

redis回顾

```
1 redis01 回顾
2   1, redis基础概念
3     1, 内存型数据库
4     2, 数据结构特别多, kv型存储
5     3, 单进程单线程
6   2, 通用命令
7     3, String 字符串类型
8       1, 缓存
9         点赞
10        秒杀
11       2, 并发计数
12         3, 验证码时效
13 redis02回顾
14   1, List 重两边操作 轻 中间操作
15     场景:
16       1, 生产者消费者模型
17       2, 遍历数据
18   2, hash
19     1, 压缩特点 【字段数512, 任意value<64字节】
20     2, 按需获取的特点
21     场景:
22       缓存
23       统计计数
24   3, 位图操作
25     set k1 ab    00000000 00000000
26     setbit key offset value
27     对完全不存在的key 直接执行 setbit
28     setbit k2 0 1    10000000
29 redis03回顾
30   1, 集合      交集差集并集
31   2, 有序集合  score 排行榜
32   3, 事务 - 不保证原子性, 报证一定的隔离性
33     开启事务后, 命令进入服务端的命令队列, 当服务端exec这个指令时, redis逐一执行命令
34     队列的redis命令
35       客户端技术 - 流水线
36       乐观锁 - watch
37   4, 持久化
38     rdb
39       存储: 实实在在的数据
40       触发: save 300 10
41       数据量: 全量持久化
42     aof
43       存储: 执行的命令
44       触发: 来一个写入命令, 就执行一次
45       数据量: 增量备份
```

redis_day01笔记

Redis介绍

• 特点及优点

- 1 1、开源的，使用C编写，基于内存且支持持久化
- 2 2、高性能的Key-value的NoSQL数据库
- 3 3、支持数据类型丰富，字符串strings，散列hashes，列表lists，集合sets，有序集合sorted sets 等等
- 4 4、支持多种编程语言（C C++ Python Java PHP ...）
- 5 5、单进程单线程

• 与其他数据库对比

- 1 1、MySQL：关系型数据库，表格，基于磁盘，慢
- 2 2、MongoDB：键值对文档型数据库，值为类似JSON文档，数据结构相对单一
- 3 3、Redis的诞生是为了解决什么问题？？
解决硬盘IO带来的性能瓶颈

• 应用场景

- 1 1, 缓存
- 2 2, 并发计数
 点赞 秒杀
- 3 3, 排行榜
- 4 4, 生产者消费者模型
- 5 ...

• redis版本

- 1 1、最新版本：5.0
- 2 2、常用版本：2.4、2.6、2.8、3.0(里程碑)、3.2、3.4、4.0(教学环境版本)、5.0

• Redis附加功能

- 1 1、持久化
 将内存中数据保存到磁盘中，保证数据安全，方便进行数据备份和恢复
- 2 2、过期键功能
 为键设置一个过期时间，让它在指定时间内自动删除
 <节省内存空间>
 # 音乐播放器，日播放排名，过期自动删除
- 3 3、事务功能
 原子的执行多个操作
- 4 4、主从复制
- 5 5、Sentinel哨兵

安装

• Ubuntu

```
# 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
```

配置文件详解

- 配置文件所在路径

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

- 设置连接密码

```
1 1、# 500 requirepass 密码  
2 2、重启服务  
3     sudo /etc/init.d/redis-server restart  
4 3、客户端连接  
5     redis-cli -h 127.0.0.1 -p 6379 -a 123456  
6     127.0.0.1:6379>ping
```

- 允许远程连接

```
1 1、注释掉本地IP地址绑定  
2     69行: # bind 127.0.0.1 ::1  
3 2、关闭保护模式(把yes改为no)  
4     88行: protected-mode no  
5 3、重启服务  
6     sudo /etc/init.d/redis-server restart
```

- 通用命令 **适用于所有数据类型**

```
1 #查看redis-server的信息  
2 info  
3 # 切换库(number的值在0-15之间, db0 ~ db15)  
4 select number  
5 # 查看键  
6 keys 表达式 # keys *  
7 # 数据类型  
8 type key  
9 # 键是否存在  
10 exists key  
11 # 删除键  
12 del key  
13 # 键重命名  
14 rename key newkey  
15 # 清除当前库中所有数据（慎用）  
16 flushdb  
17 # 清除所有库中所有数据（慎用）  
18 flushall
```

数据类型

字符串类型(string)

- 特点

- 1 1、字符串、数字，都会转为字符串来存储
- 2 2、以二进制的方式存储在内存中

- 字符串常用命令-必须掌握

```
1 # 1. 设置一个key-value
2 set key value
3 # 2. 获取key的值
4 get key
5 # 3. key不存在时再进行设置(nx)
6 set key value nx # not exists
7 # 4. 设置过期时间(ex)
8 set key value ex seconds
9
10 # 5. 同时设置多个key-value
11 mset key1 value1 key2 value2 key3 value3
12 # 6. 同时获取多个key-value
13 mget key1 key2 key3
```

- 字符串常用命令-作为了解

```
1 # 1. 获取长度
2 strlen key
3 # 2. 获取指定范围切片内容 [包含start stop]
4 getrange key start stop
5 # 3. 从索引值开始, value替换原内容
6 setrange key index value
```

- 数值操作-字符串类型数字(必须掌握)

```
1 # 整数操作
2 incrby key 步长
3 decrby key 步长
4 incr key : +1操作
5 decr key : -1操作
6 # 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
7 # 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
8 incrbyfloat key step
```

- string命令汇总

```
1 # 字符串操作
2 1、set key value
3 2、set key value nx
4 3、get key
5 3、mset key1 value1 key2 value2
6 4、mget key1 key2 key3
7 5、set key value nx ex seconds
8 6、strlen key
9 # 返回旧值并设置新值（如果键不存在，就创建并赋值）
```

```

10 7、getset key value
11 # 数字操作
12 7、incrby key 步长
13 8、decrby key 步长
14 9、incr key
15 10、decr key
16 11、incrbyfloat key number#(可为正数或负数)
17
18 # 设置过期时间的两种方式
19 # 方式一
20 1、set key value ex 3
21 # 方式二
22 1、set key value
23 2、expire key 5 # 秒
24 3、pexpire key 5 # 毫秒
25 # 查看存活时间
26 ttl key
27 -1 : 当前key存在，没有过期时间
28 -2 : 当前key不存在
29 >0 : key的剩余时间
30
31 # 删除过期
32 persist key

```

- **string数据类型注意**

```

1 # key命名规范
2 可采用 - wang:email
3 # key命名原则
4 1、key名字不宜过长，消耗内存，且在数据中查找这类键值的计算成本高
5 2、不宜过短，可读性较差
6 # 值
7 1、一个字符串类型的值最多能存储512M内容

```

- **业务场景**

- 缓存
 - 将mysql中的数据存储到redis字符串类型中
- 并发计数 - 点赞/秒杀
 - 说明：通过redis单进程单线程的特点，由redis负责计数，并发问题转为串行问题
- 带有效期的验证码 - 短信验证码
 - 借助过期时间，存放验证码；到期后，自动消亡

练习

```

1 1、查看 db0 库中所有的键
2 select 0
3 keys *
4
5 2、设置键 trill:username 对应的值为 user001，并查看
6     set trill:username user001
7
8 3、获取 trill:username 值的长度
9     strlen trill:username
10

```

```
11 4、一次性设置 trill:password 、trill:gender、trill:fansnumber 并查看（值自定义）
12     mset trill:password 123 trill:gender M trill:fansnumber 500
13
14 5、查看键 trill:score 是否存在
15     exists trill:score
16
17 6、增加10个粉丝
18     incrby trill:fansnumber 10
19
20 7、增加2个粉丝（一个一个加）
21     incr trill:fansnumber
22     incr trill:fansnumber
23 8、有3个粉丝取消关注你了
24 9、又有1个粉丝取消关注你了
25 10、思考、思考、思考...，清除当前库
26
27 11、一万个思考之后，清除所有库
28
```

列表数据类型 (List)

- 特点

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

- 列表常用命令

```
1 # 增
2 1、从列表头部压入元素
3     LPUSH key value1 value2
4     返回：list长度
5 2、从列表尾部压入元素
6     RPUSH key value1 value2
7     返回：list长度
8 3、从列表src尾部弹出1个元素，压入到列表dst的头部
9     RPOPLPUSH src dst
10    返回：被弹出的元素
11 4、在列表指定元素后/前插入元素
12     LINSERT key after|before value newvalue
13
14     linsert l1 before a z
15     返回：
16         1，如果命令执行成功，返回列表的长度
17         2，如果没有找到 pivot，返回 -1
18         3，如果 key 不存在或为空列表，返回 0
19
20 # 查
21 5、查看列表中元素
22     lrange
23     LRANGE key start stop
24     # 查看列表中所有元素：LRANGE key 0 -1
25 6、获取列表长度
```

```

26    LLEN key
27
28 # 删
29 7、从列表头部弹出1个元素
30     LPOP key
31 8、从列表尾部弹出1个元素
32     RPOP key
33 9、列表头部,阻塞弹出,列表为空时阻塞
34     BLPOP key timeout
35 10、列表尾部,阻塞弹出,列表为空时阻塞
36     BRPOP key timeout
37 # 关于BLPOP 和 BRPOP
38 1、如果弹出的列表不存在或者为空,就会阻塞
39 2、超时时间设置为0,就是永久阻塞,直到有数据可以弹出
40 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
41 11、删除指定元素
42     LREM key count value
43     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
44     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
45     count=0: 移除表中所有与value相等的值
46     返回: 被移除元素的数量
47
48 12、保留指定范围内的元素
49     LTRIM key start stop
50     返回: ok
51     样例:
52         LTRIM mylist1 0 2 # 只保留前3条
53         # 应用场景: 保存微博评论最后500条
54         LTRIM weibo:comments 0 499
55 # 改
56 13、将列表 key 下标为 index 的元素的值设置为 value
57     LSET key index newvalue

```

练习

```

1 1、查看所有的键
2     #keys *
3 2、向列表 spider:urls 中以RPUSH放入如下几个元素: 01_baidu.com、02_taobao.com、
4     03_sina.com、04_jd.com、05_xxx.com
5         #rpush spider:urls 01_baidu.com 02_taobao.com 03_sina.com 04_jd.com
6         05_xxx.com
7
8 3、查看列表中所有元素
9     #lrange spider:urls 0 -1
10 4、查看列表长度
11     #llen spider:urls
12 5、将列表中01_baidu.com 改为 01_tmall.com
13     #lset spider:urls 0 01_tmall.com
14 6、在列表中04_jd.com之后再加1个元素 02_taobao.com
15     #linsert spider:urls after 04_jd.com 02_taobao.com
16 7、弹出列表中的最后一个元素
17     # rpop
18 8、删除列表中所有的 02_taobao.com
19     # lrem spider:urls 0 02_taobao.com
20 9、剔除列表中的其他元素, 只剩前3条
21     # ltrim spider:urls 0 2

```

python交互redis

- 模块(redis)

Ubuntu

```
1 | sudo pip3 install redis
```

- 使用流程

```
1 | import redis
2 | # 创建数据库连接对象
3 | r = redis.Redis(host='127.0.0.1', port=6379, db=0, password='123456')
```

- 通用命令代码示例

```
1 | import redis
2 |
3 | #生成连接对象
4 | r = redis.Redis(host='127.0.0.1', port=6379, db=0, password='123456')
5 |
6 | #无密码
7 | #r = redis.Redis(host='127.0.0.1', port=6379, db=0)
8 |
9 | #基础通用命令
10 | #key_list = r.keys('*')
11 | #[b'u2', b'username', b'username5', b'l1', b'u1', b'u3', b'z4', b'u4']
12 | #print(key_list)
13 |
14 | #print(r.exists('u1'))
15 | #print(r.delete('u1'))
```

- python操作list

```
1 | #r.lpush('pyl1', 'a', 'b', 'c', 'd')
2 | #[b'd', b'c', b'b', b'a']
3 | #print(r.lrange('pyl1', 0, -1))
4 |
5 | #print(r.rpop('pyl1'))
6 | #print(r.ltrim('pyl1', 0, 1))
7 | #print(r.lrange('pyl1', 0, -1))
```

list案例: 一个进程负责生产任务, 一个进程负责消费任务

进程1: 生产者

```

1 #生产者
2 import redis
3
4 r = redis.Redis(host='127.0.0.1', port=6379, db=0, password='123456')
5
6 # 任务类别/收件人/发件人/内容
7 s = '%s_%s_%s_%s'%( 'sendEmail' , 'guoxiaonao@tedu.cn' ,
8 'guo@tedu.cn' , 'hahaha')
9 #任务发到redis的原则 先进先出[lpush, brpop]
10 r.lpush('pylt1', s)

```

进程2: 消费者

```

1 #消费者
2 import redis
3 r = redis.Redis(host='127.0.0.1', port=6379, db=0, password='123456')
4
5 while True:
6
7     task = r.brpop('pylt1', 10)
8     #task : (b'pylt1', b'sendEmail_guoxiaonao@tedu.cn_guo@tedu.cn_hahaha')
9     print(task)
10    if task:
11        task_data = task[1]
12        task_str = task_data.decode()
13        task_list = task_str.split('_')
14        print('--receiver task, task type is %s'%(task_list[0]))
15
16    else:
17        print('--no task--')

```

- python操作string

```

1 # r.set('pys1', 'guoxiaonao')
2 # print(r.get('pys1'))
3
4 #r.mset({'pys2':'guo', 'pys3':'xiao'})
5 #[b'guo', b'xiao']
6 #print(r.mget('pys2','pys3'))
7
8 #print(r.incr('pys6'))
9 #print(r.incrby('pys6',10))

```

淘汰策略

1, 主动出击

1, 将带过期时间的key存到一个独立的字典中

默认每100毫秒进行一次过期扫描

1, 在过期字典中随机 20个 key

2, 检查过期时间, 删除已过期的key

3, 如果过期key 比例 超过 1/4 重复 1-3

默认25ms 超时时间， 避免扫描卡死

问题： 大量key同时过期， redis会有卡顿现象

解决方案： set key value ex 1000 + random.randint(1,100)

2, 惰性删除

1, get 【检查key的过期时间， key过期，直接删除】

2, set maxmemory检查

noeviction 拒接写服务，可接受读请求 - 默认配置

volatile-lru 尝试淘汰设置了过期时间的key, [最少使用原则]

allkeys-lru 尝试淘汰所有key [最少使用原则]

redis_day02笔记

位图操作bitmap

定义

```
1 1、位图不是真正的数据类型，它是定义在字符串类型中
2 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
3 # 1MB = 1024KB
4 # 1KB = 1024Byte(字节)
5 # 1Byte = 8bit(位)
6
7
8 1_login_20200518 : 1
9 2_login_20200518 : 1
10
11 1_login: [20200518, 20200605]
12
13 1_login: 1100000000100
14
```

强势点

1 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

SETBIT 命令

- 说明：设置某位置上的二进制值
- 语法：SETBIT key offset value
- 参数：offset - 偏移量 从0开始
value - 0或者1
- 示例：

```
1 # 默认扩展位以0填充
2 127.0.0.1:6379> SET mykey ab
3 OK
4 127.0.0.1:6379> GET mykey
5 "ab"
6 127.0.0.1:6379> SETBIT mykey 0 1
7 (integer) 0
8 127.0.0.1:6379> GET mykey
9 "\xe1b"
10 127.0.0.1:6379>
```

GETBIT 命令

- 说明：获取某一位上的值
- 语法：GETBIT key offset
- 示例：

```
1 127.0.0.1:6379> GETBIT mykey 3
2 (integer) 0
3 127.0.0.1:6379> GETBIT mykey 0
4 (integer) 1
5 127.0.0.1:6379>
```

BITCOUNT 命令

- 说明：统计键所对应的值中有多少个1
- 语法：BITCOUNT key start end
- 参数：start/end 代表的是字节索引
- 示例：

```
1 127.0.0.1:6379> SET mykey1 ab
2 OK
3 127.0.0.1:6379[4]> BITCOUNT mykey
4 (integer) 6
5 127.0.0.1:6379[4]> BITCOUNT mykey 0 0
6 (integer) 3
7
```

应用场景案例

```
1 # 网站用户的上线次数统计（寻找活跃用户）
2     用户名为key，上线的天作为offset，上线设置为1
3 # 示例
4     用户名为 user1:login 的用户，今年第1天上线，第30天上线
5     SETBIT user1:login 0 1
6     SETBIT user1:login 29 1
7     BITCOUNT user1:login
```

代码实现

```
1 r.setbit('key', 索引位, 1/0)
2
3 登陆
4     r.setbit('1_login_2020', 4 ,1)
5
```

```
6 注册:  
7     r.lpush('users', '用户id')  
8     #筛选出 注册用户中， 2020年登录次数超过100次的玩家，发放奖品  
9     for user_id in range('users', 0, -1):  
10  
11         key = '%s_login_2020'%(user_id)  
12         login_count = r.bitcount(key)  
13         if login_count > 1:  
14             符号要求  
15
```

Hash散列数据类型

- 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（约43亿）

- 优点

- 1、节约内存空间 – 特定条件下【1, 字段小于512个, 2: value不能超过64字节】
- 2、可按需获取字段的值

- 缺点 (不适合hash情况)

- 1, 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
- 2, 存储消耗大于字符串结构

- 基本命令操作

```
1 # 1、设置单个字段  
2 HSET key field value  
3 HSETNX key field value  
4 # 2、设置多个字段  
5 HMSET key field1 value field2 value  
6 # 3、返回字段个数  
7 HLEN key  
8 # 4、判断字段是否存在（不存在返回0）  
9 HEXISTS key field  
10 # 5、返回字段值  
11 HGET key field  
12 # 6、返回多个字段值  
13 HMGET key field filed  
14 # 7、返回所有的键值对  
15 HGETALL key  
16 # 8、返回所有字段名  
17 HKEYS key  
18 # 9、返回所有值  
19 HVALS key  
20 # 10、删除指定字段  
21 HDEL key field  
22 # 11、在字段对应值上进行整数增量运算  
23 HINCRBY key filed increment  
24 # 12、在字段对应值上进行浮点数增量运算
```

python操作hash

```

1 # 1、更新一条数据的属性，没有则新建
2 hset(name, key, value)
3 # 2、读取这条数据的指定属性， 返回字符串类型
4 hget(name, key)
5 # 3、批量更新数据（没有则新建）属性，参数为字典
6 hmset(name, mapping)
7 # 4、批量读取数据（没有则新建）属性
8 hmget(name, keys)
9 # 5、获取这条数据的所有属性和对应的值，返回字典类型
10 hgetall(name)
11 # 6、获取这条数据的所有属性名，返回列表类型
12 hkeys(name)
13 # 7、删除这条数据的指定属性
14 hdel(name, *keys)

```

Python代码hash散列

```

1 #r.hset('pyh8', 'uname', 'wangweichao')
2 #{b'uname': b'wangweichao'}
3 #print(r.hgetall('pyh8'))
4 #r.hmset('pyh8', {'age':22, 'desc':'spider'})
5 #print(r.hgetall('pyh8'))

```

应用场景：用户维度数据统计

```

1 用户维度统计
2 统计数包括：关注数、粉丝数、喜欢商品数、发帖数
3 用户为key，不同维度为field，value为统计数
4 比如关注了5人
5 HSET user:10000 fans 5
6 HINCRBY user:10000 fans 1

```

应用场景：缓存 - redis+mysql+hash组合使用

- 原理

```

1 用户想要查询个人信息
2 1、到redis缓存中查询个人信息
3 2、redis中查询不到，到mysql查询，并缓存到redis
4 3、再次查询个人信息

```

- 代码实现

```

1 博客 个人主页
2 项目 amysite1 数据库 amysite1 应用user
3 class User
4     username -> str 用户
5     age -> int
6
7 个人主页 /user/detail/1
8

```

```

9    缓存思想: 1, 先找缓存中有没有, 2, 缓存没有 - 去查数据库/存储缓存 , 3 缓存有数据
10   则直接返回
11   return 'username %s age %s'
12   更新请求 /user/update/<int:user_id>
13   /user/update/1?age=20
14   更新mysql中的age
15   删除redis缓存数据
16
17
18   from django.http import HttpResponse
19   from django.shortcuts import render
20   from .models import User
21   import redis
22   # Create your views here.
23   r = redis.Redis(host='127.0.0.1', port=6379, db=0, password='123456')
24
25   def user_detail(request, user_id):
26       #1,先查缓存
27       # 没有: 数据库 -> 数据回写缓存
28       # 有: 返回缓存内容
29       cache_key = 'user:%s'%(user_id)
30       if r.exists(cache_key):
31           data = r.hgetall(cache_key)
32           #{b'username':b'guoxiaonao', b'age':b'20'}
33           new_data = {k.decode():v.decode() for k,v in data.items()}
34           username = new_data['username']
35           age = new_data['age']
36           html = 'Cache username is %s age is %s'%(username, age)
37           return HttpResponse(html)
38
39       #无缓存时
40       try:
41           user = User.objects.get(id=user_id)
42       except Exception as e:
43           print(e)
44           return HttpResponse('--no user')
45
46       username = user.username
47       age = user.age
48       html = 'username is %s age is %s' % (username, age)
49       #更新缓存
50       r.hmset(cache_key, {'username':username, 'age':age})
51       r.expire(cache_key, 60)
52       return HttpResponse(html)
53
54
55   def user_update(request, user_id):
56
57       #/user/update/1?age=30
58       age = request.GET.get('age', 0)
59
60       try:
61           user = User.objects.get(id=user_id)
62       except Exception as e:
63           return HttpResponse('--no user')
64
65       user.age = age

```

```
66     user.save()
67
68     #删除缓存
69     cache_key = 'user:%s'%(user_id)
70     r.delete(cache_key)
71
72     return HttpResponse('--update is ok--')
73
```

集合数据类型 (set)

- 特点

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

- 基本命令

```
1 # 1、增加一个或者多个元素，自动去重；返回值为成功插入到集合的元素个数
2 SADD key member1 member2
3 # 2、查看集合中所有元素
4 SMEMBERS key
5 # 3、删除一个或者多个元素，元素不存在自动忽略
6 SRLEM key member1 member2
7 # 4、元素是否存在
8 SISMEMBER key member
9 # 5、随机返回集合中指定个数的元素，默认为1个
10 SRANDMEMBER key [count]
11 # 6、弹出成员
12 SPOP key [count]
13 # 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
14 SCARD key
15 # 8、把元素从源集合移动到目标集合
16 SMOVE source destination member
17
18 # 9、差集(number1 1 2 3 number2 1 2 4 结果为3)
19 SDIFF key1 key2
20 # 10、差集保存到另一个集合中
21 SDIFFSTORE destination key1 key2
22
23 # 11、交集
24 SINTER key1 key2
25 SINTERSTORE destination key1 key2
26
27 # 11、并集
28 SUNION key1 key2
29 SUNIONSTORE destination key1 key2
```

案例: 新浪微博的共同关注

```

1 # 需求：当用户访问另一个用户的时候，会显示出两个用户共同关注过哪些相同的用户
2 # 设计：将每个用户关注的用户放在集合中，求交集即可
3 # 实现：
4     user001 = {'peiqi', 'qiaozhi', 'danni'}
5     user002 = {'peiqi', 'qiaozhi', 'lingyang'}
6
7 user001和user002的共同关注为：
8     SINTER user001 user002
9     结果为：{'peiqi', 'qiaozhi'}

```

python操作set

```

1 #r.sadd('pyset1', 'tom', 'jack')
2 #{b'jack', b'tom'}
3 #print(r.smembers('pyset1'))
4
5 #r.sadd('pyset2', 'tom', 'lily', 'xixi')
6 #{b'tom'}
7 print(r.sinter('pyset1', 'pyset2'))

```

有序集合sortedset

- 特点

```

1 1、有序、去重
2 2、元素是字符串类型
3 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 4、最多包含2^32-1元素

```

- 示例

一个保存了水果价格的有序集合

分值	2.0	4.0	6.0	8.0	10.0
元素	西瓜	葡萄	芒果	香蕉	苹果

一个保存了员工薪水的有序集合

分值	6000	8000	10000	12000	
元素	lucy	tom	jim	jack	

一个保存了正在阅读某些技术书的人数

分值	300	400	555	666	777
元素	核心编程	阿凡提	本拉登	阿姆斯特朗	比尔盖茨

- 有序集合常用命令

```

1 # 在有序集合中添加一个成员 返回值为 成功插入到集合中的元素个数
2 zadd key score member

```

```

3 # 查看指定区间元素（升序）
4 zrange key start stop [withscores]
5 # 查看指定区间元素（降序）
6 zrevrange key start stop [withscores]
7 # 查看指定元素的分值
8 zscore key member
9
10 # 返回指定区间元素
11 # offset : 跳过多少个元素
12 # count : 返回几个
13 # 小括号 : 开区间 zrangebyscore fruits (2.0 8.0
14 zrangebyscore key min max [withscores] [limit offset count]
15 # 每页显示10个成员,显示第5页的成员信息:
16 # limit 40 10
17 # MySQL: 每页显示10条记录,显示第5页的记录
18 # limit 40,10
19 # limit 2,3 显示: 第3 4 5条记录
20
21 # 删除成员
22 zrem key member
23 # 增加或者减少分值
24 zincrby key increment member
25
26 阅读量
27
28 文章id_read_count 普通集合 ip
29 文章_阅读量 有序集合
30
31
32 # 返回元素排名
33 zrank key member
34 # 返回元素逆序排名
35 zrevrank key member
36 # 删除指定区间内的元素
37 zremrangebyscore key min max
38 # 返回集合中元素个数
39 zcard key
40 # 返回指定范围内元素的个数
41 zcount key min max
42 zcount salary 6000 8000
43 zcount salary (6000 8000# 6000<salary<=8000
44 zcount salary (6000 (8000#6000<salary<8000
45
46 # 并集
47 zunionstore destination numkeys key [weights 权重值] [AGGREGATE SUM|MIN|MAX]
48 # zunionstore salary3 2 salary salary2 weights 1 0.5 AGGREGATE MAX
49 # 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
50
51 # 交集: 和并集类似, 只取相同的元素
52 zinterstore destination numkeys key1 key2 weights weight AGGREGATE SUM(默
认)|MIN|MAX

```

python操作sorted set

```

1 #r.zadd('pyss1', {'tom':6000,'jim':5000})
2 #[b'jim', 5000.0), (b'tom', 6000.0)]
3 #print(r.zrange('pyss1',0,-1, withscores=True))
4
5 #print(r.zrangebyscore('pyss1', '(5000', 7500, withscores=True))
6
7 #r.zadd('pyss2', {'tom': 8000})
8
9 #r.zinterstore('pyss3',('pyss1','pyss2'), aggregate='max')
10
11 #print(r.zrange('pyss3', 0, -1, withscores=True))

```

redis_day02回顾

五大数据类型及应用场景

类型	特点	使用场景
string	简单key-value类型, value可为字符串和数字	常规计数 (微博数, 粉丝数等功能)
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	存储部分可能需要变更的数据 (比如用户信息)
list	有序可重复列表	消息队列等
set	无序不可重复列表	存储并计算关系 (如微博, 关注人或粉丝存放在集合, 可通过交集、并集、差集等操作实现如共同关注、共同喜好等功能)
sorted set	每个元素带有分值的集合	各种排行榜

redis_day03笔记

事务

定义：事务指程序中一系列严密的操作逻辑【sql语句】，所有操作必须全部完成，或者全部不完成，不能出现中间状态

转账：小明转账 200 元 -> 小红

事务的四大特性 (ACID)

原子性 (Atomicity)：事务中的操作，要么全都执行成功，要么全都不执行

一致性(Consistency): 事务执行始得数据从一个状态转换为另一个状态，这个过程保持完整性

隔离性(Isolation)：当多个并发访问数据库时，多个事务之间相互隔离，不能被其他事务操作所干扰

持久性 (Durability)：事务完成后，对于数据的改变是永久的

特点

- 1 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 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

事务命令

```
1 1、MULTI # 开启事务          mysql begin
2 2、命令1 # 执行命令
3 3、命令2 ... ...
4 4、EXEC # 提交到数据库执行    mysql commit
5 4、DISCARD # 取消事务        mysql 'rollback'
```

使用步骤

```
1 # 开启事务
2 127.0.0.1:6379> MULTI
3 OK
4 # 命令1入队列
5 127.0.0.1:6379> INCR n1
6 QUEUED
7 # 命令2入队列
8 127.0.0.1:6379> INCR n2
9 QUEUED
10 # 提交到数据库执行
11 127.0.0.1:6379> EXEC
12 1) (integer) 1
13 2) (integer) 1
```

事务中命令错误处理

```
1 # 1、命令语法错误，命令入队失败，直接自动discard退出这个事务
2 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）
3 处理方案：语法错误则自动执行discard
4
5 案例：
6 127.0.0.1:6379[7]> MULTI
7 OK
8 127.0.0.1:6379[7]> get a
9 QUEUED
10 127.0.0.1:6379[7]> getsss a
11 (error) ERR unknown command 'getsss'
12 127.0.0.1:6379[7]>
13 127.0.0.1:6379[7]>
14 127.0.0.1:6379[7]> EXEC
15 (error) EXECABORT Transaction discarded because of previous errors.
16
17 # 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
18 我们执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）
19 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，其他命令都
将会被执行。
```

```
20
21 案例
22 127.0.0.1:6379> MULTI
23 OK
24 127.0.0.1:6379> SET num 10
25 QUEUED
26 127.0.0.1:6379> LPOP num
27 QUEUED
28 127.0.0.1:6379> EXEC
29 1) OK
30 2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
31 127.0.0.1:6379> GET num
32 "10"
33 127.0.0.1:6379>
```

思考为什么redis不支持回滚？

pipeline 流水线

定义：批量执行redis命令，减少通信io

注意：此为客户端技术

示例

```
1 import redis
2 # 创建连接池并连接到redis
3 pool = redis.ConnectionPool(host = '127.0.0.1', db=0, port=6379)
4 r = redis.Redis(connection_pool=pool)
5
6 pipe = r.pipeline()
7 pipe.set('fans', 50)
8 pipe.incr('fans')
9 pipe.incrby('fans', 100)
10 pipe.execute()
```

性能对比

```
1 # 创建连接池并连接到redis
2 pool = redis.ConnectionPool(host = '127.0.0.1', db=0, port=6379)
3 r = redis.Redis(connection_pool=pool)
4
5 def withpipeline(r):
6     p = r.pipeline()
7     for i in range(1000):
8         key = 'test1' + str(i)
9         value = i+1
10        p.set(key, value)
11    p.execute()
12
13 def withoutpipeline(r):
14     for i in range(1000):
15         key = 'test2' + str(i)
16         value = i+1
17         r.set(key, value)
```

python 操作 redis事务

```
1 | with r.pipeline(transaction=true) as pipe
2 |     pipe.multi()
3 |     pipe.incr("books")
4 |     pipe.incr("books")
5 |     values = pipe.execute()
6 |
7 |     -> "multi  incr books  incr books  exec"
```

watch - 乐观锁

作用：事务过程中，可对指定key进行监听，命令提交时，若被监听key对应的值未被修改时，事务方可提交成功，否则失败

```
1 | > watch books
2 | OK
3 | > multi
4 | OK
5 | > incr books
6 | QUEUED
7 | > exec # 事务执行失败
8 | (nil)
9 |
10|
11| watch之后，再开一个终端进入redis
12| > incr books # 修改book值
13| (integer) 1
14|
15|
16|
```

python操作watch

```
1 | #同时对一个账户进行操作， 当前余额 * 2
2 | import redis
3 | import time
4 | pool = redis.ConnectionPool(host='127.0.0.1', db=1, port=6379,
5 |                             password='123456')
6 | r = redis.Redis(connection_pool=pool)
7 |
8 | def double_account(user_id):
9 |
10|     key = 'account_%s'%(user_id)
11|     with r.pipeline(transaction=True) as pipe:
12|         while True:
13|             try:
14|                 #pipe.watch命令 调用后 即刻发给服务器
15|                 pipe.watch(key)
16|                 value = int(r.get(key))
17|                 value *= 2
18|                 print('--new value is %s'%(value))
19|                 print('--sleep is start')
20|                 time.sleep(20)
21|                 print('--sleep is over')
```

```
22         pipe.multi()
23         pipe.set(key, value)
24         pipe.execute()
25         break
26     except redis.WatchError:
27         print('---value changed')
28         continue
29
30     return int(r.get(key))
31
32
33 if __name__ == '__main__':
34     #account_guoxiaonao
35     print(double_account('guoxiaonao'))
36
```

数据持久化

持久化定义

1 | 将数据从掉电易失的内存放到永久存储的设备上

为什么需要持久化

1 | 因为所有的数据都在内存上，所以必须得持久化

RDB模式（默认开启）

- 1 | 1、保存真实的数据
- 2 | 2、将服务器包含的所有数据库数据以二进制文件的形式保存到硬盘里面
- 3 | 3、默认文件名：`/var/lib/redis/dump.rdb`

创建rdb文件的两种方式

方式一：redis终端中使用SAVE或者BGSAVE命令

```
1 127.0.0.1:6379> SAVE
2 OK
3 # 特点
4 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
5 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
6 # 工作中定时持久化保存一个文件
7
8 127.0.0.1:6379> BGSAVE
9 Background saving started
10 # 执行过程如下
11 1、客户端发送 BGSAVE 给服务器
12 2、服务器马上返回 Background saving started 给客户端
13 3、服务器 fork() 子进程做这件事情
14 4、服务器继续提供服务
15 5、子进程创建完RDB文件后再告知Redis服务器
16
```

```
17 # 配置文件相关
18 /etc/redis/redis.conf
19 263行: dir /var/lib/redis # 表示rdb文件存放路径
20 253行: dbfilename dump.rdb # 文件名
21
22 # 两个命令比较
23 SAVE比BGSAVE快, 因为需要创建子进程, 消耗额外的内存
24
25 # 补充: 可以通过查看日志文件来查看redis都做了哪些操作
26 # 日志文件: 配置文件中搜索 logfile
27 logfile /var/log/redis/redis-server.log
```

方式二：设置配置文件条件满足时自动保存（使用最多）

```
1 # redis配置文件默认
2 218行: save 900 1
3 219行: save 300 10
4 220行: save 60 10000
5 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
6
7 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
8 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
9
10 # 该配置项也可以在命令行执行 [不推荐]
11 redis>save 60 10000
```

RDB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

AOF (AppendOnlyFile)

```
1 1、存储的是命令, 而不是真实数据
2 2、默认不开启
3 # 开启方式 (修改配置文件)
4 1、sudo vim /etc/redis/redis.conf
5 672行: appendonly yes # 把 no 改为 yes
6 676行: appendfilename "appendonly.aof"
7 2、重启服务
8 sudo /etc/init.d/redis-server restart
```

AOF持久化原理及优点

```
1 # 原理
2   1、每当有修改数据库的命令被执行时,
3     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
4       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
5
6 # 优点
7   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
8     丢失一秒钟的数据, 这比RDB持久化丢失的数据要少的多
```

特殊说明

```
1 # 因为
2   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
3     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
4       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
5         (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
6
7 # 所以
8   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
9     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
10    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
```

策略 - 配置文件

```
1 # 打开配置文件:/etc/redis/redis.conf, 找到相关策略如下
2 1、701行: always
3   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
4     任何已经成功执行的命令数据
5 2、702行: everysec (# 默认)
6   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
7     丢失1秒的数据
8 3、703行: no
9   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
10    数量不确定
11
12 # 运行速度比较
13 always: 速度慢
14 everysec和no都很快, 默认值为everysec
```

AOF重写

思考: AOF文件中是否会产生很多的冗余命令?

```
1 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
2    器可以产生一个新的AOF文件
3      -- 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
4      -- 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文件的提及通常会小很多
5      -- 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AOF文件	重写后的AOF文件
select 0	SELECT 0
sadd myset peiqi	SADD myset peiqi qiaozhi danni lingsyang
sadd myset qiaozhi	SET msg 'hello tarena'
sadd myset danni	RPUSH mylist 2 3 5
sadd myset lingsyang	
INCR number	
INCR number	
DEL number	
SET message 'hello world'	
SET message 'hello tarena'	
RPUSH mylist 1 2 3	
RPUSH mylist 5	
LPOP mylist	

AOF重写-触发

```

1 1、客户端向服务器发送BGREWRITEAOF命令
2 127.0.0.1:6379> BGREWRITEAOF
3   Background append only file rewriting started
4
5 2、修改配置文件让服务器自动执行BGREWRITEAOF命令
6   auto-aof-rewrite-percentage 100
7   auto-aof-rewrite-min-size 64mb
8   # 解释
9     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
10    # 第一次重写新增: 64M
11    # 第二次重写新增: 128M
12    # 第三次重写新增: 256M (新增128M)

```

RDB和AOF持久化对比

RDB持久化	AOF持久化
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

1 | # 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。

数据恢复（无需手动操作）

- 1 | 既有dump.rdb，又有appendonly.aof，恢复时找谁？
- 2 | 先找appendonly.aof

配置文件常用配置总结

```

1 # 设置密码
2 1、requirepass password
3 # 开启远程连接
4 2、bind 127.0.0.1 ::1 注释掉
5 3、protected-mode no 把默认的 yes 改为 no
6 # rdb持久化-默认配置
7 4、dbfilename 'dump.rdb'
8 5、dir /var/lib/redis
9 # rdb持久化-自动触发(条件)
10 6、save 900 1
11 7、save 300 10
12 8、save 60 10000
13 # aof持久化开启
14 9、appendonly yes
15 10、appendfilename 'appendonly.aof'
16 # aof持久化策略
17 11、appendfsync always
18 12、appendfsync everysec # 默认
19 13、appendfsync no
20 # aof重写触发
21 14、auto-aof-rewrite-percentage 100
22 15、auto-aof-rewrite-min-size 64mb
23 # 设置为从服务器
24 16、salveof <master-ip> <master-port>

```

Redis相关文件存放路径

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

Redis主从复制

- 定义

```
1 | 1、一个Redis服务可以有多个该服务的复制品，这个Redis服务成为master，其他复制品成为slaves  
2 | 2、master会一直将自己的数据更新同步给slaves，保持主从同步  
3 | 3、只有master可以执行写命令，slave只能执行读命令
```

- 作用

```
1 | 1、分担了读的压力（高并发），提高服务能力  
2 | 2、避免单点问题 【如果系统中 一个进程挂掉，整个系统挂掉，即为单点问题】
```

- 原理

```
1 | 从服务器执行客户端发送的读命令，比如GET、LRANGE、SMEMBERS、HGET、ZRANGE等等，客户端可以连接slaves执行读请求，来降低master的读压力
```

- 实现方式

- 方式一 (Linux命令行实现)

```
redis-server --slaveof --masterauth
```

```
1 | # 从服务端  
2 | redis-server --port 6300 --slaveof 127.0.0.1 6379  
3 | # 从客户端  
4 | redis-cli -p 6300  
5 | 127.0.0.1:6300> keys *  
6 | # 发现是复制了原6379端口的redis中数据  
7 | 127.0.0.1:6300> set mykey 123  
8 | (error) READONLY You can't write against a read only slave.  
9 | 127.0.0.1:6300>  
10 | # 从服务器只能读数据，不能写数据
```

- 方式二 (Redis命令行实现)

```
1 | # 两条命令  
2 | 1、>slaveof IP PORT  
3 | 2、>slaveof no one  
4 |  
5 | # 服务端启动  
6 | redis-server --port 6301  
7 | # 客户端连接  
8 | tarena@tedu:~$ redis-cli -p 6301  
9 | 127.0.0.1:6301> keys *
```

```

10 1) "myset"
11 2) "mylist"
12 127.0.0.1:6301> set mykey 123
13 OK
14 # 切换为从
15 127.0.0.1:6301> slaveof 127.0.0.1 6379
16 OK
17 127.0.0.1:6301> set newkey 456
18 (error) READONLY You can't write against a read only slave.
19 127.0.0.1:6301> keys *
20 1) "myset"
21 2) "mylist"
22 # 再切换为主
23 127.0.0.1:6301> slaveof no one
24 OK
25 127.0.0.1:6301> set name hello
26 OK
27

```

◦ 方式三(利用配置文件)

```

1 # 每个redis服务，都有1个和他对应的配置文件
2 # 两个redis服务
3   1、6379 -> /etc/redis/redis.conf
4   2、6300 -> /home/tarena/redis_6300.conf
5
6 # 修改配置文件
7 vi redis_6300.conf
8 slaveof 127.0.0.1 6379
9 port 6300
10
11 # 启动redis服务
12 redis-server redis_6300.conf
13 # 客户端连接测试
14 redis-cli -p 6300
15 127.0.0.1:6300> hset user:1 username guods
16 (error) READONLY You can't write against a read only slave.

```

问题：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，能够实现自动，这就需要Sentinel哨兵，实现故障转移Failover操作

演示

```

1 1、启动端口6400redis，设置为6379的slave
2     redis-server --port 6400
3     redis-cli -p 6400
4     redis>slaveof 127.0.0.1 6379
5 2、启动端口6401redis，设置为6379的slave
6     redis-server --port 6401

```

```
7 redis-cli -p 6401
8     redis>slaveof 127.0.0.1 6379
9 3、关闭6379redis
10    sudo /etc/init.d/redis-server stop
11 4、把6400redis设置为master
12    redis-cli -p 6400
13     redis>slaveof no one
14 5、把6401的redis设置为6400redis的slave
15    redis-cli -p 6401
16     redis>slaveof 127.0.0.1 6400
17 # 这是手动操作，效率低，而且需要时间，有没有自动的？？？
```

Sentinel哨兵

Redis之哨兵 - sentinel

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

案例演示

1、环境搭建

```
1 # 共3个redis的服务
2 1、启动6379的redis服务器
3     sudo /etc/init.d/redis-server start
4 2、启动6380的redis服务器，设置为6379的从
5     redis-server --port 6380
6     tarena@tedu:~$ redis-cli -p 6380
7     127.0.0.1:6380> slaveof 127.0.0.1 6379
8     OK
9 3、启动6381的redis服务器，设置为6379的从
10    redis-server --port 6381
11    tarena@tedu:~$ redis-cli -p 6381
12    127.0.0.1:6381> slaveof 127.0.0.1 6379
```

2、安装并搭建sentinel哨兵

```
1 # 1、安装redis-sentinel
2 sudo apt install redis-sentinel
3 验证: sudo /etc/init.d/redis-sentinel stop
4 # 2、新建配置文件sentinel.conf
5 port 26379
6 sentinel monitor tedu 127.0.0.1 6379 1
7 # 3、启动sentinel
8 方式一: redis-sentinel sentinel.conf
9 方式二: redis-server sentinel.conf --sentinel
10 #4、将master的redis服务终止，查看从是否会提升为主
11 sudo /etc/init.d/redis-server stop
12 # 发现提升6381为master，其他两个为从
13 # 在6381上设置新值，6380查看
14 127.0.0.1:6381> set name tedu
15 OK
16
17 # 启动6379，观察日志，发现变为了6381的从
18 主从+哨兵基本就够了
```

sentinel.conf解释

```
1 # sentinel监听端口，默认是26379，可以修改
2 port 26379
3 # 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
4 #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
5 sentinel monitor <master-name> <ip> <redis-port> <quorum>
6
7 #如果master有密码，则需要添加该配置
8 sentinel auth-pass <master-name> <password>
9
10 #master多久失联才认为是不可用了，默认是30秒
11 sentinel down-after-milliseconds <master-name> <milliseconds>
```

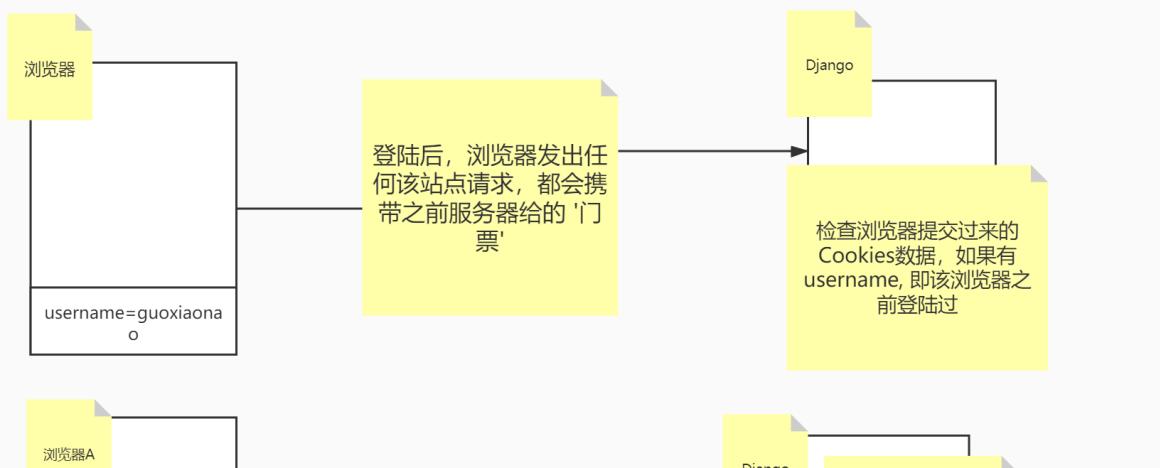
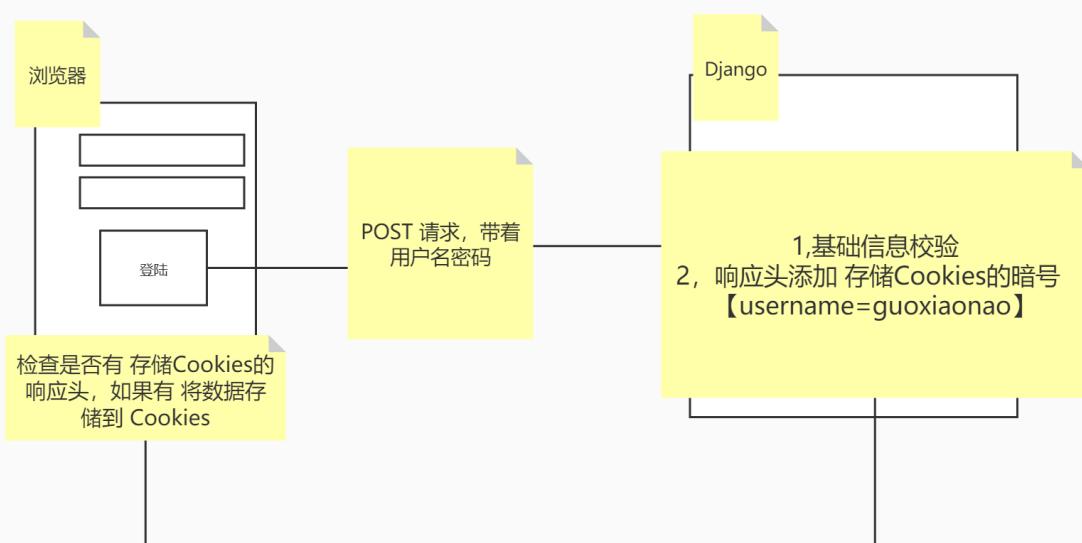
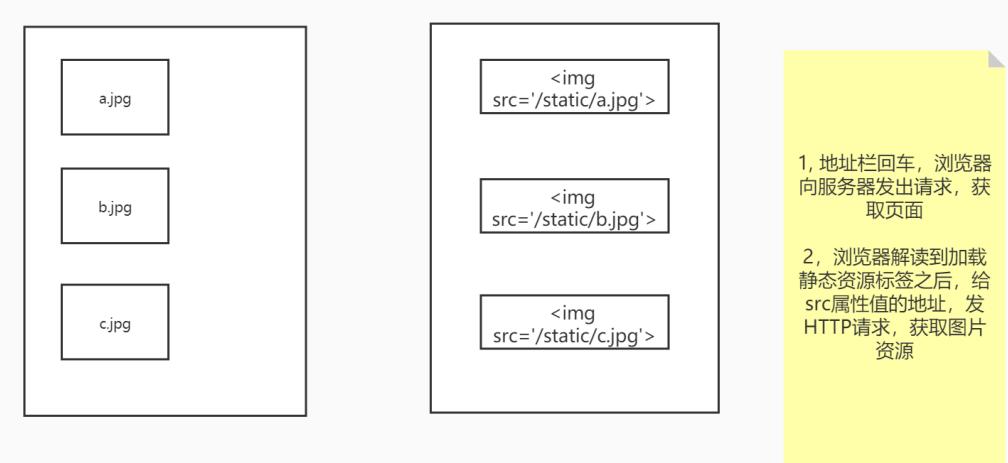
python获取master

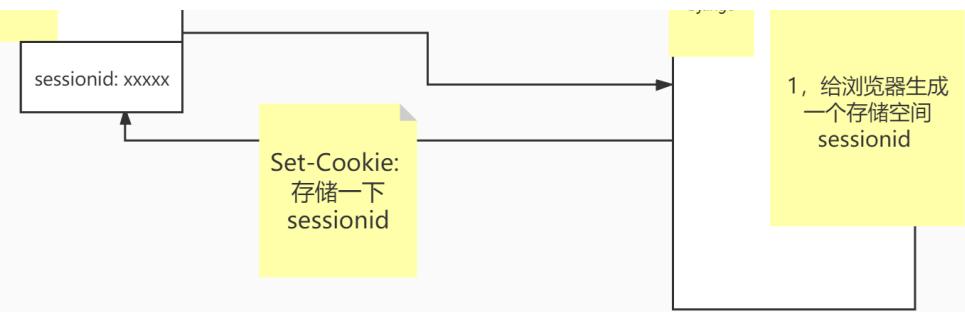
```
1 from redis.sentinel import Sentinel
2
3 #生成哨兵连接
4 sentinel = Sentinel([('localhost', 26379)], socket_timeout=0.1)
5
6 #初始化master连接
7 master = sentinel.master_for('tedu', socket_timeout=0.1, db=1)
8 slave = sentinel.slave_for('tedu', socket_timeout=0.1, db=1)
9
10 #使用redis相关命令
11 master.set('mymaster', 'yes')
12 print(slave.get('mymaster'))
```

```
1 我司 以rdb形式 运行redis 1月有余
2
3 这个月期间，由于采用rdb模式，且遇到了几次服务器宕机【关机】，导致数据发生了少面积丢失
4
5 接到上级指示，我们要将rdb修改成aof
6 直接开启了 redis.conf 配置中的 aof功能 且重启了redis
7
8
9 rdb 和 aof 均开启时，进程启动默认找aof文件 做数据恢复，由于刚启动aof，aof文件为空，启动时以空文件恢复数据，恢复数据时触发rdb，回写rdb
10
```

redis配置

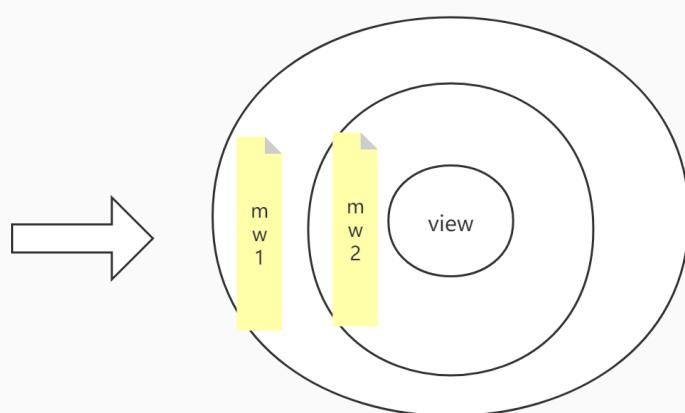
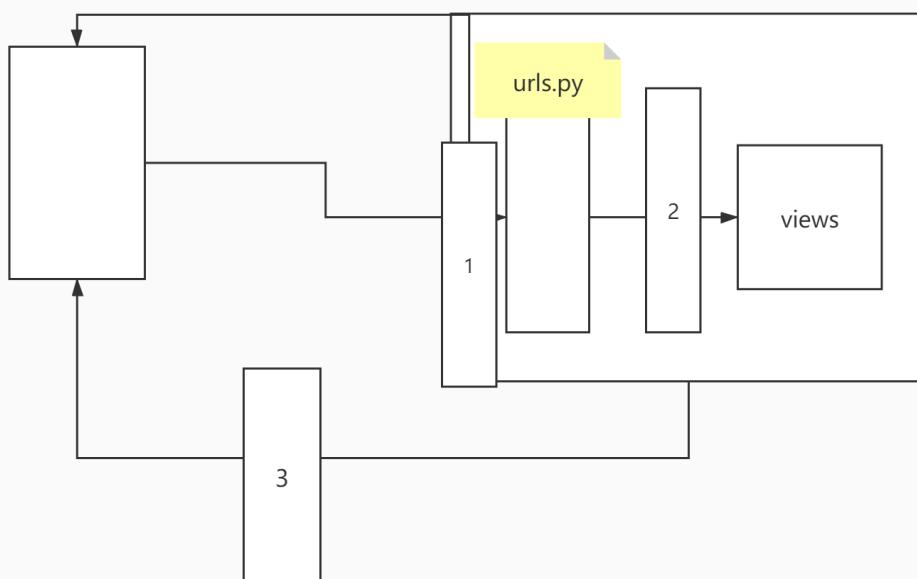
```
1 【1】修改配置文件
2     sudo gedit /etc/redis/redis.conf
3     修改如下2个内容后保存退出：
4         # bind 127.0.0.1 ::1 把此行注释掉
5         protected-mode no      把默认的yes改为no
6
7 【2】重启redis服务
8     sudo /etc/init.d/redis-server restart
9
10 【3】远程连接测试(在远程机器上)
11     redis-cli -h IP地址
```



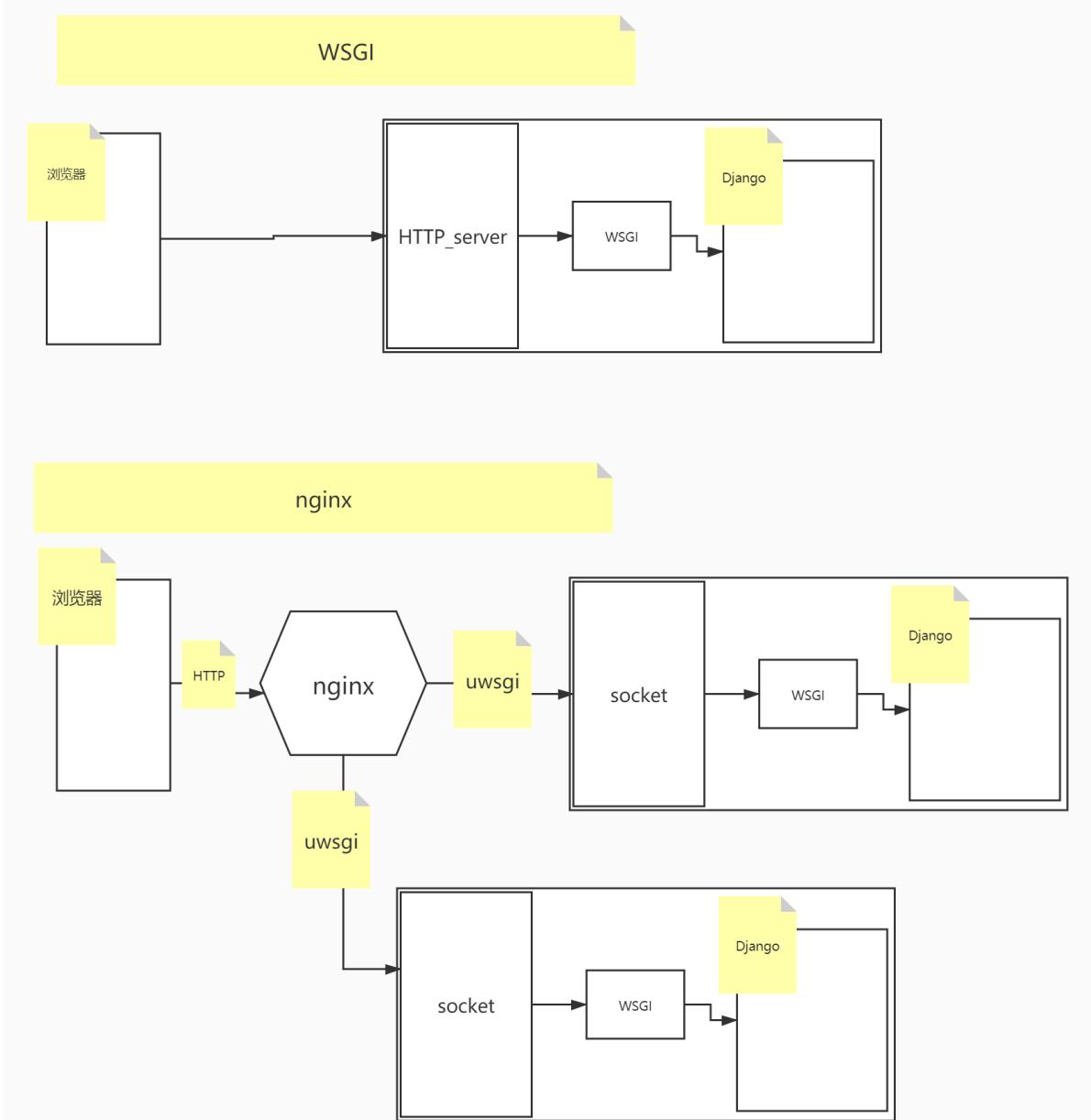
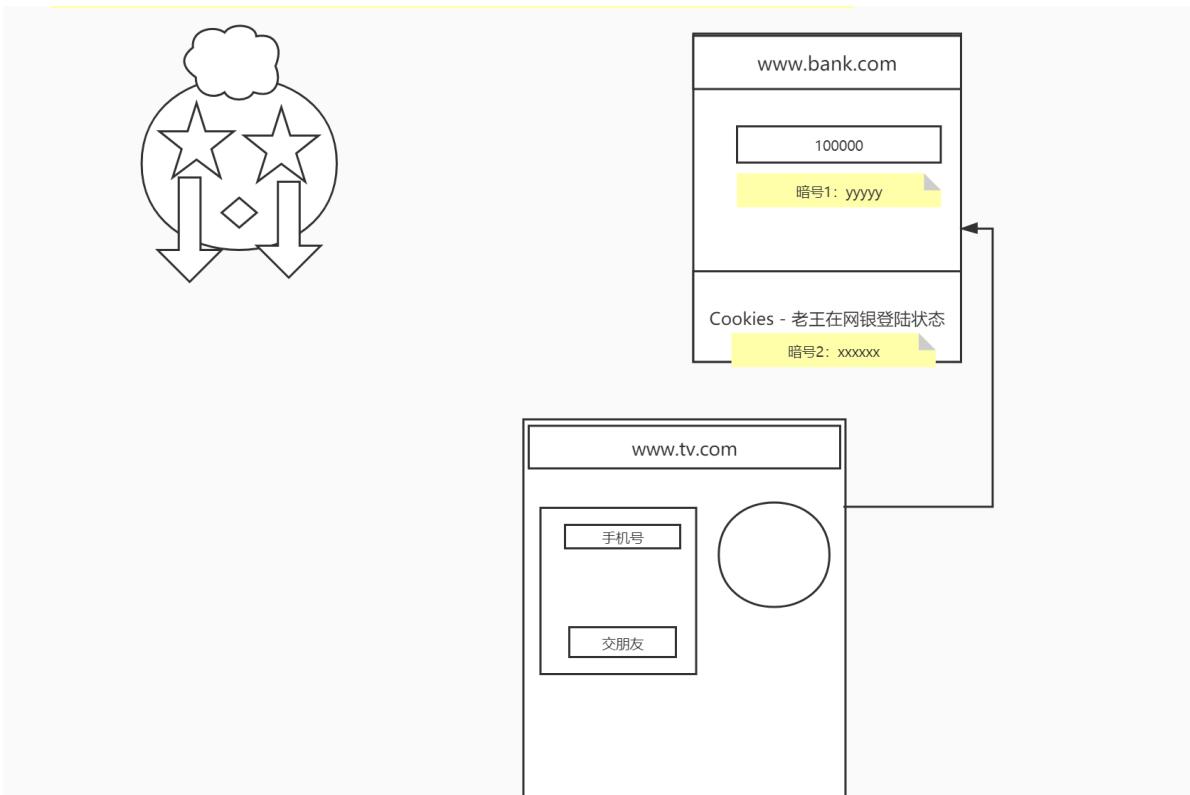


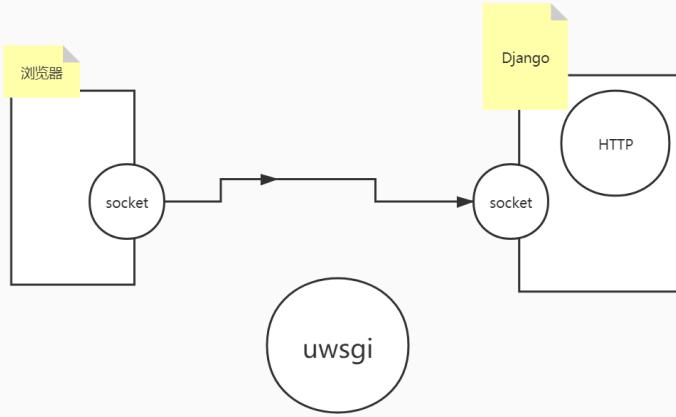
随机sessionid-a:
{username:'guoxiaonao'}
随机sessionid-b: {}

中间件

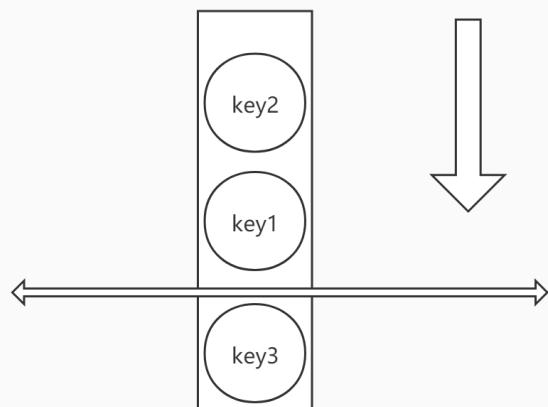
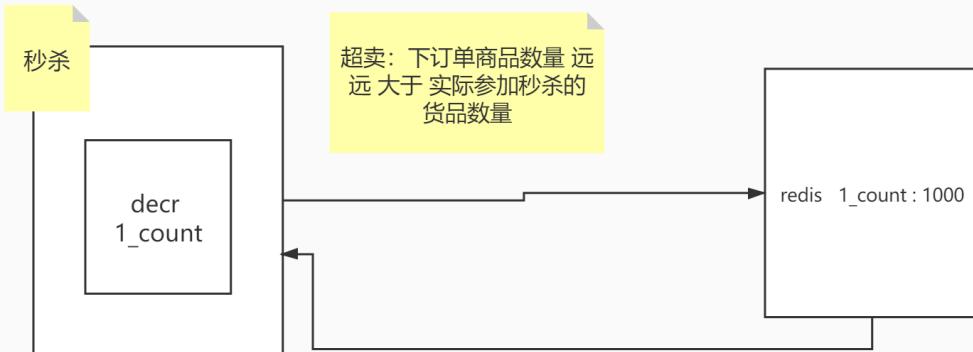


CSRF

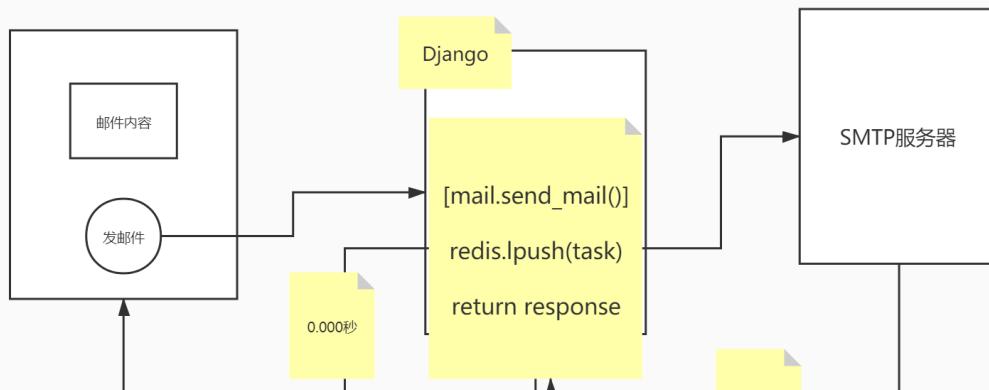


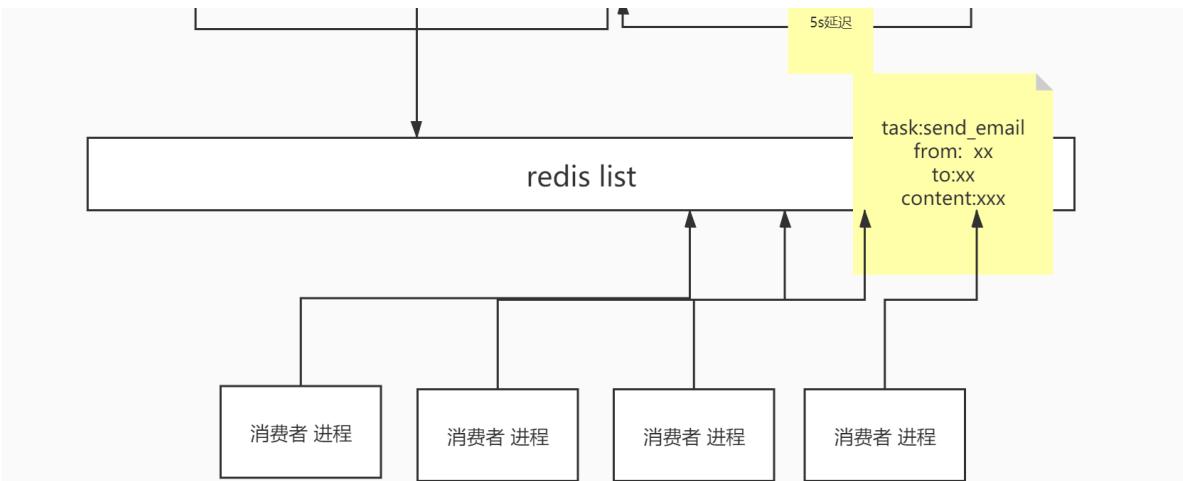


并发计数

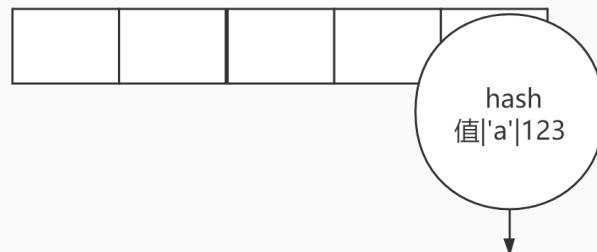


生产者 消费者模型





python + redis hash 的实现



Python

```
d = {} -> 初始化了长度为8的数组
d['a'] = 123
```

```
d['z'] = 123
```

```
hash('a') % 8 = 4
hash('z') % 8 = 4
```

哈希碰撞：

Python 开放地址法

```
hash(4+偏移量) = new_value
```

由于使用开放地址法，py删除 key时，采用伪删除，保留完整的探测链

扩容：已经使用的座位超过三分之二开始扩容 4倍/2倍[>50000]

扩容后 触发 座位重排(rehash) - 旧数组的元素重新计算位置，用新位置存储到新数组中

Redis hash

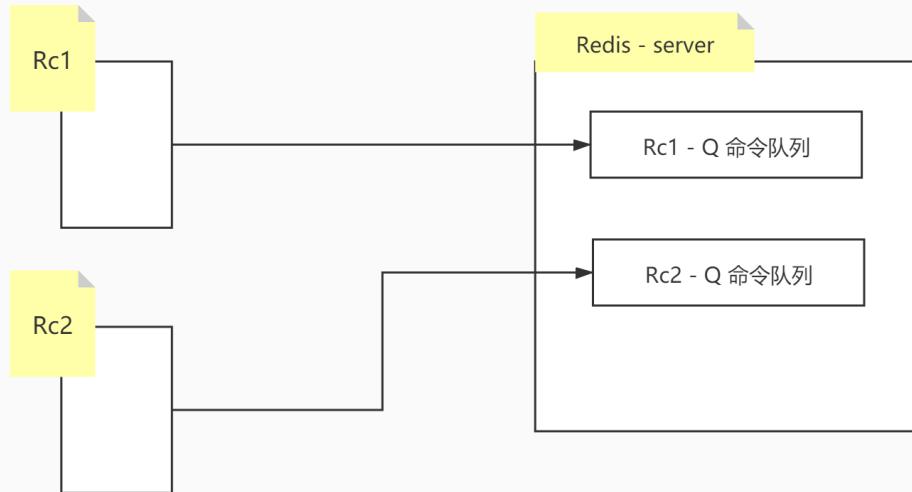
哈希碰撞 - 单链法

扩容 - userd/len(数组) > 1 可能扩 >5 肯定扩

第一个大于 userd * 2 的 2的n次方 以4为例 2的3次方大于4，则使用2的3次方

rehash -> 渐进式rehash

Redis 事务



redis 主从复制

