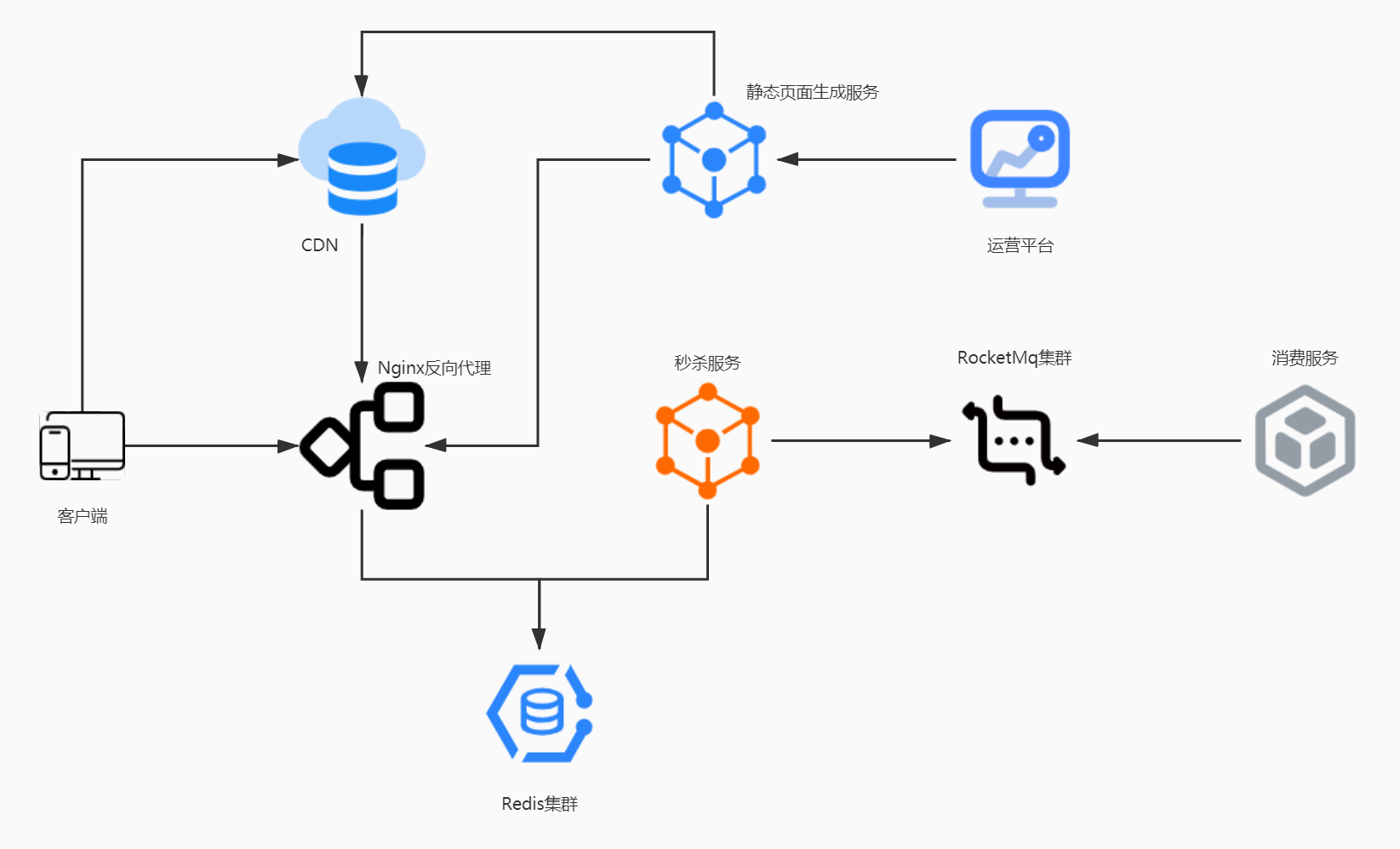
# 秒杀系统设计文档

## 整体架构



1. **静态资源**

**通过动静分离去除业务处理和前端资源的耦合，降低业务服务器压力。**

**运营平台**在秒杀活动上线前调用**静态页面生成服务**生成秒杀商品相关的静态页面资源。然后服务将静态资源上传至**Nginx**访问的静态资源服务路径下。同时，**静态页面生成服务**调用**CDN**的API来配置**Nginx**静态资源路径以及资源刷新策略。**客户端**访问秒杀商品页面时，则直接访问**CDN**。当**CDN**找不到对应资源时，再访问**Nginx**静态资源路径，并将资源拉取至**CDN**本地储存再返回**客户端**。

1. **商品、活动数据**

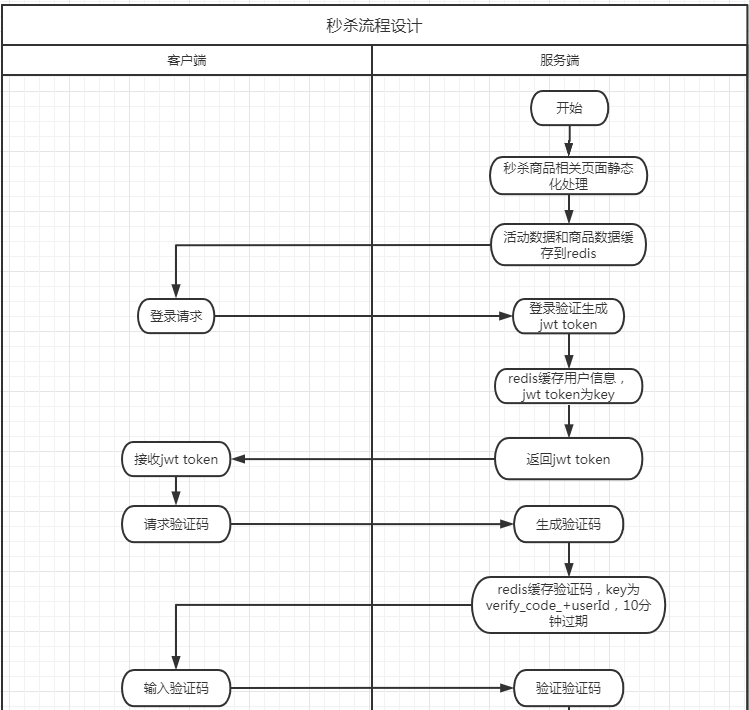
秒杀业务会经常访问到秒杀商品信息以及秒杀活动的信息，因此需要使用到缓存来减轻数据库压力。在响应速度方面，离客户端最近的缓存才是响应最快的缓存，所以将缓存提前到**Nginx**中。**Nginx**通过LUA脚本访问**Redis**中的缓存，若**Redis**中无缓存再去访问秒杀服务。当然**Redis**中缓存可以透过秒杀服务进行启动预加载。同时缓存前置可以在一定程度上减轻秒杀服务压力。

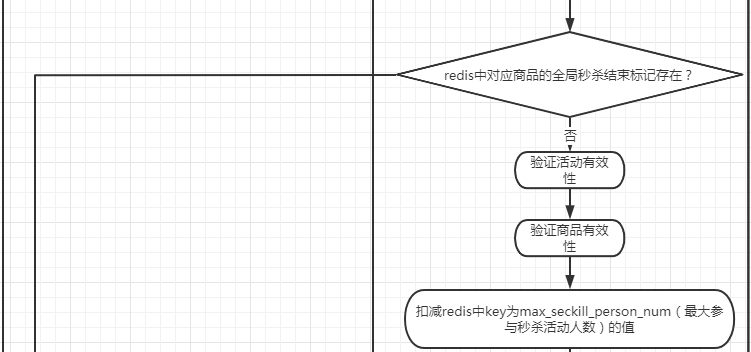
1. **秒杀服务**

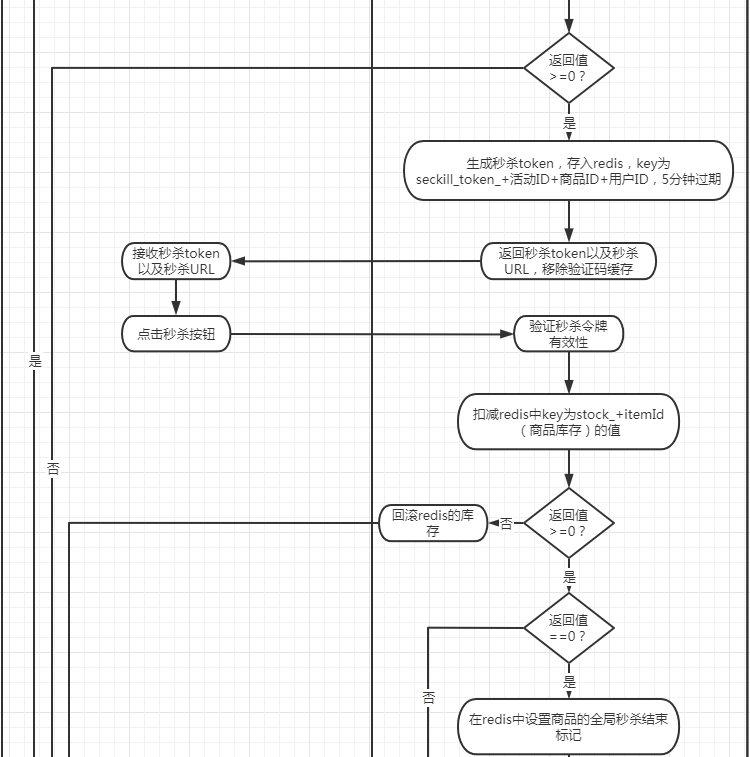
一方面，秒杀业务需要用户登录，后续所有流程都将频繁进行用户登录校验和权限校验。这里考虑用户登录后的登录信息和权限信息缓存到Redis中，来提高服务的TPS。

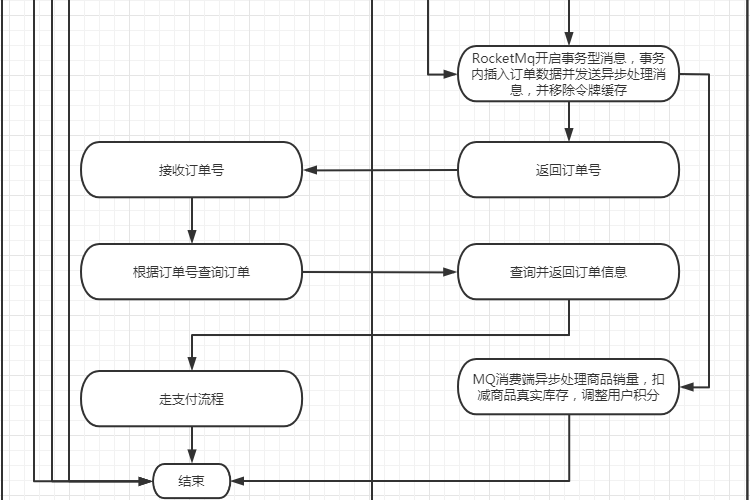
另一方面，秒杀业务涉及核心流程和非核心流程。其中核心流程就是用户秒杀、下单，**非**核心流程则是扣减库存、处理商品销量或者处理用户积分。基于高并发考虑，用户秒杀的商品数量放到**Redis**中去控制。下单后发送非核心流程通知消息发送到**RocketMq**，下游**消费服务**消费**RocketMq**的消息再进行相应的**非**核心流程处理。特别注意的是这里下单和发送消息需要使用**RocketMq**的**事务性消息**来确保核心业务线和**非**核心业务线的原子性。

## **流程设计**









秒杀处理流程主要步骤有获取验证码->获取秒杀token->秒杀->下单->异步扣库存，其中比较关键的对象有：

1. 验证码

可以验证和拦截机器人操作

1. 最大参与秒杀活动人数（秒杀大闸）

Redis中设置该key（一般设为秒杀商品数量的4-5倍），为的最大限度去降低真实的抢单流量以提高服务可用性。同时又确保业务真实性，真实地让多数人去抢少数商品。至于为何不在验证码生成阶段就介入该key的拦截，也是为了确保业务真实性。

1. 秒杀token与秒杀URL

秒杀token是进行用户进行秒杀业务的唯一凭证。同时秒杀的真实URL在此阶段才真实返回前端，为的是防止秒杀URL泄露被攻击。

1. 虚拟库存

Redis中存放秒杀商品的虚拟库存，秒杀阶段将对该key进行扣减，扣减结果>=0则认为秒杀成功。

1. 全局秒杀结束标记

当Redis中虚拟库存扣减完毕后，对应商品则会在Redis中存储对应的全局秒杀结束标记。该标记拦截是否生成秒杀token。