九章算法
www.jiuzhang.com

秒杀系统与订票系统设计

Seckill & Booking System Design

主讲人：西毒老师

系统架构设计试听福利群

该二维码7天内(8月24日前)有效，重新进入将更新

4S 分析法

Scenario 场景

- 说人话：需要设计哪些功能，设计得多牛
- Ask / Features / QPS / DAU / Interfaces

Service 服务

- 说人话：将大系统拆分为小服务
- Split / Application / Module

Storage 存储

- 说人话：数据如何存储与访问
- Schema / Data / SQL / NoSQL / File System

Scale 升级

- 说人话：解决缺陷，处理可能遇到的问题
- Sharding / Optimize / Special Case

Work Solution
Not Perfect Solution

Scenario 场景

- 具体场景有哪些?
- 实际需求有什么?
- 详细流程怎么样?

秒杀系统场景

2020 年6 月18 日 0 点开始，京东自营限量 100 台，以 4000 元的价格，抢购 iPhone 11 64G 版本，先到先得，一人限购一台，售完即止。

微信抢红包

抢春运火车票

抢购小米手机

秒杀系统场景

关注 分享 对比 举报

京东物流 【2020新款】微影Q8手机投影仪家用迷你便携办公全高清卧室家庭影院1080P投影电视大屏智能投影机 微影Q8（待机蓝牙音箱-支持侧投高清）

京东秒杀 距离结束 22 : 25 : 45

秒 杀 价 ¥599.00 [~~¥999.00~~] 降价通知

累计评价 2600+

促 销

增值业务 以旧换新，卖了换钱

配 送 至 [redacted] 有货 免运费

由 京东 发货，微影旗舰店 提供售后服务. 18:00前下单，预计明天(08月21日)送达

服务支持 放心购 送运费险 | 闪电退款

预约送货 部分收货 货到付款 自提

京东服务

1 + -

加入购物车

温馨提示 · 支持7天无理由退货

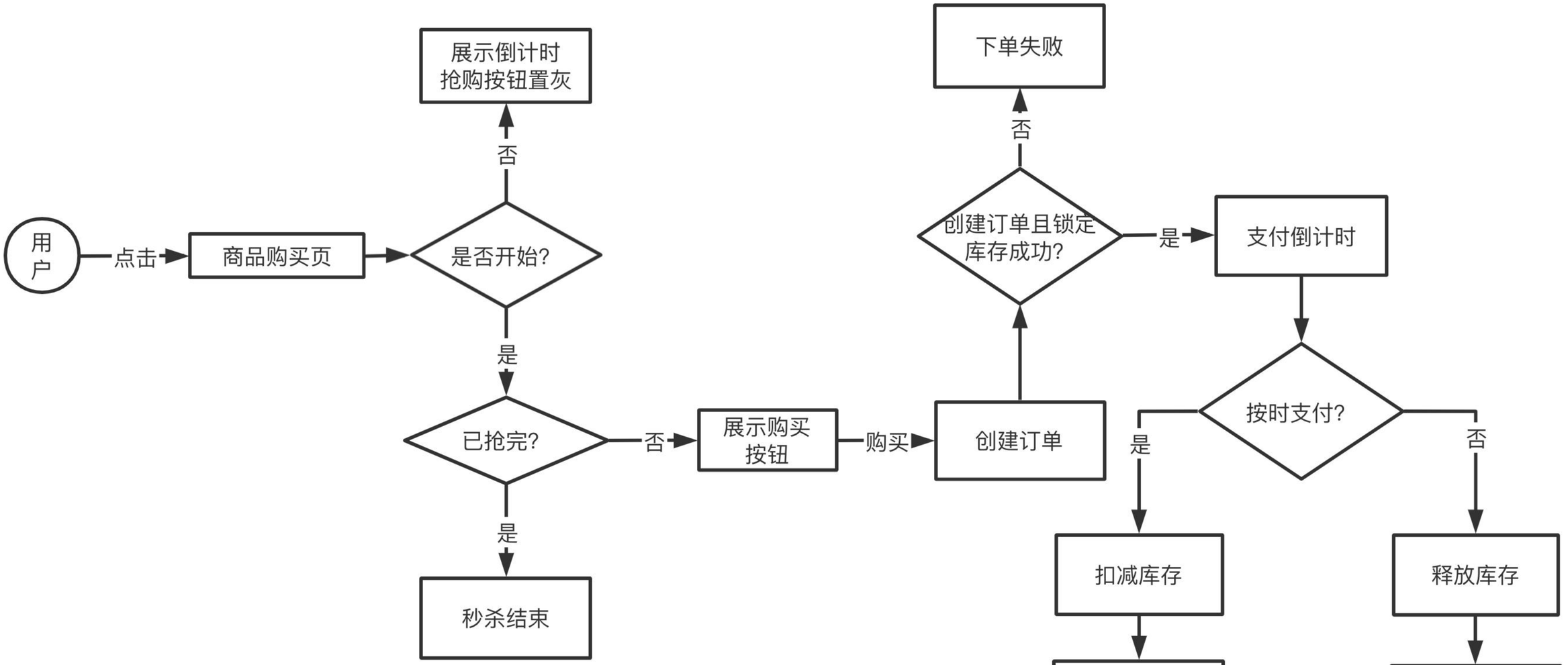
秒杀系统场景 - QPS 分析

平日每秒 1000 人访问该页面。

秒杀时每秒数10万人访问该页面。

QPS 增加 100 倍以上。

商品购买和下单流程



秒杀系统需要解决问题

瞬时大流量高并发	服务器、数据库等能承载的 QPS 有限，如数据库一般是单机 1000 QPS。需要根据业务预估并发量。
有限库存，不能超卖	库存是有限的，需要精准地保证，就是卖掉了 N 个商品。不能超卖，当然也不能少卖了。
黄牛恶意请求	使用脚本模拟用户购买，模拟出十几万个请求去抢购。
固定时间开启	时间到了才能购买，提前一秒都不可以（以商家「京东」「淘宝」的时间为准）。
严格限购	一个用户，只能购买 1 个或 N 个。

需求拆解

商家侧（京东自营、淘宝天猫店家）

- 新建秒杀活动
- 配置秒杀活动

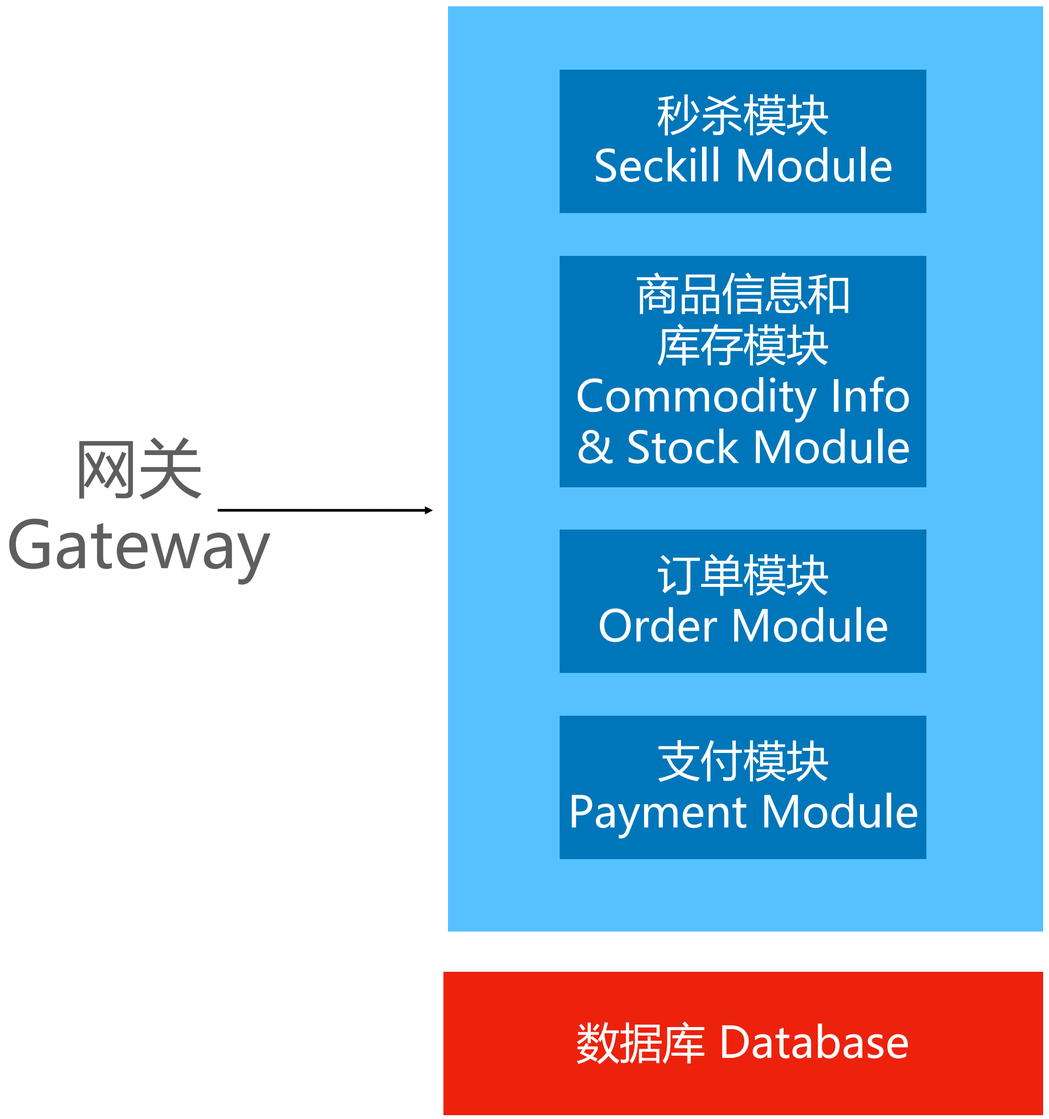
用户侧

- 商品秒杀页面（前端或客户端）
- 购买
- 下单
- 付款

Service 服务

单体架构 or 微服务?

服务结构设计 - 单体架构



前后端耦合(Coupling)，服务压力较大。

各功能模块耦合严重。

系统复杂，一个模块的升级需要导致整个服务都升级。

扩展性差，难以针对某个模块单独扩展。

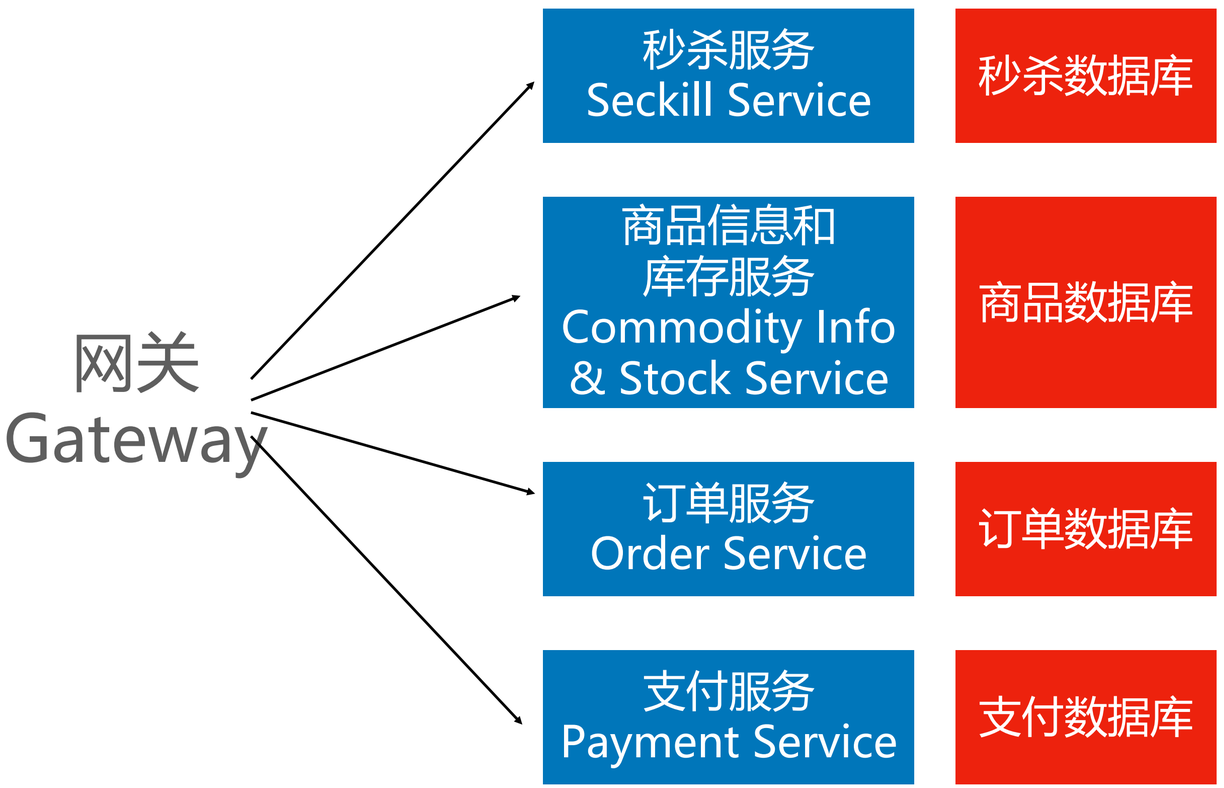
开发协作困难，各个部门的人都在开发同一个代码仓库。

级联故障(Cascading failure)，一个模块的故障导致整个服务不可用。

陷入某种单一技术和语言中。

数据库崩溃导致整个服务崩溃。

服务结构设计 - 微服务 (MicroService)



各功能模块解耦，保证单一职责。

系统简单，升级某个服务不影响其他服务。

扩展性强。可对某个服务进行单独扩容或缩容。

各个部门协作明晰。

故障隔离。某个服务出现故障不完全影响其他服务。

可对不同的服务选用更合适的技术架构或语言。

数据库独立，互不干扰。

Service 存储

数据如何存储与访问

1. Select 为每个 Service 选择存储结构

2. Schema 细化表结构

数据库表设计

商品信息表

commodity_info

商品id id	商品名称 name	商品描述 desc	价格 price
189	iPhone 11 64G	xxxxxxxx	5999

库存信息表

stock_info

库存id id	商品id commodity_id	活动id seckill_id	库存 stock	锁定 lock
1	189	0	1000000	0
2	189	28	100	5

秒杀活动表

seckill_info

秒杀id id	秒杀名称 name	商品id commodity_id	价格 price	数量 number
28	618 iPhone 11 64G 秒杀	189	4000	100

订单信息表

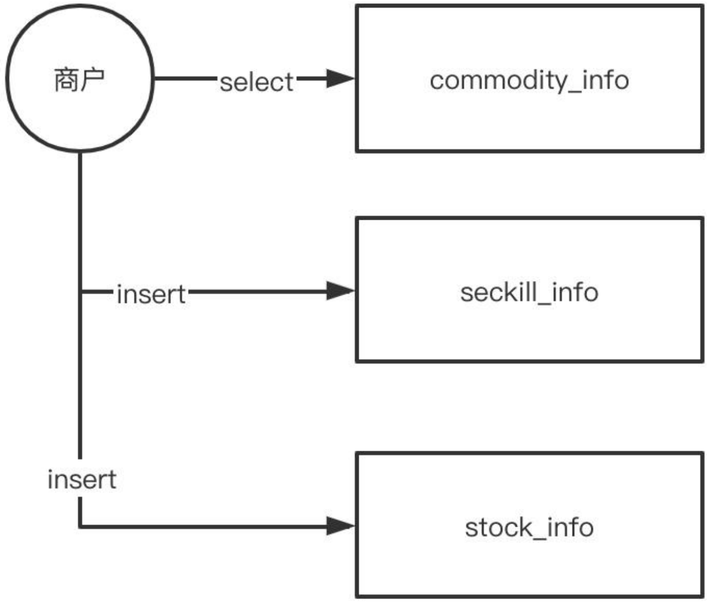
order_info

订单id id	商品id commodity_id	活动id seckill_id	用户id user_id	是否付款 paid
1	189	28	Jack	1

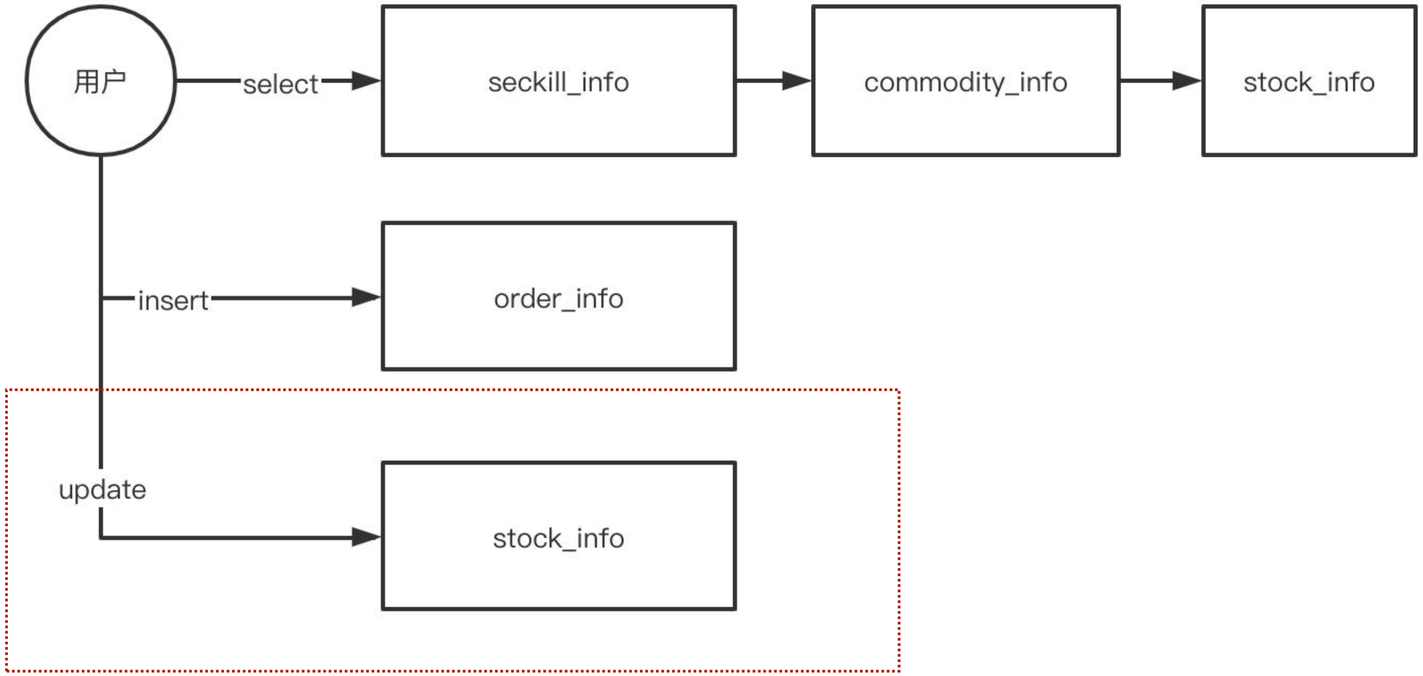
如何添加索引(Index)?

数据流

商家侧（京东自营、淘宝天猫店家）



用户侧



秒杀操作 - 扣减库存

读取判断库存，然后扣减库存

1. 查询库存余量

```
SELECT stock FROM `stock_info`
WHERE commodity_id = 189 AND seckill_id = 28;
```

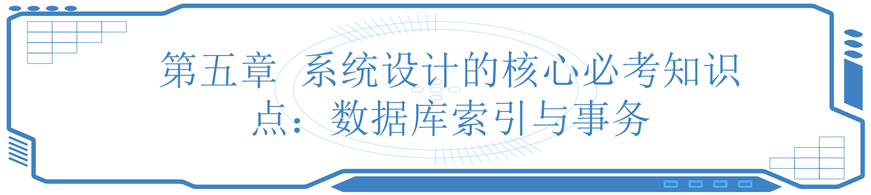
2. 扣减库存

```
UPDATE `stock_info` SET stock = stock - 1
WHERE commodity_id = 189 AND seckill_id = 28;
```

并发导致超卖的问题如何解决?

秒杀操作 - 扣减库存

读取和判断过程中加上**事务**



1. 事务开始

```
START TRANSACTION;
```

2. 查询库存余量，并锁住数据

```
SELECT stock FROM `stock_info`
WHERE commodity_id = 189 AND seckill_id = 28 FOR UPDATE;
```

3. 扣减库存

```
UPDATE `stock_info` SET stock = stock - 1
WHERE commodity_id = 189 AND seckill_id = 28;
```

4. 事务提交

秒杀操作 - 扣减库存

使用 UPDATE 语句自带的行锁

1. 查询库存余量

```
SELECT stock FROM `stock_info`
WHERE commodity_id = 189 AND seckill_id = 28;
```

2. 扣减库存

```
UPDATE `stock_info` SET stock = stock - 1
WHERE commodity_id = 189 AND seckill_id = 28 AND stock > 0;
```

超卖问题解决了，其他问题呢?

1. 大量请求都访问 MySQL，导致 MySQL 崩溃。

对于抢购活动来说，可能几十万人抢 100 台 iPhone，实际大部分请求都是无效的，不需要下沉到 MySQL。

秒杀操作 - 库存预热

秒杀的本质，就是对库存的抢夺。

每个秒杀的用户来都去数据库查询库存校验库存，然后扣减库存，导致数据库崩溃。

MySQL 数据库单点能支撑 1000 QPS，但是 Redis 单点能支撑 10万 QPS，可以考虑将库存信息加载到 Redis 中。
直接通过 Redis 来判断并扣减库存。

Redis 简介

一种主要将数据存储于内存中的非关系型的键值对数据库 (NoSQL 的一种)，但也可以将数据持久化 (Data Persistence) 到磁盘中。

支持多种数据非关系型的数据结构。

1.字符串/数字 (STRING)
2.哈希表 (HASH)
3.链表 (LIST)
4.集合 (SET)
5.有序集合 (ZSET)

单线程的数据库。通过IO多路复用实现并发。

支持数据的主备容灾 (Disaster Tolerance) 存储。

所有单个指令操作都是原子的，即要么完全执行成功，要么完全执行失败。多个指令也可以通过 Lua 脚本事务操作实现原子性。

因为都在内存中操作，性能极高，单机一般可支撑 10万数量级的 QPS。

可用作数据缓存 (Cache)、数据持久存储和消息队列 (Message Queue)。

什么时候进行预热 (Warm-up)?

活动开始前

语法：SET KEY VALUE

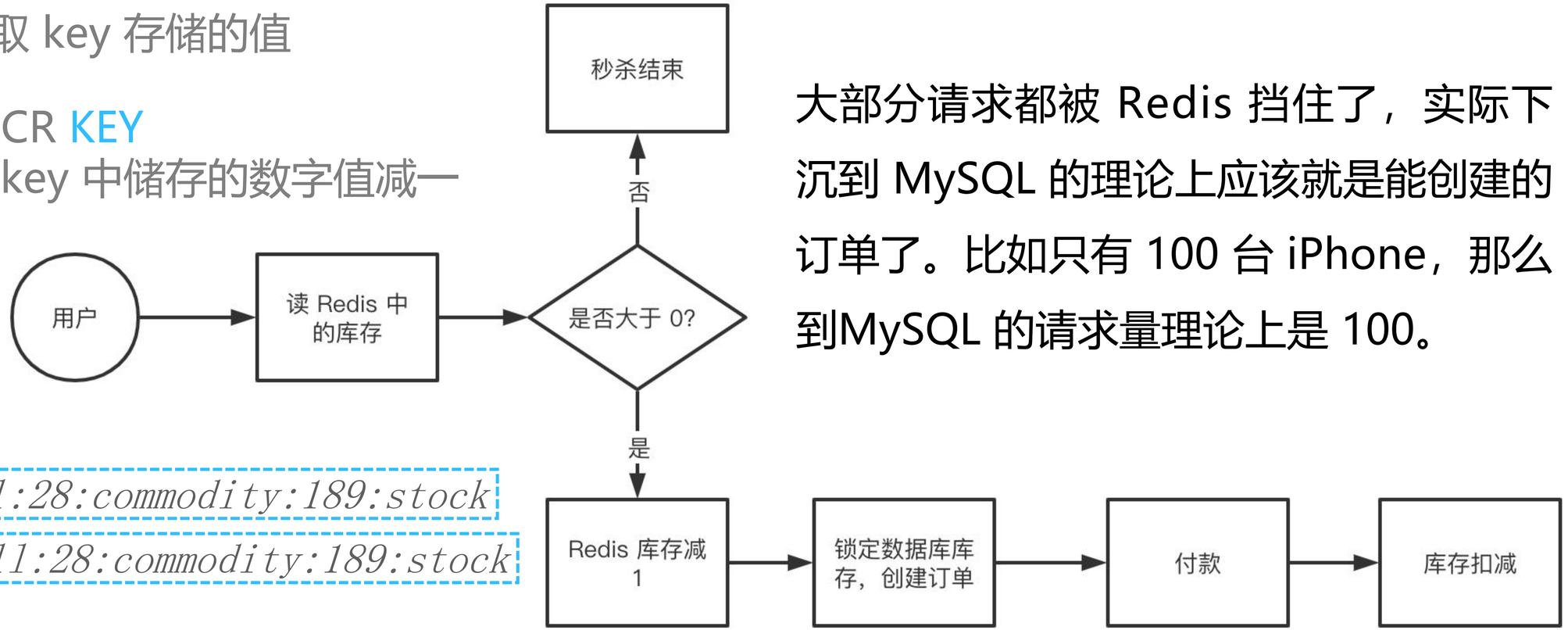
作用：设置指定 key 的值

SET seckill:28:commodity:189:stock 100

秒杀操作 - 通过 Redis 扣减库存

语法: GET KEY
作用: 获取 key 存储的值

语法: DECR KEY
作用: 将 key 中储存的数字值减一

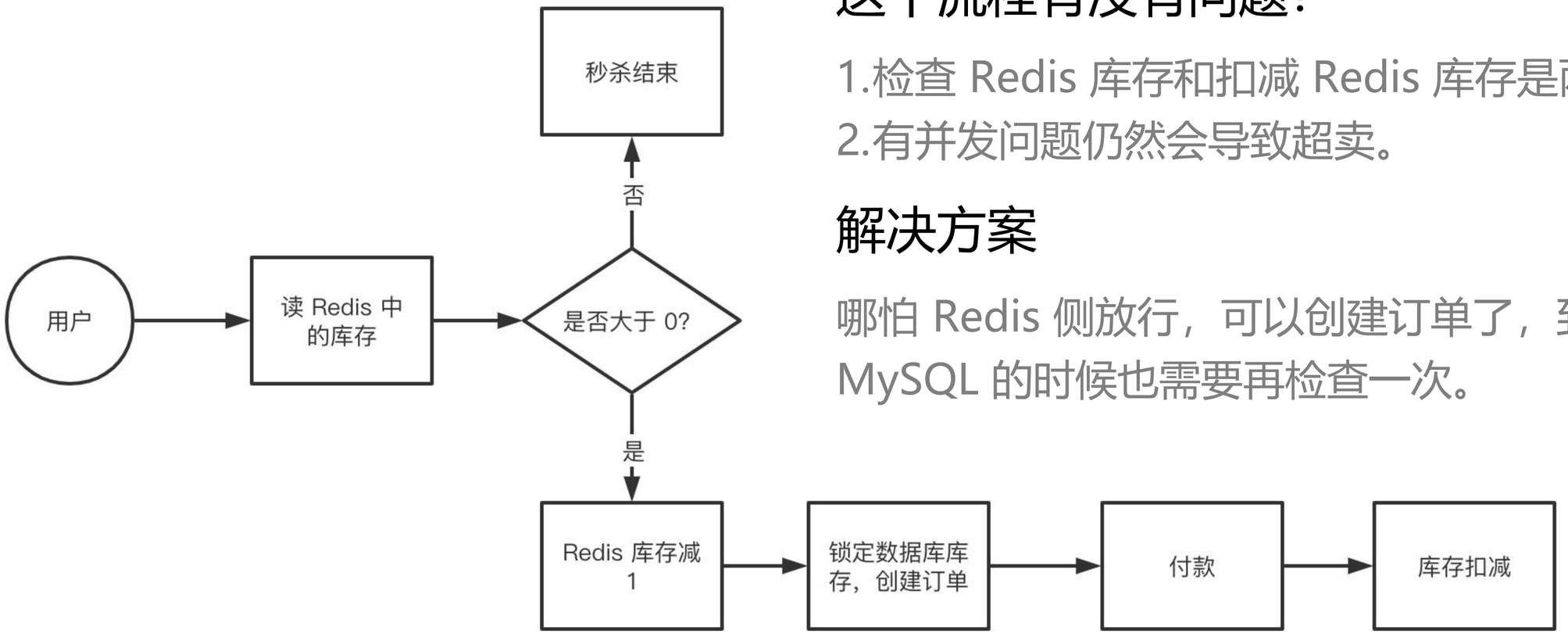


GET seckill:28:commodity:189:stock

DECR seckill:28:commodity:189:stock

大部分请求都被 Redis 挡住了，实际下沉到 MySQL 的理论上应该就是能创建的订单了。比如只有 100 台 iPhone，那么到MySQL 的请求量理论上是 100。

秒杀操作 - 通过 Redis 扣减库存



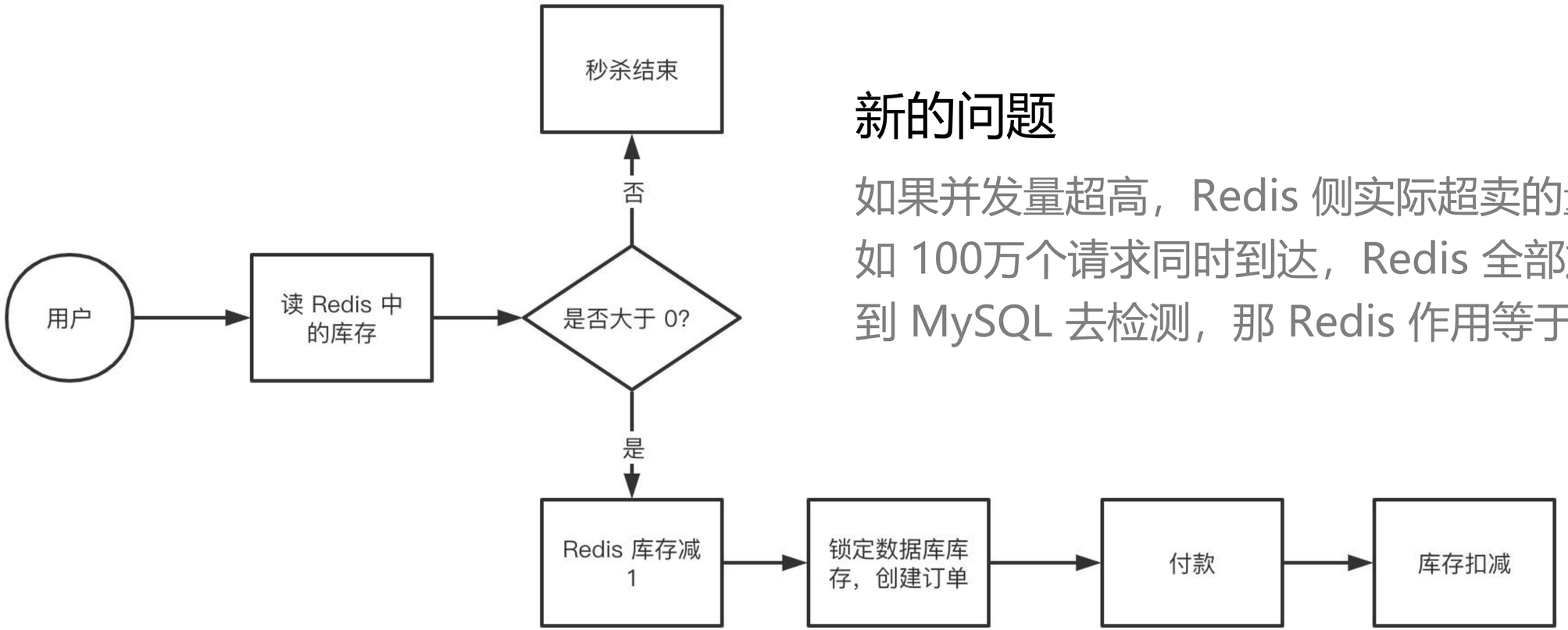
这个流程有没有问题?

1.检查 Redis 库存和扣减 Redis 库存是两步操作。
2.有并发问题仍然会导致超卖。

解决方案

哪怕 Redis 侧放行，可以创建订单了，到 MySQL 的时候也需要再检查一次。

秒杀操作 - 通过 Redis 扣减库存



新的问题

如果并发量超高，Redis 侧实际超卖的量过大，如 100万个请求同时到达，Redis 全部放行。再到 MySQL 去检测，那 Redis 作用等于没有。

秒杀操作 - 通过 Redis 扣减库存

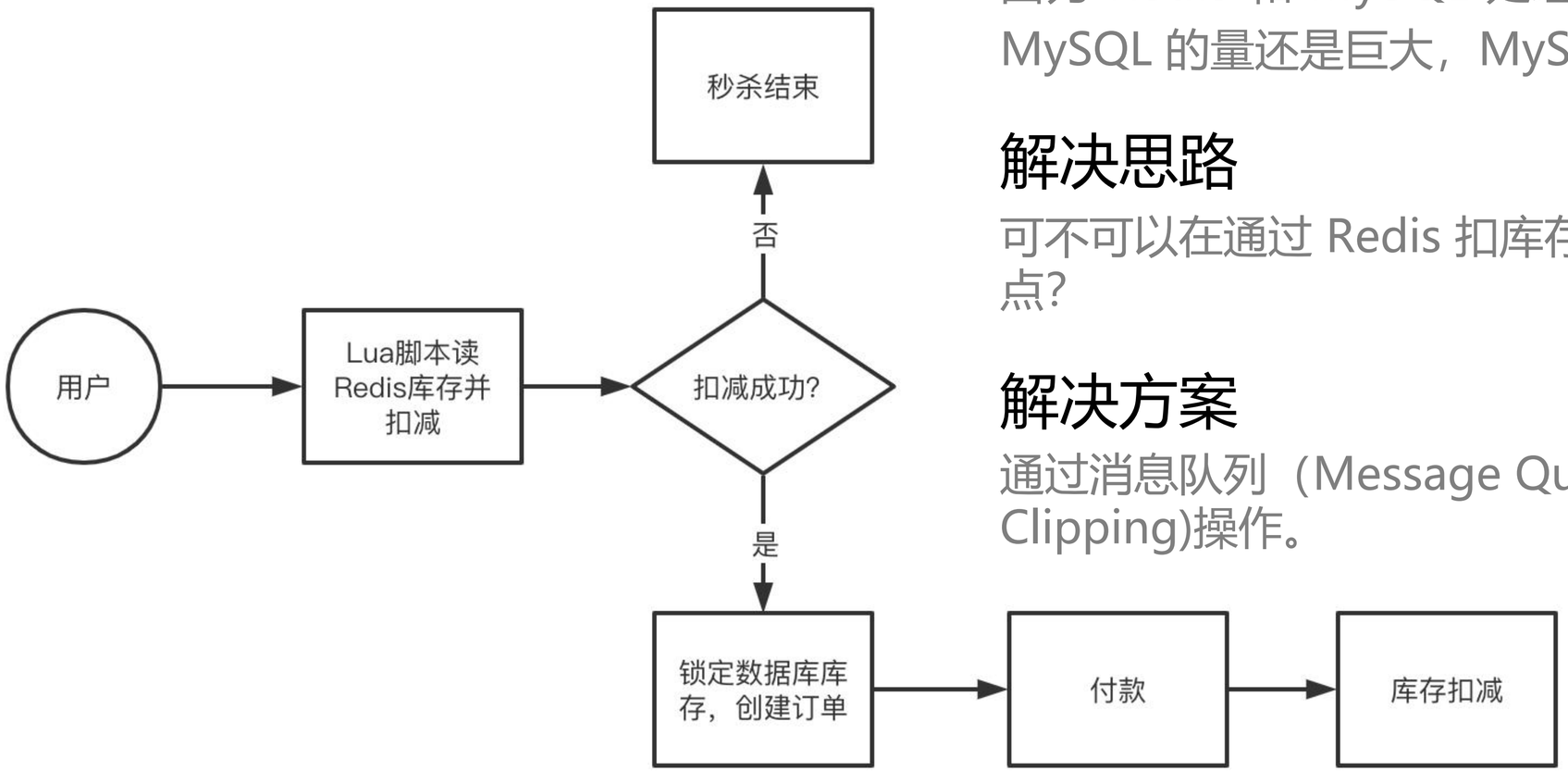

```
if (redis.call('exists', KEYS[1]==1)) then
    local stock = tonumber(redis.call('get',KEYS[1]));
    if (stock <= 0) then
        return -1
    end;
    redis.call('decr',KEYS[1]);
    return stock - 1;
end;
return -1;
```

通过 Lua 脚本执行原子操作

Lua 脚本功能是 Reids 在 2.6 版本中推出， 通过内嵌对 Lua 环境的支持，Redis 解决了长久以来不能高效地处理 CAS （check-and-set）命令的缺点， 并且可以通过组合使用多个命令， 轻松实现以前很难实现或者不能高效实现的模式。

Lua 脚本是类似 Redis 事务，有一定的原子性，不会被其他命令插队，可以完成一些 Redis 事务性的操作。

秒杀操作 - 通过 Redis 扣减库存



如果秒杀数量是1万台，或者10万台呢?

因为 Redis 和 MySQL 处理能力的巨大差异。实际下沉到 MySQL 的量还是巨大，MySQL 无法承受。

解决思路

可不可以在通过 Redis 扣库存后，到 MySQL 的请求慢一点?

解决方案

通过消息队列（Message Queue，MQ）进行削峰(Peak Clipping)操作。

消息队列简介

一类基于生产者/消费者模型的组件。

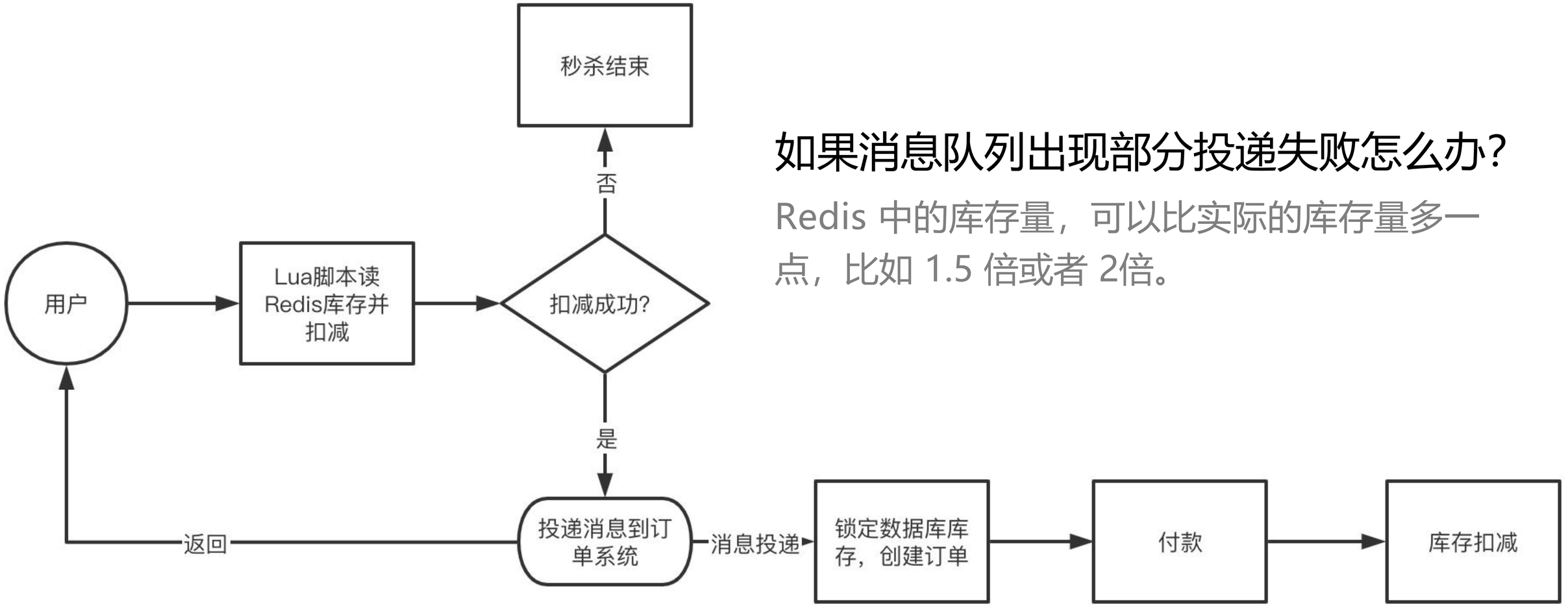
用于实现两个不同的系统之间的解耦和异步 (Asynchronous) 操作。

生产者可以高速地向消息队列中投递（生产）消息。

消费者可以按照自己的节奏去消费生产者投递的消息。

消息队列一般带有重试的能力。可以持续投递，直到消费者消费成功。

秒杀操作 - 通过消息队列异步地创建订单



如果消息队列出现部分投递失败怎么办?

Redis 中的库存量，可以比实际的库存量多一点，比如 1.5 倍或者 2倍。

库存扣减时机

下单时立即减库存。

用户体验最好，控制最精准，只要下单成功，利用数据库锁机制，用户一定能成功付款。

可能被恶意下单。下单后不付款，别人也买不了了。

先下单，不减库存。实际支付成功后减库存。

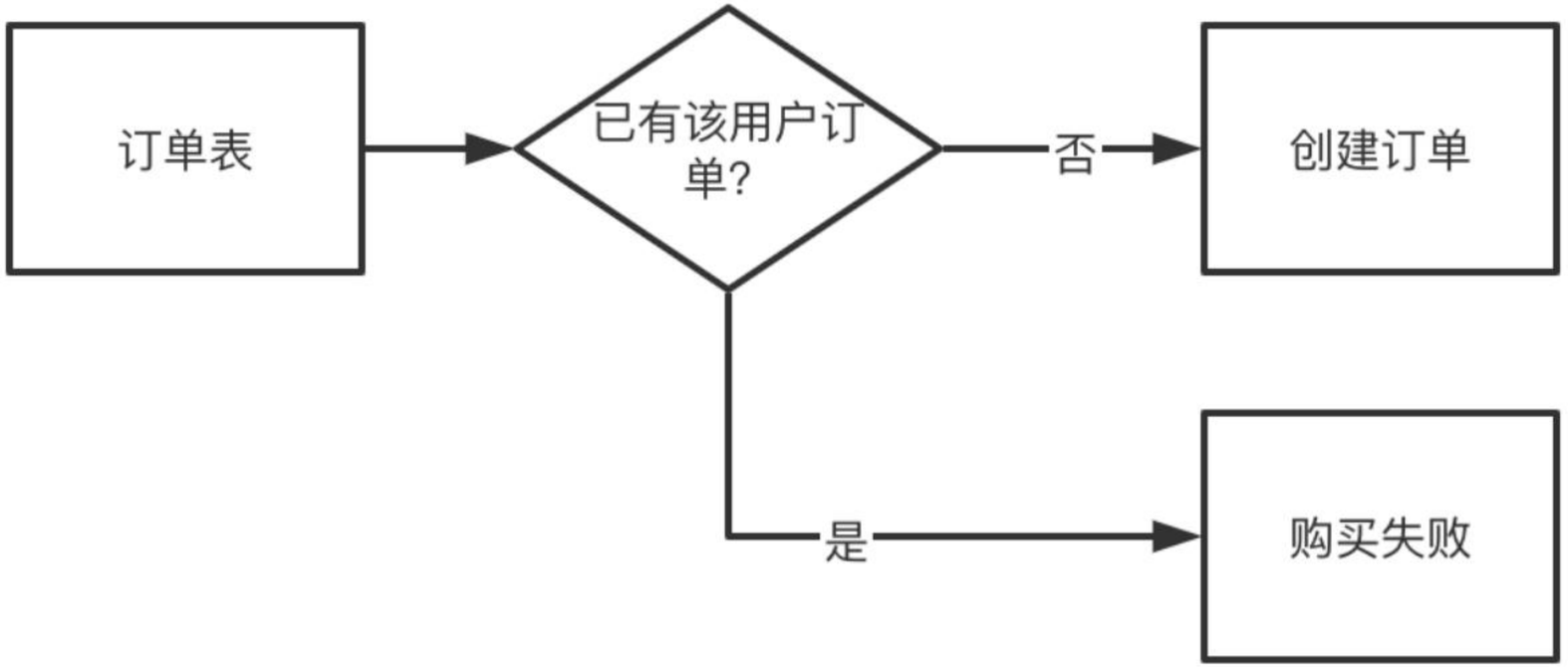
可以有效避免恶意下单。

对用户体验极差，因为下单时没有减库存，可能造成用户下单成功但无法付款。

下单后锁定库存，支付成功后，减库存。

如何限购

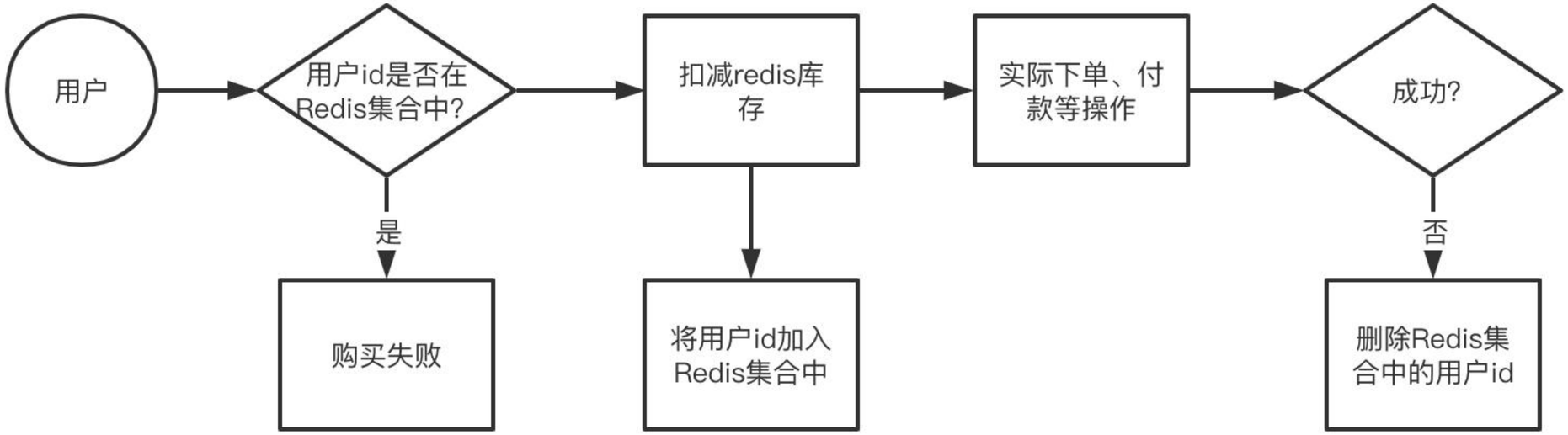
MySQL 数据校验



如何限购

Redis 数据校验

使用 Redis 提供的集合数据结构，将扣减 Redis 库存的用户 ID 写入。



语法: SADD KEY VALUE1.....VALUEN

作用: 将一个或多个成员元素加入到集合中，已经存在于集合的成员元素将被忽略。

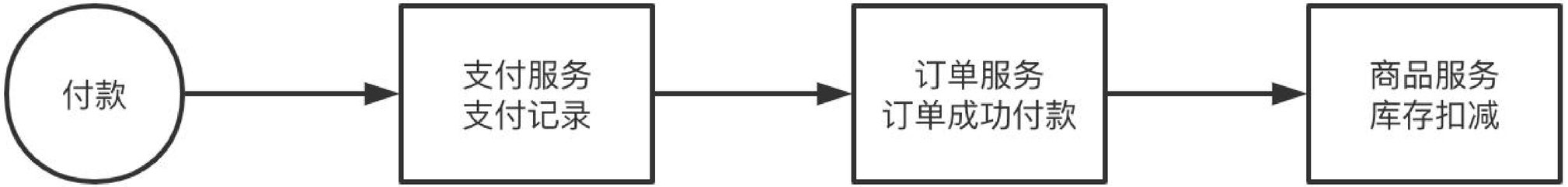
```
SADD seckill:28:commodity:189:user uid1
```

语法: SISMEMBER KEY VALUE

作用: 判断成员元素是否是集合的成员。

```
SISMEMBER seckill:28:commodity:189:user uid1
```

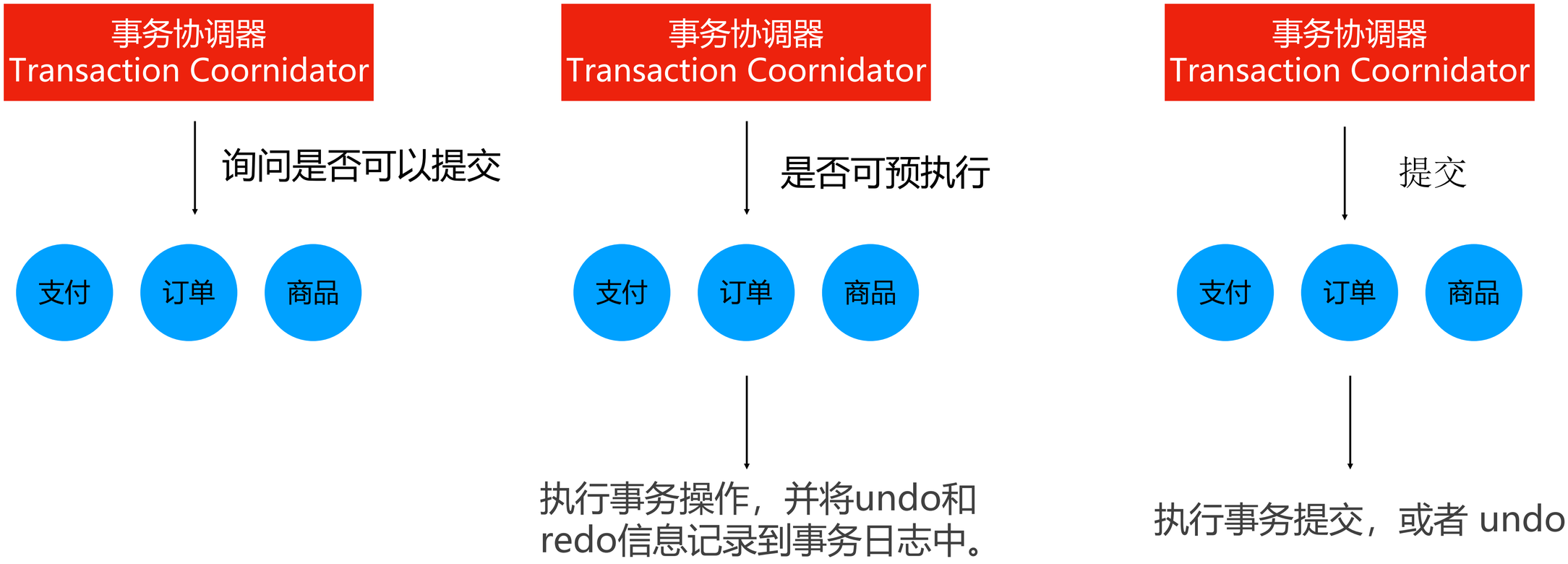
付款和减库存的数据一致性



付款和减库存的数据一致性 - 分布式事务

保证多个存在于不同数据库的数据操作，要么同时成功，要么同时失败。主要用于强一致性的保证。

三阶段提交，有超时机制



Scale 拓展

如何优化系统

加分项

是否有遗漏什么功能?

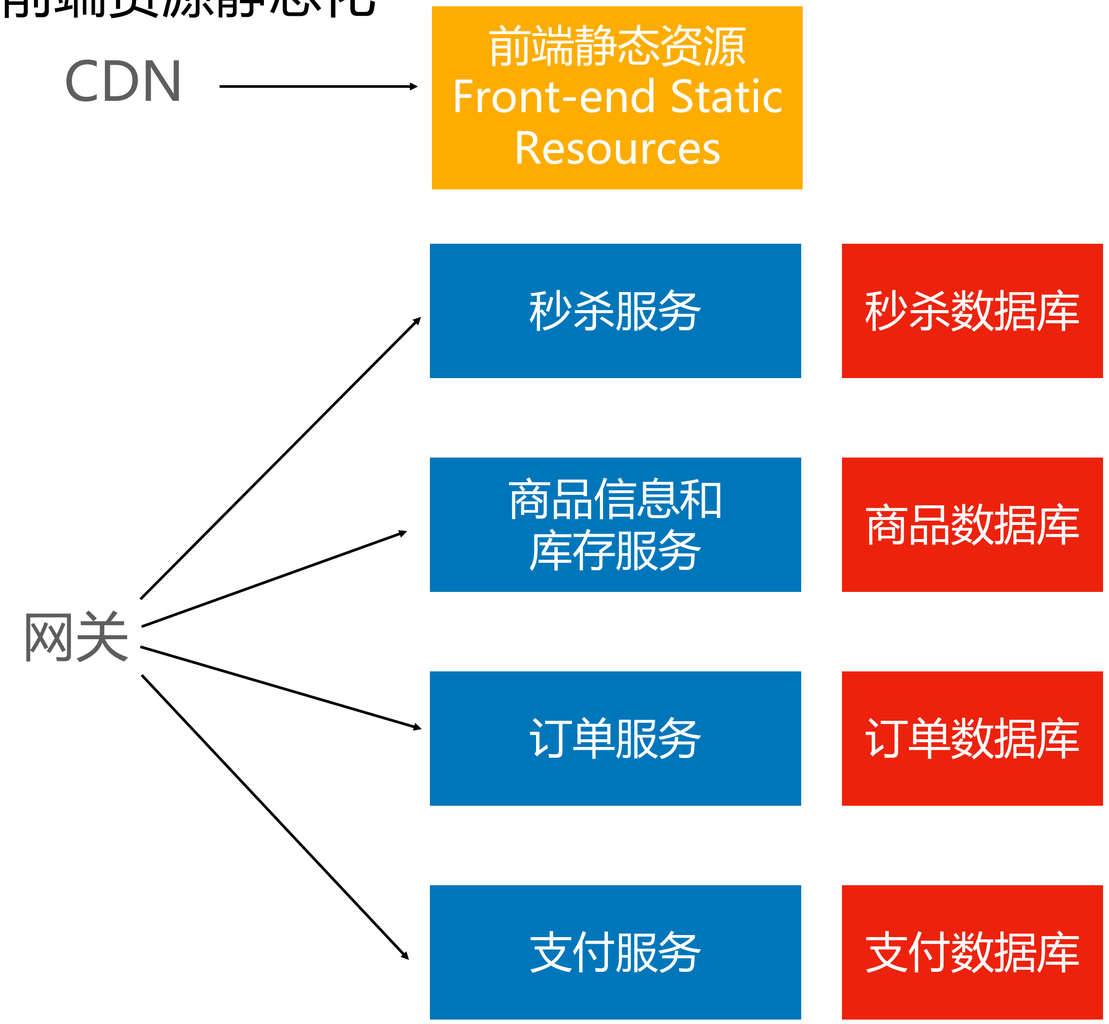
可能十万人抢购 100 台 iPhone，大部分请求是无效的。

Redis 能力高过 MySQL，但能力还是有限。

Redis 库存扣减完毕后，是否后面的请求可以直接拒绝了?

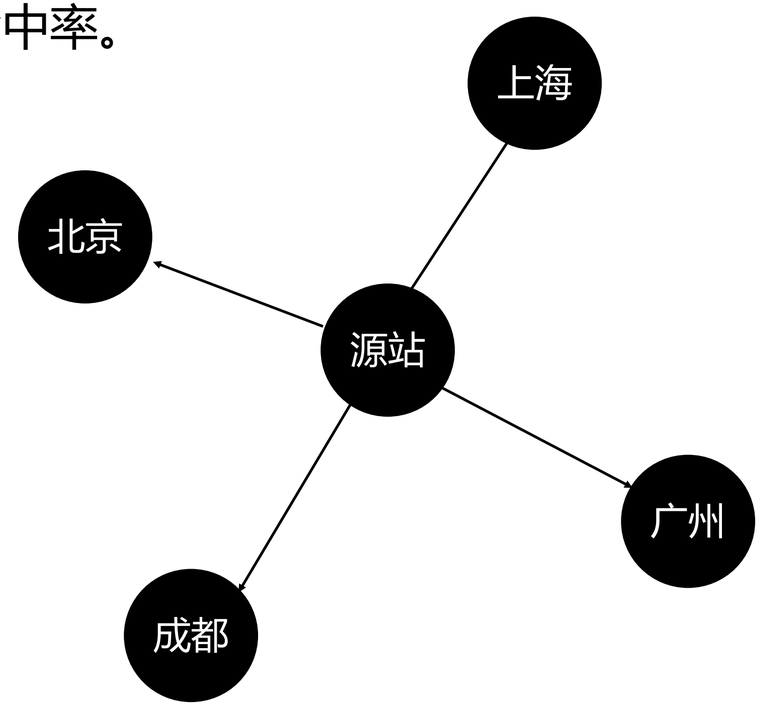
防止刷爆商品页面

前端资源静态化



CDN 的全称是 Content Delivery Network，即内容分发网络。

CDN 是依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。



防止刷爆商品页面

前端限流

点击一次后，按钮
短时间内置灰

部分请求直接
跳转到「繁忙页」

防止刷爆商品页面

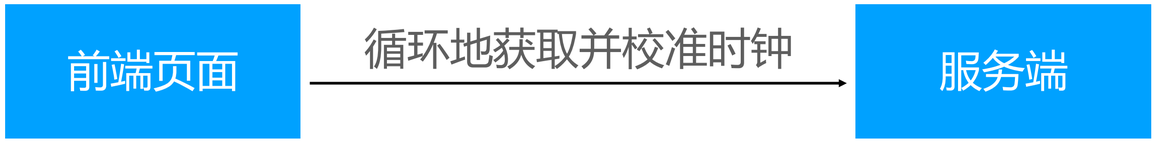
未开始抢购时，禁用抢购按钮。

如何计算倒计时?

1. 打开页面获取活动开始时间，然后前端页面开始倒计时

2. 打开页面获取距离活动开始的时间差，然后前端页面开始倒计时

3. 前端轮询 (Poll) 服务器的时间，并获取距离活动开始的时间差



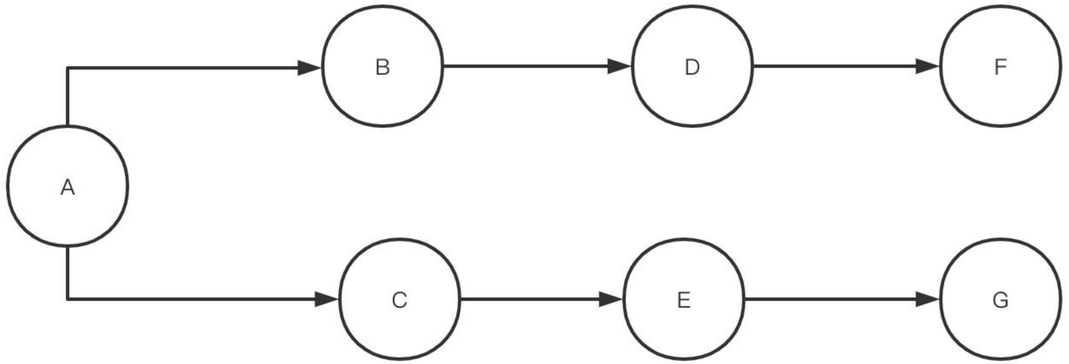
秒杀服务器挂掉，怎么办？

尽量不要影响其他服务，尤其是非秒杀商品的正常购买。

服务雪崩 (Avalanche)

多个微服务之间调用的时候，假设 微服务A 调用 微服务B 和 微服务C，微服务B 和微服务C 又调用其他的微服务，这就是所谓的”扇出 (Fan-out)”，如扇出的链路上某个微服务的调用响应式过长或者不可用，对 微服务A 的调用就会占用越来越多的系统资源，进而引起系统雪崩，所谓的”雪崩效应”。

服务雪崩效应是一种因“服务提供者”的不可用导致“服务消费着”的不可用并将这种不可用逐渐放大的过程。



秒杀服务器挂掉，怎么办？

尽量不要影响其他服务，尤其是非秒杀商品的正常购买。

服务熔断 (Fuse or Circuit-breaker)

熔断机制是应对雪崩效应的一种微服务链路保护机制，当扇出链路的某个微服务不可用或者响应时间太长时，熔断该节点微服务的调用，快速返回”错误”的响应信息。当检测到该节点微服务响应正常后恢复调用链路。

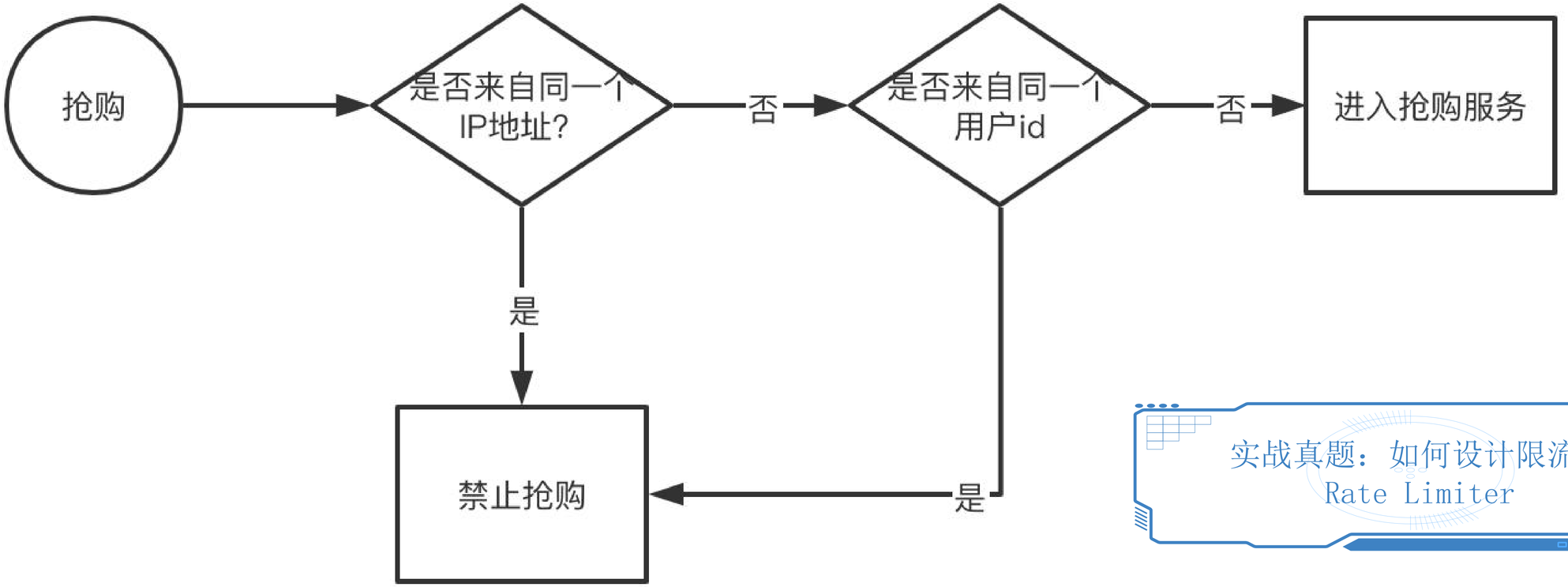
Netflix Hystrix
Alibaba Sentinel

防止恶意刷请求或者爬虫请求

验证码机制
Verification Code Mechanism

防止恶意刷请求或者爬虫请求

限流机制
Ratelimit Mechanism



实战真题：如何设计限流器
Rate Limiter

防止恶意刷请求或者爬虫请求

黑名单机制
Blacklist Mechanism

1.黑名单 IP 地址

2.黑名单用户ID

秒杀系统 vs 订票系统

在业务上，他们有哪些差异?

100 台 iPhone 没有区别
但是 100 张同一车次的火车票，有座位的区别（暂时忽略一等座二等座等）

车次信息表

train_number_info

车次id id	车次 number	车次描述 desc	价格 price
189	G1100	xxxxxxxx	100

库存信息表

stock_info

库存id id	商品id commodity_id	座位号 seat	库存 stock	锁定 lock
1	189	1A	1	0
2	189	1B	1	1

系统架构设计试听福利群

该二维码7天内(8月24日前)有效，重新进入将更新

扫描二维码进入
试听活动群

任何问题请加
圆圆