# 促销数据的Redis缓存设计

## 原数据结构：

1. Hash散列：命令hSet

hset key field value

field 不存在，直接存储。

field 存在，执行覆盖操作。

Key：

air\_base\_prom\_activity\_sku

Filed:

venderId\_flightCode\_clazzCode,如："621906\_FM9531\_B"

Value：

{

"activityId": 1800,

"clazzCode": "B",

"clazzPrice": 300,

"createDate": 1497602389000,

"flightCode": "FM9531",

"id": 11609147,

"modifiedDate": 1497602389000,

"promotionBeans": 0,

"promotionPrice": 280,

"reducePrice": 20,

"rfId": 4886089506,

"skuId": 200105677302,

"status": "1",

"userType": "0",

"venderId": 621906,

"cacheFiled": "621906\_FM9531\_B"

}

1. 问题：

Key下面的subKey太多，可能有100万左右。这样设计不合理

## 新数据结构

将key打散，存储成Object关联到key，底层实现原理是set(key, JSON.toJSONString(value)),

这样key下面value值少，方便get

1. String：命令Set

Set(key，value)

Key：air\_base\_rule\_prom\_act\_sku\_ venderId\_flightCode\_clazzCode

Value：List<ActivitySKU>的JSON串

{

"activityId": 1800,

"id": 11609147,

"Beans": 0,

"reducePrice": 20,

"Type": "0",

"cacheKey": "621906\_FM9531\_B"

}

## 写缓存

1. 消费MQ消息时，分页反查京米数据，数据组装并插入数据库，每页数据200条组成一个List<ActivitySKU>，将其写入阻塞队列，队列大小20

BlockingQueue<List<ActivitySKU>> activitySKUQueue = new ArrayBlockingQueue<List<ActivitySKU>>(20);

1. 使用单线程的线程池去执行任务，任务是从阻塞队列中取出list，然后开启多线程去写缓存，包括提供前端的缓存、关联缓存、航司\_航线3个

多线程有8个线程，队列大小2048，拒绝策略是主线程执行

1. 商家\_航司下的vender\_flight\_clazz信息刷新缓存，使用set结构

Key：621906\_CA

Value: 621906\_CA3566\_Y

1. 关联缓存，set结构

Key：air\_base\_rule\_prom\_act\_sku\_relation

Value：621906\_CA

1. 提供给前端的缓存，普通String

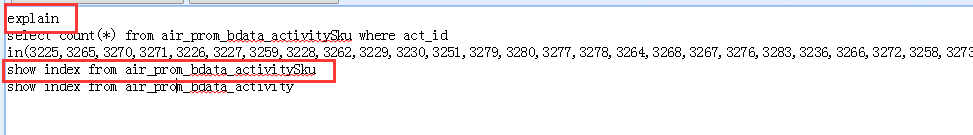
Key：air\_base\_rule\_prom\_act\_sku\_ venderId\_flightCode\_clazzCode

Value：List<ActivitySKU>的JSON串

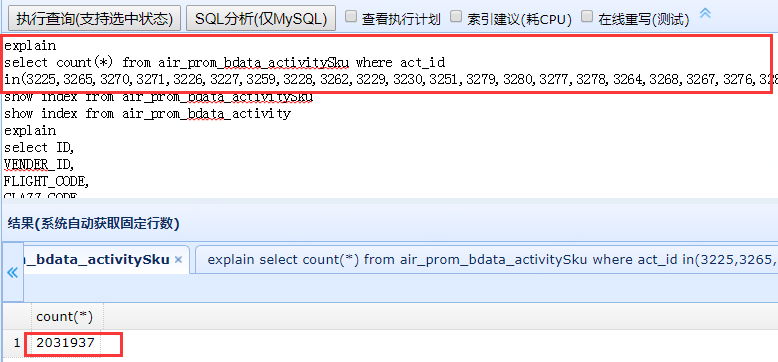
1. 航线下活动的缓存（新加，机火引流用）

Key：air\_base\_rule\_prom\_airline\_act\_PEK\_SHA

Value: actId

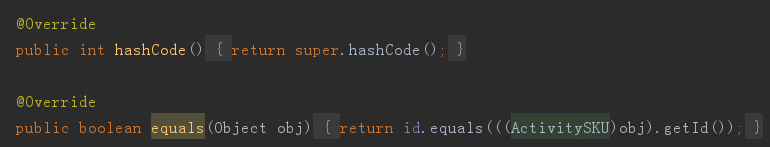


有效活动的数量



## 去重：

重写ActivitySKU对象的equals()方法和hashCode()方法。



## 删缓存

1. 查询前一天失效的活动，根据活动id删除该活动下的缓存

<if test="endTime != null and endTime != ''">  
 AND END\_TIME &gt;= #{endTime}-86400000 AND END\_TIME &lt;= #{endTime}  
</if>

1. 查询status=4且修改时间是前一天的活动，根据活动id删除该活动下的缓存。与1是or的关系

<if test="statusEnum != null and statusEnum != '' and endTime != null and endTime != ''">  
 OR (ACT\_STATUS = #{statusEnum,typeHandler=com.jd.airplane.base.dao.handler.ActivityStatusEnumCodeHandler}  
 AND unix\_timestamp(MODIFIED)\*1000 &gt;= #{endTime}-86400000 AND unix\_timestamp(MODIFIED)\*1000 &lt;= #{endTime})  
</if>

删缓存逻辑：

1. 通过活动去activitySku表中查询数据拼接key，通过key从redis中get到

Value

1. Value是List<ActivitySKU>的JSON串，将其反序列化成list，去list中找当前activitySKU，然后remove
2. Remove后，若list为空则删除这个key（提供给前端的），且删除商家\_航司下的vender\_flight\_clazz缓存，若不为空，则将处理后的list继续set到redis中
3. 判断商家\_航司下的vender\_flight\_clazz为空则，删除（由于sMembers耗时长，先去掉）

Key: air\_base\_rule\_prom\_act\_sku\_relation

Value: 商家\_航司

1. 删除航线下活动的缓存、

不用删除key，因为用的时候，是通过出发到达拼接key，然后看该key下是否有活动，并且活动是有效的。

1. 删除该活动

Key：“activityId\_Cache”

Value：actId

<http://jimdb.jd.com>中配置的报警策略：

删缓存时，由于报警redis的出流量过大（故障持续30s会报警），因此使用Thread.sleep(5000),让当前线程休息5秒（每处理完一页pageSize数据）

## 去掉临时缓存

因为现在move中的逻辑是遍历所有的关联key，这样每个活动的move操作很耗时（写缓存1分钟，move则5分钟多）

直接往正式key中写数据

若有rename方法，则可以使用临时key，解决分布式同步问题

# 缓存新增

## 方案一：MQ

自己发送mq消息，自己消费（已不用）

主题：

#jmq actCreate topic  
base.ground.create.prom.activity.topic=createPromoAct

消息体：

{"activityId":983,"beginTime":1522252800000,"creator":"bjdiqi","endTime":1522771199000,"id":3601,"name":"首单立减test","promotionEnum":"NEW","statusEnum":"OVER","userLevelEnum":"REGISTER\_MEMBER"}

问题：

无法知道一个活动下的sku数据什么时候消费完，一个活动下的消息主题可能会有很多个

## 方案二：分页异步写

每页，将List<ActivitySKU>写入阻塞队列，使用异步线程进行处理。线程使用的是Executors.newSingleThreadExecutor()；内部使用的是无界队列。

# 上线前：

将man端的线上调用server的jsf别名改过来，现在是用的预发

# Jipiao前台使用：

一次并发读取sku规则数目：air\_base\_rule\_prom\_act\_sku\_concurrency\_number

