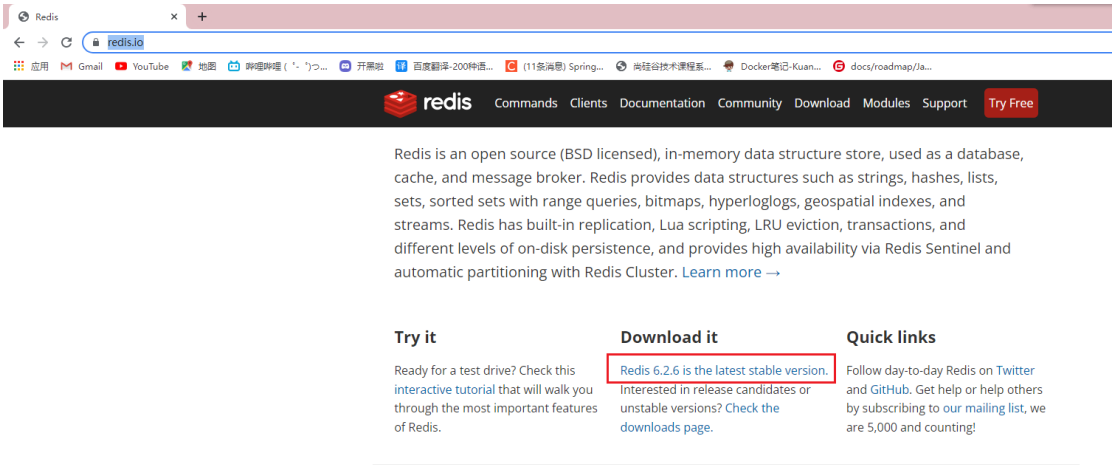
Redis6

一、安装

(一)、进入官网

https://redis.io/



(二)、Centos7 准备环境

安装 C 语言的编译环境

yum install –y centos-release-scl scl-utils-build

yum install -y devtoolset-8-toolchain

scl enable devtoolset-8 bash

gcc –verson

```
[root@localhost ~]# gcc --version
gcc (GCC) 8.3.1 20190311 (Red Hat 8.3.1-3)
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

(三)、上传压缩包

ctrl+p
cd /opt

```
sftp> cd /opt/
sftp> put -r "C:\Users\aa\Downloads\redis-6.2.6.tar.gz"
Uploading redis-6.2.6.tar.gz to /opt/redis-6.2.6.tar.gz
  100% 2418KB    2418KB/s 00:00:00
C:\Users\aa\Downloads\redis-6.2.6.tar.gz: 2476542 bytes transferred in 0 seconds (2418 KB/s)
sftp>
```

(三)、解压文件

tar –zxvf redis-6.2.6.tar.gz
cd redis-6.2.6
make

(四)、安装

make install

(五)、进入/usr/local/bin 查看文件

cd /usr/local/bin
ls

```
[root@localhost bin]# ls
redis-benchmark  redis-check-aof  redis-check-rdb  redis-cli  redis-sentinel  redis-server
[root@localhost bin]#
```

安装成功

(六)、目录介绍

查看默认安装目录：
redis-benchmark:性能测试工具，可以在自己本子运行，看看自己本子性能如何
redis-check-aof：修复有问题的 AOF 文件，rdb 和 aof 后面讲

redis-check-dump：修复有问题的 dump.rdb 文件
redis-sentinel：Redis 集群使用
redis-server：Redis 服务器启动命令
redis-cli：客户端，操作入口

(七)、后台启动

编辑/opt/redis-6.2.6/下的 redis.conf
第 257 行修改为 yes,支持后台启动

```
257 daemonize yes
```

使用 redis-server 加这个路径文件名就能后台启动了
redis-server redis.conf

```
[root@localhost redis-6.2.6]# redis-server redis.conf
[root@localhost redis-6.2.6]#
```

(八)、测试连接

```
[root@localhost redis-6.2.6]# redis-cli
127.0.0.1:6379>
```

(九)、停止 redis

进去服务器

```
[root@localhost redis-6.2.6]# redis-cli
127.0.0.1:6379>
```

shutdown

```
127.0.0.1:6379> shutdown
```

二、常用五大数据类型

(一)、Redis 键(key)操作

keys *　　# 查看当前库所有 key　　(匹配：keys *1)
exists key　　# 判断某个 key 是否存在
type key　　# 查看你的 key 是什么类型
del key　　# 删除指定的 key 数据
unlink key　　# 根据 value 选择非阻塞删除
仅将 keys 从 keyspace 元数据中删除，真正的删除会在后续异步操作。
expire key 10　　# 10 秒钟：为给定的 key 设置过期时间
ttl key　　# 查看还有多少秒过期，-1 表示永不过期，-2 表示已过期
select　　# 命令切换数据库
dbsize　　# 查看当前数据库的 key 的数量
flushdb　　# 清空当前库
flushall　　# 通杀全部库

(二)、Redis 字符串(String)

1、简介

String 是 Redis 最基本的类型，一个 key 对应一个 value。
String 类型是二进制安全的。意味着 Redis 的 string 可以包含任何数据。比如 jpg 图片或者序列化的对象。
String 类型是 Redis 最基本的数据类型，一个 Redis 中字符串 value 最多可以是 512M

2、String 常用命令

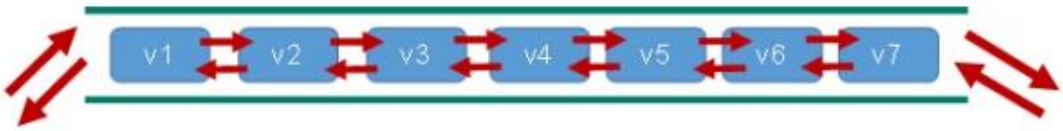
set key value # 如果不存在则创建一个 key value,如果 key 已存在修改 value 的值
get key # 获取 key 的 value 值
strlen key # 获取 key 的长度
setnx key value # 创建一个 key value,如果 key 已存在报错
incr key # 将 key 中储存的值+1,value 必须为数字类型
decr key # 将 key 中储存的值-1,value 必须为数字类型
incr key 步长 # 将 key 中储存的值+步长,value 必须为数字类型
decr key 步长 # 将 key 中储存的值-步长,value 必须为数字类型
mset key value key value # 创建多个 key value
mget key value key value # 获取多个 key 的 value 值
msetnx key value key vale# 创建多个 key value 值,key 不存在情况下
getrange key 3 4 # 获取一个 key 的第三个字符到第四个字符,包前包后
setrange key 3 value # 在第三个字符后面插入一个 value
setex key 过期时间 value # 规定一个 key 过期时间
getset key value # 查看一个 key 的 value,但随后被替换为新的 value

(三)、Redis 列表(List)

1、简介

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）。
它的底层实际是个双向链表，对两端的操作性能很高，通过索引下标的操作中间的节点性能会较差。



2、常用命令

lpush key value1 value2 valu3 # 在左边插入值,结果是 value3 value2 value1
rpush key value1 value2 valu3 # 在右边插入值,结果是 value1 value2 value3
rpoplpush key1 key2 # 从 key1 的右边吐出一个值,就像迭代器一样,每执行一次吐出一个值加到 key2 后面
lrange key value start stop # 取出这个 key 的 start stop 的值,0 -1 是取出全部
lindex key index # 根据 index 取出 value 的值
llen key # 获取 key 的长度
linsert key before value newvalue # 在本来的 value 值后面插入一个值
lrem key n value # 从左边删除 n 个 value
lset key index value # 将 key 下标为 index 的 value 替换为新的 value

(四)、Redis 集合(Set)

1、简介

无序集合

2、常用命令

sadd key value value # 创建一个新的集合
smembers key # 取出该集合所有的值
sismember key value # 判断该集合是否含有 value 值 有 1 无 0
scard key # 返回该集合的元素个数
srem key value1 value2 # 删除集合中的 value 元素
spop key # 随机吐出集合的一个值,吐出就删除了
srandmember key n # 随机从集合中取出 n 个值,不删除
smove key1 key2 value # 从 key1 中取出 value 添加到 key2 中
sinter key1 key2 # 返回两个集合交集的元素,2 个集合都有的值
sunion key1 key2 # 返回两个集合并集的元素,2 个集合全部的值
sdiff key1 key2 # 返回两个集合差集的元素,key1 中没有 key2 的值

(五)、Redis 哈希(Hash)

1、简介

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。

user

2、常用命令

hset key field value　　# 添加一个哈希集合,添加一个值 hset user id 1
hget key field　　# 获取 key 中 field 的 value 值
hmset key field value1 field value2 # 添加一个哈希结合,添加多个值 hmset user id 1 name zhangsan
hexists key field　　# 查看哈希表 key 中,field 是否存在
hkeys key　　# 列出该 hash 集合所有的 value
hincrby key field increment　　# 为哈希表 key 中 field 的 value 值加步长
hsetnx key field value　　# 在哈希表 key 中添加一列 firld-value

(六)、Redis 有序集合(sorted set)

1、简介

Redis 有序集合 zset 与普通集合 set 非常相似，是一个没有重复

元素的字符串集合。
不同之处是有序集合的每个成员都关联了一个**评分（score）**,这个评分（score）被用来按照从最低分到最高分的方式排序集合中的成员。集合的成员是唯一的，但是评分可以是重复了 。

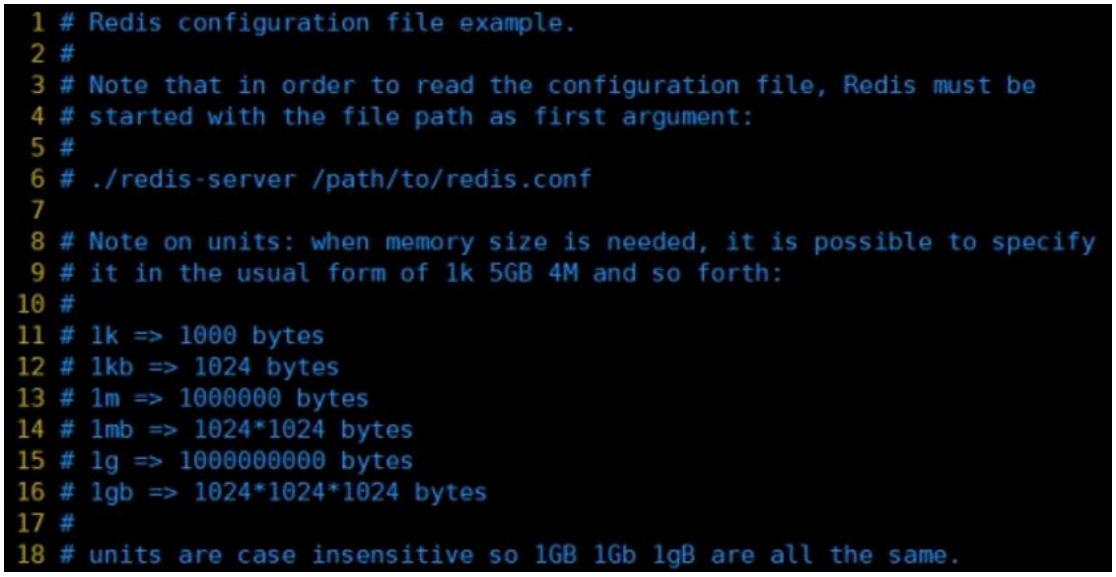
2、常用命令

zadd key 评分 value 评分 value # 创建一个有序集合,根据评分选择先后
zrange key start end withscores # 返回下标为几到几的,withscores 显示评分
zrangebyscore key 评分~评分 [withscores] # 返回评分几到几,withscores 显示评分
zrevrangebyscore key 评分~评分 [withscores] # 返回 key 中频分几到几的值,withscores 显示评分,从大到小排列
zincrby key 步长 value # 为 key 中的 value 评分加步长
zrem key value # 删除指定的值
zcount key min max # 评分几到几的个数
zrank key value # 返回该值在集合中的排名,从 0 开始

三、配置文件介绍

(一)、units 单位

配置大小单位,开头定义了一些基本的度量单位，只支持 bytes，不支持 bit
大小写不敏感



(二)、Incloud 包含

类似 jsp 中的 include，多实例的情况可以把公用的配置文件提取出来

(三)、网络相关配置

1、bind

默认情况 bind=127.0.0.1 只能接受本机的访问请求
不写的情况下，无限制接受任何 ip 地址的访问

生产环境肯定要写你应用服务器的地址；服务器是需要远程访问的，所以需要将其注释掉

如果开启了 protected-mode，那么在没有设定 bind ip 且没有设密码的情况下，Redis 只允许接受本机的响应

```
40 # By default, if no "bind" configuration directive is specified, Redis listens
41 # for connections from all the network interfaces available on the server.
42 # It is possible to listen to just one or multiple selected interfaces using
43 # the "bind" configuration directive, followed by one or more IP addresses.
44 #
45 # Examples:
46 #
47 # bind 192.168.1.100 10.0.0.1
48 # bind 127.0.0.1 ::1
49 #
50 # ~~~ WARNING ~~~ If the computer running Redis is directly exposed to the
51 # internet, binding to all the interfaces is dangerous and will expose the
52 # instance to everybody on the internet. So by default we uncomment the
53 # following bind directive, that will force Redis to listen only into
54 # the IPv4 lookback interface address (this means Redis will be able to
55 # accept connections only from clients running into the same computer it
56 # is running).
57 #
58 # IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES
59 # JUST COMMENT THE FOLLOWING LINE.
60 # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
61 #bind 127.0.0.1
```

2、protected-mod

将本机访问保护模式设置 no

```
63 # Protected mode is a layer of security protection, in order to avoid that
64 # Redis instances left open on the internet are accessed and exploited.
65 #
66 # When protected mode is on and if:
67 #
68 # 1) The server is not binding explicitly to a set of addresses using the
69 #    "bind" directive.
70 # 2) No password is configured.
71 #
72 # The server only accepts connections from clients connecting from the
73 # IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain
74 # sockets.
75 #
76 # By default protected mode is enabled. You should disable it only if
77 # you are sure you want clients from other hosts to connect to Redis
78 # even if no authentication is configured, nor a specific set of interfaces
79 # are explicitly listed using the "bind" directive.
80 protected-mode no
```

3、post

端口号，默认 6379

```
82 # Accept connections on the specified port, default is 6379 (IANA #815344).
83 # If port 0 is specified Redis will not listen on a TCP socket.
84 port 6379
```

4、tcp-backlog

设置 tcp 的 backlog，backlog 其实是一个连接队列，backlog 队列总和=未完成三次握手队列 + 已经完成三次握手队列。
在高并发环境下你需要一个高 backlog 值来避免慢客户端连接问题。

```
86 # TCP listen() backlog.
87 #
88 # In high requests-per-second environments you need an high backlog in order
89 # to avoid slow clients connections issues. Note that the Linux kernel
90 # will silently truncate it to the value of /proc/sys/net/core/somaxconn so
91 # make sure to raise both the value of somaxconn and tcp_max_syn_backlog
92 # in order to get the desired effect.
93 tcp-backlog 511
```

5、timeout

当客户端连接停止多少秒不操作停止关闭连接, 0 表示关闭该功能。即永不关闭。

```
 95 # Unix socket.
 96 #
 97 # Specify the path for the Unix socket that will be used to listen for
 98 # incoming connections. There is no default, so Redis will not listen
 99 # on a unix socket when not specified.
100 #
101 # unixsocket /tmp/redis.sock
102 # unixsocketperm 700
103
104 # Close the connection after a client is idle for N seconds (0 to disable)
105 timeout 0
```

6、tcp-keepalive

时隔多少秒对客户端进行检测是否存活
对访问客户端的一种心跳检测，每个 n 秒检测一次。
单位为秒，如果设置为 0，则不会进行 Keepalive 检测，建议设置成 60

```
107 # TCP keepalive.
108 #
109 # If non-zero, use SO_KEEPALIVE to send TCP ACKs to clients in absence
110 # of communication. This is useful for two reasons:
111 #
112 # 1) Detect dead peers.
113 # 2) Take the connection alive from the point of view of network
114 #    equipment in the middle.
115 #
116 # On Linux, the specified value (in seconds) is the period used to send ACKs.
117 # Note that to close the connection the double of the time is needed.
118 # On other kernels the period depends on the kernel configuration.
119 #
120 # A reasonable value for this option is 300 seconds, which is the new
121 # Redis default starting with Redis 3.2.1.
122 tcp-keepalive 300
```

(四)、General

1、daemonize

是否为后台进程，设置为 yes

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

2、pidfile

存放 pid 文件的位置，每个实例会产生一个不同的 pid 文件

```
141 # If a pid file is specified, Redis writes it where specified at startup
142 # and removes it at exit.
143 #
144 # When the server runs non daemonized, no pid file is created if none is
145 # specified in the configuration. When the server is daemonized, the pid file
146 # is used even if not specified, defaulting to "/var/run/redis.pid".
147 #
148 # Creating a pid file is best effort: if Redis is not able to create it
149 # nothing bad happens, the server will start and run normally.
150 pidfile /var/run/redis_6379.pid
```

3、loglevel

指定日志记录级别，Redis 总共支持四个级别：debug、verbose、notice、warning，默认为 **notice**
四个级别根据使用阶段来选择，生产环境选择 notice 或者 warning

```
152 # Specify the server verbosity level.
153 # This can be one of:
154 # debug (a lot of information, useful for development/testing)
155 # verbose (many rarely useful info, but not a mess like the debug level)
156 # notice (moderately verbose, what you want in production probably)
157 # warning (only very important / critical messages are logged)
158 loglevel notice
```

4、database 16

设定库的数量 默认 16，默认数据库为 0，可以使用 SELECT <dbid>命令在连接上指定数据库 id

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

(五)、security 安全

1、设置密码

```
467 ################################## SECURITY ###################################
468
469 # Require clients to issue AUTH <PASSWORD> before processing any other
470 # commands.  This might be useful in environments in which you do not trust
471 # others with access to the host running redis-server.
472 #
473 # This should stay commented out for backward compatibility and because most
474 # people do not need auth (e.g. they run their own servers).
475 #
476 # Warning: since Redis is pretty fast an outside user can try up to
477 # 150k passwords per second against a good box. This means that you should
478 # use a very strong password otherwise it will be very easy to break.
479 #
480 # requirepass foobared
```

访问密码的查看、设置和取消
在命令中设置密码，只是临时的。重启 redis 服务器，密码就还

原了。
永久设置，需要再配置文件中进行设置。

```
127.0.0.1:6379> config get requirepass
1) "requirepass"
2) ""
127.0.0.1:6379> config set requirepass "123456"
OK
127.0.0.1:6379> config get requirepass
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> config set requirepass ""
OK
127.0.0.1:6379> config get requirepass
1) "requirepass"
2) ""
127.0.0.1:6379>
```

(六)、limits 限制

1、maxclients

设置 redis 同时可以与多少个客户端进行连接。
默认情况下为 10000 个客户端。
如果达到了此限制，redis 则会拒绝新的连接请求，并且向这些连接请求方发出“max number of clients reached”以作回应。

```
503 # Set the max number of connected clients at the same time. By default
504 # this limit is set to 10000 clients, however if the Redis server is not
505 # able to configure the process file limit to allow for the specified limit
506 # the max number of allowed clients is set to the current file limit
507 # minus 32 (as Redis reserves a few file descriptors for internal uses).
508 #
509 # Once the limit is reached Redis will close all the new connections sending
510 # an error 'max number of clients reached'.
511 #
512 # maxclients 10000
```

2、maxmemory

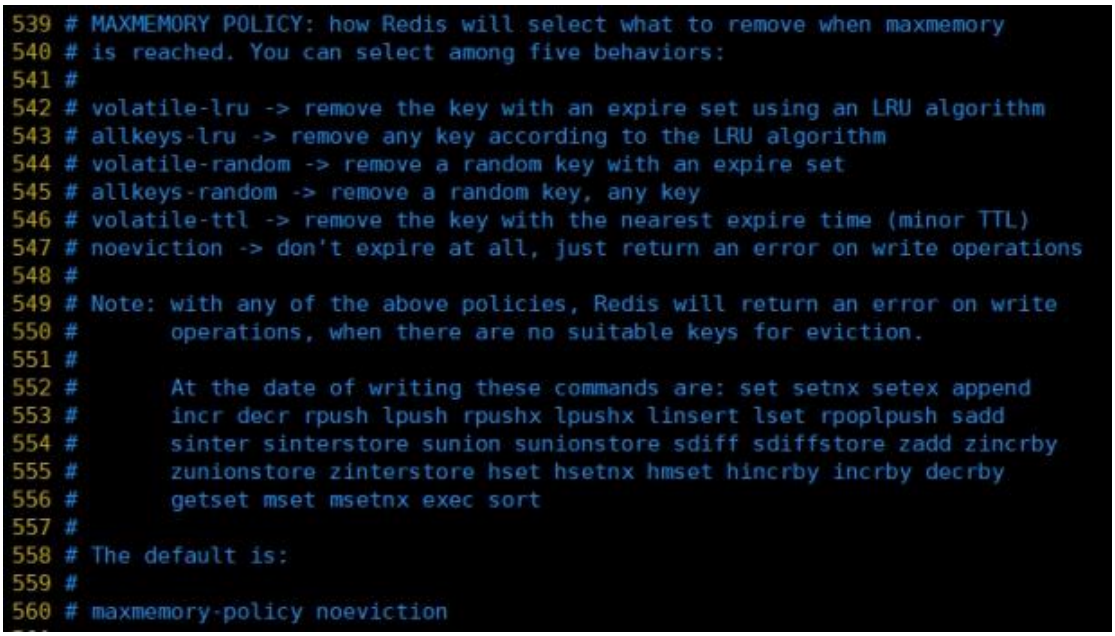
建议**必须设置**，否则，将内存占满，造成服务器宕机
设置 redis 可以使用的内存量。一旦到达内存使用上限，redis 将会试图移除内部数据，移除规则可以通过 maxmemory-policy 来指定。
如果 redis 无法根据移除规则来移除内存中的数据，或者设置了“不允许移除”，那么 redis 则会针对那些需要申请内存的指令返回错误信息，比如 SET、LPUSH 等。
但是对于无内存申请的指令，仍然会正常响应，比如 GET 等。
如果你的 redis 是主 redis（说明你的 redis 有从 redis），那么在设置内存使用上限时，需要在系统中留出一些内存空间给同步队列缓存，只有在你设置的是“不移除”的情况下，才不用考虑这个因素。

```
514 # Don't use more memory than the specified amount of bytes.
515 # When the memory limit is reached Redis will try to remove keys
516 # according to the eviction policy selected (see maxmemory-policy).
517 #
518 # If Redis can't remove keys according to the policy, or if the policy is
519 # set to 'noeviction', Redis will start to reply with errors to commands
520 # that would use more memory, like SET, LPUSH, and so on, and will continue
521 # to reply to read-only commands like GET.
522 #
523 # This option is usually useful when using Redis as an LRU cache, or to set
524 # a hard memory limit for an instance (using the 'noeviction' policy).
525 #
526 # WARNING: If you have slaves attached to an instance with maxmemory on,
527 # the size of the output buffers needed to feed the slaves are subtracted
528 # from the used memory count, so that network problems / resyncs will
529 # not trigger a loop where keys are evicted, and in turn the output
530 # buffer of slaves is full with DELs of keys evicted triggering the deletion
531 # of more keys, and so forth until the database is completely emptied.
532 #
533 # In short... if you have slaves attached it is suggested that you set a lower
534 # limit for maxmemory so that there is some free RAM on the system for slave
535 # output buffers (but this is not needed if the policy is 'noeviction').
536 #
537 # maxmemory <bytes>
```

3、maxmemory-policy

volatile-lru：使用 LRU 算法移除 key，只对设置了过期时间

的键；（最近最少使用）
allkeys-lru：在所有集合 key 中，使用 LRU 算法移除 key
volatile-random：在过期集合中移除随机的 key，只对设置了过期时间的键
allkeys-random：在所有集合 key 中，移除随机的 key
volatile-ttl：移除那些 TTL 值最小的 key，即那些最近要过期的 key
noeviction：不进行移除。针对写操作，只是返回错误信息



4、maxmemory-samples

设置样本数量，LRU 算法和最小 TTL 算法都并非是精确的算法，而是估算值，所以你可以设置样本的大小，redis 默认会检查这么多个 key 并选择其中 LRU 的那个。
一般设置 3 到 7 的数字，数值越小样本越不准确，但性能消耗越小。

```
562 # LRU and minimal TTL algorithms are not precise algorithms but approximated
563 # algorithms (in order to save memory), so you can tune it for speed or
564 # accuracy. For default Redis will check five keys and pick the one that was
565 # used less recently, you can change the sample size using the following
566 # configuration directive.
567 #
568 # The default of 5 produces good enough results. 10 Approximates very closely
569 # true LRU but costs a bit more CPU. 3 is very fast but not very accurate.
570 #
571 # maxmemory-samples 5
```

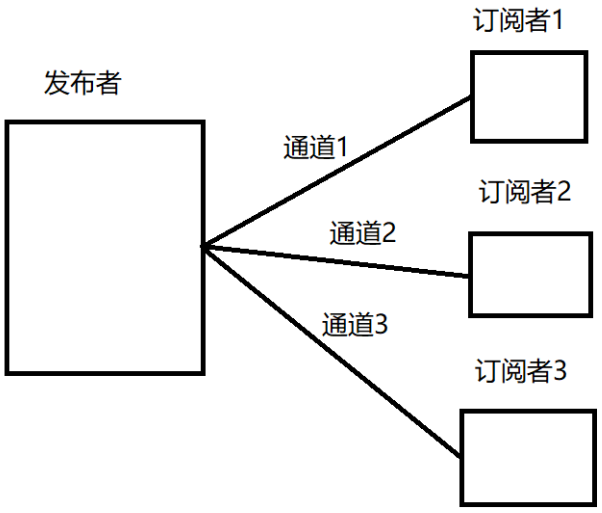
四、

Redis 的发布和订阅

(一)、什么是发布和订阅

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

Redis 客户端可以订阅任意数量的



发布者在通道1发布了消息hello
订阅者因为订阅了通道1,所以可以接受到hello的消息
而订阅者2和订阅者3无法收到,因为没有订阅通道1
订阅者可以订阅多个频道

(二)、发布和订阅的实现

1、打开一个客户端订阅通道一

subscribe channel1

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

2、再打开一个客户端在 channel1 发布消息

publish channel1 hello

```
127.0.0.1:6379> publish channel1 hello
(integer) 1
```

返回的 1 是订阅者数量

3、打开第一个客户端可以看到发送的信息

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

五、Redis 新数据类型

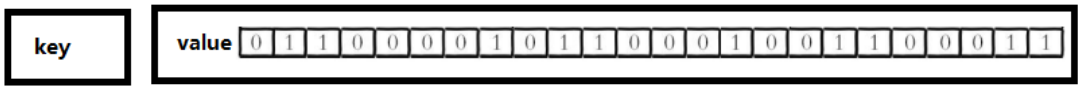
(一)、BitMaps

1、 简介

Bitmaps 本身不是一种数据类型，实际上它就是把字符串(key-

value),但是它可以对字符串的位进行操作

Bitmaps 是一个位单位的数组，数组的每个单元只能存储 0 和 1，数组下标 bitmaps 中叫做偏移量



2、 命令

Setbit key offset value # 创建一个 Bitmaps 中某个偏移量的值 0 或 1

演示案例：

当某某用户访问网站的时候就把他的偏移量变为 1

设置 1 6 11 15 19 号用户访问过网站

```
127.0.0.1:6379> setbit unique:users:20201106 1 1
(integer) 0
127.0.0.1:6379> setbit unique:users:20201106 6 1
(integer) 0
127.0.0.1:6379> setbit unique:users:20201106 11 1
(integer) 0
127.0.0.1:6379> setbit unique:users:20201106 15 1
(integer) 0
127.0.0.1:6379> setbit unique:users:20201106 19 1
(integer) 0
```

Getbit key offset # 获取 bitmaps 中的某个偏移量

演示案例：

获取 id 为 8 的用户是否在某天访问过，返回 0 说明没有访问过：

```
127.0.0.1:6379> getbit unique:users:20201106 8
(integer) 0
127.0.0.1:6379> getbit unique:users:20201106 1
(integer) 1
127.0.0.1:6379> getbit unique:users:20201106 100
(integer) 0
```

Bitcount key # 获取这个 key 中为 1 的数量

Bitcount key start end # 计算 start*8 到 end*8 中 1 的个数

举例： K1 【01000001 01000000 00000000 00100001】，对应【0，1，2，3】

Bitop and(or/not/xor) destkey key # bitop 是一个复合操作，它可以做多个 Bitmaps 的 and(交集)、or、(并集) 、not(非) 、xor(异或) 操作并将结果保存在 destkey 中。
演示案例：
2020-11-04 日访问网站的 userid=1,2,5,9。
setbit unique:users:20201104 1 1
setbit unique:users:20201104 2 1
setbit unique:users:20201104 5 1
setbit unique:users:20201104 9 1

2020-11-03 日访问网站的 userid=0,1,4,9。
setbit unique:users:20201103 0 1
setbit unique:users:20201103 1 1
setbit unique:users:20201103 4 1
setbit unique:users:20201103 9 1

计算出两天都访问过网站的用户数量
bitop and unique:users:and:20201104_03 unique:users:20201103 unique:users:20201104
Bitop 并集 暂时存储的集合 20201103 号 20201104 号

3、 说明

用户多的时候用 bitmaps 用户少的时候使用 key-value

(二)、HyperLogLog

1、简介

用于记录不重复的数据,比如说在这个类型中的一个 key 中存储

了 java,再存 java 就失败了

2、命令

pfadd key 值 1 值 2 # 创建一个 set 差不多的数据类型

```
127.0.0.1:6379> pfadd hll1 "redis"
(integer) 1
127.0.0.1:6379> pfadd hll1 "mysql"
(integer) 1
127.0.0.1:6379> pfadd hll1 "redis"
(integer) 0
127.0.0.1:6379>
```

pfcount key [key…] # 统计 1 个或多个 key 中不重复的数据

```
127.0.0.1:6379> pfadd hll1 "redis"
(integer) 1
127.0.0.1:6379> pfadd hll1 "mysql"
(integer) 1
127.0.0.1:6379> pfadd hll1 "redis"
(integer) 0
127.0.0.1:6379> pfcount hll1
(integer) 2
127.0.0.1:6379> pfadd hll2 "redis"
(integer) 1
127.0.0.1:6379> pfadd hll2 "mongodb"
(integer) 1
127.0.0.1:6379> pfcount hll1 hll2
(integer) 3
127.0.0.1:6379>
```

pfmerge key3 key2 key1 # 将 key1 和 key2 的值合并到 key3 中

```
127.0.0.1:6379> pfcount hll1 hll2
(integer) 3
127.0.0.1:6379> pfmerge hll3 hll1 hll2
OK
127.0.0.1:6379> pfcount hll3
(integer) 3
127.0.0.1:6379>
```

(三)、Geospatial

1、简介

对地理位置的操作

2、命令

geoadd key 经度 纬度 名称 # 添加一个或多个地理位置坐标

```
127.0.0.1:6379> geoadd china:city 121.47 31.23 shanghai
(integer) 1
127.0.0.1:6379> geoadd china:city 106.50 29.53 chongqing 114.05 22.52 shenzhen 116.38 39.90 beijing
(integer) 3
127.0.0.1:6379>
```

两极无法直接添加，一般会下载城市数据，直接通过 Java 程序一次性导入。
有效的经度从 -180 度到 180 度。有效的纬度从 -85.05112878 度到 85.05112878 度。
当坐标位置超出指定范围时，该命令将会返回一个错误。
已经添加的数据，是无法再次往里面添加的。

geopos key 城市名 # 返回地理位置的经度,纬度

```
127.0.0.1:6379> geopos china:city shanghai
1) 1) "121.47000163793563843"
   2) "31.22999903975783553"
```

geodist key 地理位置 地理位置 [m|km|ft|mi] # 获取 2 个位置之间的直线距离

```
127.0.0.1:6379> geodist china:city beijing shanghai km
"1087.4816"
```

单位：
m 表示单位为米[默认值]。

km 表示单位为千米。
mi 表示单位为英里。
ft 表示单位为英尺。
如果用户没有显式地指定单位参数， 那么 GEODIST 默认使用米作为单位
georadius key 经度 纬度 距离 单位 # 找出经度纬度半径距离画圆之内的城市

六、Redis_Jedis

(一)、环境配置

1、 Jedis 需要的 jar 包

创建一个普通的 Maven---导入 jar 包

```
<dependencies>
    <dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
        <version>3.2.0</version>
    </dependency>
</dependencies>
```

2、 linux 开启对外开放

vi /opt/redis-6.2.6/redis.conf
75 行注释

```
75 #bind 127.0.0.1 -::1
```

94 行改为 no

```
94 protected-mode no
```

关闭防火墙和 selinux

```
[root@localhost bin]# systemctl stop firewalld
[root@localhost bin]# setenforce 0
[root@localhost bin]#
```

3、 创建测试类

```
@Test
public void test1(){
        Jedis jedis = new Jedis("192.168.2.166", 6379);
        String pong = jedis.ping();
        System.out.println("连接成功:"+pong);
        jedis.close();
}
```

```
Tests passed: 1 of 1 test – 27 ms
E:\java-11-openjdk-11.0.11.9-1.windows.redhat.x
连接成功:PONG

Process finished with exit code 0
```

(二)、key

```
@Test
public void test2(){
        //设置第一个 key 值
        jedis.set("k1","value1");
        //设置第二个 key 值
        jedis.set("k2","value2");
        //获取对应 key 的 value
        jedis.get("k1");
        //获取所有的 key
        Set<String> keys = jedis.keys("*");
```

```
        for (String key : keys) {
                System.out.println(key);
        }
        //查看这个 key 是否有值
        System.out.println(jedis.exists("k1"));
        //查看这个 key 的存活时间
        System.out.println(jedis.ttl("k2"));
        jedis.close();
}
```

(三)、String

```
@Test
public void test3(){
        //添加多个值
        jedis.mset("k1","value1","k2","value2");
        //获取多个 key 的 value 值
        System.out.println(jedis.mget("k1","k2"));
        jedis.close();
}
```

(四)、List

```
@Test
public void test4(){
        //创建一个 list,存放 2 个值
        jedis.rpush("myList","aaa","bbb");
        //获取 mylist 的 value
        List<String> myList = jedis.lrange("myList", 0, -1);
        for (String s : myList) {
                System.out.println(s);
```

```
    }
}
```

(五)、Set

```
@Test
public void test5(){
    //创建一个 set 添加值
    jedis.sadd("myset","aaa");
    jedis.sadd("myset","bbb");
    jedis.sadd("myset","ccc");
    //获取值
    Set<String> myset = jedis.smembers("myset");
    for (String s : myset) {
        System.out.println(s);
    }
    //查看 value 在 set 的哪一个位置
    System.out.println(jedis.srem("myset", "aaa"));
}
```

(六)、Hash

```
@Test
public void test6(){
    //创建一个 hash 的 person 并赋值
    jedis.hset("person", "id","1" );
    jedis.hset("person", "name", "zhangsan");
    //获取 pserson 的 name 中的 value
    System.out.println(jedis.hget("person", "name"));
    //声明一个 map
    Map<String, String> map = new HashMap<>();
```

```
        map.put("dianhua", "18595926383");
        map.put("address", "192.168.1.1");
        map.put("email", "299@qq.cm");
        //创建一个 set 把 map 信息装进去
        jedis.hmset("personInfo", map);
        //获得一个 key 中属性的值
        List<String> hmget = jedis.hmget("personInfo", "dianhua", "address");
        for (String s : hmget) {
                System.out.println(s);
        }
}
```

(七)、zset

```
@Test
public void test7(){
        jedis.zadd("myZadd", 100, "a");
        jedis.zadd("myZadd", 90, "b");
        jedis.zadd("myZadd", 80, "c");
        Set<String> zadd = jedis.zrange("myZadd", 0, -1);
        for (String s : zadd) {
                System.out.println(s);
        }
}
```

七、SpringBoot 整合 Redis

(一)、导入 jar 包

```
<!-- redis -->
<dependency>
        <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

(二)、application 配置 redis

```
#Redis 服务器地址
spring.redis.host=192.168.2.166
#Redis 服务器连接端口
spring.redis.port=6379
#Redis 数据库索引（默认为 0）
spring.redis.database=0
#连接超时时间（毫秒）
spring.redis.timeout=1800000
#连接池最大连接数（使用负值表示没有限制）
spring.redis.lettuce.pool.max-active=20
#最大阻塞等待时间(负数表示没限制)
spring.redis.lettuce.pool.max-wait=-1
#连接池中的最大空闲连接
spring.redis.lettuce.pool.max-idle=5
#连接池中的最小空闲连接
spring.redis.lettuce.pool.min-idle=0
```

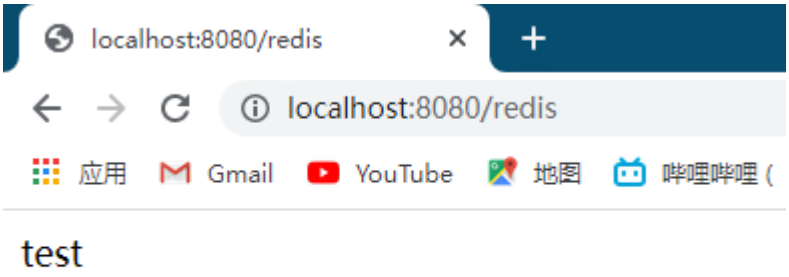
(三)、导入配置类(固定写法)

```
@EnableCaching
@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
```

```
        RedisSerializer<String> redisSerializer = new StringRedisSerializer();
        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer(Object.class);
        ObjectMapper om = new ObjectMapper();
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        jackson2JsonRedisSerializer.setObjectMapper(om);
        template.setConnectionFactory(factory);
        //key 序列化方式
        template.setKeySerializer(redisSerializer);
        //value 序列化
        template.setValueSerializer(jackson2JsonRedisSerializer);
        //value hashmap 序列化
        template.setHashValueSerializer(jackson2JsonRedisSerializer);
        return template;
    }

    @Bean
    public CacheManager cacheManager(RedisConnectionFactory factory) {
        RedisSerializer<String> redisSerializer = new StringRedisSerializer();
        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer(Object.class);
        //解决查询缓存转换异常的问题
        ObjectMapper om = new ObjectMapper();
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        jackson2JsonRedisSerializer.setObjectMapper(om);
        // 配置序列化（解决乱码的问题）,过期时间 600 秒
        RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()
                .entryTtl(Duration.ofSeconds(600))
                .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(redisSeri
alizer))
```

```
                .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(jackso
n2JsonRedisSerializer))
                .disableCachingNullValues();
        RedisCacheManager cacheManager = RedisCacheManager.builder(factory)
                .cacheDefaults(config)
                .build();
        return cacheManager;
    }

}
```

(四)、创建 Controller 进行测试



八、事务_锁_秒杀

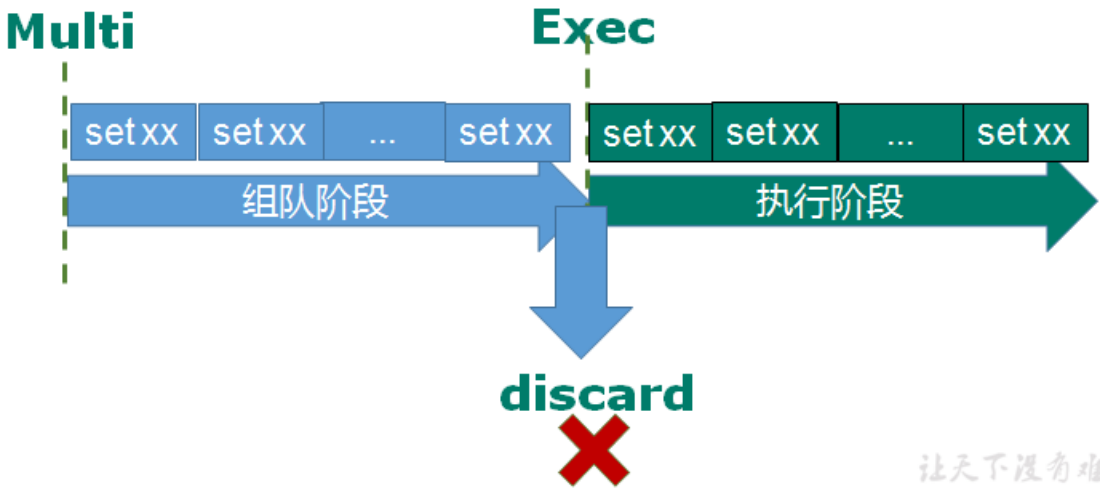
(一)、事务的定义

Redis 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
Redis 事务的主要作用就是串联多个命令防止别的命令插队。

(二)、Multi(组队)、Exec(执行)、discard(取消组队)

1、介绍

从输入 Multi 命令开始，输入的命令都会依次进入命令队列中，但不会执行，直到输入 Exec 后，Redis 会将之前的命令队列中的命令依次执行。



2、案例

2.1、三种情况:

组队执行成功----------顺利执行
组队失败,执行成功---所有失败
组队成功,执行失败---就执行失败的哪一条失败

2.2、组队执行成功

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set k2 v2
QUEUED
127.0.0.1:6379> set k3 v3
QUEUED
127.0.0.1:6379> exec
1) OK
2) OK
```

2.2、组队阶段失败,全部失败

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set m1 v1
QUEUED
127.0.0.1:6379> set m2
(error) ERR wrong number of arguments for 'set' command
127.0.0.1:6379> set m3 v3
QUEUED
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
```

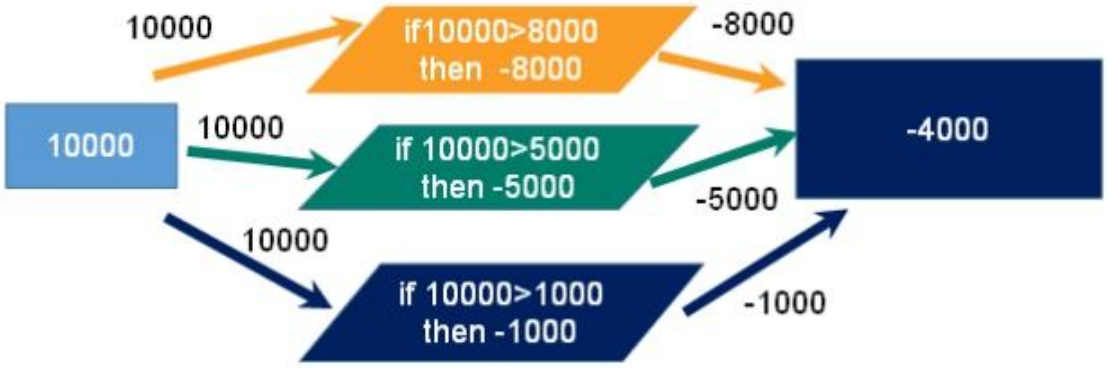
2.3、组队成功,执行失败

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set m1 v1
QUEUED
127.0.0.1:6379> incr m1
QUEUED
127.0.0.1:6379> set m2 v2
QUEUED
127.0.0.1:6379> exec
1) OK
2) (error) ERR value is not an integer or out of range
3) OK
```

(三)、事务冲突问题

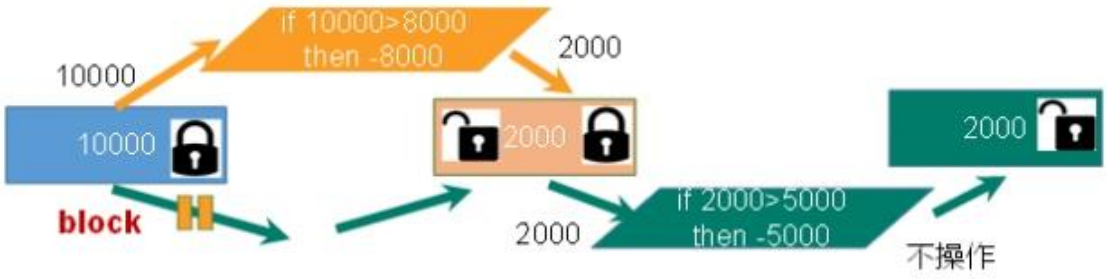
一个请求想给金额减 8000
一个请求想给金额减 5000

一个请求想给金额减 1000



(四)、锁

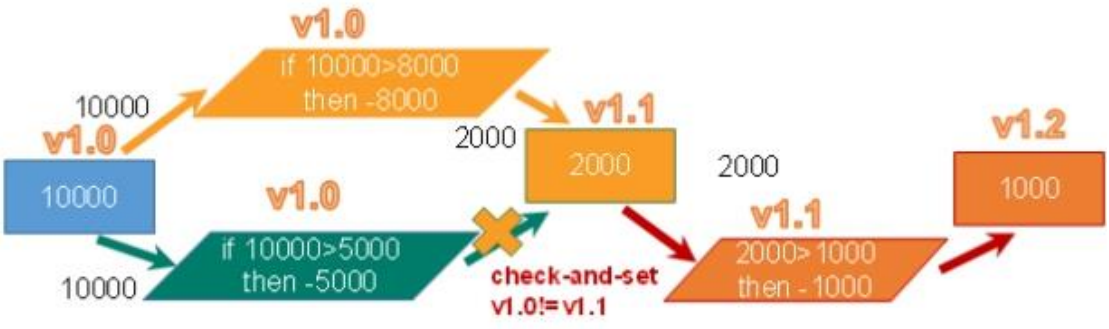
1、悲观锁



当一个去操作数据是给数据上锁,其他人不能使用该数据,当使用完释放锁

传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

2、乐观锁



当一个人去操作数据的时候不会上锁,其他人也可以拿数据,但是当一个人修改完数据,其他人操作的时候会去检测是否有人修改了数据,如果有人修改了数据,则进行更新

乐观锁适用于多读的应用类型，这样可以提高吞吐量。Redis 就是利用这种 check-and-set 机制实现事务的。

(五)、乐观锁演示

1、第一个客户端

```
set blance 100          //创建一个 key
watch blance            //监视这个 key
multi                 //开启事务组队模式
decrby blance 10        //给这个 key+10
exec                  //执行+10 操作
```

2、第二个客户端

```
watch blance            //第二个也监视这个 key
multi                 //开启事务组队
decrby blance 10        //同样进行+10 操作
exec                  //开始执行
```

(null)　　//执行失败,因为第一个客户端已经修改了这个值

(六)、unwatch

1、取消 WATCH 命令对所有 key 的监视。
2、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。
3、因为 EXEC 命令会执行事务，因此 WATCH 命令的效果已经产生了；而 DISCARD 命令在取消事务的同时也会取消所有对 key 的监视，因此这两个命令执行之后，就没有必要执行 UNWATCH 了。

(七)、Redis 事务三特性

- 单独的隔离操作
  - 事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
- 没有隔离级别的概念
  - 队列中的命令没有提交之前都不会实际被执行，因为事务提交前任何指令都不会被实际执行
- 不保证原子性
  - 事务中如果有一条命令执行失败，其后的命令仍然会被执行，没有回滚

(八)、Redis 事务--秒杀并发模拟

使用工具 ab 模拟测试
CentOS6 默认安装
CentOS7 需要手动安装

1、准备处理业务程序

```
//秒杀过程
public static boolean doSecKill(String uid, String prodid) throws IOException {
    //判断参数是否为空
    if (uid == null && prodid == null) {
        return false;
    }
    //连接数据库
    Jedis jedis = new Jedis("192.168.2.166", 6379);



    //拼接字符串
    String kckey = "sk:" + prodid + ":qt";
    String userkey = "sk:" + prodid + ":user";
    //判断秒杀是否开始
    String kc = jedis.get(kckey);
    if (kc == null) {
        System.out.println("秒杀还没有开始");
        jedis.close();
        return false;
    }
    //判断用户是否秒杀成功过
    if (jedis.sismember(userkey, uid)) {
        System.out.println("您已经秒杀过了");
        jedis.close();
        return false;
    }
    //判断商品是否为空
    if(Integer.parseInt(kc)<=0){
        System.out.println("商品已被抢空");
        jedis.close();
```

```
            return false;
        }
        //库存-1
        jedis.decr(kckey);
        System.out.println("秒杀成功");
        //添加抢购成功用户 id
        jedis.sadd(userkey, uid);
        return true;
    }
```

2、 联网安装

yum –y install httpd-tools

3、 在当前文件夹下创建一个 postfile 文件

vi postfile
prodid=0101&

4、 在 redis 中创建商品的 key

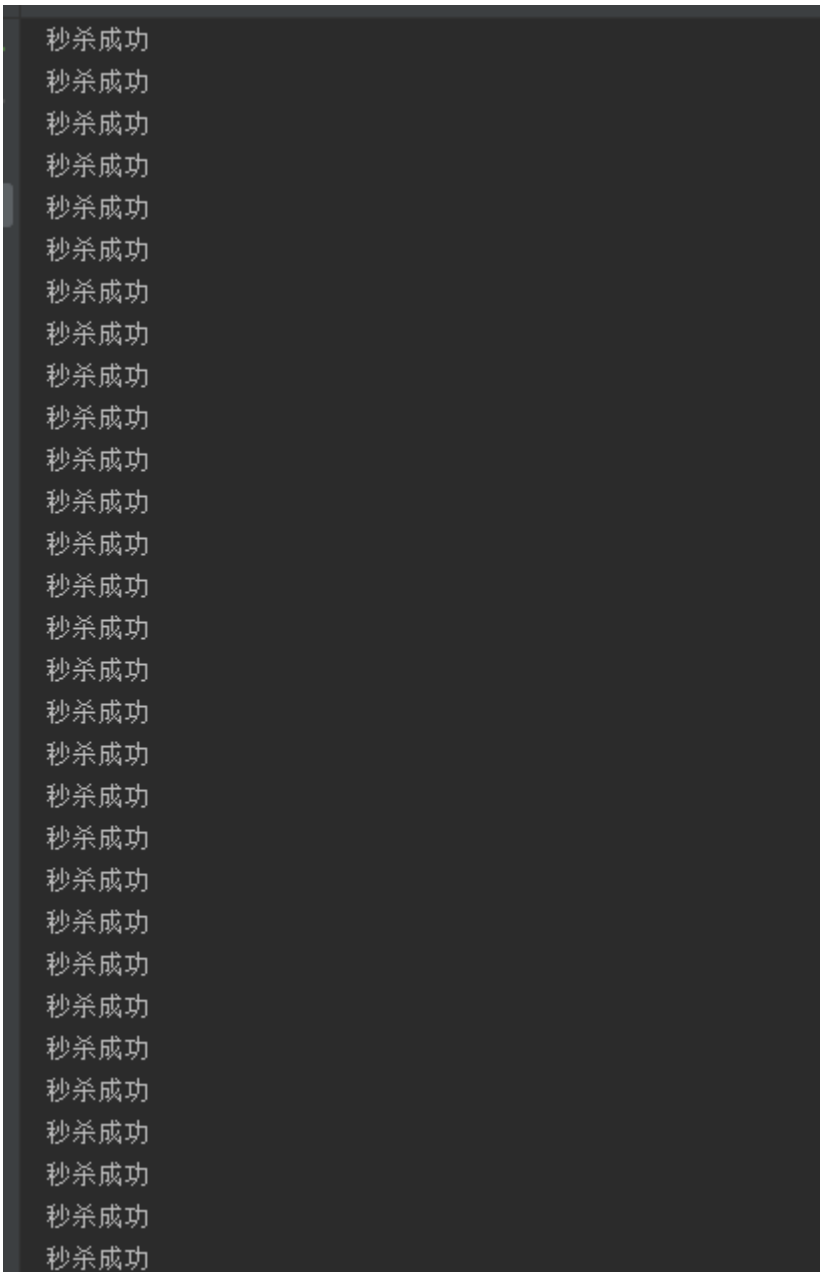
set sk:0101:qt 10

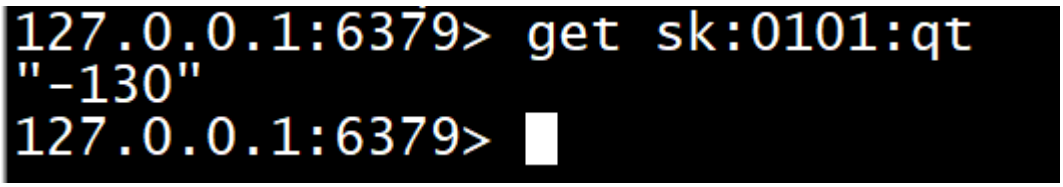
5、 并发测试

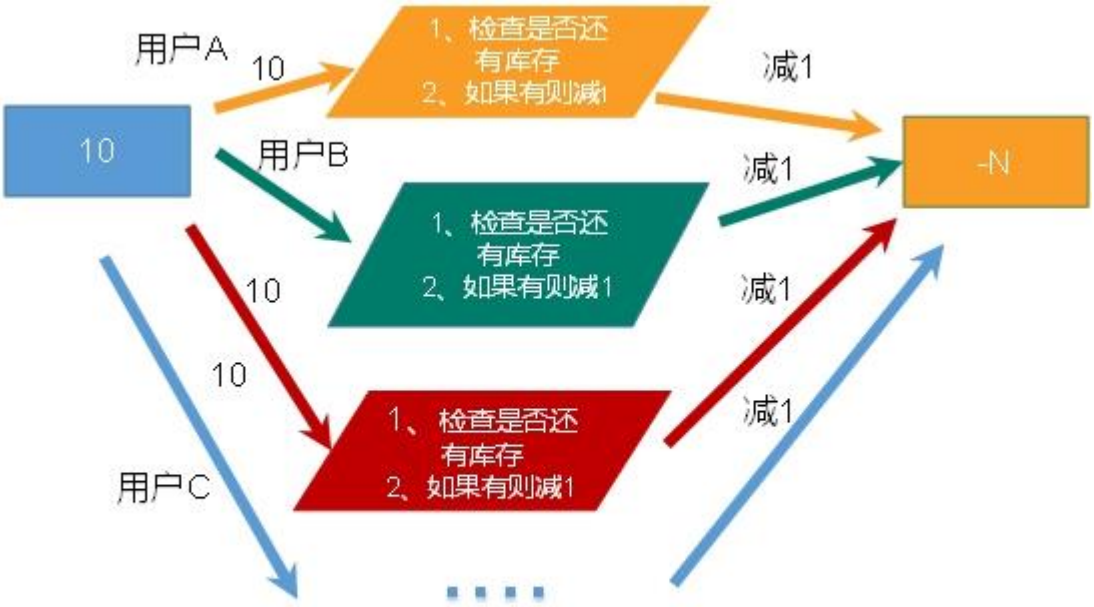
2000 连接数　　200 个位并发请求

ab -n 2000 -c 200 -k -p ~/postfile -T application/x-www-form-urlencoded http://192.168.2.115:8081/Seckill/doseckill

6、 发现商品溢出


秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功
秒杀成功


127.0.0.1:6379> get sk:0101:qt
"-130"
127.0.0.1:6379>

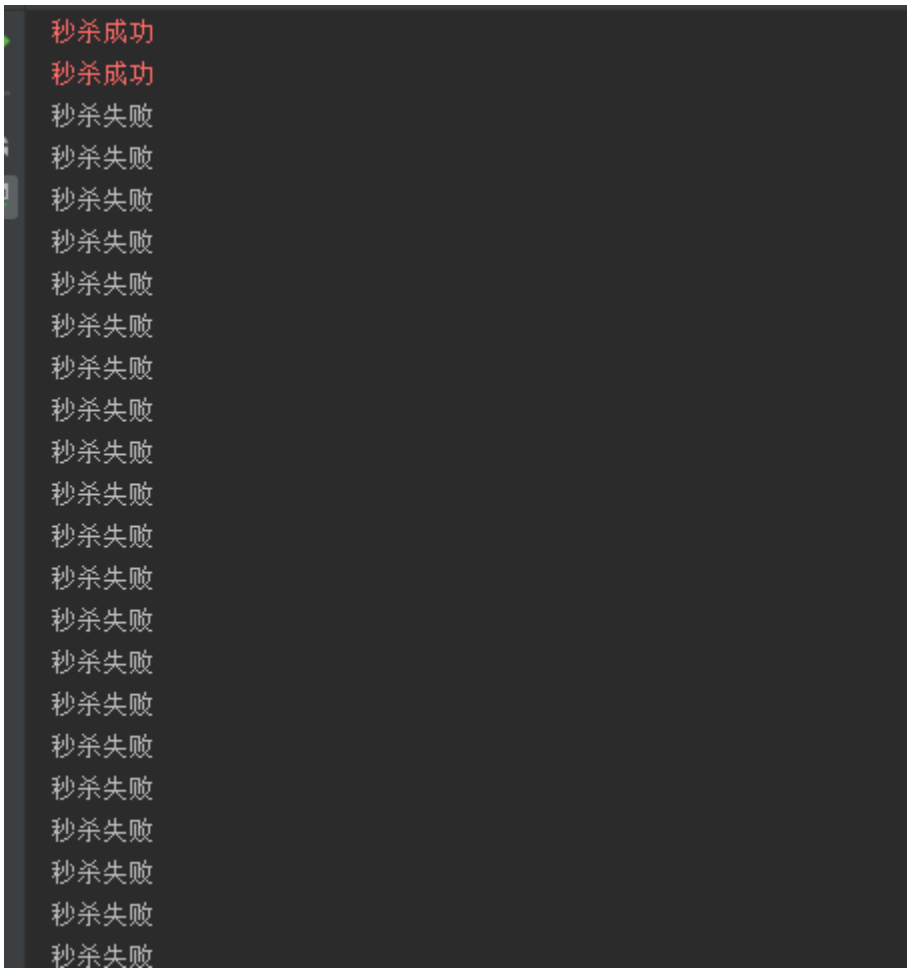


(九)、乐观锁解决超卖问题

```
//秒杀过程
public static boolean doSecKill(String uid, String prodid) throws IOException {
    //判断参数是否为空
    if (uid == null && prodid == null) {
        return false;
    }
    //连接数据库
    Jedis jedis = new Jedis("192.168.2.166", 6379);

    //拼接字符串
    String kckey = "sk:" + prodid + ":qt";
    String userkey = "sk:" + prodid + ":user";

    //增加乐观锁
    jedis.watch(kckey);

    //判断秒杀是否开始
```

```
String kc = jedis.get(kckey);
if (kc == null || "".equals(kc.trim())) {
        System.out.println("秒杀还没有开始");
        jedis.close();
        return false;
}
//判断用户是否秒杀成功过
if (jedis.sismember(userkey, uid)) {
        System.out.println("您已经秒杀过了");
        jedis.close();
        return false;
}
//判断商品是否为空
if(Integer.parseInt(kc)<=0){
        System.out.println("商品已被抢空");
        jedis.close();
        return false;
}

//增加事务
Transaction multi = jedis.multi();
//库存-1
multi.decr(kckey);
//添加抢购成功用户 id
multi.sadd(userkey, uid);
List<Object> exec = multi.exec();
if(exec==null||exec.size()==0){
        System.out.println("秒杀失败");
        jedis.close();
        return false;
}
System.err.println("秒杀成功");
```

```
    jedis.close();
    return true;
}
```



九、连接池

节省每次连接 redis 服务带来的消耗，把连接好的实例反复利用。
通过参数管理连接的行为
代码见项目中

- 链接池参数
  - MaxTotal：控制一个 pool 可分配多少个 jedis 实例，通过 pool.getResource()来获取；如果赋值为-1，则表示不限

制；如果 pool 已经分配了 MaxTotal 个 jedis 实例，则此时 pool 的状态为 exhausted。

- maxIdle：控制一个 pool 最多有多少个状态为 idle(空闲)的 jedis 实例；
- MaxWaitMillis：表示当 borrow 一个 jedis 实例时，最大的等待毫秒数，如果超过等待时间，则直接抛 JedisConnectionException；
- testOnBorrow：获得一个 jedis 实例的时候是否检查连接可用性（ping()）；如果为 true，则得到的 jedis 实例均是可用的；

```
public static JedisPool getJedisPoolInstance() {
    if (null == jedisPool) {
        synchronized (JedisPoolUtil.class) {
            if (null == jedisPool) {
                JedisPoolConfig poolConfig = new JedisPoolConfig();
                poolConfig.setMaxTotal(200);
                poolConfig.setMaxIdle(32);
                poolConfig.setMaxWaitMillis(100*1000);
                poolConfig.setBlockWhenExhausted(true);
                poolConfig.setTestOnBorrow(true);   // ping   PONG
                jedisPool = new JedisPool(poolConfig, "192.168.44.168", 6379, 60000 );
            }
        }
    }
    return jedisPool;
}

public static void release(JedisPool jedisPool, Jedis jedis) {
    if (null != jedis) {
        jedisPool.returnResource(jedis);
```

```
    }
}
```