



北京动力节点教育科技有限公司

动力节点课程讲义

DONGLIJIEDIANKECHENGJIANGYI

www.bjpowernode.com

第1章 NoSQL

1.1 什么是 NoSQL

NoSQL = Not Only SQL(不仅仅是 SQL) , 也解释为 non-relational(非关系型数据库)。在 NoSQL 数据库中数据之间是无联系的, 无关系的。数据的结构是松散的, 可变的。

1.2 为什么使用 NoSQL

关系型数据库的瓶颈:

1) 无法应对每秒上万次的读写请求, 无法处理大量集中的高并发操作。关系型数据库的是 IO 密集的应用。硬盘 IO 也变为性能瓶颈

2) 表中存储记录数量有限, 横向可扩展能力有限, 一张表最大二百多列。纵向数据可承受能力也是有限的, 一张表的数据到达百万级, 读写的速度就会逐渐的下降。面对海量数据, 必须使用主从复制, 分库分表。这样的系统架构是难以维护的。

大数据查询 SQL 效率极低, 数据量到达一定程度时, 查询时间会呈指数级别增长

3) 无法简单地通过增加硬件、服务节点来提高系统性能。数据整个存储在一个数据库中的。多个服务器没有很好的解决办法, 来复制这些数据。

4) 关系型数据库大多是收费的, 对硬件的要求较高。软件和硬件的成本花费比重较大。

1.3 NoSQL 的优势

(1) 大数据量, 高性能

NoSQL 数据库都具有非常高的读写性能, 尤其在大数据量下, 同样表现优秀。这得益于它的无关系性, 数据库的结构简单。关系型数据库(例如 MySQL)使用查询缓存。这种查询缓存在更新数据后, 缓存就是失效了。在频繁的数据读写交互应用中。缓存的性能不高。NoSQL 的缓存性能要高的多。

(2) 灵活的数据模型

NoSQL 无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。尤其在快速变化的市场环境中，用户的需求总是在不断变化的。

(3) 高可用

NoSQL 在不太影响性能的情况，就可以方便的实现高可用的架构。

NoSQL 能很好的解决关系型数据库扩展性差的问题。弥补了关系数据（比如 MySQL）在某些方面的不足，在某些方面能极大的节省开发成本和维护成本。

MySQL 和 NoSQL 都有各自的特点和使用的应用场景，两者结合使用。让关系数据库关注在关系上，NoSQL 关注在存储上。

(4) 低成本

这是大多数分布式数据库共有的特点，因为主要都是开源软件，没有昂贵的 License 成本

1.4 NoSQL 的劣势

- (1) 无关系，数据之间是无联系的。
- (2) 不支持标准的 SQL,没有公认的 NoSQL 标准
- (3) 没有关系型数据库的约束，大多数也没有索引的概念
- (4) 没有事务，不能依靠事务实现 ACID.
- (5) 没有丰富的数据类型（数值，日期，字符，二进制，大文本等）

第2章 Redis 安装和使用

Redis 是当今非常流行的基于 KV 结构的作为 Cache 使用的 NoSQL 数据库

2.1 Redis 介绍

Remote Dictionary Server(Redis) 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的 Key-Value 数据库。Key 字符类型，其值 (value) 可以是 字符串(String), 哈希(Map), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型, 每种数据类型有自己的专属命令。所以它通常也被称为数据结构服务器。

Redis 的作者是 Salvatore Sanfilippo, 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 (Pivotal 是 Spring 框架的开发团队), Salvatore Sanfilippo 被称为 Redis 之父。



Redis 之父 (Salvatore Sanfilippo)

官网: <https://redis.io/>

中文: <http://www.redis.cn/>

Redis的历史 :

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便开始对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，而是希望让更多的人使用它，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。2012年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、知乎，国外如GitHub、Stack Overflow、Flickr、暴雪和Instagram，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也分别于同年的3月和5月加入VMware，全职开发Redis。

Redis的代码托管在GitHub上<https://github.com/antirez/redis>，开发十分活跃，代码量只有3万多行。

2.2 Window 上安装 Redis

Windows 版本的 Redis 是 Microsoft 的开源部门提供的 Redis。这个版本的 Redis 适合开发人员学习使用，生产环境中使用 Linux 系统上的 Redis

(1) 下载

官网：<https://redis.io/>

windows 版本：<https://github.com/MSOpenTech/redis/releases>

3.2.100

 enricogior released this Jul 1, 2016 · 1208 commits to 3.0 since this release





This is the first release of Redis on Windows 3.2.

This release is based on antirez/redis/3.2.1 plus some Windows specific fixes. It has passed all the standard tests but it hasn't been tested in a production environment.

Therefore, **before considering using this release in production, make sure to test it thoroughly in your own test environment.**

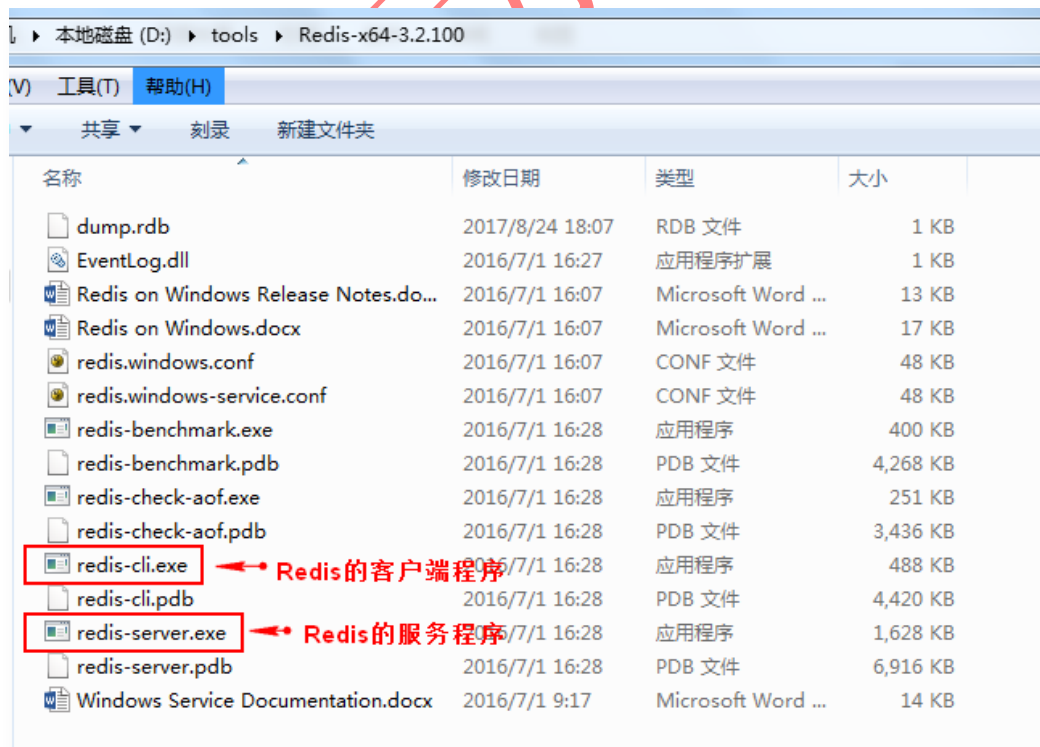
See the [release notes](#) for details.

Downloads

 Redis-x64-3.2.100.msi	5.8 MB
 Redis-x64-3.2.100.zip	4.98 MB
 Source code (zip)	
 Source code (tar.gz)	

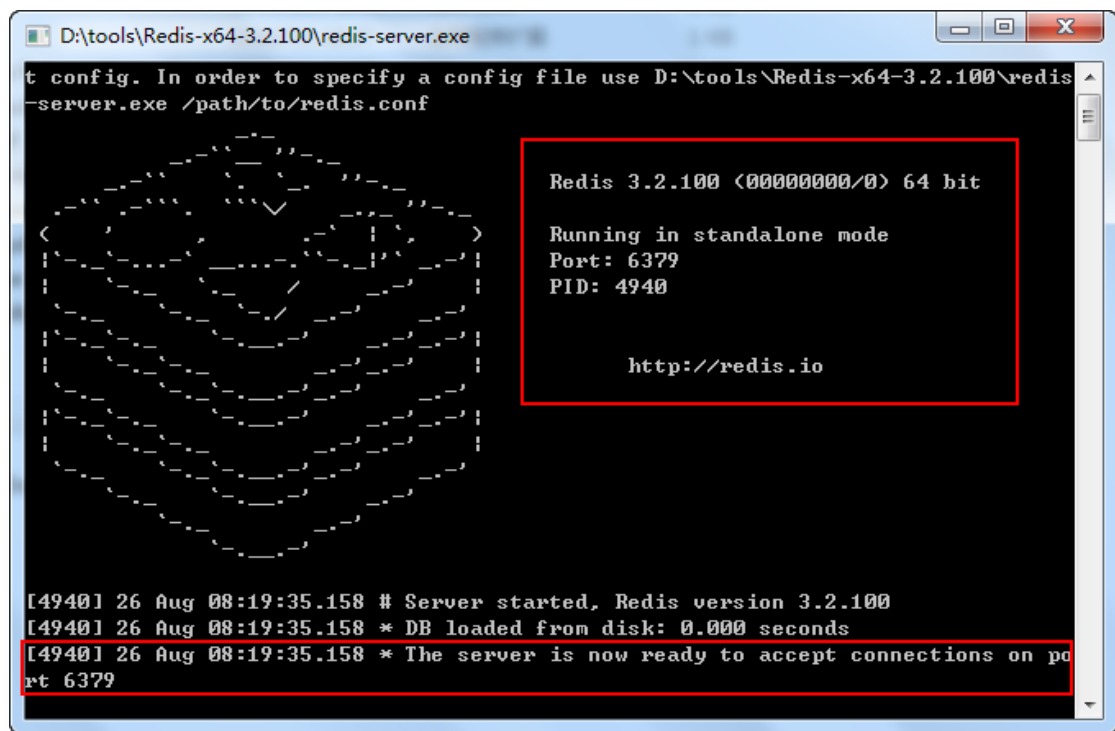
(2) 安装

下载的 Redis-x64-3.2.100.zip 解压后，放到某个目录（例如 d:\tools\），即可使用。
目录结构：



(3) 启动

A、Windows7 系统双击 redis-server.exe 启动 Redis

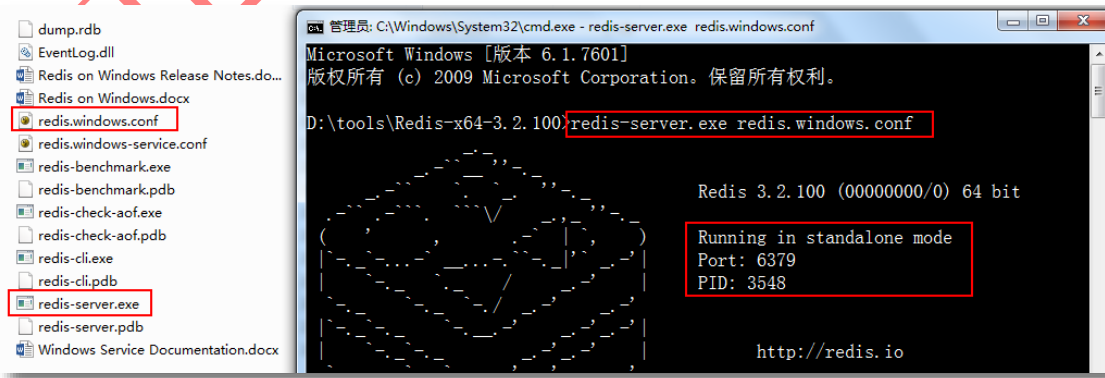


B、Windows 10 系统

有的机器双击 redis-server.exe 执行失败，找不到配置文件，可以采用以下执行方式：
在命令行（cmd）中按如下方式执行：

D:\tools\Redis-x64-3.2.100>redis-server.exe redis.windows.conf

如图：



(4) 关闭

按 ctrl+c 退出 Redis 服务程序。

2.3 Linux 上安装 Redis

(1) 下载

wget http://219.238.7.66/files/502600000A29C8D5/download.redis.io/releases/redis-3.2.9.tar.gz

(2) 安装

A、上传 redis-3.2.9.tar 到 linux 系统。使用 Xftp 工具

```
[root@localhost ~]# ll redis-3.2.9.tar.gz  
-rw-r--r--. 1 root root 1547695 Aug 21 20:36 redis-3.2.9.tar.gz  
[root@localhost ~]#
```

B、解压 redis-3.2.9.tar 到 usr/local 目录

```
[root@localhost ~]#  
[root@localhost ~]# tar -zxvf redis-3.2.9.tar.gz -C /usr/local/  
redis-3.2.9/  
redis-3.2.9/.gitignore  
redis-3.2.9/00-RELEASENOTES  
redis-3.2.9/BUGS  
redis-3.2.9/CONTRIBUTING  
redis-3.2.9/COPYING
```


C、查看解压后的文件

```
[root@localhost ~]# cd /usr/local/
[root@localhost local]# ll
total 4
drwxr-xr-x.  9 root  root   160 Aug 16 11:16 apache-tomcat-9.0.0.M26
drwxr-xr-x.  2 root  root    6 Nov  5 2016 bin
drwxr-xr-x.  2 root  root    6 Nov  5 2016 etc
drwxr-xr-x.  2 root  root    6 Nov  5 2016 games
drwxr-xr-x.  2 root  root    6 Nov  5 2016 include
drwxr-xr-x.  8  10  143 255 Dec 13 2016 jdk1.8.0_121
drwxr-xr-x.  2 root  root    6 Nov  5 2016 lib
drwxr-xr-x.  2 root  root    6 Nov  5 2016 lib64
drwxr-xr-x.  2 root  root    6 Nov  5 2016 libexec
drwxr-xr-x. 10 mysql mysql  141 Aug 16 14:57 mysql5.7.18
drwxrwxr-x.  6 root  root 4096 May 17 23:39 redis-3.2.9
drwxr-xr-x.  2 root  root    6 Nov  5 2016 sbin
drwxr-xr-x.  5 root  root   49 Aug  5 23:11 share
drwxr-xr-x.  2 root  root    6 Nov  5 2016 src
[root@localhost local]#
```

D、编译 Redis 文件， Redis 是使用 c 语言编写的。 会使用 gcc 编译器。

在解压后的 Redis 目录下执行 （cd /usr/local/redis-3.2.9） make 命令。

注意事项：

1) make 命令执行过程中可能报错，根据控制台输出的错误信息进行解决

2) 错误一： gcc 命令找不到，是由于没有安装 gcc 导致

解决方式：安装 gcc 编译器后在执行 make 命令

什么是 gcc？

gcc 是 GNU compiler collection 的缩写，它是 Linux 下一个编译器集合(相当于 javac)，是 c 或 c++程序的编译器。

怎么安装gcc？

使用yum进行安装gcc 。执行命令： yum -y install gcc

3) 错误二： error: jemalloc/jemalloc.h: No such file or directory

解决方式执行 make MALLOC=libc

开始执行 make

```
[root@localhost redis-3.2.9]# pwd
/usr/local/redis-3.2.9
[root@localhost redis-3.2.9]# make
cd src && make all
```

出现错误：

```
ngs -g -ggdb net.c
make[3]: gcc: Command not found ①
make[3]: *** [net.o] Error 127
make[3]: Leaving directory `/usr/local/redis-3.2.9/deps/hiredis'
make[2]: *** [hiredis] Error 2
make[2]: Leaving directory `/usr/local/redis-3.2.9/deps'
make[1]: [persist-settings] Error 2 (ignored)
cc adlist.o
/bin/sh: cc: command not found ②
make[1]: *** [adlist.o] Error 127
make[1]: Leaving directory `/usr/local/redis-3.2.9/src'
make: *** [all] Error 2
[root@localhost redis-3.2.9]#
```

没有 gcc, cc 编译器，解决安装 gcc

使用 yum -y install gcc

```
[root@localhost redis-3.2.9]# yum -y install gcc
Loaded plugins: fastestmirror, langpacks
base                                     | 3.6 kB      00:00
extras                                 | 3.4 kB      00:00
updates                               | 3.4 kB      00:02
extras/7/x86_64/primary_db            | 191 kB      00:00
Loading mirror speeds from cached hostfile
* base: mirrors.btte.net
```

重新再编译 make。注意：安装完 gcc 之后，再执行 make，先执行 make distclean 清理一下上次 make 后产生的文件。

先执行 make distclean

```
[root@localhost redis-3.2.9]#
[root@localhost redis-3.2.9]# make distclean
cd src && make distclean
make[1]: Entering directory `/usr/local/redis-3.2.9/src'
```

在执行 make

```
[root@localhost redis-3.2.9]# pwd
/usr/local/redis-3.2.9
[root@localhost redis-3.2.9]# make
cd src && make all
```

执行 make 成功的标志

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

```
make[1]: Leaving directory `/usr/local/redis-3.2.9/src'
[root@localhost redis-3.2.9]#
```

注意：在make执行之后再执行 make install，该操作则将 src下的许多可执行文件复制到 /usr/local/bin 目录下，这样做可以在任意目录执行redis的软件的命令（例如启动，停止，客户端连接服务器等）， make install 可以不用执行，看个人习惯。

查看make编译结果，cd src目录

```
[root@localhost redis-3.2.9]# pwd
/usr/local/redis-3.2.9
[root@localhost redis-3.2.9]# ls
00-RELEASENOTES  COPYING  Makefile  redis.conf  runtest-sentinel  tests
BUGS             deps     MANIFESTO  runtest     sentinel.conf     utils
CONTRIBUTING    INSTALL  README.md  runtest-cluster  src
```

cd src 在执行 ls

```
[root@localhost ~]# cd /usr/local/redis-3.2.9/src
[root@localhost src]# ls
intset.c  Makefile.dep  pqsort.h  redisassert.h  redis-sentinel
intset.h  memtest.c     pqsort.o  redis-benchmark  redis-server
intset.o  memtest.o     pubsub.c  redis-benchmark.c  redis-trib.rb
latency.c mkreleasendr.sh  pubsub.o  redis-benchmark.o  release.c
latency.h multi.c        quicklist.c  redis-check-aof  release.h
latency.o multi.o        quicklist.h  redis-check-aof.c  release.o
lzf_c.c  networking.c  quicklist.o  redis-check-aof.o  replication.c
lzf_c.o  networking.o  rand.c       redis-check-rdb  replication.o
lzf_d.c  notify.c     rand.h       redis-check-rdb.c  rio.c
lzf_d.o  notify.o     rand.o       redis-check-rdb.o  rio.h
lzf.h    object.c     rdb.c        redis-cli         rio.o
lzfp.h   object.o     rdb.h        redis-cli.c       scripting.c
Makefile pqsort.c     rdb.o        redis-cli.o       scripting.o
[root@localhost src]#
```

(3) 启动 Redis

启动方式：

- ① 前台启动 ./redis-server
- ② 后台启动 ./redis-server &

第①种 前台启动

启动 Redis 的服务器端：切换到 src 目录下执行 redis-server 程序

```
[root@localhost src]# pwd
/usr/local/redis-3.2.9/src
[root@localhost src]# ./redis-server
8185:C 21 Aug 22:29:03.824 # Warning: no config file specified, using the default
/to/redis.conf
8185:M 21 Aug 22:29:03.825 * Increased maximum number of open files to 10032

Redis 3.2.9 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 8185

http://redis.io
```

redis 应用以前台的方式启动，不能退出当前窗口， 退出窗口，应用终止。

在其他窗口查看 redis 启动的进程

```
[root@localhost ~]#
[root@localhost ~]# ps -ef | grep redis
root      8185    7610    0 22:29 pts/2    00:00:00 ./redis-server *:6379
root      8290    8243    0 22:33 pts/3    00:00:00 grep --color=auto redis
[root@localhost ~]#
```

第②种 后台启动

src目录下执行 `./redis-server &` 此时关闭窗口，查看redis进程，依然存在。

```
[root@localhost src]#
[root@localhost src]# ./redis-server &
[1] 8320
[root@localhost src]# 8320:C 21 Aug 22:36:15.176 # Warning: no config file specified
e ./redis-server /path/to/redis.conf
8320:M 21 Aug 22:36:15.177 * Increased maximum number of open files to 10032

Redis 3.2.9 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 8320
```

查看redis进程

```
[root@localhost ~]# ps -ef | grep redis
root      8636    8243    0 22:59 pts/3    00:00:00 /usr/local/redis-3.2.9/src/redis-server *:6379
root      8686    8243    0 23:02 pts/3    00:00:00 grep --color=auto redis
[root@localhost ~]#
```

(4) 关闭 Redis

关闭方式:

- ① 使用 redis 客户端关闭， 向服务器发出关闭命令
切换到 redis-3.2.9/src/ 目录，执行 ./redis-cli shutdown
推荐使用这种方式， redis 先完成数据操作，然后再关闭。

例如:

```
[root@localhost src]# ps -ef | grep redis
root      8320    7610    0 22:36 pts/2    00:00:01 ./redis-server *:6379
root      8498    8243    0 22:49 pts/3    00:00:00 grep --color=auto redis
[root@localhost src]# pwd
/usr/local/redis-3.2.9/src
[root@localhost src]# ./redis-cli shutdown
[root@localhost src]#
[root@localhost src]# ps -ef | grep redis
root      8509    8243    0 22:49 pts/3    00:00:00 grep --color=auto redis
[root@localhost src]#
```

先查看存在redis进程，使用
redis-cli shutdown 关闭进程，
再查看进程已经没有了

- ② kill pid 或者 kill -9 pid

这种不会考虑当前应用是否有数据正在执行操作，直接就关闭应用。

先使用 ps -ef | grep redis 查出进程号， 在使用 kill pid

```
[root@localhost ~]# ps -ef | grep redis
root      8636    8243    0 22:59 pts/3    00:00:00 /usr/local/redis-3.2.9/src/redis-server *:6379
root      8686    8243    0 23:02 pts/3    00:00:00 grep --color=auto redis
[root@localhost ~]#
[root@localhost ~]# kill 8636
[root@localhost ~]# ps -ef | grep redis
root      8720    8243    0 23:04 pts/3    00:00:00 grep --color=auto redis
```

2.4 Redis 客户端

Redis 客户端是一个程序，通过网络连接到 Redis 服务器， 在客户端软件中使用 Redis 可以识别的命令，向 Redis 服务器发送命令， 告诉 Redis 想要做什么。Redis 把处理结果显示在客户端界面上。 通过 Redis 客户端和 Redis 服务器交互。

Redis 客户端发送命令，同时显示 Redis 服务器的处理结果在。

2.4.1 redis 命令行客户端:

redis-cli (Redis Command Line Interface) 是 Redis 自带的基于命令行的 Redis 客户端, 用于与服务端交互, 我们可以使用该客户端来执行 redis 的各种命令。
两种常用的连接方式:

A、直接连接 redis (默认 ip127.0.0.1, 端口 6379): ./redis-cli

在 redis 安装目录\src, 执行 ./redis-cli
此命令是连接本机 127.0.0.1 , 端口 6379 的 redis

B、指定 IP 和端口连接 redis: ./redis-cli -h 127.0.0.1 -p 6379

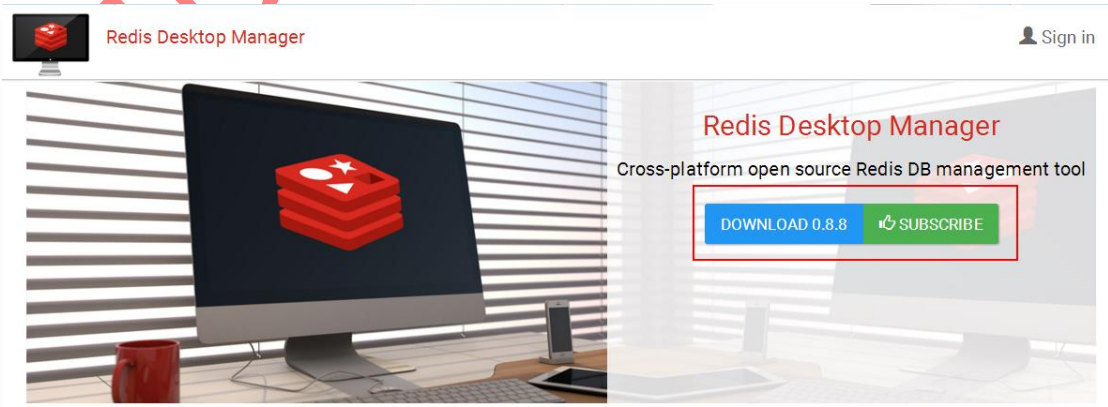
-h redis 主机 IP (可以指定任意的 redis 服务器)
-p 端口号 (不同的端口表示不同的 redis 应用)

在 redis 安装目录\src, 执行 ./redis-cli -h 127.0.0.1 -p 6379
例 1:

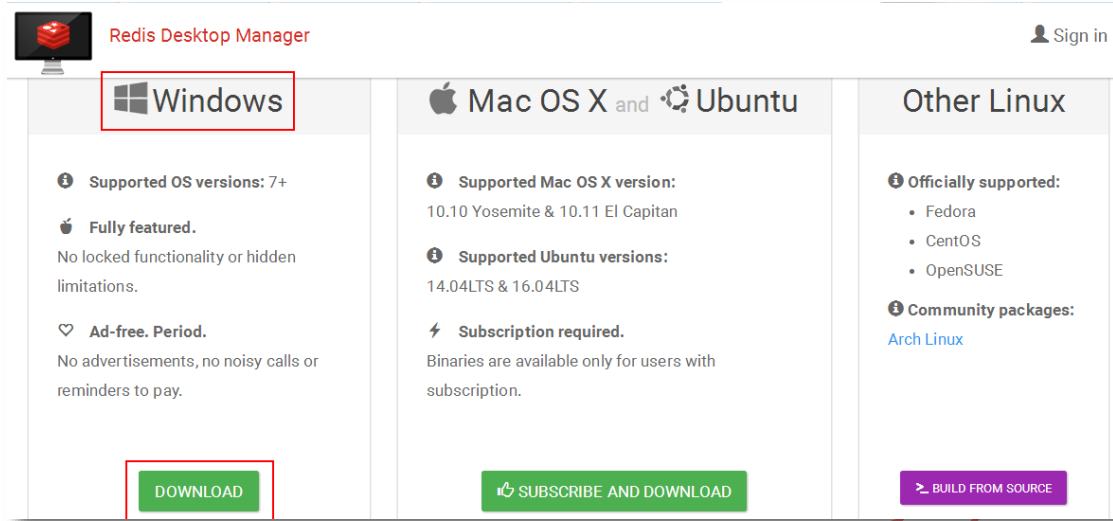
```
[root@localhost src]# ./redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> get k1
"v1"
```

2.4.2 redis 远程客户端

Redis Desktop Manager: C++ 编写, 响应迅速, 性能好。
官网地址: <https://redisdesktop.com/>
github: <https://github.com/uglide/RedisDesktopManager>
使用文档: <http://docs.redisdesktop.com/en/latest/>



点击 "DOWNLOAD"

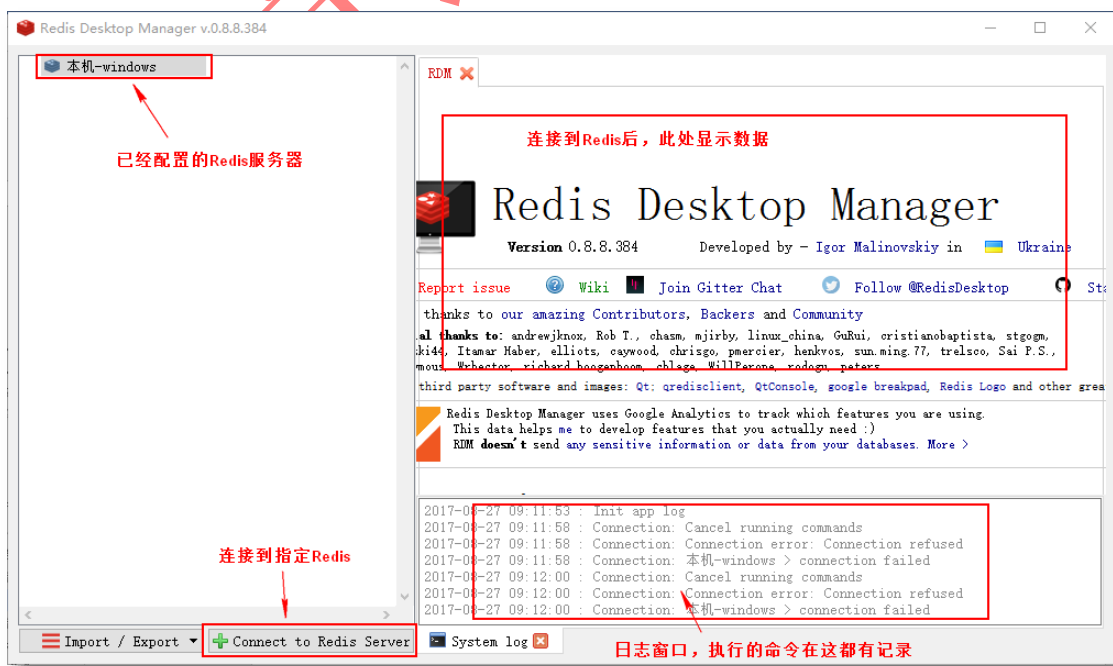


A、安装客户端软件

在 Windows 系统使用此工具，连接 Linux 上或 Windows 上的 Redis，双击此 exe 文件执行安装

redis-desktop-manager-0.8.8.384

安装后启动界面：



B、使用客户端连接 Linux 的 Redis

连接Linux的Redis之前需要修改Redis服务器的配置信息。Redis服务器有安全保护措施，默认只有本机（安装Redis的那台机器）能够访问。配置信息存放在Redis安装目录下的redis.conf文件。修改此文件的两个设置。

远程连接redis需要修改redis主目录下的redis.conf配置文件：

①、bind ip 绑定ip此行注释

②、protected-mode yes 保护模式改为 no

使用 vim 命令修改 redis.conf 文件，修改文件前备份此文件，执行 cp 命令

```
[root@localhost redis-3.2.9]#  
[root@localhost redis-3.2.9]# cp redis.conf bak_redis.conf
```

执行 vim redis.conf

```
# bind 127.0.0.1 ← #表示注释  
  
# Protected mode is a layer of security protection, in order to avoid that  
# Redis instances left open on the internet are accessed and exploited.  
#  
# When protected mode is on and if:  
#  
# 1) The server is not binding explicitly to a set of addresses using the  
#    bind directive.  
# 2) No password is configured.  
#  
# The server only accepts connections from clients connecting from the  
# IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain  
# sockets.  
#  
# By default protected mode is enabled. You should disable it only if  
# you are sure you want clients from other hosts to connect to Redis  
# even if no authentication is configured, nor a specific set of interfaces  
# are explicitly listed using the bind directive.  
protected-mode no ← 原来是yes 改为 no
```

C、使用 redis.conf 启动 Redis

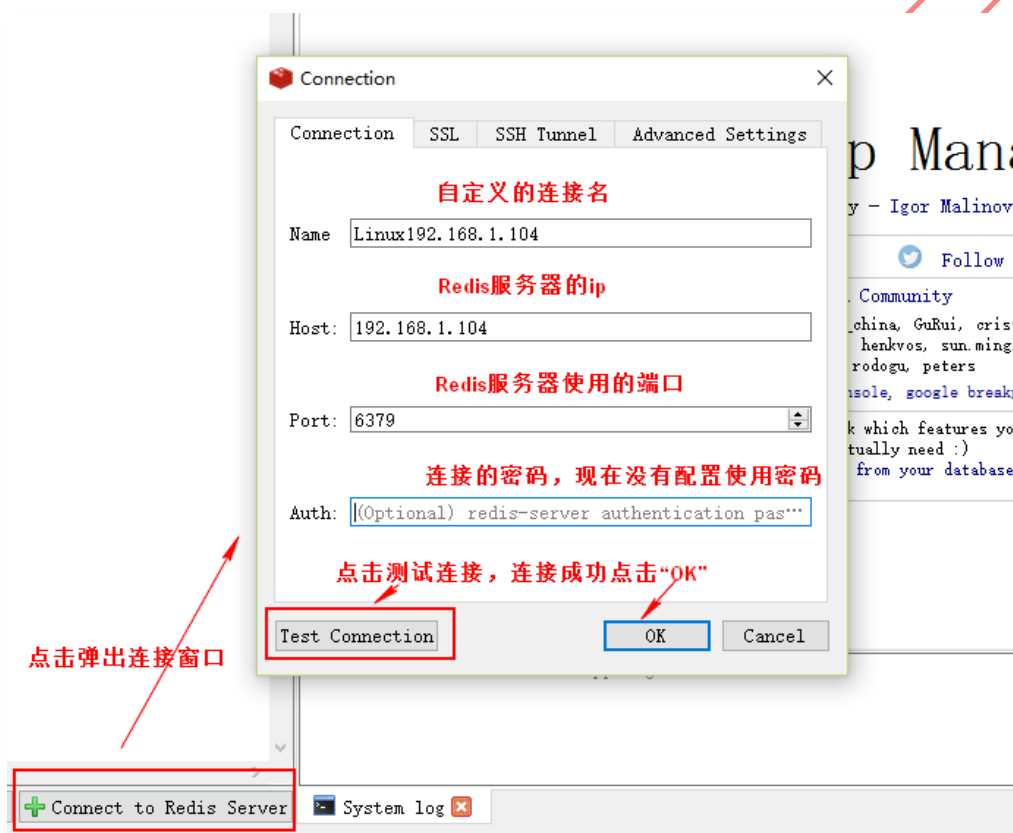
修改配置文件后，需要使用配置文件重新启动 Redis，默认不加载配置文件。先关闭已经启动的 Redis，使用以下命令启动 Redis 在 Redis 安装目录执行：

```
./redis-server ../redis.conf &
```

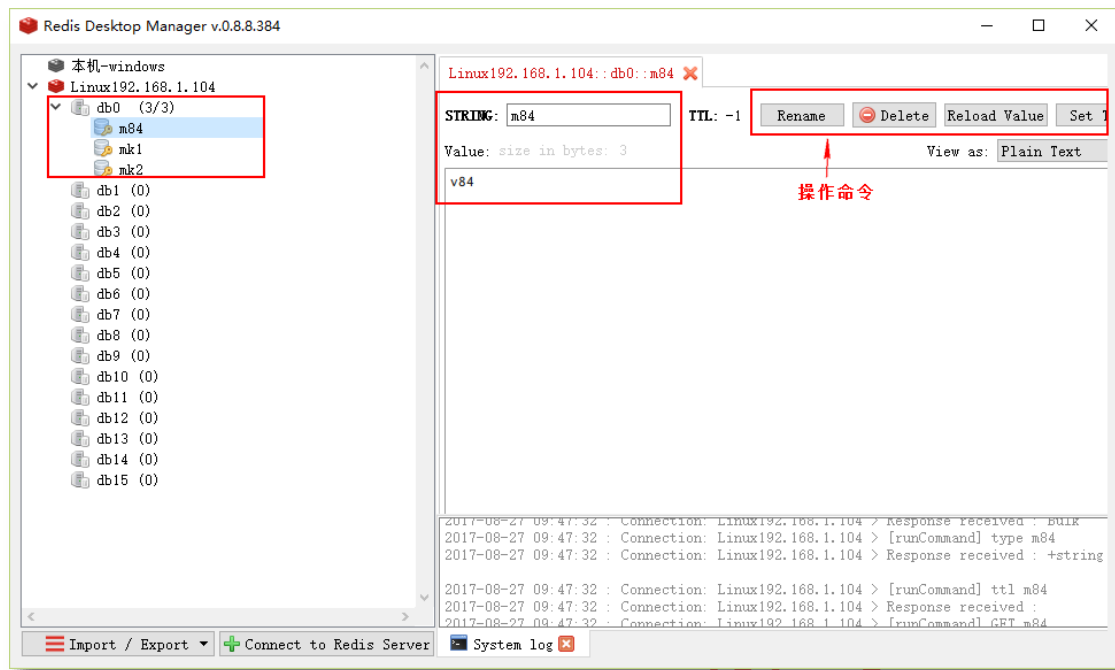
```
[root@localhost src]#  
[root@localhost src]# ./redis-server ../redis.conf &  
[1] 4132  
[root@localhost src]# 4132:M 27 Aug 09:36:10.755 * Increased maximum number of open files to 1000 (from 1024 to 1000)  
Redis 3.2.9 (00000000/0) 64 bit  
Running in standalone mode
```

D、配置 Redis Desktop Manamager(RDM)，连接 Redis

在 RDM 的主窗口，点击左下的“Connect to Redis Server”



连接成功后：



2.4.3 redis 编程客户端

A、Jedis

redis 的 Java 编程客户端，Redis 官方首选推荐使用 Jedis，jedis 是一个很小但很健全的 redis 的 java 客户端。通过 Jedis 可以像使用 Redis 命令行一样使用 Redis。

- jedis 完全兼容 redis 2.8.x and 3.x.x
- Jedis 源码: <https://github.com/xetorthio/jedis>
- api 文档: <http://xetorthio.github.io/jedis/>

B、redis 的其他编程语言客户端:

C、C++、C#、Erlang、Lua、Objective-C、Perl、PHP、Python、Ruby、Scala、Go 等 40 多种语言都有连接 redis 的编程客户端

2.5 Redis 基本操作命令

redis 默认为 16 个库 (在 redis.conf 文件可配置, 该文件很重要, 后续很多操作都是这个配置文件) redis 默认自动使用 0 号库

(1) 沟通命令，查看状态

redis > ping 返回 PONG

解释：输入 ping，redis 给我们返回 PONG，表示 redis 服务运行正常

```
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

(2) 查看当前数据库中 key 的数目：dbsize

语法：dbsize

作用：返回当前数据库的 key 的数量。

返回值：数字，key 的数量

例：先查索引 5 的 key 个数，再查 0 库的 key 个数

```
127.0.0.1:6379> select 5
OK
127.0.0.1:6379[5]> dbsize
(integer) 0
127.0.0.1:6379[5]> select 0
OK
127.0.0.1:6379> dbsize
(integer) 2
127.0.0.1:6379>
```

查询 5 索引库的key数量是 0，select 0 之后是查看 0 库的key的数量是 2，有 2 个key

(3) redis 默认使用 16 个库

Redis 默认使用 16 个库，从 0 到 15。对数据库个数的修改，在 redis.conf 文件中 databases 16

```
# Set the number of databases. The default database is DB 0, you can select
# a different one on a per-connection basis using SELECT <dbid> where
# dbid is a number between 0 and 'databases'-1
databases 16
```

数据库初始是16个，默认是第0个数据库

(4) 切换库命令：select db

使用其他数据库，命令是 select index

例 1：select 5

```
127.0.0.1:6379> select 5
OK
127.0.0.1:6379[5]> set k1 v1
OK
127.0.0.1:6379[5]> select 0
OK
127.0.0.1:6379>
```

使用第6个库（库从0开始）
库的索引值

（5）删除当前库的数据：flushdb

```
127.0.0.1:6379> flushdb
OK
```

删除当前库的数据

（6）redis 自带的客户端退出当前 redis 连接: exit 或 quit

```
127.0.0.1:6379> exit
```

2.6 Redis 的 Key 的操作命令

A、keys

语法：keys pattern

作用：查找所有符合模式 pattern 的 key。 pattern 可以使用通配符。

通配符：

- *：表示 0-多个字符，例如：keys * 查询所有的 key。
- ?：表示单个字符，例如：wo?d，匹配 word，wood

例 1：显示所有的 key

```
127.0.0.1:6379> keys *
1) "arch"
2) "list2"
3) "k2"
```

例 2：使用 * 表示 0 或多个字符

```
127.0.0.1:6379> set word word
OK
127.0.0.1:6379> set wood wood
OK
127.0.0.1:6379> keys wo*d
1) "wood"
2) "word"
```

例 3：使用 ? 表示单个字符

```
127.0.0.1:6379> keys *
1) "wood"
2) "word"
3) "woood"
127.0.0.1:6379> keys wo?d
1) "wood"
2) "word"
```

B、exists

语法：exists key [key...]

作用：判断 key 是否存在

返回值：整数，存在 key 返回 1，其他返回 0。使用多个 key，返回存在的 key 的数量。

例 1：检查指定 key 是否存在

```
127.0.0.1:6379> keys *
1) "word"
2) "wood"
3) "woood"
127.0.0.1:6379> exists wood
(integer) 1
127.0.0.1:6379> exists myword
(integer) 0
127.0.0.1:6379>
```

例 2：检查多个 key

```
127.0.0.1:6379> keys *
1) "word"
2) "wood"
3) "woood"
127.0.0.1:6379> exists word wood hello
(integer) 2
127.0.0.1:6379>
```

存在的key是2，不存在的key不计算在结果内。

C、expire

语法: expire key seconds

作用: 设置 key 的生存时间, 超过时间, key 自动删除。单位是秒。

返回值: 设置成功返回数字 1, 其他情况是 0。

例 1: 设置红灯的倒计时是 5 秒

```
127.0.0.1:6379> expire redlight 5
(integer) 0
127.0.0.1:6379> set redlight 10
OK
127.0.0.1:6379> expire redlight 5
(integer) 1
127.0.0.1:6379>
```

不存在redlight 设置过期失败

设置redlight的过期时间是5秒

D、ttl

语法: ttl key

作用: 以秒为单位, 返回 key 的剩余生存时间 (ttl: time to live)

返回值:

- -1 : 没有设置 key 的生存时间, key 永不过期。
- -2 : key 不存在
- 数字: key 的剩余时间, 秒为单位

例 1: 设置 redlight 的过期时间是 10, 查看剩余时间


```
127.0.0.1:6379> exists redlight
(integer) 0
127.0.0.1:6379> set redlight 30
OK
127.0.0.1:6379> expire redlight 10
(integer) 1
127.0.0.1:6379> ttl redlight
(integer) 4
127.0.0.1:6379> ttl redlight
(integer) -2
```

10秒之后，key不存在了

E、type

语法：type key

作用：查看 key 所存储值的数据类型

返回值：字符串表示的数据类型

- none (key 不存在)
- string (字符串)
- list (列表)
- set (集合)
- zset (有序集)
- hash (哈希表)

例 1：查看存储字符串的 key：wood

```
127.0.0.1:6379> keys *
1) "wood"
2) "woood"
127.0.0.1:6379> get wood
"wood"
127.0.0.1:6379> type wood
string
```

例 2：查看不存在的 key

```
127.0.0.1:6379> type not-exists
none
127.0.0.1:6379>
```

F、del

语法: `del key [key...]`

作用: 删除存在的 `key` , 不存在的 `key` 忽略。

返回值: 数字, 删除的 `key` 的数量。

例 1: 删除指定的 `key`

```
127.0.0.1:6379> set name redis
OK
127.0.0.1:6379> set website redis.com
OK
127.0.0.1:6379> del redis
(integer) 0
127.0.0.1:6379> del name
(integer) 1
127.0.0.1:6379>
```

删除不存在的key, 返回0

删除一个存在的key, 返回成功删除的数量 1

2.7 Redis 的 5 种数据类型

A、字符串类型 string

字符串类型是 Redis 中最基本的数据类型, 它能存储任何形式的字符串, 包括二进制数据, 序列化后的数据, JSON 化的对象甚至是一张图片。最大 512M。

key	value
username	张三和李四

描述大段的文本信息

B、哈希类型 hash

Redis hash 是一个 string 类型的 field 和 value 的映射表, hash 特别适合于存储对象。

存对象

Redis 中所有的key都是文本类型

key	loginuser
field	value
uname	张三
times	5
region	北京

C、列表类型 list

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）**保证数据出现的顺序，按照一定顺序出现**

key	value
region	北京 上海 天津

D、集合类型 set

Redis 的 Set 是 string 类型的无序集合，集合成员是唯一的，即集合中不能出现重复的数据。

key	value
framework	spring
	mybatis
	struts

E、有序集合类型 zset （sorted set）

Redis 有序集合 zset 和集合 set 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 zset 的每个元素都会关联一个分数（分数可以重复），redis 通过分数来为集合中的成员进行从小到大的排序。

key	value score
salary	张三 3500
	李四 5000
	周丽 8000

北京动力节点

第3章 Redis 数据类型操作命令

3.1 字符串类型 (string)

字符串类型是 Redis 中最基本的数据类型，它能存储任何形式的字符串，包括二进制数据，序列化后的数据，JSON 化的对象甚至是一张图片。

3.1.1 基本命令

先测试能连接到 redis 服务器

```
D:\tools\Redis-x64-3.2.100>redis-cli.exe
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

A、set

将字符串值 value 设置到 key 中

语法：set key value

```
127.0.0.1:6379> set username zhangsan
OK
127.0.0.1:6379> set height 170cm
OK
127.0.0.1:6379> set weight 65kg
OK
```

查看已经插入的 key

```
127.0.0.1:6379> keys *
1) "weight"
2) "height"
3) "username"
127.0.0.1:6379>
```

向已经存在的 key 设置新的 value，会覆盖原来的值

```
127.0.0.1:6379> set username lisi  
OK
```

B、get

获取 key 中设置的字符串值

语法: get key

例如: 获取 username 这个 key 对应的 value

```
127.0.0.1:6379> get username  
"lisi"  
127.0.0.1:6379>
```

C、incr

将 key 中储存的数字值加 1, 如果 key 不存在, 则 key 的值先被初始化为 0 再执行 incr 操作 (只能对数字类型的数据操作)

语法: incr key

例 1: 操作key,值增加 1

```
127.0.0.1:6379> incr index  
(integer) 1  
127.0.0.1:6379> get index  
"1"
```

例 2: 对非数字的值操作是不行的

```
127.0.0.1:6379> incr username  
(error) ERR value is not an integer or out of range  
127.0.0.1:6379>
```

D、decr

将 key 中储存的数字值减1, 如果 key 不存在, 则么 key 的值先被初始化为 0 再执行 decr 操作 (只能对数字类型的数据操作)

语法: decr key

例1: 不存在的key, 初值为0, 再减 1。

```
127.0.0.1:6379>
127.0.0.1:6379> decr myindex
(integer) -1
```

myindex没有，初值0，再减1，结果-1

例2：对存在的数字值的 key ，减 1 。
先执行 incr index ,增加到 3

```
127.0.0.1:6379> get index
"3"
127.0.0.1:6379> decr index
(integer) 2
127.0.0.1:6379> get index
"2"
```

incr , decr 在实现关注人数上，文章的点击数上。

E、append

语法：append key value

说明：如果 key 存在， 则将 value 追加到 key 原来旧值的末尾
如果 key 不存在， 则将 key 设置值为 value

返回值：追加字符串之后的总长度

例 1：追加内容到存在的 key

```
127.0.0.1:6379> get k3
"v3"
127.0.0.1:6379>
127.0.0.1:6379> append k3 nokia
(integer) 7
127.0.0.1:6379> get k3
"v3nokia"
```

例 2：追加到不存在的 key，同 set key value

```
127.0.0.1:6379> get k5
(nil)
127.0.0.1:6379> append k5 hello-v5
(integer) 8
127.0.0.1:6379> get k5
"hello-v5"
```


3.1.2 常用命令

A、strlen

语法: strlen key

说明: 返回 key 所储存的字符串值的长度

返回值:

- ①: 如果key存在, 返回字符串值的长度
- ②: key不存在, 返回0

例 1: 计算存在 key 的字符串长度

```
127.0.0.1:6379> get k2
"v3-edit"
127.0.0.1:6379> strlen k2
(integer) 7
```

设置中文 set k4 中文长度, 按字符个数计算

例 2: 计算不存在的 key

```
127.0.0.1:6379> get k4
(nil)
127.0.0.1:6379> strlen k4
(integer) 0
```

B、getrange

语法: getrange key start end

作用: 获取 key 中字符串值从 start 开始 到 end 结束 的子字符串, 包括 start 和 end, 负数表示从字符串的末尾开始, -1 表示最后一个字符

返回值: 截取的字字符串。

使用的字符串 key: school, value: bjpowernode

例 1: 截取从 2 到 5 的字符

```
127.0.0.1:6379> set shcool bjpowernode
OK
127.0.0.1:6379> getrange school 2 5
"powe"
127.0.0.1:6379> 位置从0开始, p是2
```

例 2：从字符串尾部截取，start,end 是负数，最后一位是 -1

```
127.0.0.1:6379> get school
"bjpowernode"
127.0.0.1:6379> getrange school -5 -2
"rnod"
127.0.0.1:6379>
```

例 3：超出字符串范围的截取，获取合理的子串

```
127.0.0.1:6379> get school
"bjpowernode"
127.0.0.1:6379> getrange school 2 100
"powernode"
```

C、setrange

语法：setrange key offset value

说明：用 value 覆盖（替换）key 的存储的值从 offset 开始,不存在的 key 做空白字符串。

返回值：修改后的字符串的长度

例 1：替换给定的字符串

```
127.0.0.1:6379> set item java-tmocat
OK
127.0.0.1:6379> get item
"java-tmocat"
127.0.0.1:6379> setrange item 5 tom
(integer) 11
127.0.0.1:6379> get item
"java-tomcat"
```

例 2：设置不存在的 key

```
127.0.0.1:6379> get myphone
(nil)
127.0.0.1:6379> setrange myphone 0 17811111199
(integer) 11
127.0.0.1:6379> get myphone
"17811111199"
```

D、mset

语法: mset key value [key value...]

说明: 同时设置一个或多个 key-value 对

返回值: OK

例 1: 一次设置多个 key, value

```
127.0.0.1:6379> mset k1 v2 k2 v2 k3 v3
OK
127.0.0.1:6379> get k1
"v2"
127.0.0.1:6379> get k2
"v2"
127.0.0.1:6379> get k3
"v3"
```

E、mget

语法: mget key [key ...]

作用: 获取所有(一个或多个)给定 key 的值

返回值: 包含所有 key 的列表

例 1: 返回多个 key 的存储值

```
127.0.0.1:6379> mget k1 k2 k3
1) "v2"
2) "v2"
3) "v3"
```

例 2: 返回不存在的 key

```
127.0.0.1:6379> get k5
(nil)
127.0.0.1:6379> mget k1 k5 k2
1) "v2"
2) (nil)
3) "v2"
```

3.2 哈希类型 hash

redis 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。

3.2.1 基本命令

A、hset

语法：hset hash 表的 key field value

作用：将哈希表 key 中的域 field 的值设为 value，如果 key 不存在，则新建 hash 表，执行赋值，如果有 field，则覆盖值。

返回值：

- ①如果 field 是 hash 表中新 field，且设置值成功，返回 1
- ②如果 field 已经存在，旧值覆盖新值，返回 0

例 1：新的 field

```
127.0.0.1:6379> hset website baidu http://www.baidu.com
(integer) 1
```

例 2：覆盖旧的 field

```
127.0.0.1:6379> hset website baidu www.baidu.com
(integer) 0
127.0.0.1:6379>
```

B、hget

语法：hget key field

作用：获取哈希表 key 中给定域 field 的值

返回值：field 域的值，如果 key 不存在或者 field 不存在返回 nil

例 1：获取存在 key 值的某个域的值

```
127.0.0.1:6379> hset website baidu http://www.baidu.com
(integer) 1
127.0.0.1:6379> hget website baidu
"http://www.baidu.com"
127.0.0.1:6379>
```

例 2：获取不存在的 field

```
127.0.0.1:6379> hget website google
(nil)
```

C、hmset

语法: `hmset key field value [field value...]`

说明: 同时将多个 field-value (域-值) 设置到哈希表 key 中, 此命令会覆盖已经存在的 field, hash 表 key 不存在, 创建空的 hash 表, 执行 hmset.

返回值: 设置成功返回 ok, 如果失败返回一个错误

例 1: 同时设置多个 field-value

```
127.0.0.1:6379>
127.0.0.1:6379> hmset website baidu http://www.baidu.com google www.google.com
OK
127.0.0.1:6379>
```

key field field存储的值 field field存储的值

使用 redis-desktop-manager 工具查看 hash 表 website 的数据结构

HASH: website Size: 2		
row	key	value
1	baidu	http://www.baidu.com
2	google	www.google.com

例 2: key 类型不是 hash, 产生错误

```
127.0.0.1:6379> type k3
string
127.0.0.1:6379> hmset k3 k1 v1
(error) WRONGTYPE Operation against a key holding the wrong kind of value
```

D、hmget

语法: `hmget key field [field...]`

作用: 获取哈希表 key 中一个或多个给定域的值

返回值: 返回和 field 顺序对应的值, 如果 field 不存在, 返回 nil

例 1: 获取多个 field 的值

```
127.0.0.1:6379> hmset fruit banana xiangjiao-1 apple pingguo-2
OK
127.0.0.1:6379> hmget fruit apple banana orange
1) "pingguo-2"
2) "xiangjiao-1"
3) (nil)
```

显示值的顺序和hmget后面field顺序一致

E、hgetall

语法: hgetall key

作用: 获取哈希表 key 中所有的域和值

返回值: 以列表形式返回 hash 中域和域的值, key 不存在, 返回空 hash

例 1: 返回 key 对应的所有域和值

```
127.0.0.1:6379> hgetall fruit
1) "banana"
2) "xiangjiao-1"
3) "apple"
4) "pingguo-2"
```

域的名称
域对应的值

例 2: 不存在的 key, 返回空列表

```
127.0.0.1:6379> hgetall mypersion
(empty list or set)
127.0.0.1:6379>
```

F、hdel

语法: hdel key field [field...]

作用: 删除哈希表 key 中的一个或多个指定域 field, 不存在 field 直接忽略

返回值: 成功删除的 field 的数量

例 1: 删除指定的 field

```
127.0.0.1:6379> hmset number k1 v1 k2 v2 k3 v3 k4 v4 k5 v5
OK
127.0.0.1:6379> hgetall number
1) "k1"
2) "v1"
3) "k2"
4) "v2"
5) "k3"
6) "v3"
7) "k4"
8) "v4"
9) "k5"
10) "v5"
127.0.0.1:6379> hdel number k2 k3 k5
(integer) 3
127.0.0.1:6379> hgetall number
1) "k1"
2) "v1"
3) "k4"
4) "v4"
127.0.0.1:6379>
```

3.2.2 常用命令

A、hkeys

语法: hkeys key

作用: 查看哈希表 key 中的所有 field 域

返回值: 包含所有 field 的列表, key 不存在返回空列表

例 1: 查看 website 所有的域名称

```
127.0.0.1:6379> hkeys website
1) "baidu"
2) "google"
```

B、hvals

语法: hvals key

作用: 返回哈希表中所有域的值

返回值: 包含哈希表所有域值的列表, key 不存在返回空列表

例 1: 显示 website 哈希表所有域的值


```
127.0.0.1:6379> hvals website
1) "http://www.baidu.com"
2) "www.google.com"
127.0.0.1:6379>
```

C、hexists

语法: hexists key field

作用: 查看哈希表 key 中, 给定域 field 是否存在

返回值: 如果 field 存在, 返回 1, 其他返回 0

例 1: 查看存在 key 中 field 域是否存在

```
127.0.0.1:6379>
127.0.0.1:6379> hexists website baidu
(integer) 1
```

3.3 列表 list

Redis 列表是简单的字符串列表, 按照插入顺序排序。你可以添加一个元素到列表的头部(左边)或者尾部(右边)

3.3.1 基本命令

A、lpush

语法: lpush key value [value...]

作用: 将一个或多个值 value 插入到列表 key 的表头(最左边), 从左边开始加入值, 从左到右的顺序依次插入到表头

返回值: 数字, 新列表的长度

例 1: 将 a,b,c 插入到 mylist 列表类型

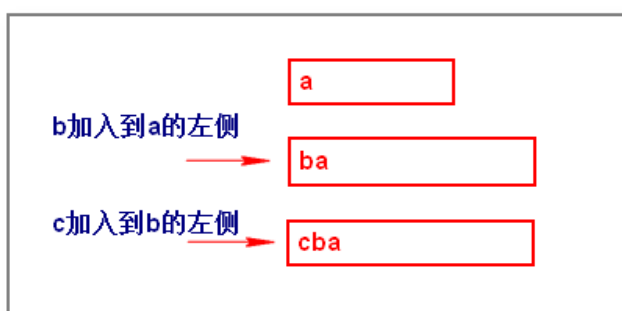
```
127.0.0.1:6379> lpush mylist a b c
(integer) 3
127.0.0.1:6379>
```

在 redis-desktop-manager 显示

LIST: mylist Size: 3

row	value
1	c
2	b
3	a

插入图示:



例 2: 插入重复值到 list 列表类型

```
127.0.0.1:6379> lpush mylist a
(integer) 4
127.0.0.1:6379>
```

在 redis-desktop-manager 显示

LIST: mylist Size: 4

row	value
1	a
2	c
3	b
4	a

B、rpush

语法: `rpush key value [value...]`

作用: 将一个或多个值 `value` 插入到列表 `key` 的表尾 (最右边), 各个 `value` 值按从左到右的顺序依次插入到表尾

返回值: 数字, 新列表的长度

例 1: 插入多个值到列表

```
127.0.0.1:6379> rpush list2 a b c
(integer) 3
127.0.0.1:6379>
```

在 redis-desktop-manager 显示：

LIST: list2	
row	value
1	a
2	b
3	c

C、lrange

语法：lrange key start stop

作用：获取列表 key 中指定区间内的元素，0 表示列表的第一个元素，以 1 表示列表的第二个元素；start, stop 是列表的下标值，也可以负数的下标，-1 表示列表的最后一个元素，-2 表示列表的倒数第二个元素，以此类推。start, stop 超出列表的范围不会出现错误。

返回值：指定区间的列表

例 1: 返回列表的全部内容

```
127.0.0.1:6379> lpush language java c c++ php python
(integer) 5
127.0.0.1:6379> lrange language 0 5
(empty list or set)
127.0.0.1:6379> lrange language 0 5
1) "python"
2) "php"
3) "c++"
4) "c"
5) "java"
```

不存在的key，返回空列表

例 2: 显示列表中第 2 个元素，下标从 0 开始

```
127.0.0.1:6379> lrange language 1 1
1) "php"
```

D、lindex

语法: lindex key index

作用: 获取列表 key 中下标为指定 index 的元素, 列表元素不删除, 只是查询。0 表示列表的第一个元素, 以 1 表示列表的第二个元素; start, stop 是列表的下标值, 也可以负数的下标, -1 表示列表的最后一个元素, -2 表示列表的倒数第二个元素, 以此类推。

返回值: 指定下标的元素; index 不在列表范围, 返回 nil

例 1: 返回下标是 1 的元素

```
127.0.0.1:6379> lrange language 0 4
1) "php"
2) "c++"
3) "c"
4) "java"
127.0.0.1:6379> lindex language 1
"c++"
```

例 2: 不存在的下标

```
127.0.0.1:6379> lindex language 100
(nil)
127.0.0.1:6379> 不存在的下标100, 返回nil
```

E、llen

语法: llen key

作用: 获取列表 key 的长度

返回值: 数值, 列表的长度; key 不存在返回 0

例 1: 显示存在 key 的列表元素的个数

```
127.0.0.1:6379> llen language
(integer) 4
```

3.3.2 常用命令

A、lrem

语法: lrem key count value

作用: 根据参数 count 的值, 移除列表中与参数 value 相等的元素, count > 0, 从列表的左侧向右开始移除; count < 0 从列表的尾部开始移除; count = 0 移除表中所有与 value 相等的值。

返回值: 数值, 移除的元素个数

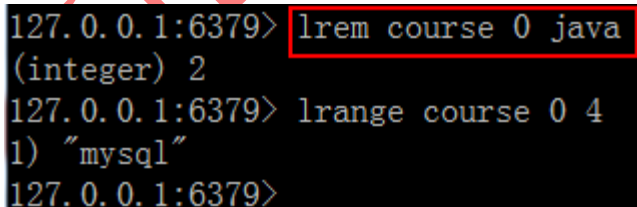
例 1: 删除 2 个相同的列表元素



```
127.0.0.1:6379> lpush course java html java html mysql
(integer) 5
127.0.0.1:6379> lrange course 0 4
1) "mysql"
2) "html"
3) "java"
4) "html"
5) "java"
127.0.0.1:6379> lrem course 2 html
(integer) 2
127.0.0.1:6379> lrange course 0 4
1) "mysql"
2) "java"
3) "java"
127.0.0.1:6379>
```

相同的html, java, 使用lrem删除了 2 个html, 从左侧开始最先遇到的2个html, 删除了

例 2: 删除列表中所有的指定元素, 删除所有的 java



```
127.0.0.1:6379> lrem course 0 java
(integer) 2
127.0.0.1:6379> lrange course 0 4
1) "mysql"
127.0.0.1:6379>
```

B、lset

语法: lset key index value

作用: 将列表 key 下标为 index 的元素的值设置为 value。

返回值：设置成功返回 ok ; key 不存在或者 index 超出范围返回错误信息

例 1：设置下标 2 的 value 为 “c”。

```
127.0.0.1:6379> del letter
(integer) 1
127.0.0.1:6379> lpush letter a b d d e
(integer) 5
127.0.0.1:6379> lrange letter 0 -1
1) "e"
2) "d"
3) "d"
4) "b"
5) "a"
127.0.0.1:6379> lset letter 2 c
OK
127.0.0.1:6379> lrange letter 0 -1
1) "e"
2) "d"
3) "c"
4) "b"
5) "a"
```

C、linsert

语法：linsert key BEFORE|AFTER pivot value

作用：将值 value 插入到列表 key 当中位于值 pivot 之前或之后的位置。key 不存在，pivot 不在列表中，不执行任何操作。

返回值：命令执行成功，返回新列表的长度。没有找到 pivot 返回 -1， key 不存在返回 0。

例 1：修改列表 arch，在值 dao 之前加入 service

```
127.0.0.1:6379> rpush arch view dao
(integer) 2
127.0.0.1:6379> lrange arch 0 -1
1) "view"
2) "dao"
127.0.0.1:6379> linsert arch BEFORE dao service
(integer) 3
127.0.0.1:6379> lrange arch 0 -1
1) "view"
2) "service"
3) "dao"
```

linsert执行成功，返回新列表的长度

例 2：操作不存在的 pivot

```
127.0.0.1:6379> linsert arch BEFORE hello new-service
(integer) -1
127.0.0.1:6379>
```

3.4 集合类型 set

redis 的 Set 是 string 类型的无序集合，集合成员是唯一的，即集合中不能出现重复的数据

3.4.1 基本命令

A、sadd

语法：sadd key member [member...]

作用：将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略，不会再加入。

返回值：加入到集合的新元素的个数。不包括被忽略的元素。

例 1：添加单个元素

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> sadd sql insert
(integer) 1
127.0.0.1:6379> sadd sql insert
(integer) 0
127.0.0.1:6379>
```

成功添加一个元素，
返回数字 1

添加集合存在的元素，
返回 0，没有新增的元素

例 2：添加多个元素

```
127.0.0.1:6379> sadd sql update delete select
(integer) 3
127.0.0.1:6379> smembers sql
1) "select"
2) "delete"
3) "insert"
4) "update"
```

B、smembers

语法：smembers key

作用：获取集合 key 中的所有成员元素，不存在的 key 视为空集合

例 1：查看集合的所有元素

```
127.0.0.1:6379> sadd html-tag table href p
(integer) 3
127.0.0.1:6379> smembers html-tag
1) "href"
2) "p"
3) "table"
```

例 2：查看不存在的集合

```
127.0.0.1:6379> del html-tag
(integer) 1
127.0.0.1:6379> smembers html-tag
(empty list or set)
127.0.0.1:6379>
```

C、sismember

语法：sismember key member

作用：判断 member 元素是否是集合 key 的成员

返回值：member 是集合成员返回 1，其他返回 0。

例 1：检查元素是否存在集合中

```
127.0.0.1:6379> sadd course html mysql spring
(integer) 3
127.0.0.1:6379> sismember course spring
(integer) 1
127.0.0.1:6379> sismember course redis
(integer) 0
```

D、scard

语法：scard key

作用：获取集合里面的元素个数

返回值：数字，key 的元素个数。其他情况返回 0。

例 1：统计集合的大小

```
127.0.0.1:6379> sadd course html mysql crm spring springmvc
(integer) 5
127.0.0.1:6379> scard course
(integer) 5
```

例 2：统计不存在的 key

```
127.0.0.1:6379> exists mycourse
(integer) 0
127.0.0.1:6379> scard mycourse
(integer) 0
```

E、srem

语法：srem key member [member...]

作用：删除集合 key 中的一个或多个 member 元素，不存在的元素被忽略。

返回值：数字，成功删除的元素个数，不包括被忽略的元素。

例 1：删除存在的一个元素，返回数字 1

```
127.0.0.1:6379> sadd lang c php java ruby python
(integer) 5
127.0.0.1:6379> srem lang php
(integer) 1
```

例 2：删除不存在的元素

```
127.0.0.1:6379> smembers lang
1) "java"
2) "c"
3) "ruby"
4) "python"
127.0.0.1:6379> srem lang golang
(integer) 0
```

3.4.2 常用命令

A、srandmember

语法：srandmember key [count]

作用：只提供 key，随机返回集合中一个元素，元素不删除，依然在集合中；提供了 count 时，count 正数，返回包含 count 个数元素的集合，集合元素各不相同。count 是负数，返回一个 count 绝对值的长度的集合，集合中元素可能会重复多次。

返回值：一个元素；多个元素的集合

例 1：随机显示集合的一个元素

```
127.0.0.1:6379> sadd type int long char double byte boolean
(integer) 6
127.0.0.1:6379> srandmember type
"char"
127.0.0.1:6379> srandmember type
"int"
```

例 2：使用 count 参数，count 是正数

```
127.0.0.1:6379> smembers type
1) "boolean"
2) "char"
3) "double"
4) "byte"
5) "int"
6) "long"
127.0.0.1:6379> srandmember type 2
1) "char"
2) "double"
127.0.0.1:6379> srandmember type 2
1) "byte"
2) "long"
```

例 3：使用 count 参数，count 是负数

```
127.0.0.1:6379> smembers type
1) "boolean"
2) "char"
3) "double"
4) "byte"
5) "int"
6) "long"
127.0.0.1:6379> srandmember type -2
1) "byte"
2) "byte"
127.0.0.1:6379> srandmember type -3
1) "char"
2) "int"
3) "long"
```

B、spop

语法：spop key [count]

作用：随机从集合中删除一个元素，count 是删除的元素个数。

返回值：被删除的元素，key 不存在或空集合返回 nil

例如 1：随机从集合删除一个元素

```
127.0.0.1:6379> sadd db MySQL Oracle MongoDB reids
(integer) 4
127.0.0.1:6379> spop db
"Oracle"
127.0.0.1:6379> smembers db
1) "MySQL"
2) "reids"
3) "MongoDB"
```

例 2：随机删除指定个数的元素

```
127.0.0.1:6379> smembers db
1) "MySQL"
2) "reids"
3) "MongoDB"
127.0.0.1:6379> spop db 2
1) "reids"
2) "MySQL"
127.0.0.1:6379> smembers db
1) "MongoDB"
```

3.5 有序集合类型 zset (sorted set)

redis 有序集合zset和集合set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 zset 的每个元素都会关联一个分数（分数可以重复），redis 通过分数来为集合中的成员进行从小到大的排序。

3.5.1 基本命令

A、zadd

语法：zadd key score member [score member...]

作用：将一个或多个 member 元素及其 score 值加入到有序集合 key 中，如果 member 存在集合中，则更新值；score 可以是整数或浮点数

返回值：数字，新添加的元素个数

例 1：创建保存学生成绩的集合

```
127.0.0.1:6379> zadd studentscore 80 zhangsan 92 lisi 75 wangwu
(integer) 3
127.0.0.1:6379>
```

例 2：使用浮点数作为 score

```
127.0.0.1:6379> zadd mycourse 82.25 html 75.56 mysql 92.5 java
(integer) 3
127.0.0.1:6379>
```

B、zrange

语法：zrange key start stop [WITHSCORES]

作用：查询有序集合，指定区间的内的元素。集合成员按 score 值从小到大来排序。start, stop 都是从 0 开始。0 是第一个元素，1 是第二个元素，依次类推。以 -1 表示最后一个成员，-2 表示倒数第二个成员。WITHSCORES 选项让 score 和 value 一同返回。

返回值：自定区间的成员集合

例 1：显示集合的全部元素，不显示 score，不使用 WITHSCORES

```
127.0.0.1:6379> zrange studentscore 0 -1
1) "wangwu"
2) "zhangsan"
3) "lisi"
```

例 2：显示集合全部元素，并使用 WITHSCORES

```
127.0.0.1:6379> zrange studentscore 0 -1 withscores
1) "wangwu"
2) "75"
3) "zhangsan"
4) "80"
5) "lisi"
6) "92"
127.0.0.1:6379>
```

注意数据的显示格式：
元素值
score
元素值
score
...

例 3：显示第 0,1 二个成员

```
127.0.0.1:6379>
127.0.0.1:6379> zrange studentscore 0 1 withscores
1) "wangwu"
2) "75"
3) "zhangsan"
4) "80"
```

例 4：排序显示浮点数的 score

```
127.0.0.1:6379> zrange mycourse 0 -1 withscores  
1) "mysql"  
2) "75.560000000000002"  
3) "html"  
4) "82.25"  
5) "java"  
6) "92.5"
```

C、zrevrange

语法: `zrevrange key start stop [WITHSCORES]`

作用: 返回有序集 `key` 中, 指定区间内的成员。其中成员的位置按 `score` 值递减(从大到小)来排列。其它同 `zrange` 命令。

返回值: 自定区间的成员集合

例 1: 成绩榜

```
127.0.0.1:6379> zrevrange studentscore 0 -1 withscores  
1) "lisi"  
2) "92"  
3) "zhangsan"  
4) "80"  
5) "wangwu"  
6) "75"  
127.0.0.1:6379>
```

D、zrem

语法: `zrem key member [member...]`

作用: 删除有序集合 `key` 中的一个或多个成员, 不存在的成员被忽略

返回值: 被成功删除的成员数量, 不包括被忽略的成员。

例 1: 删除指定一个成员 `wangwu`

```
127.0.0.1:6379> zrange studentscore 0 -1
1) "wangwu"
2) "zhangsan"
3) "lisi"
127.0.0.1:6379> zrem studentscore wangwu
(integer) 1
127.0.0.1:6379> zrange studentscore 0 -1
1) "zhangsan"
2) "lisi"
```

E、zcard

语法: zcard key

作用: 获取有序集 key 的元素成员的个数

返回值: key 存在返回集合元素的个数, key 不存在, 返回 0

例 1: 查询集合的元素个数

```
127.0.0.1:6379> zadd salary 3000 tom 6500 jack 2000 john
(integer) 3
127.0.0.1:6379> zcard salary
(integer) 3
127.0.0.1:6379> zadd salary 4500 rose
(integer) 1
127.0.0.1:6379> zcard salary
(integer) 4
127.0.0.1:6379>
```

3.5.2 常用命令

A、zrangebyscore

语法: zrangebyscore key min max [WITHSCORES] [LIMIT offset count]

作用: 获取有序集 key 中, 所有 score 值介于 min 和 max 之间 (包括 min 和 max) 的成员, 有序成员是按递增 (从小到大) 排序。

min ,max 是包括在内, 使用符号 (表示不包括。 min , max 可以使用 -inf , +inf 表示最小和最大

limit 用来限制返回结果的数量和区间。

withscores 显示 score 和 value

返回值: 指定区间的集合数据

使用的准备数据

```
127.0.0.1:6379> zrange salary 0 -1 withscores
1) "john"
2) "2000"
3) "tom"
4) "3000"
5) "rose"
6) "4500"
7) "jack"
8) "6500"
```

例 1: 显示指定具体区间的数据

```
127.0.0.1:6379> zrangebyscore salary 2000 4500 withscores
1) "john"
2) "2000"
3) "tom"
4) "3000"
5) "rose"
6) "4500"
```

包含区间2000 和 4500

例 2: 显示指定具体区间的集合数据, 开区间 (不包括 min, max)

```
127.0.0.1:6379> zrangebyscore salary (2000 (4500 withscores
1) "tom"
2) "3000"
```

例 3: 显示整个集合的所有数据

```
127.0.0.1:6379> zrangebyscore salary -inf +inf withscores
1) "john"
2) "2000"
3) "tom"
4) "3000"
5) "rose"
6) "4500"
7) "jack"
8) "6500"
```

使用 min 用 -inf
max 使用 +inf

例 4: 使用 limit

增加新的数据:


```
127.0.0.1:6379> zadd salary 3600 marray
(integer) 1
```

```
127.0.0.1:6379> zrange salary 0 -1 withscores
1) "john"
2) "2000"
3) "tom"
4) "3000"
5) "marray"
6) "3600"
7) "rose"
8) "4500"
9) "jack"
10) "6500"
127.0.0.1:6379> zrangebyscore salary 3000 5000 withscores limit 0 2
1) "tom"
2) "3000"
3) "marray"
4) "3600"
127.0.0.1:6379>
```

3000 - 5000范围内 tom是第0个元素；marray是第1个元素；
rose是第2个元素

limit语句0是开始位置，2是数量

显示从第一个位置开始，取一个元素。

```
127.0.0.1:6379> zrangebyscore salary 3000 5000 withscores limit 1 1
1) "marray"
2) "3600"
```

B、zrevrangebyscore

语法：zrevrangebyscore key max min [WITHSCORES] [LIMIT offset count]

作用：返回有序集 key 中，score 值介于 max 和 min 之间(默认包括等于 max 或 min)的所有成员。有序集成员按 score 值递减(从大到小)的次序排列。其他同 zrangebyscore

例 1：查询工资最高到 3000 之间的员工

```
127.0.0.1:6379> zrange salary 0 -1 withscores
1) "john"
2) "2000"
3) "tom"
4) "3000"
5) "marray"
6) "3600"
7) "rose"
8) "4500"
9) "jack"
10) "6500"
127.0.0.1:6379> zrevrangebyscore salary +inf 3000 withscores
1) "jack"
2) "6500"
3) "rose"
4) "4500"
5) "marray"
6) "3600"
7) "tom"
8) "3000"
127.0.0.1:6379>
```

C、zcount

语法: zcount key min max

作用: 返回有序集 key 中, score 值在 min 和 max 之间(默认包括 score 值等于 min 或 max) 的成员的数量

例 1: 求工资在 3000-5000 的员工数量

```
127.0.0.1:6379> zrange salary 0 -1 withscores
1) "john"
2) "2000"
3) "tom"
4) "3000"
5) "rose"
6) "4500"
7) "jack"
8) "6500"
127.0.0.1:6379> zcount salary 3000 5000
(integer) 2
127.0.0.1:6379>
```

第4章 高级话题

4.1 Redis 事务

4.1.1 什么是事务

事务是指一系列操作步骤,这一系列的操作步骤,要么完全地执行,要么完全地不执行。

Redis 中的事务 (transaction) 是一组命令的集合,至少是两个或两个以上的命令,redis 事务保证这些命令被执行时中间不会被任何其他操作打断。

4.1.2 事务操作的命令

(1) multi

语法: multi

作用: 标记一个事务的开始。事务内的多条命令会按照先后顺序被放进一个队列当中。

返回值: 总是返回 ok

(2) exec

语法: exec

作用: 执行所有事务块内的命令

返回值: 事务内的所有执行语句内容, 事务被打断 (影响) 返回 nil

(3) discard

语法: discard

作用: 取消事务, 放弃执行事务块内的所有命令

返回值: 总是返回 ok

(4) watch

语法: watch key [key ...]

作用: 监视一个(或多个) key, 如果在事务执行之前这个(或这些) key 被其他命令所改动, 那么事务将被打断。

返回值: 总是返回 ok

(5) unwatch

语法: unwatch

作用: 取消 WATCH 命令对所有 key 的监视。如果在执行 WATCH 命令之后, EXEC 命令或 DISCARD 命令先被执行了的话, 那么就不需要再执行 UNWATCH 了

返回值: 总是返回 ok

4.1.3 事务的实现

(1) 正常执行事务

事务的执行步骤: 首先开启事务, 其次向事务队列中加入命令, 最后执行事务提交

例 1: 事务的执行:

- 1) multi : 用 multi 命令告诉 Redis, 接下来要执行的命令你先不要执行, 而是把它们暂时存起来 (开启事务)
- 2) sadd works john 第一条命令进入等待队列 (命令入队)
- 3) sadd works rose 第二条命令进入等待队列 (命令入队)
- 4) exce 告知 redis 执行前面发送的两条命令 (提交事务)

```
127.0.0.1:6379>
127.0.0.1:6379> multi
OK
127.0.0.1:6379> sadd works john
QUEUED
127.0.0.1:6379> sadd works rose
QUEUED
127.0.0.1:6379> exec
1) (integer) 1
2) (integer) 1
127.0.0.1:6379>
```

开启事务, 其后的命令加入命令队列, 暂不执行命令

执行两条命令, QUEUED 表示命令进入队列, 等待执行

执行事务, 队列中的命令全部开始执行, exec 命令返回的是顺序执行的每条命令的执行结果

查看 works 集合

```
127.0.0.1:6379> smembers works
1) "john"
2) "rose"
```

(2) 事务执行 **exec** 之前，入队命令错误（语法错误；严重错误导致服务器不能正常工作（例如内存不足）），放弃事务。

执行事务步骤：

- 1) MULTI 正常命令
- 2) SET key value 正常命令
- 3) INCR 命令语法错误
- 4) EXEC 无法执行事务，那么第一条正确的命令也不会执行，所以 key 的值不会设置成功

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set logindate 2017-05-20
QUEUED
127.0.0.1:6379> incr a b c ← incr 语法错误
(error) ERR wrong number of arguments for 'incr' command
127.0.0.1:6379>
127.0.0.1:6379> get logindate
QUEUED ← 事务执行，没有成功，exec之前的错误，Redis放弃事务的执行
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
127.0.0.1:6379> get logindate
(nil) ← 查询logindate的值 nil，没有设置成功，事务没有执行，被放弃
127.0.0.1:6379>
```

结论：事务执行 **exec** 之前，入队命令错误，事务终止，取消，不执行。

(3) 事务执行 **exec** 命令后，执行队列命令，命令执行错误，事务提交

执行步骤：

- 1) MULTI 正常命令
- 2) SET username zhangsan 正常命令
- 3) lpop username 正常命令，语法没有错误，执行命令时才会有错误。
- 4) EXEC 正常执行，发现错误可以在事务提交前放弃事务，执行 **discard**。

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set username zhangsan
QUEUED
127.0.0.1:6379> lpop username
QUEUED
127.0.0.1:6379> exec
1) OK
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
127.0.0.1:6379>
127.0.0.1:6379> get username
"zhangsan"
127.0.0.1:6379>
```

这条命令是错误的，但命令进入队列。命令语法没有错误，`lpop` 是列表类型的命令，不能操作字符串类型的数据

`exec` 命令后执行 `lpop username` 出错，但事务中的其他命令依然执行。

结论: 在 `exec` 执行后的所产生的错误，即使事务中有某个/某些命令在执行时产生了错误，事务中的其他命令仍然会继续执行。

Redis 在事务失败时不进行回滚，而是继续执行余下的命令。

Redis 这种设计原则是：Redis 命令只会因为错误的语法而失败（这些问题不能在入队时发现），或是命令用在了错误类型的键上面，失败的命令并不是 Redis 导致，而是由编程错误造成的，这样错误应该在开发的过程中被发现，生产环境中不应出现语法的错误。就是在程序的运行环境中不应该出现语法的错误。而 Redis 能够保证正确的命令一定会被执行。再者不需要对回滚进行支持，所以 Redis 的内部可以保持简单且快速。

（4）放弃事务

执行步骤：

- 1) MULTI 开启事务
- 2) SET age 25 命令入队
- 3) SET age 30 命令入队
- 4) DISCARD 放弃事务，则命令队列不会被执行

例 1：

```
127.0.0.1:6379>
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set age 25
QUEUED
127.0.0.1:6379> set age 30
QUEUED
127.0.0.1:6379> discard 放弃事务
OK
127.0.0.1:6379> get age
(nil)
```

(5) Redis 的 watch 机制

A、Redis 的 WATCH 机制

WATCH 机制原理：

WATCH 机制：使用 WATCH 监视一个或多个 key，跟踪 key 的 value 修改情况，如果有 key 的 value 值在事务 EXEC 执行之前被修改了，整个事务被取消。EXEC 返回提示信息，表示事务已经失败。

WATCH 机制使的事务 EXEC 变的有条件，事务只有在被 WATCH 的 key 没有修改的前提下才能执行。不满足条件，事务被取消。使用 WATCH 监视了一个带过期时间的键，那么即使这个键过期了，事务仍然可以正常执行

大多数情况下，不同的客户端会访问不同的键，相互同时竞争同一 key 的情况一般都很少，乐观锁能够以很好的性能解决数据冲突的问题。

B、何时取消 key 的监视 (WATCH) ?

- ① WATCH 命令可以被调用多次。对键的监视从 WATCH 执行之后开始生效，直到调用 EXEC 为止。不管事务是否成功执行，对所有键的监视都会被取消。
- ② 当客户端断开连接时，该客户端对键的监视也会被取消。
- ③ UNWATCH 命令可以手动取消对所有键的监视

C、WATCH 的事例

执行步骤：

首先启动 redis-server，在开启两个客户端连接。分别叫 A 客户端 和 B 客户端。

启动 Redis 服务器

```
[root@localhost src]#  
[root@localhost src]# ./redis-server &  
[1] 4256  
[root@localhost src]# 4256:C 22 Aug 22:30:55.125 # W  
config file use ./redis-server /path/to/redis.conf
```

A 客户端（红色）：WATCH 某个 key，同时执行事务

```
1 练习192.168.1.104 x +  
[root@localhost src]# clear  
[root@localhost src]#  
[root@localhost src]#  
[root@localhost src]# ./redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379> █
```

B 客户端（黄色）：对 A 客户端 WATCH 的 key 修改其 value 值。

```
1 练习192.168.1.104 x +  
[root@localhost ~]# cd /usr/local/redis  
[root@localhost src]# clear  
[root@localhost src]#  
[root@localhost src]#  
[root@localhost src]# ./redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379> █
```

- 1) 在 A 客户端设置 key : str.lp 登录人数为 10
- 2) 在 A 客户端监视 key : str.lp
- 3) 在 A 客户端开启事务 multi
- 4) 在 A 客户端修改 str.lp 的值为 11
- 5) 在 B 客户端修改 str.lp 的值为 15
- 6) 在 A 客户端执行事务 exec
- 7) 在 A 客户端查看 str.lp 值，A 客户端执行的事务没有提交，因为 WATCH 的 str.lp 的值已经被修改了，所有放弃事务。

例 1：乐观锁


```

127.0.0.1:6379>
127.0.0.1:6379> set str.lp 10
OK
127.0.0.1:6379> watch str.lp
OK
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set str.lp 11
QUEUED
127.0.0.1:6379> exec
(nil)
127.0.0.1:6379> get str.lp
"15"

双击此处添加一个新的按钮。

127.0.0.1:6379> set str.lp 15
OK
  
```

4.2 持久化

4.2.1 持久化概述

持久化可以理解为存储，就是将数据存储到一个不会丢失的地方，如果把数据放在内存中，电脑关闭或重启数据就会丢失，所以放在内存中的数据不是持久化的，而放在磁盘就算是一种持久化。

Redis 的数据存储在内存中，内存是瞬时的，如果 linux 宕机或重启，又或者 Redis 崩溃或重启，所有的内存数据都会丢失，为了解决这个问题，Redis 提供两种机制对数据进行持久化存储，便于发生故障后能迅速恢复数据。

4.2.2 持久化方式

(1) RDB 方式

A、什么是 RDB 方式?

Redis Database (RDB), 就是在指定的时间间隔内将内存中的数据快照写入磁盘, 数据恢复时将快照文件直接再读到内存。

RDB 保存了在某个时间点的数据集 (全部数据)。存储在一个二进制文件中, 只有一个文件。默认是 `dump.rdb`。RDB 技术非常适合做备份, 可以保存最近一个小时, 一天, 一个月的全部数据。保存数据是在单独的进程中写文件, 不影响 Redis 的正常使用。RDB 恢复数据时比其他 AOF 速度快。

B、如何实现?

RDB 方式的数据持久化, 仅需在 `redis.conf` 文件中配置即可, 默认配置是启用的。

在配置文件 `redis.conf` 中搜索 `SNAPSHOTTING`, 查找在注释开始和结束之间的关于 RDB 的配置说明。配 `SNAPSHOTTING` 置地方有三处。

①: 配置执行 RDB 生成快照文件的时间策略。

对 Redis 进行设置, 让它在“N 秒内数据集至少有 M 个 key 改动”这一条件被满足时, 自动保存一次数据集。

配置格式: `save <seconds> <changes>`

`save 900 1`

`save 300 10`

`save 60 10000`

②: `dbfilename`: 设置 RDB 的文件名, 默认文件名为 `dump.rdb`

③: `dir`: 指定 RDB 文件的存储位置, 默认是 `./` 当前目录

配置步骤:

①: 查看 `ps -ef | grep redis`, 如果 redis 服务启动, 先停止。

```
[root@localhost src]# ps -ef | grep redis
root      6739      3462  0 14:37 pts/0    00:00:00 grep --color=auto redis
[root@localhost src]#
```

②: 修改 `redis.conf` 文件, 修改前先备份, 执行 `cp redis.conf bak_redis.conf`

```
[root@localhost redis-3.2.9]# cp redis.conf bak_redis.conf
[root@localhost redis-3.2.9]# ll bak_redis.conf
-rw-r--r-- 1 root root 46696 Aug 23 14:40 bak_redis.conf
```

查看默认启用的 RDB 文件

```
[root@localhost src]# pwd
/usr/local/redis-3.2.9/src
[root@localhost src]# ll *.rdb
-rw-r--r--. 1 root root 212 Aug 23 14:47 dump.rdb
```

③：编辑 redis.conf 增加 save 配置， 修改文件名等。vim redis.conf

```
##### SNAPSHOTTING #####
#
# Save the DB on disk:
#
#   save <seconds> <changes>
#
```

从这个注释开始是RDB的配置

修改的内容：

```
save 900 1
#save 300 10
save 60 10000
save 20 3
```

```
# dbfilename dump.rdb
dbfilename mydump.rdb
```

把原来的默认的 dump.rdb 删除，修改 redis.conf 后，重新启动 redis

④：在 20 秒内容，修改三个 key 的值

```
127.0.0.1:6379> set a1 v1
OK
127.0.0.1:6379> set a2 v2
OK
127.0.0.1:6379> set a3 v3
OK
```

⑤：查看生成的 rdb 文件

```
[root@localhost src]# pwd
/usr/local/redis-3.2.9/src
[root@localhost src]# ll *.rdb
-rw-r--r--. 1 root root 102 Aug 23 15:03 mydump.rdb
[root@localhost src]#
```

C、总结

优点：由于存储的是数据快照文件，恢复数据很方便，也比较快

缺点：

1) 会丢失最后一次快照以后更改的数据。如果你的应用能容忍一定数据的丢失，那么使用 rdb 是不错的选择；如果你不能容忍一定数据的丢失，使用 rdb 就不是一个很好的选择。

2) 由于需要经常操作磁盘，RDB 会分出一个子进程。如果你的 redis 数据库很大的话，子进程占用比较多的时间，并且可能会影响 Redis 暂停服务一段时间（millisecond 级别），如果你的数据库超级大并且你的服务器 CPU 比较弱，有可能是会达到一秒。

(2) AOF 方式

A、什么是 AOF 方式

Append-only File (AOF)，Redis 每次接收到一条改变数据的命令时，它将把该命令写到一个 AOF 文件中（只记录写操作，读操作不记录），当 Redis 重启时，它通过执行 AOF 文件中所有的命令来恢复数据。

B、如何实现

AOF 方式的数据持久化，仅需在 redis.conf 文件中配置即可
配置项：

- ①: appendonly: 默认是 no，改成 yes 即开启了 aof 持久化
- ②: appendfilename: 指定 AOF 文件名，默认文件名为 appendonly.aof
- ③: dir: 指定 RDB 和 AOF 文件存放的目录，默认是 ./
- ④: appendfsync: 配置向 aof 文件写命令数据的策略：
 - no: 不主动进行同步操作，而是完全交由操作系统来做（即每 30 秒一次），比较快但不是很安全。
 - always: 每次执行写入都会执行同步，慢一些但是比较安全。
 - everysec: 每秒执行一次同步操作，比较平衡，介于速度和安全之间。这是默认项。
- ⑤: auto-aof-rewrite-min-size: 允许重写的最小 AOF 文件大小，默认是 64M 。当 aof 文件大于 64M 时，开始整理 aof 文件， 去掉无用的操作命令。缩小 aof 文件。

例 1:

- ①: 停止运行的 redis ， 备份要修改的 redis.conf
- ②: 查看 redis 安装目录/src 下有无 .aof 文件。 默认是在 redis 的当前目录

```
[root@localhost src]# ll *.aof
ls: cannot access *.aof: No such file or directory
```

- ③: 编辑 redis.conf
设置 appendonly 为 yes 即可。

查看 appendfsync 的当前策略。

查看 appendfilename 的文件名称

```
# Please check http://redis.io/topics/persistence for more information.

appendonly no  ←----- no ----> yes

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

appendfilename "appendonly.aof"
```

```
# appendfsync always
appendfsync everysec
# appendfsync no
```

AOF策略默认是每秒

④：在 redis 客户端执行 写入命令

```
127.0.0.1:6379>
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> set k2 v2
OK
127.0.0.1:6379> get v2
(nil)
127.0.0.1:6379> set k3 v3
OK
```

⑤ 查看 aof 文件

```
[root@localhost src]# ll *.aof
-rw-r--r--. 1 root root 110 Aug 23 15:39 appendonly.aof
[root@localhost src]#
```

```
set
$2
k2
$2
v2
*3
$3
set
$2
k3
$2
v3
```

在set k3之前没有读取get k2的命令记录。AOF只记录写的命令

(3) 总结

- 1) append-only 文件是另一个可以提供完全数据保障的方案;
- 2) AOF 文件会在操作过程中变得越来越大。比如, 如果你做一百次加法计算, 最后你只会在数据库里面得到最终的数值, 但是在你的 AOF 里面会存在 100 次记录, 其中 99 条记录对最终的结果是无用的; 但 Redis 支持在不影响服务的前提下在后台重构 AOF 文件, 让文件得以整理变小
- 3) 可以同时使用这两种方式, redis 默认优先加载 aof 文件 (aof 数据最完整);

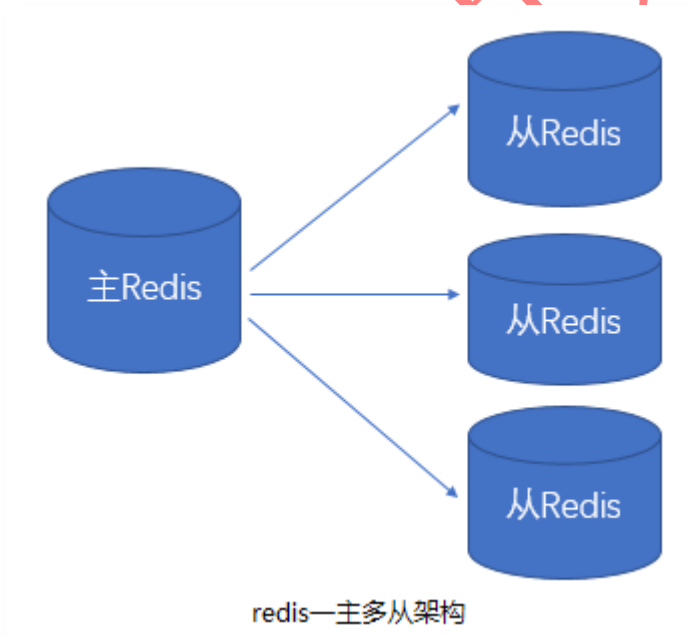
4.3 主从复制

4.3.1 主从复制--读写分离

通过持久化功能, Redis 保证了即使在服务器重启的情况下也不会丢失 (或少量丢失) 数据, 但是由于数据是存储在一台服务器上的, 如果这台服务器出现故障, 比如硬盘坏了, 也会导致数据丢失。

为了避免单点故障, 我们需要将数据复制多份部署在多台不同的服务器上, 即使有一台服务器出现故障其他服务器依然可以继续提供服务。

这就要求当一台服务器上的数据更新后, 自动将更新的数据同步到其他服务器上, 那该怎么实现呢? Redis 的主从复制。



Redis 提供了复制 (replication) 功能来自动实现多台 redis 服务器的数据同步 (每天 19 点 新闻联播, 基本从 CCTV1-8, 各大卫视都会播放)

我们可以通过部署多台 redis, 并在配置文件中指定这几台 redis 之间的主从关系, 主负责写入数据, 同时把写入的数据实时同步到从机器, 这种模式叫做主从复制, 即 master/slave, 并且 redis 默认 master 用于写, slave 用于读, 向 slave 写数据会导致错误

(1) Redis 主从复制实现 (master/salve)

方式 1: 修改配置文件, 启动时, 服务器读取配置文件, 并自动成为指定服务器的从服务器, 从而构成主从复制的关系

方式 2: `./redis-server --slaveof <master-ip> <master-port>`, 在启动 redis 时指定当前服务成为某个主 Redis 服务的从 Slave

方式 1 的实现步骤:

模拟多 Reids 服务器, 在一台已经安装 Redis 的机器上, 运行多个 Redis 应用模拟多个 Reids 服务器。一个 Master, 两个 Slave.

A、新建三个 Redis 的配置文件

如果 Redis 启动, 先停止。

作为 Master 的 Redis 端口是 6380

作为 Slaver 的 Redis 端口分别是 6382, 6384

从原有的 redis.conf 拷贝三份, 分别命名为 redis6380.conf, redis6382.conf, redis6384.conf

```
[root@localhost redis-3.2.9]# cp redis.conf redis6380.conf
[root@localhost redis-3.2.9]#
[root@localhost redis-3.2.9]# > redis6380.conf 输出空内容到redis6380.conf
[root@localhost redis-3.2.9]# cat redis6380.conf
[root@localhost redis-3.2.9]#
[root@localhost redis-3.2.9]# cp redis6380.conf redis6382.conf
[root@localhost redis-3.2.9]# cp redis6380.conf redis6384.conf
[root@localhost redis-3.2.9]# ll redis*.conf
-rw-r--r--. 1 root root    0 Aug 23 17:55 redis6380.conf
-rw-r--r--. 1 root root    0 Aug 23 17:56 redis6382.conf
-rw-r--r--. 1 root root    0 Aug 23 17:56 redis6384.conf
-rw-rw-r--. 1 root root 46721 Aug 23 15:48 redis.conf
[root@localhost redis-3.2.9]#
```

B、编辑 Master 配置文件

编辑 Master 的配置文件 redis6380.conf: 在空文件加入如下内容

`include /usr/local/redis-3.2.9/redis.conf`

`daemonize yes`

`port 6380`

`pidfile /var/run/redis_6380.pid`

`logfile 6380.log`

`dbfilename dump6380.rdb`

配置项说明:

`include` : 包含原来的配置文件内容。`/usr/local/redis-3.2.9/redis.conf` 按照自己的目录设置。

`daemonize: yes` 后台启动应用, 相当于 `./redis-server &`, `&`的作用。

port: 自定义的端口号

pidfile: 自定义的文件, 表示当前程序的 pid, 进程 id。

logfile: 日志文件名

dbfilename: 持久化的 rdb 文件名

C、编辑 Slave 配置文件

编辑 Slave 的配置文件 redis6382.conf 和 redis6384.conf: 在空文件加入如下内容

①: redis6382.conf:

```
include /usr/local/redis-3.2.9/redis.conf
daemonize yes
port 6382
pidfile /var/run/redis_6382.pid
logfile 6382.log
dbfilename dump6382.rdb
slaveof 127.0.0.1 6380
```

配置项说明:

slaveof : 表示当前 Redis 是谁的从。当前是 127.0.0.1 端口 6380 这个 Master 的从。

②: redis6384.conf:

```
include /usr/local/redis-3.2.9/redis.conf
daemonize yes
port 6384
pidfile /var/run/redis_6384.pid
logfile 6384.log
dbfilename dump6384.rdb
slaveof 127.0.0.1 6380
```

D、启动服务器 Master/Slave 都启动

启动方式 ./redis-server 配置文件

启动 Redis, 并查看启动进程

```
[root@localhost redis-3.2.9]# pwd
/usr/local/redis-3.2.9
[root@localhost redis-3.2.9]# ls *.conf
bak_redis.conf redis6380.conf redis6382.conf redis6384.conf redis.conf sentinel.conf
[root@localhost redis-3.2.9]# cd src/
[root@localhost src]# ./redis-server ../redis6380.conf
[root@localhost src]# ./redis-server ../redis6382.conf
[root@localhost src]# ./redis-server ../redis6384.conf
[root@localhost src]# ps -ef | grep redis
root      9227      1   0 18:13 ?        00:00:00 ./redis-server *:6380
root      9231      1   0 18:13 ?        00:00:00 ./redis-server *:6382
root      9237      1   0 18:13 ?        00:00:00 ./redis-server *:6384
root      9243    3462   0 18:13 pts/0    00:00:00 grep --color=auto redis
[root@localhost src]#
```


E、查看配置后的服务信息

命令:

①: Redis 客户端使用指定端口连接 Redis 服务器

./redis-cli -p 端口

②: 查看服务器信息

info replication

登录到 Master: 6380

```
[root@localhost src]#  
[root@localhost src]# ./redis-cli -p 6380  
127.0.0.1:6380>  
127.0.0.1:6380>
```

查看当前服务信息

在客户端的 Redis 内执行命令 info replication

Master 服务的查看结果:

```
127.0.0.1:6380>  
127.0.0.1:6380> info replication  
# Replication  
role:master  
connected slaves:2  
slave0:ip=127.0.0.1,port=6382,state=online,offset=519,lag=1  
slave1:ip=127.0.0.1,port=6384,state=online,offset=519,lag=0  
master_repl_offset:519  
repl_backlog_active:1
```

在新的 Xshell 窗口分别登录到 6382 , 6384 查看信息

```
[root@localhost src]#  
[root@localhost src]# ./redis-cli -p 6382  
127.0.0.1:6382> info replication  
# Replication  
role:slave  
master_host:127.0.0.1  
master_port:6380  
master_link_status:up  
master_last_io_seconds_ago:10  
master_sync_in_progress:0
```

6384 也登录内容同 6382.

F、向 Master 写入数据

在 6380 执行 flushall 清除数据，避免干扰的测试数据。 生产环境避免使用。

```
127.0.0.1:6380>
127.0.0.1:6380> flushall
OK
127.0.0.1:6380> set mk1 v1
OK
127.0.0.1:6380> set mk2 v2
OK
127.0.0.1:6380>
```

G、在从 Slave 读数据

6382,6384 都可以读主 Master 的数据，不能写

```
127.0.0.1:6382>
127.0.0.1:6382> get mk1
"v1"
127.0.0.1:6382> get mk2
"v2"
127.0.0.1:6382>
127.0.0.1:6382> get mk3
(nil) Master没有mk3
```

Slave 写数据失败

```
127.0.0.1:6382> set mk5 v5
(error) READONLY You can't write against a read only slave.
127.0.0.1:6382>
```

(2) 容灾处理

当 Master 服务出现故障，需手动将 slave 中的一个提升为 master，剩下的 slave 挂至新的 master 上（冷处理：机器挂掉了，再处理）

命令：

- ①：slaveof no one，将一台 slave 服务器提升为 Master （提升某 slave 为 master）
- ②：slaveof 127.0.0.1 6381 （将 slave 挂至新的 master 上）

执行步骤：

A、将 Master:6380 停止（模拟挂掉）

```
127.0.0.1:6380>
127.0.0.1:6380> shutdown
not connected>
```

```
[root@localhost src]# ps -ef | grep redis
root      9231      1  0 18:13 ?        00:00:02 ./redis-server *:6382
root      9237      1  0 18:13 ?        00:00:02 ./redis-server *:6384
root      9395    6887  0 18:25 pts/2    00:00:00 ./redis-cli -p 6382
root      9484    7009  0 18:32 pts/3    00:00:00 ./redis-cli -p 6384
root      9619    3462  0 18:43 pts/0    00:00:00 grep --color=auto redis
```

B、选择一个 Slave 升到 Master，其它的 Slave 挂到新提升的 Master

```
127.0.0.1:6382>
127.0.0.1:6382> slaveof no one ← 提升为 Master
OK
127.0.0.1:6382> info replication
# Replication
role:master ← 提升后变为master, 当前的Slave是 0
connected_slaves:0
master_repl_offset:0
```

C、将其他 Slave 挂到新的 Master

在 Slave 6384 上执行

```
127.0.0.1:6384> slaveof 127.0.0.1 6382
OK
127.0.0.1:6384> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6382
master_link_status:up
```

现在的主从（Master/Slave）关系：Master 是 6382 ， Slave 是 6384
查看 6382：

```
127.0.0.1:6382>
127.0.0.1:6382> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=6384,state=online,offset=281,lag=1
master_repl_offset:281
```

D、原来的服务器重新添加到主从结构中

6380 的服务器修改后，从新工作，需要把它添加到现有的 Master/Slave 中
先启动 6380 的 Redis 服务

```
[root@localhost src]#
[root@localhost src]# ./redis-server ../redis6380.conf
[root@localhost src]#
[root@localhost src]# ps -ef | grep redis
root      9231      1  0 18:13 ?        00:00:03 ./redis-server *:6382
root      9237      1  0 18:13 ?        00:00:03 ./redis-server *:6384
root      9395    6887  0 18:25 pts/2    00:00:00 ./redis-cli -p 6382
root      9484    7009  0 18:32 pts/3    00:00:00 ./redis-cli -p 6384
root      9782      1  0 18:56 ?        00:00:00 ./redis-server *:6380
root      9786    3462  0 18:56 pts/0    00:00:00 grep --color=auto redis
```

连接到 6380 端口

```
[root@localhost src]#
[root@localhost src]# ./redis-cli -p 6380
127.0.0.1:6380> info replication
# Replication
role:master
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
```

默认新加的服务是master

当前服务挂到 Master 上

```
127.0.0.1:6380> slaveof 127.0.0.1 6382
OK
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6382
master_link_status:up
```

E、查看新的 Master 信息

在 6382 执行：

```
127.0.0.1:6382> info replication
# Replication
role:master
connected slaves:2
slave0:ip=127.0.0.1,port=6384,state=online,offset=1121,lag=0
slave1:ip=127.0.0.1,port=6380,state=online,offset=1121,lag=0
master_repl_offset:1121
```

现在的 Master/Slaver 关系是：

Master: 6382

Slave: 6380

6384

(3) 操作命令

进入客户端需指定端口：./redis-cli -p 6380

不配置启动默认都是主 master

info replication 查看 redis 服务器所处角色

(4) 总结

- 1、一个 master 可以有多个 slave
- 2、slave 下线，读请求的处理性能下降
- 3、master 下线，写请求无法执行
- 4、当 master 发生故障，需手动将其中一台 slave 使用 slaveof no one 命令提升为 master，其它 slave 执行 slaveof 命令指向这个新的 master，从新的 master 处同步数据
- 5、主从复制模式的故障转移需要手动操作，要实现自动化处理，这就需要 Sentinel 哨兵，实现故障自动转移

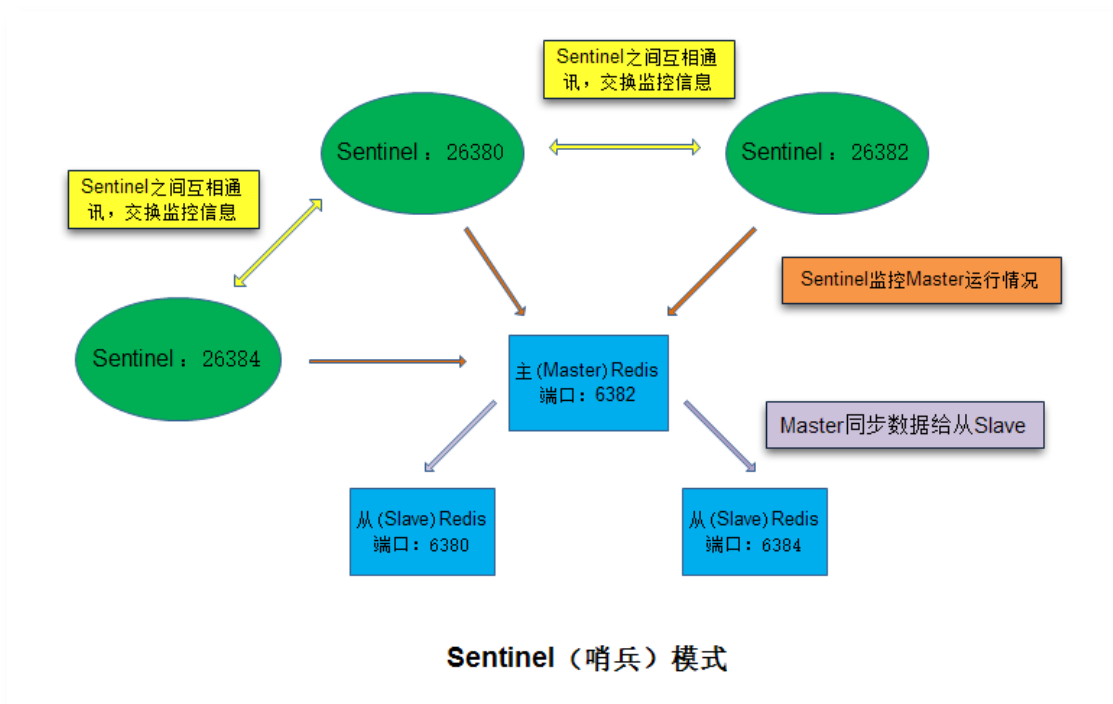
4.3.2 高可用 Sentinel 哨兵

Sentinel 哨兵是 redis 官方提供的高可用方案，可以用它来监控多个 Redis 服务实例的运行情况。Redis Sentinel 是一个运行在特殊模式下的 Redis 服务器。Redis Sentinel 是在多个 Sentinel 进程环境下互相协作工作的。

Sentinel 系统有三个主要任务：

- 监控：Sentinel 不断的检查主服务和从服务器是否按照预期正常工作。
- 提醒：被监控的 Redis 出现问题时，Sentinel 会通知管理员或其他应用程序。
- 自动故障转移：监控的主 Redis 不能正常工作，Sentinel 会开始进行故障迁移操作。将

一个从服务器升级新的主服务器。 让其他从服务器挂到新的主服务器。同时向客户端提供新的主服务器地址。



(1) Sentinel 配置

1) Sentinel 配置文件

复制三份sentinel.conf文件：

```
[root@localhost redis-3.2.9]# pwd
/usr/local/redis-3.2.9
[root@localhost redis-3.2.9]# ll *.conf
-rw-r--r--. 1 root root 46696 Aug 23 14:40 bak_redis.conf
-rw-r--r--. 1 root root 139 Aug 23 18:04 redis6380.conf
-rw-r--r--. 1 root root 162 Aug 23 18:09 redis6382.conf
-rw-r--r--. 1 root root 163 Aug 23 18:10 redis6384.conf
-rw-rw-r--. 1 root root 46721 Aug 23 15:48 redis.conf
-rw-rw-r--. 1 root root 7606 May 17 23:39 sentinel.conf
[root@localhost redis-3.2.9]#
```

Redis安装主目录下的哨兵配置文件

Sentinel系统默认 port 是 26379 。三个配置port分别设置为 26380 , 26382 , 26384 。三个文件分别命名：

- sentinel26380.conf
- sentinel26382.conf

- sentinel26384.conf

执行复制命令 cp sentinel.conf xxx.conf

```
[root@localhost redis-3.2.9]# cp sentinel.conf sentinel26380.conf
[root@localhost redis-3.2.9]# cp sentinel.conf sentinel26382.conf
[root@localhost redis-3.2.9]# cp sentinel.conf sentinel26384.conf
```

(2) 三份 sentinel 配置文件修改:

1、修改 port 26380、 port 26382、 port 26384

2、修改 sentinel monitor mymaster 127.0.0.1 6382 2

格式: sentinel monitor <name> <masterIP> <masterPort> <Quorum 投票数>

Sentinel监控主(Master)Redis, Sentinel根据Master的配置自动发现Master的Slave,Sentinel默认端口号为26379。

```
# port <sentinel-port>
# The port that this sentinel instance will run on
port 26379
```

sentinel26380.conf

1) 修改 port

```
# port <sentinel-port>
# The port that this sentinel instance will run on
port 26380
```

2) 修改监控的 master 地址

```
# The valid charset is A-z 0-9 and the three characters "-_.".
sentinel monitor mymaster 127.0.0.1 6382 2
```

固定的命令行 可以修改名称 master的ip master的port 投票数量

sentinel26382.conf 修改port 26382 , master的port 6382

sentinel26384.conf 修改port 26384 , master的port 6382

(3) 启动主从 (Master/Slave) Redis

启动 Reids

```
[root@localhost src]# ./redis-server ../redis6380.conf
[root@localhost src]# ./redis-server ../redis6382.conf
[root@localhost src]# ./redis-server ../redis6384.conf
[root@localhost src]# ps -ef | grep redis
root      4341      1  0 22:52 ?        00:00:00 ./redis-server *:6380
root      4345      1  0 22:52 ?        00:00:00 ./redis-server *:6382
root      4351      1  0 22:52 ?        00:00:00 ./redis-server *:6384
root      4365     3866  0 22:52 pts/0    00:00:00 grep --color=auto redis
```

查看 Master 的配置信息

连接到 6382 端口

```
127.0.0.1:6382>
[root@localhost src]# ./redis-cli -p 6382
```

使用 info 命令查看 Master/Slave

```
127.0.0.1:6382> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=408,lag=0
slave1:ip=127.0.0.1,port=6384,state=online,offset=408,lag=1
```

(4) 启动 Sentinel

redis安装时make编译后就产生了redis-sentinel程序文件，可以在一个redis中运行多个sentinel进程。

启动一个运行在Sentinel模式下的Redis服务实例

./redis-sentinel sentinel 配置文件

执行以下三条命令，将创建三个监视主服务器的Sentinel实例：

```
./redis-sentinel ../sentinel26380.conf
./redis-sentinel ../sentinel26382.conf
./redis-sentinel ../sentinel26384.conf
```

在 XShell 开启三个窗口分别执行：


```
[root@localhost src]#  
[root@localhost src]# ./redis-sentinel ../sentinel26380.conf  
4568:X 24 Aug 23:02:35.174 * Increased maximum number of open fil  
  
Redis 3.2.9 (00000000/0)  
Running in sentinel mode  
Port: 26380  
PID: 4568
```

```
[root@localhost ~]# cd /usr/local/redis-3.2.9/src/  
[root@localhost src]# ./redis-sentinel ../sentinel26382.conf  
4627:X 24 Aug 23:03:24.881 * Increased maximum number of open fil  
  
Redis 3.2.9 (00000000/0)  
Running in sentinel mode  
Port: 26382  
PID: 4627
```

```
[root@localhost ~]# cd /usr/local/redis-3.2.9/src/  
[root@localhost src]# ./redis-sentinel ../sentinel26384.conf  
4675:X 24 Aug 23:04:01.495 * Increased maximum number of open fil  
  
Redis 3.2.9 (00000000/0)  
Running in sentinel mode  
Port: 26384  
PID: 4675
```

(5) 主 Redis 不能工作

让 Master 的 Redis 停止服务， 执行 shutdown

先执行 info replication 确认 Master 的 Redis ， 再执行 shutdown

```
127.0.0.1:6382> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=60946,lag=1
slave1:ip=127.0.0.1,port=6384,state=online,offset=61212,lag=1
master_repl_offset:61212
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:325
repl_backlog_histlen:60888
127.0.0.1:6382> shutdown
not connected>
```

查看当前 Redis 的进程情况

```
[root@localhost src]# ps -ef | grep redis
root      4341      1   0 22:52 ?        00:00:03 ./redis-server *:6380
root      4351      1   0 22:52 ?        00:00:03 ./redis-server *:6384
root      4568    4418   0 23:02 pts/1    00:00:03 ./redis-sentinel *:26380 [sentinel]
root      4627    4575   0 23:03 pts/2    00:00:03 ./redis-sentinel *:26382 [sentinel]
root      4675    4634   0 23:04 pts/3    00:00:02 ./redis-sentinel *:26384 [sentinel]
root      4827    3866   0 23:12 pts/0    00:00:00 grep --color=auto redis
```

(6) Sentinel 的起作用

在 Master 执行 shutdown 后，稍微等一会 Sentinel 要进行投票计算，从可用的 Slave 选举新的 Master。

查看 Sentinel 日志，三个 Sentinel 窗口的日志是一样的。

```
+switch-master mymaster 127.0.0.1 6382 127.0.0.1 6384
+slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster 127.0.0.1 6384
+slave slave 127.0.0.1:6382 127.0.0.1 6382 @ mymaster 127.0.0.1 6384
```

查看新的 Master

```
127.0.0.1:6384> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=6380,state=online,
```

查看原 Slave 的变化

```
[root@localhost src]# ./redis-cli -p 6380
127.0.0.1:6380> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6384
```

(7) 新的 Redis 加入 Sentinel 系统，自动加入 Master

重新启动 6382

```
[root@localhost src]# ./redis-server ../redis6382.conf
[root@localhost src]#
```

查看 6384 的信息

```
[root@localhost src]# ./redis-cli -p 6384
127.0.0.1:6384> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,
slave1:ip=127.0.0.1,port=6382,state=online,
```

测试数据：在 Master 写入数据

```
127.0.0.1:6384> set m84 v84
OK
127.0.0.1:6384> 在Master写入数据
```

在 6382 上读取数据，不能写入

```
127.0.0.1:6382> get m84
"v84"
127.0.0.1:6382> set sla82 myvalue
(error) READONLY You can't write against a read only slave.
127.0.0.1:6382>
```

(8) 监控

- 1) Sentinel 会不断检查 Master 和 Slave 是否正常
- 2) 如果 Sentinel 挂了, 就无法监控, 所以需要多个哨兵, 组成 Sentinel 网络, 一个健康的 Sentinel 至少有 3 个 Sentinel 应用。彼此在独立的物理机器或虚拟机。
- 3) 监控同一个 Master 的 Sentinel 会自动连接, 组成一个分布式的 Sentinel 网络, 互相通信并交换彼此关于被监控服务器的信息
- 4) 当一个 Sentinel 认为被监控的服务器已经下线时, 它会向网络中的其它 Sentinel 进行确认, 判断该服务器是否真的已经下线
- 5) 如果下线的服务器为主服务器, 那么 Sentinel 网络将对下线主服务器进行自动故障转移, 通过将下线主服务器的某个从服务器提升为新的主服务器, 并让其从服务器转移到新的主服务器下, 以此来让系统重新回到正常状态
- 6) 下线的旧主服务器重新上线, Sentinel 会让它成为从, 挂到新的主服务器下

(9) 总结

主从复制, 解决了读请求的分担, 从节点下线, 会使得读请求能力有所下降, Master 下线, 写请求无法执行

Sentinel 会在 Master 下线后自动执行故障转移操作, 提升一台 Slave 为 Master, 并让其它 Slave 成为新 Master 的 Slave

4.4 安全设置

(1) 设置密码

访问 Redis 默认是没有密码的, 这样不安全, 任意用户都可以访问。可以启用使用密码才能访问 Redis。设置 Redis 的访问密码, 修改 redis.conf 中这行 `requirepass` 密码。密码要比较复杂, 不容易破解, 而且需要定期修改。因为 redis 速度相当快, 所以在一台比较好的服务器下, 一个外部的用户可以在一秒钟进行 150K 次的密码尝试, 需要指定非常非常强大的密码来防止暴力破解。

A、开启访问密码设置

修改 redis.conf, 使用 vim 命令。找到 `requirepass` 行去掉注释, `requirepass` 空格后就是密码。

例 1: 设置访问密码是 123456, 这是练习使用, 生产环境要设置复杂密码

修改 redis.conf, 文件 480 行左右。原始内容:

```
#
# requirepass foobared
```

修改后:

```
#
requirepass 123456
```

查看修改结果:

```
[root@localhost redis-3.2.9]# cat redis.conf | grep requirepass
# If the master is password protected (using the "requirepass" configuration
requirepass 123456
[root@localhost redis-3.2.9]#
```

B、访问有密码的 Redis

如果 Redis 已经启动，关闭后，重新启动。

访问有密码的 Redis 两种方式:

- ①: 在连接到客户端后，使用命令 `auth 密码`，命令执行成功后，可以正常使用 Redis
- ②: 在连接客户端时使用 `-a 密码`。例如 `./redis-cli -h ip -p port -a password`

启动 Redis

```
[root@localhost src]# pwd
/usr/local/redis-3.2.9/src
[root@localhost src]# ps -ef | grep redis
root      3630   3251    0 16:12 pts/0    00:00:00 grep --color=auto redis
[root@localhost src]# ./redis-server ../redis.conf &
[1] 3635
[root@localhost src]# 3635:M 27 Aug 16:13:18.555 * Increased maximum number
( . . . . . )
Redis 3.2.9 (00000000/0) 64 bit
Running in standalone mode
```

使用 ① 访问

```
[root@localhost src]#
[root@localhost src]# ./redis-cli -p 6379
127.0.0.1:6379> set myk1 v1
(error) NOAUTH Authentication required.
127.0.0.1:6379>
127.0.0.1:6379> 连接成功，操作失败
```

输入命令 `auth 密码`

```
(error) NOAUTH Authentication required.
127.0.0.1:6379>
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> set myk1 v1
OK
127.0.0.1:6379>
```

使用 ② 方式

```
[root@localhost src]#
[root@localhost src]# ./redis-cli -p 6379 -a 123456
127.0.0.1:6379> get myk1
"v1"
```

(2) 绑定 ip

修改 redis.conf 文件，把# bind 127.0.0.1 前面的注释#号去掉，然后把 127.0.0.1 改成允许访问你 redis 服务器的 ip 地址，表示只允许该 ip 进行访问。多个 ip 使用空格分隔。

例如 bind 192.168.1.100 192.168.2.10

```
# bind 127.0.0.1
去掉#，bind是可以访问reids的ip
多个ip，使用空格分隔
```

(3) 修改默认端口

修改 redis 的端口，这一点很重要，使用默认的端口很危险，redis.conf 中修改 port 6379 将其修改为自己指定的端口（可随意），端口 1024 是保留给操作系统使用的。用户可以使用的范围是 1024-65535

```
# Accept connections on the specified port:
# If port 0 is specified Redis will use any available port
port 6379
```

使用 -p 参数指定端口，例如：./redis-cli -p 新设置端口

第5章 Jedis 操作 Redis

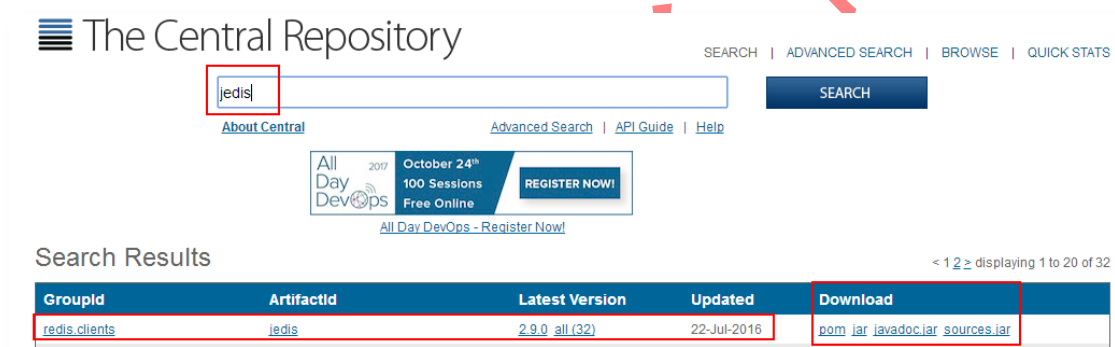
使用 Redis 官方推荐的 Jedis，在 java 应用中操作 Redis。Jedis 几乎涵盖了 Redis 的所有命令。操作 Redis 的命令在 Jedis 中以方法的形式出现。jedis 完全兼容 redis 2.8.x and 3.x.x

- Jedis 源码: <https://github.com/xetorthio/jedis>
- api 文档: <http://xetorthio.github.io/jedis/>
- 下载: <http://search.maven.org/>，搜索 jedis

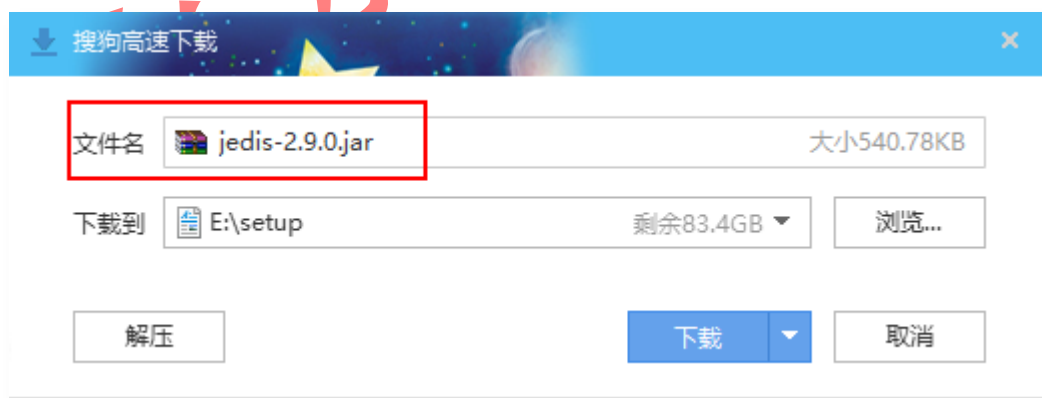
5.1 下载 Jedis 和 Commons-Pool

5.1.1 下载 Jedis

浏览器打开: <http://search.maven.org/>，搜索 jedis。在 Download 处，点击 jar



最新的版本 jedis-2.9.0



5.1.2 下载 Commons-Pool

Jedis 对象并不是线程安全的，在多线程下使用同一个 Jedis 对象会出现并发问题。为了避免每次使用 Jedis 对象时都需要重新构建，Jedis 提供了 JedisPool。JedisPool 是基于 Commons Pool 2 实现的一个线程安全的连接池

浏览器打开：<http://search.maven.org/>，搜索 commons-pool2。在 Download 处，点击 jar



点击“jar”

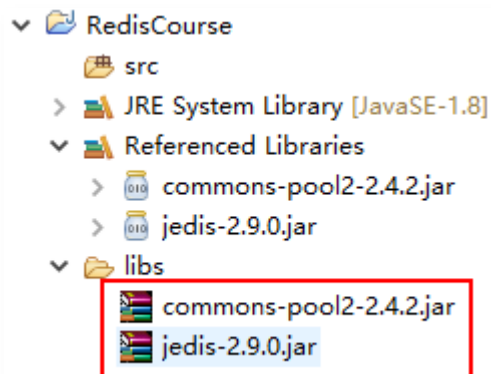


5.2 Java 应用使用 Jedis 准备

项目中加入 jar:

- jedis-2.9.0.jar
- commons-pool2-2.4.2.jar

加入项目后如图:



5.2.1 字符串(string)

```
public class RedisString {
    public static void main(String[] args) {
        //创建Jedis对象，连接到Redis，需要提供ip和port
        Jedis jedis = new Jedis("127.0.0.1",6379);
        //添加字符串
        jedis.set("breakfast", "豆浆和包子");
        String mybreak = jedis.get("breakfast");
        System.out.println("我的早餐1: "+mybreak);

        //追加内容
        jedis.append("breakfast", "还有鸡蛋");
        mybreak = jedis.get("breakfast");
        System.out.println("我的早餐2: "+mybreak);

        //一次设置多个key-value
        jedis.mset("lunch", "红烧肉", "dinner", "牛肉面");
        //获取多个key的value
        List<String> dinners = jedis.mget("lunch", "dinner");
        for(String din : dinners){
            System.out.println("我吃的是"+din);
        }
    }
}
```

5.2.2 哈希 (hash)

A、使用 Jedis 连接实例池。

```
public class RedisUtils {  
    // 定义连接池对象  
    private static JedisPool pool = null;  
    // 创建连接池  
    public static JedisPool open(String host, int port) {  
        if (pool == null) {  
            // 使用JedisPool  
            JedisPoolConfig config = new JedisPoolConfig();  
            // 最大的Jedis实例数（连接池中是Jedis实例，默认是8  
            config.setMaxTotal(10);  
            // 最大的空闲实例数，设置这个可以保留足够的连接，快速的获取到Jedis对象  
            config.setMaxIdle(3);  
            // 提前检查Jedis对象，为true获取的Jedis一定是可用的  
            config.setTestOnBorrow(true);  
            // 创建Jedis连接池，Redis没有访问密码时的使用方式  
            // pool = new JedisPool(config, host, port);  
            /*  
             * 创建Jedis连接池，Redis有访问密码时的使用方式  
             * 参数： JedisPoolConfig Redis的主机地址 端口 连接超时时间单位毫秒 访问密码  
             */  
            pool = new JedisPool(config, host, port, 6 * 1000, "123456");  
        }  
        return pool;  
    }  
    // 关闭连接池  
    public static void close() {  
        if (pool != null) {  
            pool.close();  
        }  
    }  
}
```

B、使用连接池操作 hash 数据类型

```
@Test
public void test01() {
    //创建连接池
    JedisPool pool = RedisUtils.open("127.0.0.1", 6379);
    Jedis jedis = null;
    try{
        //从连接池中获取Jedis对象
        jedis = pool.getResource();
        //设置 hash类型。key:loginuser , field:username ,value:zhangsang
        jedis.hset("loginuser", "username", "zhangsang");
        System.out.println("username的值"+jedis.hget("loginuser", "username"));
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        if( jedis != null){
            //使用完的连接池对象，放回连接池
            jedis.close();
        }
    }
}
```

```
@Test
public void test02(){
    JedisPool pool = RedisUtils.open("127.0.0.1", 6379);
    Jedis jedis = null;
    try{
        jedis = pool.getResource();
        //jedis.hmset(String, Map<String,String>)
        Map<String,String> map = new HashMap<String,String>();
        map.put("username", "bjpowernode");
        map.put("age", "20");
        map.put("website", "www.bjpowernode.com");
        //设置多个值, 使用Map, 存入Redis
        jedis.hmset("logininfo", map);
        //从Redis取hash数据
        List<String> fieldValues = jedis.hmget("logininfo", "username", "website");
        for(String fv : fieldValues){
            System.out.println("field值: "+fv);
        }
        //存在 username吗?
        System.out.println("返回boolean, 存在username:"+jedis.exists("logininfo", "username"));
        //查看所有的field
        Set<String> set = jedis.hkeys("logininfo");
        Iterator<String> iter = set.iterator();
        while(iter.hasNext()){
            System.out.println("field name:"+ iter.next());
        }
        //删除feild : age
        jedis.hdel("logininfo", "age");
        //获取age
        System.out.println("age是 null:"+jedis.hget("logininfo", "age"));
        //所有field的数量
        System.out.println("feild个数: "+jedis.hlen("logininfo"));
    }finally{
        if( jedis !=null){
            jedis.close();
        }
    }
}
```

5.2.3 列表 list

```
@Test
public void test03(){
    Jedis jedis = new Jedis("127.0.0.1", 6379);
    String key="framework";
    jedis.del(key);
    jedis.lpush(key, "mybatis");
    jedis.lpush(key, "hibernate", "spring", "springmvc");
    List<String> lists= jedis.lrange(key, 0, -1);
    for(String str:lists){
        System.out.println("列表数据: "+str);
    }
    System.out.println("列表长度: "+jedis.llen(key));
    System.out.println("插入新值后列表长度:"+jedis.linsert(key, LIST_POSITION.AFTER, "spring", "jpa"));
    //从列表右侧插入数据
    jedis.rpush(key, "struts", "webwork");
    System.out.println("列表数据: "+jedis.lrange(key, 0, -1));
    //列表数据: [springmvc, spring, jpa, hibernate, mybatis, struts, webwork]
    System.out.println("第 1 个下标的值: "+jedis.lindex(key, 1)); //spring

    for(long i=0, len = jedis.llen(key); i<len; i++){
        System.out.println("弹出值: "+jedis.lpop(key));
    }
}
```

5.2.4 集合 Set

```
@Test
public void test03(){
    Jedis jedis = new Jedis("127.0.0.1",6379);
    String key="course";
    //添加一个数据
    jedis.sadd(key, "html");
    //添加多个数据
    jedis.sadd(key, "css","javascript","mysql","spring");
    Set<String> sets=jedis.smembers(key); //返回集合的所有成员
    Iterator<String> iter = sets.iterator();
    while(iter.hasNext()){
        System.out.println("集合Set成员: "+iter.next());
    }
    //判断struts是否在集合中
    System.out.println("struts有吗? "+jedis.sismember(key, "struts")); //false
    //集合的成员数量
    System.out.println("集合成员个数: "+jedis.scard(key)); //5
}
```

5.2.5 有序集合 Sorted Set

```
@Test
public void test03(){
    Jedis jedis = new Jedis("127.0.0.1",6379);
    String key="salary";
    jedis.del(key);
    jedis.zadd(key, 2000D, "John");
    //使用方法: zadd(String key,Map<String,Double>) 添加多个数据
    Map<String,Double> map = new HashMap<String,Double>();
    map.put("Tom", 3500D);
    map.put("Marry", 6500D);
    map.put("Rose", 3600D);
    map.put("Mike", 5060D);
    //添加多个数据
    jedis.zadd(key, map);
    //查询返回全部的数据, 没有score
    Set<String> sets=jedis.zrangeByScore(key, "-inf", "+inf");
    Iterator<String> iter= sets.iterator();
    while(iter.hasNext()){
        System.out.println("排序小-大的成员: "+iter.next());
    }
    //带有score的数据
    Set<Tuple> tuple= jedis.zrangeByScoreWithScores(key, "-inf", "+inf");
    Iterator<Tuple> iters = tuple.iterator();
    while(iters.hasNext()){
        Tuple tu = iters.next();
        System.out.println("排序小-大的成员: "+tu.getElement()+"#score:"+tu.getScore());
    }
    System.out.println("有序集合成员数量: "+jedis.zcard(key));
}
```

5.2.6 事务 (Transaction)

```
public class RedisTransaction {
    public static void main(String[] args) {
        //创建Jedis对象，连接到Redis，需要提供ip和port
        JedisPool pool = RedisUtils.open("127.0.0.1",6379);
        Jedis jedis = null;
        try{
            //开启事务
            Transaction trans = jedis.multi();
            //添加字符串
            trans.set("breakfast", "豆浆和包子");
            //一次设置多个key-value
            trans.mset("lunch","红烧肉","dinner","牛肉面");
            List<Object> resultList = trans.exec();
            //事务的处理结果
            for(Object result : resultList){
                System.out.println("成功的事务操作: "+result);
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            if( jedis != null){
                jedis.close();
            }
        }
    }
}
```

支持部分事务操作