优惠券系统设计

Coupon System

主讲人 南帝老师

Keywords

Message Queue
分布式事务 表单设计 消息队列 索引
Touch System
SQL Table Design 优惠券 API Ratelimit
缓存预热 过期券通知
Distributed Transaction Coupon
触达系统
Expired conpon notification

Scenario 场景

设计一个优惠券系统 Coupon System

优惠券的种类

优惠券系统的核心流程

电商系统的促销手段

优惠券 Coupon

满减券

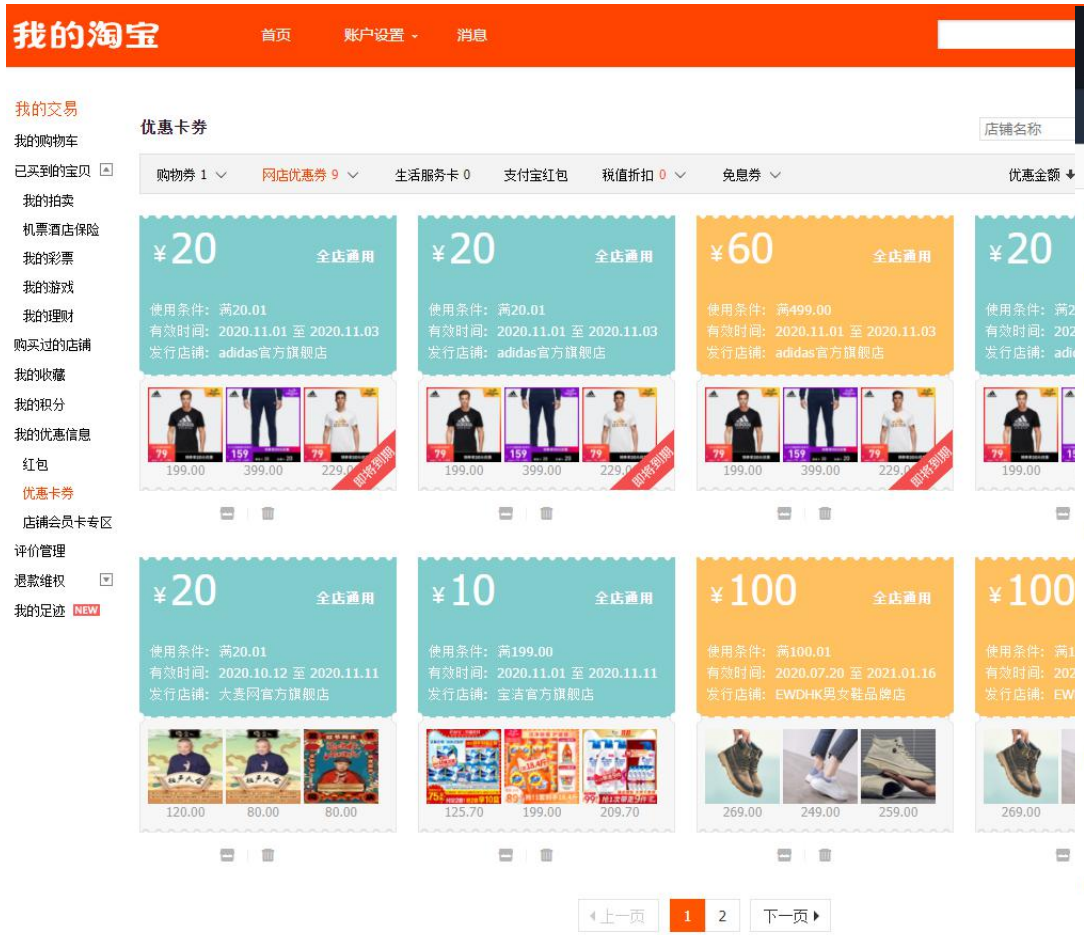
直减券

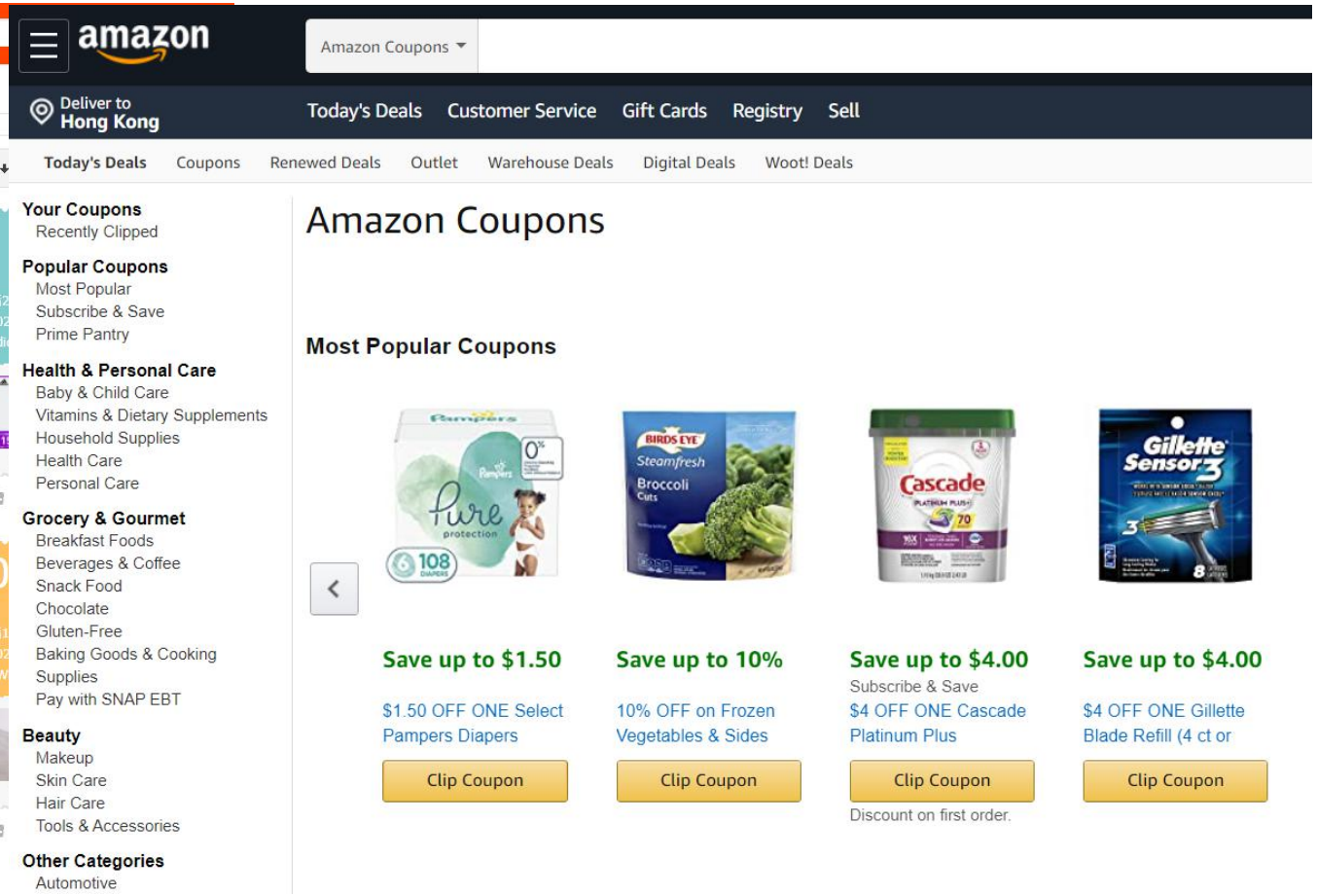
折扣券

。。。

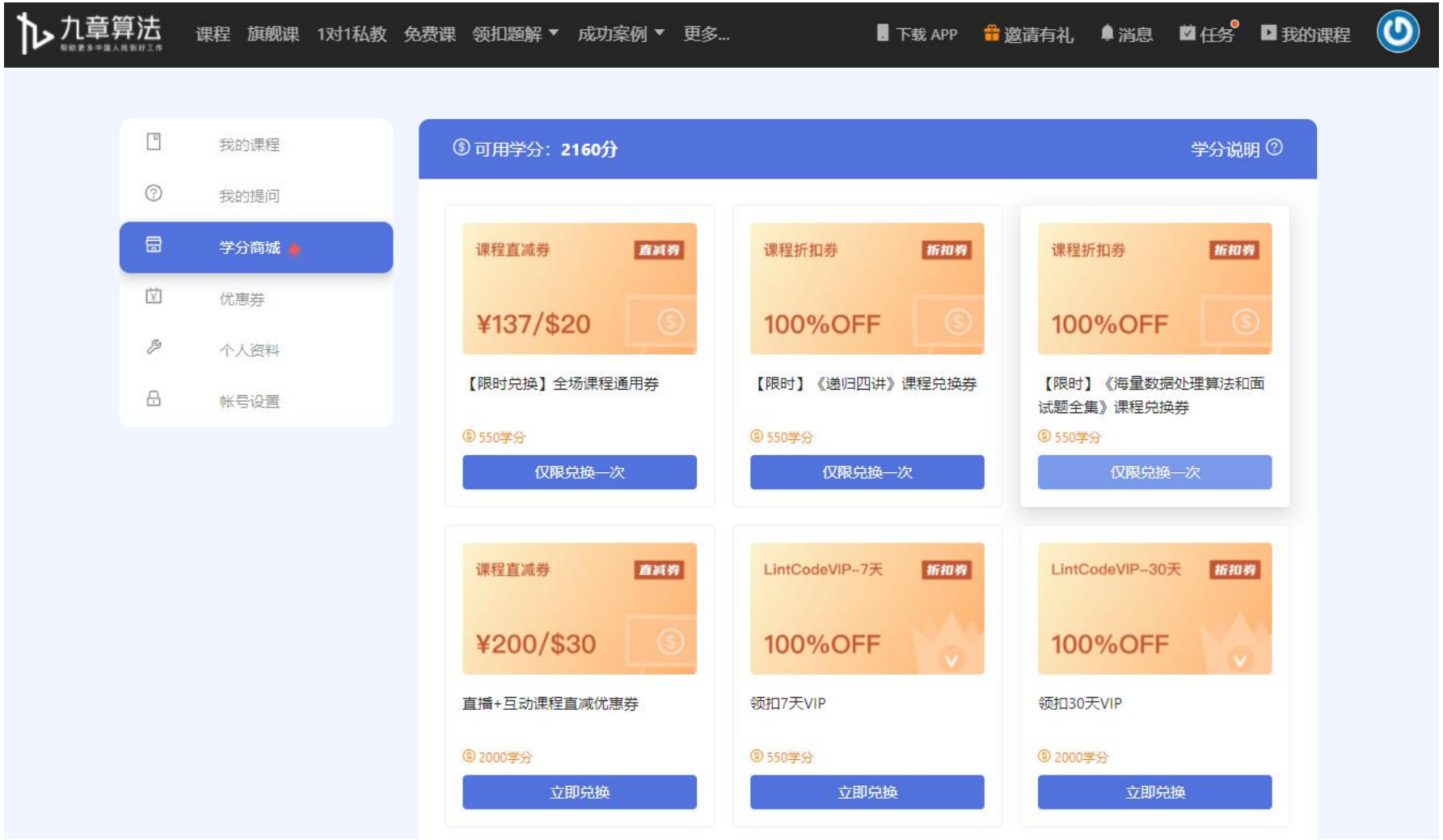
淘宝 & Amazon 优惠券

Coupon





Coupon
九章优惠券



Coupon

滴滴 & Uber 优惠券

优惠券核心流程

发券的方式：同步发送 or 异步发送

谁能领：所有用户 or 指定的用户

领取上限：一个优惠券最多能领取多少张？

领取方式：用户主动领取 or 自动发放被动领取

作用范围：商品、商户、类目

计算方式：是否互斥、是否达到门槛等

需求拆解

商家侧

Coupon 创建优惠券

Coupon 发送优惠券

用户侧

Coupon 领取优惠券

下单

Coupon 使用优惠券

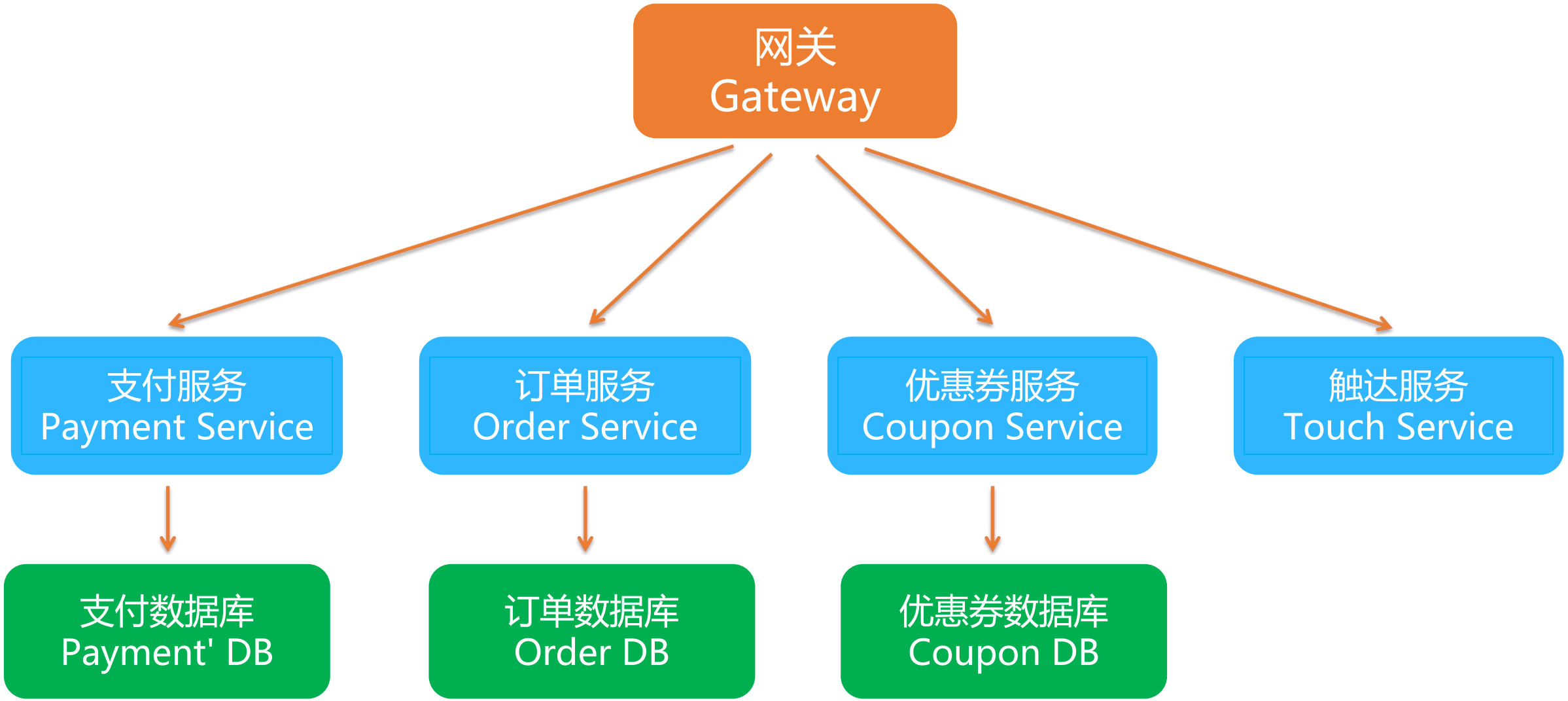
支付

Service服务

了解优惠券系统 Coupon System 相关服务

优惠券系统设计难点

服务结构设计



优惠券系统难点

Storage存储

模型的设计

优惠券系统 Coupon System 模型定义

优惠券系统的难点精讲

重要概念

coupon_batch

1.券批次（券模板）

Coupon Coupon

指一批优惠券的抽象、模板，包含优惠券的大部分属性。

例：商家创建了一批优惠券，共1000张，使用时间为202011-11 00:00:00 ~ 2020-11-11 23:59:59，规定只有化妆品类目商品才能使用，满100减50。

2.券

发放到用户的一个实体，已与用户绑定。

Coupon Coupon

例：将某批次的优惠券中的一张发送给某个用户，此时优惠券属于用户。

3.规则

优惠券的使用有规则和条件限制，比如满100减50券，需要达到门槛金额 100元才能使用

券批次表 coupon_batch Table

字段 column	类型 type	解释 explain	例子 example
batch_id	INTEGER	批次ID	1111
batch_name	INTEGER	批次名称	“双十一发送券”
coupon_name	VARCHAR	券名称	“劳斯莱斯5元代金券”
rule_id	INTEGER	规则，外键	1010
total_count	INTEGER	总数量	10000
assign_count	INTEGER	已发放券数量	5000
used_count	INTEGER	已使用数量	2500

规则表 rule Table

字段 column	类型 type	解释 explain	例子 example
rule_id	INTEGER	规则id，自增	1010
name	VARCHAR	名称	“满减规则”
type	INTEGER	优惠券类型，0—满减，1—折扣	0
rule_content	BLOB	规则内容	'{ threshold: 100 // 使用门槛 amount: 10 // 优惠金额 ..... }'

规则内容

```
'{
    threshold: 5.01                          // 使用门槛
    amount: 5                                // 优惠金额
    use_range: 3                             // 使用范围，0—全场，1—商家，2—类别，3—商品
    commodity_id: 10                         // 商品 id
    receive_count: 1                         // 每个用户可以领取的数量
    is_mutex: true                           // 是否互斥，true 表示互斥，false 表示不互斥
    receive_started_at: 2020-11-1 00:08:00   // 领取开始时间
    receive_ended_at: 2020-11-6 00:08:00     // 领取结束时间
    use_started_at: 2020-11-1 00:00:00       //  使用开始时间
    use_ended_at: 2020-11-11 11:59:59        //  使用结束时间
}'
```

优惠券表 coupon Table

字段 column	类型 type	解释 explain	例子 example
coupon_id	INTEGER	券ID，主键	66889
user_id	INTEGER	用户id	1001
batch_id	INTEGER	批次ID	1111
status	INTEGER	状态，0—未使用，1—已使用，2—已过期，3—冻结	1
order_id	VARCHAR	对应订单ID	12168257647382907847
received_time	DATETIME	领取时间	2020-11-07 00:12:48
validate_time	DATETIME	有效日期	2020-11-18 00:01:48
used_time	DATETIME	使用时间	2020-11-11 00:01:48

优惠券系统 — 建券

1. 新建规则

```
INSERT INTO rule (name, type, rule_content)
VALUES(“满减规则”, 0, '{
    threshold: 100
    amount: 10
    ......
}');
```

商家建券

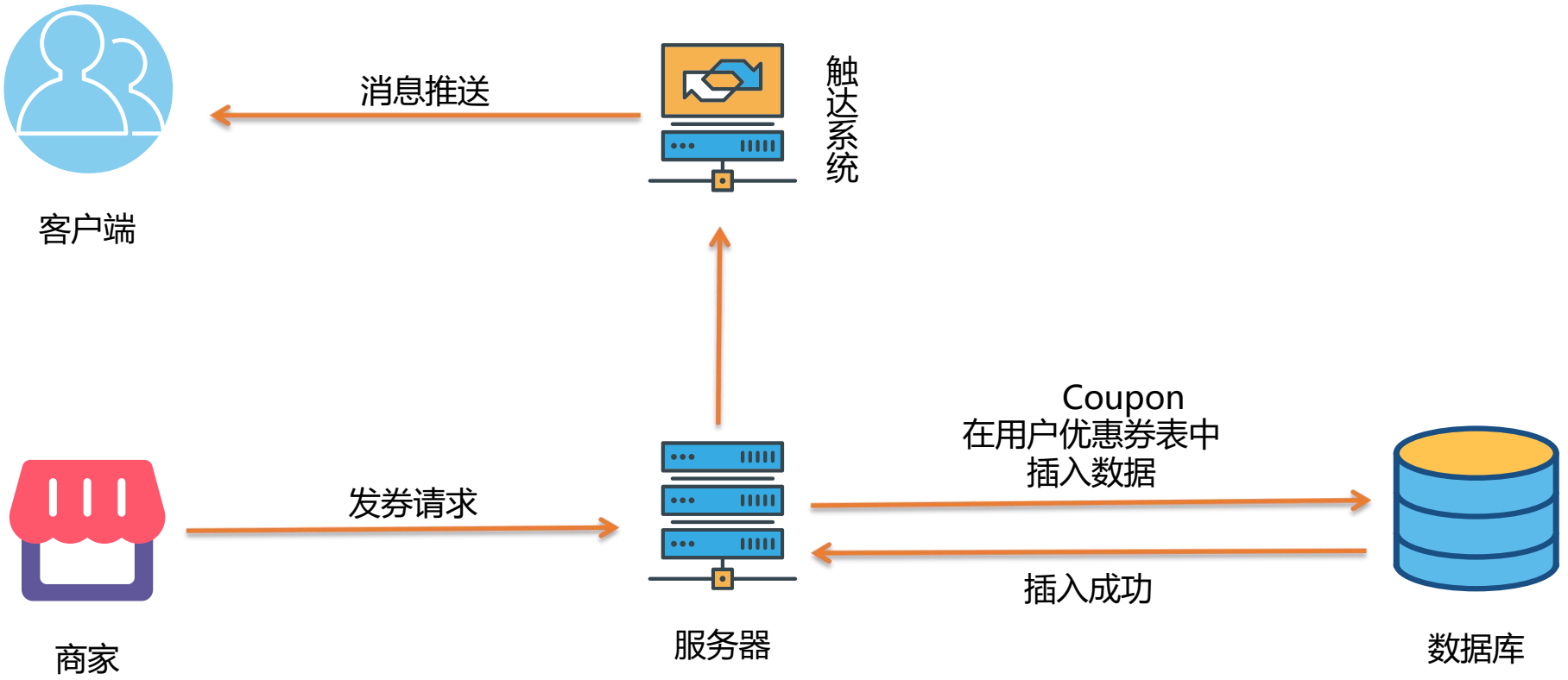
Coupon

2. 新建优惠券批次

```
INSERT INTO coupon_batch (coupon_name, rule_id, total_count )
VALUES(“劳斯莱斯5元代金券”, 1010, 10000);
```

优惠券系统 — 发券

优惠券系统 — 发券

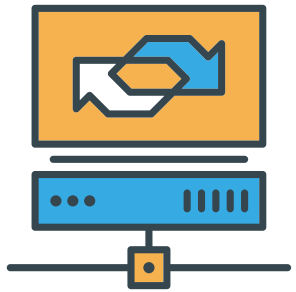


如何给大量用户发券?

asynchronous

异步发送

优惠券系统 — 触达系统



触达系统

1. 短信
2. 邮件

可以通过调用第三方接口的方式实现

3. 站内信　　通过数据库插入记录来实现

信息表 message Table

字段 column	类型 type	解释 explain	例子 example
id	INTEGER	信息id	1111
send_id	INTEGER	发送者id	
rec_id	INTEGER	接受者id	
content	VARCHAR	站内信内容	
is_read	INTEGER	是否已读	
send_time	DATETIME	发送时间	

我们先考虑用户量很少的情况，如果商家要给所有人发站内信，则先遍历用户表，再按照用户表中的所有用户依次将站内信插入到 message 表中。这样，如果有100个用户，则群发一条站内信要执行100个插入操作。

如果系统的用户数量增加到上万人呢?

这样发一条站内信，就要重复插入上万条数据。而且这上万条数据的 content 是一样的，假设一条站内信占 100 个字节，发一次站内信就要消耗十几 MB。因此我们可以将原来的表拆成两个表。

信息表 message Table

字段 column	类型 type	解释 explain	例子 example
id	INTEGER	信息id	1111
send_id	INTEGER	发送者id	
rec_id	INTEGER	接受者id	
message_id	INTEGER	外键，信息内容id	
is_read	INTEGER	是否已读	

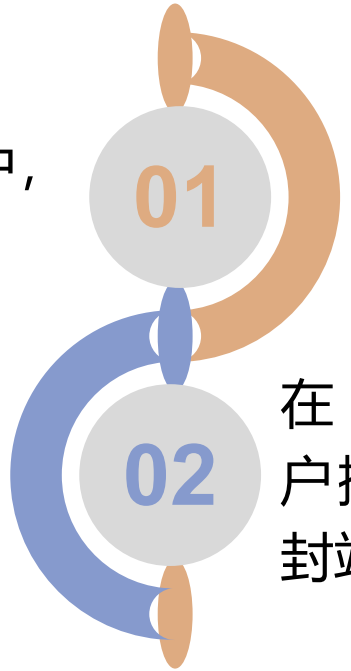
信息内容表 message_content Table

字段 column	类型 type	解释 explain	例子 example
id	INTEGER	信息内容id	1111
content	VARCHAR	内容	
send_time	DATETIME	发送时间	

优惠券系统 — 触达系统

在发一封站内信的时候，执行两步操作。

在 message_content 表中，插入站内信的内容



在 message 表中给所有的用户插入一条记录，标识有一封站内信。

如果系统的用户数量增加到上千万人呢?

- 这里会有非活跃用户的问题，假设注册用户一千万，其中活跃用户只占其中的20%。
- 如果采用上面拆分成两个表的情况，发一封“站内信”，那得执行一千万个插入操作。可能剩下的80%用户基本都不会再登录了，我们只需要对其中20%的用户插入数据即可。

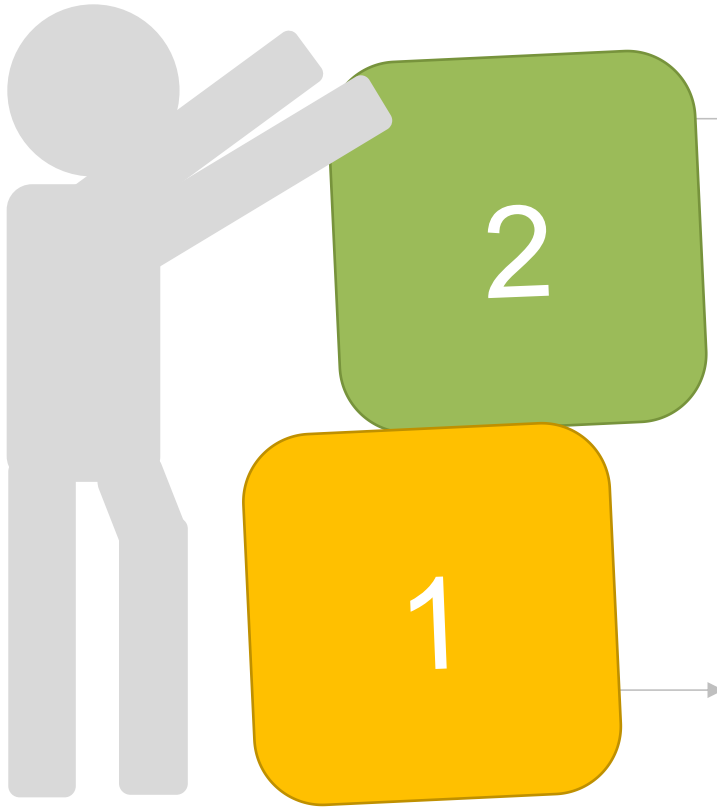
信息表 message Table

字段 column	类型 type	解释 explain	例子 example
id	INTEGER	信息id	1111
rec_id	INTEGER	接受者id	
message_id	INTEGER	外键，信息内容id	
is_read	INTEGER	是否已读	

信息内容表 message_content Table

字段 column	类型 type	解释 explain	例子 example
id	INTEGER	信息内容id	1111
send_id	INTEGER	发送者id	
content	VARCHAR	内容	
send_time	DATETIME	发送时间	

优惠券系统 — 触达系统



系统

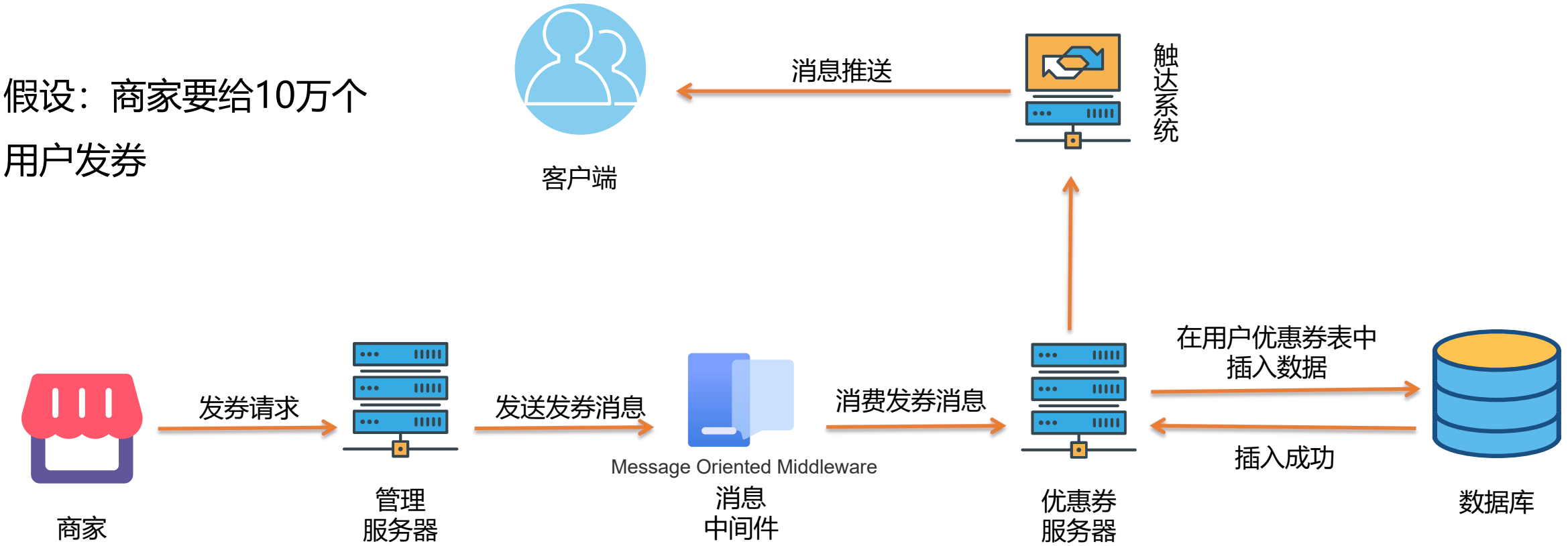
发站内信的时候，只在 message_content 插入站内信的主体内容，message 里不插入记录。

用户

用户登录后，首先查询 message_content 中的那些没有在 message 中有记录的数据，表示是未读的站内信。在查阅站内信的内容时，再将相关的记录插入到 message 中。

优惠券系统 — 发券

假设：商家要给10万个用户发券



这样还会有什么问题?

重复消费的问题，会引发超发

优惠券系统 — 发券

1. 运营提供满足条件的用户文件，上传到发券管理后台并选择要发送的优惠券

2. 管理服务器根据用户ID和券批次ID生成消息，发送到消息中间件中

Message Oriented Middleware

商家发券

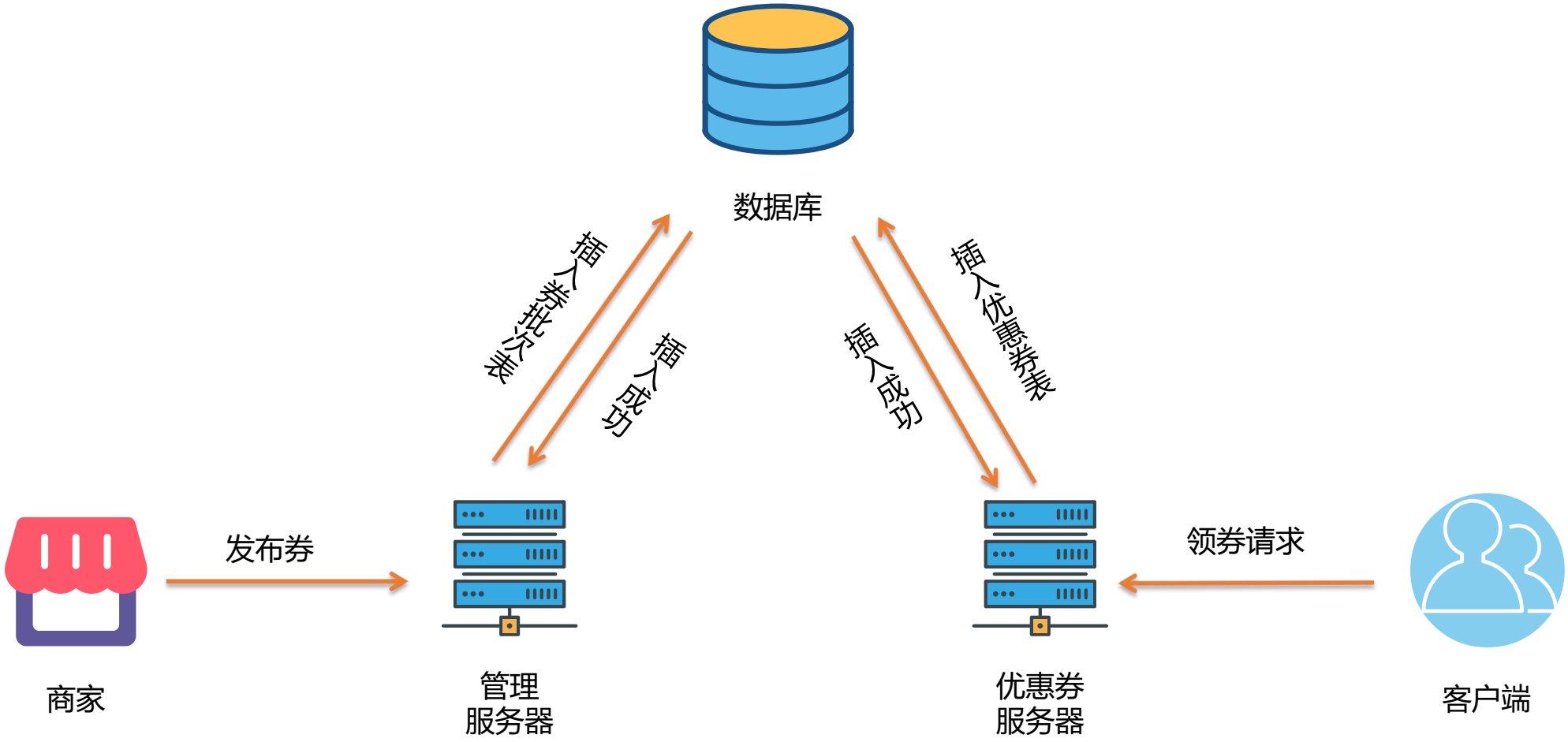
3. 优惠券服务器消费消息


```
INSERT INTO coupon (user_id, coupon_id，  batch_id)
VALUES(1001, 66889, 1111);

UPDATE coupon_batch SET total_count = total_count - 1, assign_count =
assign_count + 1
WHERE  batch_id = 1111 AND total_count > 0;
```

优惠券系统 — 领券

优惠券系统 — 领券



1. 校验优惠券余量

```
SELECT total_count FROM coupon_batch
WHERE  batch_id = 1111;
```

其实这里也会有因为并发导致的超领问题，大家可以思考一下，这里是怎么防止超领的？

2. 新增优惠券用户表，扣减余量

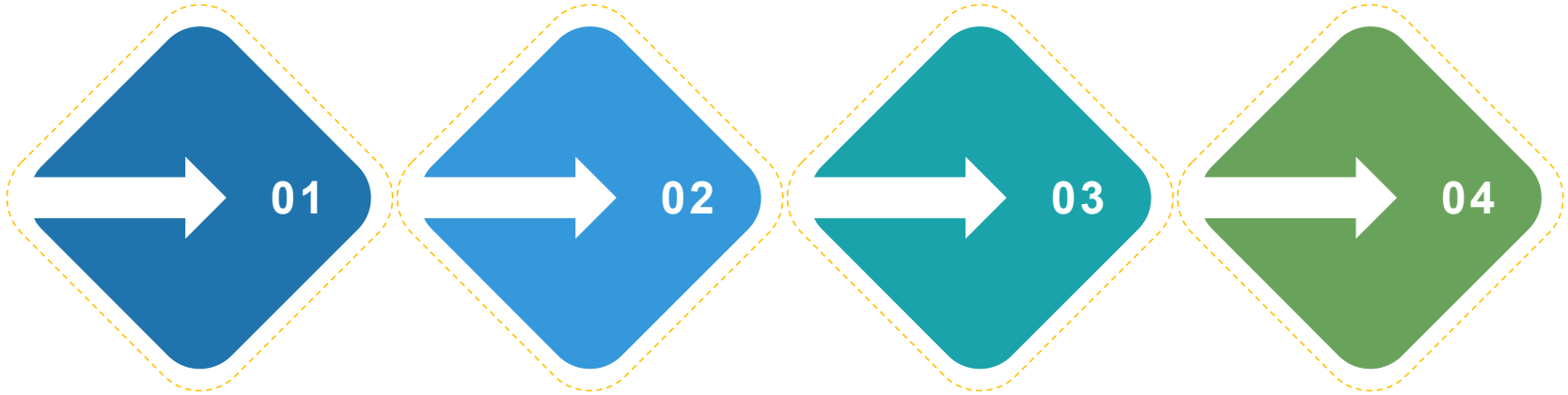
```
INSERT INTO coupon (user_id, coupon_id，  batch_id)
VALUES(1001, 66889, 1111);
```

transaction
别忘了用事务

```
UPDATE coupon_batch SET total_count = total_count - 1, assign_count =
assign_count + 1
WHERE  batch_id = 1111 AND total_count > 0;
```

用户领券

在用户领券的过程中，其实也会出现类似秒杀的情况，大家可以回想一下在秒杀场景下会有哪些问题已经我们是如何解决的？



用户领券

问题

高并发导致数据库崩溃

措施

缓存预热

问题

超高并发导致缓存放行量大使数据库崩溃

措施

message queue

消息队列异步处理

如何防止用户重复领取或多领？ Redis数据校验

1. 在领券前先查缓存

语法： SISMEMBER KEY VALUE
作用： 判断成员元素是否是集合的成员。
实例： SISMEMBER batch_id:1111:user_id 1001

2. 领券

3. 领券后更新缓存

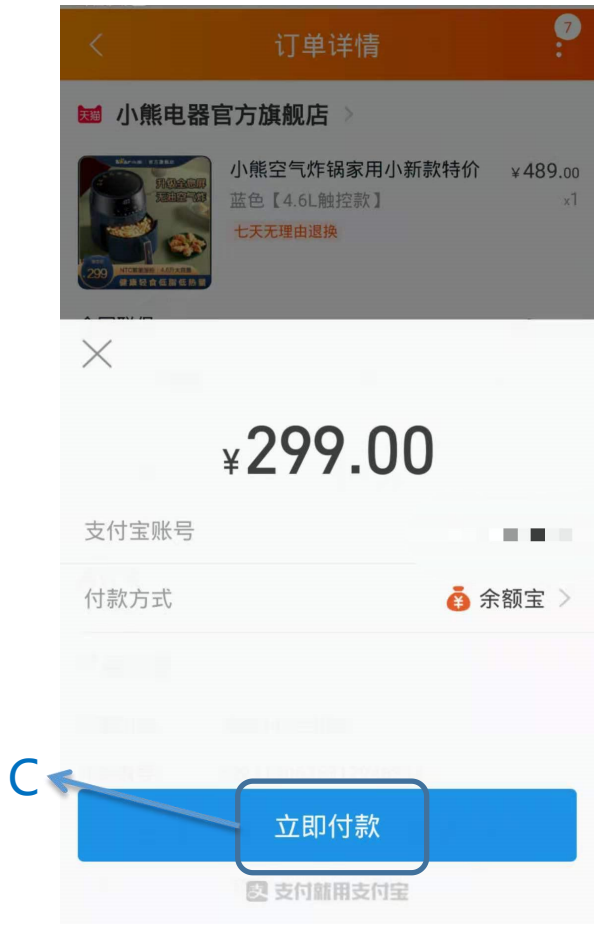
语法： SADD KEY VALUE1.....VALUEN
作用： 将一个或多个成员元素加入到集合中，已经存在于集合的成员元素将被忽略。
实例： SADD batch_id:1111:user_id 1001

什么时候对优惠券使用规则进行校验?

A. 确认订单 ✓

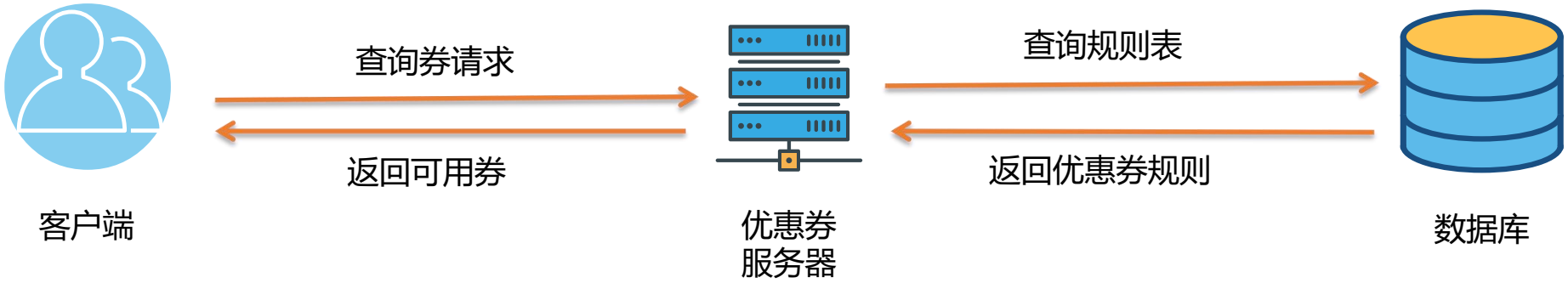
B. 提交订单

C. 立即付款



确认订单页，对优惠券进行校验

返回可用券



选择可用券

1. 返回可用券

```
SELECT batch_id FROM conpou
WHERE  user_id = 1001 AND status = 0;

SELECT rule_id FROM coupon_batch
WHERE  batch_id = 1111;

SELECT name, type, rule_content FROM rule
WHERE  rule_id = 1010;
```

用户用券

2. 选择可用券，并返回结果

优惠券系统 — 用券

阶段	系统
确认订单	券系统
提交订单	券系统、订单系统
支付订单	券系统、订单系统、支付系统

同时操作多个系统，我们要如何保证一致性？

使用分布式事务 TCC

Distributed Transaction Try-Confirm-Cancel

优惠券操作记录表
Coupon_opt_record Table

字段 column	类型 type	解释 explain	例子 example
user_id	INTEGER	用户id	1001
coupon_id	INTEGER	优惠券id	1111
operating	INTEGER	操作，0—锁定，1—核销，2—解锁	0
operated_at	DATETIME	操作时间	2020-11-11 00:01:48

Distributed Transaction

TCC是Try-Confirm-Cancel的简称，是目前分布式事务主流解决方案

TCC 实现阶段一：Try

对资源进行冻结，预留业务资源

在创建订单时，将优惠券状态改为 “冻结”

TCC 实现阶段二：Confirm

确认执行业务操作，做真正的提交，将第一步Try中冻结的资源，真正扣减

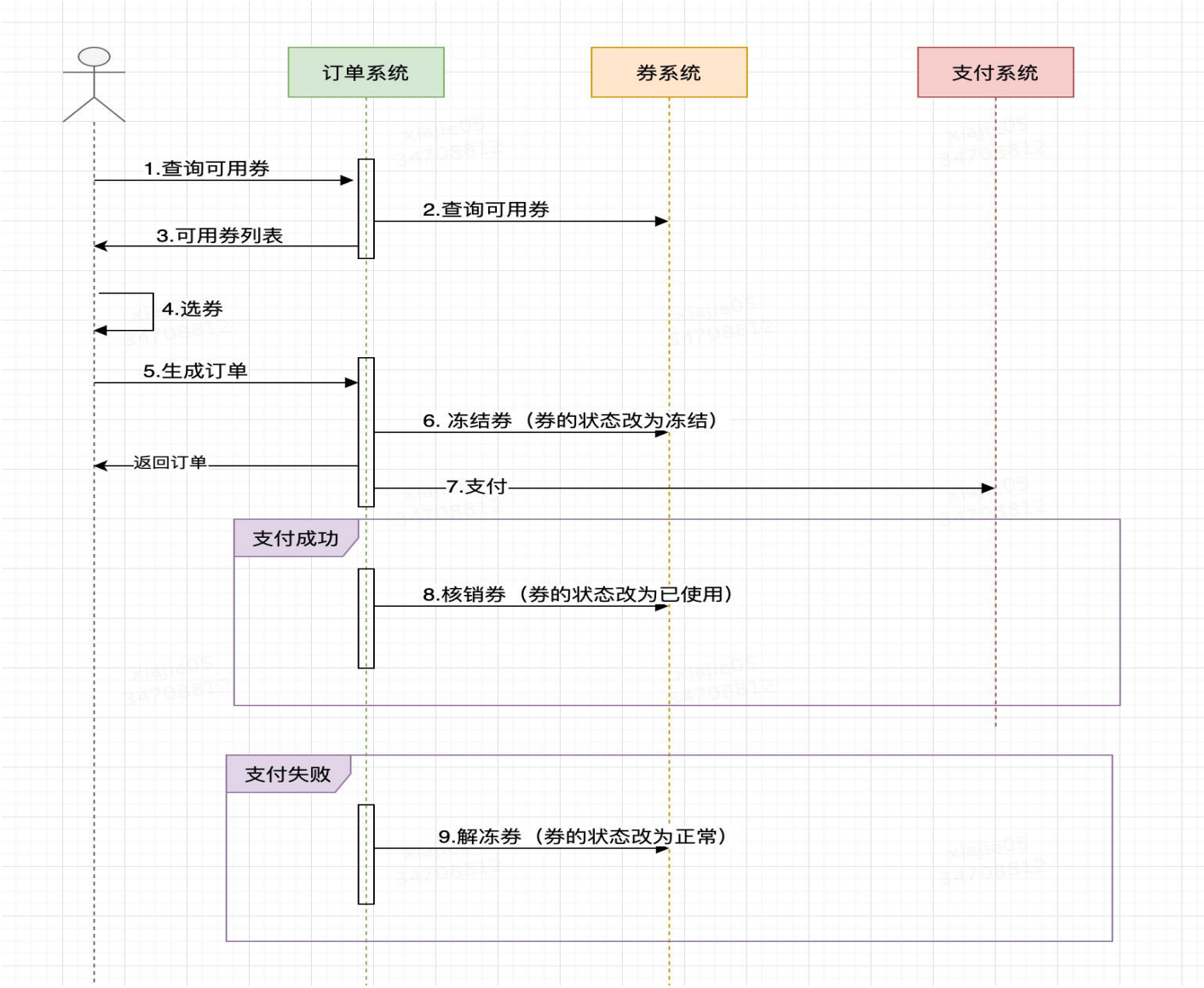
在订单支付成功，将优惠券状态改为 “已使用”

TCC 实现阶段三：Cancel

取消执行业务操作，取消Try阶段预留的业务资源

在支付失败/超时，或者订单关闭情况，将优惠券状态改为 “未使用”

优惠券系统 — 用券



Scale 扩展

快过期券提醒

数据库层面优化

发券接口限流保护

过期券提醒

定时扫描券表

1

缺点：扫描的数据量太大，随着历史数据越来越多，会影响线上主业务，会出现慢 SQL。

新增通知表

3

优点：扫描的数据量小，效率高

删除无用的已通知的数据记录

延时消息

2

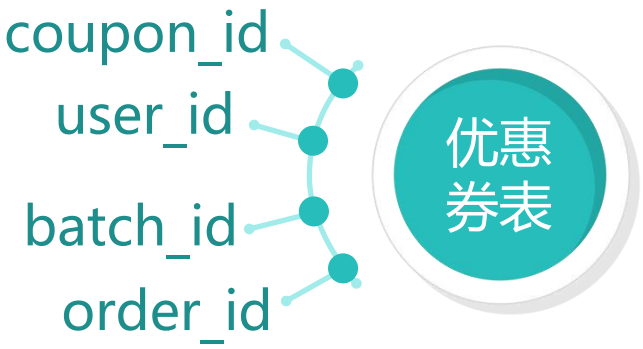
缺点：有些券的有效时间太长了(30天)以上，有可能造成大量 MQ 积压

通知信息表 Notify_msg Table

字段 column	类型 type	解释 explain	例子 example
id	INTEGER	自增主键	1100
coupon_id	BIGINT	券id	1000
user_id	BIGINT	用户id	1001
notify_day	VARCHAR	需要进行通知的日期	“2020-11-11”
notify_type	INTEGER	通知类型，1过期提醒	1
notify_time	TIMESTAMP	通知的时间，在该时间戳所在天内通知	1605059346
status	INTEGER	通知状态，0初始状态 1成功 2失败	0

过期券提醒

数据库优化——索引



发券接口限流

current limiting

前端限流

点击一次后，按钮
短时间内置灰

抢券

抢券

current limiting

后端限流

部分请求直接
跳转到「繁忙页」