**Redis教程**

# NoSQL(Not Only SQL)数据库介绍



全文搜索：Apache Lucene

Solr

ElasticSearch（基于Lucene开发的）

# Redis介绍

Redis的作者，叫SalvatoreSanfilippo，来自意大利的西西里岛，现在居住在卡塔尼亚。目前供职于**Pivotal**公司。他使用的网名是antirez。

2012年08月02日

Redis2.4.16 小更新版本 NoSQL；

2012年08月31日

Redis2.4.17 小更新版本 NoSQL；

2012年11月7日Redis 2.6.3 发布，高性能K/V服务器；

2013年4月30日Redis2.6.13 发布，高性能K/V服务器；

2013年11月25日，Redis2.8.1发布；

2015年2月，Redis3.0.0发布。

## Redis概述

**Redis是一个开放源代码（BSD许可）的内存数据结构存储，可以用作数据库、缓存和消息代理；支持字符串、哈希、列表、集合、带范围查询的排序集合等数据结构。Redis具有内置的复制、Lua脚本、LRU收回、事务和不同级别的磁盘上持久性，并通过Redis Sentinel和Redis集群的自动分区提供高可用性。**

**源码地址：https://github.com/antirez/redis**

**官方网站：https://redis.io**

## Redis的特点、优势

### Redis的特点

**1.内存数据库，速度快；支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。**

**2.Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。（Redis5新增了Stream）**

**3.Redis支持数据的备份，即主从模式的数据备份，数据可扩展性高。**

**4.支持事务。**

### Redis的优势

**性能极高**：Redis读的速度是110000次/s，写的速度是81000次/s。

**丰富的数据类型**：String、List、Hash、Set 及Sorted Set数据类型。 (Stream)

**原子操作**：Redis的所有操作都是原子性的。

**丰富的特性**：Redis可用于缓存，消息队列、排行榜、点赞等。

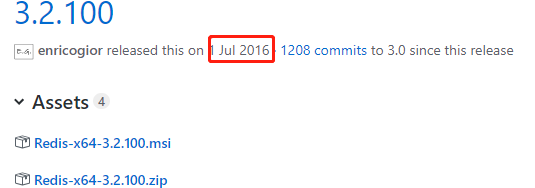
# Centos7安装redis-5.0.4

## 简要介绍

* **安装步骤，我分享的有道云笔记**

<http://note.youdao.com/noteshare?id=60ab3c5071f1acab4d90d32e5ee5f130&sub=A486484D71BB454C8A6F0CBA0B9E2A5C>

* **目前官网的最新稳定版本是5.0.5版本，Redis官网不提供windows的版本。**
* Window64位下载地址：<https://github.com/MicrosoftArchive/redis/releases>。
* 目前最新的windows版本是2016年的3.2.100版本。



## 单机版安装步骤

### 上传文件到服务器

**[root@localhost ~]# tar -zxvf redis-5.0.3.tar.gz -C /opt**

**[root@localhost redis-5.0.3]# cd /opt/redis-5.0.3**

**[root@localhost redis-5.0.3]# make**

**报错：**

**compilation terminated. make[1]: \*\*\* [adlist.o] Error 1 make[1]: Leaving directory `/usr/local/redis-5.0-rc3/src' make: \*\*\* [all] Error 2**

### 安装Development Tools

**[root@localhost redis-5.0.3]# yum groupinstall 'Development Tools'**

### 安装Redis依赖包

**[root@localhost redis-5.0.3]# cd /opt/redis-5.0.3/deps**

**[root@localhost deps]# make hiredis lua jemalloc linenoise**

**安装结果：**

**MAKE linenoise**

**cd linenoise && make**

**make[1]: Entering directory `/opt/redis-5.0.3/deps/linenoise'**

**cc -Wall -Os -g -c linenoise.c**

**make[1]: Leaving directory `/opt/redis-5.0.3/deps/linenoise'**

### 继续安装Redis：

**[root@localhost deps]# cd ../**

**[root@localhost redis-5.0.3]# make**

**安装结果**

**INSTALL redis-check-rdb**

**INSTALL redis-check-aof**

**Hint: It's a good idea to run 'make test' ;)**

**make[1]: Leaving directory `/opt/redis-5.0.3/src'**

### 测试Redis安装

**[root@localhost redis-5.0.3]# cd src**

**[root@localhost src]# make install**

**测试结果**

**Hint: It's a good idea to run 'make test' ;)**

**INSTALL install**

**INSTALL install**

**INSTALL install**

**INSTALL install**

**INSTALL install**

### 编辑redis.conf

**[root@localhost redis-5.0.3] vim /opt/redis/redis.conf**

**bind 127.0.0.1 # 注释掉**

**protected-mode no # 关闭保护模式**

**daemonize yes # 守护进程模式开启**

**~~dir ./rdb/ # 默认是./，表示在redis-5.0.3目录下生成dump.rdb,用于持久化数据~~；**

### 启动服务

**[root@localhost redis-5.0.3]/opt/redis-5.0.3/src/redis-server /opt/redis-5.0.3/redis.conf**

### 启动客户端

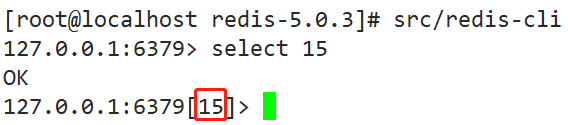
**[root@localhost redis-5.0.3]/opt/redis/src/redis-cli**

### 关闭redis-server

**[root@localhost redis-5.0.3]/opt/redis-5.0.3/src/redis-cli shutdown**

## Redis的多数据库特点

**Redis默认支持16个数据库，以一个从0开始的递增数字命名，可以通过redis.conf文件中的参数databases来修改默认数据库个数。客户端连接Redis服务后会自动选择0号数据库，可以通过SELECT命令更换数据库，例如选择15号数据库：**



**说明：不支持自定义数据库名称。**

**不支持为每个数据库设置访问密码。**

多个数据库之间不是完全隔离的，FLUSHALL命令会清空所有数据库的数据。

多数据库不适用存储不同应用的数据。

**Redis集群不支持多数据库**；

# Redis5.0新增特性

**1.新增stream数据类型。**

2.新的Redis模块API：Timers和Cluster API。

3.RDB现在存储LFU(最近最少使用)和LRU(最近不经长使用)信息。

**4.集群管理器从Ruby移植到了C。**

5.新的sorted set命令：ZPOPMIN/MAX和阻塞变种。

6.升级了主动碎片整理。

7.增强HyperLogLog实现

8.完善内存报告。

9.许多带有子命令的命令有了一个HELP子命令。

10.客户经常连接和断开连接时性能更好。

11.修复了错误和改进。

12.Jemalloc内存分配器升级到了5.1。

# Redis的数据类型

## string类型

String类型是redis中最基本的类型，它是包含了很多种类型的特殊类型，并且是二进制的安全的，可以存储序列化的对象、图片的二进制存储、一个简单的字符串、数值等。一个字符串类型键允许存储的数据最大容量是512MB。

## hash类型

hash是一个String类型的field和value的映射表。hash特别适合用于存储对象。相对于将对象序列化存储为String类型，将一个对象存储在hash类型中会占用更少的内存，且可以方便的操作对象。

## list类型

list是一个链表结构，类似JDK的LinkedList、Queue，主要功能是push、pop、获取一个范围的所有值等等，key可以认为链表的名称。list类型的每个子元素都是string类型的**双向链表**。我们可以通过push、pop操作从链表的头部或者尾部添加删除元素，这样list既可以作为栈，又可以作为队列。

## set类型

set是集合，它是string类型的无序不可重复集合。Set是通过hash table实现的，添加、删除和查找的复杂度都是o(1)。对set我们可以取并集、交集、差集。通过set的这些操作我们可以实现好友推荐功能。

**命令举例：**

**sadd：**添加元素

**smembers**：获取集合中所有元素

**sismember：**判断元素是否在集合中

**srem**：删除元素

**scard**：获取元素个数，相当于count

**spop**：随机返回删除的元素

**sdiff**：差集，返回在第一个set里面而不在后面任何一个set里面的项(谁在前以谁为标准)

**sdiffstore**：差集并保留结果

sdiffstore destination key [key ...]

**sinter**：交集，返回多个set里面都有的项

**sinterstore**：交集并保留结果

**sunion**：并集

**sunionstore**：并集并保留结果

**smove**：移动元素到另一个集合

## zset类型

zset有序集合，类似SortedSet.

**zadd**：添加元素；格式：**zadd key score value [score value]**

**zrange：**获取索引区间内的元素；**格式：zrange key start stop [withscores]**

**zrangebyscore ：**获取分数区间内的元素；

**格式：zrangebyscore key minscore maxscore [withscore] [limit offset count]，**默认是包含端点值的，如果加上"("表示不包含，后面还可以加上limit来限制。

**zrem ：**删除元素，**格式是：zrem key member**，member值可以是多个

## stream类型-Reids5.0新增类型

Redis5.0大版本新增类型

本质是抽象日志

# Redis集群

## Redis主从模式

### Redis主从复制模式

Master可以有多个Slave。

Slave也可以连接其它的Slave。

Slave同步Master数据时，Master不会阻塞，可以继续处理client的读写请求。（乐观复制策略）

Mater主库可以进行读写操作，Slave从库一般只是进行读操作。

### 主从复制配置

1主1从、1主n从、1主1从1从....

### 主从复制基本原理

* Slave启动时，向Master发送sync命令，2.8版本发送psync，以实现增量复制。
* Mater(主库)接到sync请求后，会在后台保存快照，也就是实现RDB持久化，并将保存快照期间接收到命令缓存起来。
* 快照完成后，Master(主库)会将快照文件和所有的缓存的命令发送给Slave(从库)。
* Slave(从库)接收后，会载入快照文件并执行缓存的命令，从而完成复制的初始化。
* 在数据库使用阶段，Master(主库)会自动把每次收到的写命令同步到从服务器。

### Redis乐观复制策略 （弱一致性）

Redis采用乐观复制的策略，允许一定时间内主从数据库的内容不同，当然最终的数据会相同。此策略保证了Redis性能，在进行复制时，Master(主库)并不阻塞，继续处理client的请求。

但是Redis同样提供了配置用来限制**只有当数据库至少同步给指定数量的Slave(从库)时**，Master(主库)才可写，否则返回错误。配置是：min-slaves-to-write、min-slaves-max-lag。

### 主从配置，配置redis.conf

#### Master配置

|  |
| --- |
| **bind 127.0.0.1 # 注释掉**  **protected-mode no # 关闭保护模式(将yes改为no)**  **daemonize yes # 守护进程模式开启（将no改为yes）**  **dir /opt/redis/data/ # 默认是./，表示在redis目录下生成dump.rdb,用于持久化数据；**  **logfile "/opt/redis/logs/redis.log"** |

#### Slave配置

|  |
| --- |
| **bind 127.0.0.1 # 注释掉**  **protected-mode no # 关闭保护模式**  **daemonize yes # 守护进程模式开启**  **dir /opt/redis/data/默认是./，表示在redis目录下生成dump.rdb,用于持久化数据；**  **logfile “/opt/redis/logs/redis.log”**  **replicaof master-ip master-port** |

## Redis哨兵模式

* 前面主从集群有个问题，就是Master(主库)挂了之后，无法重新选举新的节点作为主节点进行写操作，导致服务不可用。
* Redis提供了哨兵工具来实现监控Redis系统的运行情况，能够实现如下功能：
* 监控主从数据库运行是否正常。
* 当主数据库出现故障时，自动将从数据库转换为主数据库。
* 使用Redis-sentinel,redis实例必须在非集群模式下运行。

配置哨兵模式 - 配置sentinel.conf文件

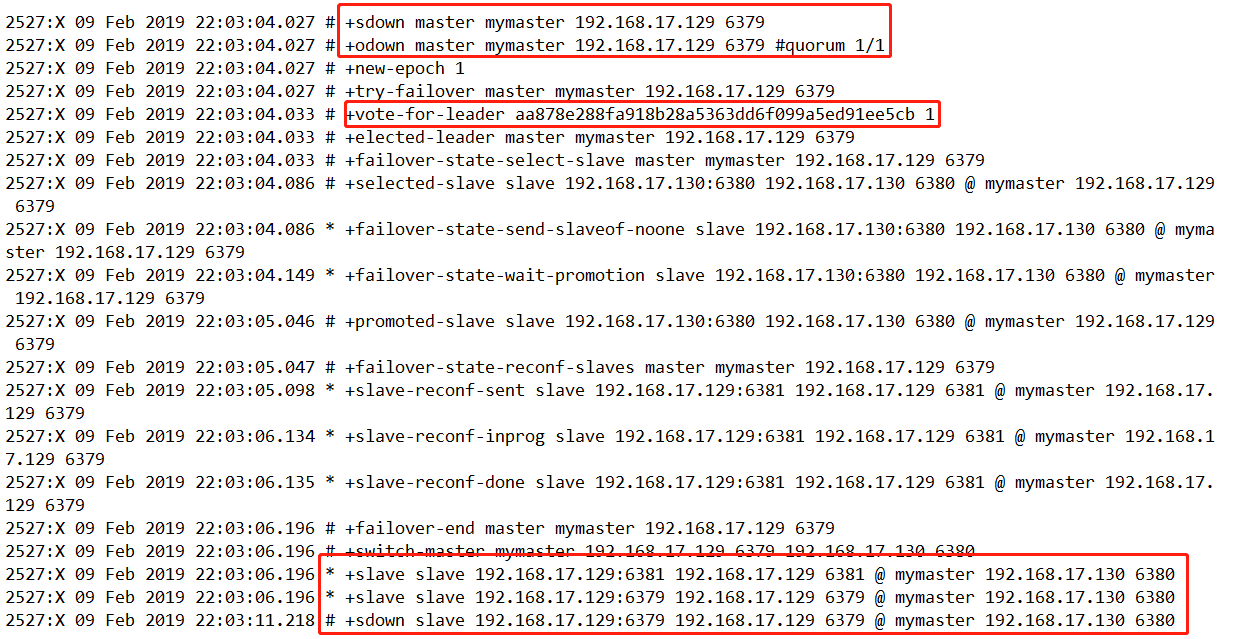
|  |
| --- |
| **daemonize yes # 改为yes**  **logfile "/opt/redis-shaobing/logs/sentinel.log"**  **sentinel monitor mymaster 192.168.17.129 6379 1**  **sentinel down-after-milliseconds mymaster 5000 # 改为5s，默认30s** |

**启动哨兵**：/opt/redis-shaobing/src/redis-sentinel /opt/redis-shaobing/sentinel.conf

**进入哨兵**：/opt/redis-shaobing/src/redis-cli -h 192.168.17.129 -p 26379

**查看哨兵信息**：/opt/redis-shaobing/src/redis-cli -h 192.168.17.129 -p 26379 info Sentinel

**停掉Master，查看哨兵日志，从库重启后不能再次成为Master。**

****

## Redis集群

### 集群介绍

Redis 集群是一个提供在多个Redis节点间共享数据的程序集。

Redis集群并不支持处理多个keys的命令(比如mset、mget等)，因为这需要在不同的节点间移动数据,从而达不到像Redis那样的性能,在高负载的情况下可能会导致不可预料的错误.

Redis 集群通过分区来提供一定程度的可用性，在实际环境中当某个节点宕机或者不可达的情况下继续处理命令。

### Redis 集群的优势

自动分割数据到不同的节点上。

整个集群的部分节点失败或者不可达的情况下能够继续处理命令。

### Redis 集群的数据分片 \*\*\*\*\*\*

Redis集群引入了**哈希槽**的概念。

Redis 集群有16384个哈希槽，每个key通过CRC16校验后对16384取模来决定放置哪个槽。集群的每个节点负责一部分hash槽。

举个例子：比如当前集群有3个节点,那么：节点 A 包含 0 到 5500号哈希槽。节点 B 包含5501 到 11000 号哈希槽。节点 C 包含11001 到 16383号哈希槽。

**这种结构很容易添加或者删除节点**： 比如我们想新添加个节点D, 我们需要将节点 A、B、 C中的部分槽移动到D上。如果我想移除节点A，需要将A中的槽移到B和C节点上。然后将没有任何槽的A节点从集群中移除即可。由于从一个节点将哈希槽移动到另一个节点并不会停止服务,所以无论添加删除或者改变某个节点的哈希槽的数量都不会造成集群不可用的状态.

### Redis集群安装

**安装方式**：在一台机器上搭建集群

**安装节点数**：必须有3个或3个以上的主节点，所以采用3主3从方式安装；

|  |
| --- |
| 少于3个主节点，创建集群时会报错  \*\*\* ERROR: Invalid configuration for cluster creation.  \*\*\* Redis Cluster requires at least 3 master nodes.  \*\*\* This is not possible with 4 nodes and 1 replicas per node.  \*\*\* At least 6 nodes are required. |

#### 创建目录

* **在/opt下创建redis-cluster目录；**
* **在redis-cluster创建8000至8005六个目录；**
* **将redis.conf文件移动到8000-8005文件夹下；**

#### 修改redis.conf文件

每个redis.conf文件修改配置点：\*表示要替换的字符：如8000、8001、8002、8003、8004、8005

|  |
| --- |
| **protected-mode no**  **daemonize yes**  **# 1.绑定ip(必须要写) 如：bind 192.168.179.131**  bind 机器的ip  **# 2.端口号，依次修改为8000、8001、8002、8003、8004、8005**  port 800\*  **# 3.由于是在同一台机器上，所以pid文件也要修改**  pidfile "/var/run/redis\_800\*.pid"  **# 4. 日志文件**  logfile "/opt/redis-cluster/800\*/800\*.log"  **# 5.持久化文件等的存储位置**  dir "/opt/redis-cluster/800\*"  **# 6.开启AOF及持久化**  appendonly yes  **# 7.启动集群模式**  cluster-enabled yes  **# 8这个文件保存集群信息，文件会自动生成到上面dir配置的目录下**  cluster-config-file nodes-800\*.conf |

#### 按次序启动每一个节点

|  |
| --- |
| /opt/redis/src/redis-server /opt/redis-cluster/8000/redis.conf  /opt/redis/src/redis-server /opt/redis-cluster/8001/redis.conf  /opt/redis/src/redis-server /opt/redis-cluster/8002/redis.conf  /opt/redis/src/redis-server /opt/redis-cluster/8003/redis.conf  /opt/redis/src/redis-server /opt/redis-cluster/8004/redis.conf  /opt/redis/src/redis-server /opt/redis-cluster/8005/redis.conf |

#### 创建集群

Redis5.0将Ruby创建集群的方式改为了C语言创建，创建命令也进行了修改；

**执行下面的一条命令：**

/opt/redis-5.0.3/src/redis-cli --cluster create 192.168.179.128:8000 1192.168.179.128:8001 192.168.179.128:8002 192.168.179.128:8003 192.168.179.128:8004 192.168.179.128:8005 --cluster-replicas 1

**输入命令后，redis会自动划分hash槽，看下面日志，在划分完成后，输入yes即可**

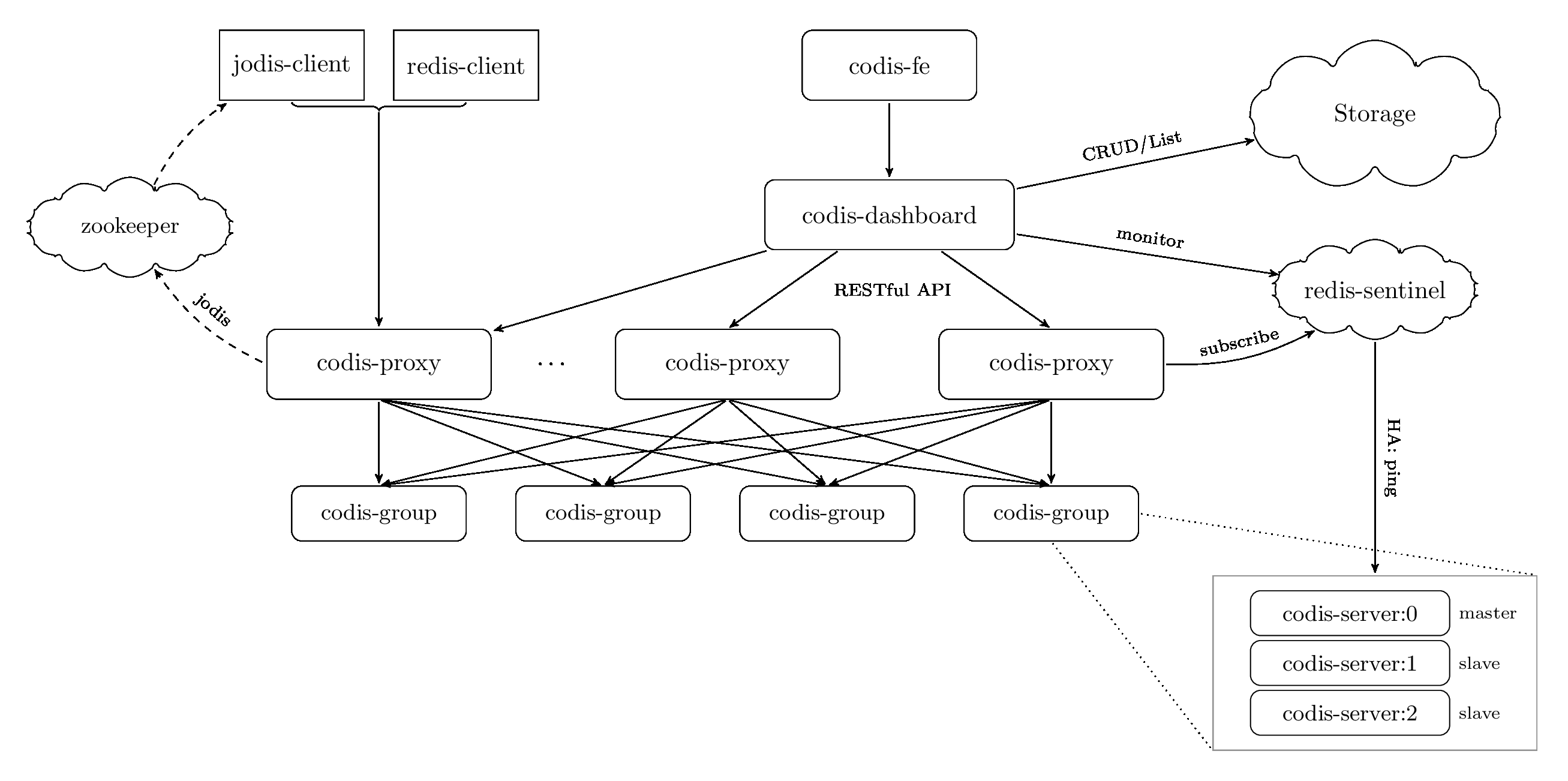
|  |
| --- |
| >>> Performing hash slots allocation on 6 nodes...  Master[0] -> Slots 0 - 5460  Master[1] -> Slots 5461 - 10922  Master[2] -> Slots 10923 - 16383  Adding replica 192.168.17.130:7002 to 192.168.17.130:7000  Adding replica 192.168.17.130:7004 to 192.168.17.130:7001  Adding replica 192.168.17.130:7005 to 192.168.17.130:7003  >>> Trying to optimize slaves allocation for anti-affinity  [WARNING] Some slaves are in the same host as their master  M: 26e59e9eff724a6d9f8e512456ed572d8c7af80b 192.168.17.130:7000  slots:[0-5460] (5461 slots) master  M: 5f4430c20feb5f6753eb3a385bced0ddfd335f23 192.168.17.130:7001  slots:[5461-10922] (5462 slots) master  M: f4f4b14afe313521e8c60bbf6a708ba7225d2473 192.168.17.130:7003  slots:[10923-16383] (5461 slots) master  S: b44ad740571ffd93f29ec383182259069884e3ca 192.168.17.130:7002  replicates f4f4b14afe313521e8c60bbf6a708ba7225d2473  S: f161cdbef2227122883892da372c42360ff093a8 192.168.17.130:7004  replicates 26e59e9eff724a6d9f8e512456ed572d8c7af80b  S: 7a83dd985cfa2cec04aa210d0efd8372cbda2bc3 192.168.17.130:7005  replicates 5f4430c20feb5f6753eb3a385bced0ddfd335f23  **Can I set the above configuration? (type 'yes' to accept): yes**  >>> Nodes configuration updated  >>> Assign a different config epoch to each node  >>> Sending CLUSTER MEET messages to join the cluster  Waiting for the cluster to join  ....  >>> Performing Cluster Check (using node 192.168.17.130:7000)  M: 26e59e9eff724a6d9f8e512456ed572d8c7af80b 192.168.17.130:7000  slots:[0-5460] (5461 slots) master  1 additional replica(s)  M: f4f4b14afe313521e8c60bbf6a708ba7225d2473 192.168.17.130:7003  slots:[10923-16383] (5461 slots) master  1 additional replica(s)  S: 7a83dd985cfa2cec04aa210d0efd8372cbda2bc3 192.168.17.130:7005  slots: (0 slots) slave  replicates 5f4430c20feb5f6753eb3a385bced0ddfd335f23  S: b44ad740571ffd93f29ec383182259069884e3ca 192.168.17.130:7002  slots: (0 slots) slave  replicates f4f4b14afe313521e8c60bbf6a708ba7225d2473  S: f161cdbef2227122883892da372c42360ff093a8 192.168.17.130:7004  slots: (0 slots) slave  replicates 26e59e9eff724a6d9f8e512456ed572d8c7af80b  M: 5f4430c20feb5f6753eb3a385bced0ddfd335f23 192.168.17.130:7001  slots:[5461-10922] (5462 slots) master  1 additional replica(s)  [OK] All nodes agree about slots configuration.  >>> Check for open slots...  >>> Check slots coverage...  **[OK] All 16384 slots covered.** |

## 其他版本Redis集群

豌豆荚的codis：**https://github.com/CodisLabs/codis**

Codis is a proxy based high performance Redis cluster solution written in Go. It is production-ready and widely used at **wandoujia.com** and many companies. You can see Codis Releases for latest and most stable realeases.

**Codis架构图：**



**其它版本的redis集群大家自己了解一下即可；**

# Redis的事务

## Redis事务概述

**Redis对事务的支持比较简单，它是一组命令的集合，命令被依次顺序的执行，也可以放弃事务的执行，此时所有事务里面的命令都不会执行。**

**Redis 只能保证一个client发起的事务中的命令可以连续的执行，而中间不会插入其他client的命令，因为Redis是单线程架构，所以在执行完事务内所有指令前是不可能再去同时执行其他客户端的请求的。**

**Redis的事务没有隔离级别的概念，因为事务提交前任何指令都不会被实际执行，也就不存在“事务内的查询要看到事务里的更新，在事务外查询不能看到”这种问题了。**

**Redis的事务不保证原子性，也就是不保证所有指令同时成功或同时失败，只有决定是否开始执行全部指令的能力，没有执行到一半进行回滚的能力。**

## 事务操作的基本命令

**multi** : 设置事务开始

**exec** : 执行事务

**discard** : 放弃事务

**watch** : 监控键值，如果键值被修改或删除，后面的一个事务就不会执行

**unwatch** : 取消watch

**watch命令说明：**

**Redis使用watch来提供乐观锁；**

**当exec被调用后，所有的之前被监视的键值会被取消监视，不管事务是否被取消或执行。并且当客户端连接丢失的时候，所有东西都会被取消监视。**

## 事务扩展-乐观锁

### 乐观锁介绍

**乐观锁大多是基于数据版本(version)的记录机制实现的。何谓数据版本？即为数据增加一个版本标识，更新数据时，对此版本号加 1，当线程A要更新数据值时，在读取数据的同时也会读取version值，在提交更新时，若刚才读取到的version值比当前数据库中的version值大于时才更新，否则更新失败。**

### 乐观锁举例

小明的账户有余额1000刀；

(1)操作员A将小明的信息读出（此时 version=1），并准备从其帐户余额中扣除100刀（1000-100）； 900

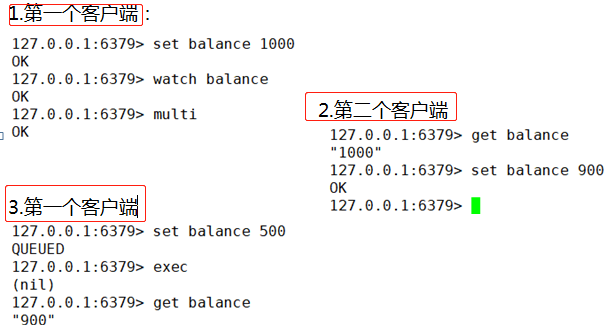
(2)在操作员A操作的过程中，操作员B也读入小明的信息（此时 version=1），并准备从其帐户余额中扣除500刀（1000-500）； 500

(3)A完成了修改工作后，将数据版本号加 1（此时 version=2），帐户扣除后余额为900刀，提交至数据库更新，此时由于提交数据版本大于数据库记录当前版本(2>1)，数据被更新，数据库记录 version 更新为 2;

(4)在A完成更新操作之后，B也完成了操作，她也将版本号加 1（version=2）并试图向数据库提交数据(此时小明的余额为500刀)，但比对数据库记录版本时发现，B提交的数据版本号为 2，数据库记录当前版本也为 2，不满足“提交版本必须大于记录当前版本才能执行更新”的乐观锁策略，因此，B的提交被驳回；

### Redis的乐观锁测试

开启两个客户端



## Redis事务的基本过程

1.发送一个事务的命令multi给redis；

2.依次把要执行的命令发送给redis，redis接到这些命令，并不会立即执行，而是放到等待执行的事务队列里面；

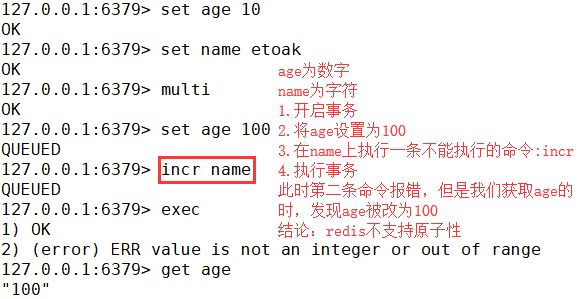
3.发送执行事务的命令exec给redis；

4.redis会保证一个事务内的命令依次执行，而不会被其它命令插入；

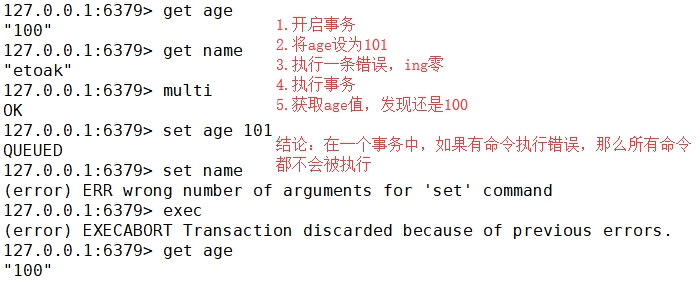
## Redis事务过程中的错误处理

**redis 的事务非常简单，当然会存在一些问题。只能保证事务的每个命令连续执行，但是如果事务中的一个命令失败了，并不回滚其他命令，下面举例看错误处理；**

**1.如果是某个命令执行错误(使用方式错了)，那么其它的命令仍然会正常执行，然后在执行后返回错误信息；**



**2.如果任何一个命令语法有错，redis会直接返回错误，所有的命令都不会执行；**



**3.redis不提供事务滚回的功能，开发者必须在事务执行出错后，自行恢复数据库状态；**

# Redis持久化

## 持久化方案

Redis的持久化方案目前有两种方式：RDB和AOF模式。

### RDB持久化

**实现机制 rdb方式是通过在是一定时间内将内存中的数据集快照（Snapshot）写入磁盘；在数据恢复时，再将快照中的信息读取到内存中。**

**RDB持久化是Redis定期将内存中的"数据集快照"写入磁盘，实际操作过程是创建(fork)一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。整个过程主进程不进行任何IO操作，保证Redis性能的高效性。**

**RDB持久化的触发方式**：自动触发和手动触发

* 自动触发主要看redis.conf

save 900 1 900秒内至少有一个key被改变就自动触发RDB持久化

save 300 10 300秒内至少有10个key被改变就自动触发RDB持久化

save 60 10000 60秒内至少有10000个key被改变就自动触发RDB持久化

|  |
| --- |
| 900秒内至少有一个key被改变就自动触发RDB持久化  2477:M 27 May 2019 15:56:51.959 \* 1 changes in 900 seconds. Saving...  2477:M 27 May 2019 15:56:52.037 \* Background saving started by pid 2517  **2517:C** 27 May 2019 15:56:52.093 \* DB saved on disk  **2517:C** 27 May 2019 15:56:52.095 \* RDB: 4 MB of memory used by copy-on-write  2477:M 27 May 2019 15:56:52.138 \* Background saving terminated with success  300秒内至少有10个key被改变就自动触发RDB持久化  2297:M 28 May 2019 11:45:58.081 \* 10 changes in 300 seconds. Saving...  2297:M 28 May 2019 11:45:58.081 \* Background saving started by pid 2307  2307:C 28 May 2019 11:45:58.092 \* DB saved on disk  2307:C 28 May 2019 11:45:58.093 \* RDB: 4 MB of memory used by copy-on-write  2297:M 28 May 2019 11:45:58.183 \* Background saving terminated with success |

* 手动触发

**save命令**：执行此命令会阻塞Redis服务器，执行命令期间，Redis不能处理其它命令，直到RDB过程完成为止。

**bgsave命令**：执行该命令时，Redis会在后台异步进行快照操作，做快照的同时还可以响应客户端请求；此时Redis进程执行fork操作创建子进程，RDB持久化过程由子进程负责，完成后自动结束。阻塞只发生在fork阶段，一般时间很短。

|  |
| --- |
| 127.0.0.1:6379> bgsave  Background saving started  2297:M 28 May 2019 14:44:02.863 \* Background saving started by pid 2447  2447:C 28 May 2019 14:44:02.872 \* DB saved on disk  2447:C 28 May 2019 14:44:02.872 \* RDB: 2 MB of memory used by copy-on-write  2297:M 28 May 2019 14:44:02.948 \* Background saving terminated with success |

### AOF持久化

**AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。**

|  |
| --- |
| # 默认情况下，redis会在后台异步的把数据库镜像备份到磁盘，但是该备份是非常耗时的，而且备份也不能很频繁，如果发生断电等状况，那么将造成比较大范围的数据丢失。(rdb)  # 所以redis提供了另外一种更加高效的数据库备份及灾难恢复方式。  # 开启append only模式之后，redis会把所接收到的每一次写操作请求都追加到appendonly.aof文件中，当redis重新启动时，会从该文件恢复出之前的状态。  # 但是这样会造成appendonly.aof文件过大，所以redis还支持了BGREWRITEAOF指令，对appendonly.aof 进行重新整理。  # 你可以同时开启asynchronous dumps 和 AOF  appendonly yes  # AOF文件名称 (默认: "appendonly.aof")  appendfilename appendonly.aof  # Redis支持三种同步AOF文件的策略:  **# no: 不进行同步. Faster.**  **# always: always表示每次有写操作都进行同步. Slow, Safest.**  **# everysec: 表示对写操作进行累积，每秒同步一次. Compromise.**  # 默认是"everysec"，按照速度和安全折中这是最好的。  # 如果想让Redis能更高效的运行，你也可以设置为“no”，让操作系统决定什么时候去执行或者相反想让数据更安全你也可以设置为“always”，如果不确定就用 "everysec".  # appendfsync always  # appendfsync no  **appendfsync everysec**  # AOF策略设置为always或者everysec时，后台处理进程(后台保存或者AOF日志重写)会执行大量的I/O操作  # 在某些Linux配置中会阻止过长的fsync()请求。注意现在没有任何修复，即使fsync在另外一个线程进行处理  # 为了减缓这个问题，可以设置下面这个参数no-appendfsync-on-rewrite  no-appendfsync-on-rewrite no  # Automatic rewrite of the append only file.  # AOF 自动重写  # 当AOF文件增长到一定大小的时候Redis能够调用 BGREWRITEAOF 对日志文件进行重写  # 它是这样工作的：Redis会记住上次进行些日志后文件的大小(如果从开机以来还没进行过重写，那日志大小在开机的时候确定)  # 基础大小会同现在的大小进行比较。如果现在的大小比基础大小大制定的百分比，重写功能将启动  # 同时需要指定一个最小大小用于AOF重写，这个用于阻止即使文件很小但是增长幅度很大也去重写AOF文件的情况  # 设置 percentage 为0就关闭这个特性  auto-aof-rewrite-percentage 100  auto-aof-rewrite-min-size 64mb |

# Redis发布与订阅

## 介绍

Redis的发布订阅模式可以实现进程间的消息传递

## 发布订阅模式的操作指令

publish：发布消息，格式是： publish channel 消息

subscribe：订阅消息，格式是：subscribe channel, 可以是多个channel

psubscribe：订阅消息，格式是：psubscribe channel, 支持glob风格的通配符

unsubscribe：取消订阅，格式是：unsubscribe channel,不指定频道表示取消所有subscribe命令的订阅

punsubscribe：取消订阅，格式是：punsubscribe channel,不指定频道表示取消所有的punsubscribe命令的订阅，注意这里匹配模式的时候，是不会将通配符展开的，是严格进行字符串匹配的，比如：punsubscribe \* 是无法退订 c1.\*的，必须严格使用punsubscribe c1.\* 才可以。

# Redis的java客户端-Jedis

源码地址：<https://github.com/xetorthio/jedis>

官方最新版

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>3.0.1</version>

</dependency>

## Redis使用jedis原生api



## Spring整合Redis

## spring-data-redis