

黑马点评

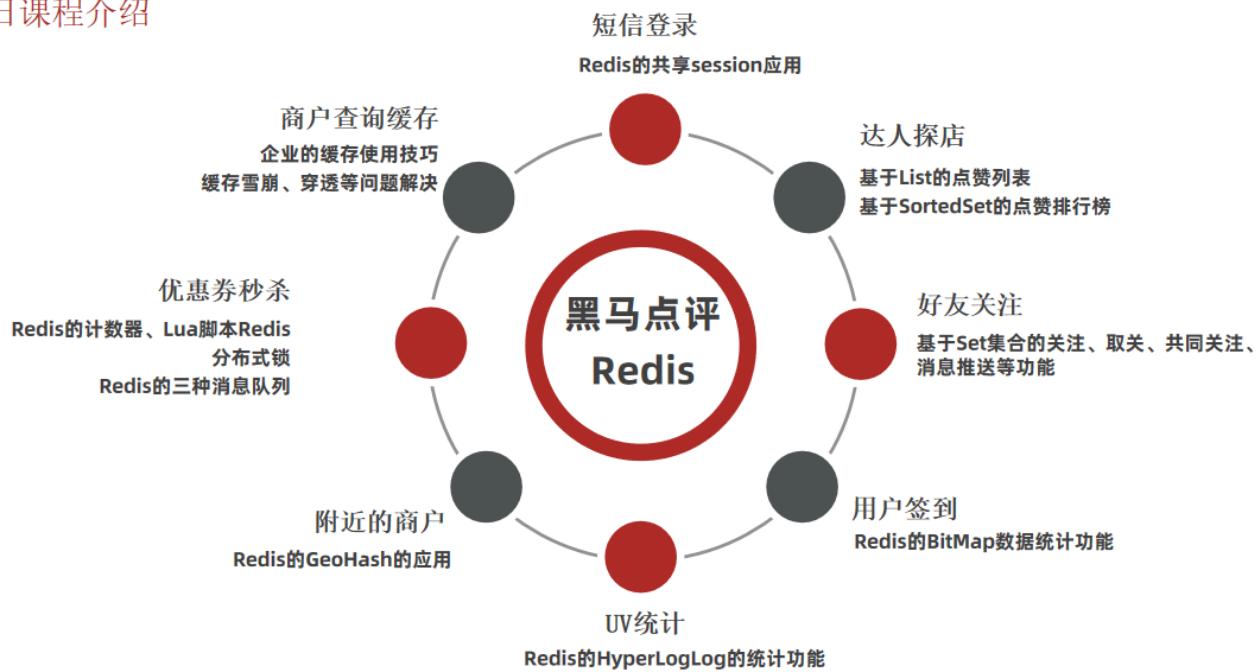
【具体代码笔记查看note.md】

项目介绍

黑马点评是一个大量使用Redis的项目，该项目的功能类似大众点评。

- 短信登录：使用redis共享session来实现
- 商户查询缓存：理解缓存击穿，缓存穿透，缓存雪崩等问题
- 优惠券秒杀：Redis的计数器功能，结合Lua完成高性能的redis操作，同时学会Redis分布式锁的原理，包括Redis的三种消息队列
- 打人探店：基于List来完成点赞列表的操作，同时基于SortedSet来完成点赞的排行榜功能
- 好友关注：基于Set集合的关注、取消关注，共同关注等等功能
- 附近的商户：利用Redis的GEOHash来完成对于地理坐标的操作
- 用户签到：使用Redis的BitMap数据统计功能
- UV统计：使用Redis来完成统计功能

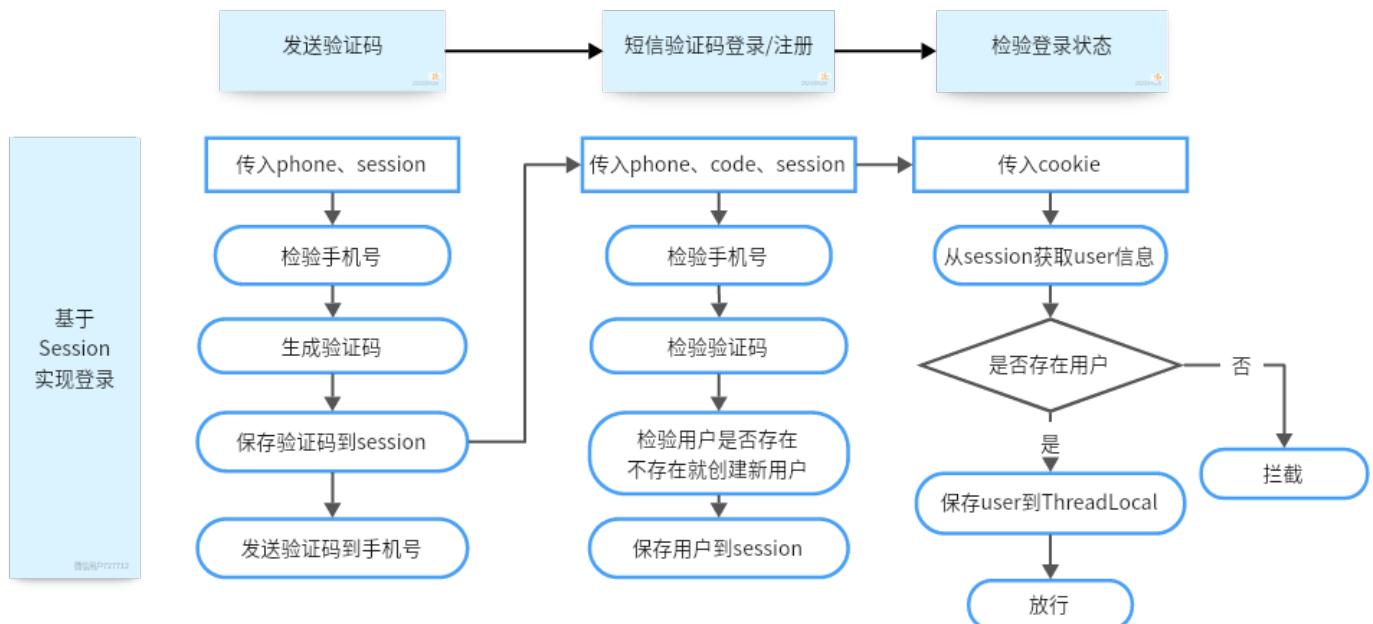
今日课程介绍



短信登陆

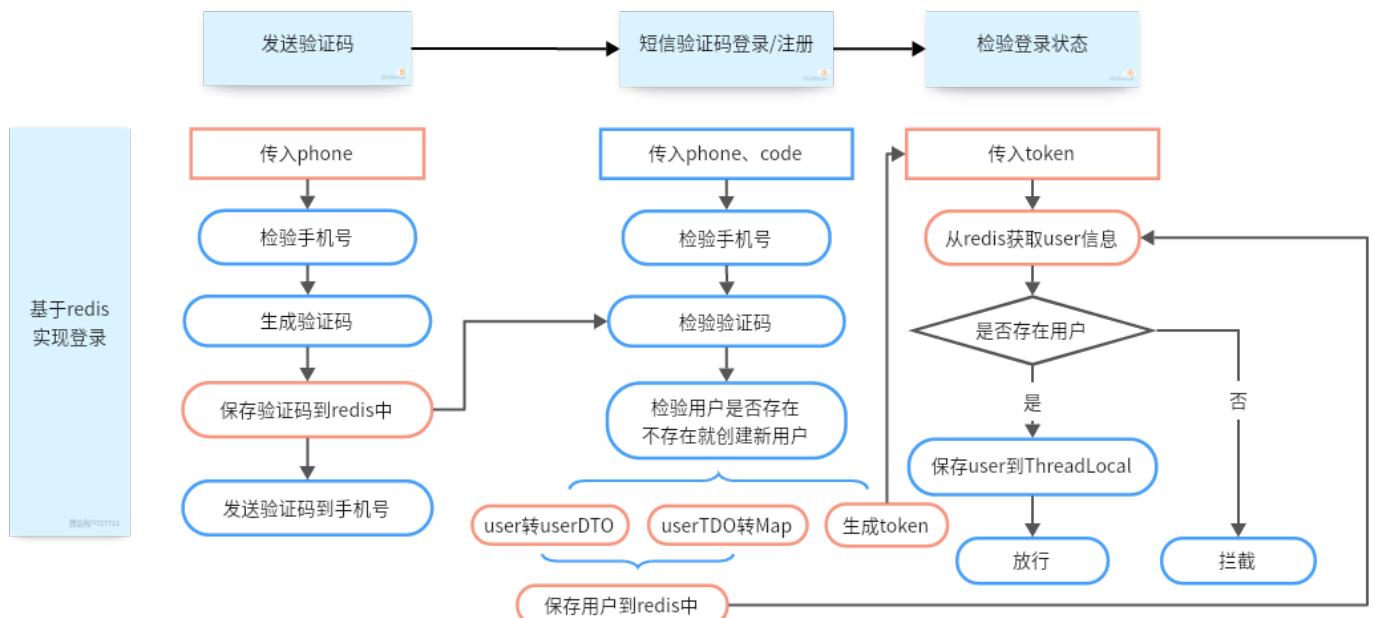
基于Session实现登录流程

实现逻辑：



基于Redis实现共享session登录流程

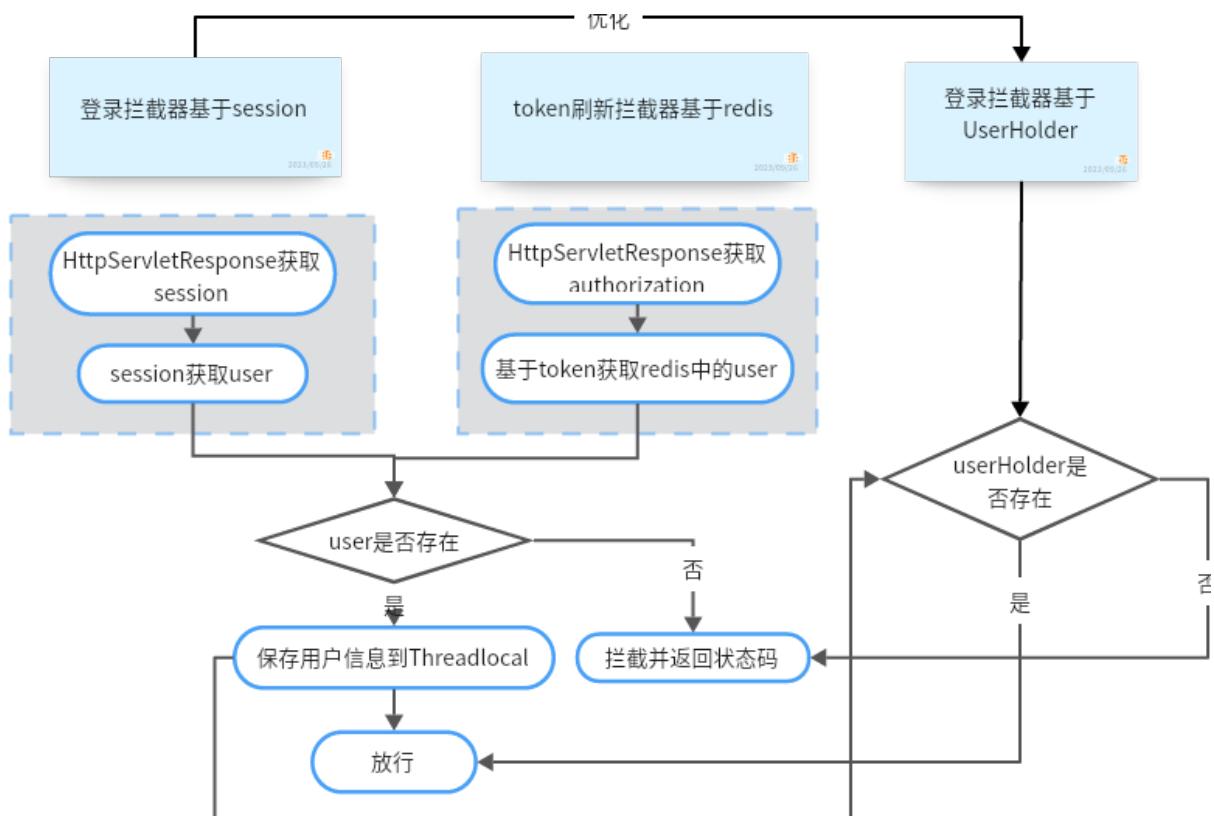
实现逻辑：橙色为修改部分



拦截器及优化

实现逻辑：

拦截器及优化

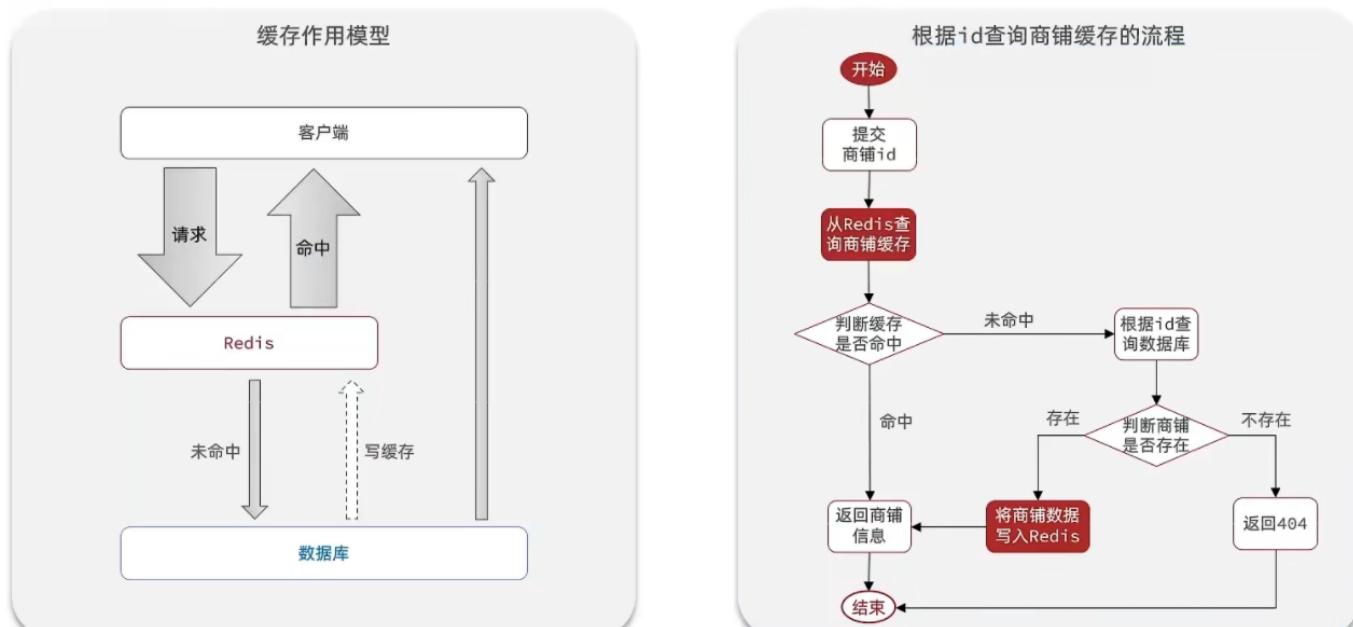


商户查询缓存

添加redis

实现逻辑：

添加Redis缓存

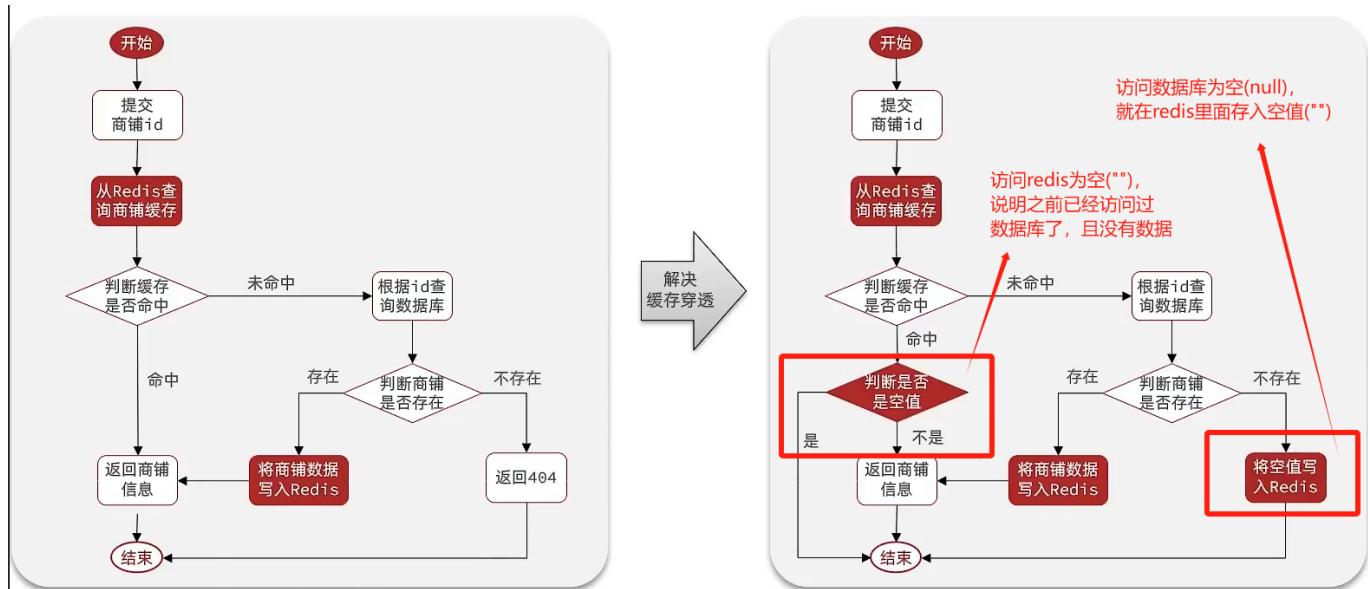


缓存穿透

问题出现：

是指客户端请求的数据在缓存中和数据库中都不存在，这样缓存永远不会生效，这些请求都会打到数据库。

解决逻辑：



缓存雪崩

问题出现：

同一时段大量的缓存key同时失效或者Redis服务宕机，导致大量请求到达数据库，带来巨大压力。

解决逻辑：

- 给不同的Key的TTL添加随机值（key失效）
- 给业务添加多级缓存（key失效）
- 利用Redis集群提高服务的可用性（redis宕机）
- 给缓存业务添加降级限流策略（redis宕机）
-

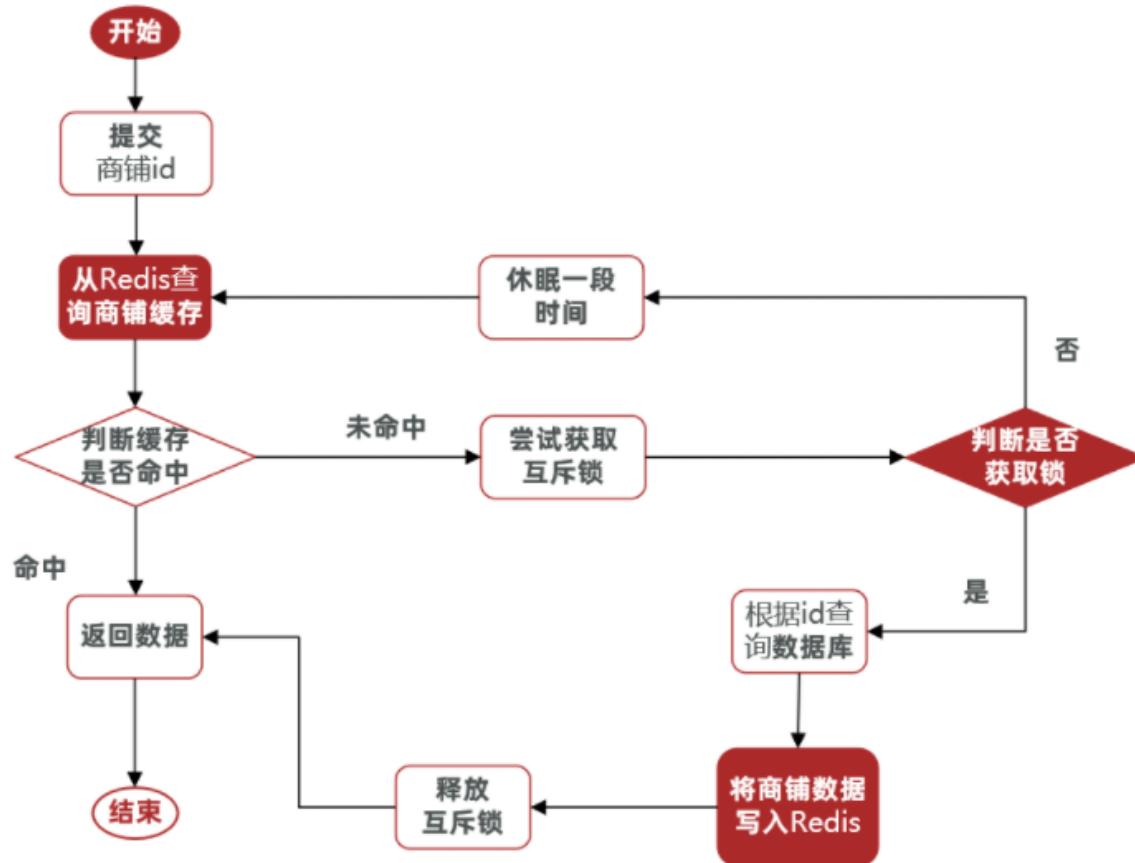
缓存击穿

问题出现：

一个被高并发访问并且缓存重建业务较复杂的key突然失效了，无数的请求访问会在瞬间给数据库带来巨大的冲击

解决逻辑：

需求：修改根据id查询商铺的业务，基于互斥锁的方式来解决缓存击穿问题

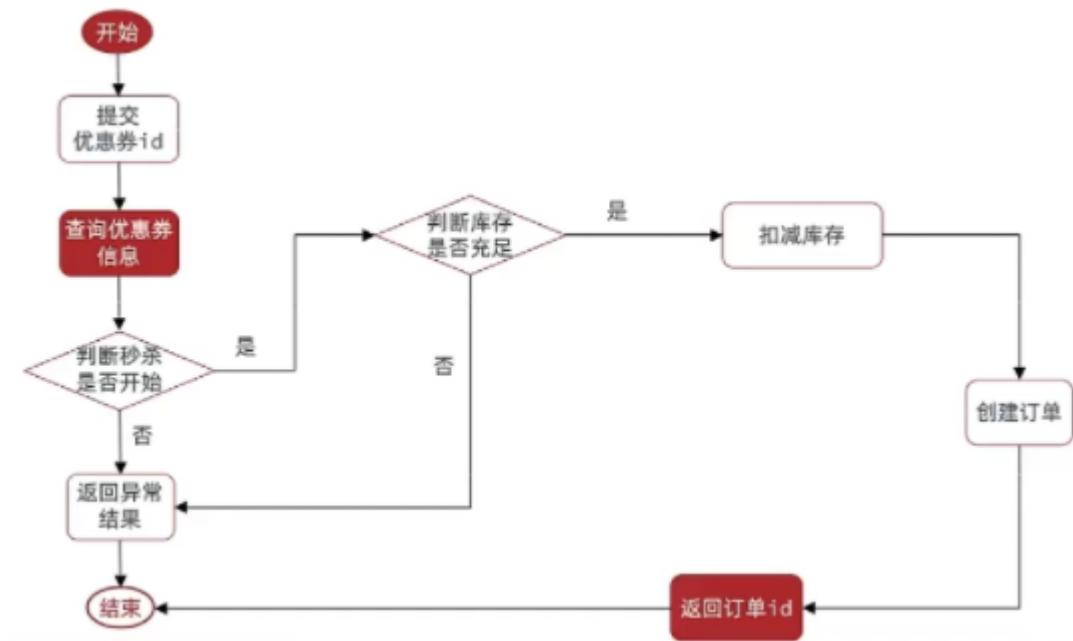


优惠券秒杀

生成全局唯一ID

实现秒杀下单

实现逻辑：

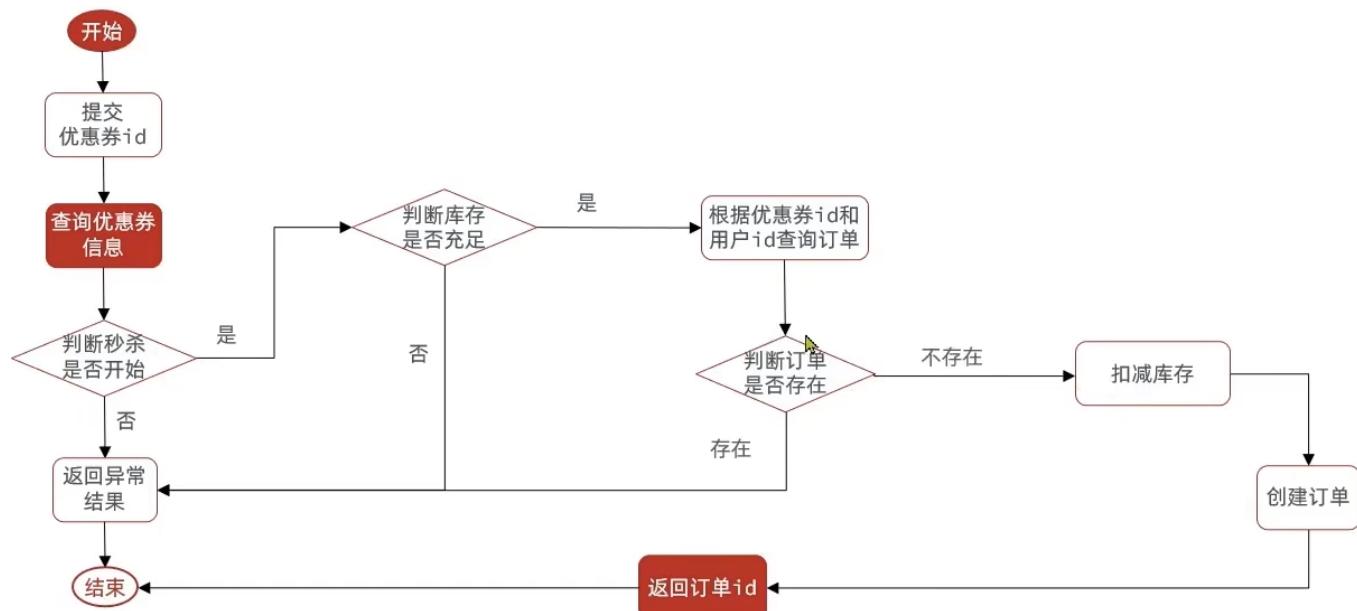


超卖问题

解决逻辑：乐观锁判断版本号

```
boolean success = seckillVoucherService.update()
    .setSql("stock= stock -1")
    .eq("voucher_id", voucherId).update().gt("stock",0); //where id = ?
and stock > 0
```

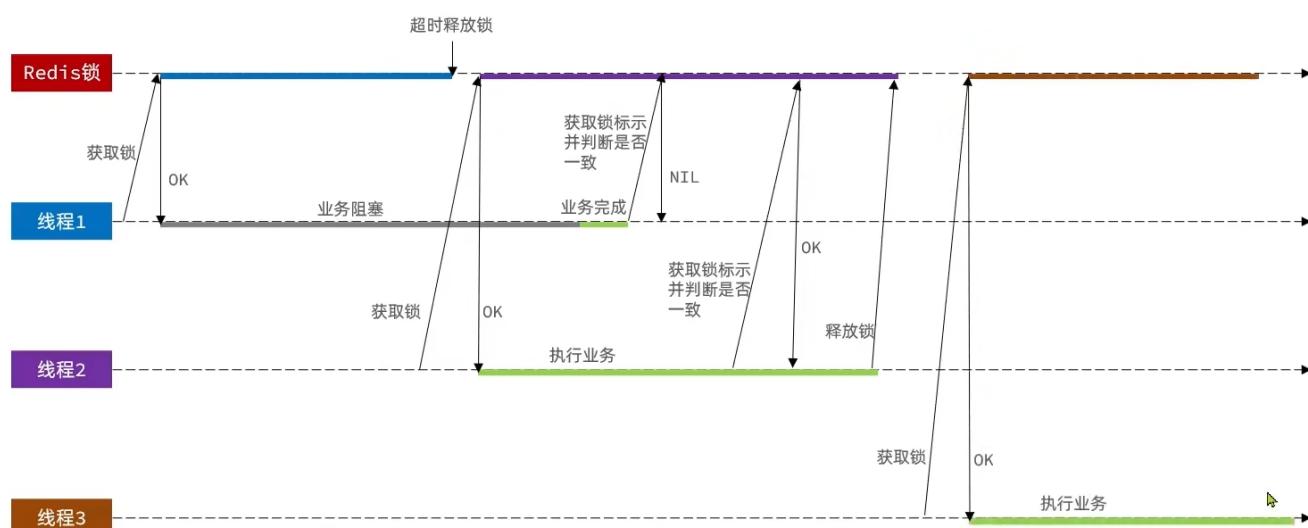
一人一单



分布式锁

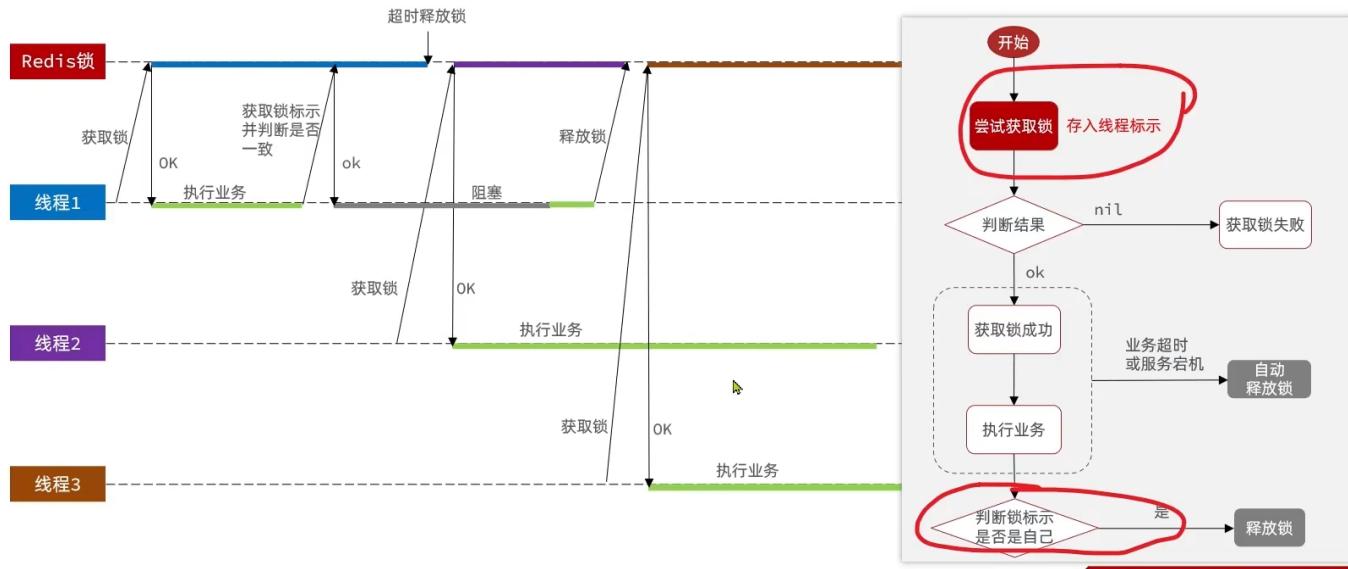
误删问题

问题出现：



解决逻辑：在获取锁时存入线程标示（可以用UUID表示），一致放锁，不一致不放锁

原子性问题



解决思路：Lua脚本解决多条命令原子性问题

redisson

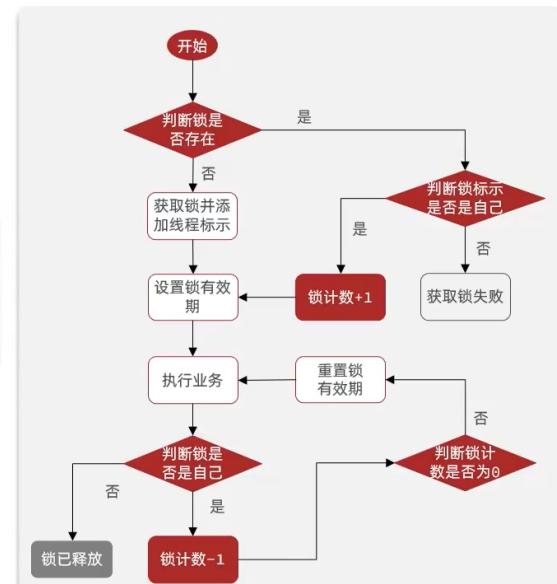
可重入锁原理

```

// 创建锁对象
RLock lock = redissonClient.getLock("lock");

@Test
void method1() {
    boolean isLock = lock.tryLock();
    if(!isLock){
        log.error("获取锁失败, 1");
        return;
    }
    try {
        log.info("获取锁成功, 1");
        method2();
    } finally {
        log.info("释放锁, 1");
        lock.unlock();
    }
}
void method2(){
    boolean isLock = lock.tryLock();
    if(!isLock){
        log.error("获取锁失败, 2");
        return;
    }
    try {
        log.info("获取锁成功, 2");
    } finally {
        log.info("释放锁, 2");
        lock.unlock();
    }
}
  
```

KEY	VALUE	
	field	value
lock	thread1	1



lua脚本实现可重入锁

Redisson可重入锁原理

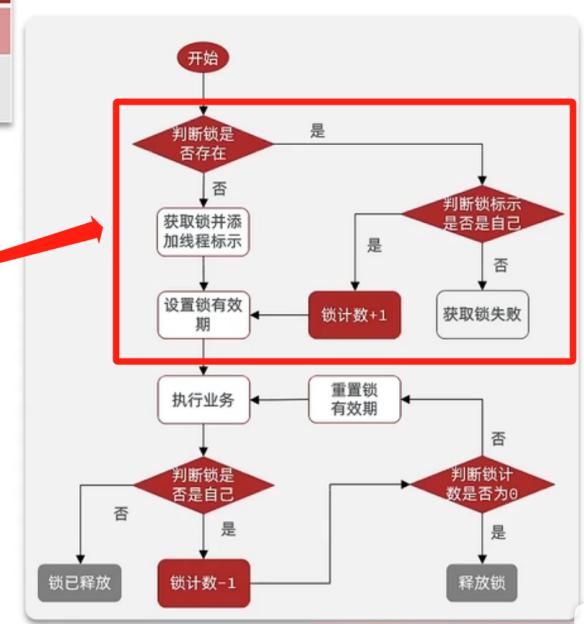
获取锁的Lua脚本：

```

local key = KEYS[1]; -- 锁的key
local threadId = ARGV[1]; -- 线程唯一标识
local releaseTime = ARGV[2]; -- 锁的自动释放时间
-- 判断是否存在
if(redis.call('exists', key) == 0) then
    -- 不存在, 获取锁
    redis.call('hset', key, threadId, '1');
    -- 设置有效期
    redis.call('expire', key, releaseTime);
    return 1; -- 返回结果
end;
-- 锁已经存在, 判断threadId是否是自己
if(redis.call('hexists', key, threadId) == 1) then
    -- 不存在, 获取锁, 重入次数+1
    redis.call('hincrby', key, threadId, '1');
    -- 设置有效期
    redis.call('expire', key, releaseTime);
    return 1; -- 返回结果
end;
return 0; -- 代码走到这里, 说明获取锁的不是自己, 获取锁失败

```

KEY	VALUE	
	field	value
lock	thread1	0



Redisson可重入锁原理

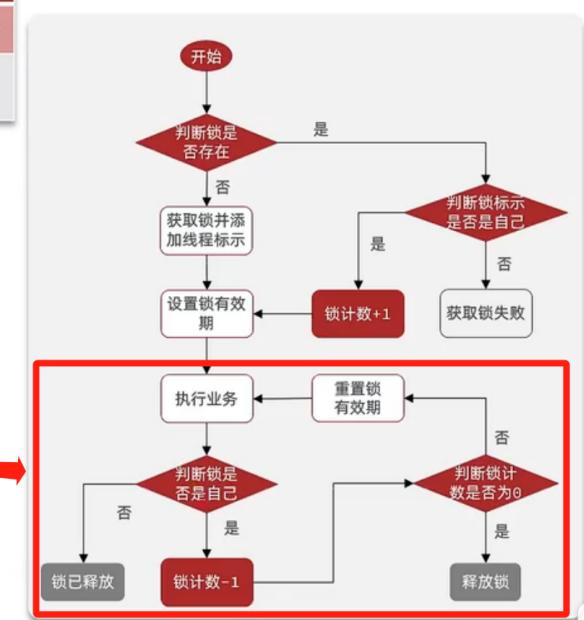
释放锁的Lua脚本：

```

local key = KEYS[1]; -- 锁的key
local threadId = ARGV[1]; -- 线程唯一标识
local releaseTime = ARGV[2]; -- 锁的自动释放时间
-- 判断当前锁是否还是被自己持有
if (redis.call('HEXISTS', key, threadId) == 0) then
    return nil; -- 如果已经不是自己, 则直接返回
end;
-- 是自己的锁, 则重入次数-1
local count = redis.call('HINCRBY', key, threadId, -1);
-- 判断是否重入次数是否已经为0
if (count > 0) then
    -- 大于0说明不能释放锁, 重置有效期然后返回
    redis.call('EXPIRE', key, releaseTime);
    return nil;
else -- 等于0说明可以释放锁, 直接删除
    redis.call('DEL', key);
    return nil;
end;

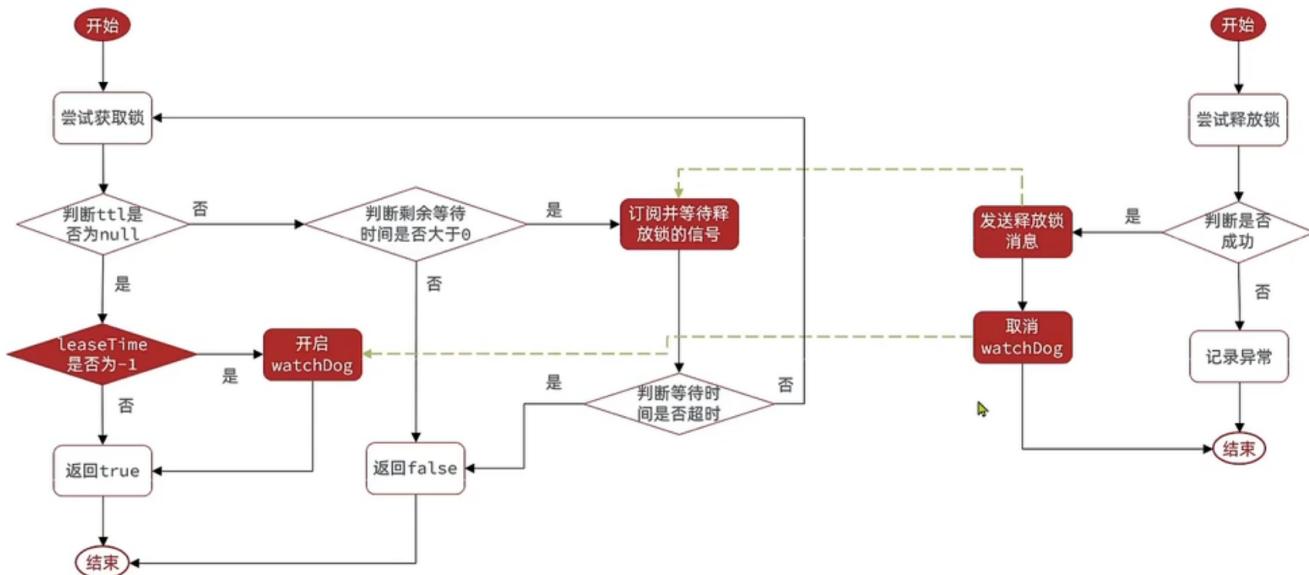
```

KEY	VALUE	
	field	value
lock:order	thread1	0



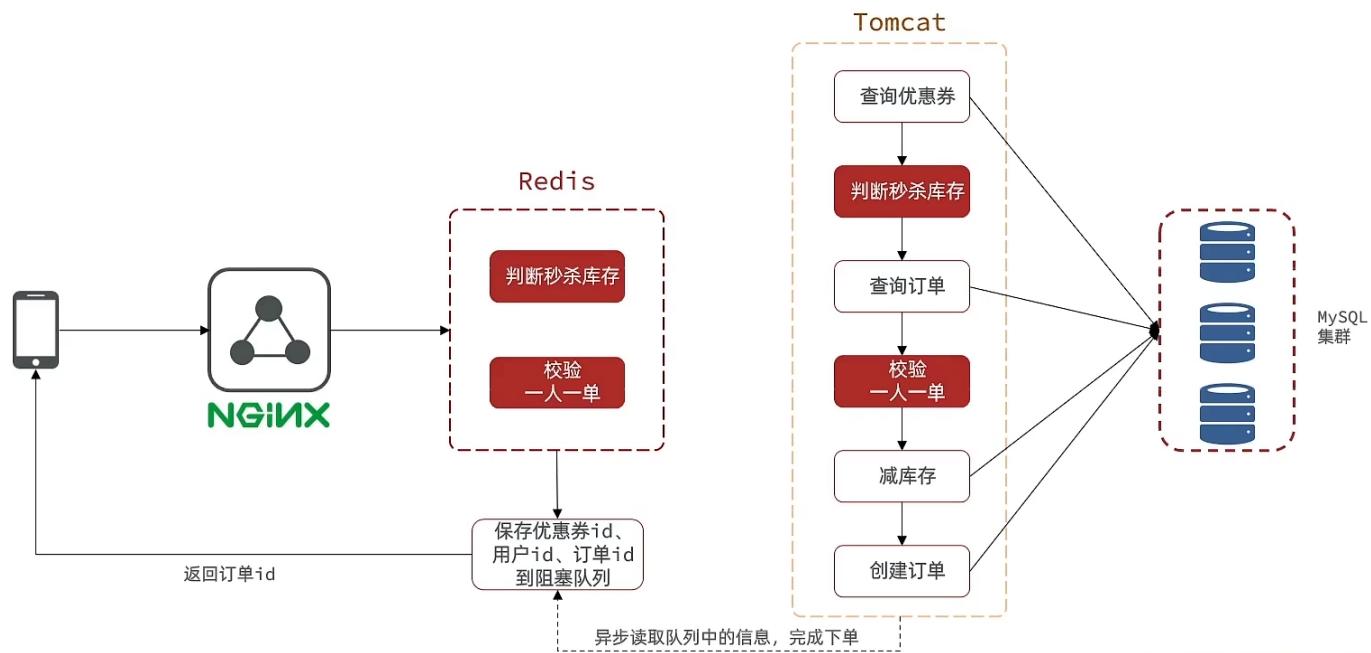
锁重试和WatchDog机制

Redisson分布式锁原理



优化秒杀

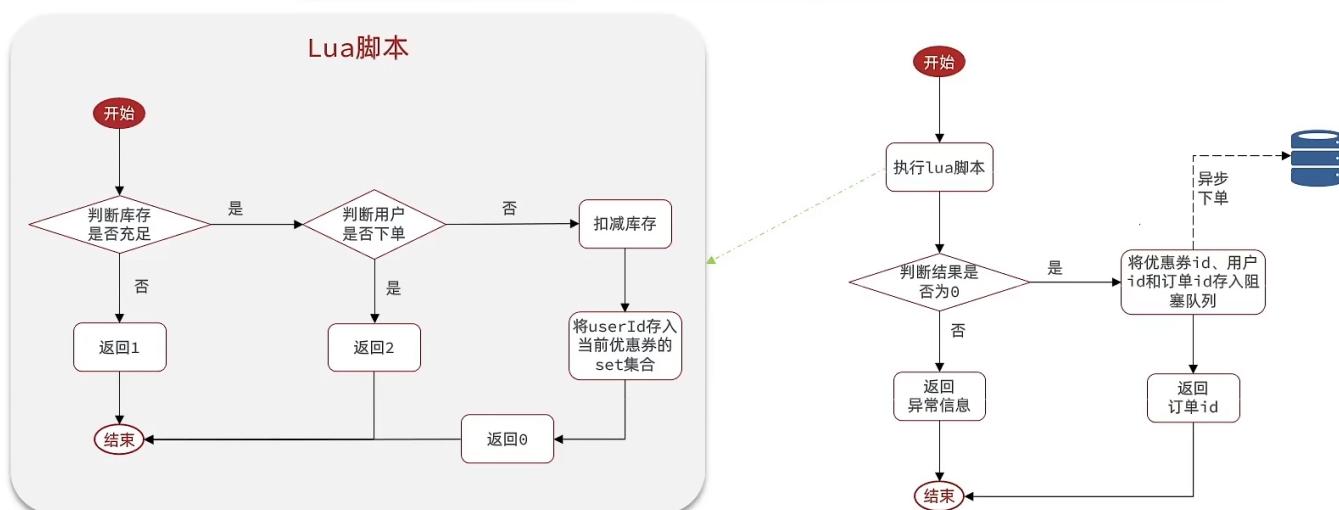
实现逻辑：



lua保证原子性

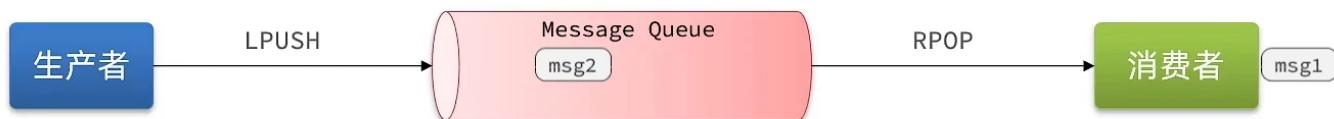
Redis优化秒杀

KEY	VALUE	KEY	VALUE
stock:vid:7	100	order:vid:7	1 , 2 , 3 , 5 , 7 , 8

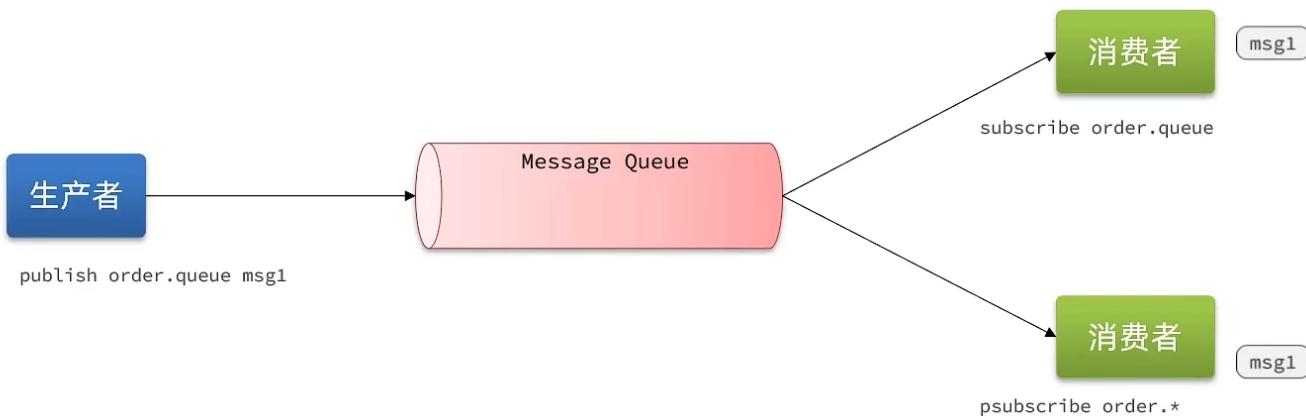


消息队列

基于List实现消息队列



基于PubSub的消息队列



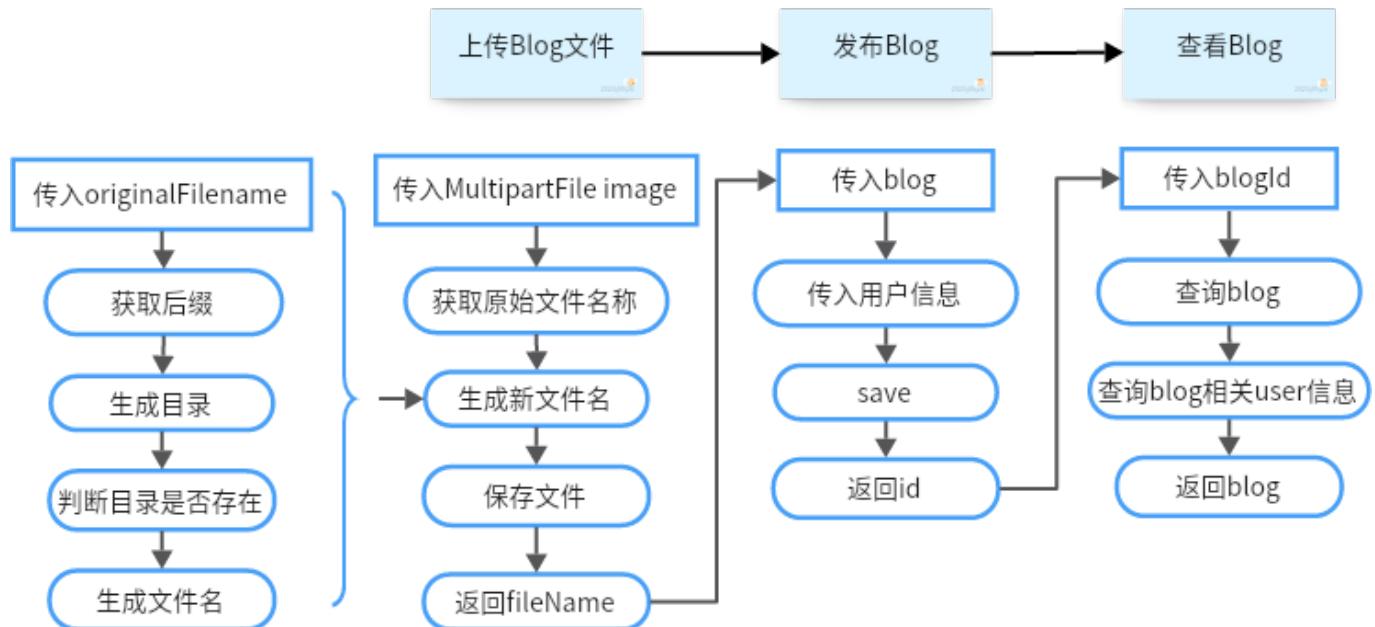
Redis的Stream消息队列实现异步秒杀



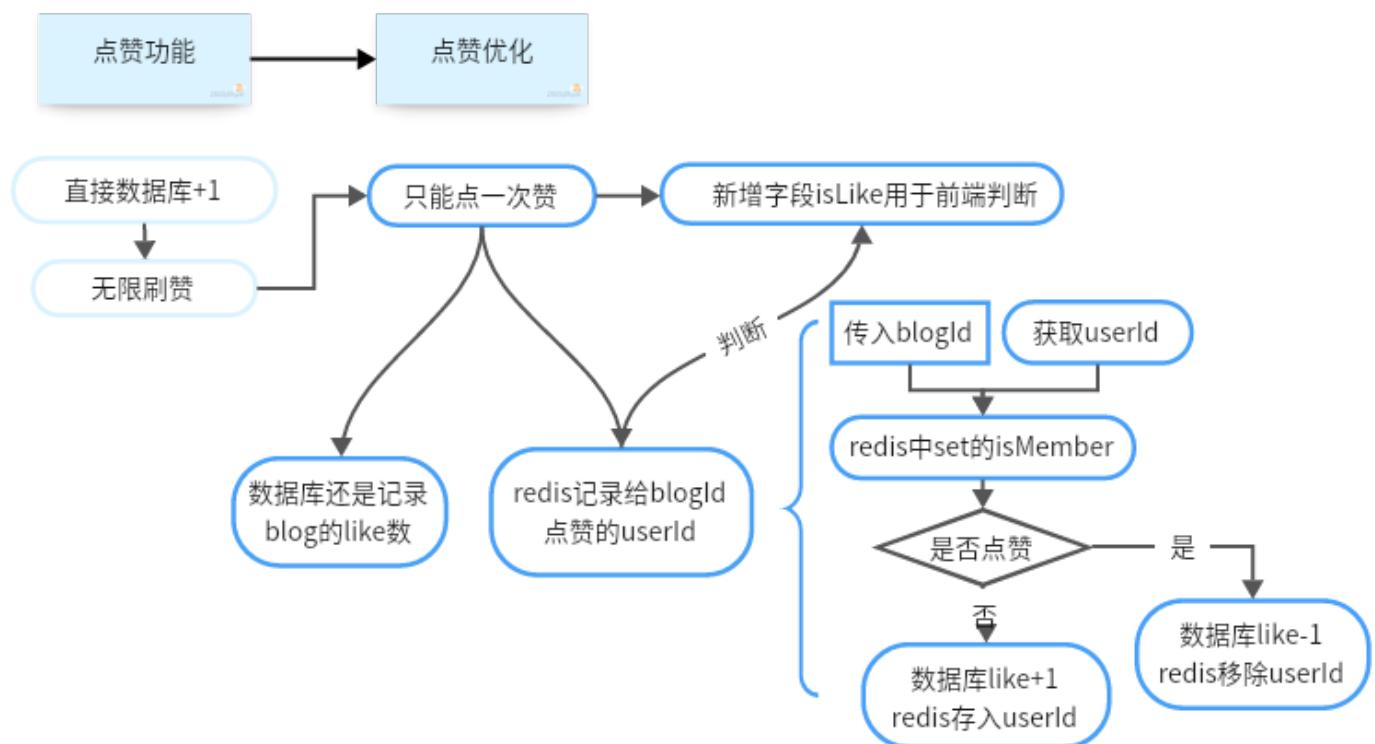
```
1 while(true){  
2     // 尝试监听队列，使用阻塞模式，最长等待 2000 毫秒  
3     Object msg = redis.call("XREADGROUP GROUP g1 c1 COUNT 1 BLOCK 2000 STREAMS s1 >");  
4     if(msg == null){ // null说明没有消息，继续下一次  
5         continue;  
6     }  
7     try {  
8         // 处理消息，完成后一定要ACK  
9         handleMessage(msg);  
10    } catch(Exception e){  
11        while(true){  
12            Object msg = redis.call("XREADGROUP GROUP g1 c1 COUNT 1 STREAMS s1 0");  
13            if(msg == null){ // null说明没有异常消息，所有消息都已确认，结束循环  
14                break;  
15            }  
16            try {  
17                // 说明有异常消息，再次处理  
18                handleMessage(msg);  
19            } catch(Exception e){  
20                // 再次出现异常，记录日志，继续循环  
21                continue;  
22            }  
23        }  
24    }  
25 }
```

达人探店

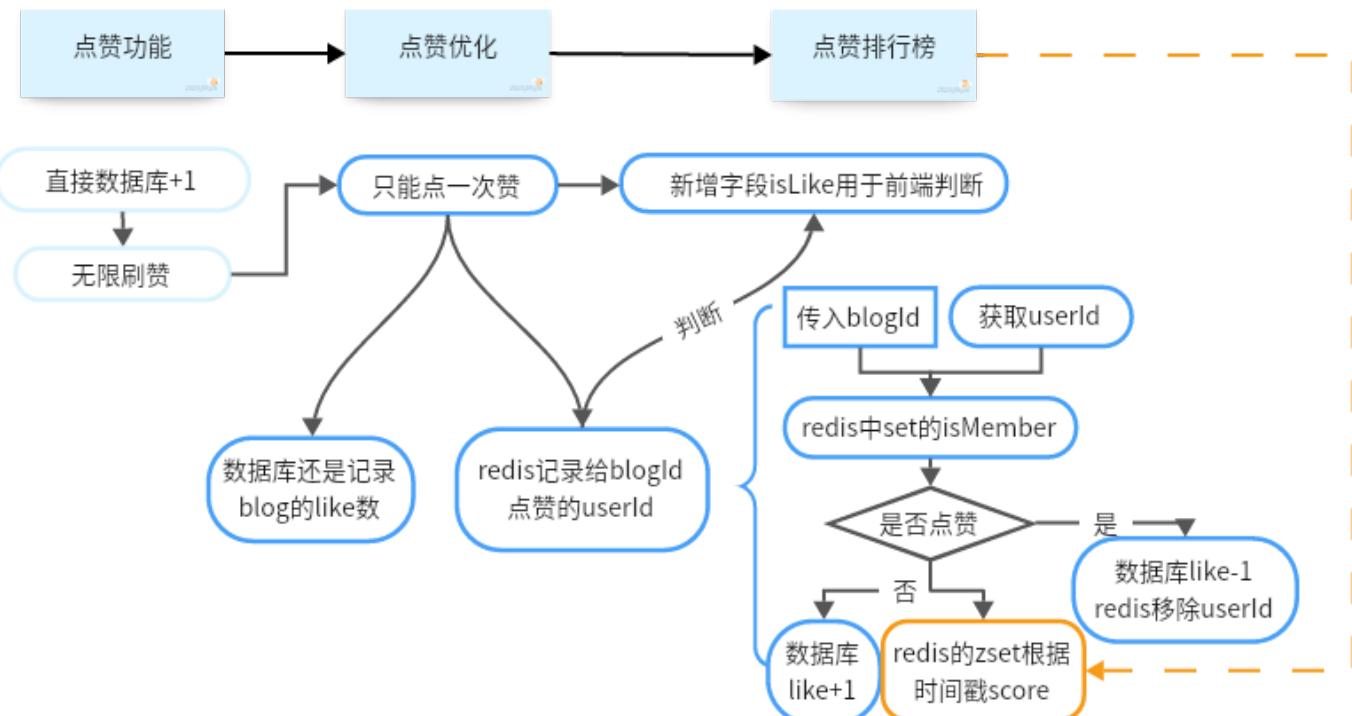
发布、查看探店笔记



点赞

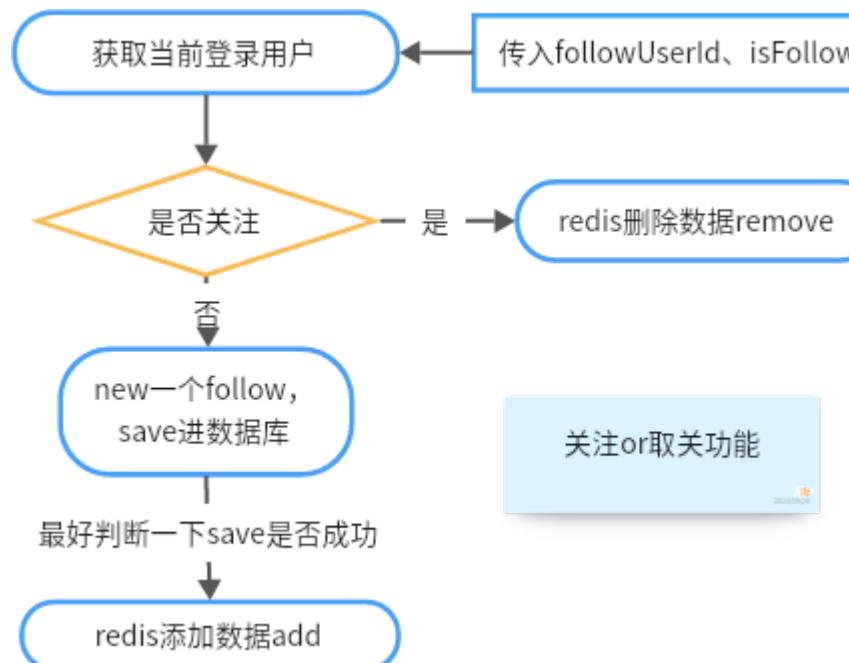


点赞排行榜



好友关注

实现关注/取关



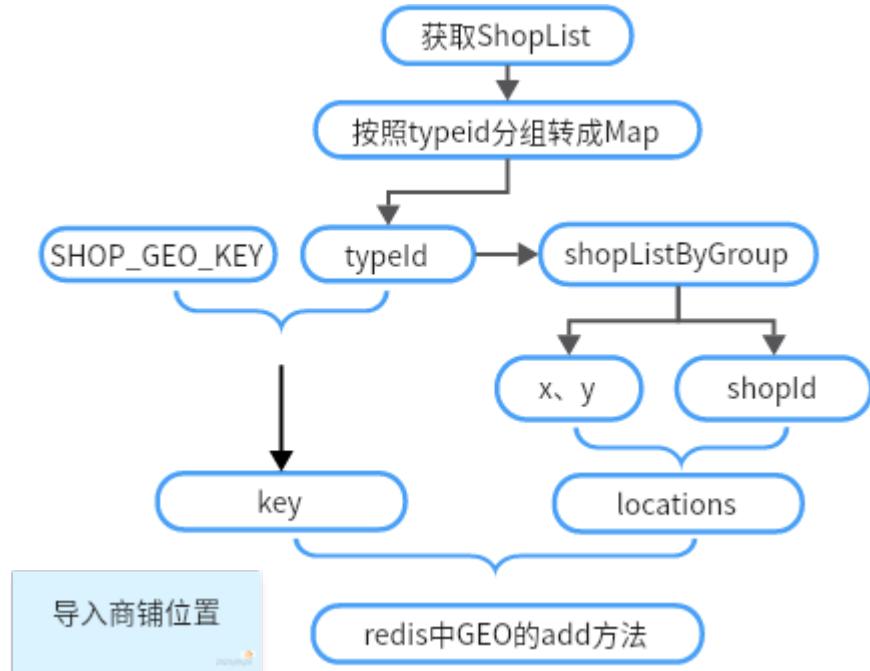
查看共同关注

set的**intersect**方法查询交集，传入两个key查找交集value

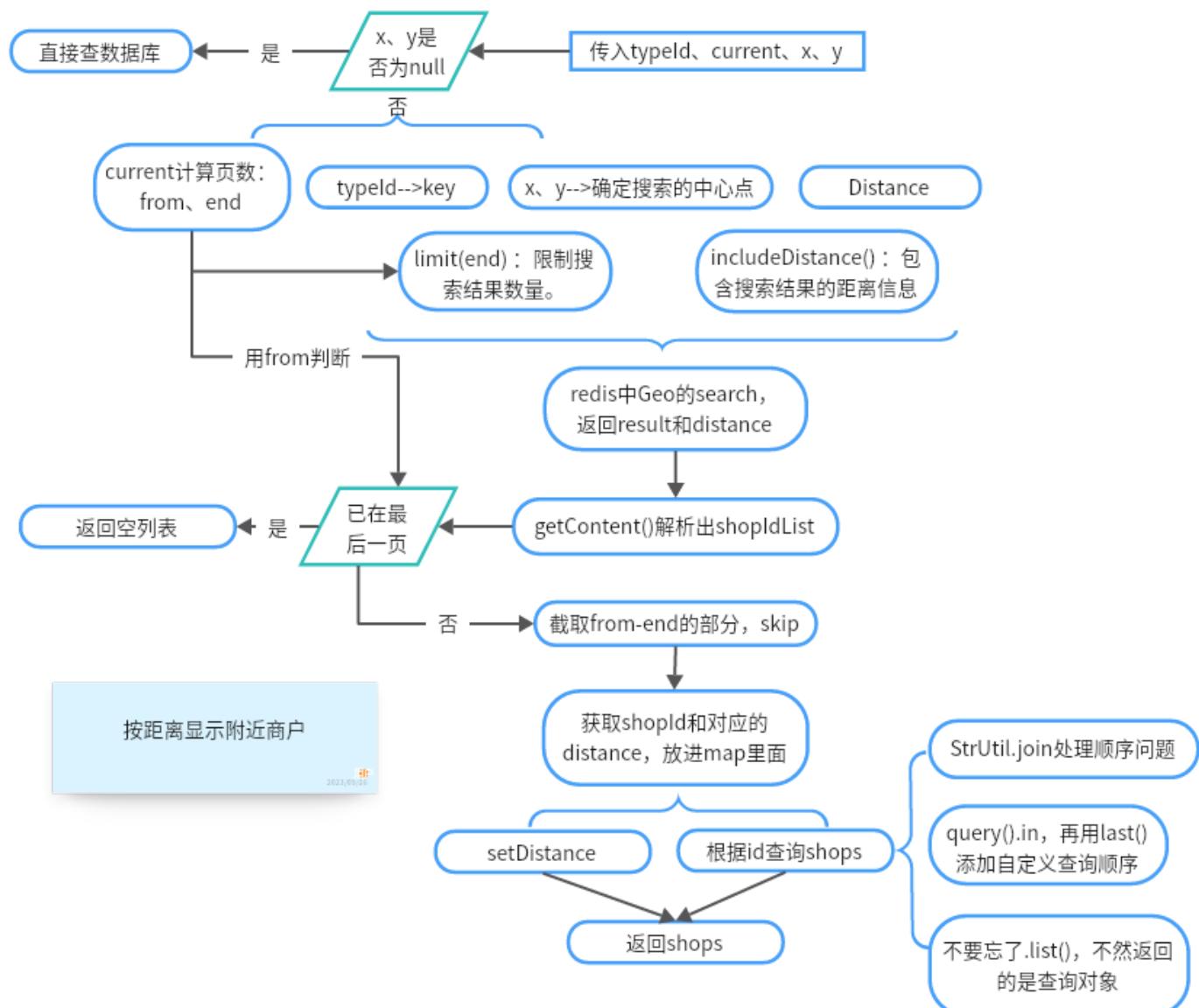
```
Set<String> intersect = stringRedisTemplate.opsForSet().intersect(userKey, otherKey);
```

附近商户

导入商铺位置

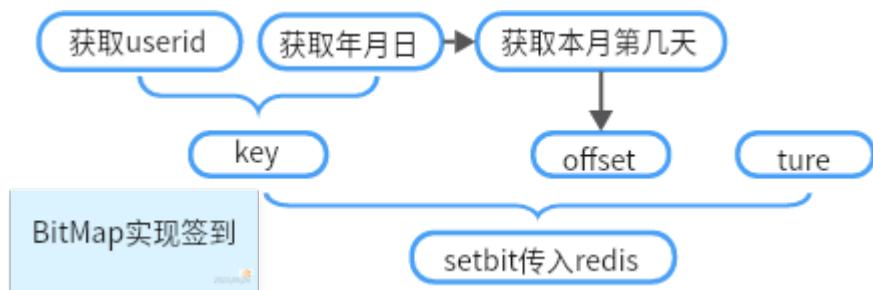


附近商户



用户签到

签到



统计签到次数



UV统计

实现逻辑:

```
stringRedisTemplate.opsForHyperLogLog().add("testHyperLogLog", values);
```