

Redis

版本：

redis 3.2.11

第一节 Redis简介

1.1 NoSQL

NoSQL，泛指非关系型的数据库，NoSQL即Not-Only SQL，它可以作为关系型数据库的良好补充。随着互联网web2.0网站的兴起，非关系型的数据库现在成了一个极其热门的新领域，非关系数据库产品的发展非常迅速

而传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，例如：

1、High performance - 对数据库高并发读写的需求

web2.0网站要根据用户个性化信息来实时生成动态页面和提供动态信息，所以基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强顶得住，但是应付上万次SQL写数据请求，硬盘IO就已经无法承受了。其实对于普通的BBS网站，往往也存在对高并发写请求的需求，例如网站的实时统计在线用户状态，记录热门帖子的点击次数，投票计数等，因此这是一个相当普遍的需求。

2、Huge Storage - 对海量数据的高效率存储和访问的需求

类似Facebook, twitter, Friendfeed这样的SNS网站，每天用户产生海量的用户动态，以Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下乃至不可忍受的。再例如大型web网站的用户登录系统，例如腾讯，盛大，动辄数以亿计的帐号，关系数据库也很难应付。

3、High Scalability & High Availability- 对数据库的高可扩展性和高可用性的需求

在基于web的架构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，你的数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移，为什么数据库不能通过不断的添加服务器节点来实现扩展呢？

NoSQL数据库的产生就是为了解决大规模数据集多重数据种类带来的挑战，尤其是大数据应用难题

一些主流的NoSQL产品：



1.2 NoSQL的类别

1.2.1 键值(Key-Value)存储数据库

相关产品：Tokyo Cabinet/Tyrant、Redis、Voldemort、Berkeley DB

典型应用：内容缓存，主要用于处理大量数据的高访问负载。

数据模型：一系列键值对

优势：快速查询

劣势：存储的数据缺少结构化

1.2.2 列存储数据库

相关产品：Cassandra, HBase, Riak

典型应用：分布式的文件系统

数据模型：以列簇式存储，将同一列数据存在一起

优势：查找速度快，可扩展性强，更容易进行分布式扩展

劣势：功能相对局限

1.2.3 文档型数据库

相关产品：CouchDB、MongoDB

典型应用：Web应用（与Key-Value类似，Value是结构化的）

数据模型：一系列键值对

优势：数据结构要求不严格

劣势：查询性能不高，而且缺乏统一的查询语法

1.2.4 图形(Graph)数据库

相关数据库：Neo4J、InfoGrid、Infinite Graph

典型应用：社交网络

数据模型：图结构

优势：利用图结构相关算法。

劣势：需要对整个图做计算才能得出结果，不容易做分布式的集群方案。

1.3 Redis是什么

2008年，意大利的一家创业公司Merzia推出了一款基于MySQL的网站实时统计系统LLOOGG，然而没过多久该公司的创始人 Salvatore Sanfilippo便对MySQL的性能感到失望，于是他决定亲自为LLOOGG量身定做一个数据库，并于2009年开发完成，这个数据库就是Redis。不过Salvatore Sanfilippo并不满足只将Redis用于LLOOGG这一款产品，而是希望更多的人使用它，于是在同一年Salvatore Sanfilippo将Redis开源发布，并开始和Redis的另一名主要的代码贡献者Pieter Noordhuis一起继续着Redis的开发，直到今天。

Salvatore Sanfilippo自己也没有想到，短短的几年时间，Redis就拥有了庞大的用户群体。Hacker News在2012年发布了一份数据库的使用情况调查，结果显示有近12%的公司在使用Redis。国内如新浪微博、街旁网、知乎网，国外如GitHub、Stack Overflow、Flickr等都是Redis的用户。VMware公司从2010年开始赞助Redis的开发，Salvatore Sanfilippo和Pieter Noordhuis也分别在3月和5月加入VMware，全职开发Redis。

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库。它通过提供多种键值数据类型来适应不同场景下的存储需求，目前为止Redis支持的键值数据类型如下：

- 字符串类型
- 散列类型
- 列表类型
- 集合类型
- 有序集合类型

Redis 与其他 key - value 缓存产品有以下三个特点：

- Redis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。
- 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。
- Redis支持数据的备份，即master-slave模式的数据备份。

Redis 提供的API支持：C、C++、C#、Clojure、Java、JavaScript、Lua、PHP、Python、Ruby、Go、Scala、Perl等多种语言。

1.4 Redis的应用场景

目前全球最大的Redis用户是新浪微博，在新浪有200多台物理机，400多个端口正在运行Redis，有+4G的数据在Redis上来为微博用户提供服务

- 取最新的N个数据（取最新文档、排行榜等）
- 需要精确设定过期时间的应用
- 计数器应用
- 实时性要求的高并发读写
- 消息系统Pub/Sub
- 构建队列
- 缓存

1.5 Redis优缺点

1.5.1 Redis 优势

对数据高并发读写（基于内存）
对海量数据的高效率存储和访问（基于内存）
对数据的可扩展性和高可用性
垂直扩展：提升硬件
水平扩展：集群

1.5.2 Redis 缺点

redis（ACID处理非常简单）无法做到太复杂的关系数据库模型

1.6 Redis面向互联网的解决方案

- 主从：一主多从，主机可写，从机备份。类似于Mysql的读写分离，存在问题是一旦主节点down掉，整个Redis不可用。
- 哨兵(2.x)：启用一个哨兵程序(节点)，监控其余节点的状态，根据选举策略，进行主从切换。
缺点：每个节点的数据依旧是一致的，仍无法实现分布式的数据库。
- 集群(3.x)：结合上述两种模式，多主多从，实现高可用、分布式数据存储

第二节 Redis的安装

2.1 下载

从官网下载，[Redis官网点击下载](#)

通过SecureCRT将下载的文件上传到/opt/work目录

解压&编译&安装

```
cd /opt/work          切换到指定目录
tar -zxvf redis-3.2.11.tar.gz    解压
cd redis-3.2.11        切换到解压目录
make                   编译
make PREFIX=/opt/work/redis install  安装
```

源码目录分析：

在/opt/work/src/redis3.2/下有一个redis.conf文件，这个文件为redis核心配置文件。

在/opt/work/src/redis3.2/src/下，有redis的常用命令，安装完成后，会将这些命令自动放入到安装路径下的bin目录下

在/opt/work/src/redis3.2/utils/下，有redis的服务启动脚本

2.2 配置和启动

redis.conf是redis的配置文件，redis.conf在redis源码目录。

注意修改port作为redis进程的端口,port默认6379。

拷贝配置文件到安装目录下

进入源码目录，里面有一份配置文件 redis.conf，然后将其拷贝到安装路径下

```
cd /opt/work/redis      切换目录
mkdir conf              创建配置文件的目录
cp /opt/work/redis-3.2.11/redis.conf /opt/work/redis/conf    复制配置文件
./redis-server           启动
```

redis关闭的方式

```
pkill redis server
kill 进程号
/opt/work/redis/bin/redis-cli shutdown
```

```
[root@CentOS6 redis-3.2.11]# cd ..
[root@CentOS6 work]# cd redis
[root@CentOS6 redis]# ls
bin  conf
[root@CentOS6 redis]# cd bin
[root@CentOS6 bin]# ./redis-server
11100:C 08 Nov 15:56:08.432 # Warning: no config file specified, using the default config. In order to specify a config file use ./redis-server /path/to/redis.conf
11100:M 08 Nov 15:56:08.433 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.2.11 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 11100

http://redis.io

11100:M 08 Nov 15:56:08.447 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
11100:M 08 Nov 15:56:08.447 # Server started, Redis version 3.2.11
11100:M 08 Nov 15:56:08.447 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
11100:M 08 Nov 15:56:08.448 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
11100:M 08 Nov 15:56:08.448 * The server is now ready to accept connections on port 6379
```

注意：这里直接执行Redis-server 启动的Redis服务，是在前台直接运行的(效果如上图)，也就是说，执行完该命令后，如果Linux关闭当前会话，则Redis服务也随即关闭。正常情况下，启动Redis服务需要从后台启动，并且指定启动配置文件。

2.3 后端模式启动

修改redis.conf配置文件， daemonize yes 以后端模式启动

执行如下命令启动redis：

`vim /opt/work/redis/conf/redis.conf` 编辑redis的配置文件，修改daemonize yes 保存即可

```
# of communication. This is useful for two reasons:
#
# 1) Detect dead peers.
# 2) Take the connection alive from the point of view of network
#    equipment in the middle.
#
# On Linux, the specified value (in seconds) is the period used to send ACKs.
# Note that to close the connection the double of the time is needed.
# On other kernels the period depends on the kernel configuration.
#
# A reasonable value for this option is 300 seconds, which is the new
# Redis default starting with Redis 3.2.1.
tcp-keepalive 300

##### GENERAL #####

# By default Redis does not run as a daemon. Use 'yes' if you need it.
# Note that Redis will write a pid file in /var/run/redis.pid when daemonized.
daemonize yes

# If you run Redis from upstart or systemd, Redis can interact with your
```

```
ps aux|grep redis      查询redis是否启动
/opt/work/redis/bin/redis-server    启动redis
```

redis.conf配置文件详解

4.1. Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程

```
daemonize no
```

4.2. 当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定

```
pidfile /var/run/redis.pid
```

4.3. 指定Redis监听端口，默认端口为6379，作者在自己的一篇博文中解释了为什么选用6379作为默认端口，因为6379在手机按键上MERZ对应的号码，而MERZ取自意大利歌女Alessia Merz的名字

```
port 6379
```

4.4. 绑定的主机地址

```
bind 127.0.0.1
```

4.5. 当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能

```
timeout 300
```

4.6. 指定日志记录级别，Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose

```
loglevel verbose
```

4.7. 日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null

```
logfile stdout
```

4.8. 设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id

```
databases 16
```

4.9. 指定在多长时间，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合

```
save <seconds> <changes>
```

Redis默认配置文件中提供了三个条件：

```
save 900 1
```

```
save 300 10
```

```
save 60 10000
```

分别表示900秒（15分钟）内有1个更改，300秒（5分钟）内有10个更改以及60秒内有10000个更改。

4.10. 指定存储至本地数据库时是否压缩数据，默认为yes，Redis采用LZF压缩，如果为了节省CPU时间，可以关闭该选项，但会导致数据库文件变的巨大

```
rediscompression yes
```

4.11. 指定本地数据库文件名，默认值为dump.rdb

```
dbfilename dump.rdb
```

4.12. 指定本地数据库存放目录

```
dir ./
```

4.13. 设置当本机为slav服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步

```
slaveof <masterip> <masterport>
```

4.14. 当master服务设置了密码保护时，slav服务连接master的密码

```
masterauth <master-password>
```

4.15. 设置Redis连接密码，如果配置了连接密码，客户端在连接Redis时需要通过AUTH <password>命令提供密码，默认关闭

```
requirepass foobared
```

4.16. 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，Redis会关闭新的连接并向客户端返回max number of clients reached错误信息

```
maxclients 128
```

4.17. 指定Redis最大内存限制，Redis在启动时会把数据加载到内存中，达到最大内存后，Redis会先尝试清除已到期或即将到期的Key，当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis新的vm机制，会把Key存放内存，Value会存放在swap区

```
maxmemory <bytes>
```


4.18. 指定是否在每次更新操作后进行日志记录, Redis在默认情况下是异步的把数据写入磁盘, 如果不开启, 可能会在断电时导致一段时间内的数据丢失。因为 redis本身同步数据文件是按上面save条件来同步的, 所以有的数据会在一段时间内只存在于内存中。默认为no

```
appendonly no
```

4.19. 指定更新日志文件名, 默认为appendonly.aof

```
appendfilename appendonly.aof
```

4.20. 指定更新日志条件, 共有3个可选值:

no: 表示等操作系统进行数据缓存同步到磁盘 (快)

always: 表示每次更新操作后手动调用fsync()将数据写到磁盘 (慢, 安全)

everysec: 表示每秒同步一次 (折衷, 默认值)

```
appendfsync everysec
```

4.21. 指定是否启用虚拟内存机制, 默认值为no, 简单的介绍一下, VM机制将数据分页存放, 由Redis将访问量较少的页即冷数据swap到磁盘上, 访问多的页面由磁盘自动换出到内存中 (在后面的文章我会仔细分析Redis的VM机制)

```
vm-enabled no
```

4.22. 虚拟内存文件路径, 默认值为/tmp/redis.swap, 不可多个Redis实例共享

```
vm-swap-file /tmp/redis.swap
```

4.23. 将所有大于vm-max-memory的数据存入虚拟内存, 无论vm-max-memory设置多小, 所有索引数据都是内存存储的(Redis的索引数据 就是keys), 也就是说, 当vm-max-memory设置为0的时候, 其实是所有value都存在于磁盘。默认值为0

```
vm-max-memory 0
```

4.24. Redis swap文件分成了很多的page, 一个对象可以保存在多个page上面, 但一个page上不能被多个对象共享, vm-page-size是要根据存储的 数据大小来设定的, 作者建议如果存储很多小对象, page大小最好设置为32或者64bytes; 如果存储很大大对象, 则可以使用更大的page, 如果不 确定, 就使用默认值

```
vm-page-size 32
```

4.25. 设置swap文件中的page数量, 由于页表 (一种表示页面空闲或使用的bitmap) 是在放在内存中的, , 在磁盘上每8个pages将消耗1byte的内存。

```
vm-pages 134217728
```

4.26. 设置访问swap文件的线程数, 最好不要超过机器的核数, 如果设置为0, 那么所有对swap文件的操作都是串行的, 可能会造成比较长时间的延迟。默认值为4

```
vm-max-threads 4
```

4.27. 设置在向客户端应答时, 是否把较小的包合并为一个包发送, 默认为开启

```
glueoutputbuf yes
```

4.28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法

```
hash-max-zipmap-entries 64
```

```
hash-max-zipmap-value 512
```

4.29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）

```
activeresharding yes
```

4.30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

```
include /path/to/local.conf
```

2.4 启动多个redis进程

2.4.1 启动时指定端口

启动时指定端口可在一台服务器启动多个redis进程

```
cd /opt/work/redis/bin
./redis-server ../conf/redis.conf --port 6380
```

2.4.2 创建多个redis目录

创建多个redis目录，以端口号命名,推荐使用此种方式

比如：创建6381、6382两个目录，将redis的安装文件bin和conf拷贝至这两个目录。

修改6381目录下的redis.conf设置端口号为6381

修改6382目录下的redis.conf设置端口号为6382

启动6381和6382目录下的redis-server程序：

```
cd 6381
./redis-server . /redis.conf
cd 6382
./redis-server . /redis.conf
```

2.5 redis客户端

在redis的安装目录中有redis的客户端，即redis-cli (Redis Command Line Interface)，它是Redis自带的基于命令行的Redis客户端

```
ps aux|grep redis    查询redis是否启动
./redis-server ../conf/redis.conf    启动redis
./redis-cli -h 127.0.0.1 -p 6379    启动redis客户端
ping    Redis提供了PING命令来测试客户端与Redis的连接是否正常，如果连接正常会收到回复
PONG
```

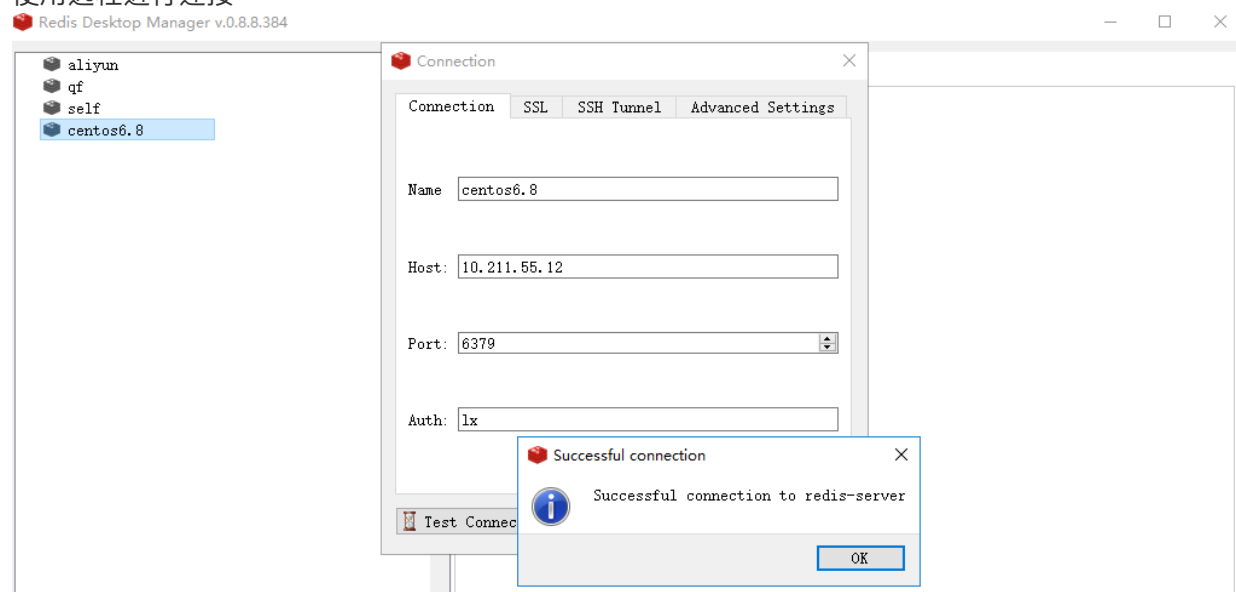
```
[root@CentOS6 bin]# ps aux|grep redis
root      11322  0.0   0.0 103316   848 pts/2    S+   16:33   0:00 grep redis
[root@CentOS6 bin]# ./redis-server ../conf/redis.conf
[root@CentOS6 bin]# ps aux|grep redis
root      11324  0.0   0.2 133536   2184 ?        Ssl  16:33   0:00 ./redis-server
127.0.0.1:6379
root      11328  0.0   0.0 103316   848 pts/2    S+   16:34   0:00 grep redis
[root@CentOS6 bin]# ./redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> █
```

2.6 远程连接

默认不允许远程连接，需要修改一下信息才可以进行修改

```
将redis.conf中的bind 127.0.0.1进行注释
vim /opt/work/redis/conf/redis.conf    编辑配置文件
/opt/work/redis/bin/redis-server ../conf/redis.conf    启动redis
/opt/work/redis/bin/redis-cli    打开客户端，连接成功，再进行下一步
config set requirepass lx    设置密码
quit    退出客户端
/opt/work/redis/bin/redis-cli    打开客户端
auth lx    输入密码
```

使用远程进行连接



第三节 redis常用命令

3.1 基础命令

3.1.1 操作String类型

String 数据结构是简单的key-value类型，value其实不仅是String，也可以是数字，是包含很多种类型的特殊类型，并且是二进制安全的。比如序列化的对象进行存储，比如一张图片进行二进制存储，比如一个简单的字符串，数值等等。

常用命令：

设值：set name zhangsan (说明：多次设置name会覆盖)

命令：

setnx key1 value1: (not exist) 如果key1不存在，则设值 并返回1。如果key1存在，则不设值并返回0；

setex key1 10 lx :(expired) 设置key1的值为lx，过期时间为10秒，10秒后key1清除 (key也清除)

setrange string range value 替换字符串

取值：get key

删值：del keys

批量写：mset k1 v1 k2 v2 ... 一次性写入多个值

批量读：mget k1 k2 k3

一次性设值和读取 (返回旧值，写上新值) :getset name lx

数值类型自增减：incr key, decr key 注意这些 key 对应的必须是数字类型字符串,否则会出错,自增或者自减1

自增或自减指定长度 incrby key increment, decrby key increment 对应的 key 自增或者自减 increment值

字符串尾部拼接：append key value 向 key 对应的字符串尾部追加 value

字符串长度：strlen key

命令演示

```
[root@CentOS6 bin]# ./redis-cli
127.0.0.1:6379> auth lx
OK
127.0.0.1:6379> set username zhangsan
OK
127.0.0.1:6379> get username
"zhangsan"
127.0.0.1:6379> setnx username lisi
(integer) 0
127.0.0.1:6379> setex username 10 ll
OK
127.0.0.1:6379> get username
"ll"
127.0.0.1:6379> get username
"ll"
127.0.0.1:6379> get username
(nil)
127.0.0.1:6379> set name lx
OK
127.0.0.1:6379> get name
"lx"
127.0.0.1:6379> █
```

```
127.0.0.1:6379> set pw admin123
OK
127.0.0.1:6379> get pw
"admin123"
127.0.0.1:6379> setrange pw 1 user
(integer) 8
127.0.0.1:6379> get pw
"auser123"
127.0.0.1:6379> █
```

setrange pw 1 user 将key为pw的值从索引为1的开始进行替换，后面的由原来的字符补齐

3.1.2 Hash类型

Hash类型是String类型的field和value的映射表，或者说是一个String集合。它特别适合存储对象，相比较而言，将一个对象类型存储在Hash类型要存储在String类型里占用更少的内存空间，并方便整个对象的存取。

常用命令

设值: `hset hostname field value` (`hset`是设值命令, `hostname`是集合名字, `field`是字段名, `value`是值)

取值: `hget hostname field`

批量设置: `hmset hostname field1 value1 field2 value2 ...`

批量取值: `hmget hostname field1 field2 ...`

`hsetnx key field value`: 和`setnx`大同小异

`HINCRBY key field increment`: 指定字段增加指定值

`hexists key field`: 指定 `key` 中是否存在指定 `field`, 如果存在返回1, 不存在返回0

`hdel key field` 删除指定`key`的`hash`的`field`

`hlen`: 返回`hash`集合里的所有的键数量(`size`)

`hkeys key`: 返回`hash`里所有的`field`

`hvals key`: 返回`hash`的所有`field` 对应的 `value`

`hgetall key`: 返回`hash`里所有的`field`和`value`

命令演示

```
127.0.0.1:6379> hset users name 'admin'
(integer) 1
127.0.0.1:6379> hget users name
"admin"
127.0.0.1:6379> hmset users pass '123' age 12
OK
127.0.0.1:6379> hmget users name age pass
1) "admin"
2) "12"
3) "123"
127.0.0.1:6379> hsetnx users name 'user'
(integer) 0
```

```
127.0.0.1:6379> hget users age
"15"
127.0.0.1:6379> hincrby users age 1
(integer) 16
127.0.0.1:6379> hget users age
"16"
127.0.0.1:6379> hexists users name
(integer) 1
127.0.0.1:6379> hdel user name
(integer) 0
127.0.0.1:6379> hexists users name
(integer) 1
127.0.0.1:6379> hlen users
(integer) 3
127.0.0.1:6379> hkeys users
1) "name"
2) "pass"
3) "age"
```

```
127.0.0.1:6379> hgetall users
1) "name"
2) "admin"
3) "pass"
4) "123"
5) "age"
6) "16"
127.0.0.1:6379>
```

3.1.3 List类型

List类型是一个链表结构的集合，其主要功能有push、pop、获取元素等。更详细的说，List类型是一个双端链表的节后，我们可以通过相关的操作进行集合的头部或者尾部添加和删除元素，List的设计非常简单精巧，即可以作为栈，又可以作为队列，满足绝大多数的需求。

常用命令

`lpush key1 value1 value2...`: 从头部加入元素（栈，先进后出）

`rpush key1 value1 value2 ...`: 从尾部加入元素（队列，先进先出）

`linsert key BEFORE|AFTER pivot value`

该命令首先会在列表中从左到右查找值为pivot的元素，然后根据第二个参数是BEFORE还是AFTER来决定将value插入到该元素的前面还是后面

`lrange key start stop`: 获取指定索引内的所有元素, 只可从左到右 0 -1代表所有

`lset key index value`: 将key 集合中 index下表的元素替换掉

`lrem key count value`

`lrem`命令会删除列表中前count个值为value的元素，返回实际删除的元素个数。根据count值的不同，该命令的执行方式会有所不同：

当count>0时， `LREM`会从列表左边开始删除。

当count<0时， `LREM`会从列表后边开始删除。

当count=0时， `LREM`删除所有值为value的元素。

`ltrim key start stop`: 保留制定key的值范围内的数据，其他数据会删掉，和 `lrange` 一样的参数范围

`lpop key`: 从list的头部删除元素，并返回删除元素。

`rpop key`: 从list的尾部删除元素，并返回删除元素

`rpoplpush list1 list2`: 从list1尾部删除元素，并将被移除的元素添加到list2的头部，返回被移除的元素，可以实现MQ

`llen key`: 返回元素个数

`lindex key index`: 返回名称为key的list中index位置的元素

命令演示

```
127.0.0.1:6379> lpush list1 aaaa
(integer) 1
127.0.0.1:6379> lpush list1 bbbb
(integer) 2
127.0.0.1:6379> lrange list1 0 1
1) "bbbb"
2) "aaaa"
127.0.0.1:6379> rpush list1 cccc
(integer) 3
127.0.0.1:6379> linsert list1 before bbbb ffff
(integer) 4
127.0.0.1:6379> linsert list1 after bbbb www
(integer) 5
127.0.0.1:6379> lrange list1 0 5
1) "ffff"
2) "bbbb"
3) "www"
4) "aaaa"
5) "cccc"
127.0.0.1:6379> █
```

```
127.0.0.1:6379> lset list1 3 abc
OK
127.0.0.1:6379> lrem list1 1 cccc
(integer) 1
127.0.0.1:6379> lrange list1 0 5
1) "ffff"
2) "bbbb"
3) "www"
4) "abc"
127.0.0.1:6379> lpush list1 gggg
(integer) 5
127.0.0.1:6379> lpop list1
"gggg"
127.0.0.1:6379> rpop list1
"abc"
127.0.0.1:6379> lrange list1 0 5
1) "ffff"
2) "bbbb"
3) "www"
127.0.0.1:6379> █
```



```
127.0.0.1:6379> rpoplpush list1 list2
"www"
127.0.0.1:6379> lrange list2 0 1
1) "www"
127.0.0.1:6379> llen list1
(integer) 2
127.0.0.1:6379> lindex list1 0
"ffff"
127.0.0.1:6379> █
```

3.1.4 Set类型

set集合是string类型的无序集合，set是通过hashtable实现的，对集合我们可以取交集、并集、差集

常用命令

SADD key member [member ...]: 向名称为key的set中添加元素，set集合不允许重复元素。
SMEMBERS key: 查看set集合中的元素。
SREM key member [member ...]: 删除set集合的元素
SPOP key: 随机删除指定set中的一个内容并将删除的内容返回
SDIFF key [key ...]: 差集运算,返回在第一个set 中存在,第二个set 中不存在的内容
sdiffstore set4 set2 set3 将set2 set3不同元素的比较结果保存到set4中
SINTER key [key ...]: 取交集,集合重复的数据
sinterstore: set3 set1 set2取交集后保存到 set3
SUNION key [key ...]:取并集,因为是 set 所以相同部分只会取一次
sunionstore set3 set1 set2: 取并集后保存到 set1
smove set1 set2: 从一个set集合移动到另一个set集合里
SCARD key: 查看集合里的元素个数
SISMEMBER key member: 判断某个元素是否为集合中的元素，是，返回1。不是，返回0。
srandmember key: 随机返回一个元素

命令演示

```
127.0.0.1:6379> sadd st1 1111
(integer) 1
127.0.0.1:6379> sadd st1 2222
(integer) 1
127.0.0.1:6379> sadd st1 3333
(integer) 1
127.0.0.1:6379> sadd st2 aaaa
(integer) 1
127.0.0.1:6379> sadd st2 bbbb
(integer) 1
127.0.0.1:6379> smembers st1
1) "1111"
2) "2222"
3) "3333"
127.0.0.1:6379> srem st1 1111
(integer) 1
127.0.0.1:6379> spop st2
"aaaa"
127.0.0.1:6379> sdiff st1 st2
1) "2222"
2) "3333"
127.0.0.1:6379> sdiffstore st2 st1
(integer) 2
127.0.0.1:6379> smembers st2
1) "2222"
2) "3333"
127.0.0.1:6379> smembers st1
1) "2222"
2) "3333"
127.0.0.1:6379> █
```

```
127.0.0.1:6379> sadd st3 abc
(integer) 1
127.0.0.1:6379> sadd st3 dfg
(integer) 1
127.0.0.1:6379> sadd st4 dfg
(integer) 1
127.0.0.1:6379> sadd st4 222
(integer) 1
127.0.0.1:6379> sinter st3 st4
1) "dfg"
127.0.0.1:6379> sinterstore st3 st4 st5
(integer) 0
127.0.0.1:6379> smembers st5
(empty list or set)
```

```
127.0.0.1:6379> smembers st4
1) "222"
2) "dfg"
127.0.0.1:6379> smembers st3
(empty list or set)
127.0.0.1:6379> sadd st5 23
(integer) 1
127.0.0.1:6379> sunion st4 st5
1) "dfg"
2) "222"
3) "23"
127.0.0.1:6379> sunionstore st4 st5
(integer) 1
127.0.0.1:6379> smembers st5
1) "23"
127.0.0.1:6379> smembers st4
1) "23"
127.0.0.1:6379> scard st5
(integer) 1
127.0.0.1:6379> smove st4 st5 23
(integer) 1
127.0.0.1:6379> sismember st5 1
(integer) 0
127.0.0.1:6379> srandmember st5
"23"
127.0.0.1:6379> █
```

3.1.5 Zset类型

有序集合和集合一样也是string类型元素的集合,且不允许重复的成员。不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

有序集合的成员是唯一的,但分数(score)却可以重复。

集合是通过哈希表实现的,所以添加,删除,查找的复杂度都是 $O(1)$ 。集合中最大的成员数为 $2^{32} - 1$ (4294967295, 每个集合可存储40多亿个成员)。

常用命令

ZADD key score member [score member ...]: score 是分, member 是内容, score 必须是数字,向有序集合中添加一个元素,该元素如果存在则更新顺序,如果分值相同元素不同会同时存在两个元素。

ZSCORE key member 获取指定key 中指定内容的分数

ZREM key member [member ...] : 删除zset名称key中的member元素

ZRANGE key start stop [WITHSCORES] 获得排名在某个范围的元素列表,照元素分数从小到大的顺序返回索引从start到stop之间的所有元素(包含两端的元素) [WITHSCORES]为可选项,代表是否在结果中显示分数

ZREVRANGE key start stop [WITHSCORES] 照元素分数从大到小的顺序返回索引从start到stop之间的所有元素(包含两端的元素)

ZRANK key member 返回有序集合中指定成员的索引(从小到大排序)

ZREVRANK key member 返回有序集合中指定成员的排名,有序集成员按分数值递减(从大到小)排序

ZCARD key 返回集合里所有元素的个数

ZCOUNT key min max 返回集合中score在给定区间中的数量

zincrby key increment member: 有序集合中对指定成员的分数加上增量 increment

zrangebyscore key min max [WITHSCORES] [LIMIT offset count] : 通过分数返回有序集合指定区间内的成员 min max 代表分数范围,offset 代表偏移量, count 代表获取多少个,类似于数据库

zremrangebyrank key start stop : 移除有序集合中给定的排名区间的所有成员

zremrangebyscore key min max: 移除有序集合中给定的分数区间的所有成员

ZINCRBY key increment member 增加member元素的分数increment,返回值是更改后的分数

命令演示

```
127.0.0.1:6379> zadd zt1 1 zhangsan
(integer) 1
127.0.0.1:6379> zadd zt1 2 lisi
(integer) 1
127.0.0.1:6379> zadd zt1 2 wangwu
(integer) 1
127.0.0.1:6379> zrange zt1 0 3 withscores
1) "zhangsan"
2) "1"
3) "lisi"
4) "2"
5) "wangwu"
6) "2"
127.0.0.1:6379> zcard zt1
(integer) 3
127.0.0.1:6379> zcount zt1 1 2
(integer) 3
127.0.0.1:6379> zrem zt1 lisi
(integer) 1
127.0.0.1:6379> zrank zt1 wangwu
(integer) 1
127.0.0.1:6379> zrevrank zt1 zhangsan
(integer) 1
127.0.0.1:6379> zincrby zt1 3 zhangsan
"4"
127.0.0.1:6379> zrangebyscore zt1 0 5 withscores
1) "wangwu"
2) "2"
3) "zhangsan"
4) "4"
127.0.0.1:6379> zremrangebyrank zt1 1 2
(integer) 1
```

```
127.0.0.1:6379> zadd zt2 1 aaa 2 bbb 3 ccc
(integer) 3
127.0.0.1:6379> zrange zt2 1 3
1) "bbb"
2) "ccc"
127.0.0.1:6379> zrange zt2 0 3
1) "aaa"
2) "bbb"
3) "ccc"
127.0.0.1:6379> zrange zt2 0 3 withscores
1) "aaa"
2) "1"
3) "bbb"
4) "2"
5) "ccc"
6) "3"
127.0.0.1:6379> zremrangebyscore zt2 0 1
(integer) 1
127.0.0.1:6379> zrange zt2 0 3 withscores
1) "bbb"
2) "2"
3) "ccc"
4) "3"
127.0.0.1:6379> █
```

3.1.6 HyperLogLog

Redis 在 2.8.9 版本添加了 HyperLogLog 结构。

Redis HyperLogLog 是用来做基数统计的算法, HyperLogLog 的优点是, 在输入元素的数量或者体积非常非常大时, 计算基数所需的空间总是固定 的、并且是很小的。

在 Redis 里面, 每个 HyperLogLog 键只需要花费 12 KB 内存, 就可以计算接近 2^{64} 个不同元素的基数。这和计算基数时, 元素越多耗费内存就越多的集合形成鲜明对比。

但是, 因为 HyperLogLog 只会根据输入元素来计算基数, 而不会储存输入元素本身, 所以 HyperLogLog 不能像集合那样, 返回输入的各个元素。

小知识:

什么是基数?

比如数据集 {1, 2, 1, 2} 那么这个数据集的基数集为 {1, 2}, 基数(不重复元素)为2。 基数估计就是在误差可接受的范围内, 快速计算基数。

常用命令:

PFADD	新增元素
PFCOUNT	获取基数的估计值
PFMERGE	将多个 HyperLogLog 合并为一个 HyperLogLog

命令演示

```
127.0.0.1:6379> pfadd hlog1 aaaa
(integer) 1
127.0.0.1:6379> pfadd hlog1 bbbb
(integer) 1
127.0.0.1:6379> pfadd hlog1 aaaa
(integer) 0
127.0.0.1:6379> pfcount hlog1
(integer) 2
127.0.0.1:6379> pfadd hlog2 cccc
(integer) 1
127.0.0.1:6379> pfmerge hlog3 hlog1 hlog2
OK
127.0.0.1:6379> pfcount hlog3
(integer) 3
127.0.0.1:6379> █
```

3.2 高级命令

3.2.1 常用命令

keys * : 返回满足的所有键 , 可以模糊匹配 比如 keys abc* 代表 abc 开头的 key

exists key : 是否存在指定的key, 存在返回1, 不存在返回0

expire key second: 设置某个key的过期时间 时间为秒

del key: 删除某个key

ttl key: 查看剩余时间, 当key不存在时, 返回 -2; 存在但没有设置剩余生存时间时, 返回 -1, 否则, 以秒为单位, 返回 key 的剩余生存时间。

persist key: 取消过去时间

PEXPIRE key milliseconds 修改key 的过期时间为毫秒

`select` : 选择数据库 数据库为0-15（默认一共16个数据库） `s`
设计成多个数据库实际上是为了数据库安全和备份
`move key dbindex` : 将当前数据中的key转移到其他数据库
`randomkey`: 随机返回一个key
`rename key key2`: 重命名key
`echo`: 打印命令
`dbsize`: 查看数据库的key数量
`info`: 查看数据库信息
`config get *` 实时传储收到的请求，返回相关的配置
`flushdb` : 清空当前数据库
`flushall` : 清空所有数据库

命令演示

```
127.0.0.1:6379> keys *
1) "a"
127.0.0.1:6379> exists a
(integer) 1
127.0.0.1:6379> expire a 0
(integer) 1
127.0.0.1:6379> get a
(nil)
127.0.0.1:6379> set b 111
OK
127.0.0.1:6379> ttl b
(integer) -1
127.0.0.1:6379> expire b 20
(integer) 1
127.0.0.1:6379> persist b
(integer) 1
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> set c 222
OK
127.0.0.1:6379[1]> move c 0
(integer) 1
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get c
"222"
127.0.0.1:6379> randomkey
"c"
127.0.0.1:6379> rename c c1
OK
127.0.0.1:6379> echo lx
"lx"
127.0.0.1:6379> dbsize
(integer) 2
127.0.0.1:6379> info
# Server
redis_version:3.2.11
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:1ba5a72a190793c9
redis_mode:standalone
os:Linux 2.6.32-642.el6.x86_64 x86_64
arch_bits:64
multiplexing_api:epoll
gcc_version:4.4.7
process_id:10487
```

3.2.2 Redis事务

Redis 事务可以一次执行多个命令，并且带有以下两个重要的保证：

事务是一个单独的隔离操作：

事务中的所有命令都会序列化、按顺序地执行。

事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：

事务中的命令要么全部被执行，要么全部都不执行

一个事务从开始到执行会经历以下三个阶段：

开始事务
命令入队
执行事务

常用命令

MULTI	开启事务
EXEC	执行事务
DISCARD	取消事务
WATCH key	监听某个 key 的值是否发生变化,如果发生变化, watch 之后的操作会失败

命令演示：

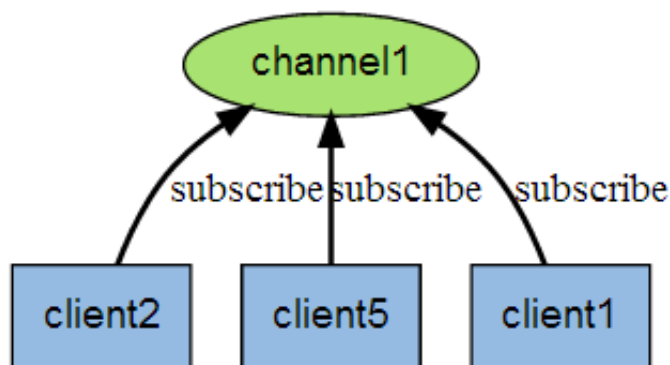
```
127.0.0.1:6379> keys *
1) "b"
2) "c1"
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set a 1
QUEUED
127.0.0.1:6379> set d 3
QUEUED
127.0.0.1:6379> get c1
QUEUED
127.0.0.1:6379> exec
1) OK
2) OK
3) "222"
127.0.0.1:6379> █
```

3.2.3 发布与订阅消息

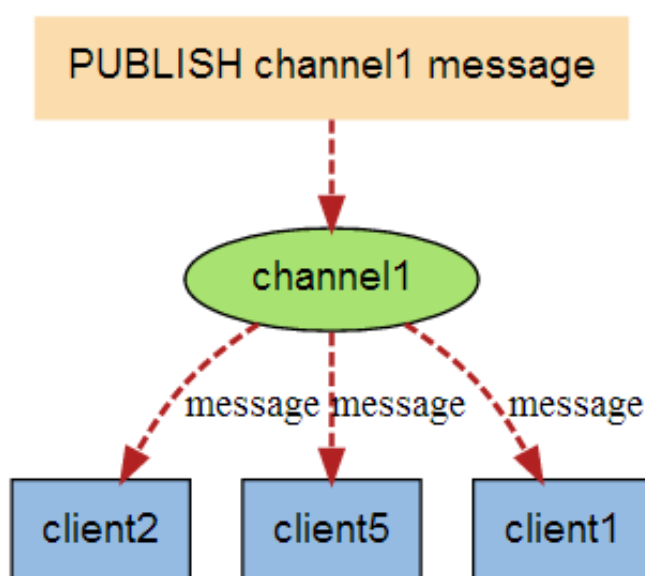
Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

Redis 客户端可以订阅任意数量的频道。

下图展示了频道 channel1，以及订阅这个频道的三个客户端 — client2、client5 和 client1 之间的关系



当有新消息通过 PUBLISH 命令发送给频道 channel1 时，这个消息就会被发送给订阅它的三个客户端：

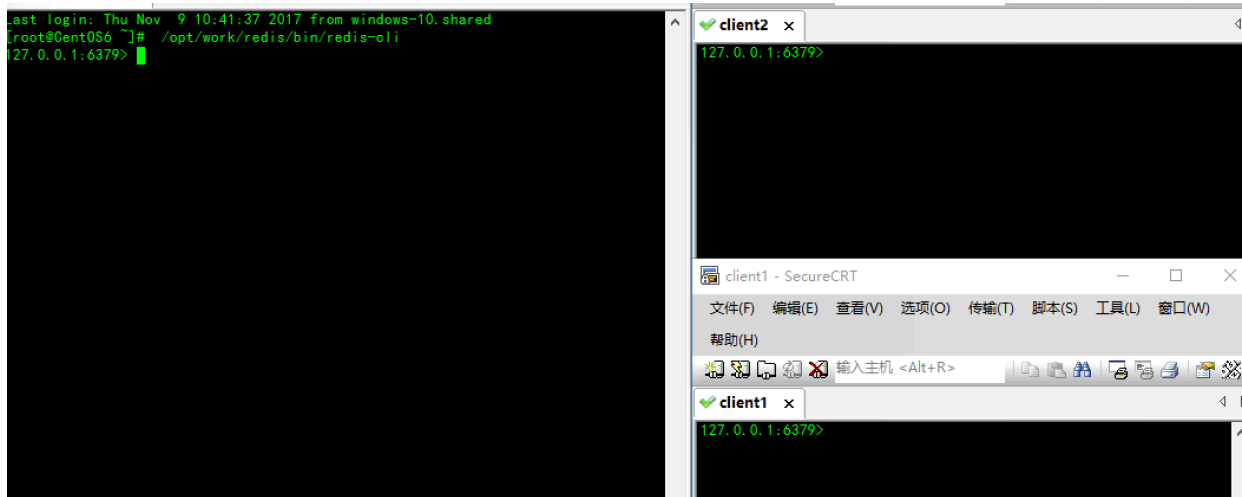


配置订阅和发布

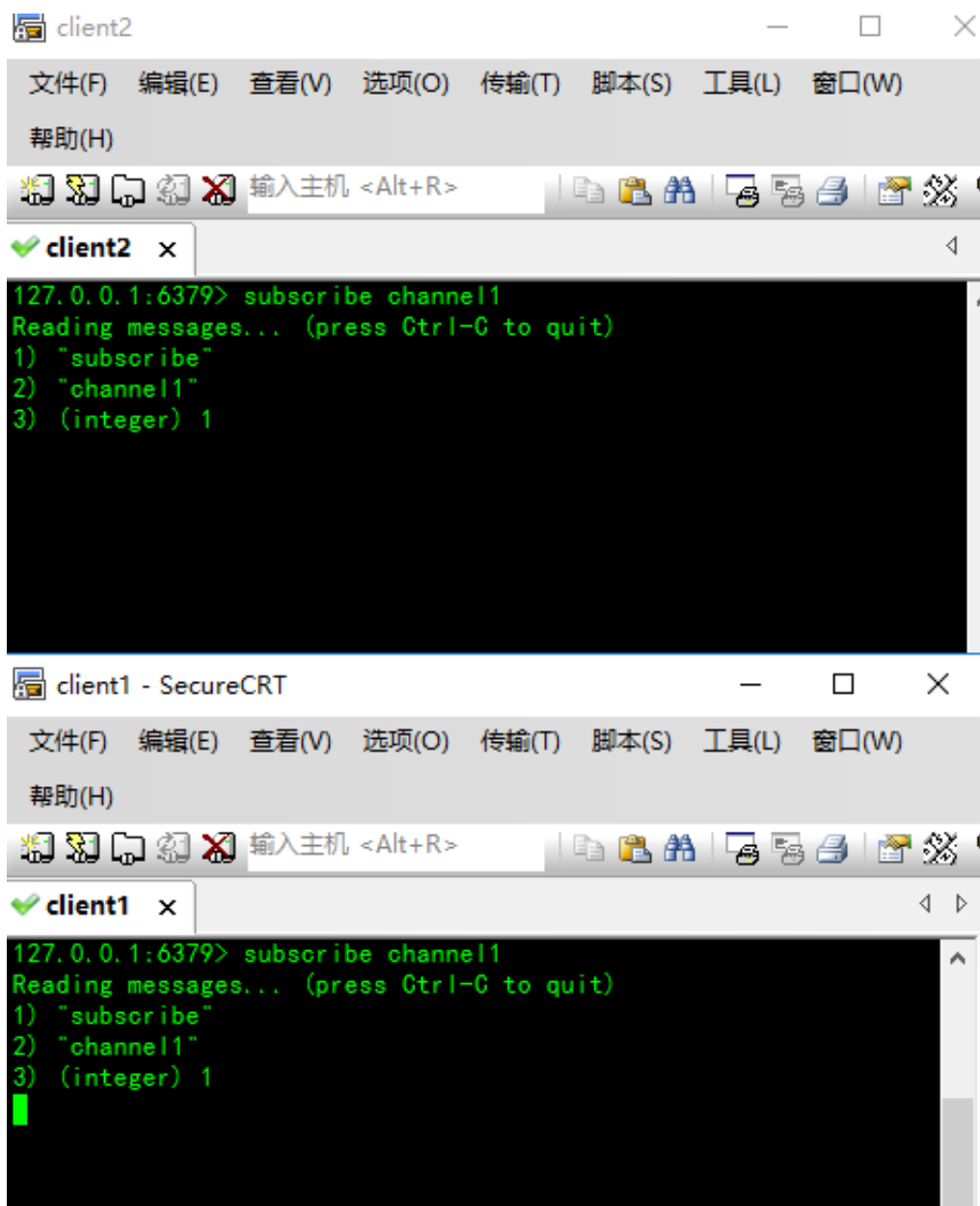
常用命令

`subscribe [频道]` 进行订阅监听
`publish [频道 发布内容]` 进行发布消息广播

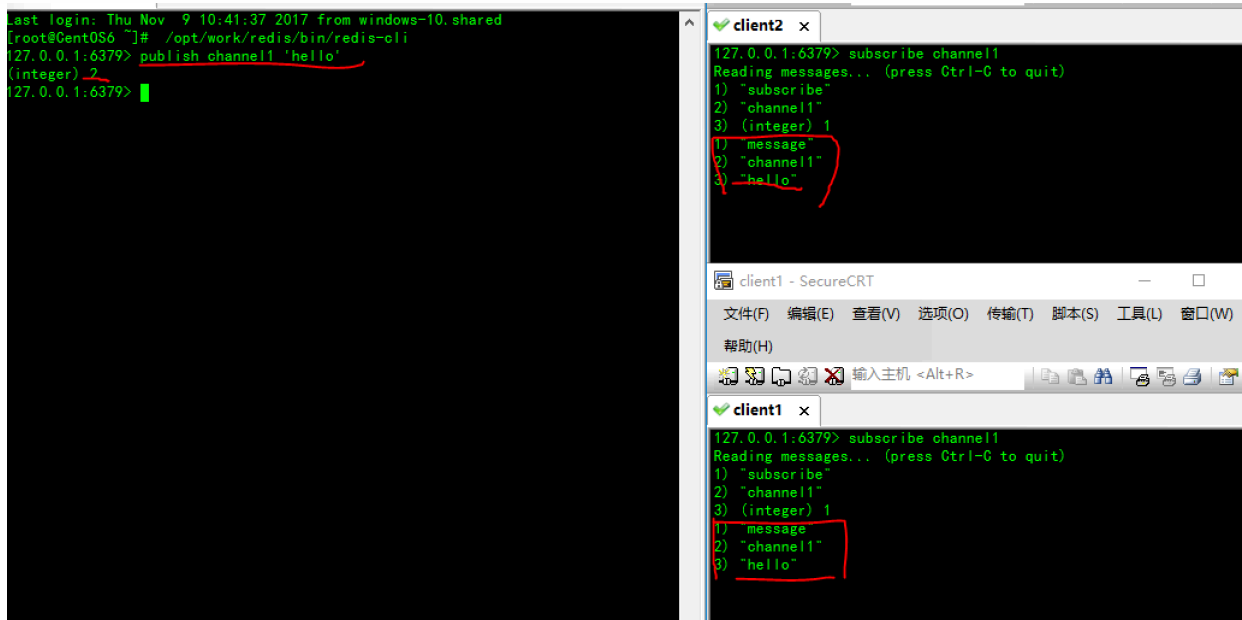
1、分别克隆额外的2个会话，重命名client1、client2，而且三个会话都运行redis的客户端



2、分别在client1和client2进行订阅



3、进行消息发布



The screenshot shows two terminal windows. The left window is the Redis CLI, where the command `publish channel1 'hello'` is executed. The right window is a SecureCRT session titled 'client2', which has subscribed to 'channel1' and is receiving the message 'hello' as part of a list of three items: 'subscribe', 'channel1', and 'hello'.

```
last login: Thu Nov 9 10:41:37 2017 from windows-10.shared
[root@CentOS6 ~]# /opt/work/redis/bin/redis-cli
127.0.0.1:6379> publish channel1 'hello'
(integer) 2
127.0.0.1:6379>
```

```
client2 x
127.0.0.1:6379> subscribe channel1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channel1"
3) (integer) 1
1) "message"
2) "channel1"
3) "hello"
```

client1 - SecureCRT

```
client1 x
127.0.0.1:6379> subscribe channel1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channel1"
3) (integer) 1
1) "message"
2) "channel1"
3) "hello"
```

3.2.4 Redis 数据备份与恢复

数据备份

Redis `SAVE` 命令用于创建当前数据库的备份。

恢复数据

如果需要恢复数据，只需将备份文件（`dump.rdb`）移动到 `redis` 安装目录并启动服务即可。

```
127.0.0.1:6379> save
OK
127.0.0.1:6379> quit
[root@CentOS6 ~]# cd /opt/work/redis
[root@CentOS6 redis]# ls
bin  conf  dump.rdb
```

3.2.5 Redis 安全

因为redis速度相当快，所以一台比较好的服务器下，一个外部用户在一秒内可以进行15W次密码尝试，这意味着你需要设定非常强大的密码来防止暴力破解

可以通过 `redis` 的配置文件设置密码参数，这样客户端连接到 `redis` 服务就需要密码验证，这样可以让你的 `redis` 服务更安全

常用命令

```
vim /opt/work/redis/conf/redis.conf      编辑配置文件，
修改: #requirepass foobared 为:         requirepass redis(你的密码)
```

```
# the port or the IP address.
#
# slave-announce-ip 5.5.5.5
# slave-announce-port 1234

##### SECURITY #####

# Require clients to issue AUTH <PASSWORD> before processing any
# commands. This might be useful in environments in which you do
# others with access to the host running redis-server.
#
# This should stay commented out for backward compatibility and b
# people do not need auth (e.g. they run their own servers).
#
# Warning: since Redis is pretty fast an outside user can try up
# 150k passwords per second against a good box. This means that y
# use a very strong password otherwise it will be very easy to br
#
requirepass lx

# Command renaming.
#
# It is possible to change the name of dangerous commands in a sh
# environment. For instance the CONFIG command may be renamed int
# hard to guess so that it will still be available for internal-u
— INSEFT — 480
```

```
kill redis-server      关闭redis-server
./bin/redis-server ./conf/redis.conf  启动redis
./bin/redis-cli        打开客户端
```

```
[root@CentOS6 redis]# kill redis-server
[root@CentOS6 redis]# ./bin/redis-server ./conf/redis.conf
[root@CentOS6 redis]# ./bin/redis-cli
127.0.0.1:6379> keys *
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth lx
OK
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379>
```

第四节 redis高级使用

4.1 主从复制

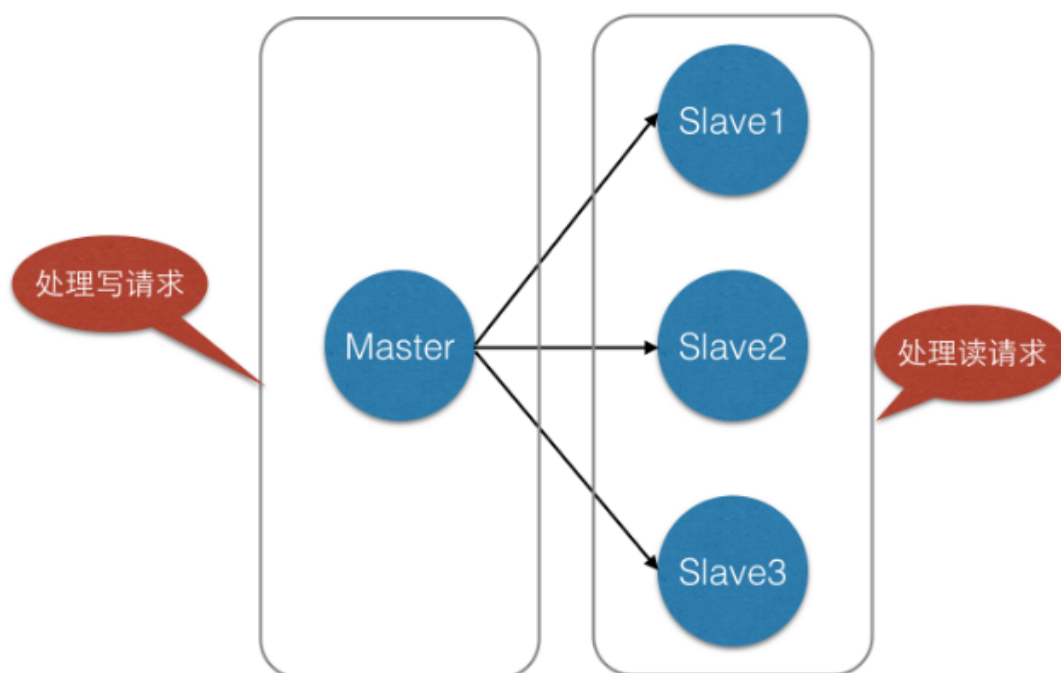
使用一台服务器进行模拟Redis的主从复制

4.1.1 概念

一般来说，要将Redis运用于工程项目中，只使用一台Redis是万万不能的，原因如下：
从结构上，单个Redis服务器会发生单点故障，并且一台服务器需要处理所有的请求负载，压力较大；
从容量上，单个Redis服务器内存容量有限，就算一台Redis服务器内容容量为256G，也不能将所有内容用作Redis存储内存，一般来说，单台Redis最大使用内存不应该超过20G。

考虑如下一种场景：电子商务网站上的商品，一般都是一次上传，无数次浏览的，说专业点也就是“多读少写”。

对于这种场景，我们可以使如下这种架构：



如图所示，将一台Redis服务器作主库(Master)，其他三台作为从库(Slave)，主库只负责写数据，每次有数据更新都将更新的数据同步到它所有的从库，而从库只负责读数据。

这样一来，就有了两个好处：

读写分离，不仅可以提高服务器的负载能力并且可以根据读请求的规模自由增加或者减少从库的数量棒极了；

数据被复制成了好几份，就算有一台机器出现故障，也可以使用其他机器的数据快速恢复。需要注意的是：在Redis主从模式中，一台主库可以拥有多个从库，但是一个从库只能隶属于一个主库。

4.1.2 redis主从复制特点

- 1、Master可以拥有多个Slave；
- 2、多个slave可以连接同一个Master，还可以链接到其他slave
- 3、主从复制不会阻塞Master，在同步数据时，master可以继续处理client请求
- 4、提供系统的伸缩性。

4.1.3 redis主从复制配置

在Redis中，要实现主从复制架构非常简单，只需要在从数据库的配置文件中加上如下命令即可：

```
slaveof 主数据库地址 主数据库端口
```

主数据库不需要任何配置。

配置步骤：

- 1、重新安装一份redis,作为从库，拷贝redis.conf 并修改

端口号为6380

设置slaveof 主库ip地址 端口号

设置masterauth 主库密码

```
[root@CentOS6 work]# mkdir redisslave1
[root@CentOS6 work]# cd redis-3.2.11
[root@CentOS6 redis-3.2.11]# make prefix=/opt/work/redisslave1 install
cd src && make install
make[1]: Entering directory `/opt/work/redis-3.2.11/src'

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

INSTALL install
INSTALL install
INSTALL install
INSTALL install
INSTALL install
make[1]: Leaving directory `/opt/work/redis-3.2.11/src'
[root@CentOS6 redis-3.2.11]# cp /opt/work/redis-3.2.11/redis.conf /opt/work/redisslave1/bin
[root@CentOS6 redis-3.2.11]#
```

```
# The Append Only File will also be created inside this directory.
#
# Note that you must specify a directory here, not a file name.
dir ./

##### REPLICATION #####

# Master-Slave replication. Use slaveof to make a Redis instance a copy of
# another Redis server. A few things to understand ASAP about Redis replicat
#
# 1) Redis replication is asynchronous, but you can configure a master to
# stop accepting writes if it appears to be not connected with at least
# a given number of slaves.
# 2) Redis slaves are able to perform a partial resynchronization with the
# master if the replication link is lost for a relatively small amount of
# time. You may want to configure the replication backlog size (see the n
# sections of this file) with a sensible value depending on your needs.
# 3) Replication is automatic and does not need user intervention. After a
# network partition slaves automatically try to reconnect to masters
# and resynchronize with them.
#
# slaveof <masterip> <masterport>
slaveof 10.211.55.12 6379
# If the master is password protected (using the "requirepass" configuration
# directive below) it is possible to tell the slave to authenticate before
# starting the replication synchronization process, otherwise the master will
# refuse the slave request.
#
# masterauth <password>
masterauth lx
#
# When a slave loses its connection with the master, or when the replication
# is still in progress, the slave can act in two different ways:
#
# 1) if slave-serve-stale-data is set to 'yes' (the default) the slave will
# still reply to client requests, possibly with out of date data, or the
# data set may just be empty if this is the first synchronization.
#
# 2) if slave-serve-stale-data is set to 'no' the slave will reply with
# an error "SYNC with master in progress" to all the kind of commands
# but to INFO and SLAVEOF.
#
slave-serve-stale-data yes
```

2、启动从库

```
/opt/work/redisslave1/bin/redis-server /opt/work/redisslave1/bin/redis.conf
```

```
[root@CentOS6 ~]# /opt/work/redisslave1/bin/redis-server /opt/work/redisslave1/
bin/redis.conf
[root@CentOS6 ~]# ps aux|grep redis
root      11796  0.0  0.7 133536  7612 ?        Ssl  13:48   0:00 /opt/work/redis
/bin/redis-server *:6379
root      11818  0.1  0.7 133536  7572 ?        Ssl  13:55   0:00 /opt/work/redis
slave1/bin/redis-server *:6380
root      11822  0.0  0.0 103316   848 pts/1    S+   13:55   0:00 grep redis
[root@CentOS6 ~]#
```

3、分别使用2个客户端连接主库和从库

拷贝连接会话，并重命名slave，打开客户端连接从库

在主库和从库的连接客户端中输入：

info replication

```
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> set a 123
OK
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6379>
```

slave

文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 窗口(W) 帮助(H)

输入主机 <Alt+R>

slave x

```
[root@CentOS6 ~]# /opt/work/redis/bin/redis-cli -p 6380
127.0.0.1:6380> auth lx
OK
127.0.0.1:6380> get a
(nil)
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:10.211.55.12
master_port:6379
master_link_status:down
master_last_io_seconds_ago:-1
master_sync_in_progress:0
slave_repl_offset:1
master_link_down_since_seconds:1510207303
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6380>
```

4、测试数据的主从复制

```
127.0.0.1:6379> set a 111
OK
127.0.0.1:6379> set msg 'hello'
OK
127.0.0.1:6379> keys *
1) "msg"
2) "a"
127.0.0.1:6379>

slave
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 窗口(W) 帮助(H)
输入主机 <Alt+R>
✓ slave x
127.0.0.1:6380> get a
"111"
127.0.0.1:6380> get msg
"hello"
127.0.0.1:6380> keys *
1) "msg"
2) "a"
127.0.0.1:6380> █
```

如果主库宕机，那么

如果主库宕机，那么就无法再继续写入数据

4.2 哨兵模式

4.2.1 概念

Redis-Sentinel（哨兵模式）是官方推荐的高可用解决方案，当redis在做master-slave的高可用方案时，假如master宕机了，redis本身（以及其很多客户端）都没有实现自动进行主备切换，而redis-sentinel本身也是独立运行的进程，可以部署在其他与redis集群可通讯的机器中监控redis集群。

有了主从复制的实现之后，我们如果想从服务器进行监控，那么在redis2.6以后提供了一个“哨兵”机制，并在2.8版本以后功能稳定起来。

哨兵：顾名思义，就是监控Redis系统的运行状况

哨兵模式的特点

- 1、不时地监控redis是否按照预期良好地运行；
- 2、如果发现某个redis节点运行出现状况，能够通知另外一个进程(例如它的客户端)；
- 3、能够进行自动切换。当一个master节点不可用时，能够选举出master的多个slave(如果有超过一个slave的话)中的一个来作为新的master,其它的slave节点会将它所追随的master的地址改为被提升为master的slave的新地址。
- 4、哨兵为客户端提供服务发现，客户端链接哨兵，哨兵提供当前master的地址然后提供服务，如果出现切换，也就是master挂了，哨兵会提供客户端一个新地址。

哨兵 (sentinel) 本身也是支持集群的

很显然, 单个哨兵会存在自己挂掉而无法监控整个集群的问题, 所以哨兵也是支持集群的, 通常用三台哨兵机器来监控一组redis集群。

4.2.2 配置

配置哨兵模式主要是防止主库宕机, 所以本文档就在从库中进行配置哨兵模式

拷贝配置文件, 并修改

命令:

```
cp /opt/work/redis-3.2.11/sentinel.conf /opt/work/redisslave1/bin/    拷贝哨兵的配置
vim /opt/work/redisslave1/bin/sentinel.conf    修改哨兵模式的配置文件
```

配置解释

dir: 日志路径, 根据自己的要求保存。

sentinel monitor mymaster 10.0.31.144 6379 1 解释: #名称 #ip #端口号 # 投票选举次数 (即有多少篇就被选举为主, 这里节点少, 就配置为1)。

sentinel down-after-millisecond mymaster 5000 解释: 哨兵程序多久去监控一次集群, #默认30s 检查一次, 这里配置超时30000毫秒为宕机。

sentinel parallel-syncs mymaster 1 解释: 有多少个从节点

sentinel failover-timeout mymaster 600000 解释: 若哨兵在该配置值内未完成failover操作 (即发生故障时, m/s切换), 本次failover失败

sentinel parallel-syncs mymaster 1 解释: 有多少个从节点

```
# slave is promoted to master.
#
# Note: master name should not include special characters or spaces.
# The valid charset is A-z 0-9 and the three characters ".-_"
sentinel monitor mymaster 10.211.55.12 6379 1

# sentinel auth-pass <master-name> <password>
#
# Set the password to use to authenticate with the master and slaves.
# Useful if there is a password set in the Redis instances to monitor.
#
# Note that the master password is also used for slaves, so it is not
# possible to set a different password in masters and slaves instances
# if you want to be able to monitor these instances with Sentinel.
#
# However you can have Redis instances without the authentication enabled
# mixed with Redis instances requiring the authentication (as long as the
# password set is the same for all the instances requiring the password) as
# the AUTH command will have no effect in Redis instances with authentication
# switched off.
#
# Example:
#
# sentinel auth-pass mymaster MySUPER--secret-0123passw0rd

# sentinel down-after-milliseconds <master-name> <milliseconds>
#
# Number of milliseconds the master (or any attached slave or sentinel) should
# be unreachable (as in, not acceptable reply to PING, continuously, for the
# specified period) in order to consider it in S_DOWN state (Subjectively
# Down).
#
# Default is 30 seconds.
sentinel down-after-milliseconds mymaster 30000

# sentinel parallel-syncs <master-name> <numslaves>
#
# How many slaves we can reconfigure to point to the new slave simultaneously
```

```
# during the failover. Use a low number if you use the slaves to serve query
# to avoid that all the slaves will be unreachable at about the same
# time while performing the synchronization with the master.
sentinel parallel-syncs mymaster 1

# sentinel failover-timeout <master-name> <milliseconds>
#
# Specifies the failover timeout in milliseconds. It is used in many ways:
#
# - The time needed to re-start a failover after a previous failover was
#   already tried against the same master by a given Sentinel, is two
#   times the failover timeout.
#
# - The time needed for a slave replicating to a wrong master according
#   to a Sentinel current configuration, to be forced to replicate
#   with the right master, is exactly the failover timeout (counting since
#   the moment a Sentinel detected the misconfiguration).
#
# - The time needed to cancel a failover that is already in progress but
#   did not produced any configuration change (SLAVEOF NO ONE yet not
#   acknowledged by the promoted slave).
#
# - The maximum time a failover in progress waits for all the slaves to be
#   reconfigured as slaves of the new master. However even after this time
#   the slaves will be reconfigured by the Sentinels anyway, but not with
#   the exact parallel-syncs progression as specified.
#
# Default is 3 minutes.
sentinel failover-timeout mymaster 60000

# SCRIPTS EXECUTION
#
# sentinel notification-script and sentinel reconfig-script are used in order
```

4.2.3 启动

```
/opt/work/redisslave1/bin/redis-server /opt/work/redisslave1/bin/sentinel.conf --
sentinel
启动哨兵模式
/opt/work/redis/bin/redis-cli -h 10.211.55.12 -p 26379 info sentinel
```

```

slave x
[root@CentOS6 bin]# /opt/work/redisslave1/bin/redis-server /opt/work/redisslave1/bin/sentinel.conf --sentinel
12108:X 09 Nov 14:47:02.489 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.2.11 (00000000/0) 64 bit

Running in sentinel mode
Port: 26379
PID: 12108

http://redis.io

12108:X 09 Nov 14:47:02.490 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
12108:X 09 Nov 14:47:02.491 # Sentinel ID is eb351b2afc813db55304ece91a74501a0b29d671
12108:X 09 Nov 14:47:02.491 # +monitor master mymaster 10.211.55.12 6379 quorum 1
12108:X 09 Nov 14:47:32.538 # +sdown master mymaster 10.211.55.12 6379
12108:X 09 Nov 14:47:32.538 # +odown master mymaster 10.211.55.12 6379 #quorum 1/1
12108:X 09 Nov 14:47:32.538 # +new-epoch 1
12108:X 09 Nov 14:47:32.538 # +try-failover master mymaster 10.211.55.12 6379
12108:X 09 Nov 14:47:32.551 # +vote-for-leader eb351b2afc813db55304ece91a74501a0b29d671 1
12108:X 09 Nov 14:47:32.551 # +elected-leader master mymaster 10.211.55.12 6379
12108:X 09 Nov 14:47:32.551 # +failover-state-select-slave master mymaster 10.211.55.12 6379
12108:X 09 Nov 14:47:32.635 # -failover-abort-no-good-slave master mymaster 10.211.55.12 6379
12108:X 09 Nov 14:47:32.701 # Next failover delay: I will not start a failover before Thu Nov 9 14:49:33 2017

xingpenghui x
[root@CentOS6 /]# /opt/work/redis/bin/redis-cli -p 26379 info sentinel
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=odown,address=10.211.55.12:6379,slaves=0,sentinels=1
[root@CentOS6 /]#

```

4.3 持久化机制

Redis持久化存储支持两种方式：RDB和AOF。RDB一定时间取存储文件，AOF默认每秒去存储历史命令，官方建议两种方式同时使用
没有持久化的redis和memcache一样，相当于一个纯内存的数据库

Redis是支持持久化的内存数据库，也就是说redis需要经常将内存中的数据同步到硬盘来保证持久化。

4.3.1 RDB

RDB(snapshotting快照)是将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。

优点：使用单独子进程来进行持久化，主进程不会进行任何IO操作，保证了redis的高性能

缺点：RDB是间隔一段时间进行持久化，如果持久化之间redis发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候

默认方式，将内存中以快照的方式写入到二进制文件中，默认为dump.rdb，可以通过配置设置自动做快照持久化的方式。我们可以配置redis在n秒内如果m个key修改，就自动做快照

```
vim /opt/work/redis/conf/redis.conf 修改配置文件
```

RDB默认开启，redis.conf中的具体配置参数如下：

```
save 900 1 #900秒内，超过1个key被修改，则发起快照保存
```

```
save 300 10 #300秒内，超过10个key被修改，则发起快照保存
```

```
save 60 10000 #60秒内，超过10000个key被修改，则发起快照保存
```

```
dbfilename dump.rdb 持久化数据存储在本地的文件
```

```
dir ./ 持久化数据存储在本地的路径，如果是在/redis/redis-3.0.6/src下启动的redis-cli，则数据会存储在当前src目录下
```

```
save 900 1
save 300 10
save 60 10000

# By default Redis will stop accepting writes if RDB snapshots are enabled
# (at least one save point) and the latest background save failed.
# This will make the user aware (in a hard way) that data is not persisting
# on disk properly, otherwise chances are that no one will notice and some
# disaster will happen.
#
# If the background saving process will start working again Redis will
# automatically allow writes again.
#
# However if you have setup your proper monitoring of the Redis server
# and persistence, you may want to disable this feature so that Redis will
# continue to work as usual even if there are problems with disk,
# permissions, and so forth.
stop-writes-on-bgsave-error yes

# Compress string objects using LZF when dump .rdb databases?
# For default that's set to 'yes' as it's almost always a win.
# If you want to save some CPU in the saving child set it to 'no' but
# the dataset will likely be bigger if you have compressible values or keys.
rdbcompression yes

# Since version 5 of RDB a CRC64 checksum is placed at the end of the file.
# This makes the format more resistant to corruption but there is a performance
# hit to pay (around 10%) when saving and loading RDB files, so you can disable
# it
# for maximum performances.
#
# RDB files created with checksum disabled have a checksum of zero that will
# tell the loading code to skip the check.
rdbchecksum yes

# The filename where to dump the DB
dbfilename dump.rdb

# The working directory.
#
# The DB will be written inside this directory, with the filename specified
# above using the 'dbfilename' configuration directive.
#
# The Append Only File will also be created inside this directory.
#
# Note that you must specify a directory here, not a file name.
dir ./
```

— INSERT — 247,7 19%

持久化过程：

当满足save的条件时，比如更改了1个key，900s后会将数据写入临时文件，持久化完成后将临时文件替换旧的dump.rdb。（存储数据的节点是到触发时间时的节点）

使用RDB恢复数据：

自动的持久化数据存储到dump.rdb后。实际只要重启redis服务即可完成（启动redis的server时会从dump.rdb中先同步数据）

使用命令进行持久化save存储：

```
./redis-cli -h ip -p port save
```

```
./redis-cli -h ip -p port bgsave
```

一个是在前台进行存储，一个是在后台进行存储。

```
[root@CentOS6 bin]# /opt/work/redis/bin/redis-cli -a lx
127.0.0.1:6379> keys *
1) "msg"
2) "c"
3) "a"
127.0.0.1:6379> set d 3333
OK
127.0.0.1:6379> keys *
1) "d"
2) "msg"
3) "c"
4) "a"
127.0.0.1:6379> quit
[root@CentOS6 bin]# /opt/work/redis/bin/redis-cli -a lx save
OK
```

4.3.2 AOF

AOF(append-only file)是将执行过的指令记录下来，数据恢复时按照从前到后的顺序再将指令执行一遍，实现数据恢复

优点：可以保持更高的数据完整性，如果设置追加file的时间是1s，如果redis发生故障，最多会丢失1s的数据；且如果日志写入不完整支持redis-check-aof来进行日志修复；AOF文件没被rewrite之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的flushall）。

缺点：AOF文件比RDB文件大，且恢复速度慢。

类似于mysql日志，由于快照方式是在一定时间间隔做一次，所以可能发生redis意外宕机的情况就会丢失最后一次快照后的所有被修改的数据，aof比快照方式有更好的持久化型，是由于redis在使用aof时，redis会将每一个收到的写命令都通过write函数追加到命令中，在redis重新启动时会重新执行文件中保存的写命令在内存中重建这个数据库的内容，这个文件在redis/bin目录下，appendonly.aof。aof不是立即写到硬盘上，可以通过配置文件修改强制写到硬盘中。

修改配置

appendonly yes //启动aof持久化，持久化有三种方式：

#appendfsync always //收到写命令就立即写入到磁盘，效率最慢，但是保证完整的持久化（最常用）

#appendfsync everysec //每秒写一次硬盘，在性能和持久化方面做了很好的这种

#appendfsync no //完全依赖os，性能最好，持久化没保证。

```
# Save running correctly:
#
# AOF and RDB persistence can be enabled at the same time without problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the file
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly yes

# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"

# The fsync() call tells the Operating System to actually write data on disk
# instead of waiting for more data in the output buffer. Some OS will really fl
sh
# data on disk, some other OS will just try to do it ASAP.
#
# Redis supports three different modes:
#
# no: don't fsync, just let the OS flush the data when it wants. Faster.
# always: fsync after every write to the append only log. Slow, Safest.
# everysec: fsync only one time every second. Compromise.
#
# The default is "everysec", as that's usually the right compromise between
# speed and data safety. It's up to you to understand if you can relax this to
# "no" that will let the operating system flush the output buffer when
# it wants, for better performances (but if you can live with the idea of
# some data loss consider the default persistence mode that's snapshotting),
# or on the contrary, use "always" that's very slow but a bit safer than
# everysec.
#
# More details please check the following article:
# http://antirez.com/post/redis-persistence-demystified.html
#
# If unsure, use "everysec".

# appendfsync always
appendfsync everysec
# appendfsync no

# When the AOF fsync policy is set to always or everysec, and a background
```

重启redis发现bin/目录下多了一个appendonly.aof

提示：开启aof后之前的rdb模式就失效了，且之前的数据会被清空。