# 1.安装：

- **Ubunto:**

```
# 安装
sudo apt-get install redis-server
# 服务端启动
sudo /etc/init.d/redis-server status | start | stop | restart
# 客户端连接
redis-cli -h IP地址 -p 6379 -a 密码
```

- **Windows**

```
1、下载安装包
   https://github.com/ServiceStack/redis-
windows/blob/master/downloads/redis-64.3.0.503.zip
2、解压
3、启动服务端
   双击解压后的 redis-server.exe
4、客户端连接
   双击解压后的 redis-cli.exe

# Windows下产生的问题：关闭终端后服务终止
# 解决方案：将Redis服务安装到本地服务
1、重命名 redis.windows.conf 为 redis.conf,作为redis服务的配置文件
2、cmd命令行，进入到redis-server.exe所在目录
3、执行：redis-server --service-install redis.conf --loglevel verbose
4、计算机-管理-服务-Redis-启动

# 卸载
到 redis-server.exe 所在路径执行：
1、redis-server --service-uninstall
2、sc delete Redis
```

- **Mac**

```
# 安装
brew install redis
# 服务端启动
redis-server /usr/local/etc/redis.conf
brew services start redis  # 还未测试
# 客户端启动
redis-cli -h IP地址 -p 6379 -a 密码
# 路径
/usr/local/Cellar/redis/5.0.6
# 关闭
redis-cli shutdown
```

## 2.配置文件

### 2-1.配置文件的路径

```
1、Ubuntu
  /etc/redis/redis.conf
  mysql的配置文件在哪里?  : /etc/mysql/mysql.conf.d/mysqld.cnf

2、windows 下载解压后的redis文件夹中
  redis.windows.conf
  redis.conf

3、Mac
  /usr/local/etc/redis.conf
```

### 2-2.设置连接密码

```
1、requirepass 密码
2、重启服务
   sudo /etc/init.d/redis-server restart
3、客户端连接
   redis-cli -h 127.0.0.1 -p 6379 -a 123456
   127.0.0.1:6379>ping
```

### 2-3.允许远程访问

```
1、注释掉本地IP地址绑定
69行: # bind 127.0.0.1 ::1
2、关闭保护模式(把yes改为no)
88行: protected-mode no
3、重启服务
sudo /etc/init.d/redis-server restart
```

### 2-4.常用配置总结

```
# 设置密码
1、requirepass password
# 开启远程连接
2、bind 127.0.0.1 ::1 注释掉
3、protected-mode no 把默认的 yes 改为 no
# rdb持久化-默认配置
4、dbfilename 'dump.rdb'
5、dir /var/lib/redis
# rdb持久化-自动触发(条件)
6、save 900 1
7、save 300 10
8、save 60 10000
# aof持久化开启
9、appendonly yes
10、appendfilename 'appendonly.aof'
# aof持久化策略
11、appendfsync always
12、appendfsync everysec # 默认
13、appendfsync no
# aof重写触发
14、auto-aof-rewrite-percentage 100
15、auto-aof-rewrite-min-size 64mb
# 设置为从服务器
16、salveof <master-ip> <master-port>
```

2-5.redis下相关文件存放路径

```
1、配置文件: /etc/redis/redis.conf
2、备份文件: /var/lib/redis/*.rdb|*.aof
3、日志文件: /var/log/redis/redis-server.log
4、启动文件: /etc/init.d/redis-server
# /etc/下存放配置文件
# /etc/init.d/下存放服务启动文件
```

# 3.数据类型--基本操作

## 3-1.通用命令

```
# 适用于所有数据类型
# 切换库(number的值在0-15之间,db0 ~ db15)
select number
# 查看键
keys 表达式
# 数据类型
TYPE key
# 键是否存在
exists key
# 删除键
del key
# 键重命名
rename key newkey
# 清除当前库中所有数据(慎用)
flushdb
# 清除所有库中所有数据(慎用)
flushall
```

## 3-2.字符串类型（string）

特点

```
1、字符串、数字，都会转为字符串来存储
2、以二进制的方式存储在内存中
3、一个字符串类型的值最多能存储512M字节的内容，位上限:2^32
```

常用命令

```
# 1. 设置一个key-value
set key value
# 2. 获取key的值
get key
# 3. key不存在时再进行设置(nx)
set key value nx # not exists
# 4. 设置过期时间(ex)
set key value ex seconds
# 5. 同时设置多个key-value
mset key1 value1 key2 value2 key3 value3
# 6. 同时获取多个key-value
mget key1 key2 key3
```

不常用命令

```
# 1.获取长度
strlen key
# 2.获取指定范围切片内容
getrange key start stop
# 3.从索引值开始，value替换原内容
setrange key index value
# 4.追加拼接value的值
append key value
# 设置过期时间的两种方式
# 方式一
1、set key value ex 3
# 方式二
1、set key value
2、expire key 5 # 秒
3、pexpire key 5 # 毫秒
# 查看存活时间
ttl key
# 删除过期
persist key
```

数值操作(必须掌握)

```
# 整数操作
INCRBY key 步长
DECRBY key 步长
INCR key : +1操作
DECR key : -1操作
# 应用场景：抖音上有人关注你了，是不是可以用INCR呢，如果取消关注了是不是可以用DECR # 浮点数操作：自动先转为数字类型，然后再进行相加减，不能使用append
incrbyfloat key step
```

string类型注意

```
# key值取值原则
1、key值不宜过长，消耗内存，且在数据中查找这类键值的计算成本高
2、不宜过短，可读性较差
```

### 3-3.列表类型（list）

特点

```
1、元素是字符串类型
2、列表头尾增删快，中间增删慢，增删元素是常态
3、元素可重复
4、最多可包含2^32 -1个元素
5、索引同python列表
```

基本命令

```
# 增
# 1. 从列表头部压入元素
LPUSH key value1 value2
# 2. 从列表尾部压入元素
RPUSH key value1 value2
# 3. 从列表src尾部弹出1个元素,压入到列表dst的头部
RPOPLPUSH src dst
# 4. 在列表指定元素后/前插入元素
LINSERT key after|before value newvalue

# 查
# 1. 查看列表中所有元素:
LRANGE key 0 -1 6
# 2. 获取列表长度
LLEN key
```

```
# 删
# 1. 从列表头部弹出1个元素
LPOP key
# 2. 从列表尾部弹出1个元素
RPOP key
# 3. 列表头部,阻塞弹出,列表为空时阻塞
BLPOP key timeout
# 4. 列表尾部,阻塞弹出,列表为空时阻塞
BRPOP key timeout
  # 关于BLPOP 和 BRPOP
  1、如果弹出的列表不存在或者为空，就会阻塞
  2、超时时间设置为0，就是永久阻塞，直到有数据可以弹出
  3、如果多个客户端阻塞再同一个列表上，使用First In First Service原则，先到先服务
# 5. 删除指定元素
LREM key count value # count>0:表示从头部开始向表尾搜索，移除与value相等的元素，数量为count count<0:表示从尾部开始向表头搜索，移除与value相等的元素，数量为count count=0:移除表中所有与value相等的值
# 6. 保留指定范围内的元素
LTRIM key start stop
LTRIM mylist1 0 2 # 只保留前3条 # 应用场景: 保存微博评论最后500条 LTRIM weibo:comments 0 499

# 改
# 1. 修改某个元素的值
LSET key index newvalue
```

### 3-4.哈希类型（hash）

- 定义

```
1、由field和关联的value组成的键值对
2、field和value是字符串类型
3、一个hash中最多包含 2^32-1 个键值对
```

- 优点

```
1、节约内存空间
2、每创建一个键，它都会为这个键储存一些附加的管理信息(比如这个键的类型，这个键最后一次被访问的时间等)
3、键越多，redis数据库在储存附件管理信息方面耗费内存越多，花在管理数据库键上的CPU也会越多
```

- 缺点（不适合hash的情况）

```
1、使用二进制位操作命令:SETBIT、GETBIT、BITCOUNT等，如果想使用这些操作，只能用字符串
2、使用过期键功能:键过期功能只能对键进行过期操作，而不能对散列的字段进行过期操作
```

- 常用命令

```
# 1、设置单个字段
HSET key field value
HSETNX key field value
# 2、设置多个字段
HMSET key field value field value
# 3、返回字段个数
HLEN key
# 4、判断字段是否存在(不存在返回0)
HEXISTS key field
# 5、返回字段值
HGET key field
# 6、返回多个字段值
HMGET key field filed
# 7、返回所有的键值对
HGETALL key
# 8、返回所有字段名
HKEYS key
# 9、返回所有值
HVALS key
# 10、删除指定字段
HDEL key field
# 11、在字段对应值上进行整数增量运算
HINCRBY key filed increment
# 12、在字段对应值上进行浮点数增量运算
HINCRBYFLOAT key field increment
```

### 3-5.集合类型（set）

特点

```
1、无序、去重
2、元素是字符串类型
3、最多包含2^32-1个元素
```

基本命令

```
# 1、增加一个或者多个元素,自动去重
SADD key member1 member2
# 2、查看集合中所有元素
SMEMBERS key
# 3、删除一个或者多个元素，元素不存在自动忽略
SREM key member1 member2
# 4、元素是否存在
SISMEMBER key member
# 5、随机返回集合中指定个数的元素，默认为1个
SRANDMEMBER key [count]
# 6、弹出成员
SPOP key [count]
# 7、返回集合中元素的个数，不会遍历整个集合，只是存储在键当中了
SCARD key
# 8、把元素从源集合移动到目标集合
SMOVE source destination member
# 9、差集(number1 1 2 3 number2 1 2 4 结果为3)
SDIFF key1 key2
# 10、差集保存到另一个集合中
SDIFFSTORE destination key1 key2
# 11、交集
SINTER key1 key2
SINTERSTORE destination key1 key2
# 11、并集
SUNION key1 key2
SUNIONSTORE destination key1 key2
```

### 3-6.有序集合类型（sorted set）

特点

```
1、有序、去重
2、元素是字符串类型
3、每个元素都关联着一个浮点数分值(score)，并按照分值从小到大的顺序排列集合中的元素(分值可以相同)
4、最多包含2^32-1元素
```

常用命令

```
# 1. 在有序集合中添加一个成员
zadd key score member
# 2. 查看指定区间元素(升序)
```

```
zrange key start stop [withscores]
# 3. 查看指定区间元素(降序)
ZREVRANGE key start stop [withscores]
# 4. 查看指定元素的分值
ZSCORE key member
# 5. 返回指定区间元素  offset : 跳过多少个元素  count : 返回几个
# 小括号 : 开区间, 不包含 , 不加小括号就包含
zrangebyscore key min max [withscores] [limit offset count]
# 每页显示10个成员,显示第5页的成员信息:
# limit 40 10
# MySQL: 每页显示10条记录,显示第5页的记录
# limit 40,10
# limit 2,3 显示: 第3 4 5条记录
# 6. 删除成员
zrem key member
# 增加或者减少分值
zincrby key increment member
# 返回元素排名
zrank key member
# 返回元素逆序排名
zrevrank key member
# 删除指定区间内的元素
zremrangebyscore key min max
# 返回集合中元素个数
zcard key
# 返回指定范围中元素的个数
zcount key min max
zcount salary 6000 8000
zcount salary (6000 8000# 6000<salary<=8000
zcount salary (6000 (8000#6000<salary<8000
# 并集
zunionstore destination numkeys key [weights 权重值] [AGGREGATE SUM|MIN|MAX]
# zunionstore salary3 2 salary salary2 weights 1 0.5 AGGREGATE MAX
# 2代表集合数量,weights之后 权重1给salary,权重0.5给salary2集合,算完权重之后执行聚合
AGGREGATE
# 交集:和并集类似, 只取相同的元素
ZINTERSTORE destination numkeys key1 key2 WEIGHTS weight AGGREGATE SUM(默
认)|MIN|MAX
```

### 3-7.位图操作（bitmap）

定义

```
1、位图不是真正的数据类型，它是定义在字符串类型中
2、一个字符串类型的值最多能存储512M字节的内容，位上限:2^32
# 1MB = 1024KB
# 1KB = 1024Byte(字节)
# 1Byte = 8bit(位)
```

强势点

```
可以实时的进行统计，极其节省空间。官方在模拟1亿2千8百万用户的模拟环境下，在一台MacBookPro上，典型的统计如“日用户数”的时间消耗小于50ms，占用16MB内存
```

基本操作

```
# 设置某一位上的值(offset是偏移量，从0开始)
setbit key offset value
# 获取某一位上的值
getbit key offset
# 统计键所对应的值中有多少个 1
bitcount key
```

# 4.持久化

### 4-1.数据持久化之 – RDB模式(默认开启)

方式一：使用 SAVE 或者 BGSAVE 命令

```
127.0.0.1:6379> SAVE
OK
# 特点 1、执行SAVE命令过程中，redis服务器将被阻塞，无法处理客户端发送的命令请求，在SAVE命令执 行完毕后，服务器才会重新开始处理客户端发送的命令请求 2、如果RDB文件已经存在，那么服务器将自动使用新的RDB文件代替旧的RDB文件

127.0.0.1:6379> BGSAVE
Background saving started # 执行过程如下
1、客户端 发送 BGSAVE 给服务器
2、服务器马上返回 Background saving started 给客户端
3、服务器 fork() 子进程做这件事情
4、服务器继续提供服务 5、子进程创建完RDB文件后再告知Redis服务器

# 配置文件相关操作
/etc/redis/redis.conf
263行: dir /var/lib/redis # 表示rdb文件存放路径
```

```
253行: dbfilename dump.rdb # 文件名
# 两个命令比较 SAVE比BGSAVE快，因为需要创建子进程，消耗额外的内存
# 补充:可以通过查看日志文件来查看redis都做了哪些操作
# 日志文件:配置文件中搜索 logfile
logfile /var/log/redis/redis-server.log
```

方式二：设置配置文件(使用最多)

```
# 命令行示例
redis>save 300 10
#表示如果距离上一次创建RDB文件已经过去了300秒，并且服务器的所有数据库总共已经发生了不少于10次修改，那么自动执行BGSAVE命令
redis>save 60 10000
#表示如果距离上一次创建rdb文件已经过去60秒，并且服务器所有数据库总共已经发生了不少于10000次修改，那么执行bgsave命令

# redis配置文件默认
218行: save 900 1
219行: save 300 10
220行: save 60 10000
1、只要三个条件中的任意一个被满足时，服务器就会自动执行BGSAVE
2、每次创建RDB文件之后，服务器为实现自动持久化而设置的时间计数器和次数计数器就会被清零，并重新开始计数，所以多个保存条件的效果不会叠加
```

### 4-2.数据持久化之 - AOF(AppendOnlyFile，默认未开启)

特点

```
1、存储的是命令，而不是真实数据
2、默认不开启
```

开启方式

```
# 修改配置文件
1、/etc/redis/redis.conf
672行: appendonly yes # 把 no 改为 yes
676行: appendfilename "appendonly.aof"
2、重启服务
sudo /etc/init.d/redis-server restart
```

### 4-3.RDB的缺点

1、创建RDB文件需要将服务器所有的数据库的数据都保存起来，这是一个非常消耗资源和时间的操作，所以服务器需要隔一段时间才创建一个新的RDB文件，也就是说，创建RDB文件不能执行的过于频繁，否则会严重影响服务器的性能
2、可能丢失数据

### 4-4.AOF的原理以及优点

```
# 原理
1、每当有修改数据库的命令被执行时，服务器就会将执行的命令写入到AOF文件的末尾
2、因为AOF文件里面存储了服务器执行过的所有数据库修改的命令，所以给定一个AOF文件，服务器只要重新执行一遍AOF文件里面包含的所有命令，就可以达到还原数据库的目的
# 优点
用户可以根据自己的需要对AOF持久化进行调整，让Redis在遭遇意外停机时不丢失任何数据，或者只丢失一秒钟的数据，这比RDB持久化丢失的数据要少的多
```

### 4-5.安全问题

```
# 因为
虽然服务器执行一个修改数据库的命令，就会把执行的命令写入到AOF文件，但这并不意味着AOF文件持久化不会丢失任何数据，在目前常见的操作系统中，执行系统调用write函数，将一些内容写入到某个文件里面时，为了提高效率，系统通常不会直接将内容写入硬盘里面，而是将内容放入一个内存缓存区(buffer)里面，等到缓冲区被填满时才将存储在缓冲区里面的内容真正写入到硬盘里
# 所以
1、AOF持久化:当一条命令真正的被写入到硬盘里面时，这条命令才不会因为停机而意外丢失
2、AOF持久化在遭遇停机时丢失命令的数量，取决于命令被写入到硬盘的时间
3、越早将命令写入到硬盘，发生意外停机时丢失的数据就越少，反之亦然
```

### 4-6.策略-配置文件

```
# 打开配置文件:/etc/redis/redis.conf，找到相关策略如下
1、701行: alwarys
服务器每写入一条命令，就将缓冲区里面的命令写入到硬盘里面，服务器就算意外停机，也不会丢失任何已经成功执行的命令数据
2、702行: everysec(# 默认)
服务器每一秒将缓冲区里面的命令写入到硬盘里面，这种模式下，服务器即使遭遇意外停机，最多只丢失1秒的数据
3、703行: no
服务器不主动将命令写入硬盘,由操作系统决定何时将缓冲区里面的命令写入到硬盘里面，丢失命令数量不确定
# 运行速度比较
always:速度慢 everysec和no都很快，默认值为everysec
```

### 4-7.AOF文件中是否会产生很多的冗余命令

> 为了让AOF文件的大小控制在合理范围，避免胡乱增长，redis提供了AOF重写功能，通过这个功能，服务器可以产生一个新的AOF文件
> -- 新的AOF文件记录的数据库数据和原由的AOF文件记录的数据库数据完全一样
> -- 新的AOF文件会使用尽可能少的命令来记录数据库数据，因此新的AOF文件通常会小很多
> -- AOF重写期间，服务器不会被阻塞，可以正常处理客户端发送的命令请求

示例：

| 原有的AOF文件 | 重写后的AOF文件 |
| --- | --- |
| select 0 | SELECT 0 |
| sadd myset peiqi | SADD myset peiqi qiaozhi danni |
| sadd myset qiaozhi | SET message 'hello tarena' |
| sadd myset danni | RPUSH mylist 2 3 5 |
| INCR number | |
| DEL number | |
| SET message 'hello world' | |
| SET message 'hello tarena' | |
| RPUSH mylist 1 2 3 | |
| RPUSH mylist 5 | |
| LPOP mylist | |

### 4-8.AOF文件重写触发的方法

```
1、客户端向服务器发送BGREWRITEAOF命令
127.0.0.1:6379> BGREWRITEAOF
Background append only file rewriting started
2、修改配置文件让服务器自动执行BGREWRITEAOF命令
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
# 解释
1、只有当AOF文件的增量大于100%时才进行重写，也就是大一倍的时候才触发
# 第一次重写新增:64M
# 第二次重写新增:128M
# 第三次重写新增:256M(新增128M)
```

### 4-9.RDB和AOF持久化对比

| RDB | AOF |
| --- | --- |
| 全量备份，一次保存整个数据库 | 增量备份，一次保存一个修改数据库的命令 |
| 保存的间隔较长 | 保存的间隔默认为一秒钟 |
| 数据还原速度快 | 数据还原速度一般，冗余命令多，还原速度慢 |
| 执行SAVE命令时会阻塞服务器，但手动或者自动触发的BGSAVE不会阻塞服务器 | 无论是平时还是进行AOF重写时，都不会阻塞服务器 |
| 更适合数据备份 | 更适合用来保存数据，通常意义上的数据持久化，在appendfsync always模式下运行 |

> # 用redis用来存储真正数据，每一条都不能丢失，都要用always，有的做缓存，有的保存真数据，我可以开多个redis服务，不同业务使用不同的持久化，新浪每个服务器上有4个redis服务，整个业务中有上千个redis服务，分不同的业务，每个持久化的级别都是不一样的。

### 4-10.数据恢复（无需手动操作）

> 既有dump.rdb，又有appendonly.aof，恢复时找谁?
> 先找appendonly.aof

## 5.主从复制

> 从服务器只能读取，不能写入

### 5-1.实现方式

- 实现方式一

```
# Linux命令行实现
redis-server --port 6300 --slaveof 127.0.0.1 6379
```

- 实现方式二

```
# Redis命令行实现
1. 启动 Redis 服务，端口为 6301
redis-server --port 6301
2. 客户端登录 6301 端口的 Redis 服务
redis-cli -h 127.0.0.1 -p 6301
3. 指定该服务为 127.0.0.1 6379 的从服务器
slaveof 127.0.0.1 6379  # 此时，2 个 Redis 数据库都不能修改
4. 将从服务器切换为主服务器
slaveof on one # 此时，主服务器可以修改数据了,此时如果关掉了6301的主服务器，6379的服务器任然不能修改数据
```

- 实现方式三

```
# 配置文件实现
# 每个redis服务,都有1个和他对应的配置文件
# 两个redis服务
1、6379 -> /etc/redis/redis.conf
2、6300 -> /home/tarena/redis_6300.conf
# 修改配置文件
vi redis_6300.conf
slaveof 127.0.0.1 6379
port 6300
# 启动redis服务
redis-server redis_6300.conf
```

### 5-2.Master挂了怎么办

```
1、一个Master可以有多个Slaves
2、Slave下线，只是读请求的处理性能下降
3、Master下线，写请求无法执行
4、其中一台Slave使用SLAVEOF no one命令成为Master，其他Slaves执行SLAVEOF命令指向这个新的Master，从它这里同步数据
# 以上过程是手动的，如果要实现自动，这就需要Sentinel哨兵，实现故障转移Failover操作
```

## 6.官方高可用方案Sentinel

### 6-1.Redis之哨兵 - sentinel

```
1、Sentinel会不断检查Master和Slaves是否正常
2、每一个Sentinel可以监控任意多个Master和该Master下的Slaves
```

### 6-2.环境搭建

```
# 搭建主从服务器
# 共3台redis的服务器，如果是不同机器端口号可以是一样的
1、启动6379的redis服务器
sudo /etc/init.d/redis-server start
2、启动6380的redis服务器，设置为6379的从
redis-server --port 6380 --slaveof 127.0.0.1 6379
3、启动6381的redis服务器，设置为6379的从
redis-server --port 6381 --slaveof 127.0.0.1 6379

# 搭建 sentinel 服务
# 1、安装redis-sentinel
sudo apt install redis-sentinel
验证: sudo /etc/init.d/redis-sentinel stop
# 2、新建配置文件sentinel.conf
port 26379
Sentinel monitor tedu 127.0.0.1 6379 1
# 3、启动sentinel
方式一: redis-sentinel sentinel.conf
方式二: redis-server sentinel.conf --sentinel
#4、将master的redis服务终止，查看从是否会提升为主 sudo /etc/init.d/redis-server
stop
# 发现提升6381为master，其他两个为从
# 在6381上设置新值，6380查看
127.0.0.1:6381> set name tedu OK
# 启动6379，观察日志，发现变为了6381的从

# 主从+哨兵基本就够用了
```

### 6-3.sentinel.conf

```
# sentinel监听端口，默认是26379，可以修改
port 26379
# 告诉sentinel去监听地址为ip:port的一个master，这里的master-name可以自定义，quorum
是一个数字，指明当有多少个sentinel认为一个master失效时，master才算真正失效
sentinel monitor <master-name> <ip> <redis-port> <quorum>
```

## 7.分布式锁

### 7-1.高并发产生的问题

```
1、购票： 多个用户抢到同一张票?
2、购物： 库存只剩1个,被多个用户成功买到?
... ...
```

### 7-2.怎么办

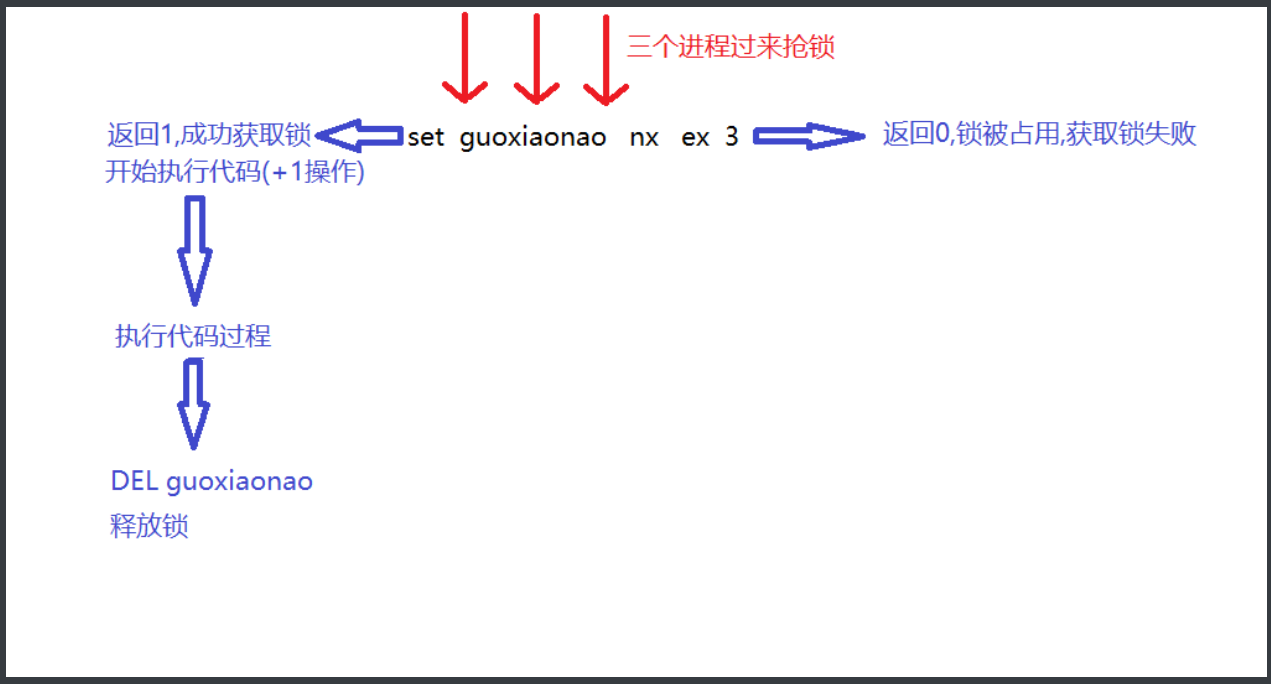
```
在不同进程需要互斥地访问共享资源时，分布式锁是一种非常有用的技术手段
```

### 7-3.原理

```
1、多个客户端先到redis数据库中获取一把锁,得到锁的用户才可以操作数据库
2、此用户操作完成后释放锁，下一个成功获取锁的用户再继续操作数据库
```

### 7-4.实现

```
# 1、从redis2.8开始，set命令集成了两个参数，nx和ex，先拿nx来争抢锁，抢到之后，再用ex参数给锁加一个过期时间防止锁无法释放，造成死锁
set key value nx ex 3
```




### 7-5.代码实现（python）

```
# 启多个服务端，模拟30个并发请求
# 1. 多台服务器启动项目
python3 manage.py runserver 127.0.0.1:8000
python3 manage.py runserver 127.0.0.1:8001
```

```
# 2. 在tools中新建py文件 test_api.py，模拟30个并发请求
import threading
import requests
import random
def getRequest():
  url='http://127.0.0.1:8000/test'
  url2='http://127.0.0.1:8001/test'
  get_url = random.choice([url, url2]) requests.get(get_url)

ts = []
for i in range(30):
  t=threading.Thread(target=getRequest,args=())
  ts.append(t)

if __name__ == '__main__':
  for t in ts:
    t.start()
  for t in ts:
    t.join()

# 在数据库中查看 score 字段的值
# 并没有+30，而且没有规律，每次加的次数都不同

# 解决方案:redis分布式锁
import redis
def test(request):
  pool = redis.ConnectionPool(host='localhost', port=6379, db=0)
  r =  redis.Redis(connection_pool=pool)
  while True:
    try:
      with r.lock('guoxiaonao', blocking_timeout=3) as lock:
        u = UserProfile.objects.get(username='guoxiaonao') u.score += 1
        u.save()
    except Exception as e:
      print('lock is failed')

  return HttpResponse('HI HI HI')
```

# 8.事务

## 8-1.特点

1. 单独的隔离操作:事务中的所有命令会被序列化、按顺序执行，在执行的过程中不会被其他客户端发送来的命令打断
2. 不保证原子性:redis中的一个事务中如果存在命令执行失败，那么其他命令依然会被执行，没有回滚机制

### 8-2.命令

```
# 开启事务
127.0.0.1:6379> MULTI
OK
# 命令1入队列
127.0.0.1:6379> INCR n1
QUEUED
# 命令2入队列
127.0.0.1:6379> INCR n2
QUEUED
# 提交到数据库执行
127.0.0.1:6379> EXEC
1) (integer) 1
2) (integer) 1


DISCARD # 取消事务
```

### 8-3.事务中命令错误处理

```
# 1、命令语法错误，命令入队失败，直接自动discard退出这个事务
这个在命令在执行调用之前会发生错误。例如，这个命令可能有语法错误(错误的参数数量，错误 的命令名)
处理方案:客户端发生了第一个错误情况，在exec执行之前发生的。通过检查队列命令返回值:如 果这个命令回答这个队列的命令是正确的，否者redis会返回一个错误。如果那里发生了一个队列命令错误，大部分客户端将会退出并丢弃这个事务
# 2、命令语法没错，但类型操作有误，则事务执行调用之后失败，无法进行事务回滚 从我们施行了一个由于错误的value的key操作(例如对着String类型的value施行了List命令操作) 处理方案:发生在EXEC之后的是没有特殊方式去处理的:即使某些命令在事务中失败，所有的其他命令都将会被执行。
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set num 10
QUEUED
127.0.0.1:6379> LPOP num
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> get num
"10"
```

# 9.redis的优点

```
1、读写速度快. 数据存放在内存中
2、支持数据类型丰富,string,hash,list,set,sorted
3、支持事务
4、可以用于缓存,消息队列,按key设置过期时间,到期后自动删除
5、支持数据持久化(将内存数据持久化到磁盘),支持AOF和RDB两种持久化方式,从而进行数据恢复操作,可以有效地防止数据丢失
6、支持主从(master-slave)复制来实现数据备份,主机会自动将数据同步到从机
```

# 10.缓存

## 10-1.缓存穿透

```
# 原理
缓存和数据库都没有的数据，而用户反复发起请求， 如 假的用户ID
# 场景 比如发起为id为“-1”的数据或id为特别大不存在的数据。这时的用户很可能是攻击者，攻击会导致数 据库压力过大
# 解决方案：
1、请求校验，接口层增加校验，如对id做基础校验，id<=0的直接拦截
2、都无法取到数据时也可以将key-value对写为key-null，缓存有效时间比如30秒左右，这样可以防止攻击用户反复用同一个id暴力攻击
```

## 10-2.缓存击穿

```
# 原理
缓存没有，数据库有，一般是缓存时间到期， 顺势并发太大
#解决方案
1、热点数据不过期
2、上锁： 重新设计缓存的使用方式，当我们通过key去查询数据时，首先查询缓存，如果没有， 就通过分布式锁进行加锁，取得锁的进程查DB并设置缓存，然后解锁;其他进程如果发现有锁就 等待，然后等解锁后返回缓存数据或者再次查询DB
```

## 10-3.缓存雪崩

```
# 原理 缓存中大批量数据过期，导致瞬时大批量不同请求注入DB
# 解决方案
解决方案
1、缓存设置随机时间(避免缓存设置相近的有效期;为有效期增加随机值)
2、热点数据不过期
```