Contents

[Sqlite3 2](#_Toc457640964)

[创建sql文件 2](#_Toc457640965)

[打开或创建数据库 3](#_Toc457640966)

[导入sql文件 3](#_Toc457640967)

[CRUD操作 3](#_Toc457640968)

[事务操作 3](#_Toc457640969)

[Redis (remote dictionary service) 3](#_Toc457640970)

[MongoDB 6](#_Toc457640971)

[安装 6](#_Toc457640972)

[启动mongod --ubuntu 7](#_Toc457640973)

[MongoDB shell 7](#_Toc457640974)

[命令行 8](#_Toc457640975)

[基本数据类型 10](#_Toc457640976)

[使用shell执行脚本 10](#_Toc457640977)

[索引 11](#_Toc457640978)

[创建索引，基于字段或多个字段 11](#_Toc457640979)

[固定集合 11](#_Toc457640980)

[TTL索引 11](#_Toc457640981)

[全文本索引 12](#_Toc457640982)

[地理空间索引 12](#_Toc457640983)

[GridFS存储文件 12](#_Toc457640984)

[聚合 13](#_Toc457640985)

[聚合框架 13](#_Toc457640986)

[MapReduce 14](#_Toc457640987)

[PostgreSQL 16](#_Toc457640988)

# Sqlite3

SQLite是一个进程中关系数据库，SQLite设计为嵌入在应用中，而不是作为一个单独的数据库服务器程序

支持数据库大小至2TB

ACID事务

$sudo apt-get install sqlite3 安装sqlite3

## 创建sql文件

create table project (

name text primary key,

description text,

deadline date

);

create table task (

id integer primary key autoincrement not null,

priority integer default 1,

details text,

status text,

deadline date,

completed\_on date,

project text not null references project(name)

);

insert into project (name, description, deadline)

values ('pymotw', 'Python MOdule of the Week', '2010-11-01');

insert into task (details, status, deadline, project)

values('write about select', 'done', '2010-10-03', 'pymotw');

insert into task (details, status, deadline, project)

values('write about random', 'waiting', '2010-10-10', 'pymotw');

insert into task (details, status, deadline, project)

values('write about sqlite3', 'actie', '2010-10-17', 'pymotw');

## 打开或创建数据库

$sqlite3 todo.db3

## 导入sql文件

sqlite> .read todo\_schema.sql; 执行sql文件

## CRUD操作

sqlite> select \* from task; 查询

sqlite> delete from task where id=2; 删除

sqlite> insert into task (details, status, deadline, project) 插入

values('write about random', 'waiting', '2010-10-10', 'pymotw');

sqlite> update task set status='done' where id=4; 更新

## 事务操作

sqlite> begin;

sqlite> … //crud operation

sqlite> commit; 递交

(或者sqlite> rollback; 撤销)

# Redis (remote dictionary service)

也称为数据结构服务器，发布订阅系统，作为有用的数据结构算法和程序的工具包。作为栈，队列或优先队列， 对象存储系统（通过hashtable），集合；支持集合操作（并，交，差），提供原子命令，和多步命令的事务机制；键到期及缓存

1. Download and install [Redis-x64-3.0.500-rc2.msi](https://github.com/MSOpenTech/redis/releases/download/win-3.0.500-rc2/Redis-x64-3.0.500-rc2.msi) from <https://github.com/MSOpenTech/redis/releases>
2. Run Redis server:

C:\Program Files\redis > redis-server redis.windows.conf

1. Run Redis cli:

C:\Program Files\redis > redis-cli

127.0.0.1:6379> ping

// 键值

>set key value //mset key1 value1 key2 value2 …

> get key //mget key1 key2

//hash table (map) 嵌套的Redis对象

mset key1 value1 key2 value2

mget key1 key2

hmset key key1 value1 key2 value2 …

hvals key

hkeys key

for example:

mset user:qzlin:name “qizhong lin” user:qzlin:password changeit //使用:作为分离的键

hmset user:qzlin name “qizhong lin” password changeit //不使用分离的键

hvals user:qzlin

hkeys user:qzlin

// 列表

rpush qzlin:wishlist google yahoo baidu //建立列表

lrange qzlin:wishlist 0 -1 //取子集

lrem qzlin:wishlist 0 google //删除元素

rpoplpush qzlin:wishlist qzlin:visited //从左列表弹出元素， 在右列表压入元素

// set 集合

sad key value1 value2 … //创建集合

smembers key //获取集合

sinterstore key key1 key2 //求交集

sdiffstore key key1 key2 //差集

sunionstore key key1 key2 //并集key1, key2, 结果存入key中

for example:

sad news nytimes.com baidu.com

smembers news

// 有序集合 (随机存取的优先级队列)

zadd key score1 key1 score2 key2

zrange key idx0 idx 1

zrangebyscore key score0 score1

// 加权后求并集

zunionstore destination numkeys key1 key2 … [weights weight1 weight2 …] [aggregate sum|min|max]

for example:

zadd visits 500 google 9 yahoo 10 baidu

zunionstore importance 2 visits votes weights 1 2 aggregate sum

事务: multi … exec ; multi和exec内多命令按事务执行，在使用multi命令时，命令在实际上并不执行，而是排入队列，然后按顺序执行。

到期：有助于避免总的键集无限增长，做法是安排Redis经过一定的时间就删除一个键-值对

expire key seconds

setex key second value // 设置key value的生命

ttl key //获取key的生命

persist key //最近使用缓存算法(MRU, Most Recently Used)： 每当你检索一个值时，更新它的到期时间，确保你最近使用的键将继续保留在Redis中，而被忽视的键将正常到期

Redis命名空间称为数据库(database),以数字为键。默认的命名空间0（也称为数据库0）交互

// select 0

set greeting hello

get greeting

select 1 //通过select切换到数据库1

get greeting //返回(nil)

所有的数据库都运行在同一服务器实例内，Redis就允许用move命令，在不同命令空间之间移动键. 若单个Redis服务器运行不同的应用程序，又要允许这些应用程序相互之间交换数据

move greeting 2

select 2

get greeting

别的操作命令： rename, type, del, flushdb, flushall.

//发布-接收模式

127.0.0.1:6379> brpop comments 300 //某客户端监听 key=comments, 超时为5分钟,

127.0.0.1:6379> lpush comments “hello world!” //某客户端发布 key-values

Key points:

redis server启动后，独占进程，能否改为后台服务？

redis server服务是单线程的，能否在同一台机器上启动多个实例？也就是修改默认端口6379？

$redis-server --port 6378 --daemonize yes 服务端

$redis-cli -p 6378 客户端

Redis允许在运行的过程中，在不重启服务器的情况下更改服务器配置

127.0.0.1:6378> config set tcp-keepalive 60

127.0.0.1:6378> config set port 6380

127.0.0.1:6380>

注意：先把心跳包发送时间间隔修改成60秒。

Redis主从复制

特性：

* redis使用异步复制
* 一个master拥有多个slave
* master死了，会自举一个slave作为master

做法：

打开三个终端启动三个redis 服务实例，然后启动三个redis client连接

$redis-server --daemonize yes

$redis-server --daemonize yes --slaveof 127.0.0.1 10000

$redis-server --daemonize yes --slaveof 127.0.0.1 10000

从而两个slave会同步一个主

Redis集群

集群技术是构建高性能网站架构的重要手段，试想在网站承受高并发访问压力的同时，还需要从海量数据中查询出满足条件的数据，并快速响应，我们必然想到的是将数据进行切片，把数据根据某种规则放入多个不同的服务器节点，来降低单节点服务器的压力。

集群主要是利用切片技术来组建的集群

集群要实现的目的是要将不同的 key 分散放置到不同的 redis 节点，这里我们需要一个规则或者算法.

Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点。

使用哈希槽的好处就在于可以方便的添加或移除节点。

当需要增加节点时，只需要把其他节点的某些哈希槽挪到新节点就可以了；

当需要移除节点时，只需要把移除节点上的哈希槽挪到其他节点就行了；

在新增或移除节点的时候不要让我们先停掉所有的 redis 服务

做法：

先启动三个redis服务实例

创建集群，使三个实例互相通讯, （具体见网上）

$./redis-tri.rb create –replicas 0 127.0.0.1:9001 127.0.0.1:9002 127.0.0.1:9003

连接127.0.0.1:9001的客户端，设置或获取的数据，可能被redis cluster自动重定向到127.0.0.1:9003。

# MongoDB

## 安装

Install and set env

install mongoDB in ubuntu

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

Import the public key used by the package management system.

$sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927

Create a list file for MongoDB

$echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list

Reload local package database.

$sudo apt-get update

Install the MongoDB packages.

$sudo apt-get install -y mongodb-org

## 启动mongod --ubuntu

The MongoDB instance stores its data files in /var/lib/mongodb and its log files in /var/log/mongodb by default, and runs using the mongodb user account. You can specify alternate log and data file directories in /etc/mongod.conf. See systemLog.path and storage.dbPath for additional information.

Start MongoDB

$sudo service mongod start

Verify that the mongod process has started successfully by checking the contents of the log file at /var/log/mongodb/mongod.log for a line reading

[initandlisten] waiting for connections on port <port>

Stop MongoDB

$sudo service mongod stop

Restart MongoDB

$sudo service mongod restart

类似window方式使用

卸载mongo包

$dpkg -l | grep mongo

$sudo apt-get remove --purge mongodb-org

$ sudo apt-get remove --purge mongodb-org-mongos

$ sudo apt-get remove --purge mongodb-org-server

$ sudo apt-get remove --purge mongodb-org-server

$ sudo apt-get remove --purge mongodb-org-tools

wget https://fastdl.mongodb.org/linux/mongodb-linux-x86\_64-ubuntu1404-3.2.8.tgz

tar zxf mongodb...gz

$ gedit ~/.bashrc 配置环境变量

PATH=$PATH:~/Downloads/mongo…/bin

export PATH

$source ~/.bashrc

$mkdir -p /data/db

$chown -R $USER:$USER /data/db

启动mongod --window

>mongod --dbpath ./data 默认是c:/data/db

默认情况下，MongoDB监听27017端口，mongod还启动一个HTTP服务器，监听数字比主端口号高1000的端口，也就是28017端口

$curl localhost:28017

## MongoDB shell

MongoDB自带JavaScript shell,可以shell中使用命令行与MongoDB实例交互。Shell是一个功能完备的JavaScript解释器，可运行任意JavaScript程序

>mongo localhost:27017/test

启动时，shell会连到MongoDB服务器的test数据库，并将数据库连接赋值给全局变量db.这个变量是通过shell访问MongoDB的主要入口点

### 命令行

为了方便习惯SQL shell用户，shell包含一些非JavaScript语法的扩展

>help

> show dbs //show all database

> use test // select database

> db.dropDatabase() // remove database

> show collections //show collections of selected database

> db.customer.drop() //delete collection

//read

第一个参数查询条件，第二个参数投影

db.customer.find() // read all documents of one collection of one database

db.customer.find({‘firstname’:’Alice’}) //根据查询条件检索

db.users.find({"usrname": "joe", "age": 27}); //查询条件=条件1 and 条件2

db.users.find({}, {"email": 1, "\_id": 0}); //0表示剔除，1表示想要

查询条件：$lt, $lte, $gt, $gte, $in, $nin, $or,

db.users.find({"age": {"$gte": 18, "$lte", 30}});

db.raffle.find({"$or": [{"ticket\_no": 725}, {"winner": true}]});

正则表达式查询

db.users.find({"name": /joe/i});

数组查询

找到既有appble,又有banana的文档

db.food.find({"fruit": {"$all": ["appble", "banana"]}});

db.food.find({"fruit": {"$size": 3}}) 根据数组长度

db.blog.posts.findOne(criteria, {"comments": {"$slice": -10}}); 返回后10条评论

内嵌文档查询

条件：子文档完全匹配

db.people.find("name": {"first": "Joe", "last": "Schemoe"});

内嵌文档特定键

db.people.find("name.first": "Joe", "name.last": "Schmoe");

// update

update是一种特殊的更新，若没有找到符合更新条件的文档，就会以这个条件和更新文档为基础创建一个新文档。如果找到了匹配的文档，则正常更新。说明同一套代码既可以用于创建文档又可以更新文档

常规程序：

blog = db.analytics.findOne({url: "/blog"});

if (blog) {

blog.pageviews ++;

db.analytics.save(blog);

} else {

db.analytics.save({url: "/blog", pageviews: 1});

}

Mongo update

db.analytics.update({"url": "/blog"},

{"$inc": {"pageviews": 1}}, true);

默认情况下，更新只能对符合匹配条件的第一个文档执行操作。要是有多个文档符合条件，只有第一个文档会被更新，其他文档不会发生变化。要更新所有匹配的文档，可以将update的第4个参数设置为true

文档替换： 第二个参数作为文档替换第一个参数查询到的文档

>db.customer.update({‘firstname’:’Alice’}, {‘firstname’:’QiZhong’, ‘lastname’:’Lin’})

模式改变

{

“\_id”: …,

“name”: “joe”,

“friends”: 32,

“enemies”: 2

}

改成

{

“\_id”: …,

“username”: “joe”,

“relationships”: {

“friends”: 32,

“enemies”: 2

}

}

命令如下：

var joe = db.users.findOne({"name": "joe"});

joe.relationships = {"friends": joe.friends, "enemies": joe.enemies};

joe.username = joe.name;

delete joe.friends;

delete joe.enemies;

delete joe.name;

db.user.update({"name": "joe"}, joe);

部分更新：第一个参数是查询文档，第二个参数是修改器

若字段不存在，则创建它

增加值

db.analytics.update({"url": "www.example.com"},

{"$inc": {"pageviews": 1}});

修改某字段值

db.users.update({"\_id": ...},

{"$set": {"favorite book": "War and Peace"}});

删除某字段

db.users.update({"name": "joe"},

{"$unset":{"favorite book": 1}});

数组修改器

db.blog.posts.update({"title": "A blog post"},

{"$push", {"comments": {

"name": "joe",

"email": "joe@example.com",

"content": "nice post."

}}});

加入到集合中

db.users.update({"\_id", ...},

{"$addToSet": {"emails", "joe@gmail.com"}});

数组元素访问

db.blog.update({"post": post\_id},

{"$inc": {"comments.0.votes": 1}});

>db.customer.remove({‘lastname’:’Lin’}) //delete

### 基本数据类型

Null, true/false, 数值（默认使用64位浮点型，对于整数用NumberInt()/NumberLong()）, 字符串，日期(new Date()), 正则表达式(/foobar/i), 数组([“a”, “b”]), 内嵌文档({“foo”: “bar”}, 对象id(ObjectId()), 二进制数据，代码(Javascript代码function(){…})

### 使用shell执行脚本

$mongo ip:port/db script1.js script2.js script3.js

>load(“script1.js”)

注意：在脚本中，可以访问db变量，以及其他全局变量，但不能使用shell语法糖（如use db; show collections;等），只能使用对应的shell辅助函数

db.getSisterDB(“foo”) use foo

db.getMongo().getDBs() show dbs

db.getCollectionNames() show collections

## 索引

数据库索引与书籍索引类似

不使用索引的查询称为全表扫描

Explain()函数查看mongodb在执行查询相关的的信息

db.users.find({"username": "user01"}).explain()

### 创建索引，基于字段或多个字段

由于机器性能和集合大小的不同，创建索引有可能需要花几分钟时间。

使用索引的查询可以瞬间完成，然而使用索引是有代价的：对于添加的每一个索引，每次写操作（插入，更新，删除）都将耗费更多的间。这是因为，当数据发生变动时，mongodb不仅要更新文档，还要更新集合上的所有索引。通常在一个特定的集合上，不应该拥有两个以上的索引。

db.users.ensureIndex({"username": 1});

如果有一个拥有N个键的索引，那么你同时免费得到了所有这N个键的前缀组成的索引

通常，一个字段的基数越高（字段拥有不同值的数量），这个键上的索引就越有用，比如id

所有的数据库索引信息都存储在system.indexes集合中。这是一个保留集合，不能在其中插入或者删除文档。只能通过ensureIndex or dropIndexes对其进行操作

查看给定集合上的所有索引信息

db.collectionName.getIndexes()

默认情况下，mongodb会尽可能快地创建索引，阻塞所有对数据库的读请求和写请求，一直到索引创建完成

### 固定集合

Mongodb中的普通集合是动态创建的，而且可以自动增长以容纳更多的数据

固定集合相当于循环队列，如果已经没有空间了，最老的文档会被删除以释放空间，新插入的文档会占据这块空间。

db.createCollection("my\_collection", {"capped": true, "size": 10000})

### TTL索引

time-to-live index具有生命周期的索引，这种索引允许为每一个文档设置一个超时时间。一个文档到达预设置的老化程序之后就会被删除。这种类型的索引对于缓存问题（比如会话的保存）非常有用

db.foo.ensureIndex({"lastUpdated": 1}, {"expiredAfterSecs": 60\*60\*24});

mongodb每分钟对TTL索引进行一次清理，所以不应该依赖以秒为单位的时间保证索引的存活状态

### 全文本索引

应该在离线状态下创建全文本索引

全文本索引也会导致比“普通”索引更严重的性能问题，因为所有字符都需要被分解、分词、并且保存到一些地方

### 地理空间索引

2dsphere索引：用于地球表面类型的地图

2d索引：用于平面地图和时间连续的数据

2dsphere允许使用GeoJSON格式指定点、线和多边形。占可以用形如[longitude, latitude]经度纬度表示

查询类型：交集intersection,包含within,接近nearness

### GridFS存储文件

存储大型二进制文件

使用GridFSg来代替独立的文件存储工具

GridFS会自动平衡已有的复制或者为mongodb设置的自动分片

通常来说，如果你有一 些不常改变但是经常需要连续访问的大文件，那么使用GridFS再合适不过了

可以用mongofiles工具在GridFS中上传文件、下载文件、查看文件列表、搜索文件、以及删除文件。

mongofiles --help

mongofiles put foo.txt 上传

mongofiles list 列表

mongofiles get foo.txt 下载

mongofiles search

mongofiles delete

GridFS背后的理念：可以将大文件分割为多个比较大的块，将每个块作为独立的文档进行存储。除了将文件的每一个块单独存储之外，还有一个文档用于将这些块组织在一起并存储该文件的元信息

GridFS中的块会被存储到专用的集合中。块默认使用的集合是fs.chunks，不过可以修改为其他集合。在块集合内部，各个文档的结构如下：

{

"\_id": ..., 块id

"n": 0, 块在文件中的位置

"data": BinData("..."), 块所包含的二进制数据

"files\_id": ... 所属文件

}

每个文件的元信息被保存在一个单独的集合中，默认情况下这个集合是fs.files.这个文件集合中的每一个文档表示GridFS中的一个文件，文档中可以包含与这个文件相关的任意用户自定义信息。文件元信息结构如下：

{

"\_id": ...,

"length": ,

"chunkSize": ...,

"uploadDate": ...

"md5": ...

}

## 聚合

数据提取后，一般需要进行数据分析，可以用聚合工具

### 聚合框架

对集合中的文档进行变换和组合

基本上，可以用多个构件创建一个管道pipeline,用于对一连串的文档进行处理。这些构件包括筛选filtering, 投射projecting,分组grouping,排序sorting,限制limiting,和跳过skipping

找出发表文章最多的那个作者

db.articles.aggregate(

{"$project": {"author": 1}}, 将每篇文章文档中的作者投射

{"$group": {"\_id": "$author", "count": {"$sum": 1}}}, 将作者按名字group,统计次数

{"$sort": {"count": -1}}, 降序排列

{"$limit": 5} 限制返回结果

)

数学运算： $add, $subtract, $multiply, $divide, $mod

db.employees.aggregate(

{

"$project": {

"totalPay": {

"$subtract": [{"$add": ["$salary", "$bonus"]}, "$401k"]

}

}

}

);

日期表达式：$year, $month, $week, $dayOfMonth, $dayOfWeek, $dayOfYear, $hour, $minute, $second

字符串表达式：$substr, $concat, $toLower, $toUpper

逻辑表达式：$cmp, $strcasecmp, $eq, $ne, $gt, $gte, $lt, $lte, $and, $or, $not, $ifNull

分组：$group, $sum, $avg, $max, $min, $first, $last,

数组操作符：$addToSet, $push

大部分操作符的工作方式都是流式的，只要有新文档进入，就可以对新文档进行处理，但是$group and $sort必须要等收到所有的文档之后，才能对文档进行分组，然后才能将各个分组发送给管道中的下一个操作符

### MapReduce

使用JavaScript作为查询语言，因此它能够表达任意复杂的逻辑。然而，这种强大是有代价的：MapReduce非常慢，不应该用在实时的数据分析中

MapReduce能够在多台服务器之间并行执行。它会将一个大问题拆分为多个小问题，将各个小问题发送到不同的机器上，每台机器只负责完成一部分工作。

map -> shuffle -> reduce

计算集合中所有文档所有键出现的次数

map = function() {

for (var key in this) { //this指当前文档

emit(key, {count: 1})

}

};

reduce = function(key, emits) {

total = 0;

for (var i in emits) {

total += emits[i].count;

}

return {"count": total};

}

mr = db.runCommand({

"mapreduce": "foo",

"map": map,

"reduce": reduce);

网页分类，找出哪个主题最为热门

map = function() {

for (var i in this.tags) {

var recency = 1/(new Date() - this.date);

var score = recency \* this.score;

emit(this.tags[i], {"urls": [this.url], "score": score});

}

};

reduce = function (key, emits) {

var total = {urls: [], score: 0}

for (var i in emits) {

emits[i].urls.forEach(function(url) {

total.urls.push(url);

});

total.score += emits[i].score;

}

};

### 聚合命令

db.foo.count(); 返回集合中的文档数量

db.runCommand({"distinct": "people", "key": "age"}); 给定键的所有值 （去重）

## 复制

创建副本集

副本集是一组服务器，其中有一个主服务器(primary),用于处理客户端请求；还有多个备份服务器(secondary)，用于保存主服务器的数据副本。如果主服务器崩溃了，备份服务器会自动将其中一个成员升级为新的主服务器

关键点：

客户端在单台服务器上可以执行的请求，都可以发送到主节点执行（读，写，执行命令，创建索引等）

客户端不能在备份节点上执行写操作

默认情况下，客户端不能从备份节点中读取数据。在备份节点上显式地执行setSlaveOk之后，客户端就可以从备份节点中读取数据了

### 单台服务器副本集

第一个mongo shell

前提条件：创建数据目录/data/db，同时用root帐户启动shell

mongo --nodb 启动mongo shell, 但不连接到任何mongod

replicaSet = new ReplSetTest({"nodes": 3}) 创建三个服务器的副本集

replicaSet.startSet() 启动副本集

replicaSet.initiate() 配置复制功能

现在有3个mongod进程，分别运行在20000、20001、20002端口

第二个mongo shell，客户端1

conn1 = new Mongo("localhost:21000")

testReplSet:PRIMARY> primaryDB = conn1.getDB("test")

>primaryDB.isMaster() 查看副本集的状态

conn1 = new Mongo("qzlin-virtualBox: 20000");

primaryDB = conn1.getDB("test");

//insert

for (i = 0 ; i < 1000; ++ i) {

primaryDB.coll.insert({count: i})

}

primaryDB.coll.count()

备份节点可能会落后于主节点，可能没有最新写入的数据，所以备份节点在默认情况下会拒绝读取请求，以防止应用程序意外拿到过期的数据

这是为了保护应用程序，以免意外连接到备份节点，读取到过期数据。如果希望从备份节点读取数据，需要设置“从备份节点读取数据没有问题”标识。

conn2 = new Mongo("localhost:20001");

conn2.setSlaveOk(); 设置“从备份节点读取数据没有问题”标识

secondaryDB = conn2.getDB("test");

secondaryDB.coll.find();

自动故障转移automatic failover, 如果主节点挂了，其中一个备份节点会自动选举为主节点

primaryDB.adminCommand({"shutdown": 1}) 关闭某一节点

secondaryDB.isMaster() 查看主节点

replicaSet.stopSet() 关闭副本集

### 多台服务器副本集

前提条件：

多台机器互通, $ping server1, $ping server2 （若是多虚拟机，见linux.docx）

创建空的mongod.conf and mongod.log文件

启动Server1上的mongod

$mongod --replSet spock -f mongod.conf --fork --logpath "mongod.log"

启动server2上的mongod

$ssh server2

$mongod --replSet spock -f mongod.conf --fork --logpath "mongod.log"

只是这几个服务器mongod无法知道彼此存在，为了让每个mongod知道彼此，需要创建一个配置文件，在配置文件中列出每一个成员，并且将配置文件发送给任何一个服务器，该服务器会负责将配置文件传播给其他成员

创建配置文件

>config = {

"\_id": "spock",

"members": [

{"\_id": 0, "host": "server1:27017"},

{"\_id": 0, "host": "server2:27017"}

]

}

配置副本集，从而关联服务器，server1会解析这个配置对象，然后向其他成员发送消息，提醒它们使用新的配置。所有成员都配置完成之后，它们会自动选出一个主节点，然后就可以同单机一样，读写请求了

>db = (new Mongo("server1:27017")).getDB("test");

>rs.init

修改副本集配置，重新配置副本集时，主节点需要先退化为普通的备份节点，以便接受新的配置，然后会恢复。

rs.add("server3:27017");

rs.remove("server1:27017");

rs.config();

var config = rs.config();

config.members[1].host = "server4:27017";

rs.reconfig(config)

## 分片

几乎所有数据库软件都能进行手动分片，应用需要维护与若干不同数据库服务器的连接，每个连接还是完全独立的。应用程序管理不同服务器上不同数据的存储，还管理在合适的数据库上查询数据的工作。这种方法可以很好地工作，但是非常难以维护，比如向集群添加节点或从集群删除节点都很困难，调整数据分布和负载模式也不轻松。

MongoDB支持自动分片，可以使数据库架构对应用程序不可见，也可以简化系统管理对应用程序而言，好像始终在使用一个单机的MongoDB服务器一样。MongoDB自动处理数据在分片上的分布，容易添加和删除分片

分片与复制的区别？

复制是让多台服务器都拥有同样的数据副本，每一台服务器都是其他服务器的镜像。而每一个分片都有其他分片拥有不同的数据子集

分片的目标之一：创建一个拥有很多机器的集群，整个集群对应用程序来说就像是一台单机服务器。为了对应用程序隐藏数据库架构的细节，在分片之前要先执行mongos进行一次路由过程。这个路由服务器维护着一个“内容列表”，指明了每个分片包含什么数据内容。应用程序只需要连接到路由服务器，就可以像使用单机服务器一样进行正常的请求了。

### 单台服务器集群

$mongo --nodb

创建3个分片(mongod进程）的集群，分别运行在20001,20002, 20003端口，默认情况下，ShardingTest会在20006端口启动mongos

>cluster = new ShardingTest({"shards": 3, "chunksize": 1})

客户端连接路由服务器Mongos

> db = (new Mongo("localhost:20006")).getDB("test")

mongos会自动将请求路由到合适的分片。客户端不需要知道分片的任何信息，比如分片数据和分片地址。只要有分™片存在，就可以向mongos发送请求，它会自动将请求转发到合适的分片上。

for (var i = 0 ; i < 100000; ++ i) {

db.users.insert({"username": "user"+i, "create\_at": new Date()});

}

>db.users.count()

查询集群的状态：分片摘要信息，数据库摘要信息，集合摘要信息

sh同rs一样，提供分片操作的辅助函数

>sh.help()

>sh.status()

数据库分片

>sh.enableSharding("test")

数据库内的集合分片

对集合分片时，要选择一个片键. MongoDB根据这个键拆分数据。只要被索引过的键才能够作为片键

>db.users.ensureIndex({"username": 1}); 键 -> 索引

>sh.shardCollection("test.users", {"username": 1}) 基于键分片

在分片之前，集合实际上是一个单一的数据块。分片依据片键将集合拆分为多个数据块

查看MongoDB的查询过程

>db.users.find({username: "user12345"}).explain() 定向查询

>db.users.find().explain() 分散-聚集查询

>cluster.stop()

多台服务器集群

通常，分片用来：

* 增加可用RAM
* 增加可用磁盘空间
* 减轻单台服务器的负载
* 处理单个mongod无法承受的吞吐量

由于迁移数据、维护元数据、路由等开销，少量分片的系统与未分片的系统相比，通常延迟更大，吞吐量甚至可能会更小。因此，至少应该创建3个或以上的分片

配置服务器

普通的mongod服务器，保存着集群的配置信息：集群中有哪些分片、分片的是哪些集合、以及数据块的分布

配置服务器相当于集群的大脑，保存着集群和分片的元数据

应该首先建立配置服务器，鉴于它所包含数据的极端重要性，必须启用其日志功能，并确保其数据保存在非易失性驱动器上。每个配置服务器都应位于单独的物理机器上，最好是分布在不同地理位置的机器上

Mongos会向所有配置服务器发送写请求，执行一个两步提交类型的操作，以确保服务器拥有相同的数据，所以这些配置服务器是可的

$mongod --configsvr --dbpath /var/lib/mongodb -f mongod.conf

若没有权限,执行$chown -R $USER:$USER /var/lib/mongodb

$ssh server2

$mongod --configsvr --dbpath /var/lib/mongodb -f /var/lib/config/mongod.conf

--configsvr选项指定mongod为新的配置服务器。

配置服务器并不需要太多的空间和资源。配置服务器的1kb空是约等于200MB真实数据，它保存的只是数据的分布表，由于配置服务器不需要太多的资源，因此可将其部署在运行着其他程序的机器上，如应用服务器、分片的mongod服务器、或mongos进程的服务器上

要常对配置服务器做数据备份。应常在执行集群维护操作之前备份配置服务器的数据

mongos进程

配置服务器均处于运行状态后，启动一个mongos进程供应用程序连接

$mongos --configdb spock/server1:27019, server2:27019 -f /var/lib/mongos.conf

默认情况下mongos运行在27017端口。注意，并不需要指定数据目录(mongos自身并不保存数据，它会在启动时从配置服务器加载集群数据）

可启动任意数量的mongos进程。通常的配置是每个应用程序服务器使用一个mongos进程（与应用服务器运行在同一台机器上）

将副本集转换为分片

该副本集会成为第一个分片

>sh.addShard("spock/server1:27017, server2:27017")

Mongos能够自动检测到没有包含在副本集成员表中的成员。

副本集名称spock被用作分片名称

将副本集作为分片添加到集群后，就可以将应用程序设置从连接到副本集改为连接到mongos.添加分片后，mongos会将副本集内的所有数据库注册为分片的数据库，因此所有查询都会被发送到新的分片上。

增加集群容量

可通过增加分片来增加集群容量。为添加一个新的、空的分片，可先创建一个副本集。确保副本集的名字与其他分片不同。副本集完成初始化并拥有一个主节点后，可以Mongos上运行addShard()命令,将副本集作为分片添加到集群中，在参数中指定副本集的名称和主机名作为种子。

如有多个现存的副本集没有作为分片，只要它们没有同名的数据库，就可将它们作为新分片全部添加到集群中

数据分片

除非明确指定规则,否则MongoDB不会自动对数据进行拆分。如有必要，必须明确告知数据库和集合

>db.enableSharding("music")

>sh.shardCollection("music.artists", {"name": 1})

shardCollection()命令会将集合拆分为多个数据块，这是mongodb迁移数据的基本单元。命令成功执行后，mongodb会均衡地将集合数据分散到集群的分片上。

MongoDB如何追踪集群数据

MongoDB将文档分组为块(chunk)，每个块由给定片键特定范围内的文档组成。一个块只存在于一个分片上，所以MongoDB用一个比较小的表就能够维护块跟分片的映射

进行写操作时，块内的文档数量和大小可能会发生改变

当一个块增长到特定大小时，mongodb会自动将其拆分为两个较小的块。

块范围

新分片的集合起初只有一个块，所有文档都位于这个块中。随着块的增长，mongodb会自动将其分在两个块。

块信息保存在config.chunks集合中。

拆分块过程

Mongos会记录在每个块中插入了多少数据，一旦达到某个阈值，就会检查是否需要对块进行拆分

收到客户端发起的写请求时，Mongos会检查当前块的拆分阈值点

如果达到了拆分阈值点，mongos就会向分片发起一个针对该拆分点的拆分请求

分片计算块的拆分点，并将这些信息发回mongos

Mongos选择一个拆分点，然后试图将这些信息通知给配置服务器

均衡器balancer

负责数据的迁移，它会周期性地检查分片间是否存在不均衡，每隔几秒钟，Mongos就会尝试变身为均衡器。如果没有其他可用的均衡器，mongos就会对整个集群加锁，以防止配置服务器对集群进行修改，然后做一次均衡。均衡并不会影响mongos的正常路由操作，所以使用mongos的客户端不会受到影响

Mongos成为均衡器后，就会检查每个集合的分块表，从而查看是否有分片达到了均衡阈值balancing threshold.

使用集群的应用程序无需知道数据迁移：在数据迁移完成之前，所有读写读请求都会被路由到旧的块上。如果元数据更新完成，那么所有试图访问旧位置数据的mongos进程都会得到一个错误，这些错误对客户端不可见，mongos会对这些错误做静默处理，然后在新的分片上重新执行之前的操作

# PostgreSQL

Download and install postgresql-9.5.0-rc1-windows-x64.exe

安装过程会要求设置管理员密码：qzlin （默认管理员:postgre）

数据库默认地址: C:\Program Files\PostgreSQL\9.5\data

Open pgAdmin III

连接服务器： right click PostgreSQL 9.5 (localhost:5432) -> 连接服务器

数据库 -> 创建数据库 //创建数据库

登录角色 -> 创建登录角色 //创建登录角色

Psql:

CREATE ROLE qzlin LOGIN PASSWORD ‘pass@123’ VALID UNTIL ‘infinity’;

then login with qzlin

CREATE DATABASE blog;