Redis 第 1 节课笔记

一、Redis 基础入门

1，redis 介绍

redis 是一种基于键值对（key-value）数据库，其中 value 可以为 string、hash、list、set、zset 等多种数据结构，可以满足很多应用场景。还提供了键过期，发布订阅，事务，流水线，等附加功能,

流水线: Redis 的流水线功能允许客户端一次将多个命令请求发送给服务器， 并将被执行的多个命令请求的结果在一个命令回复中全部返回给客户端，使用这个功能可以有效地减少客户端在执行多个命令时需要与服务器进行通信的次数。

2，特性：

1〉速度快，数据放在内存中，官方给出的读写性能 10 万/S，与机器性能也有关

a,数据放内存中是速度快的主要原因

b,C 语言实现，与操作系统距离近

c,使用了单线程架构，预防多线程可能产生的竞争问题

2〉键值对的数据结构服务器

3〉丰富的功能：见上功能

4〉简单稳定：单线程

5〉持久化：发生断电或机器故障，数据可能会丢失，持久化到硬盘

6〉主从复制：实现多个相同数据的 redis 副本

8〉高可用和分布式：哨兵机制实现高可用，保证 redis 节点故障发现和自动转移

9〉客户端语言多：java php python c c++ nodejs 等

3，使用场景：

1，缓存：合理使用缓存加快数据访问速度，降低后端数据源压力

2，排行榜：按照热度排名，按照发布时间排行，主要用到列表和有序集合

3，计数器应用：视频网站播放数，网站浏览数，使用 redis 计数

4，社交网络：赞、踩、粉丝、下拉刷新

5，消息队列：发布和订阅

4，正确安装与启动

1，linux 上安装，windows 也能装，但我们以 linux 环境为主

————————————————————————————————

2，配置、启动、操作、关闭

可执行文件	作用
redis-server	启动 redis
redis-cli	redis 命令行客户端
redis-benchmark	基准测试工具

redis-check-aof	AOF 持久化文件检测和修复工具
redis-check-dump	RDB 持久化文件检测和修复工具
redis-sentinel	启动哨兵

3，redis-server 启动：

1，默认配置：redis-server，日志输出版本信息，端口 6379

2，运行启动：redis-server --port 6380　不建议

3，配置文件启动：　redis-server /opt/redis/redis.conf，灵活，生产环境使用这种

4,redis-cli 启动

1，>交互式：redis-cli -h {host} -p {prot}连接到 redis 服务，没有 h 默认连 127.0

redis-cli -h 127.0.0.1 -p 6379　　//没有 p 默认连 6379

2，>命令式：redis-cli -h 127.0.0.1 -p 6379 get hello　//取 key=hello 的 value

3，>停止 redis 服务：　redis-cli shutdown

注意: a，关闭时：断开连接，持久化文件生成，相对安全

b，还可以用 kill 关闭，此方式不会做持久化，还会造成缓冲区非法关闭，可能会造成 AOF 和丢失数据

c，关闭前生成持久化文件：

使用 redis-cli -a 123456 登录进去，再 shutdown nosave|save

4，>重大版本：

1，版本号第二位为奇数，为非稳定版本（2.7、2.9、3.1）

2，第二为偶数，为稳定版本（2.6、2.8、3.0）

3，当前奇数版本是下一个稳定版本的开发版本，如 2.9 是 3.0 的开发版本

二、重要的指令使用:

1>全局命令

1，查看所有键：keys *　set school enjoy　set hello world

2，键总数 dbsize　//2 个键，如果存在大量键，线上禁止使用此指令

3，检查键是否存在：exists key　//存在返回 1，不存在返回 0

4，删除键：del key　//del hello school，返回删除键个数，删除不存在键返回 0

5，键过期：expire key seconds　//set name test　expire name 10 //10 秒过期

ttl 查看剩余的过期时间

6，键的数据结构类型：type key //type hello　//返回 string,键不存在返回 none

2>单线程架构

列举例子：三个客户端同时执行命令

客户端 1：set name test

客户端 2：incr num

客户端 3：incr num

执行过程：发送指令一〉执行命令一〉返回结果

执行命令：单线程执行，所有命令进入队列，按顺序执行，使用 I/O 多路复用解决 I/O 问题，后面有介绍(通过 select/poll/epoll/kqueue 这些 I/O 多路复用函数库，我们

解决了一个线程处理多个连接的问题)

单线程快原因：纯内存访问， 非阻塞 I/O（使用多路复用），单线程避免线程切换和竞争产生资源消耗

问题：如果某个命令执行，会造成其它命令的阻塞

3>字符串<String>

3.1，字符串类型：实际上可以是字符串（包括 XML JSON），还有数字（整形 浮点数），二进制（图片 音频 视频），最大不能超过 512MB

3.2，设值命令：set age 23 ex 10 //10 秒后过期　px 10000 毫秒过期

setnx name test　//不存在键 name 时，返回 1 设置成功；存在的话失败 0

set age 25 xx　　//存在键 age 时，返回 1 成功

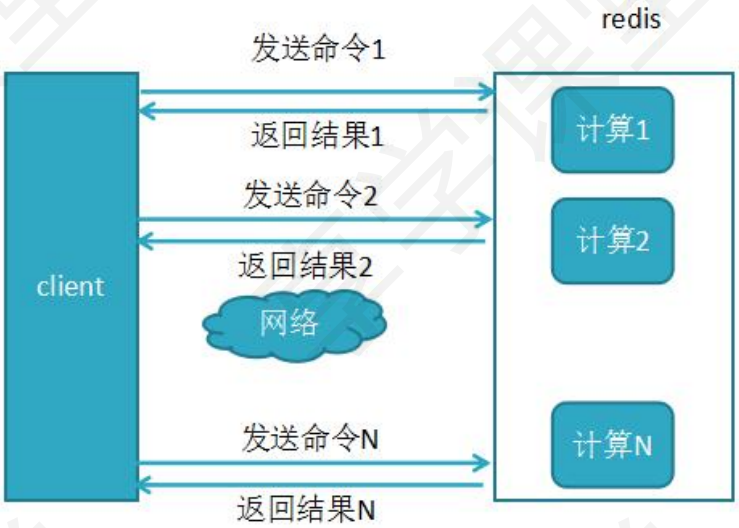
场景：如果有多客户同时执行 setnx,只有一个能设置成功，可做分布式锁

获值命令：get age //存在则返回 value，不存在返回 nil

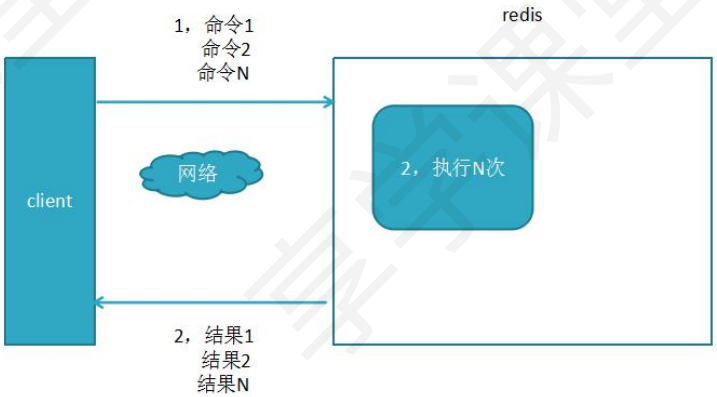
批量设值：mset country china city beijing

批量获取：mget country city address //返回 china　beigjin, address 为 nil

若没有 mget 命令，则要执行 n 次 get 命令



使用 mget＝1 次网络请求+redis 内部 n 次查询



3.3，计数：incr age //必须为整数自加 1，非整数返回错误，无 age 键从 0 自增返回 1
decr age //整数 age 减 1
incrby age 2 //整数 age+2
decrby age 2//整数 age -2
incrbyfloat score 1.1 //浮点型 score+1.1

3.4，append 追加指令：set name hello; append name world //追加后成 helloworld
3.5，字符串长度：set hello “世界”；strlen hello//结果 6，每个中文占 3 个字节
3.6，截取字符串：set name helloworld ; getrange name 2 4//返回 llo
3.7，内部编码：int:8 字节长整理//set age 100; object encoding age //返回 int
embstr:小于等于 39 字节串 set name bejin; object encodeing name//embstr
raw:大于 39 字节的字符串 set a fsdfwerwerwefffffffffffdfs//返回 raw

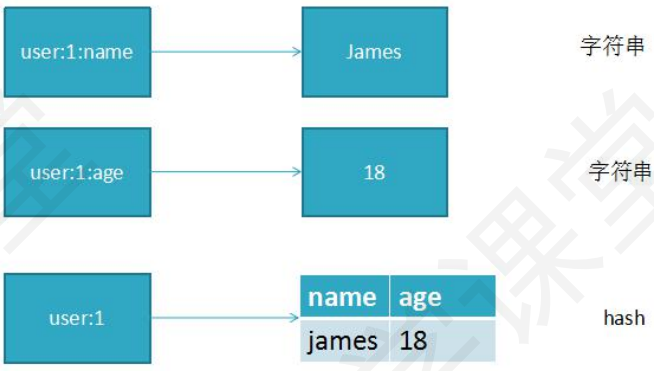
3.8，应用场景：
1，键值设计：业务名:对象名:id:[属性]
数据库为 order，用户表 user，对应的键可为 order:user:1 或 order:user:1:name
注意：redis 目前处于受保护模式，不允许非本地客户端链接，可以通过给 redis 设置密码，然后客户端链接的时候，写上密码就可以了
127.0.0.1:6379> config set requirepass 123456 临时生效

或者修改 redis.conf requirepass 123456,启动时./redis-server redis.conf 指定 conf
./redis-cli -p 6379 -a 12345678 //需要加入密码才能访问

切换数据库:select 2

4>哈希 hash：

是一个 string 类型的 field 和 value 的映射表，hash 特适合用于存储对象。



4.1 命令 hset key field value
设值：hset user:1 name james //成功返回 1，失败返回 0
取值：hget user:1 name //返回 james
删值：hdel user:1 age //返回删除的个数
计算个数：hset user:1 name james; hset user:1 age 23;

hlen user:1 //返回 2，user:1 有两个属性值

批量设值：hmset user:2 name james age 23 sex boy //返回 OK

批量取值：hmget user:2 name age sex //返回三行：james 23 boy

判断 field 是否存在：hexists user:2 name //若存在返回 1，不存在返回 0

获取所有 field: hkeys user:2 // 返回 name age sex 三个 field

获取 user:2 所有 value：hvals user:2 // 返回 james 23 boy

获取 user:2 所有 field 与 value：hgetall user:2 //name age sex james 23 boy 值

增加 1：hincrby user:2 age 1 //age+1

hincrbyfloat user:2 age 2 //浮点型加 2

4.2 内部编码：ziplist<压缩列表>和 hashtable<哈希表>

当 field 个数少且没有大的 value 时，内部编码为 ziplist

如：hmset user:3 name james age 24; object encoding user:3 //返回 ziplist

当 value 大于 64 字节，内部编码由 ziplist 变成 hashtable

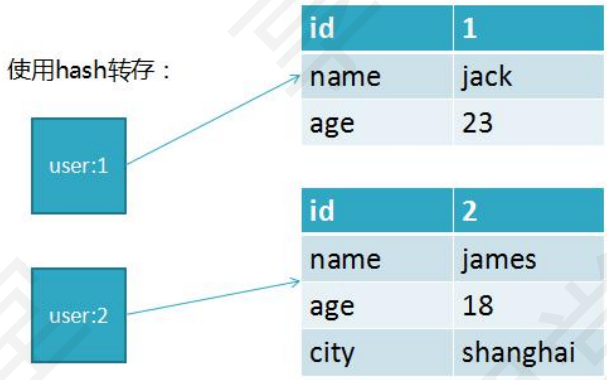
如：hset user:4 address “fsgst64 字节”; object encoding user:3 //返回 hashtable

4.3 应用场景：

比如将关系型数据表转成 redis 存储：

id	name	age	city
1	jack	23	NULL
2	james	18	shanghai

使用 hash 后的存储方式为：



如果有值为 NULL，那么如下：

HASH 类型是稀疏，每个键可以有不同的 filed，若用 redis 模拟做关系复杂查询开发因难，维护成本高

4.4 三种方案实现用户信息存储优缺点：

1，原生：set user:1:name james;

set user:1:age 23;

set user:1:sex boy;

优点：简单直观，每个键对应一个值

缺点：键数过多，占用内存多，用户信息过于分散，不用于生产环境

2，将对象序列化存入 redis

set user:1 serialize(userInfo);

优点：编程简单，若使用序列化合理内存使用率高

缺点：序列化与反序列化有一定开销，更新属性时需要把 userInfo 全取出来进行反序列化，更新后再序列化到 redis

3，使用 hash 类型：

hmset user:1 name james age 23 sex boy

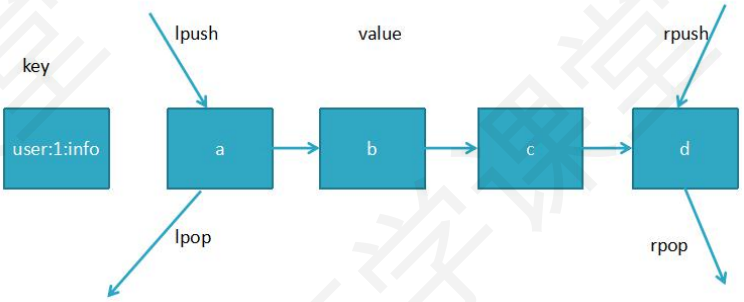
优点：简单直观，使用合理可减少内存空间消耗

缺点：要控制 ziplist 与 hashtable 两种编码转换，且 hashtable 会消耗更多内存

总结：对于更新不多的情况下，可以使用序列化，对于 VALUE 值不大于 64 字节可以使用 hash 类型

5>列表<list>

5.1 用来存储多个有序的字符串，一个列表最多可存 2 的 32 次方减 1 个元素



因为有序，可以通过索引下标获取元素或某个范围内元素列表，列表元素可以重复

5.2 列表命令：

类型	指令
添加	rpush lpush linsert
查	lrange lindex llen
修改	lset
删除	lpop rpop lrem ltrim
阻塞	blpop brpop

添加命令：rpush james c b a //从右向左插入 cba，返回值 3

lrange james 0 -1 //从左到右获取列表所有元素 返回 c b a

lpush key c b a //从左向右插入 cba

linsert james before b teacher //在 b 之前插入 teacher, after 为之后，使用 lrange james 0 -1 查看：c teacher b a

查找命令： lrange key start end //索引下标特点：从左到右为 0 到 N-1

lindex james -1 //返回最右末尾 a，-2 返回 b

llen james //返回当前列表长度

lpop james //把最左边的第一个元素 c 删除

rpop james //把最右边的元素 a 删除

lrem key count value//删除指定元素

如：lpush test b b b b b j x z //键 test 放入 z x j b b b b b

lrange test 0 -1 //查询结果为 z x j b b b b b

lrem test 4 b //从左右开始删除 b 的元素,删除 4 个，

若 lrem test 8 b，删除 8 个 b，但只有 5 个全部删除

lrange test 0 -1 //删除后的结果为 b j x z

lrem test 0 b //检索所有 b 全部删除 j x z

lpush user b b b b b j x z //键 user 从左到右放入 z x j b b b b b

ltrim user 1 3 //只保留从第 2 到第 4 的元素，其它全删

lrange user 0 -1 //查询结果为 x j b，其它已全被删掉

lpush user01 z y x //键 user01 从左到右放入 x y z

lset user01 2 java // 把第 3 个元素 z 替换成 java

lrange user01 0 -1 //查询结果为 x y java

应用场景设计： cacheListHashApplicationTest 用例

每个用户有多个订单 key 为 order:1 order:2 order:3，结合 hmset

1, hmset order:1 orderId 1 money 36.6 time 2018-01-01

hmset order:2 orderId 2 money 38.6 time 2018-01-01

hmset order:3 orderId 3 money 39.6 time 2018-01-01

把订单信息的 key 放到队列

lpush user:1:order order:1 order:2 order:3

每新产生一个订单，

hmset order:4 orderId 4 money 40.6 time 2018-01-01

追加一个 order:4 放入队列第一个位置

lpush user:1:order order:4

当需要查询用户订单记录时：

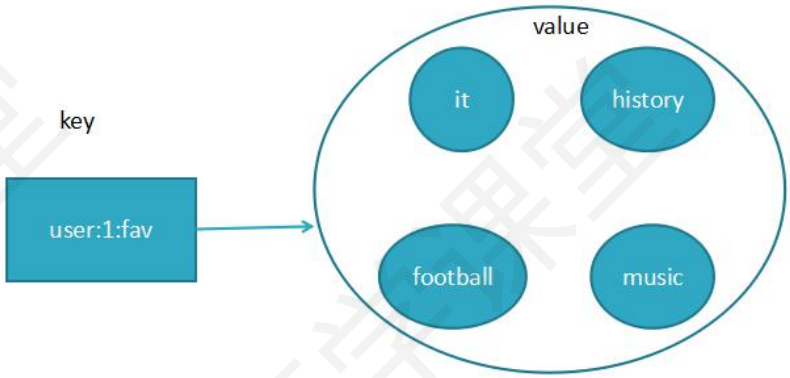
```
List orderKeys = lrange user:1 0 -1 //查询 user:1 的所有订单 key 值
for(Order order: orderKeys){
    hmget order:1
}
```

5.3 列表内部编码：

老师之前做的实验是在家里笔记本装的 redis,版本是 3.1，一直也没更换
经公司讲课的服务器上版本是 4.0，从 redis 的官网查阅了相关资料，在 3.2 版本以后，redis 提供了 quicklist 内部编码，它结合了 ziplist 和 linkedlist 两者的优势，之前的 ziplist 是存在 BUG 的，使用 quicklist 内部编码效率更高，所以我们现在 3.2 以后看不到这两个编码，只看到 quicklist，英语好的同学可以看一下 https://matt.sh/redis-quicklist 国外的这篇博客有重点提到感兴趣的同学可以下一下 3.1 之前的版本

6>集合<SET> 用户标签，社交，查询有共同兴趣爱好的人，智能推荐

保存多元素，与列表不一样的是不允许有重复元素，且集合是无序，一个集合最多可存 2 的 32 次方减 1 个元素，除了支持增删改查，还支持集合交集、并集、差集；



6.1 命令：

exists user　　//检查 user 键值是否存在
sadd user a b c//向 user 插入 3 个元素，返回 3
sadd user a b　//若再加入相同的元素，则重复无效，返回 0
smembers user //获取 user 的所有元素,返回结果无序

srem user a　　//返回 1，删除 a 元素

scard user　　//返回 2，计算元素个数

sismember user a //判断元素是否在集合存在，存在返回 1，不存在 0
srandmember user 2 //随机返回 2 个元素，2 为元素个数
spop user 2　　//随机返回 2 个元素 a b,并将 a b 从集合中删除
smembers user　//此时已没有 a b，只有 c

集合的交集：

sadd user:1 zhangsan 24 girl
sadd user:2 james 24 boy//初始化两个集合
sinter user:1 user:2　　//求两集合交集，　此时返回 24

sadd user:3 wang 24 girl //新增第三个元素
sinter user:1 user:2 user:3 //求三个集合的交集，此时返回 24

集合的并集（集合合并去重）：

sunion user:1 user:2 user:3 //三集合合并(并集)，去重 24
sdiff user:1 user:2//1 和 2 差集,(zhangsan 24 girl)-(james 24 boy)=zhangsan girl

将交集(jj)、并集(bj)、差集(cj)的结果保存：

sinterstore user_jj user:1 user:2 //将 user:1 user:2 的交集保存到 user_jj
sunionstore user_bj user:1 user:2 //将 user:1 user:2 的(并)合集保存 user_bj
sdiffstore user_cj user:1 user:2 //将 user:1-user:2 的差集保存 user_cj
smemebers user_cj // 返回 zhangsan girl

6.1 内部编码：

sadd user 1 2 3 4 //当元素个数少(小于 512 个)且都为整数，redis 使用 intset 减少内存的使用
sadd user 1 2...513 //当超过 512 个或不为整数(比如 a b)时，编码为 hashtable
object encoding user //hashtables

6.2 使用场景：

标签，社交，查询有共同兴趣爱好的人,智能推荐

使用方式：

给用户添加标签：

sadd user:1:fav basball fball pq
sadd user:2:fav basball fball
.....

或给标签添加用户

sadd basball:users user:1 user:3
sadd fball:users user:1 user:2 user:3
.....

计算出共同感兴趣的人：

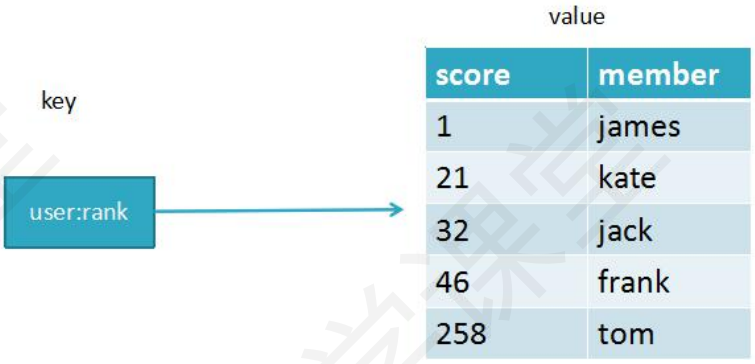
sinter user:1:fav user2:fav

规则：sadd (常用于标签) spop/srandmember(随机，比如抽奖)
sadd+sinter (用于社交，查询共同爱好的人，匹配)

7>有序集合:

常用于排行榜，如视频网站需要对用户上传视频做排行榜，或点赞数

与集合有联系，不能有重复的成员



与 LIST SET 对比

数据结构	是否允许元素重复	是否有序	有序实现方式	应用场景
列表	是	是	索引下标	时间轴，消息队列
集合	否	否	无	标签，社交
有序集合	否	是	分值	排行榜，点赞数

7.1 命令

zadd key score member [score member.....]

zadd user:zan 200 james //james 的点赞数 1，返回操作成功的条数 1

zadd user:zan 200 james 120 mike 100 lee// 返回 3

zadd test:1 nx 100 james //键 test:1 必须不存在，主用于添加

zadd test:1 xx incr 200 james //键 test:1 必须存在，主用于修改,此时为 300

zadd test:1 xx ch incr -299 james //返回操作结果 1，300-299=1

zrange test:1 0 -1 withscores //查看点赞（分数）与成员名

zcard test:1 //计算成员个数， 返回 1

查点赞数

zadd test:2 nx 100 james //新增一个集合

zscore test:2 james //查看 james 的点赞数（分数），返回 100

排名：

zadd user:3 200 james 120 mike 100 lee//先插入数据

zrange user:3 0 -1 withscores //查看分数与成员

lee	mike	james
100	120	200

zrank user:3 james //返回名次：第 3 名返回 2，从 0 开始到 2，共 3 名

zrevrank user:3 james //返回 0， 反排序，点赞数越高，排名越前

删除成员：

zrem user:3 jame mike //返回成功删除 2 个成员，还剩 lee

增加分数：

zincrby user:3 10 lee　　//成员 lee 的分数加 10

zadd user:3 xx incr 10 lee //和上面效果一样

返回指定排名范围的分数与成员

zadd user:4 200 james 120 mike 100 lee//先插入数据

zrange user:4 0 -1 withscores //返回结果如下图

```
1) "lee"
2) "100"
3) "mike"
4) "120"
5) "james"
6) "200"
```

zrevrange user:4 0 -1 withscores　　//倒序，结果如下图

```
1) "james"
2) "200"
3) "mike"
4) "120"
5) "lee"
6) "100"
```

返回指定分数范围的成员

zrangebyscore user:4 110 300 withscores //返回 120 lee ,200 James，由低到高

zrevrangebyscore user:4 300 110 withscores //返回 200james 120lee,由高到低

zrangebyscore user:4 (110 +inf withscores//110 到无限大，120mike 200james

zrevrangebyscore user:4 (110 -inf withscores//无限小到 110，返回 100 lee

返回指定分数范围的成员个数：

zcount user:4 110 300　//返回 2，由 mike120 和 james200 两条数据

删除指定排名内的升序元素：

zremrangebyrank user:4 0 1 //分数升序排列，删除第 0 个与第 1 个，只剩 james

删除指定分数范围的成员

zadd user:5 200 james 120 mike 100 lee//先插入测试数据

zremrangebyscore user:5 100 300　　//删除分数在 100 与 300 范围的成员

zremrangebyscore user:5 (100 +inf　　//删除分数大于 100(不包括 100),还剩 lee

有序集合交集：

格式：**zinterstore** destination numkeys key ... [WEIGHTS weight] [AGGREGATE SUM|MIN|MAX]

destination:交集产生新的元素存储键名称

numkeys:　要做交集计算的键个数
key :元素键值
weights:每个被选中的键对应值乘 weight，默认为 1

初始化数据：

zadd user:7 1 james 2 mike 4 jack 5 kate　　//初始化 user:7 数据
zadd user:8 3 james 4 mike 4 lucy 2 lee　6 jim　//初始化 user:8 数据

交集例子：

zinterstore user_jj 2 user:7 user:8 aggregate sum //2 代表键合并个数，
//aggregate sum 可加也不可加上，因为默认是 sum
//结果 user_jj：4james(1+3), 6mike(2+4)

zinterstore user_jjmax 2 user:7 user:8 aggregate max 或 min
//取交集最大的分数，返回结果 3james　4mike, min 取最小

weights:

zinterstore user_jjweight 2 user:7 user:8 weights 8 4 aggregate max
//1,取两个成员相同的交集，user:7->1 james　2 mike;　user:8->3 james　4 mike
//2,将 user:7->james 1*8=8,　user:7->mike 2*8 =16,
最后 user:7 结果 8 james　16 mike;
//3,将 user:8-> james 3*4=12, user:8->mike 4*4=16
最后 user:8 结果 12 james　16 mike
//4,最终相乘后的结果，取最大值为　12 james 16mike
//5, zrange user_jjweight 0 -1 withscores 查询结果为　12 james 16mike
总结：将 user:7 成员值乘 8，将 user:8 成员值乘 4，取交集，取最大

有序集合并集（合并去重）：

格式：**zunionstore** destination numkeys key ... [WEIGHTS weight] [AGGREGATE SUM|MIN|MAX]
destination:交集产生新的元素存储键名称
numkeys:　要做交集计算的键个数
key :元素键值
weights:每个被选中的键对应值乘 weight，默认为 1

zunionstore user_jjweight2 2 user:7 user:8 weights 8 4 aggregate max
//与以上 zinterstore 一样，只是取并集，指令一样

7.1 有序集合内部编码

1，ziplist:　zadd user:9 20 james 30 mike 40 lee
object encoding user:init
//当元素个数少（小于 128 个），元素值小于 64 字节时，使用 ziplist 编码，可有效减少内存的使用

2，skiplist:　zadd user:10 20 james.....
//大于 128 个元素或元素值大于 64 字节时为 skiplist 编码

7.2 使用场景：

排行榜系统，如视频网站需要对用户上传的视频做排行榜
点赞数：zadd user:1:20180106 3 mike　//mike 获得 3 个赞

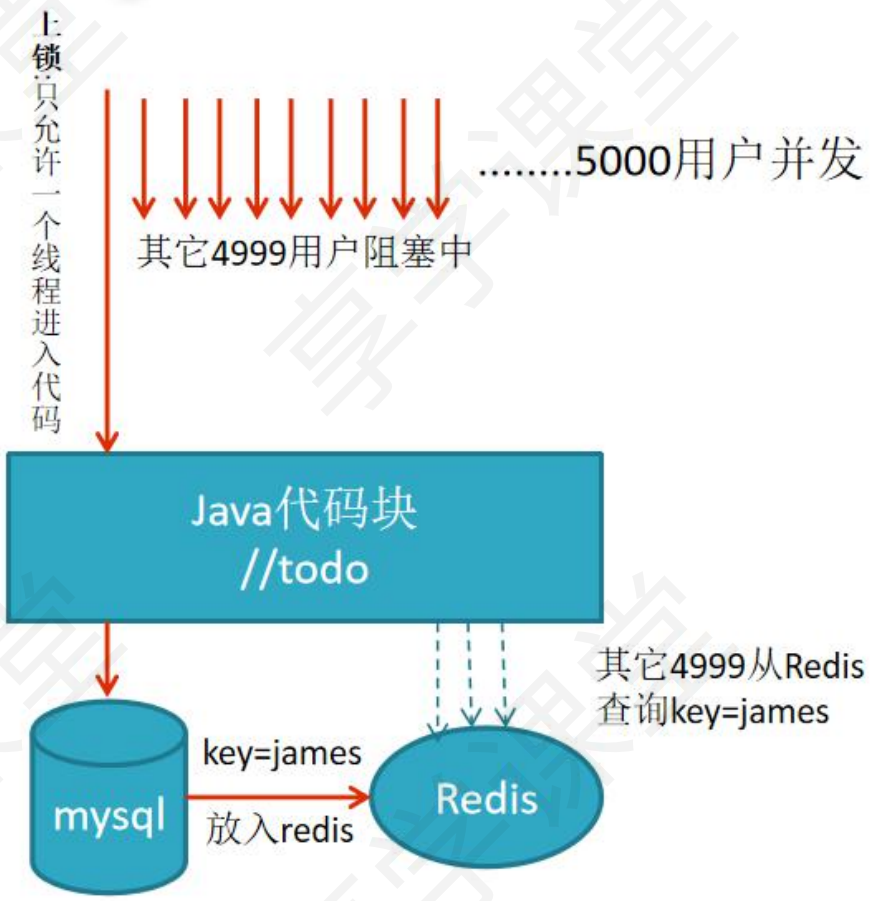
再获一赞：zincrby user:1:20180106 1 mike　//在 3 的基础上加 1
用户作弊，将用户从排行榜删掉：zrem user:1:20180106 mike
展示赞数最多的 5 个用户：
zadd user:4:20160101 9 jack 10 jj 11 dd 3 james 4 lee 6 mark 7 kate
zrevrangebylex user:4:20160101 + - limit 0 5

查看用户赞数与排名：
zscore user:1:20180106 mike　zrank user:1:20180106 mike

三、Redis 缓存雪崩与穿透

1，什么是雪崩?

前提:为节约内存,Redis 一般会做定期清除操作,
1),当查询 key=james 的值,此时 Redis 没有数据
2),如果有 5000 个用户并发来查询 key=james,全到 Mysql 里去查, Mysql 会挂掉,导致雪崩;



解决方案如下：

A,设置热点数据永远不过期。
B,加互斥锁，互斥锁参考代码如下

代码如下(若看不懂没关系, VIP 里会讲到代码实现):

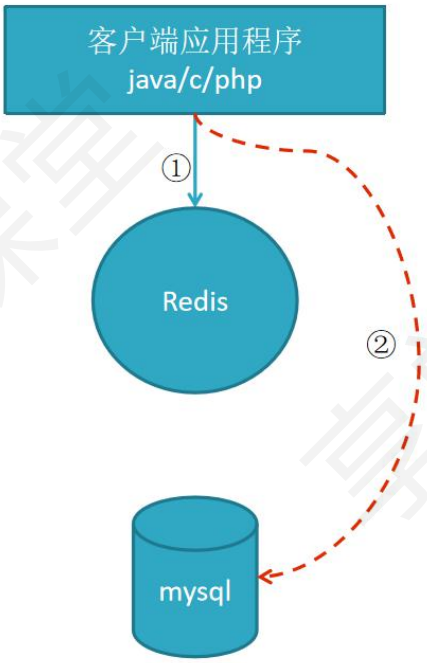
```
public static String getData(String key) throws InterruptedException
{
    //从缓存读取数据
    String result = getDataFromRedis(key);
    //缓存中不存在数据
    if (result == null)
    {
        //去获取锁，获取成功，去数据库取数据
        if (reenLock.tryLock())
        {
            //从数据获取数据
            result = getDataFromMysql(key);
            //更新缓存数据
            if (result != null)
            {
                setDataToCache(key,result);
            }
            //释放锁
            reenLock.unlock();
        }
        //获取锁失败
        else
        {
            //暂停100ms再重新去获取数据
            Thread.sleep(100);
            result =  getData(key);
        }
    }
    return result;
}
```

2，什么是穿透?

前提:黑客模拟一个不存在的订单号 xxxx

1,Redis 中无此值

2,Mysql 中也无此值，但一直被查询



解决方案:

1,对订单表所有数据查询出来放到布隆过滤器，经过布隆过滤器处理的数据很小(只存 0 或 1)

2,每次查订单表前,先到过滤器里查询当前订单号状态是 0 还是 1, 0 的话代表数据库没有数据,直接拒绝查询

四、redis 持久化

redis 支持 RDB 和 AOF 两种持久化机制，持久化可以避免因进程退出而造成数据丢失；

5.1RDB 持久化把当前进程数据生成快照（.rdb）文件保存到硬盘的过程，有手动触发和自动触发

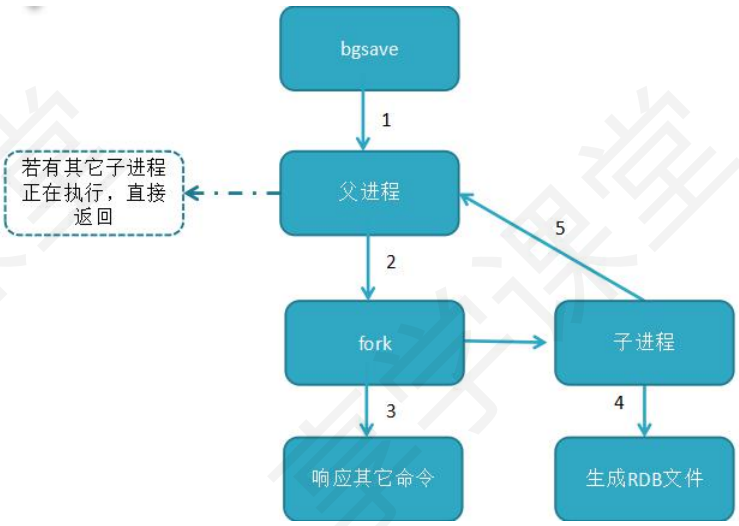
手动触发有 save 和 bgsave 两命令

save 命令：阻塞当前 Redis，直到 RDB 持久化过程完成为止，若内存实例比较大会造成长时间阻塞，线上环境不建议用它

bgsave 命令：redis 进程执行 fork 操作创建子线程，由子线程完成持久化，阻塞时间很短（微秒级），是 save 的优化,在执行 redis-cli shutdown 关闭 redis 服务时，如果没有开启 AOF 持久化，自动执行 bgsave;

显然 bgsave 是对 save 的优化。

bgsave 运行流程



5.2RDB 文件的操作

命令：config set dir /usr/local　//设置 rdb 文件保存路径

备份：bgsave　//将 dump.rdb 保存到 usr/local 下

恢复：将 dump.rdb 放到 redis 安装目录与 redis.conf 同级目录，重启 redis 即可

优点：1，压缩后的二进制文，适用于备份、全量复制，用于灾难恢复

2，加载 RDB 恢复数据远快于 AOF 方式

缺点：1，无法做到实时持久化，每次都要创建子进程，频繁操作成本过高

2，保存后的二进制文件，存在老版本不兼容新版本 rdb 文件的问题

5.3AOF 持久化

针对 RDB 不适合实时持久化，redis 提供了 AOF 持久化方式来解决

开启：redis.conf 设置：appendonly yes　(默认不开启，为 no)

默认文件名：appendfilename "appendonly.aof"

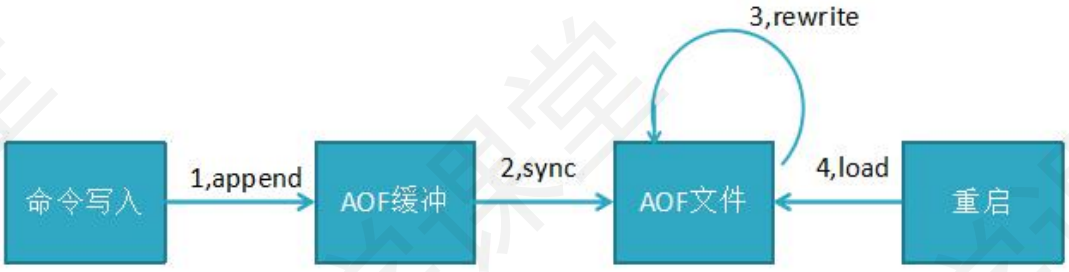
流程说明：1，所有的写入命令(set hset)会 append 追加到 aof_buf 缓冲区中

2，AOF 缓冲区向硬盘做 sync 同步

3，随着 AOF 文件越来越大，需定期对 AOF 文件 rewrite 重写，达到压缩

4，当 redis 服务重启，可 load 加载 AOF 文件进行恢复

AOF 持久化流程：命令写入(append),文件同步(sync),文件重写(rewrite),重启加载(load)



redis 的 AOF 配置详解：

appendonly yes　　//启用 aof 持久化方式

appendfsync always //每收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用

appendfsync everysec //每秒强制写入磁盘一次，性能和持久化方面做了折中，推荐
appendfsync no　　//完全依赖 os，性能最好,持久化没保证（操作系统自身的同步）
no-appendfsync-on-rewrite　yes　//正在导出 rdb 快照的过程中,要不要停止同步 aof
auto-aof-rewrite-percentage 100　//aof 文件大小比起上次重写时的大小,增长率 100%时,重写
auto-aof-rewrite-min-size 64mb　　//aof 文件,至少超过 64M 时,重写

如何从 AOF 恢复？

1. 设置 appendonly yes；
2. 将 appendonly.aof 放到 dir 参数指定的目录；
3. 启动 Redis，Redis 会自动加载 appendonly.aof 文件。

redis 重启时恢复加载 AOF 与 RDB 顺序及流程：

1，当 AOF 和 RDB 文件同时存在时，优先加载
2，若关闭了 AOF，加载 RDB 文件
3，加载 AOF/RDB 成功，redis 重启成功
4，AOF/RDB 存在错误，redis 启动失败并打印错误信息