目录

[秒杀活动场景 2](#_Toc3192997)

[1、突增的服务器及网络需求 2](#_Toc3192998)

[2、业务高并发，服务负载重 2](#_Toc3192999)

[3、业务耦合度高，引起系统“雪崩” 2](#_Toc3193000)

[如何解决秒杀技术瓶颈 3](#_Toc3193001)

[秒杀架构设计思路： 3](#_Toc3193002)

[秒杀架构设计总结： 4](#_Toc3193003)

[限流： 4](#_Toc3193004)

[削峰： 4](#_Toc3193005)

[异步处理： 5](#_Toc3193006)

[内存缓存： 5](#_Toc3193007)

[可拓展： 5](#_Toc3193008)

## 秒杀活动场景

淘宝双11秒杀场景，大量的用户短时间内涌入，瞬间流量巨大（高并发），比如：1000万人同一时间抢购100件商品。秒杀活动是一个特别考验后台数据库、缓存服务的业务，对于数据库、缓存的性能要求特别严格。

秒杀背后的技术挑战

### 1、突增的服务器及网络需求

通常情况下，双 11 的服务器使用是平时的 3-5 倍，网络带宽是平时 N倍。

### 2、业务高并发，服务负载重

我们通常衡量一个 Web 系统的吞吐率的指标是 QPS（Query Per Second，每秒处理请求数），解决每秒数万次的高并发场景，这个指标非常关键。

假设处理一个业务请求平均响应时间为 100 ms，同时，系统内有 20 台 Web 服务器，配置最大连接数为 500 个，Web 系统的理论峰值 QPS 为（理想化的计算方式）：100000 （10万QPS）意味着1 秒钟可以处理完 10 万的请求，而“秒杀”的那 5w/s 的秒杀似乎是“纸老虎”。

实际情况，在高并发的实际场景下，服务器处于高负载的状态，网络带宽被挤满，在这个时候平均响应时间会被大大增加。随着用户数量的增加，数据库连接进程增加，需要处理的上下文切换也越多，服务器造成负载压力越来越重。

### 3、业务耦合度高，引起系统“雪崩”

更可怕的问题是，当系统上某个应用因为延迟而变得不可用，用户的点击越频繁，恶性循环最终导致“雪崩”，因为其中一台服务器挂了，导致流量分散到其他正常工作的机器上，再导致正常的机器也挂，然后恶性循环，将整个系统拖垮。

## 如何解决秒杀技术瓶颈

### 秒杀架构设计思路：

将请求拦截在系统上游，降低下游压力：秒杀系统特点是并发量极大，但实际秒杀成功的请求数量却很少，所以如果不在前端拦截很可能造成数据库读写锁冲突，甚至导致死锁，最终请求超时。

充分利用缓存(redis)：利用缓存可极大提高系统读写速度。

消息中间件**(**ActiveMQ、Kafka等**)：**消息队列可以削峰，将拦截大量并发请求，这也是一个异步处理过程，后台业务根据自己的处理能力，从消息队列中主动的拉取请求消息进行业务处理。

#### 前端设计方案

**页面静态化：**将活动页面上的所有可以静态的元素全部静态化，并尽量减少动态元素。通过CDN来抗峰值。

**禁止重复提交：**用户提交之后按钮置灰，禁止重复提交

**用户限流：**在某一时间段内只允许用户提交一次请求，比如可以采取IP限流

**后端设计方案**

**服务端控制器层(网关层)**

**限制uid（UserID）访问频率：**我们上面拦截了浏览器访问的请求，但针对某些恶意攻击或其它插件，在服务端控制层需要针对同一个访问uid，限制访问频率。

#### 服务层

上面只拦截了一部分访问请求，当秒杀的用户量很大时，即使每个用户只有一个请求，到服务层的请求数量还是很大。比如我们有100W用户同时抢100台手机，服务层并发请求压力至少为100W。

**采用消息队列缓存请求：**既然服务层知道库存只有100台手机，那完全没有必要把100W个请求都传递到数据库啊，那么可以先把这些请求都写到消息队列缓存一下，数据库层订阅消息减库存，减库存成功的请求返回秒杀成功，失败的返回秒杀结束。

**利用缓存应对读请求：比如**双11秒杀抢购，是典型的读多写少业务，大部分请求是查询请求，所以可以利用缓存分担数据库压力。

**利用缓存应对写请求：**缓存也是可以应对写请求的，比如我们就可以把数据库中的库存数据转移到Redis缓存中，所有减库存操作都在Redis中进行，然后再通过后台进程把Redis中的用户秒杀请求同步到数据库中。

#### 数据库层

数据库层是最脆弱的一层，一般在应用设计时在上游就需要把请求拦截掉，数据库层只承担“能力范围内”的访问请求。所以，上面通过在服务层引入队列和缓存，让最底层的数据库高枕无忧。

**比如：利用消息中间件和缓存实现简单的秒杀系统**

Redis是一个分布式缓存系统，支持多种数据结构，我们可以利用Redis轻松实现一个强大的秒杀系统。

我们可以采用Redis 最简单的key-value数据结构，用一个原子类型的变量值(AtomicInteger)作为key，把用户id作为value，库存数量便是原子变量的最大值。对于每个用户的秒杀，我们使用 RPUSH key value插入秒杀请求， 当插入的秒杀请求数达到上限时，停止所有后续插入。

然后我们可以在台启动多个工作线程，使用 LPOP key 读取秒杀成功者的用户id，然后再操作数据库做最终的下订单减库存操作。

当然，上面Redis也可以替换成消息中间件如ActiveMQ、Kafka等，也可以将缓存和消息中间件 组合起来，缓存系统负责接收记录用户请求，消息中间件负责将缓存中的请求同步到数据库。

## 秒杀架构设计总结：

### 限流：

鉴于只有少部分用户能够秒杀成功，所以要限制大部分流量，只允许少部分流量进入服务后端。

### 削峰：

对于秒杀系统瞬时会有大量用户涌入，所以在抢购一开始会有很高的瞬间峰值。高峰值流量是压垮系统很重要的原因，所以如何把瞬间的高流量变成一段时间平稳的流量也是设计秒杀系统很重要的思路。实现削峰的常用的方法有利用缓存和消息中间件等技术。

### 异步处理：

秒杀系统是一个高并发系统，采用异步处理模式可以极大地提高系统并发量，其实异步处理就是削峰的一种实现方式。

### 内存缓存：

秒杀系统最大的瓶颈一般都是数据库读写，由于数据库读写属于磁盘IO，性能很低，如果能够把部分数据或业务逻辑转移到内存缓存，效率会有极大地提升。

### 可拓展：

当然如果我们想支持更多用户，更大的并发，最好就将系统设计成弹性可拓展的，如果流量来了，拓展机器就好了。像淘宝、京东等双十一活动时会增加大量机器应对交易高峰。