post!", "tags": ["mongodb",
"python", "pymongo"],
            "date": datetime.datetime.utcnow()}

    # 插入记录
    my_collection.insert(post)
    insert_one_doc(db)

    # 条件查询
    print my_collection.find_one({"x": "10"})

    # 查询表中所有的数据
    for iii in my_collection.find():
        print iii
    print my_collection.count()

    my_collection.update({"author": "Mike"},
                        {"author": "quyang", "text": "My first blog post!", "tags":
["mongodb", "python", "pymongo"],
                        "date": datetime.datetime.utcnow()})

    for jjj in my_collection.find():
        print jjj
    get_one_doc(db)
    get_one_by_id(db)
    get_many_docs(db)
    # clear_all_datas(db)

```