**【业务背景】**

你作为一个电商创业公司的架构师，负责设计 6.18 大促秒杀系统的设计，你们的业务模式如下：

1. 你们挑选选品各大电商平台上畅销和好评的商品进行销售，每个品类不超过 20 个商品，目前做了 10 个品类；
2. 本次 6.18 秒杀选择了 1000 个充电宝，10 台 iPhone12 作为秒杀商品；
3. 正常的日活大约 100 万用户；
4. 老板要求万无一失。

**【技术背景】**

1. 技术团队以 Java 为主，已经落地了微服务架构；

2. 主要渠道是自有的 App（包括 iOS 和 Android）和微信小程序，为了促进用户转化为 App 用户，只有下载 App 才能参加秒杀活动；

3. 目前只有单机房。

**【毕设要求】**

1.设计完整的架构，例如存储、负载均衡、缓存、高可用、可扩展等；

2.大约 10 页以内的 PPT，每页 PPT 说明一项设计，包括架构设计和设计理由，无需详细解释备选方案。

**【提示】**

1.分析考虑要全面，但并不意味着架构设计要面面俱到，如果分析后认为某些设计点可以不做，就在作业最后统一说明原因即可；

2.如果没有思路，请对照模块 9 的 IM 案例。

3.如果有的信息觉得不够全或者不够细，可以做出一定的假设，但不能天马行空，需要对照已有的业务背景和技术背景进行合理推断。

**【设计】**

**技术分析**

* 什么是秒杀：秒杀场景一般会在电商网站举行一些活动或者节假日在 12306 网站上抢票时遇到。对于电商网站中一些稀缺或者特价商品，电商网站一般会在约定时间点对其进行限量销售，因为这些商品的特殊性，会吸引大量用户前来抢购，并且会在约定的时间点同时在秒杀页面进行抢购。
* **秒杀系统难做的原因**：库存只有一份，所有人会在集中的时间读和写这些数据。 例如小米手机每周二的秒杀，可能手机只有 1 万部，但瞬时进入的流量可能是几百几千万。 又例如 12306 抢票，亦与秒杀类似，瞬时流量更甚。核心问题就是**极高并发处理**，由于系统要在瞬时承受平时数十倍甚至上百倍的流量，这往往超出系统上限，因此处理秒杀的**核心思路是流控和性能优化**, 以及应对瞬时的大流量高并发。
* **架构理念：**
* **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。

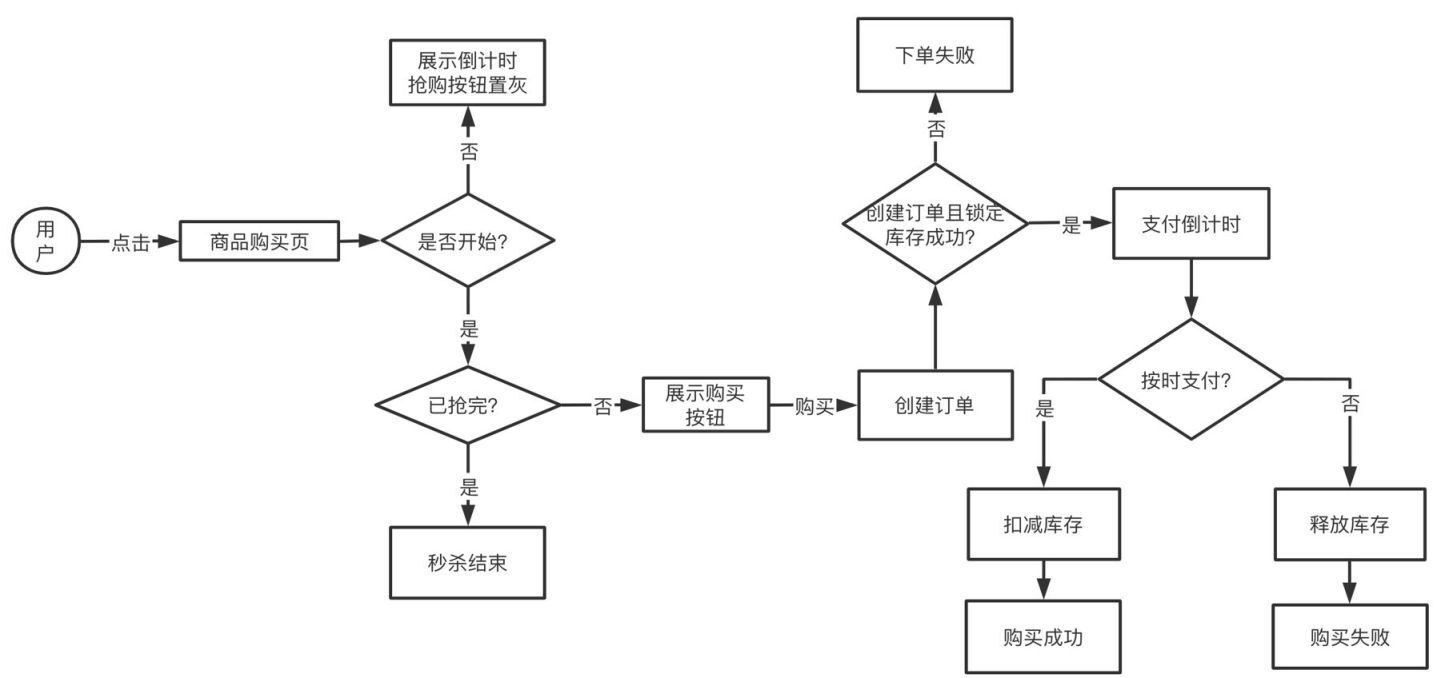
**可拓展：**当然如果我们想支持更多用户，更大的并发，最好就将系统设计成弹性可拓展的，如果流量来了，拓展机器就好了。像淘宝、京东等双十一活动时会增加大量机器应对交易高峰。

* 秒杀商品数量不多，秒杀活动的瞬间流量很高，需要用缓存和消息队列，降低后端压力。
* 日活用户 100 万，在秒杀活动期间，至少会有 3~5 倍的用户参与，架构设计至少参照百万级别架构设计。
* 由于要求万无一失，要考虑高可用，目前只有一个机房，所以需要必须上云。
* 团队已经落地微服务，而秒杀活动的流量很高，所以在原有的 spring Cloud 基础上，单独开一些服务，专门做秒杀。
* 创业型公司，技术人员不会太多，作为业务重点的秒杀活动，可用根据需求找 8 个人做后端，android 3 人，ios 3 人，微信小程序 3 人。

**流量控制**

* **请求流控：**尽可能在上游拦截和限制请求，限制流入后端的量，保证后端系统正常。因为无论多少人参与秒杀，实际成交往往是有限的，而且远小于参加秒杀的人数，因此可以通过前端系统进行拦截，限制最终流入系统的请求数量，来保证系统正常进行。
* **客户端流控：**在客户端进行访问限制，较为合适的做法是屏蔽用户高频请求，比如在网页中设置 5s 一次访问限制，可以防止用户过度刷接口。这种做法较为简单，用户体验也尚可，可以拦截大部分小白用户的异常访问，比如狂刷 F5。关键是要明确告知用户，如果像一些抢购系统那样假装提交一个排队页面但又不回应任何请求，就是赤裸裸的欺骗了。
* **后端系统流控：**对于后端系统的访问限制可以通过异步处理、消息队列、并发限制等方式实现。核心思路是保证后端系统的压力维持在可以正常处理的水平。对于超过系统负载的请求，可以选择直接拒绝，以此来对系统进行保护，保证在极限压力的情况下，系统有合理范围内的处理能力。

**业务流程图**



**系统架构设计**

1.注册登录

TPS: 日活用户 100 万人，假定活动当天新增 20 倍用户，一共 2000 万用户，80%登录注册发生在 5 小时内，则 TPS=2000 万\*0.8/5/3600=900

2.热销商品

QPS: 10 大类，每类 20 个商品，一共 200 个商品。当天一共 5 倍的人查看，每人查看 3 次，每次查看 5 个商品，且 80%的请求发生在 5 小时内，则 QPS=500 万\*3\*5\*0.8/5/3600=340

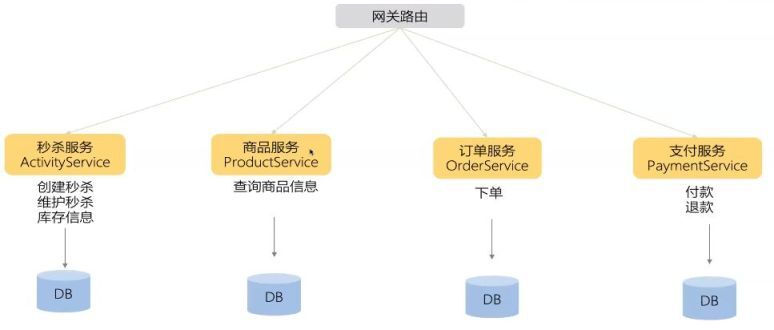
3.秒杀商品

QPS：1000 个充电宝，10 个 iphone，假定当天 5 倍人参与，共 500 万，每个商品每人点击请求 3 次，秒杀请求发生在 1 分钟内。则 TPS=500 万\*2\*3/60=50 万

4.支付

TPS： 热销商品：点击抢购后，进入支付界面，支付前查询商品剩余数量是否大于 1，支付后商品数量减 1，TPS 为 340。

TPS： 秒杀商品：和上面逻辑一致，TPS 为 50 万。



**存储架构设计**

1.注册登录

假定有 10 倍的用户进行登录注册，存储的用户注册和登录信息 1000 万

2.热销和秒杀商品库存

只有 200 条左右商品，但秒杀开始时，会有高并发库存查询，redis 单机 TPS 为 5~10 万，所以使用 redis cluster。

3.订单

预计登录用户有 50%的购买转化率，每人平均购买 2 件商品。则订单存储量为 1000 万\*0.5\*2=1000 万。

4.秒杀活动创建/维护时写入 Redis。

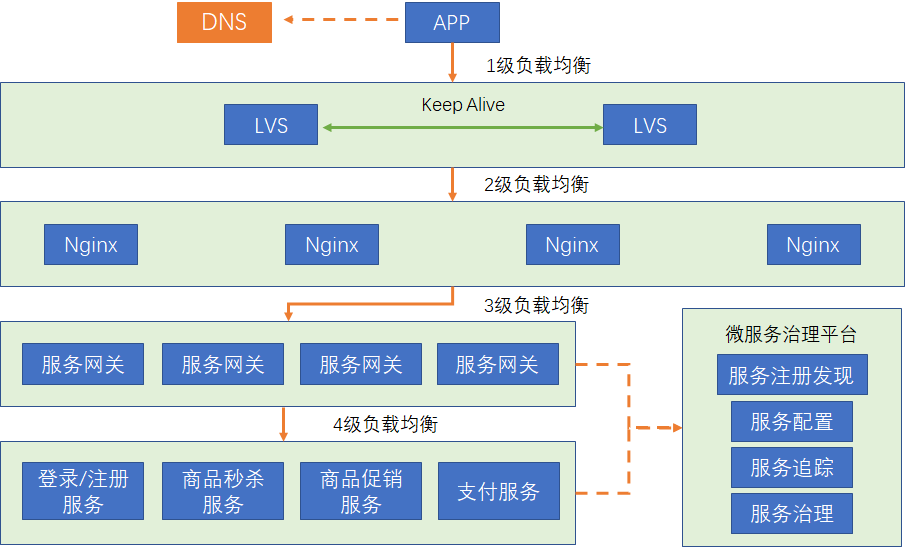
5.使用分布式事务或消息队列来保证活动数据库和库存数据一致

6.把库存数据放入到缓存中，利用缓存的原子特性保证同时只有一个线程操作库存来防止超卖



**负载均衡设计**

这里我们可以采用多级负载均衡（由于 F5 价格过高），为满足峰值数百万的请求，LVS 已足够满足需求，使用多个 LVS 作为一级负载均衡。

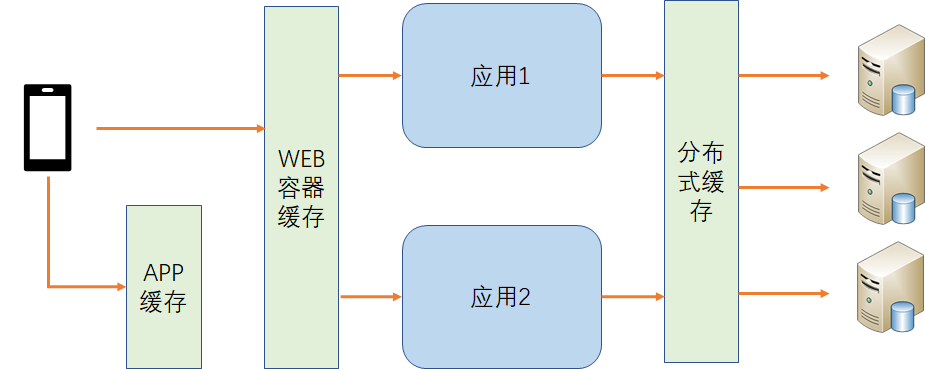


**缓存设计**

1.缓存设计可以采用 App 缓存+web 容器缓存+分布式缓存

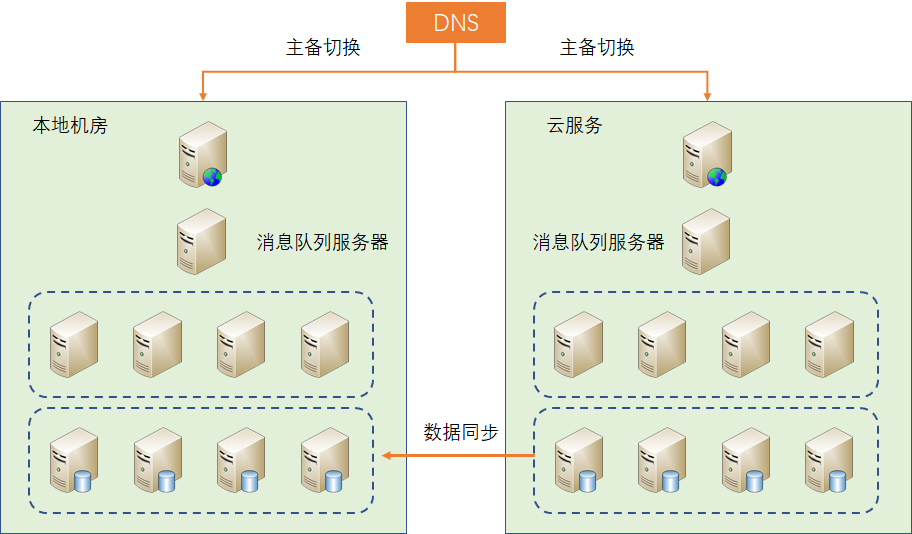
2.使用 APP 缓存和 WEB 缓存将大量静态页面保存起来，秒杀时静态页面从 APP 或 Web 获取，减轻后端压力。

3.秒杀开始后，为支持大量库存查询请求，使用 redis 保存库存，分布式缓存使用 redis，为应对大流量，redis 采用 redis cluster 方式部署。



**高可用设计**

为确保在秒杀开启后，大流量冲击下万无一失，确保高可用，这项内容是在云计算环境下才成为可能，相对于传统的 IT 行业，云计算提供了快速的系统交付能力（min VS. day），因此可以做到按需分配，在业务需要时实现资源的并行扩展。可以购买一定的异地云服务，将数据同步到云中，发生本地宕机情况下，自动切换到云端。同时将请求放入消息队列，以应对高频请求，降低服务端压力。



**总结**

总的说来，面对高流量的解决方式，首先通过各种条件筛选掉无效流量，进行流量错峰，然后再对现有的系统性能做出优化，也可以通过独立部署的方式和其它环境做隔离，并且充分利用云环境来进行扩容以及容错，保持系统的高可用。