## SSO单点登录

SSO系统即校验接口、注册、登录接口、根据token查询用户接口、安全退出接口。

具体流程：当用户点击注册的时候，跳转到注册页面，即用户信息的保存功能。检验用户名是否存在、手机号和邮箱不能为空。

当用户点击登录按钮的时候，用户输入用户名和密码，检验用户名是否在数据库中存在，然后用户名密码是否正确。这里的密码是用了spring的MD5加密技术。当全部成功后，给用户颁发一个token令牌（利用uuid实现），然后将token存入到redis中（token的key是它生成的号，值是用户的名字），然后设置在redis的过期时间。这相当于用户的session。

  然后将token写入cookie中，前台页面利用jsonp调用，根据cookie中的token的值，调用sso的根据token查询用户的服务，查看用户是否有效，如果有效则将用户返回前台页面，前台页面获取用户的用户名显示在首页，表示\*\*\*已登陆。

这里的cookie是设置了共享域，即全部子系统都可以访问到cookie。当用户登录其他子系统时，先从从cookie中获取token信息，根据token信息获取用户信息，判断用户信息是否有效，如果有效则放行，如果无效，则利用拦截器拦截跳转到登录页面。用户再次登录的时候刷新redis的时间，重新设置有效期。

### 订单模块需要用到的表

1. 数据库的设计：订单相关表设计、订单关联的诸如商品列表、会员信息、折扣、积分、打包销售等；账单相关表，包括内部账单和渠道支付账单（如微信支付、支付宝支付等），还有就是操作日志类。
2. 第三方支付：主要是支付过程中一些正常和异常的流程，微信支付你可以参考它帮助文档中推荐的测试用例，挺完整的；另外就是后台需要轧账和平账，就是你要每天和第三方平台去对一次账，看看两边数据库里的支付情况是否正确。
3. 你在上面提到了及时到账，那就证明可能有个人账户体系，这里的充值、提现要想好怎么搞？一般第三方支付针对个人是没有提现接口的，只有退款

### 购物车与商城以及订单的关系：

从一般的商城来看，可以分为B2C与C2C，也就是单商城系统和多商城系统。单商城的系统，基本上就是全部商品生成一个订单，而多商城系统里面的购物车则是可以根据店铺来分别支付生成订单（如微店）或者全部统一支付然后根据店铺拆分订单（如有赞，淘宝等）。

1. 根据每个店铺生成订单去支付，很好理解，例如我在店铺A，买了1，2，3这几个商品，我只需要生成一个订单号，然后去支付就可以了，后续的退款等各种处理，只需要根据该订单号进行处理即可。
2. 购物车里面有多个店铺的商品，假定我们使用[**微信**](http://lib.csdn.net/base/wechat)支付，微信支付每次下单只能使用唯一一个单号，那么我们只能把不同的店铺，例如店铺A和店铺B的所有商品，都统一放到一个订单号去微信下单支付。但是，这样子又违反了订单规则：不同的店铺存在着不同的订单业务，店铺和订单是一对多的关系，而且每个订单号必须是唯一的。怎么办？这个时候，我们可以把内部订单号和微信下单号做一个映射（也就是图所说的拆单），后续做各种处理例如退款等，就可以通过映射关系去进行处理。

总结一下他们之间的关系：

      （1）购物车可以存在多个店铺多个商品，可以一次性给钱购买购物车所有商品

      （2）一个订单只能对应一个店铺，一个店铺可以拥有多个订单

1. 微信下单号只有一个，一个微信下单号可以对应多个内部订单号，一个内部订单号只能对应一个微信下单号

### C2C商城购物车数据库设计与技术实现

由于B2C商城和C2C大同小异，这里暂且不讨论B2C的设计和实现，相信会C2C实现而不会B2C的同学是不存在的。且纵观目前的商城，大部分慢慢倾向于增加商家入住功能，所以建议预留多商铺功能，即先把商铺表加进去，与商品相关的带上商铺id，只不过目前商铺只有一个就是自己，就这样可以减少业务需求改动带来的大量数据库结构和代码的改动。

如果用户购物车内的商品都是一个店铺的，那么就不存在拆单、映射表这种说法，直接生成唯一订单号作为微信订单号支付就可以，但是谁都不知道需求是如何变化的，既然淘宝都是可以统一支付不同店铺的商品，那么设计的时候最好是支持购物车所有商品统一支付的，这样子就通杀了，不管你是B2C的购物车，还是微店的购物车，还是淘宝的购物车，都能满足需求。如果只能支持不同店铺做分别支付，类似微店这样，那么万一产品要改成支付宝这样子，就又得重新设计映射表，进行拆单了。本人所在公司的产品经理刚开始比较倾向于微店这种产品设计，而我设计系统时，也仅仅往产品的需求思考，而没考虑到淘宝的设计，现在换一个产品又要改为淘宝这种购物车，就感觉深深地掉进了坑里面。这里学到了一个道理，那就是永远不要相信产品经理（哭），当然也不要过度设计，这里其实不是过度设计，只是用多一点时间，就能减少以后的巨大时间，而且产品人员也很喜欢参考大公司的产品功能，毕竟一些基础功能都是经过大量的用户反馈的。

最主要的是：商铺订单号in\_trade\_no和第三方支付下单号out\_trade\_no

1. 购物车提交过来的下单最终是以不同店铺组成的数组
2. 只有一个数组时，让商铺订单号in\_trade\_no和第三方支付下单号out\_trade\_no一致，这样子的好处是，我们可以认为订单号和下单号一致时就是在一个店铺支付的，而不是多个，这样就可以复用微信下单接口返回的数据，从而对该待付款订单进行付款，而不是再调下单接口生成新的订单号去支付（既可以减少接口调用，也可以减少费单）
3. 多个数组时，每个店铺订单对应相同的第三方支付下单号，但是商铺订单号不能与下单号出现一致的情况，否则就会导致超额付款问题。
4. 订单的退款，成功付款，只需要结合内部订单号和第三方下单号处理就可以了。

### SKU常用设计方案

1）通过属性集关联SKU属性  
　　适合品类较少的网站，管理容易些。  
    如麦包包等专卖箱包或者服饰类的网站。一般就是颜色+尺码两种。而且由于品类很少，为了方便管理，可以将SKU属性纳入到属性  
集中管理，这样产品关联了属性集后，自然就关联了普通属性、查询属性、SKU属性和评论属性了。如果该网站产品种类很少，比如只卖服装，那么可以做进一步的简化，即直接将SKU属性从属关联属性集，去掉”属性集关联SKU“。  
    基于本设计的管理方式：  
     按品类创建属性集，如箱包、鞋子、服装、文胸等。然后创建多个SKU属性，即使针对内涵相似的，但是可选项不同的也创建  
多个，如尺码，用在箱包和用在服装上是完全不同的。这些分别创建，并关联不同的属性集。  
      产品创建时，关联一个属性集，通过属性集关联了1~N个SKU属性，然后选项这些SKU属性的组合，如2个颜色\*3个尺码，即6个组合，然后可以根据需要删除不支持的组合，这样最终得出了一个组合列表，点击”生成SKU“，就根据组合数量创建了产品  
SKU，每个产品SKU对应一个组合，存储在产品SKU选项值表中。对于某些SKU，可以设置专门的选项配图。  
  
2）产品和SKU属性直接关联  
      适合品类很多网站，比较灵活，但是维护起来数据量比较大。  
      为了简化，我增加SKU属性关联产品分类（可为空，表示是全局的），这样在创建产品时，可以只列出全局的+本产品分类的SKU属性，这样就不会一下子列出很多SKU属性了。SKU属性分为前端名称和后台名称两个，方便不同业务含义的SKU属性，在前端也能够用同一个名称显示，如颜色、容量等。另外在操作上可以做些优化，比如用下拉列表显示可选的SKU属性时，可以同时显示该属性的属性描述，供产品维护人员参考。  
 基于SKU方式来管理产品时，产品的价格、库存和图片等信息必然是放在产品SKU表中处理的，和订单、购物车等表的关联，也是通过产品SKU表，而不是产品表。至于产品表，实际上是一个总的业务汇总和外部关联表，但实际销售的并不是它。我们网站做的更细些，会就每个产品SKU生成独立的URL（伪静态），但从SEO方面考虑，每个产品SKU拥有独立。

### Redis作用

redis中存储的都是key-value格式的。拿商品数据来说，key就是商品id，value是商品相关信息的json数据。  
 在商城系统中当并发量比较高，频繁的对数据库进行读操作的时候都需要添加缓存。例如页面中内容数据的缓存、商品数据的缓存以及用户数据的缓存等。  
 做商品数据的缓存时，因为商品的数据量很大，而且缓存是把数据保存到内存中，此时不可能把所有的商品数据都放到缓存中。所以需要设置商品数据缓存的有效期，当用户访问到非热点数据后，此数据放到缓存中，当缓存到期后就从缓存中删除，而且长时间不会添加到缓存。而热点数据一旦从缓存中删除会马上又添加到缓存。这样可以提高缓存的利用率，同时也减轻了数据库的压力。

### 插入商品的话，要求级联插入的表

通过Redis生成商品编号（ID）。保存商品表。再保存SKU表（此表中外键，是商品表的ID）

### Zookeeper节点类型

**持久节点（PERSISTENT）**

所谓持久节点，是指在节点创建后，就一直存在，直到有删除操作来主动清除这个节点——不会因为创建该节点的客户端会话失效而消失。

**持久顺序节点（PERSISTENT\_SEQUENTIAL）**

这类节点的基本特性和上面的节点类型是一致的。额外的特性是，在ZK中，每个父节点会为他的第一级子节点维护一份时序，会记录每个子节点创建的先后顺序。基于这个特性，在创建子节点的时候，可以设置这个属性，那么在创建节点过程中，ZK会自动为给定节点名加上一个数字后缀，作为新的节点名。这个数字后缀的范围是整型的最大值。

**临时节点（EPHEMERAL）**

和持久节点不同的是，临时节点的生命周期和客户端会话绑定。也就是说，如果客户端会话失效，那么这个节点就会自动被清除掉。注意，这里提到的是会话失效，而非连接断开。另外，在临时节点下面不能创建子节点。

**临时顺序节点（EPHEMERAL\_SEQUENTIAL）**

  可以用来实现分布式锁。

1.客户端调用create()方法创建名为“\_locknode\_/guid-lock-”的节点，需要注意的是，这里节点的创建类型需要设置为EPHEMERAL\_SEQUENTIAL。

2.客户端调用getChildren(“\_locknode\_”)方法来获取所有已经创建的子节点，注意，这里不注册任何Watcher。

3.客户端获取到所有子节点path之后，如果发现自己在步骤1中创建的节点序号最小，那么就认为这个客户端获得了锁。

如果在步骤3中发现自己并非所有子节点中最小的，说明自己还没有获取到锁。此时客户端需要找到比自己小的那个节点，然后对其调用exist()方法，同时注册事件监听。

之后当这个被关注的节点被移除了，客户端会收到相应的通知。这个时候客户端需要再次调用getChildren(“\_locknode\_”)方法来获取所有已经创建的子节点，确保自己确实是最小的节点了，然后进入步骤3。

### ****软负载均衡算法****

Random LoadBalance

随机，按权重设置随机概率。在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

RoundRobin LoadBalance

轮循，按公约后的权重设置轮循比率。存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

LeastActive LoadBalance

最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

ConsistentHash LoadBalance

一致性Hash，相同参数的请求总是发到同一提供者。当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。

# 水平权限漏洞、垂直权限漏洞

水平权限漏洞是指Web应用程序接收到用户请求时，没有判断数据的所属人，或者在判断数据所属人时是从用户提交的参数中获取了userid，导致攻击者可以自行修改userid修改不属于自己的数据。

最正规的方案，把权限的控制转移到数据接口层中，要求web层在调用数据接口层的接口时额外提供userid，而这个userid在web层看来通常只能通过seesion来取到，以防用户造假。但这个方案的缺陷在于实现起来要改动底层的设计，所以不适合作为修复方案，更适合作为在最开始设计时的统一控制方案。

最直接有效的修复方案：在web层的逻辑中做鉴权，检查提交CRUD请求的操作者（通过session或token等加密信息中得到，以防造假）与目标对象的权限所有者是否一致，如果不一致则阻断。这个方案实现成本低、能确保漏洞的修复质量，缺点是增加了一次查库操作。我之前一直用这种方案来对已发生的水平权限漏洞做紧急修复。

垂直权限漏洞是指Web应用没有做权限控制，或仅仅在菜单上做了权限控制，导致恶意用户只要猜到了其他页面的URL，就可以访问或控制其他角色拥有的数据或页面，达到权限提升的目的。

只需要对url资源进行权限验证即可。

# 消息队列技术与消息中间件

消息队列技术是分布式应用间交换信息的一种技术。消息队列可驻留在内存或磁盘上,队列存储消息直到它们被应用程序读走。通过消息队列，应用程序可独立地执行--它们不需要知道彼此的位置、或在继续执行前不需要等待接收程序接收此消息。

在分布式计算环境中，为了集成分布式应用，开发者需要对异构网络环境下的分布式应用提供有效的通信手段。为了管理需要共享的信息，对应用提供公共的信息交换机制是重要的。

消息队列为构造以同步或异步方式实现的分布式应用提供了松耦合方法。消息队列的API调用被嵌入到新的或现存的应用中，通过消息发送到内存或基于磁盘的队列或从它读出而提供信息交换。消息队列可用在应用中以执行多种功能，比如要求服务、交换信息或异步处理等。

中间件是一种独立的系统软件或服务程序，分布式应用系统借助这种软件在不同的技术之间共享资源，管理计算资源和网络通讯。它在计算机系统中是一个关键软件，它能实现应用的互连和互操作性，能保证系统的安全、可靠、高效的运行。中间件位于用户应用和操作系统及网络软件之间，它为应用提供了公用的通信手段，并且独立于网络和操作系统。中间件为开发者提供了公用于所有环境的应用程序接口，当应用程序中嵌入其函数调用，它便可利用其运行的特定操作系统和网络环境的功能，为应用执行通信功能。

# 消息队列的通讯模型

点对点通讯：点对点方式是最为传统和常见的通讯方式，它支持一对一、一对多、多对多、多对一等多种配置方式，支持树状、网状等多种拓扑结构。

多点广播：MQ适用于不同类型的应用。其中重要的，也是正在发展中的是"多点广播"应用，即能够将消息发送到多个目标站点(Destination List)。可以使用一条MQ指令将单一消息发送到多个目标站点，并确保为每一站点可靠地提供信息。MQ不仅提供了多点广播的功能，而且还拥有智能消息分发功能，在将一条消息发送到同一系统上的多个用户时，MQ将消息的一个复制版本和该系统上接收者的名单发送到目标MQ系统。目标MQ系统在本地复制这些消息，并将它们发送到名单上的队列，从而尽可能减少网络的传输量。

发布/订阅(Publish/Subscribe)模式：发布/订阅功能使消息的分发可以突破目的队列地理指向的限制，使消息按照特定的主题甚至内容进行分发，用户或应用程序可以根据主题或内容接收到所需要的消息。发布/订阅功能使得发送者和接收者之间的耦合关系变得更为松散，发送者不必关心接收者的目的地址，而接收者也不必关心消息的发送地址，而只是根据消息的主题进行消息的收发。在MQ家族产品中，MQ Event Broker是专门用于使用发布/订阅技术进行数据通讯的产品，它支持基于队列和直接基于TCP/IP两种方式的发布和订阅。

群集(Cluster)：为了简化点对点通讯模式中的系统配置，MQ提供Cluster(群集)的解决方案。群集类似于一个域(Domain)，群集内部的队列管理器之间通讯时，不需要两两之间建立消息通道，而是采用群集(Cluster)通道与其它成员通讯，从而大大简化了系统配置。此外，群集中的队列管理器之间能够自动进行负载均衡，当某一队列管理器出现故障时，其它队列管理器可以接管它的工作，从而大大提高系统的高可靠性。