

## 05 | 勇于担当：秒杀的隔离策略

2021-10-04 余志东

《手把手带你搭建秒杀系统》

课程介绍 >



讲述：余志东

时长 11:51 大小 10.86M



你好，我是志东，欢迎和我一起从零打造秒杀系统。

通过前面几节课的学习，相信你对秒杀系统已经有了一个初步的认识，也已经能够按照第二课、第三课和第四课的指引一步一步搭建出一个极简的秒杀系统了。

那么，当前的秒杀系统是否已经足够强大去应对我们所说的最大的挑战了呢？当面对巨大的瞬时流量冲击的时候，当前的秒杀系统是否能够像三峡大坝一样扛住洪峰，坚不可摧呢？显然，还有距离。



为了让系统更加的坚固，屹立不倒，我们还需要做哪些工作对秒杀系统进行加固呢？

从这节课开始，我们将深入到秒杀系统的优化细节，从**高可用、高性能、高并发**等维度出发，一步一步打造一个满足真实业务需求的能够应对百万级用户参与的超大流量冲击的秒杀系统。

## 为流量而生

这里我想先请你思考一个问题，非常简单。你觉得普通商品和秒杀商品最本质的区别是什么？

显而易见的是流量不同。针对普通商品，销量当然是越多越好，所以商家备货一般都会很充足，这样用户去购买的时间就会分散开，流量也会比较均衡。而秒杀商品，说白了，就是稀缺爆品，特点就是库存少，因此用户会去抢购，刷子也会热情高涨，以致瞬时流量巨大。

另外，普通商品和秒杀商品的数量级也是完全不同的。在头部电商平台，几十亿的商品都是普通商品，只有少数（百个以下）的商品具备秒杀商品的特点。

面对这样的区别，这两类商品其实很难在电商平台上一块进行交易。因为秒杀流量是突发式的，而且流量规模很难提前准确预估，如果混合在一起，势必会对普通商品的交易造成比较大的冲击。所以就像我们开篇词讲的，需要单独搭建秒杀系统，它天然为流量而生。

## 秒杀的隔离

很自然，为了不让 0.001% 的爆品影响 99.999% 普通商品的交易，我们很快就想到了隔离。隔离是控制危险范围的最直接的手段，正如当下新冠病毒肆虐，采取严格隔离和松散管控不同方式的不同国家，取得的效果也是完全不同的。

而面对超预期的瞬时流量，我们也要采取很多措施进行流量的隔离，防止秒杀流量串访到普通商品交易流程上，带来不可预估的灾难性后果。



上图是几个比较重要的隔离策略，接下来我们详细展开讨论。

## 业务隔离

秒杀商品的稀缺性，决定了业务不会像普通商品那样进行投放售卖。一般会有计划地进行营销策划，制订详细的方案，以达到预期的目标。

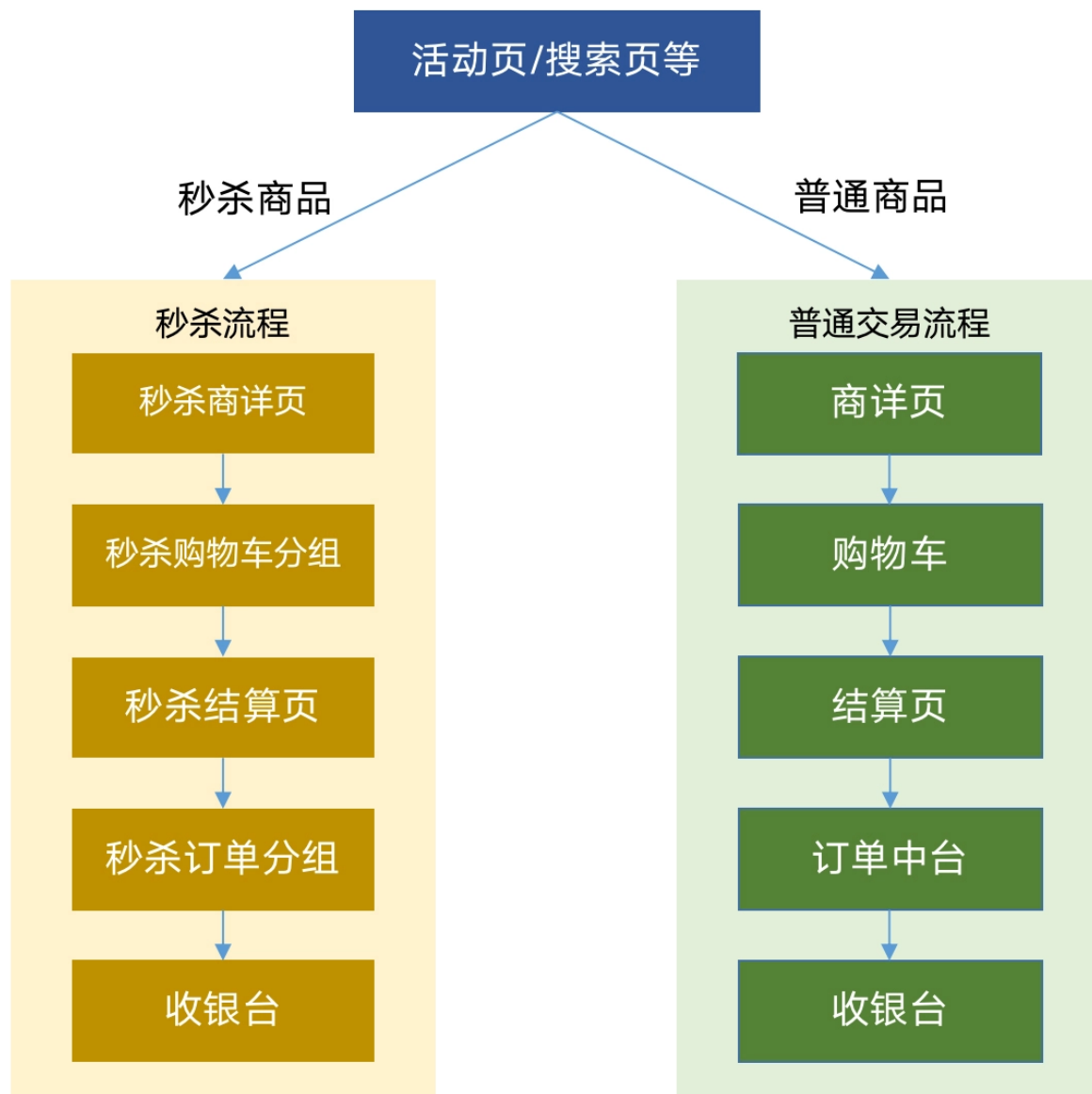
因此，从业务上看，它是和普通商品完全不一样的售卖流程，它需要一个提报过程。大部分的电商平台，会有一个专门的提报系统（提报系统的建设不是秒杀的核心部分，这里不再赘述），商家或者业务可以根据自己的运营计划在提报系统里进行活动提报，提供参与秒杀的商品编号、活动起止时间、库存量、限购规则、风控规则以及参与活动群体的地域分布、预计人数、会员级别等基本信息。

你别小看这个提报过程和这些基本信息，有了这些信息作为输入，我们就能预估出大致的流量、并发数等，并结合系统当前能支撑的容量情况，评估是否需要扩容，是否需要降级或者调整限流策略等，因此业务隔离的重要性可见一斑。

## 系统隔离

接下来我们看下系统隔离。前面已经介绍过商品交易流程大概会用到哪些系统，理论上讲，需要把交易链路上涉及到的系统都单独复制部署一套，隔离干净，但这样做成本比较高，一般大点的电商平台都采用分布式微服务的部署架构，服务数量少则几十个，多则几百个，全部复制一套进行隔离不现实。

所以比较常见的实践是对会被流量冲击比较大的核心系统进行**物理隔离**，而相对链路末端的一些系统，经过前面的削峰之后，流量比较可控了，这些系统就可以不做物理隔离。




极客时间

我们知道，用户的秒杀习惯，一般是打开商品详情页进行倒计时等待，时间到了点击秒杀按钮进行抢购。因此第一个需要关注的系统就是商品详情页，我们需要申请独立的秒杀详情页域名，独立的 Nginx 负载均衡器，以及独立的详情页后端服务，并采用 Dubbo 独立分组的方式单独提供秒杀服务。

详情页的独立部署完成之后，你可能会有疑问，流量怎么会自己跑到独立的部署集群去呢？这个问题我们先放一放，在后面数据隔离的部分再为你解答。

我们先来看下如何通过 Dubbo 的分组来提供独立的微服务集群。


## 服务端代码：

 复制代码

```

1 package com.ecommerce.product.service;
2 public interface IProductService {
3     public SkuInfo getSkuInfo(String skuId);
4 }

```


 复制代码

```

1 package com.ecommerce.product.service.impl;
2 import com.ecommerce.product.service.IProductService;
3 @Autowired
4 CacheManager cacheManager;
5 public class ProductServiceImpl implements IProductService {
6
7     //根据商品编号获取商品详细信息
8     @Override
9     public SkuInfo getSkuInfo(String skuId) {
10         return cacheManager.getSkuInfo(skuId);
11     }
12 }

```

## 服务端 applicationContext 配置：

 复制代码

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans      http
6
7     <!-- 配置Bean -->
8     <bean id="productService" class="com.ecommerce.product.service.impl.Produc
9     <!-- 引入配置文件 -->
10     <import resource="classpath:dubbo.xml"/>
11 </beans>

```

## 服务端 dubbo.xml 配置：

 复制代码

```

1 <?xml version="1.0" encoding="UTF-8"?>

```

```

2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6     http://www.springframework.org/schema/beans/spring-beans.xsd
7     http://code.alibabatech.com/schema/dubbo
8     http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
9     <!-- 指定web服务名字 -->
10    <dubbo:application name="ProductGroup"/>
11    <!-- 声明服务注册中心 -->
12    <dubbo:registry protocol="zookeeper" address="127.0.0.1:2181"/>
13    <!-- 指定传输层通信协议 -->
14    <dubbo:protocol name="dubbo" port="20880"/>
15    <!-- 暴露你的服务地址 -->
16    <dubbo:service
17        ref="productService"
18        interface="com.ecommerce.product.service.IProductService"
19        protocol="dubbo"
20        <!-- 普通交易流程，商品后端微服务的逻辑分组 -->
21        group="product-cloud"
22    />
23 </beans>

```

以上是商品后端服务的一些基础代码和配置，不知道你有没有注意到第 21 行的逻辑分组定义，这里表明当前 Dubbo 提供了一个分组名为 **product-cloud** 的微服务，给普通商品交易流程使用。

那为了隔离出专门给秒杀通道的服务，我们只需要申请相应的容器资源，复制以上配置，并对分组名进行修改就完成了。

[复制代码](#)

```

1 <!-- 暴露你的服务地址 -->
2 <dubbo:service
3     ref="productService"
4     interface="com.ecommerce.product.service.IProductService"
5     protocol="dubbo"
6     <!-- 秒杀流程，商品后端微服务的逻辑分组 -->
7     group="seckill"
8 />

```

这样我们商品后端微服务的隔离就完成了。接着我们看下游调用端如何实现隔离。

调用端 dubbo.xml 配置：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6     http://www.springframework.org/schema/beans/spring-beans.xsd
7     http://code.alibabatech.com/schema/dubbo
8     http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
9
10    <!-- 指定web服务名字 -->
11    <dubbo:application name="ProductGroup"/>
12    <!-- 声明服务注册中心 -->
13    <dubbo:registry protocol="zookeeper" address="127.0.0.1:2181"/>
14
15    <!-- 指定传输层通信协议 -->
16    <dubbo:protocol name="dubbo" port="20881"/>
17
18    <!-- 引用你的服务地址 -->
19    <dubbo:reference
20        id="productService"
21        interface="com.ecommerce.product.service.IProductService"
22        protocol="dubbo"
23        <!-- 这里引用的分组名，选择了秒杀流程的逻辑分组 -->
24        group="seckill"
25    />
26 </beans>
```

根据以上第 24 行代码可知，调用方在使用时，根据场景选择相应的分组名进行调用，那么流量就会走到不同的微服务集群里，从而达到微服务流量隔离的目的。

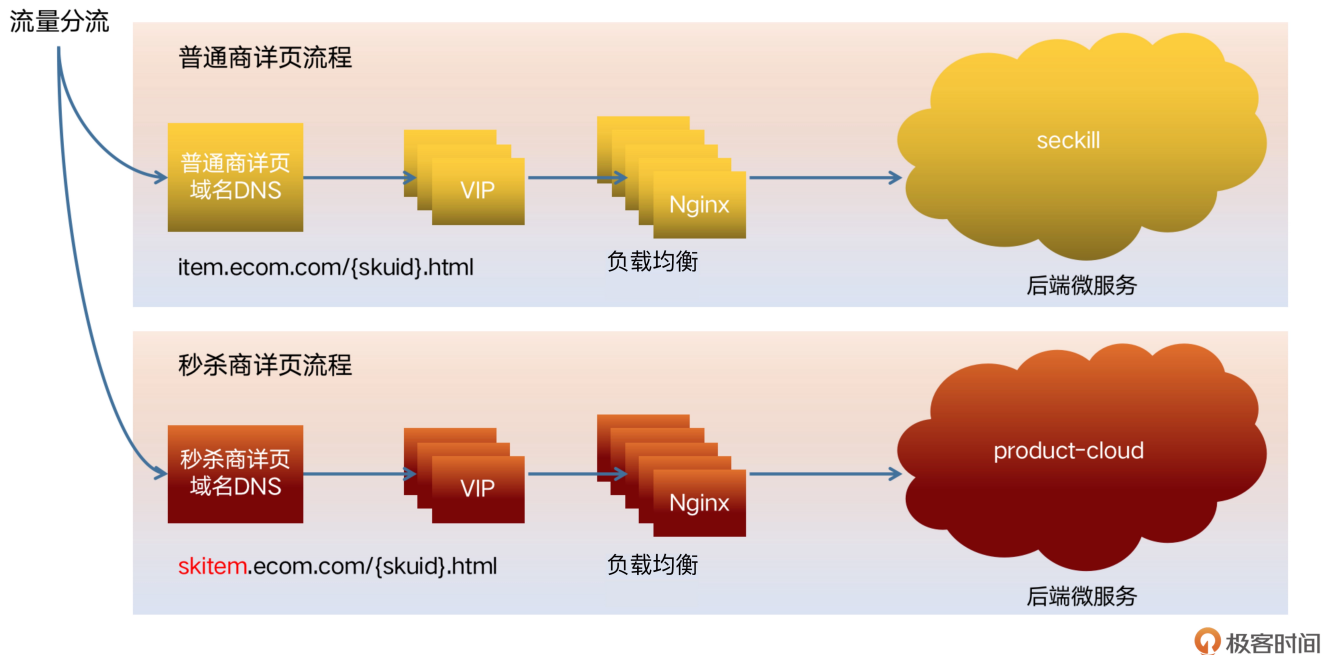
前面我们介绍了微服务集群的隔离方法，接下来我们看下负载均衡器的隔离。为了能水平扩展服务的能力，一般我们在流量入口都会通过负载均衡器来进行流量分配，常用的有硬件负载均衡，比如 F5，其功能和性能优于软件方式，但一般比较昂贵。

大厂里比较常用的负载均衡器都是软件方式，有 LVS、HAProxy、Nginx 等，一方面是出于成本考虑，毕竟大厂的规模非常大，单个 F5 的硬件成本能承受，但是大规模的硬件成本就很高了；另一方面开源的软件也更加灵活和可定制。

但不管采用的是硬件还是软件，为了不让流量互相影响，我们都有必要对负载均衡进行隔离，需要单独部署一套。隔离的方式也比较简单，部署之后，把相应的 IP 地址挂载到不同的 DNS 域名下就好了。



紧接着我们仍需要对域名进行隔离，我们可以向运维部门申请一个独立的域名，专门用来承接秒杀流量，流量从专有域名进来之后，分配到专有的负载均衡器，再路由到专门的微服务分组，这样就做到了应用服务层面从入口到微服务的流量隔离。



以上就是商品详情页的系统隔离方法，交易流程的其他系统，比如结算页、价格中心、订单中心等也可以参照类似的方式进行隔离，这里就不重复讲解了。

在我看来，这里流量冲击比较大的核心系统就是秒杀详情页、秒杀结算页，是需要我们重点关注的对象，而相对链路末端的一些系统，经过前面的削峰之后，流量比较可控了，如接单系统、收银台、支付系统，物理隔离的意义就不大，反而会增加成本。你自己设计秒杀系统的时候，可以格外注意一下。

## 数据隔离

现在，我们已经完成了应用层的隔离。接下来，在数据层面，我们也应该进行相应的隔离，否则如果共用缓存或者共用数据库，一旦瞬时流量把它们冲垮，照样会影响无辜商品的交易。

数据层的专有部署，需要结合秒杀的场景来设计部署拓扑结构，比如 Redis 缓存，一般的场景一主一从就够了，但是在秒杀场景，需要**一主多从**来扛读热点数据。关于热点数据的处理在后面的课程中我们会详细进行介绍，这里先留个悬念。



到这里为止，我们基本学习完了秒杀隔离策略。现在，我们回过头来思考系统隔离中遗留的问题：**怎么让秒杀流量正确地路由到我们隔离出来的专有环境里来呢？**

答案就是对商品进行打标，在商品的主数据上有了秒杀标，那么我们在任何一个环节都可以把这个染色过的流量进行正确地路由了。

那么既然提到了商品打标，这里我再简单介绍一下商品打标的设计思路。当然，电商平台的商品系统设计远远比这复杂。

打标就是一个标记，我们可以使用一个 long 型字段 skuTags 来保存，long 是 64 位，每一位代表一种类型的活动，0 代表否，1 代表是，通过对 skuTags 进行二进制操作即可完成商品的打标和去标。假设秒杀的标识我们定义在 skuTags 的第 11 位，那么要给一个 sku 打上秒杀标，我们就可以对这个标实际进行“或”操作： $\text{skuTags} = \text{skuTags} | 1024$ ，这样 skuTags 字段的第 11 位就变成了 1，对其它 bit 位没影响。去标过程相反，同样进行位操作， $\text{skuTags} = \text{skuTags} \& \sim 1024$ ，把第 11 位置为 0。

好了，**最后我们把整个隔离流程再串一下。**

首先业务通过提报系统对秒杀 sku 进行提报，系统对秒杀 sku 进行打标，从活动页、列表页或者搜索页点击商品的时候，系统就能识别出秒杀标，路由到秒杀的商品详情页域名，进而进入到专有 Nginx。

然后就是到专有的微服务分组，以及专有的 Redis 缓存了。这里提一下，上面介绍的流量分流实际上是从活动页就开始的，为了节约成本，我们也可以设计在商品详情页进行分流，这样做的好处是商品详情页是通用的实现，也是通用的部署，当用户在详情页点击购买的时候，才根据是否有秒杀标识进行流量分流。劣势就是进行秒杀的时候，商品详情页的流量压力会比较大。

## 小结

秒杀系统的特点倒逼我们不得不做流量隔离。如果不做隔离，任由流量互相横冲直撞，将会对电商平台造成很大的影响。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。

其中我们需要重点关注系统和数据的隔离，从 ROI 的角度看，我们需要找出电商交易平台最核心的几个系统进行隔离，从头部电商平台的实践来看，一般会单独设计和部署秒杀的商详页和结算页系统，以及结算页系统链路下游的购物车和订单系统。

在这个过程中，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。

这节课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，接下来我们将从其他方面继续探讨秒杀系统的高可用，敬请期待下一节课流量管控的内容！

## 思考题


如果采用通用的商品详情页，当用户点击购买按钮的时候才进行流量分流，那么商品详情页的流量压力就比较大，这种设计有什么办法可以避免秒杀流量对普通商品的详情页造成冲击吗？

以上就是这节课的全部内容，欢迎你在评论区和我讨论问题，交流经验！

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 0

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 指日可待：一步一步搭建秒杀系统（下）

下一篇 06 | 谋定后动：秒杀的流量管控

## 精选留言 (2)

 写留言



林伟烽  
2021-10-04

把商祥页静态化，丢到cdn上？  
展开



\_xcc  
2021-10-04

系统隔离，第二个代码段，@Autowired位置错了吧  
展开

