# 用数据库悲观锁控制

1.锁库存

2.插入“秒杀”记录

3.更新库存

低效，大家都在争锁库存，出现大量等待

# 用redis乐观锁控制

1.查询库存，带有版本号

2.更新库存--乐观锁控制是否成功

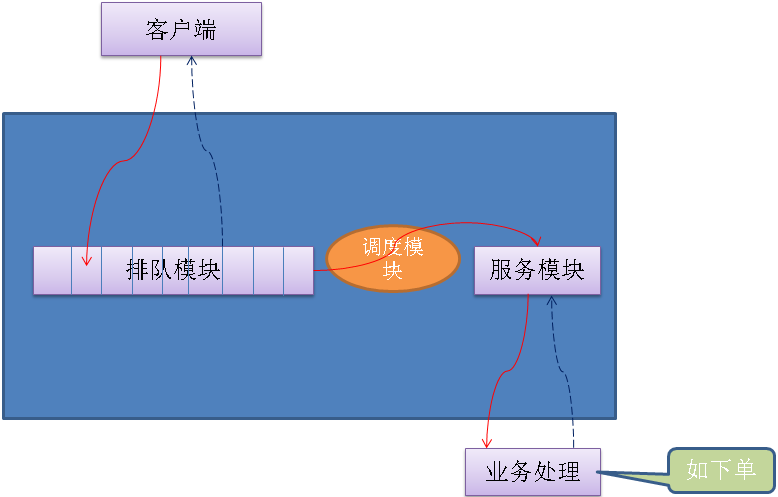
3.插入秒杀技术

http://www.csdn.net/article/2014-11-28/2822858

# 1号店秒杀系统的设计理念

基于以上秒杀场景下的痛点，1号店的秒杀排队系统在设计时主要考虑以下几点：

1. 限流：当秒杀活动开始后，只有少部分消费者能抢购到秒杀商品，意味着其实大部分用户的流量传达到后台服务后都是无效。如果能引导这大部分的流量，不让这大部分的流量传达到后台服务，其实对我们系统的压力就很小了。因此设计思路之一就是，仅让能成功抢购到商品的流量（可以有一定余量）进入我们的系统。
2. 削峰：进入系统的有效流量虽然总量不一定是很大的，但却是在很短的时间内涌入的，因此会存在很高的瞬时流量峰值。总量相同的流量在1秒钟进入系统，和在10分钟均匀地进入系统，对系统的冲击是相差很大的。高峰值的流量往往能将系统压垮。因此另一个设计思路是，如何将进入系统的瞬时高流量拉平，使得系统可以在自己处理能力范围内，将所有抢购的请求处理完毕。
3. 异步处理：传统的系统对于请求是同步处理的，即收到请求后立即处理并把结果返回给用户。我们的系统有了削峰的设计后，请求不是被立刻处理的，因此就要求我们能将同步的服务改造成异步的。
4. 可用性：我们设计时始终把系统的可用性放在重要的位置，针对系统可能出现的各种状况，都尽最大程度地保证高可用。
5. 用户体验：系统设计一定要充分考虑用户体验。消费者点击抢购按钮后，无论是否能抢到商品，期望是能得到及时的反馈。系统上发生任何故障也要尽可能的保证用户体验的损害减到最小。



# 京东 redis lua

京东采用redis + lua 来实现，在lua脚本中执行验证和扣减库存的操作，redis的特性保证了lua脚本也是单线程执行的

http://geek.csdn.net/news/detail/93986

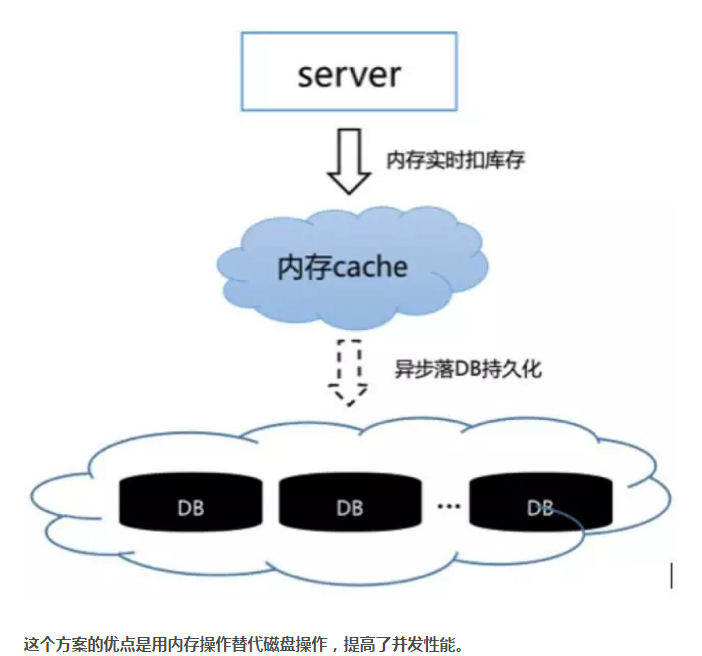
# 微信 hash分流 消息队列

微信不用锁控制抢红包，为了体验不能出现失败的情况，用的是队列来处理抢红包，FIFO先进先出

解决高并发问题常用方案

http://mp.weixin.qq.com/s?\_\_biz=MjM5MDE0Mjc4MA==&mid=2650995437&idx=1&sn=fefff4bff3e183d656a2d242e4c0a382&chksm=bdbf02be8ac88ba8d77252cf0279d708633364589a3c14e4e08fbe49e02dcf7ecfac50f141fe&mpshare=1&scene=1&srcid=0215MdZd0xOXXNwOF9Gfk3CW#rd

 方案一，使用内存操作替代实时的DB事务操作。



这个方案的优点是用内存操作替代磁盘操作，提高了并发性能。

但是缺点也很明显，在内存操作成功但DB持久化失败，或者内存Cache故障的情况下，DB持久化会丢数据，不适合微信红包这种资金交易系统。

方案二，使用乐观锁替代悲观锁。

但是如果应用于微信红包系统，则会存在下面三个问题：

如果拆红包采用乐观锁，那么在并发抢到相同版本号的拆红包请求中，只有一个能拆红包成功，其他的请求将事务回滚并返回失败，给用户报错，用户体验完全不可接受。

如果采用乐观锁，将会导致第一时间同时拆红包的用户有一部分直接返回失败，反而那些“手慢”的用户，有可能因为并发减小后拆红包成功，这会带来用户体验上的负面影响。

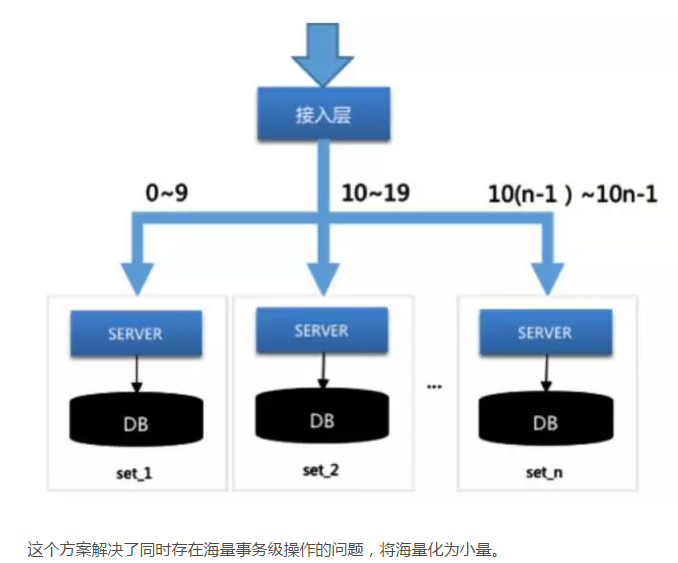
如果采用乐观锁的方式，会带来大数量的无效更新请求、事务回滚，给DB造成不必要的额外压力。

基于以上原因，微信红包系统不能采用乐观锁的方式解决并发抢锁问题。

微信红包系统的高并发解决方案

1.系统垂直SET化，分而治之。

红包系统根据这个红包ID，按一定的规则（如按ID尾号取模等），垂直上下切分。切分后，一个垂直链条上的逻辑Server服务器、DB统称为一个SET



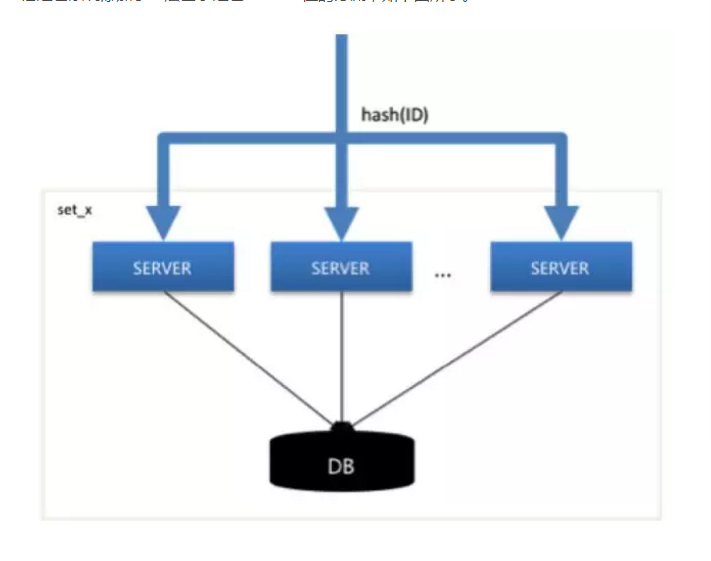
2.逻辑Server层将请求排队，解决DB并发问题。

**红包系统是资金交易系统，DB操作的事务性无法避免，所以会存在“并发抢锁”问题。但是如果到达DB的事务操作（也即拆红包行为）不是并发的，而是串行的，就不会存在“并发抢锁”的问题了。**

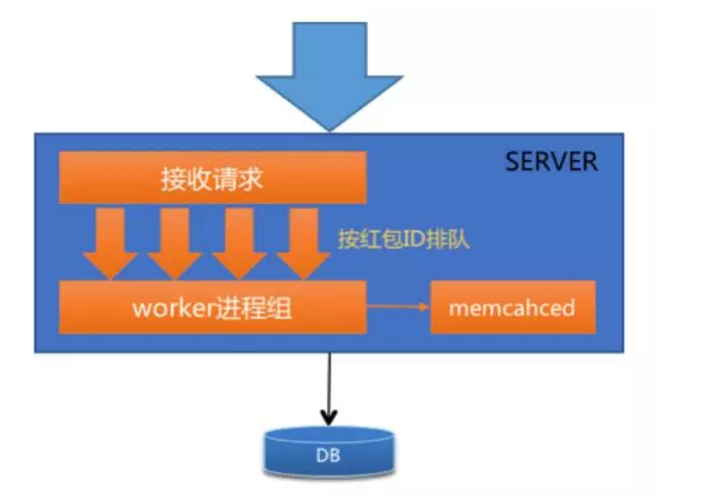
按这个思路，为了使拆红包的事务操作串行地进入DB，只需要将请求在Server层以FIFO（先进先出）的方式排队，就可以达到这个效果。从而问题就集中到Server的FIFO队列设计上。

不过在同个SET中，会存在多台Server服务器同时连接同一台DB（基于容灾、性能考虑，需要多台Server互备、均衡压力）。

为了使同一个红包ID的所有请求，stick到同一台Server服务器上，在SET化的设计之外，微信红包系统添加了一层基于红包ID hash值的分流，如下图所示。



将stick到同一台Server上的所有请求在被接收进程接收后，按红包ID进行排队。然后串行地进入worker进程（执行业务逻辑）进行处理，从而达到排队的效果，如下图所示。



**最后，增加memcached控制并发。**

为了防止Server中的请求队列过载导致队列被降级，从而所有请求拥进DB，系统增加了与Server服务器同机部署的memcached，用于控制拆同一个红包的请求并发数。

具体来说，利用memcached的CAS原子累增操作，控制同时进入DB执行拆红包事务的请求数，超过预先设定数值则直接拒绝服务。用于DB负载升高时的降级体验。

通过以上三个措施，系统有效地控制了DB的“并发抢锁”情况。

3.双维度库表设计，保障系统性能稳定

红包系统的分库表规则，初期是根据红包ID的hash值分为多库多表。随着红包数据量逐渐增大，单表数据量也逐渐增加。而DB的性能与单表数据量有一定相关性。当单表数据量达到一定程度时，DB性能会有大幅度下降，影响系统性能稳定性。采用冷热分离，将历史冷数据与当前热数据分开存储，可以解决这个问题。

处理微信红包数据的冷热分离时，系统在以红包ID维度分库表的基础上，增加了以循环天分表的维度，形成了双维度分库表的特色。

具体来说，就是分库表规则像db\_xx.t\_y\_dd设计，其中，xx/y是红包ID的hash值后三位，dd的取值范围在01~31，代表一个月天数最多31天。

通过这种双维度分库表方式，解决了DB单表数据量膨胀导致性能下降的问题，保障了系统性能的稳定性。同时，在热冷分离的问题上，又使得数据搬迁变得简单而优雅。

综上所述，微信红包系统在解决高并发问题上的设计，主要采用了**SET化分治**、**请求排队**、**双维度分库表**等方案，使得单组DB的并发性能提升了8倍左右，取得了很好的效果。