电商秒杀系统 - 业务背景

1. 参与秒杀的商品 1000 个充电宝， 10 台 iPhone 12
2. 正常日活约 100万用户
3. 老板要求万无一失

技术背景

1. 技术团队 Java 为主，已经落地了微服务架构
2. 主要渠道有自由 App（包括IOS和Andriod）和微信小程序，为了促进用户转化为App用户，只有下载了App才能参加秒杀活动
3. 目前只有单机房

秒杀业务基本场景和流程

整体的业务流程比较短，秒杀的本质是对库存的抢购。

1. 用户登录。
2. 秒杀开始前，页面展示商品，并显示展示倒计时。
3. 秒杀开始后，如果商品没被抢完，页面能展示购买按钮，否则，显示秒杀结束。
4. 用户点击购买后，后台创建订单后，锁定库存并开始支付倒计时。如果库存锁定失败，就显示购买失败。
5. 用户按时支付了，就扣减库存，显示购买成功
6. 用户没有按时支付，就释放库存，显示购买失败

存储架构设计

存储性能估算：

正常是百万的日活，假设前期宣传效果很好，秒杀活动大概有300万人来参加。

1. **登录**：假设秒杀活动开始之前的30分钟内有300万人来登录，QPS = 300 万 /（30 x 60s）= 1700/s
2. **开始秒杀**：大量用户会在同一时间抢购，网站流量瞬间激增，QPS 可达 300 万/s
3. 订单创建：订单数量和被抢购的商品数量有关，这里只有1010个商品，最多就1010个订单，所以 TPS = 1010 /s 。如果要考虑平台上本身就有的商品也可能产生订单，那么 TPS 最多 = 1500 /s

存储架构设计：

1. 登录：存用户数据，使用 MySQL 主备存。
2. 订单创建：存订单信息，也可以用 MySQL 主备存。
3. 开始秒杀：见下页。

存储架构设计

存储架构设计：

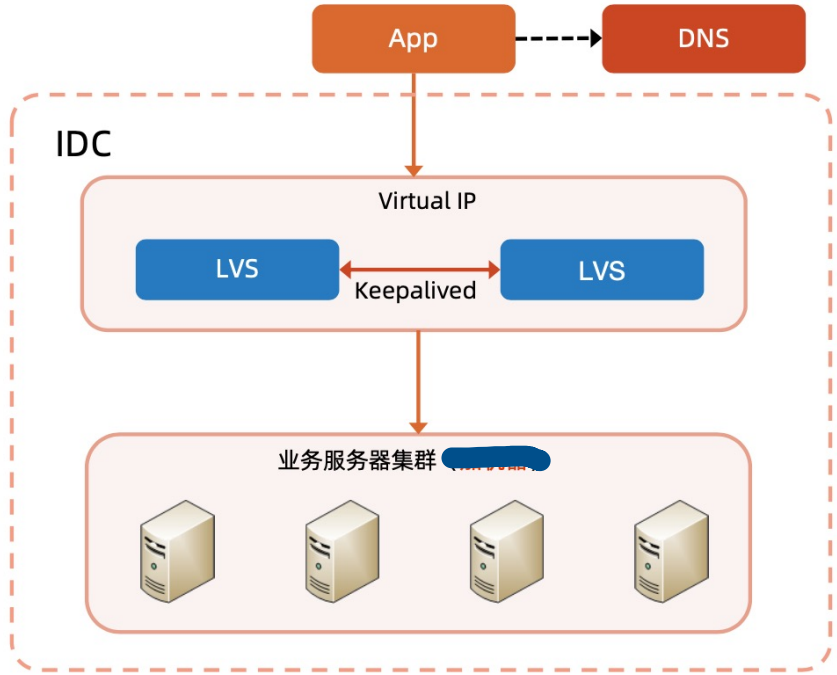
3. 开始秒杀。这里为了扛住瞬时的大流量读取操作，秒杀服务不能直接从读 MySQL 中读取库存，而是使用 Redis cluster 提前加载好秒杀商品的库存信息，然后秒杀服务通过访问 Redis Cluster 来获取秒杀商品的库存信息。

为了扛住 300 万的QPS，Redis Cluster 使用分片架构。

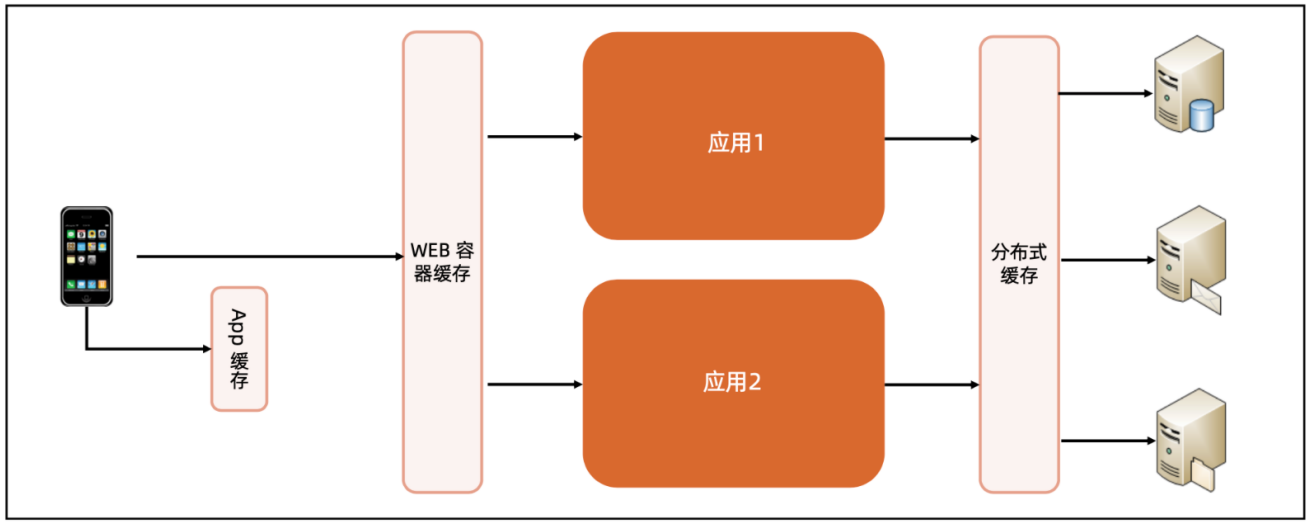
计算架构设计

负载均衡

这里为了扛住 300万用户的访问，使用 LVS。



缓存架构



其他架构设计

微服务拆分：

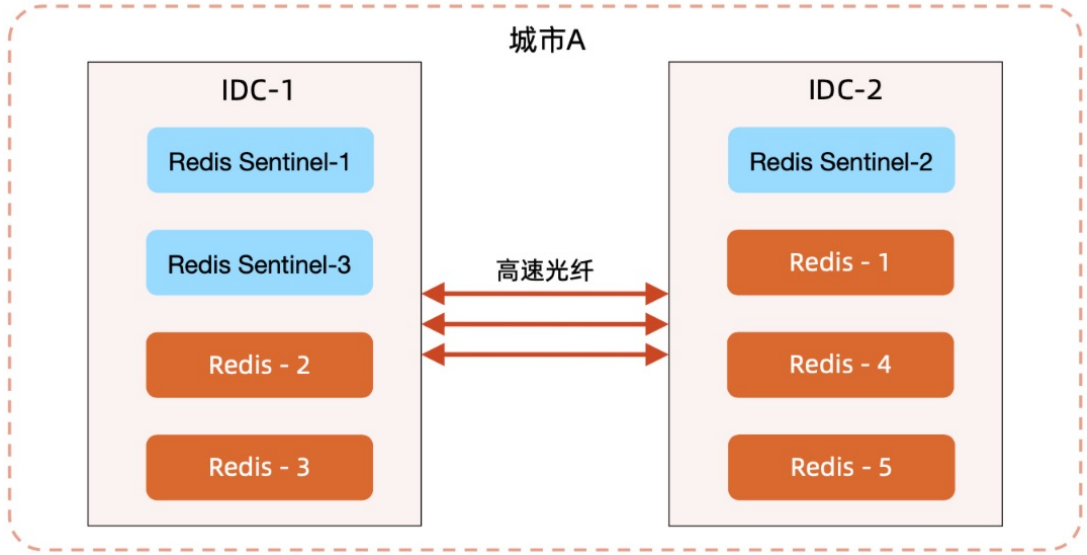
1. 用户服务
2. 秒杀服务
3. 订单服务

默认已具备微服务基础设施。

默认已具备支付功能。

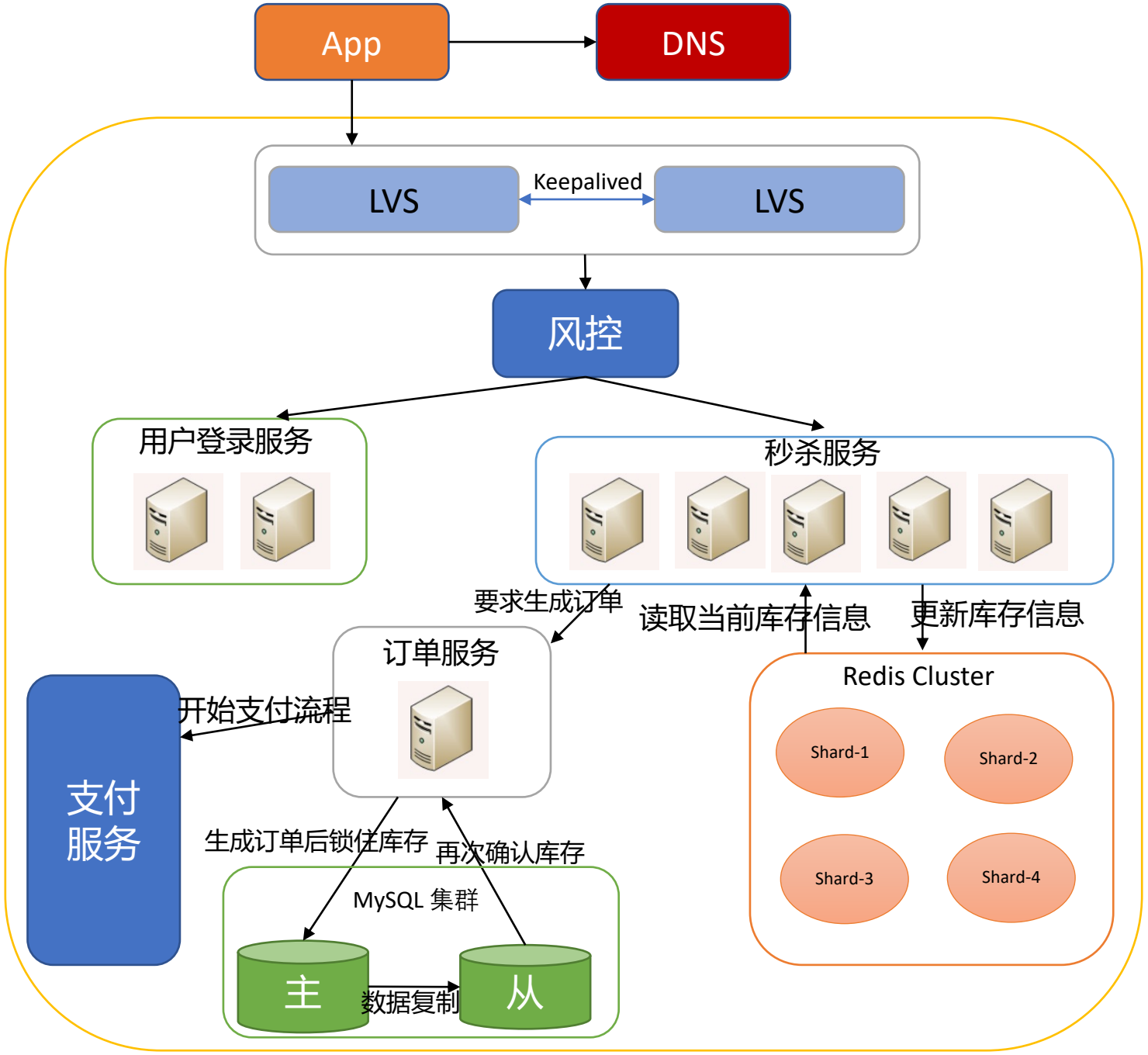
高可用架构 – 同城双中心

Redis Cluster 是关键，要保证其高可用。



架构设计方案 1

- **风控**：假设已有相应的微服务。这个是防止黄牛抢东西、爬虫和不停下单但却一直不付钱的恶意买家。
- 支付流程：复用已经有的支付微服务。



架构设计方案 2

- 风控：假设已有相应的微服务。这个是防止黄牛抢东西、爬虫和不停下单但却一直不付钱的恶意买家。
- 支付流程：复用已经有的支付微服务。
- RocketMQ：这里可以将要求下单的请求放进消息队列中，然后订单服务可以慢慢消化这些请求，从而减轻其对MySQL的请求压力。