

Redis基础

Redis介绍

什么是Redis

- **Redis** (Remote Dictionary Server) 远程字典服务器，是用**C语言**开发的一个**开源**的高性能**键值对** (key-value) **内存数据库**。
- 它提供了**五种数据类型**来存储值：字符串类型、散列类型、列表类型、集合类型、有序集合类型
- 它是一种 **NoSQL** 数据存储。

Redis发展历史

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 **LL00GG**，然而没过多久该公司的创始人 **Salvatore Sanfilippo** (*antirez*) 便对MySQL的性能感到失望，于是他决定亲自为 **LL00GG** 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**。

Redis2.6

Redis2.6在2012年正式发布，主要特性如下：

服务端支持Lua脚本、去掉虚拟内存相关功能、键的过期时间支持毫秒、从节点提供只读功能、两个新的位图命令：bitcount和bitop、重构了大量的核心代码、优化了大量的命令。

Redis2.8

Redis2.8在2013年11月22日正式发布，主要特性如下：

添加部分主从复制（增量复制）的功能、可以用bind命令绑定多个IP地址、Redis设置了明显的进程名、发布订阅添加了pubsub命令、Redis Sentinel生产可用

Redis3.0

Redis3.0在2015年4月1日正式发布，相比于Redis2.8主要特性如下：

Redis Cluster：Redis的官方分布式实现（Ruby）、全新的对象编码结果、lru算法大幅提升、部分命令的性能提升

Redis3.2

Redis3.2在2016年5月6日正式发布，相比于Redis3.0主要特征如下：

添加GEO相关功能、SDS在速度和节省空间上都做了优化、新的List编码类型：quicklist、从节点读取过期数据保证一致性、Lua脚本功能增强等

Redis4.0

Redis4.0在2017年7月发布，主要特性如下：

提供了模块系统，方便第三方开发者拓展Redis的功能、PSYNC2.0：优化了之前版本中，主从节点切换必然引起全量复制的问题、提供了新的缓存剔除算法：LFU (Last Frequently Used)，并对已有算法进行了优化、提供了RDB-AOF混合持久化格式等

Redis应用场景

- 缓存使用，减轻DB压力

- DB使用，用于临时存储数据（字典表，购买记录）
- 解决分布式场景下Session分离问题（登录信息）
- 任务队列（秒杀、抢红包等等）乐观锁
- 应用排行榜 zset
- 签到 bitmap
- 分布式锁
- 冷热数据交换

Redis单机版安装和使用

Redis下载

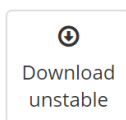
- 官网地址：<http://redis.io/>
- 中文官网地址：<http://www.redis.cn/>
- 下载地址：<http://download.redis.io/releases/>

下载

Redis 使用标准版本标记进行版本控制：**major.minor.patchlevel**。偶数的版本号表示稳定的版本，例如 1.2，2.0，2.2，2.4，2.6，2.8，奇数的版本号用来表示非标准版本，例如2.9.x是非稳定版本，它的稳定版本是3.0。

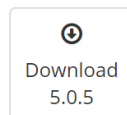
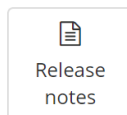
非稳定版

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.



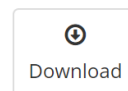
稳定版 (5.0)

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.



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.



Redis安装环境

Redis 没有官方的 windows 版本，所以建议在 Linux 系统上安装运行。

我们使用 CentOS 7 作为安装环境。

Redis安装

第一步：安装 C 语言需要的 GCC 环境

```
yum install -y gcc-c++  
yum install -y wget
```

第二步：下载并解压缩 Redis 源码压缩包

```
wget http://download.redis.io/releases/redis-5.0.5.tar.gz  
tar -zxf redis-5.0.5.tar.gz
```

第三步：编译 Redis 源码，进入 redis-5.0.5 目录，执行编译命令

```
cd redis-5.0.5/src
make
```

第四步：安装 Redis，需要通过 PREFIX 指定安装路径

```
mkdir /usr/redis -p
make install PREFIX=/usr/redis
```

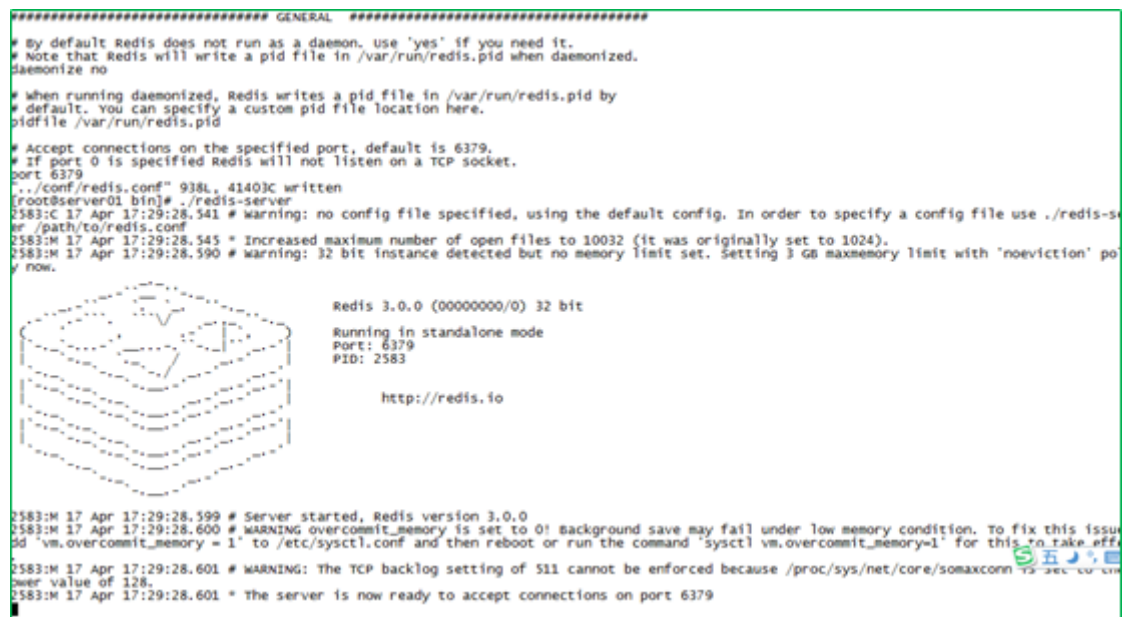
Redis启动

前端启动

- 启动命令：redis-server，直接运行 bin/redis-server 将以前端模式启动

```
./redis-server
```

- 关闭命令：ctrl+c
- 启动缺点：客户端窗口关闭则 redis-server 程序结束，不推荐使用此方法
- 启动图例：



```
##### GENERAL #####
# By default Redis does not run as a daemon. Use 'yes' if you need it.
# Note that Redis will write a pid file in /var/run/redis.pid when daemonized.
daemonize no

# When running daemonized, Redis writes a pid file in /var/run/redis.pid by
# default. You can specify a custom pid file location here.
pidfile /var/run/redis.pid

# Accept connections on the specified port, default is 6379.
# If port 0 is specified Redis will not listen on a TCP socket.
port 6379
./conf/redis.conf" 938L, 41403C written
[root@server01 bin]# ./redis-server
2583:C 17 Apr 17:29:28.541 # Warning: no config file specified, using the default config. In order to specify a config file use ./redis-s
er /path/to/redis.conf
2583:M 17 Apr 17:29:28.545 * Increased maximum number of open files to 10032 (it was originally set to 1024).
2583:M 17 Apr 17:29:28.590 # Warning: 32 bit instance detected but no memory limit set. Setting 3 GB maxmemory limit with 'noeviction' po
ly now.

Redis 3.0.0 (00000000/0) 32 bit
Running in standalone mode
Port: 6379
PID: 2583

http://redis.io

2583:M 17 Apr 17:29:28.599 # Server started, Redis version 3.0.0
2583:M 17 Apr 17:29:28.600 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue
dd 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take eff
power value of 128.
2583:M 17 Apr 17:29:28.601 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the
2583:M 17 Apr 17:29:28.601 * The server is now ready to accept connections on port 6379
```

后端启动（守护进程启动）

- 第一步：拷贝 redis-5.0.5/redis.conf 配置文件到 Redis 安装目录的 bin 目录

```
cp redis.conf /usr/redis/bin/
```

- 第二步：修改 redis.conf

```
vim redis.conf
```

```
# 将`daemonize`由`no`改为`yes`  
daemonize yes  
  
# 默认绑定的是回环地址，默认不能被其他机器访问  
# bind 127.0.0.1  
  
# 是否开启保护模式，由yes该为no  
protected-mode no
```

- 第三步：启动服务

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

后端启动的关闭方式

```
./redis-cli shutdown
```

命令说明

- `redis-server`：启动 redis 服务
- `redis-cli`：进入 redis 命令客户端
- `redis-benchmark`：性能测试的工具
- `redis-check-aof`：aof 文件进行检查的工具
- `redis-check-dump`：rdb 文件进行检查的工具
- `redis-sentinel`：启动哨兵监控服务

Redis命令行客户端

- 命令格式

```
./redis-cli -h 127.0.0.1 -p 6379
```

- 参数说明

-h: redis服务器的ip地址
-p: redis实例的端口号

- 默认方式

如果不指定主机和端口也可以

- 默认主机地址是127.0.0.1
- 默认端口是6379

```
./redis0-cli
```

Redis数据类型和应用场景

Redis是一个Key-Value的存储系统，使用ANSI C语言编写。

key的类型是字符串。

value的数据类型有：

常用的：string字符串类型、list列表类型、set集合类型、sortedset (zset) 有序集合类型、hash类型。

不常见的：bitmap位图类型、geo地理位置类型。

Redis5.0新增一种：stream类型

注意：Redis中命令是忽略大小写，（set SET），key是不忽略大小写的（NAME name）

Redis的Key的设计

- 1. 用:分割
- 2. 把表名转换为key前缀, 比如: user:
- 3. 第二段放置主键值
- 4. 第三段放置列名

比如：用户表user, 转换为redis的key-value存储

userid	username	password	email
9	zhangf	111111	zhangf@lagou.com

username 的 key: user:9:username

{userid:9,username:zhangf}

email的key user:9:email

表示明确：看key知道意思

不易被覆盖

string字符串类型

Redis的String能表达3种值的类型：字符串、整数、浮点数 100.01 是个六位的串

常见操作命令如下表：

命令名称		命令描述
set	set key value	赋值
get	get key	取值
getset	getset key value	取值并赋值
setnx	setnx key value	当value不存在时采用赋值 set key value NX PX 3000 原子操作，px 设置毫秒数
append	append key value	向尾部追加值
strlen	strlen key	获取字符串长度
incr	incr key	递增数字
incrby	incrby key increment	增加指定的整数
decr	decr key	递减数字
decrby	decrby key decrement	减少指定的整数

应用场景：

1、key和命令是字符串

2、普通的赋值

3、incr用于乐观锁

incr：递增数字，可用于实现乐观锁 watch(事务)

4、setnx用于分布式锁

当value不存在时采用赋值，可用于实现分布式锁

举例：

setnx:

```
127.0.0.1:6379> setnx name zhangf      #如果name不存在赋值
(integer) 1
127.0.0.1:6379> setnx name zhaoyun    #再次赋值失败
(integer) 0
127.0.0.1:6379> get name
"zhangf"
```

set

```
127.0.0.1:6379> set age 18 NX PX 10000 #如果不存在赋值 有效期10秒
OK
127.0.0.1:6379> set age 20 NX          #赋值失败
(nil)
127.0.0.1:6379> get age                #age失效
(nil)
127.0.0.1:6379> set age 30 NX PX 10000 #赋值成功
OK
127.0.0.1:6379> get age
"30"
```

list列表类型

list列表类型可以存储有序、可重复的元素

获取头部或尾部附近的记录是极快的

list的元素个数最多为 $2^{32}-1$ 个（40亿）

常见操作命令如下表：

命令名称	命令格式	描述
lpush	lpush key v1 v2 v3 ...	从左侧插入列表
lpop	lpop key	从列表左侧取出
rpush	rpush key v1 v2 v3 ...	从右侧插入列表
rpop	rpop key	从列表右侧取出
lpushx	lpushx key value	将值插入到列表头部
rpushx	rpushx key value	将值插入到列表尾部
blpop	blpop key timeout	从列表左侧取出，当列表为空时阻塞，可以设置最大阻塞时间，单位为秒
brpop	brpop key timeout	从列表右侧取出，当列表为空时阻塞，可以设置最大阻塞时间，单位为秒
llen	llen key	获得列表中元素个数
lindex	lindex key index	获得列表中下标为index的元素 index从0开始
lrange	lrange key start end	返回列表中指定区间的元素，区间通过start和end指定
lrem	lrem key count value	删除列表中与value相等的元素 当count>0时，lrem会从列表左边开始删除;当count<0时，lrem会从列表后边开始删除;当count=0时，lrem删除所有值为value的元素
lset	lset key index value	将列表index位置的元素设置成value的值
ltrim	ltrim key start end	对列表进行修剪，只保留start到end区间
rpoplpush	poplpush key1 key2	从key1列表右侧弹出并插入到key2列表左侧
brpoplpush	brpoplpush key1 key2	从key1列表右侧弹出并插入到key2列表左侧，会阻塞
linsert	linsert key BEFORE/AFTER pivot value	将value插入到列表，且位于值pivot之前或之后

应用场景：

1、作为栈或队列使用

列表有序可以作为栈和队列使用

2、可用于各种列表，比如用户列表、商品列表、评论列表等。

举例：

```
127.0.0.1:6379> lpush list:1 1 2 3 4 5 3
(integer) 5
127.0.0.1:6379> lrange list:1 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lpop list:1 # 从0开始
"5"
127.0.0.1:6379> rpop list:1
"1"
127.0.0.1:6379> lindex list:1 1
"3"
127.0.0.1:6379> lrange list:1 0 -1
1) "4"
2) "3"
3) "2"
127.0.0.1:6379> lindex list:1 1
"3"
127.0.0.1:6379> rpoplpush list:1 list:2
"2"
127.0.0.1:6379> lrange list:2 0 -1
1) "2"
127.0.0.1:6379> lrange list:1 0 -1
1) "4"
2) "3"
```

set集合类型

Set：无序、唯一元素

集合中最大的成员数为 $2^{32} - 1$

常见操作命令如下表：

命令名称	命令格式	描述
sadd	sadd key mem1 mem2	为集合添加新成员
srem	srem key mem1 mem2	删除集合中指定成员
smembers	smembers key	获得集合中所有元素
spop	spop key	返回集合中一个随机元素，并将该元素删除
randmember	randmember key	返回集合中一个随机元素，不会删除该元素
scard	scard key	获得集合中元素的数量
sismember	sismember key member	判断元素是否在集合内
sinter	sinter key1 key2 key3	求多集合的交集
sdiff	sdiff key1 key2 key3	求多集合的差集
sunion	sunion key1 key2 key3	求多集合的并集

应用场景：

适用于不能重复的且不需要顺序的数据结构

比如：关注的用户，还可以通过spop进行随机抽奖

举例：

```
127.0.0.1:6379> sadd set:1 a b c d
(integer) 4
127.0.0.1:6379> smembers set:1
1) "d"
2) "b"
3) "a"
4) "c"
127.0.0.1:6379> srandmember set:1
"c"
127.0.0.1:6379> srandmember set:1
"b"
127.0.0.1:6379> sadd set:2 b c r f
(integer) 4
127.0.0.1:6379> sinter set:1 set:2
1) "b"
2) "c"
127.0.0.1:6379> spop set:1
"d"
127.0.0.1:6379> smembers set:1
1) "b"
2) "a"
3) "c"
```

sortedset有序集合类型

SortedSet(ZSet) 有序集合：元素本身是无序不重复的

每个元素关联一个分数(score)

可按分数排序，分数可重复

常见操作命令如下表：

命令名称	命令格式	描述
zadd	zadd key score1 member1 score2 member2 ...	为有序集合添加新成员
zrem	zrem key mem1 mem2	删除有序集合中指定成员
zcard	zcard key	获得有序集合中的元素数量
zcount	zcount key min max	返回集合中score值在[min,max]区间的元素数量
zincrby	zincrby key increment member	在集合的member分值上加increment
zscore	zscore key member	获得集合中member的分值
zrank	zrank key member	获得集合中member的排名（按分值从小到大）
zrevrank	zrevrank key member	获得集合中member的排名（按分值从大到小）
zrange	zrange key start end	获得集合中指定区间成员，按分数递增排序
zrevrange	zrevrange key start end	获得集合中指定区间成员，按分数递减排序

应用场景：

由于可以按照分值排序，所以适用于各种排行榜。比如：点击排行榜、销量排行榜、关注排行榜等。

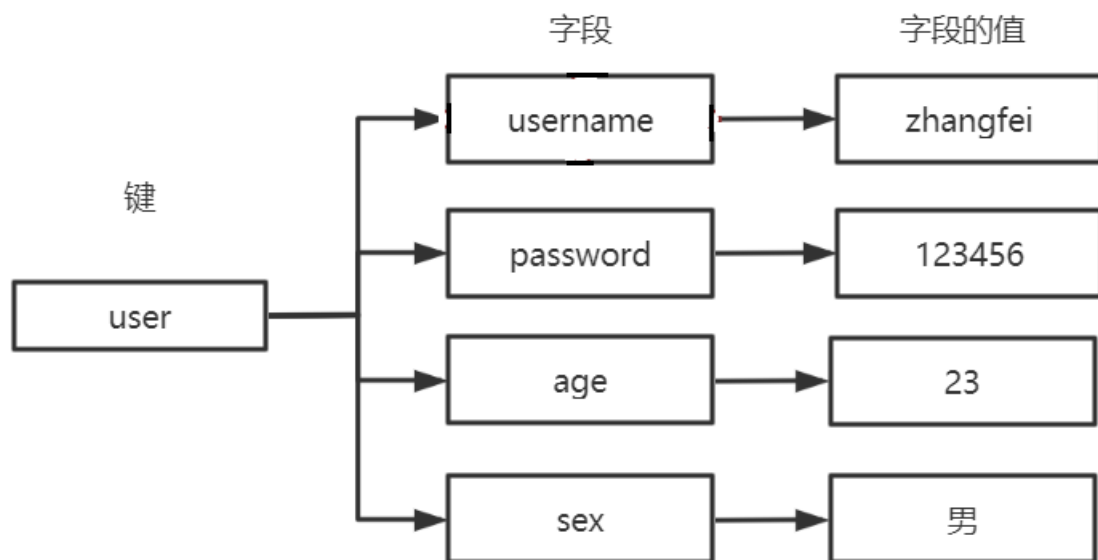
举例：

```
127.0.0.1:6379> zadd hit:1 100 item1 20 item2 45 item3
(integer) 3
127.0.0.1:6379> zcard hit:1
(integer) 3
127.0.0.1:6379> zscore hit:1 item3
"45"
127.0.0.1:6379> zrevrange hit:1 0 -1
1) "item1"
2) "item3"
3) "item2"
127.0.0.1:6379>
```

hash类型（散列表）

Redis hash 是一个 string 类型的 field 和 value 的映射表，它提供了字段和字段值的映射。

每个 hash 可以存储 $2^{32} - 1$ 键值对（40多亿）。



常见操作命令如下表：

命令名称	命令格式	描述
hset	hset key field value	赋值，不区别新增或修改
hmset	hmset key field1 value1 field2 value2	批量赋值
hsetnx	hsetnx key field value	赋值，如果field存在则不操作
hexists	hexists key field	查看某个field是否存在
hget	hget key field	获取一个字段值
hmget	hmget key field1 field2 ...	获取多个字段值
hgetall	hgetall key	
hdel	hdel key field1 field2...	删除指定字段
hincrby	hincrby key field increment	指定字段自增increment
hlen	hlen key	获得字段数量

应用场景：

对象的存储，表数据的映射

举例：

```
127.0.0.1:6379> hmset user:001 username zhangfei password 111 age 23 sex M
OK
127.0.0.1:6379> hgetall user:001
1) "username"
2) "zhangfei"
3) "password"
4) "111"
5) "age"
6) "23"
7) "sex"
```

```
8) "M"
127.0.0.1:6379> hget user:001 username
"zhangfei"
127.0.0.1:6379> hincrby user:001 age 1
(integer) 24
127.0.0.1:6379> hlen user:001
(integer) 4
```

bitmap位图类型

bitmap是进行位操作的

通过一个bit位来表示某个元素对应的值或者状态,其中的key就是对应元素本身。

bitmap本身会极大的节省储存空间。

常见操作命令如下表：

命令名称	命令格式	描述
setbit	setbit key offset value	设置key在offset处的bit值(只能是0或者1)。
getbit	getbit key offset	获得key在offset处的bit值
bitcount	bitcount key	获得key的bit位为1的个数
bitpos	bitpos key value	返回第一个被设置为bit值的索引值
bitop	bitop and[or/xor/not] destkey key [key ...]	对多个key 进行逻辑运算后存入destkey 中

应用场景：

- 1、用户每月签到，用户id为key，日期作为偏移量 1表示签到
- 2、统计活跃用户,日期为key，用户id为偏移量 1表示活跃
- 3、查询用户在线状态，日期为key，用户id为偏移量 1表示在线

举例：

```
127.0.0.1:6379> setbit user:sign:1000 20200101 1 #id为1000的用户20200101签到
(integer) 0
127.0.0.1:6379> setbit user:sign:1000 20200103 1 #id为1000的用户20200103签到
(integer) 0
127.0.0.1:6379> getbit user:sign:1000 20200101 #获得id为1000的用户20200101签到状态
1 表示签到
(integer) 1
127.0.0.1:6379> getbit user:sign:1000 20200102 #获得id为1000的用户20200102签到状态
0表示未签到
(integer) 0
127.0.0.1:6379> bitcount user:sign:1000 # 获得id为1000的用户签到次数
(integer) 2
127.0.0.1:6379> bitpos user:sign:1000 1 #id为1000的用户第一次签到的日期
(integer) 20200101
127.0.0.1:6379> setbit 20200201 1000 1 #20200201的1000号用户上线
(integer) 0
```

```

127.0.0.1:6379> setbit 20200202 1001 1           #20200202的1000号用户上线
(integer) 0
127.0.0.1:6379> setbit 20200201 1002 1           #20200201的1002号用户上线
(integer) 0
127.0.0.1:6379> bitcount 20200201                #20200201的上线用户有2个
(integer) 2
127.0.0.1:6379> bitop or desk1 20200201 20200202 #合并20200201的用户和20200202上线了的用户
(integer) 126
127.0.0.1:6379> bitcount desk1                    #统计20200201和20200202都上线的用户个数
(integer) 3

```

geo地理位置类型

geo是Redis用来处理位置信息的。在Redis3.2中正式使用。主要是利用了Z阶曲线、Base32编码和geohash算法

Z阶曲线

在x轴和y轴上将十进制数转化为二进制数，采用x轴和y轴对应的二进制数依次交叉后得到一个六位数编码。把数字从小到大依次连起来的曲线称为Z阶曲线，Z阶曲线是把多维转换成一维的一种方法。

	x:	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
y: 0	000	000000	000001	000100	000101	010000	010001	010100	010101
1	001	000010	000011	000110	000111	010010	010011	010110	010111
2	010	001000	001001	001100	001101	011000	011001	011100	011101
3	011	001010	001011	001110	001111	011010	011011	011110	011111
4	100	100000	100001	100100	100101	110000	110001	110100	110101
5	101	100010	100011	100110	100111	110010	110011	110110	110111
6	110	101000	101001	101100	101101	111000	111001	111100	111101
7	111	101010	101011	101110	101111	111010	111011	111110	111111

Base32编码

Base32这种数据编码机制，主要用来把二进制数据编码成可见的字符串，其编码规则是：任意给定一个二进制数据，以5个位(bit)为一组进行切分(base64以6个位(bit)为一组)，对切分而成的每个组进行编码得到1个可见字符。Base32编码表字符集中的字符总数为32个（0-9、b-z去掉a、i、l、o），这也是Base32名字的由来。

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base 32	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	g
Decimal	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Base 32	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

geohash算法

Gustavo在2008年2月上线了geohash.org网站。Geohash是一种地理位置信息编码方法。经过geohash映射后，地球上任意位置的经纬度坐标可以表示成一个较短的字符串。可以方便的存储在数据库中，附在邮件上，以及方便的使用在其他服务中。以北京的坐标举例，[39.928167,116.389550]可以转换成 wx4g0s8q3j f9。

Redis中经纬度使用52位的整数进行编码，放进zset中，zset的value元素是key，score是GeoHash的52位整数值。在使用Redis进行Geo查询时，其内部对应的操作其实只是zset(skiplist)的操作。通过zset的score进行排序就可以得到坐标附近的其它元素，通过将score还原成坐标值就可以得到元素的原始坐标。

常见操作命令如下表：

命令名称	命令格式	描述
geoadd	geoadd key 经度 纬度 成员名称1 经度1 纬度1 成员名称2 经度2 纬度 2 ...	添加地理坐标
geohash	geohash key 成员名称1 成员名称2...	返回标准的geohash串
geopos	geopos key 成员名称1 成员名称2...	返回成员经纬度
geodist	geodist key 成员1 成员2 单位	计算成员间距离
georadiusbymember	georadiusbymember key 成员 值单位 count 数 asc[desc]	根据成员查找附近的成员

应用场景：

- 1、记录地理位置
- 2、计算距离
- 3、查找"附近的人"

举例：

```
127.0.0.1:6379> geoadd user:addr 116.31 40.05 zhangf 116.38 39.88 zhaoyun 116.47
40.00 diaochan #添加用户地址 zhangf、zhaoyun、diaochan的经纬度
(integer) 3
```

```

127.0.0.1:6379> geohash user:addr zhangf diaochan #获得zhangf和diaochan的geohash码
1) "wx4eydyk5m0"
2) "wx4gd3fbgs0"
127.0.0.1:6379> geopos user:addr zhaoyun #获得zhaoyun的经纬度
1) 1) "116.38000041246414185"
   2) "39.88000114172373145"
127.0.0.1:6379> geodist user:addr zhangf diaochan #计算zhangf到diaochan的距离,单位是m
"14718.6972"
127.0.0.1:6379> geodist user:addr zhangf diaochan km #计算zhangf到diaochan的距离,单位是km
"14.7187"
127.0.0.1:6379> geodist user:addr zhangf zhaoyun km
"19.8276"
127.0.0.1:6379> georadiusbymember user:addr zhangf 20 km withcoord withdist count 3 asc
# 获得距离zhangf20km以内的按由近到远的顺序排出前三名的成员名称、距离及经纬度
#withcoord : 获得经纬度 withdist: 获得距离 withhash: 获得geohash码
1) 1) "zhangf"
   2) "0.0000"
   3) 1) "116.31000012159347534"
      2) "40.04999982043828055"
2) 1) "diaochan"
   2) "14.7187"
   3) 1) "116.46999925374984741"
      2) "39.99999991084916218"
3) 1) "zhaoyun"
   2) "19.8276"
   3) 1) "116.38000041246414185"
      2) "39.88000114172373145"

```

stream数据流类型

stream是Redis5.0后新增的数据结构，用于可持久化的消息队列。

几乎满足了消息队列具备的全部内容，包括：

- 消息ID的序列化生成
- 消息遍历
- 消息的阻塞和非阻塞读取
- 消息的分组消费
- 未完成消息的处理
- 消息队列监控

每个Stream都有唯一的名称，它就是Redis的key，首次使用 `xadd` 指令追加消息时自动创建。

常见操作命令如下表：

命令名称	命令格式	描述
xadd	xadd key id <*> field1 value1....	将指定消息数据追加到指定队列(key)中, * 表示最新生成的id (当前时间+序列号)
xread	xread [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	从消息队列中读取, COUNT: 读取条数, BLOCK: 阻塞读 (默认不阻塞) key: 队列名称 id: 消息id
xrange	xrange key start end [COUNT]	读取队列中给定ID范围的消息 COUNT: 返回消息条数 (消息id从小到大)
xrevrange	xrevrange key start end [COUNT]	读取队列中给定ID范围的消息 COUNT: 返回消息条数 (消息id从大到小)
xdel	xdel key id	删除队列的消息
xgroup	xgroup create key groupname id	创建一个新的消费组
xgroup	xgroup destroy key groupname	删除指定消费组
xgroup	xgroup delconsumer key groupname cname	删除指定消费组中的某个消费者
xgroup	xgroup setid key id	修改指定消息的最大id
xreadgroup	xreadgroup group groupname consumer COUNT streams key	从队列中的消费组中创建消费者并消费数据 (consumer不存在则创建)

应用场景:

消息队列的使用

```

127.0.0.1:6379> xadd topic:001 * name zhangfei age 23
"1591151905088-0"
127.0.0.1:6379> xadd topic:001 * name zhaoyun age 24 name diaochan age 16
"1591151912113-0"
127.0.0.1:6379> xrange topic:001 - +
1) 1) "1591151905088-0"
   2) 1) "name"
      2) "zhangfei"
      3) "age"
      4) "23"
2) 1) "1591151912113-0"
   2) 1) "name"
      2) "zhaoyun"
      3) "age"
      4) "24"
      5) "name"
      6) "diaochan"
      7) "age"
      8) "16"
127.0.0.1:6379> xread COUNT 1 streams topic:001 0
1) 1) "topic:001"
   2) 1) 1) "1591151905088-0"

```



```

2) 1) "name"
2) "zhangfei"
3) "age"
4) "23"

#创建的group1
127.0.0.1:6379> xgroup create topic:001 group1 0
OK
# 创建cus1加入到group1 消费 没有被消费过的消息 消费第一条
127.0.0.1:6379> xreadgroup group group1 cus1 count 1 streams topic:001 >
1) 1) "topic:001"
2) 1) 1) "1591151905088-0"
2) 1) "name"
2) "zhangfei"
3) "age"
4) "23"

#继续消费 第二条
127.0.0.1:6379> xreadgroup group group1 cus1 count 1 streams topic:001 >
1) 1) "topic:001"
2) 1) 1) "1591151912113-0"
2) 1) "name"
2) "zhaoyun"
3) "age"
4) "24"
5) "name"
6) "diaochan"
7) "age"
8) "16"

#没有可消费
127.0.0.1:6379> xreadgroup group group1 cus1 count 1 streams topic:001 >
(nil)

```

Redis常用命令

官方命令大全网址: <http://www.redis.cn/commands.html>

keys

返回满足给定pattern 的所有key

语法:

```
keys pattern
```

示例:

```

redis 127.0.0.1:6379> keys list*
1) "list"
2) "list5"
3) "list6"
4) "list7"
5) "list8"

```

del

语法:

```
DEL key
```

示例:

```
127.0.0.1:6379> del test  
(integer) 1
```

exists

确认一个key 是否存在

语法:

```
exists key
```

示例: 从结果来看, 数据库中不存在 Hongwan 这个 key, 但是 age 这个 key 是存在的

```
redis 127.0.0.1:6379> exists Hongwan  
(integer) 0  
redis 127.0.0.1:6379> exists age  
(integer) 1  
redis 127.0.0.1:6379>
```

expire

Redis在实际使用过程中更多的用作缓存, 然而缓存的数据一般都是需要设置生存时间的, 即: 到期后数据销毁。

语法:

EXPIRE key seconds	设置key的生存时间 (单位: 秒) key在多少秒后会自动删除
TTL key	查看key生时的生存时间
PERSIST key	清除生存时间
PEXPIRE key milliseconds	生存时间设置单位为: 毫秒

示例:

```
redis 127.0.0.1:6379> set test 1          设置test的值为1  
OK  
redis 127.0.0.1:6379> get test          获取test的值  
"1"  
redis 127.0.0.1:6379> EXPIRE test 5     设置test的生存时间为5秒  
(integer) 1  
redis 127.0.0.1:6379> TTL test 查看test的生时生成时间还有1秒删除  
(integer) 1  
redis 127.0.0.1:6379>redis 127.0.0.1:6379> TTL test  
(integer) -2  
redis 127.0.0.1:6379> get test          获取test的值, 已经删除  
(nil)
```

rename

重命名key

语法:

```
rename oldkey newkey
```

示例: `age` 成功的被我们改名为 `age_new` 了

```
redis 127.0.0.1:6379> keys *
1) "age"
redis 127.0.0.1:6379> rename age age_new
OK
redis 127.0.0.1:6379> keys *
1) "age_new"
redis 127.0.0.1:6379>
```

type

显示指定key的数据类型

语法:

```
type key
```

示例: 这个方法可以非常简单的判断出值的类型

```
redis 127.0.0.1:6379> type addr
string
redis 127.0.0.1:6379> type myzset2
zset
redis 127.0.0.1:6379> type mylist
list
```

Redis的Java客户端—Jedis

1、关闭RedisServer端的防火墙

```
systemctl stop firewalld (默认)
systemctl disable firewalld.service (设置开启不启动)
```

2、新建maven项目后导入Jedis包

pom.xml

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.9.0</version>
</dependency>
```

3、写程序

```
@Test
public void testConn(){
    //与Redis建立连接 IP+port
    Jedis redis = new Jedis("192.168.127.128", 6379);
    //在Redis中写字符串 key value
    redis.set("jedis:name:1","jd-zhangfei");
    //获得Redis中字符串的值
    System.out.println(redis.get("jedis:name:1"));
    //在Redis中写list
    redis.lpush("jedis:list:1","1","2","3","4","5");
    //获得list的长度
    System.out.println(redis.llen("jedis:list:1"));
}
```

缓存过期和淘汰策略

Redis性能高：

官方数据

读：110000次/s

写：81000次/s

长期使用，key会不断增加，Redis作为缓存使用，物理内存也会满
内存与硬盘交换（swap） 虚拟内存，频繁IO 性能急剧下降

maxmemory

不设置的场景

Redis的key是固定的，不会增加

Redis作为DB使用，保证数据的完整性，不能淘汰，可以做集群，横向扩展

缓存淘汰策略：禁止驱逐（默认）

设置的场景

Redis是作为缓存使用，不断增加Key

maxmemory：默认为0 不限制

问题：达到物理内存后性能急剧下降，甚至崩溃

内存与硬盘交换（swap） 虚拟内存，频繁IO 性能急剧下降

设置多少？

与业务有关

1个Redis实例，保证系统运行 1 G，剩下的就都可以设置Redis

物理内存的3/4

slaver：留出一定的内存

在redis.conf中

```
maxmemory 1024mb
```

命令：获得maxmemory数

```
CONFIG GET maxmemory
```

设置maxmemory后，当趋近maxmemory时，通过缓存淘汰策略，从内存中删除对象

不设置maxmemory 无最大内存限制 maxmemory-policy noeviction（禁止驱逐）不淘汰

设置maxmemory maxmemory-policy 要配置

expire数据结构

在Redis中可以使用expire命令设置一个键的存活时间(ttl: time to live)，过了这段时间，该键就会自动被删除。

expire的使用

expire命令的使用方法如下：expire key ttl(单位秒)

```
127.0.0.1:6379> expire name 2 #2秒失效
(integer) 1
127.0.0.1:6379> get name
(nil)
127.0.0.1:6379> set name zhangfei
OK
127.0.0.1:6379> ttl name #永久有效
(integer) -1
127.0.0.1:6379> expire name 30 #30秒失效
(integer) 1
127.0.0.1:6379> ttl name #还有24秒失效
(integer) 24
127.0.0.1:6379> ttl name #失效
(integer) -2
```

expire原理

```
typedef struct redisDb {
    dict *dict; -- key value
    dict *expires; -- key ttl
    dict *blocking_keys;
    dict *ready_keys;
    dict *watched_keys;
    int id;
} redisDb;
```

上面的代码是Redis 中关于数据库的结构体定义，这个结构体定义中除了 id 以外都是指向字典的指针，其中我们只看 dict 和 expires。

dict 用来维护一个 Redis 数据库中包含的所有 Key-Value 键值对，expires则用于维护一个 Redis 数据库中设置了失效时间的键(即key与失效时间的映射)。

当我们使用 expire命令设置一个key的失效时间时，Redis 首先到 dict 这个字典表中查找要设置的key 是否存在，如果存在就将这个key和失效时间添加到 expires 这个字典表。

当我们使用 setex命令向系统插入数据时，Redis 首先将 Key 和 Value 添加到 dict 这个字典表中，然后将 Key 和失效时间添加到 expires 这个字典表中。

简单地总结来说就是，设置了失效时间的key和具体的失效时间全部都维护在 expires 这个字典表中。

删除策略

Redis的数据删除有定时删除、惰性删除和主动删除三种方式。

Redis目前采用惰性删除+主动删除的方式。

定时删除

在设置键的过期时间的同时，创建一个定时器，让定时器在键的过期时间来临时，立即执行对键的删除操作。

需要创建定时器，而且消耗CPU，一般不推荐使用。

惰性删除

在key被访问时如果发现它已经失效，那么就删除它。

调用expireIfNeeded函数，该函数的意义是：读取数据之前先检查一下它有没有失效，如果失效了就删除它。

```
int expireIfNeeded(redisDb *db, robj *key) {
    //获取主键的失效时间    get当前时间-创建时间>ttl
    long long when = getExpire(db,key);
    //假如失效时间为负数，说明该主键未设置失效时间（失效时间默认为-1），直接返回0
    if (when < 0) return 0;
    //假如Redis服务器正在从RDB文件中加载数据，暂时不进行失效主键的删除，直接返回0
    if (server.loading) return 0;
    ...
    //如果以上条件都不满足，就将主键的失效时间与当前时间进行对比，如果发现指定的主键
    //还未失效就直接返回0
    if (mstime() <= when) return 0;
    //如果发现主键确实已经失效了，那么首先更新关于失效主键的统计个数，然后将该主键失
    //效的信息进行广播，最后将该主键从数据库中删除
    server.stat_expiredkeys++;
    propagateExpire(db,key);
    return dbDelete(db,key);
}
```

主动删除

在redis.conf文件中可以配置主动删除策略,默认是no-eviction（不删除）

LRU

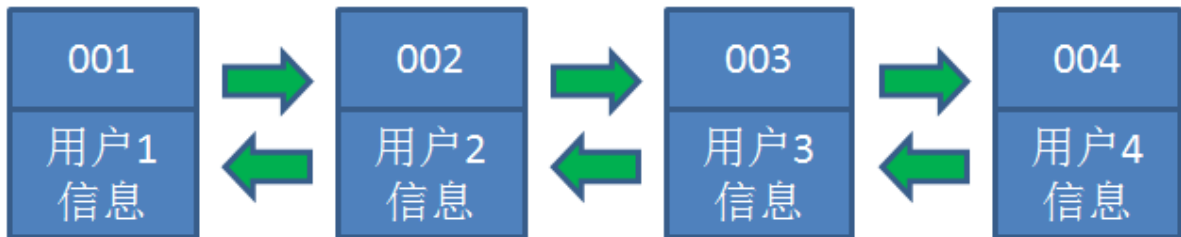
LRU (Least recently used) 最近最少使用，算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。

最常见的实现是使用一个链表保存缓存数据，详细算法实现如下：

1. 新数据插入到链表头部；
2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部；
3. 当链表满的时候，将链表尾部的数据丢弃。
4. 在Java中可以使用**LinkHashMap**（哈希链表）去实现LRU

让我们以用户信息的需求为例，来演示一下LRU算法的基本思路：

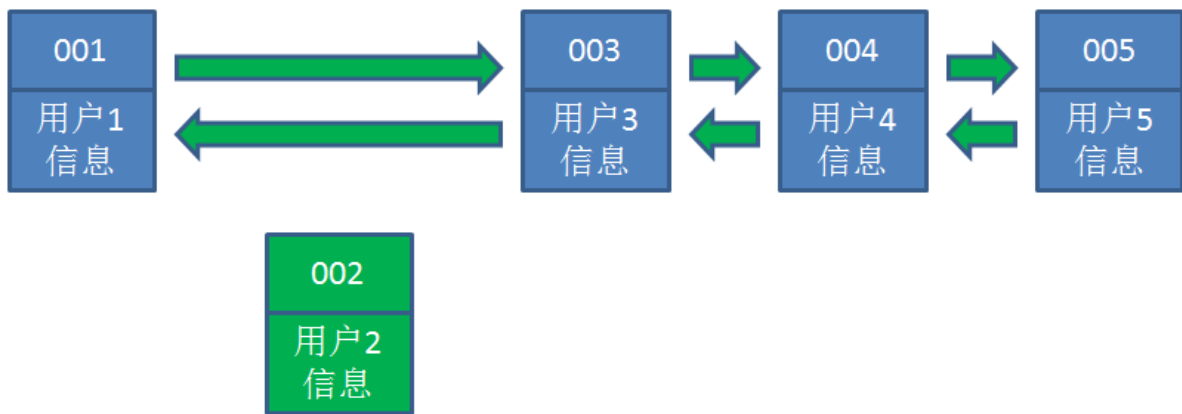
1.假设我们使用哈希链表来缓存用户信息，目前缓存了4个用户，这4个用户是按照时间顺序依次从链表右端插入的。



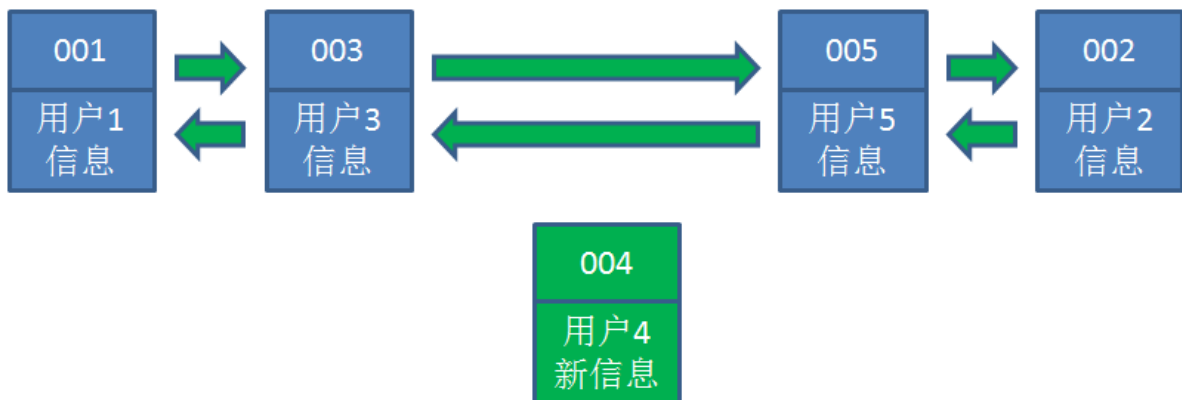
2.此时，业务方访问用户5，由于哈希链表中没有用户5的数据，我们从数据库中读取出来，插入到缓存当中。这时候，链表中最右端是最新访问到的用户5，最左端是最近最少访问的用户1。



3.接下来，业务方访问用户2，哈希链表中存在用户2的数据，我们怎么做呢？我们把用户2从它的前驱节点和后继节点之间移除，重新插入到链表最右端。这时候，链表中最右端变成了最新访问到的用户2，最左端仍然是最近最少访问的用户1。



4.接下来，业务方请求修改用户4的信息。同样道理，我们把用户4从原来的位置移动到链表最右侧，并把用户信息的值更新。这时候，链表中最右端是最新访问到的用户4，最左端仍然是最近最少访问的用户1。



5.业务访问用户6，用户6在缓存里没有，需要插入到哈希链表。假设这时候缓存容量已经达到上限，必须先删除最近最少访问的数据，那么位于哈希链表最左端的用户1就会被删除掉，然后再把用户6插入到最右端。



Redis的LRU 数据淘汰机制

在服务器配置中保存了 lru 计数器 `server.lrulock`，会定时（redis 定时程序 `serverCron()`）更新，`server.lrulock` 的值是根据 `server.unixtime` 计算出来的。

另外，从 `struct redisObject` 中可以发现，每一个 redis 对象都会设置相应的 `lru`。可以想象的是，每一次访问数据的时候，会更新 `redisObject.lru`。

LRU 数据淘汰机制是这样的：在数据集中随机挑选几个键值对，取出其中 `lru` 最大的键值对淘汰。

不可能遍历key 用当前时间-最近访问 越大 说明 访问间隔时间越长

volatile-lru

从已设置过期时间的数据集（`server.db[i].expires`）中挑选最近最少使用的数据淘汰

allkeys-lru

从数据集（`server.db[i].dict`）中挑选最近最少使用的数据淘汰

LFU

LFU (Least frequently used) 最不经常使用，如果一个数据在最近一段时间内使用次数很少，那么在将来一段时间内被使用的可能性也很小。

volatile-lfu

allkeys-lfu

random

随机

volatile-random

从已设置过期时间的数据集（`server.db[i].expires`）中任意选择数据淘汰

allkeys-random

从数据集（`server.db[i].dict`）中任意选择数据淘汰

ttr

volatile-ttr

从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

redis 数据集数据结构中保存了键值对过期时间的表，即 redisDb.expires。

TTL 数据淘汰机制：从过期时间的表中随机挑选几个键值对，取出其中 ttl 最小的键值对淘汰。

noeviction

禁止驱逐数据，不删除 默认

缓存淘汰策略的选择

- allkeys-lru：在不确定时一般采用策略。冷热数据交换
- volatile-lru：比allkeys-lru性能差 存：过期时间
- allkeys-random：希望请求符合平均分布(每个元素以相同的概率被访问)
- 自己控制：volatile-ttl 缓存穿透

案例分享：字典库失效

key-Value 业务表存 code 显示 文字

拉勾早期将字典库，设置了maxmemory，并设置缓存淘汰策略为allkeys-lru

结果造成字典库某些字段失效，缓存击穿，DB压力剧增，差点宕机。

分析：

字典库：Redis做DB使用，要保证数据的完整性

maxmemory设置较小，采用allkeys-lru，会对没有经常访问的字典库随机淘汰

当再次访问时会缓存击穿，请求会打到DB上。

解决方案：

- 1、不设置maxmemory
- 2、使用noeviction策略

Redis是作为DB使用的，要保证数据的完整性，所以不能删除数据。

可以将原始数据源（XML）在系统启动时一次性加载到Redis中。

Redis做主从+哨兵 保证高可用