Redis

Redis是什么?

Redis（`Re`mote `Di`ctionary `S`erver），即远程字典服务，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日志型、Key-Value**数据库**，并提供多种语言的API。

Redis能做什么?

- 缓存
- 排行榜
- 计数器/限速器(统计播放数据/浏览量/在线人数等)
- 好友关系（点赞）
- 简单的消息队列（订阅发布、堵塞队列）
- Session服务器

为什么使用Redis

> **速度快**

1. C语言实现，接近系统进性操作
2. 数据存储在内存中
3. 单线程，避免线程切换开销以及多线程的安全问题（JUC）
4. 采用epoll，非堵塞I/O，不在网络上浪费时间

> **支持多种数据类型**

1. String
2. Hash
3. List
4. Set
5. ZSet（有序集合）
6. Bitmaps(位图)
7. HyperLogLog(技术统计算法)
8. GEO(地理信息定位)

> **功能丰富**

1. 可设置键过期
2. 基于发布订阅可实现简单的消息队列
3. 通过Lua创建新命令，具有原子性
4. Pipeline功能，减少网络开销

> **服务器简单**

1. 代码优雅
2. 参与单线程模型
3. 不依赖操作系统类库

> **客户端语言多**

Java/PHP/Golang

> **支持持久化**

1. RDB
2. AOF

> **主从复制，高可用/分布式**

下载Redis

下载的Redis是基于Linux系统的Redis

> 官网： **https://redis.io/**

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. Learn more →

Try it

Ready for a test drive? Check this interactive tutorial that will walk you through the most important features of Redis.

Download it

Redis 6.0.8 is the latest stable version. Interested in release candidates or unstable versions? Check the downloads page.

Quick links

Follow day-to-day Redis on Twitter and GitHub. Get help or help others by subscribing to our mailing list, we are 5,000 and counting!

> 中文官网： **http://www.redis.cn/**

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。 Redis 内置了 复制（replication），LUA脚本（Lua scripting）， LRU驱动事件（LRU eviction），事务（transactions） 和不同级别的 磁盘持久化（persistence）， 并通过 Redis哨兵（Sentinel）和自动 分区（Cluster）提供高可用性（high availability）。

查看Redis命令大全 →
访问Redis论坛 →
Redis使用内存计算器 →

试用（Try it）

准备使用Redis? 通过 互动教程（interactive tutorial） 将引导您了解Redis最重要的特征。

下载（Download it）

最新稳定版本是redis 6.0.6 想了解更多候选版本或者测试版本？ 点击这里查看更多。

链接（Quick links）

在Twitter和 GitHub查看与Redis同步更新的更多资料。 通过订阅我们的邮件列表获取帮助或者帮助其他人，现在订阅人数已经超过5000人，并且还在不断增加哦。

安装Redis

非稳定版

This is where all the development happens. Only for hard-core hackers. Use only if you need to test the latest features or performance improvements. This is going to be the next Redis release in a few months.

Download unstable

稳定版 (5.0)

Redis 5.0 是第一个加入流数据类型（stream data type ）的版本，sorted sets blocking pop operations, LFU/LRU info in RDB, Cluster manager inside redis-cli, active defragmentation V2, HyperLogLogs improvements and many other improvements. Redis 5 was release as GA in October 2018.

Release notes

Download 6.0.6

Docker

It is possible to get Docker images of Redis from the Docker Hub. Multiple versions are available, usually updated in a short time after a new release is available.

Download

> 下载的版本：redis-6.0.6.tar.gz

> 安装Redis

1. `tar -zxvf redis-6.0.6.tar.gz` 默认目录在/opt/redis-6.0.6
2. 将目录移动到/usr/local/目录下 `mv /opt/redis-6.0.6 /usr/local/`
3. `cd redis-6.0.6`
4. `make`
5. 安装可能会报错：

```
[root@localhost redis-6.0.6]# make
cd src && make all
make[1]: 进入目录“/usr/local/redis-6.0.6/src”
    CC adlist.o
/bin/sh: cc: 未找到命令
make[1]: *** [Makefile:315：adlist.o] 错误 127
make[1]: 离开目录“/usr/local/redis-6.0.6/src”
make: *** [Makefile:6：all] 错误 2
[root@localhost redis-6.0.6]# ls- l
bash: ls-: 未找到命令...
```

需要安装gcc -c++的环境

`yum -y install gcc-c++`

然后再次`make`，报错如下：

```
[root@localhost redis-6.0.6]# make
cd src && make all
make[1]: 进入目录“/usr/local/redis-6.0.6/src”
    CC Makefile.dep
    CC adlist.o
In file included from adlist.c:34:
zmalloc.h:50:10: 致命错误：jemalloc/jemalloc.h：没有那个文件或目录
 #include <jemalloc/jemalloc.h>
          ^~~~~~~~~~~~~~~~~~~~~
编译中断。
make[1]: *** [Makefile:315：adlist.o] 错误 1
make[1]: 离开目录“/usr/local/redis-6.0.6/src”
make: *** [Makefile:6：all] 错误 2
[root@localhost redis-6.0.6]# /
```

执行`make MALLOC=libc`

进入到redis的目录，然后执行`make install`执行 安装，就会在/usr/local/bin/产生redis的命令

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
[root@localhost ~]# cd /usr/local/bin/
[root@localhost bin]# ls -l
总用量 18180
-rw-r--r--. 1 root root       92 9月   20 09:55 dump.rdb
-rwxr-xr-x. 1 root root   788120 9月   20 09:44 redis-benchmark
-rwxr-xr-x. 1 root root  5572048 9月   20 09:44 redis-check-aof
-rwxr-xr-x. 1 root root  5572048 9月   20 09:44 redis-check-rdb
-rwxr-xr-x. 1 root root  1094592 9月   20 09:44 redis-cli
lrwxrwxrwx. 1 root root       12 9月   20 09:44 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root  5572048 9月   20 09:44 redis-server
[root@localhost bin]#
```

6. 将redis的安装目下的redis-conf复制一份到根目录 /mycnf/redis-conf

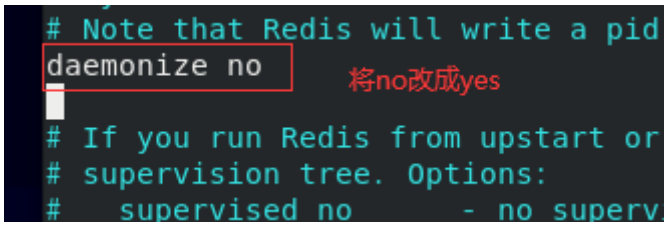
```
[root@localhost redis-6.0.6]# cp redis.conf /mycnf/redis.conf
```

7. 然后在任何目录都可以执行以下命令启动Redis
   - `redis-server /mycnf/redis.conf`
   - **每次启动redis的服务时，就不要使用安装目录下的配置文件启动，使用我们copy的那一份，可以在/mycnf/redis-conf目录下修改配置，而不影响原来的最初配置文件**

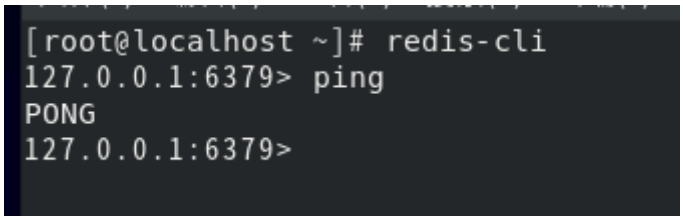
```
                _.-``    `.  `_.  ''-._           Redis 6.0.6 (00000000/0) 64 bit
            .-`` .-```.  ```\/    _.,_ ''-._
       (    '      ,       .-`  | `,    )     Running in standalone mode
       |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
       |    `-._   `._    /     _.-'    |     PID: 7031
       `-._    `-._  `-./  _.-'    _.-'
       |`-._`-._    `-.__.-'    _.-'_.-'|
       |    `-._`-._        _.-'_.-'    |           http://redis.io
       `-._    `-._`-.__.-'_.-'    _.-'
       |`-._`-._    `-.__.-'    _.-'_.-'|
       |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
           `-._    `-.__.-'    _.-'
               `-._        _.-'
                   `-.__.-'

'031:M 19 Sep 2020 19:58:21.618 # WARNING: The TCP backlog setting of 511 ca
:he lower value of 128.
'031:M 19 Sep 2020 19:58:21.618 # Server initialized
'031:M 19 Sep 2020 19:58:21.618 # WARNING overcommit_memory is set to 0! Bac
:ue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or ru
'fect.
'031:M 19 Sep 2020 19:58:21.618 # WARNING you have Transparent Huge Pages (T
 memory usage issues with Redis. To fix this issue run the command 'echo nev
ld it to your /etc/rc.local in order to retain the setting after a reboot. R
'031:M 19 Sep 2020 19:58:21.618 * Loading RDB produced by version 6.0.6
'031:M 19 Sep 2020 19:58:21.618 * RDB age 217 seconds
'031:M 19 Sep 2020 19:58:21.618 * RDB memory usage when created 0.85 Mb
'031:M 19 Sep 2020 19:58:21.618 * DB loaded from disk: 0.000 seconds
'031:M 19 Sep 2020 19:58:21.618 * Ready to accept connections
```

- **启动服务后，不要关掉窗口，否则就没法连接上**redis
- `采用后台启动Redis，关闭窗口也不会让服务停止`
  - 在/mycnf/redis.conf文件中修改



  - 然后重启服务 `redis-server /mycnf/redis.conf`
- 你可以使用内置的客户端，在任何目录都可以执行以下redis命令
  - `redis-cli`



redis的端口号：6379?

redis默认有16个数据库，默认是编号为0的数据库0~15

key-value保存值

Redis : key -[key-value]

五大数据类型

String（字符串）

```
#数据库的编号从0 开始
127.0.0.1:6379> config get databases  #查看redis有多少个数据库，默认16个，在配置文件也能看到
1) "databases"
2) "16"
#使用select切换数据库
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> select 2 #切换数据库后，可以从端口号的后面的看出目前是哪个数据库？
OK
127.0.0.1:6379[2]> select 3
OK
127.0.0.1:6379[3]> select 0
OK
#=============================================================
#清空数据库:FLUSHDB
127.0.0.1:6379> FLUSHDB #清空数据库
OK
127.0.0.1:6379> keys *
(empty array) #显示的是空集
#=============================================================
127.0.0.1:6379> keys *  #查询当前数据库所有存储的key（键）
(empty array)
127.0.0.1:6379> set hello world #set 设置key-value
OK
127.0.0.1:6379> get hello #get 通过get key去获取value
"world"
127.0.0.1:6379> keys *
1) "hello"
#=============================================================
#拼接字符串：APPEND
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379> set hello wolrd
OK
127.0.0.1:6379> get hello
"wolrd"
127.0.0.1:6379> APPEND hello myworld   # APPEND 在value后面追加字符串
(integer) 12 #返回的是整个value的字符串长度
127.0.0.1:6379> get hello
"wolrdmyworld"
#获取字符串的长度：STRLEN
127.0.0.1:6379> STRLEN hello # STRLEN key 获取value的字符创长度
(integer) 12 #返回长度
#=============================================================
#截取字符串：GETRANGE
127.0.0.1:6379> set hello "my world and you"
OK
127.0.0.1:6379> get hello
"my world and you"
127.0.0.1:6379> GETRANGE hello 3 7 #GETRANGE 可以根据字符串的索引截取子字符串
"world"  #返回的是截取后的字符串
127.0.0.1:6379> get hello
```

```
"my world and you"
127.0.0.1:6379> getrange hello  0 1   #截取hello的值，从0到1
"my"
127.0.0.1:6379> getrange hello  0 -1  #取hello的全部字符串
"my world and you"
127.0.0.1:6379>

#=========================================
#替换字符串：SETRANGE
127.0.0.1:6379> set hello "I am laowan"
OK
127.0.0.1:6379> get hello
"I am laowan"
127.0.0.1:6379> SETRANGE hello 5 "wandaye"  #从索引为5开始，使用后面的值进行替换
(integer) 12
127.0.0.1:6379> get hello
"I am wandaye"
127.0.0.1:6379>
#====================================
#一次性存储多个值：MSET
127.0.0.1:6379> MSET key1 value1 key2 value2 key3 value3 #MSET key value.....
OK
127.0.0.1:6379> keys *
1) "key3"
2) "key1"
3) "key2"
127.0.0.1:6379> get key2
"value2"
127.0.0.1:6379> mget key1 key2 key3 #一次性获取多个值：mget key value .....
1) "value1"
2) "value2"
3) "value3"
127.0.0.1:6379>
#==================================
#设置值得过期时间，以秒为单位 setex
# 获取过期时间：TTL
127.0.0.1:6379> setex hello 10 world #设置hello的值world的过期时间为10秒
OK
127.0.0.1:6379> ttl hello  #获取hello对应的值过期时间，以秒为单位，PTTL：是以毫秒为单位
(integer) 7
127.0.0.1:6379> ttl hello
(integer) 6
127.0.0.1:6379> ttl hello
(integer) 5
127.0.0.1:6379> ttl hello
(integer) 4
127.0.0.1:6379> ttl hello
(integer) 3
127.0.0.1:6379> ttl hello
(integer) 2
127.0.0.1:6379> ttl hello
(integer) 2
127.0.0.1:6379> ttl hello
(integer) 1
127.0.0.1:6379> ttl hello
(integer) 0
127.0.0.1:6379> ttl hello  #返回值为-2说明就过期了
(integer) -2
```

```
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> get hello #过期后，就获取不到值了
(nil)
127.0.0.1:6379>
#================================
#判断key是否存在？EXISTS
127.0.0.1:6379> set hello world
OK
127.0.0.1:6379> EXISTS hello #通过key判断是否存在，存在返回1
(integer) 1
127.0.0.1:6379> EXISTS world  #不存在，返回0
(integer) 0
127.0.0.1:6379>
#================================
#通过key删除value：DEL
127.0.0.1:6379> keys *
1) "hello"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> DEL hello  #通过key删除指定value，返回1代表删除成功
(integer) 1
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379>
#==================================
127.0.0.1:6379> set hello world
OK
127.0.0.1:6379> MOVE hello 1  # MOVE将key移动到另外一个数据库，移动成功返回1
(integer) 1
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379> select 1  #切换数据库
OK
127.0.0.1:6379[1]> keys *
1) "hello"
127.0.0.1:6379[1]> FLUSHALL  #删除所有数据库内的数据
OK
127.0.0.1:6379[1]> keys *
(empty array)
127.0.0.1:6379[1]> selct 0  #回到索引为0的数据库
OK
#==========================
127.0.0.1:6379> set hello world
OK
127.0.0.1:6379> EXPIRE hello 10  #为指定的key设置过期时间，单位为秒
(integer) 1
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379>
#================================
#统计网站在线人数，分数数量等等功能
#实现i++的功能:INCR
127.0.0.1:6379> set num 0
OK
127.0.0.1:6379> INCR num  # INCR :累加1
```

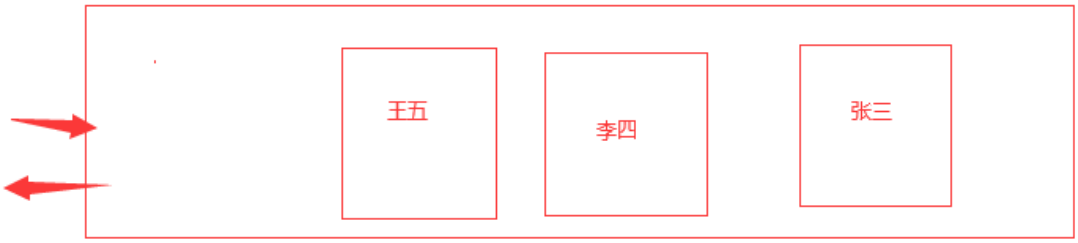
```
(integer) 1
127.0.0.1:6379> INCR num
(integer) 2
127.0.0.1:6379> get num
"2"
127.0.0.1:6379>
#============================
#i+=5;步长：INCRBY
127.0.0.1:6379> get num
"2"
127.0.0.1:6379> INCRBY num 10 #每次incr后，增加10，类似i+=10
(integer) 12
127.0.0.1:6379> INCRBY num 10
(integer) 22
127.0.0.1:6379> get num
"22"
#============================
#实现i--，根据步长减少：DECRBY
27.0.0.1:6379> DECR num  #减少1
(integer) 99
127.0.0.1:6379> DECR num
(integer) 98
127.0.0.1:6379> DECR num
(integer) 97
127.0.0.1:6379> DECRBY num 10
(integer) 87
127.0.0.1:6379> DECRBY num 10 #每次减少10  i-=10
(integer) 77
127.0.0.1:6379> DECRBY num 10
(integer) 67
127.0.0.1:6379> get num
"67"
#============================
127.0.0.1:6379> SETNX hello world  #如果key不存在，就创建，返回1  `保证原子性`
(integer) 1
127.0.0.1:6379> SETNX hello "hello world!"  #如果存在，就返回0,
(integer) 0
127.0.0.1:6379>
#============================
msetnx：保存多个值时，进行原子性操作，如果存在key，就保存失败。如果都不存在key，就会保存成功。
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> MSETNX k1 v1 k2 v2 # 如果存在key，那就就会不创建，保证原子性操作，k1存在，就不会保存k2
(integer) 0  #返回0
127.0.0.1:6379> msetnx k2 v2 k3 v3 #就要全部保存
(integer) 1 #返回1
127.0.0.1:6379> mget k1 k2 k3
1) "v1"
2) "v2"
3) "v3"
127.0.0.1:6379>
#====================================
保存对象
set user:1:{name:zhangsan,age:18} #设置一个user：1对象，值为json字符串保存一个对象
#这里使用 user:{id}:{field}给设置
```

```
127.0.0.1:6379> mset user:1:name zhangsan user:1:age 18
OK
127.0.0.1:6379> get user
(nil)
127.0.0.1:6379> get user:1
(nil)
127.0.0.1:6379> mget user:1
1) (nil)
127.0.0.1:6379> mget user:1:name user:1:age
1) "zhangsan"
2) "18"
127.0.0.1:6379>
#============================================
# getset命令，先get，再set
127.0.0.1:6379> getset k1 v1  #如果不存在k1，就保存k1，返回nil
(nil)
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> getset k1 vv1 #如果存在k1，就获取原来的值，并设置新的值，返回原来的值
"v1"
127.0.0.1:6379> get k1 # 值已经更改了
"vv1"
127.0.0.1:6379>
```

List(列表)

所有的LIst命令都是L开头的

```
#============================
#LPUSH ： 存值
#Lrange:取值
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379> lpush list zhangsan  # 存值操作，保存在第一个位置
(integer) 1 #返回的是list中的有一个元素
127.0.0.1:6379> lpush list lisi
(integer) 2
127.0.0.1:6379> lpush list wangwu
(integer) 3
127.0.0.1:6379> LRANGE list 0 -1 #取值操作
1) "wangwu"
2) "lisi"
3) "zhangsan"
127.0.0.1:6379>
127.0.0.1:6379> lrange list 0 0  #去第一个值，得到wangwu，wangwu是最后放进去的
1) "wangwu"
127.0.0.1:6379>
```



栈：后进先出

```
127.0.0.1:6379> Rpush list one  #将值存放在最后
(integer) 4
127.0.0.1:6379> lrange list 0 -1
1) "wangwu"
2) "lisi"
3) "zhangsan"
4) "one"
127.0.0.1:6379> rpush list two  #将值存放在最后
(integer) 5
127.0.0.1:6379> lrange list 0 -1  #取出list中的全部值
1) "wangwu"
2) "lisi"
3) "zhangsan"
4) "one"
5) "two"
127.0.0.1:6379>
```


```
#==============================
#移除值
127.0.0.1:6379> LPOP list #移除第一个元素
"wangwu"  #返回移除的值
127.0.0.1:6379> LPOP list  #移除第一个元素
"lisi"
127.0.0.1:6379> lrange list 0 -1 #获取剩下的值
1) "zhangsan"
2) "one"
3) "two"
127.0.0.1:6379> RPOP list  #移除最后一个元素
"two"
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
#==============================
#通过下标获取值：LINDEX
127.0.0.1:6379> LRANGE list 0 -1
1) "zhangsan"
2) "one"
127.0.0.1:6379> LINDEX list 0
```

```
"zhangsan"
127.0.0.1:6379>
#==================
#获取list中的元素个数：LLEN
127.0.0.1:6379> LLEN list
(integer) 2
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
127.0.0.1:6379>
#==================
#移除元素：LREM
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
127.0.0.1:6379> RPUSH list one
(integer) 3
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
3) "one"
127.0.0.1:6379> LREM list 1 one  #移除1个one
(integer) 1
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
127.0.0.1:6379>
#====================
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
127.0.0.1:6379> rpush list one
(integer) 3
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
2) "one"
3) "one"
127.0.0.1:6379> lrem list 2 one  #移除2个one
(integer) 2
127.0.0.1:6379> lrange list 0 -1
1) "zhangsan"
127.0.0.1:6379>
#============================
#截取指定下标的元素LTRIM
127.0.0.1:6379> RPUSH list one
(integer) 1
127.0.0.1:6379> RPUSH list two three four
(integer) 4
127.0.0.1:6379> lrange list 0 -1
1) "one"
2) "two"
3) "three"
4) "four"
127.0.0.1:6379> LTRIM list 1 2  #截取list中从下标1开始到2结束的元素，list中的值被改变了
OK
127.0.0.1:6379> lrange list 0 -1 #剩下两个元素
1) "two"
2) "three"
```

```
127.0.0.1:6379>
#================================
#RPOPLPUSH：移除列表最后一个元素的值，并且添加到另外一个列表的第一个位置上
127.0.0.1:6379> rpush list a1 a2 a3 a4
(integer) 4
127.0.0.1:6379> lrange list 0 -1
1) "a1"
2) "a2"
3) "a3"
4) "a4"
127.0.0.1:6379> RPOPLPUSH list list2
"a4"
127.0.0.1:6379> lrange list 0 -1 #原来的列表没有a4
1) "a1"
2) "a2"
3) "a3"
127.0.0.1:6379> lrange list2 0 -1 #新列表中的第一个位置上存在a4
1) "a4"
127.0.0.1:6379>
#=====================================
#LSET :将列表中指定的值替换成新值，如果不存在就会报错
127.0.0.1:6379> lset list 0 a  #列表不存在，报错
(error) ERR no such key
127.0.0.1:6379> lpush list v1
(integer) 1
127.0.0.1:6379> lrange list 0 -1
1) "v1"
127.0.0.1:6379> lset list 0 a   #将列表的第一个元素设置为a
OK
127.0.0.1:6379> lrange list 0 -1
1) "a"
127.0.0.1:6379> lset list 1 b #列表没有下标为1的元素，报错
(error) ERR index out of range
127.0.0.1:6379>
#============================
# LINSERT:在列表的指定位置（前面或者后面）插入新的元素
127.0.0.1:6379> RPUSH list a b c d
(integer) 4
127.0.0.1:6379> lrange list 0 -1
1) "a"
2) "b"
3) "c"
4) "d"
127.0.0.1:6379> LINSERT list before b XX  ##在列表list中b元素前面插入XX
(integer) 5
127.0.0.1:6379> lrange list 0 -1
1) "a"
2) "XX"
3) "b"
4) "c"
5) "d"
127.0.0.1:6379> LINSERT list after c YY #在列表list中的元素c后面插入YY
(integer) 6
127.0.0.1:6379> lrange list 0 -1
1) "a"
2) "XX"
3) "b"
4) "c"
```

```
5) "YY"
6) "d"
127.0.0.1:6379>
```

Set（集合）

set中的值是不能重复的

```
#set 无序，不重复
#====================
#增加元素：SADD
#查看元素：SMEMBER
#判断元素是否存在：SISMEMBER
127.0.0.1:6379> SADD myset "hello"  #增加元素
(integer) 1
127.0.0.1:6379> sadd myset "world" "hi" #一次性增加两个元素
(integer) 2
127.0.0.1:6379> SMEMBERS myset #查看元素
1) "hi"
2) "world"
3) "hello"
127.0.0.1:6379> SISMEMBER myset hello #判断元素是否存在
(integer) 1
127.0.0.1:6379> SISMEMBER myset you
(integer) 0
127.0.0.1:6379> SCARD myset #查看有多少个元素
(integer) 3
127.0.0.1:6379>
#=========================
#移除某一个元素
127.0.0.1:6379> SMEMBERS myset
1) "hi"
2) "world"
3) "hello"
127.0.0.1:6379> srem myset hi  #移除元素
(integer) 1
127.0.0.1:6379> SMEMBERS myset
1) "world"
2) "hello"
127.0.0.1:6379>
#set 无序，不重复
#产生随机元素：SRANDMEMBER,可以实现随机抽奖
127.0.0.1:6379> sadd myset zhangsan lisi wangwu zhaoliu
(integer) 4
127.0.0.1:6379> SMEMBERS myset
1) "wangwu"
2) "zhangsan"
3) "zhaoliu"
4) "lisi"
127.0.0.1:6379> SRANDMEMBER myset #产生随机元素
"zhangsan"
127.0.0.1:6379> SRANDMEMBER myset
"zhaoliu"
127.0.0.1:6379> SRANDMEMBER myset
"lisi"
127.0.0.1:6379> SRANDMEMBER myset
"lisi"
```

```
127.0.0.1:6379> SRANDMEMBER myset 2 #产生指定个数的随机元素=》随机产生两个元素
1) "wangwu"
2) "lisi"
127.0.0.1:6379>
#======================
# 随机移除元素：SPOP
127.0.0.1:6379> SMEMBERS myset
1) "wangwu"
2) "zhangsan"
3) "zhaoliu"
4) "lisi"
127.0.0.1:6379> SPOP myset #随机移除一个元素
"lisi"
127.0.0.1:6379> SMEMBERS myset
1) "wangwu"
2) "zhangsan"
3) "zhaoliu"
127.0.0.1:6379> SPOP myset 2 #随机移除两个元素
1) "zhangsan"
2) "wangwu"
127.0.0.1:6379> SMEMBERS myset
1) "zhaoliu"
127.0.0.1:6379>
#========================
#移动指定元素到另外一个set中：SMOVE
127.0.0.1:6379> sadd myset a b c d
(integer) 4
127.0.0.1:6379> SMEMBERS myset
1) "c"
2) "a"
3) "b"
4) "d"
127.0.0.1:6379> sadd myset2 "x"
(integer) 1
127.0.0.1:6379> SMEMBERS myset2
1) "x"
127.0.0.1:6379> SMOVE myset myset2 "a" #将myset中的a元素移动到myset2中
(integer) 1
127.0.0.1:6379> SMEMBERS myset
1) "c"
2) "b"
3) "d"
127.0.0.1:6379> SMEMBERS myset2
1) "a"
2) "x"
127.0.0.1:6379>
```

Hash(哈希)

map集合 key-value,存储k-v

```
#====================
#在hash中存值：hset
127.0.0.1:6379> HSET map hello world #将（key)hello (value)world 存进map中
(integer) 1
127.0.0.1:6379> hget map hello
"world"
```

```
127.0.0.1:6379> hset map k1 v1 k2 v2 #存值
(integer) 2
127.0.0.1:6379> hset map hello world #存在重复的值
(integer) 0
127.0.0.1:6379> HGETALL map #获取所有的key-value
1) "hello"
2) "world"
3) "k1"
4) "v1"
5) "k2"
6) "v2"
127.0.0.1:6379> hget mat k1 #获取一个key的值
(nil)
127.0.0.1:6379> hget map k1
"v1"
127.0.0.1:6379> HMGET map k1 k2 #获取多个key的值
1) "v1"
2) "v2"
127.0.0.1:6379>
#删除指定的key
127.0.0.1:6379> HDEL map k1#通过删除key，删除对应的value
(integer) 1
127.0.0.1:6379> HGETALL map
1) "hello"
2) "world"
3) "k2"
4) "v2"
127.0.0.1:6379>
#====================
#获取hash的字段数量：HLEN
127.0.0.1:6379> HLEN map  #获取map的键值对的个数
(integer) 2
127.0.0.1:6379> HEXISTS map k2 #判断key是否存在？
(integer) 1
127.0.0.1:6379> HEXISTS map k1
(integer) 0
127.0.0.1:6379>
#==================
#获取所有的key和所有的value
127.0.0.1:6379> HKEYS map #获取keys
1) "hello"
2) "k2"
127.0.0.1:6379> HVALS map #获取values
1) "world"
2) "v2"
127.0.0.1:6379>
#====================================
127.0.0.1:6379> hset map num 1
(integer) 1
127.0.0.1:6379> HINCRBY map num 1  #增加1
(integer) 2
127.0.0.1:6379> hget map num
"2"
127.0.0.1:6379> HINCRBY map num 5 #增加5
(integer) 7
127.0.0.1:6379> hget map num
"7"
127.0.0.1:6379> HINCRBY map num -1 #减少1
```

```
(integer) 6
127.0.0.1:6379> hget map num
"6"
127.0.0.1:6379> HSETNX map hello world #如果key存在，就添加失败
(integer) 0
127.0.0.1:6379> HSETNX map k5 v5 #如果key 不存在，就添加成功
(integer) 1
127.0.0.1:6379>
#=====================================
#存储对象：user:1对象：name，age，sex
127.0.0.1:6379> hset user:1 name zhangsan age 18 sex male
(integer) 3
127.0.0.1:6379> hget user:1 name
"zhangsan"
127.0.0.1:6379>
```

Zset（有序集合）

有序集合(sorted set) 命令

```
127.0.0.1:6379> ZADD myset 1 k1  #添加元素  1：分值，用来排序的
(integer) 1
127.0.0.1:6379> ZADD myset 2 k2 3 k3  4 k4  #一次性添加多个数据
(integer) 3
127.0.0.1:6379> zrange myset 0 -1  #查看有序集合中的所有元素
1) "k1"
2) "k2"
3) "k3"
4) "k4"
127.0.0.1:6379> ZRANGEBYSCORE myset -inf +inf  #按分值从小到大排序
1) "k1"
2) "k2"
3) "k3"
4) "k4"
127.0.0.1:6379> ZREVRANGE myset 0 -1 #将元素进行反转输出
1) "k4"
2) "k3"
3) "k2"
4) "k1"
127.0.0.1:6379> ZREVRANGEBYSCORE myset +inf -inf  #将元素按分值从大到小来显示
1) "k4"
2) "k3"
3) "k2"
4) "k1"
127.0.0.1:6379> ZREM myset k2  #移除k2元素
(integer) 1
127.0.0.1:6379> zrange myset 0 -1  #查看所有元素
1) "k1"
2) "k3"
3) "k4"
127.0.0.1:6379> ZCARD myset  #查看集合中的元素个数
(integer) 3
127.0.0.1:6379> ZCOUNT myset 1 2 #用于计算有序集合中指定分数区间的成员数量
(integer) 1
127.0.0.1:6379> ZCOUNT myset 1 4 #用于计算有序集合中指定分数区间的成员数量
(integer) 3
127.0.0.1:6379> ZSCORE myset k3  #通过元素查看对应的分值
```

```
"3"
```

排行榜，对分值范围的搜索等等

三种特殊类型

geospatial地理索引半径查询)

> geoadd
>
> 将指定的地理空间位置（经度、纬度、名称）添加到指定的 `key` 中。这些数据将会存储到 `sorted set` 这样的目的是为了方便使用 **GEORADIUS** 或者 **GEORADIUSBYMEMBER** 命令对数据进行半径查询等操作。

geospatial底层上实际上是有序集合的实现

```
127.0.0.1:6379> GEOADD city 104.06 30.67 chengdu  #将经度、纬度、城市添加到集合中
(integer) 1
127.0.0.1:6379> GEOADD city 104.32 30.88 jintang #将经度、纬度、城市添加到集合中
(integer) 1
#一次性添加多个经度、纬度、城市到集合中
127.0.0.1:6379> GEOADD city 104.94 30.57 shuangliu 103.86 30.8 pixian 103.81 30.97
wenjiang
(integer) 3
127.0.0.1:6379> zrange city 0 -1  #查看所有城市
1) "chengdu"
2) "pixian"
3) "wenjiang"
4) "jintang"
5) "shuangliu"
127.0.0.1:6379> ZRANGE city 0 -1 withscores  查看所有的城市以及经度、纬度
 1) "chengdu"
 2) "4025771339881718"
 3) "pixian"
 4) "4025772652220571"
 5) "wenjiang"
 6) "4025785402087436"
 7) "jintang"
 8) "4026144732708951"
 9) "shuangliu"
10) "4026171779824518"
127.0.0.1:6379>
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
##GEODIST:返回两个给定位置之间的距离。如果两个位置之间的其中一个不存在， 那么命令返回空值
指定单位的参数 unit 必须是以下单位的其中一个：
m 表示单位为米。
km 表示单位为千米。
mi 表示单位为英里。
ft 表示单位为英尺。
127.0.0.1:6379> GEODIST city jintang shuangliu km  #金堂到双流的公里数
"68.5783"
127.0.0.1:6379> GEODIST city jintang shuangliu m #米
"68578.2629"
127.0.0.1:6379> GEODIST city wenjiang pixian km #温江到成都的公里数
"19.5017"
```

```
127.0.0.1:6379> GEODIST city shuangliu chengdu km
"84.9623"
127.0.0.1:6379> GEODIST city chengdu shuangliu km
"84.9623"
127.0.0.1:6379> GEODIST city pixian chengdu km
"23.9720"
127.0.0.1:6379>
#================================================
#从key里返回所有给定位置元素的位置：geopos
127.0.0.1:6379> geopos city chengdu  #查看成都的经纬度
1) 1) "104.05999749898910522"
   2) "30.67000055930392222"
127.0.0.1:6379>
#================================
#以给定的经纬度为中心， 返回键包含的位置元素当中， 与中心的距离不超过给定最大距离的所有位置元素。
#georadius
127.0.0.1:6379> GEORADIUS city 104.06 30.67 100 km  #以经度104.06，纬度为30.67位中心，寻找100km以内的城市
1) "chengdu"
2) "pixian"
3) "wenjiang"
4) "jingtang"
5) "shuangliu"
127.0.0.1:6379> GEORADIUS city 104.06 30.67 50 km
1) "chengdu"
2) "pixian"
3) "wenjiang"
4) "jingtang"
127.0.0.1:6379>
#
# WITHDIST: 在返回位置元素的同时， 将位置元素与中心之间的距离也一并返回。 距离的单位和用户给定的范围单位保    持一致。
# WITHCOORD: 将位置元素的经度和维度也一并返回。
# WITHHASH: 以 52 位有符号整数的形式， 返回位置元素经过原始 geohash 编码的有序集合分值。 这个选项主要用于# 底层应用或者调试， 实际中的作用并不大。
#
127.0.0.1:6379> GEORADIUS city 104.06 30.67 50 km withcoord #将位置的经度和纬度一起返回
1) 1) "chengdu"
   2) 1) "104.05999749898910522"
      2) "30.67000055930392222"
2) 1) "pixian"
   2) 1) "103.86000126600265503"
      2) "30.79999880340293572"
3) 1) "wenjiang"
   2) 1) "103.80999952554702759"
      2) "30.97000001683864667"
4) 1) "jintang"
   2) 1) "104.32000011205673218"
      2) "30.8799996726339927"
127.0.0.1:6379> GEORADIUS city 104.06 30.67 50 km withdist #返回位置与中心之间的距离返回
1) 1) "chengdu"
   2) "0.0002"
2) 1) "pixian"
   2) "23.9722"
3) 1) "wenjiang"
   2) "41.0324"
4) 1) "jintang"
   2) "34.1017"
```

```
127.0.0.1:6379> GEORADIUS city 104.06 30.67 50 km withhash #实际应用没有什么用？主要用于调试
1) 1) "chengdu"
   2) (integer) 4025771339881718
2) 1) "pixian"
   2) (integer) 4025772652220571
3) 1) "wenjiang"
   2) (integer) 4025785402087436
4) 1) "jintang"
   2) (integer) 4026144732708951
127.0.0.1:6379>
########################
#GEOHASH
#返回一个或多个位置元素的 Geohash 表示。
#将二维的经纬度转换成字符串
127.0.0.1:6379> GEOHASH city chengdu
1) "wm3yrzq1tw0"
127.0.0.1:6379> GEOHASH city pixian wenjiang
1) "wm3z5quwcr0"
2) "wm9b4mbjk60"
127.0.0.1:6379>
#============================
#GEORADIUSBYMEMBER
#这个命令和 GEORADIUS 命令一样， 都可以找出位于指定范围内的元素， 但是 GEORADIUSBYMEMBER 的中心点是由给定的位置元素决定的， 而不是像 GEORADIUS 那样， 使用输入的经度和纬度来决定中心点指定成员的位置被用作查询的中心。
127.0.0.1:6379> GEORADIUSBYMEMBER city chengdu 100 km
1) "chengdu"
2) "pixian"
3) "wenjiang"
4) "jintang"
5) "shuangliu"
127.0.0.1:6379> GEORADIUSBYMEMBER city chengdu 10 km
1) "chengdu"
127.0.0.1:6379>
```

Hyperloglog（基数统计）

Redis Hyperloglog：基数统计的算法

> 什么是基数?

基数：不重复的元素，可以接受误差，标准误差在0.81%。

A:{1,3,5,7,9}

B:{2,3,4,5}

A和B的基数就是等于7

Redis Hyperloglog的优点：占用内存是固定的，2^64不同元素项，只占用12KB内存，内存占用空间小。

> 适合场景

适用于一个热点页面的去重访问次数

统计一个网站的UV（网站独立访客），一个人多次访问，只能算一个访问量。这种方式的重点在于统计（计数）,存在0.81%统计误差，可以忽略不计。

```
127.0.0.1:6379> PFADD mykey a b c d e f g h  #创建第一组元素
(integer) 1
127.0.0.1:6379> PFCOUNT mykey  #统计第一组元素的数量
(integer) 8
127.0.0.1:6379> PFADD mykey2 e f g h i j k l #创建第二组
(integer) 1
127.0.0.1:6379> PFCOUNT mykey2
(integer) 8
127.0.0.1:6379> PFMERGE mykey3 mykey mykey2 #合并两组元素（并集），放到新组中（mykey3），
OK
127.0.0.1:6379> PFCOUNT mykey3
(integer) 12
```

如果可以容错就可以是Hyperloglog，如果不允许容差，就要使用set或者是自己的数据类型。

Bitmaps（位图）

> 位存储

统计在线人数：在线、不在线；365天打卡天数：打开、未打卡；社群活跃人数统计：活跃、不活跃。两个状态的，都可以使用Bitmaps。Bitmaps是一种数据结构，都是操作二进制位来进行记录，只有0和1两个状态。

比如有10W粉丝，都是活跃用户，10w粉丝就是10wbit，100000bit/8/1000=12kb左右，很节省空间。

> 记录一周登录的次数

```
#记录周一到周日，七天的登录状态 0：未登录，1：登录过
127.0.0.1:6379> SETBIT online 0 1  #周一
(integer) 0
127.0.0.1:6379> SETBIT online 1 1 #周二
(integer) 0
127.0.0.1:6379> setbit online 2 1 #周三
(integer) 0
127.0.0.1:6379> setbit online 3 0#周四
(integer) 0
127.0.0.1:6379> setbit online 4 1#周五
(integer) 0
127.0.0.1:6379> setbit online 5 1#周六
(integer) 0
127.0.0.1:6379> setbit online 6 1#周日
(integer) 0
```

> 判断某一天是否登录过

```
127.0.0.1:6379> GETBIT online 5  #判断周六是否登录过
(integer) 1
127.0.0.1:6379> getbit online 3 #周四师傅登录过
(integer) 0
127.0.0.1:6379>
```

> 统计一周登录的天数

```
127.0.0.1:6379> BITCOUNT online  #统计登录（1）的个数
(integer) 6
127.0.0.1:6379>
```

事务（transactions）

ACID

- A（atomicity）：一个事务要么全部提交成功，要么全部失败回滚，不能只执行其中的一部分操作，这就是事务的原子性
- 一致性（consistency)：事务的执行不能破坏数据库数据的完整性和一致性，一个事务在执行之前和执行之后，数据库都必须处于一致性状态。
- 隔离性（isolation）：事务的隔离性是指在并发环境中，并发的事务时相互隔离的，一个事务的执行不能不被其他事务干扰。
- 持久性（durability）：旦事务提交，那么它对数据库中的对应数据的状态的变更就会永久保存到数据库中。即使发生系统崩溃或机器宕机等故障，只要数据库能够重新启动，那么一定能够将其恢复到事务成功结束的状态

Redis中的单条命令是具有原子性的，Redis事务具有原子性吗？

Redis事务本质：一组命令的集合，事务可以一次执行多个命令

序列化、顺序化、独立化

Redis事务

- 开启事务（multi）
- 命令入队
- 执行事务（exec）

> 正常执行事务

```
127.0.0.1:6379> multi  #开启事务
OK
127.0.0.1:6379> set k1 v1  #命令入队
QUEUED
127.0.0.1:6379> set k2 v2#命令入队
QUEUED
127.0.0.1:6379> get k2#命令入队
QUEUED
127.0.0.1:6379> set k3 v3#命令入队
QUEUED
127.0.0.1:6379> exec  #执行事务
1) OK
2) OK
3) "v2"  #命令执行的结果
4) OK
127.0.0.1:6379>
```

> 取消事务

```
127.0.0.1:6379> multi #开启事务
OK
127.0.0.1:6379> set k1 v1
QUEUED
127.0.0.1:6379> set k2 v2
QUEUED
127.0.0.1:6379> DISCARD  #取消事务
OK
127.0.0.1:6379> get k2 #获取不到值，命令执行失败
(nil)
127.0.0.1:6379> get k1  #获取不到值，命令执行失败
(nil)
127.0.0.1:6379>
```

> redis命令有错，命令入队时报错，抛出异常，事务中的所有命令都不会执行

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set k1 v1
QUEUED
127.0.0.1:6379> set k2  #命令执行报错。
(error) ERR wrong number of arguments for 'set' command
127.0.0.1:6379> set k3 v3
QUEUED
127.0.0.1:6379> set k4 v4
QUEUED
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
127.0.0.1:6379> get k1  #所有的命令都不会执行
(nil)
127.0.0.1:6379>
```

> 如果命令队列中出现逻辑错误（类似于编程中1/0），命令依然可以入队，不报错，那么在执行命令时，其他命令可以正常执行，错误命令抛出异常。

```
127.0.0.1:6379> multi #开启事务
OK
127.0.0.1:6379> set k1 v1
QUEUED
127.0.0.1:6379> INCR k1  #语法不报错，逻辑上有错误，   v1不能+1，但是命令还是入队了
QUEUED
127.0.0.1:6379> set k2 v2
QUEUED
127.0.0.1:6379> get k2
QUEUED
127.0.0.1:6379> exec  #执行
1) OK
2) (error) ERR value is not an integer or out of range  #报错，抛出异常，其他的正常执行
3) OK
4) "v2"
127.0.0.1:6379>
```

从上面的例子中可以得出：Redis的单条命令可以保证原子性，但是Redis事务不能保证原子性。