设计说明

版本 1.0

项目名称：crm-并发控制（红包领取）

**修订历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 作者 | 内容提要 | 核准人 | 发布日期 |
| 1.0 | 郦土 | 并发控制（红包领取） |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 关键技术

## 1.1 redis

在很多互联网产品应用中，有些场景需要加锁处理，比如：秒杀，全局递增ID等等。大部分的解决方案是基于DB实现的，Redis为单进程单线程模式，采用队列模式将并发访问变成串行访问，且多客户端对Redis的连接并不存在竞争关系。红包领取业务涉及到资金安全问题，综合考虑，为了保证业务的健壮性、数据强一致性，我们同时采用Redis和DB来解决高并发问题。

## 1.2 cas乐观锁

在JDK 5之前Java语言是靠synchronized关键字保证同步的，这会导致有锁机制。

锁机制存在以下问题：

（1）在多线程竞争下，加锁、释放锁会导致比较多的上下文切换和调度延时，引起性能问题。

（2）一个线程持有锁会导致其它所有需要此锁的线程挂起。

（3）如果一个优先级高的线程等待一个优先级低的线程释放锁会导致优先级倒置，引起性能风险。

synchronized就是一种悲观锁，而另一个更加有效的锁就是乐观锁。所谓乐观锁就是，每次不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止或超时。

CAS操作就是一种乐观锁，他运用cpu的新特性，现代的CPU提供了特殊的指令，可以自动更新共享数据，而且能够检测到其他线程的干扰。

# 并发控制

## 代码层面

### 2.1.1去掉无用的请求

在红包秒抢的情况下，肯定不能如此高频率的去读写数据库，会严重造成性能问题，可以使用redis缓存，将领取的红包数量放入缓存中，Redis为单进程单线程模式不存在并发问题。当接到用户领红包请求时，先将红包数量递减后，再进行数据库的操作，处理失败在将数据递增1，否则表示红包领取成功。当红包数量递减到0时，表示红包已领取完，拒绝其他用户的请求，提示红包已领完。

### 2.1.2控制请求并发数

两个地方需要控制并发：1.页面刚进来取缓存之前（减轻缓存压力） 2.数据库处理之前（减轻数据压力）

每次请求进来统计请求数，最大请求数可以动态配置（保存到缓存），比如最多只允许100个请求进来。每个请求进来以后计数器加一（加锁/解锁），那当计数器达到100的时候，表示拿不到锁，这时进入一个重试循环，直到成功获得锁或重试超时(timeout)，这样可以减轻数据库的压力。然后当一个请求处理完将计数器减一（加锁/解锁），再放其他线程进来。

这里用到CAS乐观锁，可以通过AtomicLong实现，AtomicLong原理就是CAS操作，自增incrementAndGet，自减decrementAndGet，每次判断incrementAndGet返回值，如果返回值小于或等于配置最大请求数，允许操作，说明此时并发线程数还未达到最大限制，反之需要重试，止到成功或者超时。

## 2.2数据库层面

数据库层面保证数据一致性，领红包整个动作需要放在一个事务当中：

可以用乐观锁方案

(1)并发情况下，判断红包池的红包是否领完

update hongbao\_pool set number\_obtain = number\_obtain+1

where number > number\_obtain

and pool\_id = 885951813;

返回=1,更新成功说明红包领取成功

返回=0,更新失败，抛出异常，红包已领完。

(2)同一个人并发领取红包

select \* from user\_account where user\_id=1

获取版本号，比如是1

update user\_account set hbtotal\_number = hbtotal\_number+1,

hbobtain\_amount = hbobtain\_amount +1.5,

version = version+1

where version = 1

and account\_id = 885951813;

返回=1,更新成功说明红包领取成功

返回=0,更新失败，抛出异常，红包已领过

# redis存在的问题

虽然redis是单进程单线程模式，但是在Jedis(redis java客户端)对Redis进行并发访问时会发生连接超时、数据转换错误、阻塞、客户端关闭连接等问题，这些问题均是由于客户端连接混乱造成。对此有2种解决方法：  
1.客户端角度，为保证每个客户端间正常有序与Redis进行通信，对连接进行池化，同时当并发量达到一定程度时（控制并发数），对客户端读写Redis操作采用synchronized或lock。  
2.服务器角度，利用setnx实现SETNX，GETSET，可以方便实现分布式锁机制。  
 为了简单方便，我们采用第一种方案，第二种方案当一个客户端获得锁但还未释放，如果系统瘫痪会导致死锁，处理相对比较复杂,不过可以提高性能（也是运用cas操作，后续可以考虑）。

# 4.接口设计

boolean RedisCacheUtils.put(String key,object value)

说明：缓存保存

入参：key,value

返回：true

Object RedisCacheUtils.get(String key)

说明：获取redis缓存

入参：key

返回：true

public static LstOfVal getLOV(String type,String name)

说明：获取LstOfVal配置表缓存

入参：type类型（如HING\_BAO场景），name名称(如：hongBaoObtain\_maxThread红包领取的最大线程数)

返回：LstOfVal

public boolean isHongBaoObtained (Long hbPoolId)

说明：红包池的红包是否已经领完

入参：无参数

返回：true

public boolean isHongBaoObtainThreadOut(Long hbPoolId)

说明：某个红包池的领取红包的线程是否超标

入参：无参数

返回：true