# 项目介绍

一个基于微服务的分布式电商系统，需要实现购物、收藏、优惠券、评价、推送等服务

# 功能需求

买家需求，注册账号，登录账号，修改账号，浏览商品，优惠券获取，余额充值，购买商品，个人中心（基本信息，订单信息，优惠券信息，评论信息），评价商品

卖家需求，注册账号，登陆账号，修改账号，发布商品，发布优惠券，余额提现，个人中心（基本信息，订单信息，优惠券信息，评论信息）

// 管理人员需求，登陆账号，查看订单，封禁管理，个人中心（基本信息，封禁信息）

# 数据库设计

## 一个拆表思想

思考购物的流程，不难发现一定是浏览多，购物少，即读多写少，考虑将写操作发生多的地方独自做为表，

## buyerMoney和sellerMoney

买家买家的钱经常发生变化，考虑将(buyerId,buyermoney)单独做表，seller同理

## goodsCount

商品的数量在被购买的时候也会变化，考虑将他从goods表中取出来

## buyer

我们考虑到买家需要账号、密码来登陆，需要名字来作为识别、需要性别来做后期统计、需要生日来发奖品、需要电话号码、邮箱来找回密码、然后还有vip等级、积分

## seller

账号+密码+名字+vip等级

## goods

id+名字+卖家id+价格+剩余库存+分类+评价

银行卡+地址

这里用来送货和付款

## orders

他和买家、商品有关，考虑后期可以根据订单状态拆表

## comments

这里和买家、商品有关，一个评价允许出现父评价

## 收藏/购物车

是买家的，购物车注意是多个

## 优惠券

这里要分两个，一个是库存，存数量，与卖家有关，另一个是买家拥有的优惠券，这个与买家和库存有关。

# 微服务设计

买家个人系统

登陆注册，个人信息管理，包含数据库buyer，

卖家个人系统

登陆注册，个人信息管理，包含数据库seller

订单系统

订单浏览，下订单，订单结束，数据库orders，调用买家系统和卖家系统扣钱

优惠券系统

创建优惠券，获取优惠券，查看优惠券，数据库

商品系统

创建商品，查看商品，数据库goods

评论系统

评价商品，评价店铺，查看评价，数据库comments

消息中间件

接受订单订阅，交给邮件系统、短信系统、积分系统消费，

邮件系统

发送邮件

短信系统

发送短信

积分系统

查看积分，增加积分，数据库

推送系统

根据买家浏览记录推送商品，记录浏览记录

秒杀系统

包含独立的商品系统和订单系统，其他的系统使用现有的服务，独立的暂时的商品数据库和订单数据库，然后通过消息队列延迟更新到主库

搜素系统

根据关键字检索

备注

秒杀系统的redis数据库使用消息队列更新到秒杀系统的数据库，最后延时更新到主库，即先更新redis后更新mysql，用最终一致性换取高可用性，脏数据在数据库出现

日常系统先更新mysql，后利用canal同步到redis，保证强一些的一致性，脏数据在redis出现

# 服务治理与服务发现

如果没有服务治理与服务发现，我们的微服务内部几乎难以维持通信，因为要去维护每一个微服务的IP的端口，这简直致命，几乎不可取，考虑使用SpringCloud的Eureka，这样，我们的IP加端口就转化为服务名了，IP会变化，但是服务名字，不可能变化的，唯一不好解决的现在只剩下Eureka服务的IP了，这里似乎没有其他处理办法，他们的IP一定是固定的

# API网关

后端的通信接口是搞定了，但是前端怎么访问后端，你难道要前端去找Eureka？这不现实！考虑使用Zuul，一方面他可以为我们提供一个过滤器，另一方面他简化了前端的接口调用

# Nginx负载均衡

我们的Zuul一定是一个集群，他有众多的IP，接口简化了，但是IP依然很多，同时考虑到Zuul的吞吐量问题，我们还可以在外层加一个Nginx负载均衡

# 安全认证

考虑将认证分为多个步骤，身份认证+权限认证，在zuul网关处认证身份，这个过程非常简单，只需要使用分布式session就可以实现，在微服务内部进行权限认证，内部通过身份确定权限，这要比访问redis高效的多。

# 服务熔断

当微服务调用链的底端运行缓慢的时候，为了避免雪崩发生，顶端不应该继续给他施加压力，这时候就需要熔断了，整个电商系统，并发量最高的地方一定是人们对商品、店铺的访问，我们需要对这几个地方做熔断处理

# 分布式事务

开发程序几乎难以避免事务的出现，事务本身就是一个很常见的东西，比方说转账，当转账的数据保存在不同的节点上的时候，分布式事务出现了，分布式事务一直一来都是一个难以解决的事情，常用的处理方法有两阶段提交、三阶段提交、try-confirm/cancel、本地消息表、RocketMQ事务和spring-allibaba-FESCAR，这里我们使用RocketMQ来实现，由于分布式事务不容易解决，所以我们要尽量避免使用分布式事务，扬长避短，我们这里涉及到事务的就是下订单，对我们的系统来说，下订单包含了很多操作，买家的钱减少、卖家的钱增加、商品的数量增加、给买家发邮件和短信、给卖家发邮件和短信和卖家买家加积分，这里的操作中我们注意到前三个步骤应该做成一个本地事务，为了减少买家下订单的等待时间，我们考虑将后面的邮件和短信做成分布式事务，让其变为异步处理，于是一个优化被提出来了，将买家的基本信息分表，分为基本信息表和余额表，储存在不同的数据库中，这样下订单的时候，只涉及到余额表的修改，同理卖家会分出卖家余额表，商品会分出商品余量表，我们将买家余额表、卖家余额表、商品余量表储存在一个数据库集群中，专门用来对抗订单操作，就极大的避免了分布式事务的发生，由于订单下完以后，发邮件、发短信和加积分只要无限重试，总会成功，这里我们可以利用分布式事务的最终一致性保证分布式事务的可靠性。

# RocketMQ分布式事务容灾

99.99%的高可用性依然存在宕机的可能，有一篇技术博客上说他的朋友公司碰到过这个情况，并给出了下面的解决方案，我们可以借助服务降级框架来对RocketMQ分布式事务进行容灾处理，当上游连续多次投递消息失败，则开启服务降级，用redis数据库的队列来顶替，<https://www.cnblogs.com/jajian/p/10014145.html>

# NOSQL

SQL是必须的，NOSQL也是必须的，我们要利用好redis

# 服务配置中心

我们的分布式系统数量庞大，其配置信息几乎难以人工维护，我们需要使用配置中心，让各个微服务自己去读取自己的配置文件

# 消息总线

当配置中心的配置文件发生变化的时候，难道我们要逐个重启所有的微服务？这不现实，我们需要使用动态的配置，当配置文件发生变化的时候，自动通知各微服务刷新自己的配置。