14｜百万级流量秒杀系统的关键总结

2021-10-27 佘志东

《手把手带你搭建秒杀系统》 课程介绍 >

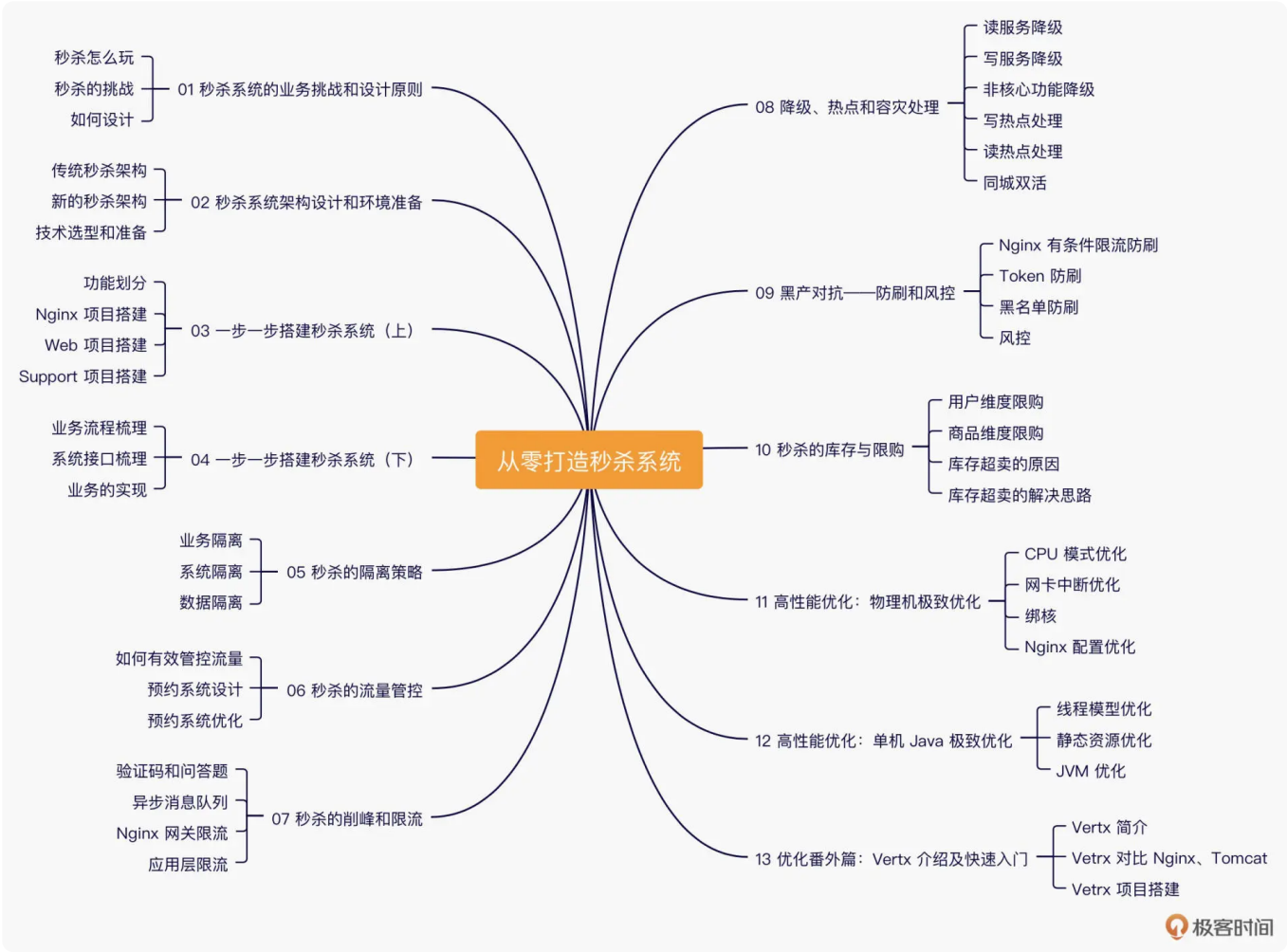
讲述：佘志东

时长 13:47 大小 12.64M

你好，我是志东，欢迎和我一起从零打造秒杀系统。

经过前面课程的介绍，相信你已经能够对秒杀系统的设计和实施有了比较深入的理解，也能够在自己的项目中去应用这些设计原则和方法了。那么我们的课程也差不多到尾声了，这一节课我们主要做一下总结，和你一块回顾之前的学习内容（关于代码库，本月会完成更新，欢迎追更）。

正如开篇词所讲，我们主要是从系统准备、着手搭建、系统高可用、一致性以及性能优化等维度进行秒杀系统的学习。为了便于你总结，我把每节课的重点整理成了下面这张维导图，带你系统复习一下秒杀系统的全部内容。



01 秒杀系统的业务挑战和设计原则

第一节，我首先介绍了秒杀的业务特点和挑战。秒杀是电商平台大促狂欢时非常重要的手段之一，用具有价格优势的稀缺商品，来增加电商平台的关注度，带来空前的流量。**因此，秒杀的主要挑战在于：**

> 高并发产生的巨大瞬时流量。秒杀活动的特点，就是将用户全部集中到同一个时刻，然后一起开抢某个热门商品，而热门商品的库存往往又非常少，因此聚集效应产生了巨大的瞬时流量。
>
> 高并发无法避开的热点数据问题。秒杀活动大家抢购的都是同一个商品，所以这个商品直接就被推到了热点的位置。
>
> 来自黑产的刷子流量。刷子高频次的请求，会挤占正常用户的抢购通道，也获得了更高的秒杀成功率。这不仅破坏了公平的抢购环境，也给系统服务带来了巨大的额外负担。

接着，我们从技术层面介绍了 HTTP 服务的请求链路路径。我们讨论了将秒杀系统提供的业务功能，按不同阶段、不同响应，合理地拆分到不同的链路层级来实现，以符合我们校

验前置、分层过滤、缩短链路的设计原则，并能够从容应对秒杀系统所面临的瞬时大流量、热点数据、黄牛刷子等各种挑战。

02 秒杀系统架构设计和环境准备

在这一节里，针对秒杀系统，我们将传统的架构设计与我们新的架构设计做了一个对比，可以看出传统架构设计的局限性。其中列举了域名带宽问题和 Tomcat 服务器性能问题，新的架构设计将 Web 网关职能前置，尽量在流量入口处拦截掉风险流量，缩短请求链路，保护下游系统，并提高服务的响应速度。

技术选型时，我们采用的是**主流的技术栈**，Web 服务和 RPC 服务的基础框架都是使用 SpringMVC，RPC 框架使用的是 Dubbo，数据库使用免费开源的 MySQL，分布式缓存数据库使用 Redis。

03 一步一步搭建秒杀系统（上）

在这节课中我们开始开发一个最简的秒杀系统，共设计了 **3 个系统项目**：一个是 demo-nginx，用来做真正的网关入口；另一个是 demo-web，用来做业务的聚合；最后一个是 web-support，用来做基础数据和服务的支撑。

04 一步一步搭建秒杀系统（下）

在第 03 课的基础上，这节课我们梳理了秒杀的业务流程和系统的关键接口，接着通过**七个步骤**实现了秒杀的关键业务，在本地实现了“秒杀活动的创建 -> 活动开始的打标 -> 从商详页进秒杀结算页 -> 提交订单 -> 活动的关闭与去标”的完整交互，让我们以“摸得着”的方式去近距离地接触秒杀系统。

05 秒杀的隔离策略

这节课开始我们进入了秒杀的高可用专题，我们介绍了秒杀隔离，它是秒杀系统高可用体系非常重要的一个环节。如果不做隔离，任由流量互相横冲直撞，将会对电商平台的普通商品售卖造成很大的影响。隔离的措施概括下来有三种：**业务隔离、系统隔离和数据隔离**。

在隔离过程中，一般购物车和订单不需要做特殊定制，只需要根据流量情况进行专门部署即可。而挑战比较大的就是秒杀的结算页系统，它是秒杀流量的主要入口，承担着把瞬时

流量承接下来并进行优质流量筛选的重任，因此如何搭建秒杀结算页的高可用、高性能和高并发至关重要。

为了对秒杀的结算页系统进行隔离，核心思路是对商品进行打标，当用户在详情页点击秒杀操作的时候，我们就能根据商品是否秒杀的标识，跳转到普通商品结算页流程，或是隔离出来的专用秒杀结算页系统。

06 秒杀的流量管控

这节课我们介绍了主流电商平台通用的营销方式：**预约 + 秒杀**。通过事前引入预约环节，进行秒杀参与人数的把控，起到秒杀流量管控的目的。通过预约控制参与人数上限，只有预约过的会员才有秒杀资格，就可以防止过多人数对秒杀抢购造成冲击。

除此之外，这节课我们还重点学习了预约系统的设计思路，介绍了预约系统的推荐架构设计，关键的两张数据库表，以及预约系统需要提供的接口。根据这些思路，你可以很快速地搭建出一个比较简单的预约系统。

07 秒杀的削峰和限流

这节课我们介绍了事中控制流量的方式，有验证码、问答题、消息队列以及限流等，这些削峰的方式都可以达到控制流量的目的。

我们重点学习了验证码的设计和实现，通过代码了解了验证码的生成和校验两个过程。在验证码设计时，为了交互更加安全，我们需要**加入签名机制**，同时验证通过后，需要**加入后端黑名单**，避免多次验证。验证码是一种非常常见的防刷手段，大多数网站的登录模块中，为避免被机器人刷，都会加入图片验证码。而在秒杀系统中，我们除了用验证码来防刷外，还有一个目的就是通过验证码进行削峰，以达到流量整形的目的。

接着我们重点介绍了秒杀的几种限流方式，和其他削峰方式相比，限流是有损的。限流是根据服务自身的容量，无差别地丢弃多余流量，对于被丢弃的流量来说，这块的体验是受损的。另外，因为秒杀流量会经历很多交易系统，所以我们在设计时需要从起始流量开始，分层过滤，逐级限流，这样流量在最后的下单环节就是少量而可控的了。

在 demo-nginx 层，我们主要采用的是 Nginx 自带模块进行网关限流，而在 demo-web 层主要采用的是线程池限流来控制并发数和基于令牌桶的 API 限流方法。

08 降级、热点和容灾处理

这节课我们主要讨论了秒杀的降级策略，热点数据的处理方式以及“同城双活”的容灾方案。

降级的设计非常重要，它是系统故障发生时你的逃生路径。这一节课里，我们学习了几种常见的降级场景和解决方法。这些方法都可以结合你的业务场景进行应用。

写服务降级：牺牲数据一致性获取更高的性能，在大厂的设计中比较常见，对于异步造成的数据丢失等一致性问题，一般会有定时任务一直在比对，以便最快发现问题，进行修复。

读服务降级：在做高可用系统设计时，我们认为微服务自身所依赖的外部中间件服务或者其他 RPC 服务，随时都可能发生故障，因此我们需要建设多级缓存，以便故障时能及时降级止损。

简化系统功能：在秒杀场景下，并不是页面信息越丰富越好，要视情况而定。秒杀系统要求尽量简单，交互越少，数据越小，链路越短，离用户越近，响应就越快，因此非核心的功能，比如商品的收藏总数量、商品的排行榜、评价和推荐等楼层在秒杀场景下都是可以降级的。

接着我们还学习了秒杀的热点数据处理，热点数据是秒杀系统的基本属性，读热点问题的解决遵循朴素的思路，通过增加数据副本数来扛流量，同时尽量让数据靠近用户。

这里也介绍了 **3 种写热点解决方法：**一是本地缓存，延迟提交；二是将写热点数据进行分片；三是单 SKU 限流。实际上，Redis 的单片写能力可以达到几万 QPS，所以即便是秒杀扣库存这样的写热点操作，通过单 SKU 限流也能应对。

最后，我重点介绍了“同城双活”的容灾方案，本质上多活的难点就是数据的复制和一致性问题，因此比较主流的做法是同城单写。通过秒杀系统的同城双活设计，你可以看到，不管是 Nginx 集群、Tomcat 集群、Redis 集群，还是 MySQL 集群，我们都可以灵活进行机房间切换，在故障时快速恢复。

09 黑产对抗——防刷和风控

这节课我们主要介绍了如何对抗黑产以及应对方案。

Nginx 有条件限流机制：直接有效拦截针对接口的高频刷子请求，可以有效解决黑产流量对单个接口的高频请求。

Token 机制：可以有效防止黑产流量跳过中间接口，直接调用下单接口。

黑名单机制：简单、高效，配置合理可以拦截大部分刷子。

风控：风控体系需要建立在大量的数据之上，并且要通过复杂的实际业务场景考验，不断地做智能修正，逐步提高风险识别的准确率。

10 秒杀的库存与限购

这节课我们重点探讨了限购和库存。限购的作用有两个，一个是限制用户在确定时间内的购买单数和商品件数，比如限制同一手机号每天只能下 1 单，每单只能购买 1 件，并且一个月内只能购买 2 件，确保秒杀的公平性，让爆品惠及更广泛的用户；另一个作用是充当活动库存的用途，通过限购控制每天的投放总量，或者整个活动的投放总量。

所以我们的重点就放在了活动库存的扣减方案设计上，讨论了出现超卖的场景以及如何规避。我们从纯技术的角度分析了**库存超卖发生的两个原因：**一个是库存扣减涉及到的两个核心操作，查询和扣减不是原子操作；另一个是高并发引起的请求无序。

我们的应对方案是利用 Redis 的单线程原理，通过 Lua 来实现库存扣减的原子性和顺序性，并且经过实测也确实能达到我们的预期，且性能良好，从而有效地解决了秒杀系统所面临的库存超卖挑战。

11 高性能优化：物理机极致优化

从这节课开始我们进入高性能优化专题，我们介绍了物理机与 Nginx 相关配置的优化，优化的方向是对内存、CPU、IO（磁盘 IO 和网络 IO）的优化。

对于物理机，我们主要**从调整 CPU 的工作模式入手**，将 CPU 模式切换成高性能模式，以得到更好的响应性能。另外在秒杀高峰期间，我们也做了针对性的措施，即通过绑定专门 CPU 来处理网卡中断。

当然绑核的操作不只可以针对网卡中断，还可以绑定 Nginx 进程以及部署的其他应用服务。这么做的目的，一方面是为了减少 CPU 调度产生的开销，另一方面也可以提高每个 CPU 核的缓存命中率。

接着我们又讨论了 Nginx 的优化，分别针对客户端以及下游服务端，从网络的连接、传输、超时等方面做了不同的配置讲解。具体的 Nginx 优化配置你可以参考以下示例：

复制代码

```
#工作进程：根据CPU核数以及机器实际部署项目来定，建议小于等于实际可使用CPU核数
worker_processes 2;

#绑核：MacOS不支持。
#worker_cpu_affinity   01 10;

#工作进程可打开的最大文件描述符数量，建议65535
worker_rlimit_nofile 65535;

#日志：路径与打印级别
error_log logs/error.log error;



events {
    #指定处理连接的方法，可以不设置，默认会根据平台选最高效的方法，比如Linux是epoll
    #use epoll;
    #一个工作进程的最大连接数：默认512，建议小于等于worker_rlimit_nofile
    worker_connections 65535;
    #工作进程接受请求互斥，默认值off,如果流量较低，可以设置为on
    #accept_mutex off;
    #accept_mutex_delay 50ms;
}

http {
        #关闭非延时设置
        tcp_nodelay  off;
        #优化文件传输效率
        sendfile     on;
        #降低网络堵塞
        tcp_nopush   on;

        #与客户端使用短连接
        keepalive_timeout  0;
        #与下游服务使用长连接,指定HTTP协议版本，并清除header中的Connection，默认是close
        proxy_http_version 1.1;
        proxy_set_header Connection "";

        #将客户端IP放在header里传给下游，不然下游获取不到客户端真实IP
        proxy_set_header X-Real-IP $remote_addr;

        #与下游服务的连接建立超时时间
        proxy_connect_timeout 500ms;
        #向下游服务发送数据超时时间
        proxy_send_timeout 500ms;
        #从下游服务拿到响应结果的超时时间（可以简单理解成Nginx多长时间内，拿不到响应结果，
```

```
        #这个根据每个接口的响应性能不同，可以在每个location单独设置
        proxy_read_timeout 3000ms;

        #开启响应结果的压缩
        gzip on;
        #压缩的最小长度，小于该配置的不压缩
        gzip_min_length  1k;
        #执行压缩的缓存区数量以及大小，可以使用默认配置，根据平台自动变化
        #gzip_buffers     4 8k;
        #执行压缩的HTTP请求的最低协议版本，可以不设置，默认就是1.1
        #gzip_http_version 1.1;
        #哪些响应类型，会执行压缩，如果静态资源放到CDN了，那这里只要配置文本和html即可
        gzip_types       text/plain;


        #acccess_log的日志格式
        log_format  access  '$remote_addr - $remote_user [$time_local] "$reque
            '"$upstream_addr" "$upstream_status" "$upstream_response_time" use

        #加载lua文件
        lua_package_path "/Users/~/Documents/seckillproject/demo-nginx/lua/?.l
        #导入其他文件
        include /Users/~/Documents/seckillproject/demo-nginx/domain/domain.com
        include /Users/~/Documents/seckillproject/demo-nginx/domain/internal.c
        include /Users/~/Documents/seckillproject/demo-nginx/config/upstream.c
        include /Users/~/Documents/seckillproject/demo-nginx/config/common.con
}
```

12 高性能优化：单机 Java 极致优化

这一节课主要围绕着 Java，分析和讲解了与其息息相关的 Tomcat、JVM、RPC 框架以及静态资源的优化。

对于 Tomcat 的优化，在秒杀的特定业务场景下针对线程模型的选择，NIO2 从理论和实际压测上看，比 NIO 是有吐吞量的提升，但不是很大，如果为了省事，选择默认的 NIO 即可。对于 RPC 框架，我们主要介绍了 Netty 的 Boss Pool 和 Worker Pool 来实现 Reactor 模式。

接着，我们介绍了静态资源的优化方案，即**将静态资源上到 CDN**，以减少对秒杀域名流量的压力，同时可以依靠 CDN 的全国部署，快速加载到对应的静态资源。

13 优化番外篇：Vertx 介绍及快速入门

这一节初步介绍了 Vertx，**Vertx 提供了异步化、非阻塞的解决方案**。和我们传统的开发方式有所不同，异步化提升了性能，当然异步化编程势必会增加代码的复杂度，这也是其弊端。

接着我们横向对比分析了 Vertx 与 Nginx、Tomcat 的优劣势，其性能介于 Tomcat 与 Nginx 之间。像 Nginx，性能最优，我们用其来做前置网关，直面大流量的冲击；Vertx 性能次之，所以我们用其来开发业务 Web 服务，但 Vertx 受众小，有一定的门槛，开发者难寻；Tomcat 使用起来最方便也最普及，可以用来发布 RPC 服务，或者是像 ERP 这种小流量的 Web 服务。

思考题

到这节课为止，我们已经把秒杀课程的核心内容介绍完了，相信经过体系化的学习，你已经掌握了秒杀的基本设计思路，以及能够着手自己进行高可用、高性能、高并发的系统搭建。

那么除了以上介绍的这些内容，你觉得还有哪些方面也是需要我们考虑的呢？例如，系统的压测、监控和应急？或者还有么？请你先思考，我们下一节课再来讨论！

上一篇　13 | 优化番外篇：Vertx介绍及快速入门

精选留言 (1)

写留言

nana

2021-10-27

想念老师啊

展开