**对表的优化**

**1.给主键、外键加索引**

**2.根据业务需求，对那些频繁查询的字段中加上索引**

**3.对表进行分表**

**1.水平分表依据：根据时间分表、根据地区分表**

**例：中国移动 中国联通 对用户信息的存储**

**2.垂直分表依据：频繁访问的字段存放在一张表中，不常用的字段存放在另外的一张表中**

**例：用户登录 只需要账号密码手机号即可登录**

1. **如果数据不是特别重要的话可以考虑采用分片的方式将数据存放在mongoDB中**

**SQL优化:**

**外键必须加索引。**

**避免在 where 子句中对有索引的字段进行运算,这会导致索引失效，从而进行全表扫描。**

**在 where 及 order by 涉及的列上建立索引，要尽量避免全表扫描。**

**在设计表时要避免表中字段出现null的情况，通常要为其设置默认值。**

**避免在查找时放弃使用索引而进行全表扫描。**

**SELECT语句中避免使用'\*’，只查询需要返回的字段 ，这样可以减少oracle解析sql语句的时间。**

**用NOT EXISTS 替换 NOT IN 操作符，用 EXISTS 替换 IN**

**Java代码优化**

**对于大批量字符串的拼接使用stringbuffer或者stringbuilder代替string进行+拼接。**

**根据业务情况使用缓存减少对数据库的访问。**

**单线程应尽量使用 HashMap, ArrayList，因为HashTable,Vector使用了同步机制，降低了性能。**

**在finally块中关闭流,断开连接，释放资源。**

**避免在循环条件中使用复杂表达式 。**

**关于性能优化的话我们还可以在程序中加入缓存来提高应用程序的性能**

**缓存是为了应用程序为了提高性能，可以通过使用缓存来达到目的，缓存的存储介质可以**

**内存或者硬盘，通常将数据存储在内存里，确切的说是jvm的内存中，缓存是**

**基于Map这种思想构建的，以键值对的方式进行存取，之所以还可以将**

**缓存的数据存储在硬盘中，是因为内存资源相当有限和宝贵，所以当内存资源**

**不足的时候，就可以将其存储到硬盘中，虽然硬盘的存取速度比内存要慢，但是**

**因为减少了网络通信量，所以还是提高程序的性能。缓存可以分为客户端缓存和**

**服务器端缓存，所谓的客户端缓存通常指的是IE浏览器的缓存，服务器端缓存指**

**的web服务器的缓存，通常可以通过第三方组件实现，如oscache,memcache**

**我们通常将那些频繁访问但是又不是经常改变的数据进行缓存。为了保证缓存数据的**

**有效性，在数据发生改变的时候，我们要刷新缓存，避免脏数据的出现。刷新缓存的**

**策略有两种，一种是定时刷新，一种手动刷新。**

**缓存的层次如下:jsp-->action-->service(通常放置在service)-->dao,**

**缓存越靠前对性能的提升越大**

**缓存的策略:(缓存空间不足需要进行清理的时候使用)**

**LRU:最近最少使用原则.(理解:存储书)**

**FIFO:先进先出的缓存策略.(理解:排队)**

**你来说说缓存？说说你对缓存的理解（如果遇到重复的，就可以省略）**

**我们在项目中使用缓存的目的是为了提高应用程序的性能，减少访问数据库**

**的次数，从而提高应用程序的吞吐量。我们通常将权限，菜单,组织机构**

**这些频繁访问但是不经常改变的基础数据进行缓存，其中我在做()某某项目的时候**

**就通过oscache对ZTree的树形菜单进行了缓存，并且在做的时候和单列设计**

**模式进行结合，考虑到多线程下的安全问题，还对单例模式加入了双重判定锁**

**的检查方式。**

# 购物车CART

## 购物车及订单支付：

购物车的数据结构是：整体上是一个list集合形式 ，包含三个属性，sellerId;//商家ID。sellerName;//商家名称。orderItemList;//购物车明细。购物车明细中包括（商家名称 单价 数量 金额 商品图片）

然后把这个结构存储到redis或者cookie中。完成购物车实际上是完成购物车内部的增删改查

总体操作思路：

**在详情页点击“加入购物车”或者“查看购物车”的时候我们做了一个判断，**

**假如在未登录状态下没有服务器进行交互，我们会将用户加入到购物车中的商品信息存入到cookie中，**

**然后在用户点击"去结算"时提醒用户进行登陆，并将购物车中的信息和用户关联起来存入到redis中，**

**如果用户已经登陆则用户将商品加入购物车时直接将信息存入redis中，并设置过期时间，**

**这样做的目的是为了防止用户频繁的对购物车进行操作增加系统压力，**

**而redis可以看做是一个持久化的缓存。**

## CORS"跨域资源共享"

**在详情页点击“加入购物车”或者“查看购物车”**将当前商品加入购物车，使用location.href跳转到购物车页面。发现无法跨越访问XMLHttpRequest cannot load

我们使用cors解决

所谓的跨域操作，就是使用js请求不同的域。不同的域是指协议，域名，端口，任一不同。

首先修改mall-cart-web 的CartController.java的addGoodsToCartList方法，添加下面两句代码response.setHeader允许访问的域名

|  |
| --- |
| //允许访问的域名  response.setHeader("Access-Control-Allow-Origin", "http://localhost:8005");  //允许请求操作cookie的证书  response.setHeader("Access-Control-Allow-Credentials", "true"); |
|  |

在客户端angluars的请求中，设置withCredentials带有允许访问域**就可以了**

//添加商品到购物车$scope.addToCart ()添加{'withCredentials':**true**}带着证书才能访问

**另外**

springMVC的版本在4.2或以上版本，可以使用注解实现跨域, 我们只需要在需要跨域的方法上添加注解@CrossOrigin即可

|  |
| --- |
| @CrossOrigin(origins="http://localhost:8005",allowCredentials="true") |

allowCredentials="true" 可以缺省

# 购物车工程搭建：

1. 创建工程mall-cart-interface ，依赖mall-pojo
2. 创建工程mall-cart-service（WAR），依赖mall-cart-interface 和mall-common工程 和spring、 dubbox 等相关依赖, 添加web.xml 与spring配置文件 tomcat插件端口设置为9007 ，dubbo端口为20887
3. 创建工程mall-cart-web ,依赖 mall-cart-interface springsecurity 、CAS 等。添加web.xml 与spring配置文件，tomcat插件端口设置为9107 ，
4. 在mall-common工程中添加util\CookieUtil.java Cookie工具类。需要在mall-common工程引入servlet-api依赖

## mall-pojo

我们在mall-pojo的com.cblue.pojogroup中创建购物车实体类 Cart， 包含三个属性，sellerId;//商家ID。sellerName;//商家名称。orderItemList;//购物车明细。这个类是对每个商家的购物车进行的封装

## 服务层负责逻辑

然后使用cookie和redis存储购物车数据。服务层负责逻辑，控制层负责读写cookie和redis 。

### 建立服务层接口CartService

1. 创建添加商品到购物车的方法

**public** List<Cart> addGoodsToCartList(List<Cart> cartList,Long itemId,Integer num );

（2）定义从redis中查询购物车的接口 findCartListFromRedis(String username);传递的参数是username；

（3）定义将购物车保存到redis的接口saveCartListToRedis(String username,List<Cart> cartList)传递的参数是username，cartList；

（4）把cookie合并到reids中的接口

### 在服务层创建实现类CartServiceImpl.java实现 CartService的方法，

#### （添加商品到购物车）

//1.根据商品SKU ID查询SKU商品信息

TbItem item = itemMapper.selectByPrimaryKey(itemId);

if(item==null){

throw new RuntimeException("商品不存在");

}

if(!item.getStatus().equals("1")){

throw new RuntimeException("商品状态无效");

}

//2.获取商家ID String sellerId = item.getSellerId();

//3.根据商家ID判断购物车列表中是否存在该商家的购物车

Cart cart = searchCartBySellerId(cartList,sellerId);

//4.如果购物车列表中不存在该商家的购物车

//4.1 新建购物车对象new Cart();

//4.2 将新建的购物车对象添加到购物车列表 cartList.add(cart);

//5.如果购物车列表中存在该商家的购物车

// 查询购物车明细列表中是否存在该商品

searchOrderItemByItemId(cart.getOrderItemList(),itemId);

//5.1. 如果没有，新增购物车明细

orderItem=createOrderItem(item,num);

cart.getOrderItemList().add(orderItem);

//5.2. 如果有，在原购物车明细上添加数量，更改金额

orderItem.setNum(orderItem.getNum()+num);

orderItem.setTotalFee(new BigDecimal(orderItem.getNum()\*orderItem.getPrice().doubleValue()));

//如果数量操作后小于等于0，则移除

if(orderItem.getNum()<=0){

cart.getOrderItemList().remove(orderItem);//移除购物车明细

}

//如果移除后cart的明细数量为0，则将cart移除

if(cart.getOrderItemList().size()==0){

cartList.remove(cart);

## 在mall-cart-web工程新建CartController.java

## 获取当前登录人账号

在mall-cart-web的findCartList方法中，获取用户名

|  |
| --- |
| //得到登陆人账号,判断当前是否有人登陆  String username = SecurityContextHolder.*getContext*().getAuthentication().getName(); |

测试：当用户未登陆时，username的值为anonymousUser

我们（1）从cookie和redis中取出购物车（2）向购物车添加商品（3）将购物车存入cookie和redis

创建从cookie和reids中查询购物车的方法findCartList()

**（1）如果未登录使用cookie，进行查询**

String cartListString = util.CookieUtil.*getCookieValue*(request, "cartList","UTF-8");

然后使用List<Cart> cartList\_cookie = JSON.*parseArray*(cartListString, Cart.**class**)转化格式返回给前端

addGoodsToCartList（）使用findCartList()获取购物车列表cartList，以cartList为键，把购物车列表存入到从cookie中，并设置字符集"UTF-8"

addGoodsToCartList(Long itemId,Integer num){

List<Cart> cartList =findCartList();//获取购物车列表

cartList = cartService.addGoodsToCartList(cartList, itemId, num);

util.CookieUtil.*setCookie*(request, response, "cartList", JSON.*toJSONString*(cartList),3600\*24,"UTF-8");

**return** **new** Result(**true**, "添加成功");

# 首页的广告显示(redis)

## 简述

我们实现首页的广告显示，因为首页访问量比较大，我们可以使用两种方式来减小服务器的压力。一种方式是使用Redis实现缓存，一种是实现页面的静态化。

我们使用redis缓存，将一些前台页面展示的数据放到redis中，减少访问数据库的次数，减缓服务器的压力。当初在进行技术选型的时候，我们主要对redis和memcache进行了一个对比。

## redis和memcache对比。

1. redis与memcache的对比

Memcached是多线程，而Redis使用单线程.

### 1)存储方式

memecache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小

redis支持数据的持久化可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。用于数据恢复

持久化方式：有快照和AOF日志两种持久化方式，在实际应用的时候，要特别注意配置文件快照参数，要不就很有可能服务器频繁满载做dump）

### 2)数据支持类型：

redis在数据支持上要比memecache多的多。不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

### 3)数据备份

Redis支持数据的备份，即master-slave模式的数据备份，防止单点故障

### 4)底层数据模型

使用底层模型不同： redis直接自己构建了VM 机制

对于两者的选择还是要看具体的应用场景，你需要关注数据持久化和主从复制，排序等一系列复杂的操作时，毫无疑问选择redis，如果你的目标是构建一个缓存在升级或者重启后之前的数据不会丢失的话，那也只能选择redis。如果需要缓存的数据只是key-value这样简单的结构时，我在项目里还是采用memcache它也足够的稳定可靠。

Ehcache是进程内缓存，memcached和redis是分布式缓存，分布式缓存会比较方便你以后项目的横向扩展；如果应用比较简单Ehcache会更加适合

## 实现首页广告缓存

使用Redis实现缓存，首先Redis是nosql数据库（非关系型数据库），主要有五种。String，hash，set，list，sortset。

我们使用SpringDataRedis来操作Redis。当首页加载的时候，访问广告数据，首先去redis中查询是否有该类广告，如果有，直接从redis获得，不需要查询数据库。如果没有，就查询数据库，并把数据添加到redis中，下次查询时候，就可以从redis中获得。

更新缓存，当添加广告的时候，把广告所属的类的缓存删除。当更新广告的时候，把广告的原来的分类和新的分类的缓存删除。当删除广告，同样把广告所属的类的缓存删除。

## 缓存品牌和规格数据

当我们根据关键字进行搜索的时候，在搜素面板会显示关键字分类，并且搜索面板的

品牌和规格，都来自于第一个商品分类。这时候，根据商品分类，从数据库中查询品牌和规格信息，会对数据库产生较大压力，所以，我们需要把品牌和规格进行redis的缓存处理。

我们在redis中存放的缓存的形式是：首先，我们把商品分类数据放入缓存中，以分类名称作为key ,以模板ID作为值。然后该分类对应的品牌数据和规格数据分别以模板ID作为key,以品牌列表或规格列表作为值，保存到缓存中。

我们之前的缓存都是在修改数据库【增，删，该】的时候添加，但是这次我们在商家的服务层的查询方法中添加缓存，因为在运营商后台，修改数据库都会执行查询操作。在添加品牌和规格的缓存中，首先修改模板服务的实现类，注入RedisTemplate操作类，定义一个将数据存入缓存的方法，在这个方法中调用查询所有模板表中的数据的方法，返回一个list集合，循环模板集合，获得模板对象，使用模板对象获取品牌信息，因为品牌信息是数组类型的json字符串，需要使用JSON.parseArray()方法将它转化层list<map>形式，然后模板以模板ID作为key,以品牌列表为值，使用RedisTemplate对象存入到Redis中。同理，使用模板对象获取模板ID，通过findSpecList()方法，获取规格及其规格选项数据，然后以模板ID作为key,以规格列表为值，使用RedisTemplate添加到缓存中。

## 实现缓存品牌和规格数据

在搜索服务的实现类中，根据第一个分类的名称，从redis中取出对应的模板ID，然后根据模板ID查询品牌列表和规格列表，将返回得到数据添加到查询结果的map中并返回给前端页面，最后使用ng-repeat显示品牌和规格及其规格选项。

#### （从redis中存取数据）

一，在cartService实现服务层接口redisTemplate.boundHashOps("cartList").put(username, cartList);向redis存入购物车数据；从redis中根据用户名提取购物车数据(List<Cart>) redisTemplate.boundHashOps("cartList").get(username);

定义从redis中查询购物车的接口 findCartListFromRedis(String username);传递的参数是username

在cartService实现服务层接口，从redis中根据用户名提取购物车数据

(List<Cart>) redisTemplate.boundHashOps("cartList").get(username);

1. **如果已登录从redis中提取**

List<Cart> cartList\_redis =cartService.findCartListFromRedis(username);

//创建添加购物车的方法

（2）如果是已登录，保存到redis

cartService.saveCartListToRedis(username, cartList);

为避免调用远程服务超时，我们可以将过期时间改为6秒（默认为1秒）

|  |
| --- |
| @Reference(timeout=6000)  **private** CartService cartService; |

## 合并购物车

创建接口**public** List<Cart> mergeCartList(List<Cart> cartList1,List<Cart> cartList2);

在实现类中循环cartList2，并把循环后的结果添加到cartList1中

在mall-cart-web工程CartController类的findCartList方法

**if**(cartList\_cookie.size()>0){//如果本地存在购物车

//合并购物车

cartList\_redis=cartService.mergeCartList(cartList\_redis, cartList\_cookie);

//清除本地cookie的数据

util.CookieUtil.*deleteCookie*(request, response, "cartList");

//将合并后的数据存入redis

cartService.saveCartListToRedis(username, cartList\_redis);

}

## 前端控制层

# AngularJS

我们在前端使用AngularJS框架，主要使用了AngularJS的MVC模式和模块化的思想，实现了前端JS代码分层、分模块，形成了和后端一样的，服务层和控制层。然后使用依赖注入把Controller层需要的服务注入进去。最后，使用双向数据绑定来异步加载数据。

ng-click： 是最常用的单击事件指令，再点击时触发控制器的某个方法

ng-repeat：指令用于循环数组变量

**ng-app** 指令中定义的就是模块的名称

ng-model 指令用于绑定变量,这样用户在文本框输入的内容会绑定到变量上，而表达式可以实时地输出变量。

ng-init指令来对变量初始化

**ng-controller** 指令用于为你的应用添加控制器。

$scope 就在视图和控制器之间建立了一个通道,基于作用域视图在修改数据时会立刻更新 $scope,同样的$scope 发生改变时也会立刻重新渲染视图.

$http **内置服务**我们的数据一般都是从后端获取的，那么如何获取数据呢？我们一般使用内置服务$http来实现。

mall-cart-web增加cartController.js 从后端得到cartList数据

|  |
| --- |
| //购物车控制层  app.controller('cartController',**function**($scope,cartService){  //查询购物车列表  $scope.findCartList=**function**(){  cartService.findCartList().success(  **function**(response){  $scope.cartList=response;  }  );  }  }); |

添加相关指令ng-app=*"mall"*，指定控制器ng-controller=*"cartController"*，调用初始化方法 ng-init=*"findCartList()"，然后使用* ng-repeat=*"cart in cartList"*循环显示购物车列表显示商家id和商家名称； ng-repeat=*"orderItem in cart.orderItemList"显示购物车明细列表*

## *然后点击的时候使用ng-click*实现数量增减

ng-repeat=*"orderItem in cart.orderItemList"显示购物车明细列表*

*{{orderItem.picPath}}图片，*{{orderItem.title}}标题，{{orderItem.totalFee.toFixed(2)}}总金额， ng-model=*"orderItem.num"* 数量

# 收件人地址显示，默认，选择

使用angluarJS根据用户的user\_id,去查询tb\_address表。在页面中循环展示。当某条记录的is\_default属性是1的时候，显示为默认信息。当我们选择某一个地址的时候，这个地址为选中地址，显示选中样式。

#### 商品清单与金额显示

调用查询购物车，显示信息。

#### 提交订单

### 搭建框架

（1）创建mall-order-interface 引入依赖mall-pojo

（2）创建mall-order-service （WAR） 参照其它服务工程引入依赖，添加web.xml spring配置文件 dubbox端口20888 tomcat7运行端口9008

（3）mall-cart-web 引入依赖mall-order-interface

因为订单数据是分布式的数据库，所以需要使用分布式id生成器--雪花算法，实现全球唯一id。在OrderServiceImpl.java的add方法中根据用户名（UserId）从redis中得到购物车数据循环购物车数据,把购物车的数据都放到order表中

把购物车明细循环保存到tb\_order\_item表中。

如果订单是多个商家，我们保存到订单表也应该是多条记录。

之后清空购物车信息

redisTemplate.boundHashOps("cartList").delete(order.getUserId());

当跳转订单以后，根据支付方式，如果支付是在线支付，就跳转到支付页面。否则，货到付款，就完成了整个页面的购物流程。

（1）将“配置文件/工具类”下的IdWorker.java 拷贝到common工程

（2）在mall-order-service工程的spring配置文件中添加配置

|  |
| --- |
| <bean id=*"idWorker"* class=*"util.IdWorker"*>  <!-- 进程ID -->  <constructor-arg index=*"0"* value=*"0"*></constructor-arg>  <!-- 数据中心ID -->  <constructor-arg index=*"1"* value=*"0"*></constructor-arg>  </bean> |

# 微信支付吗？

## （你给我讲讲你是怎么做的微信支付的，如何确认支付成功和金额正确的）

## 2.5工程搭建与准备工作

1. 建立支付服务接口模块mall-pay-interface （jar）

**public** Map createNative(String out\_trade\_no,String total\_fee);

根据out\_trade\_no 订单号和total\_fee 金额(分)生成二维码

（2）建立支付服务实现模块mall-pay-service （war） 依赖mall-pay-interface 和mall-common 、 spring dubbox 相关依赖 、**微信SDK （**wxpay）（因为不需要连接数据库所以不用引用dao工程）添加tomcat插件，运行端口为9009；添加spring配置文件 ，

1. 在mall-common工程中添加工具类HttpClient.java ，并添加依赖
2. 添加配置文件weixinpay.properties

|  |
| --- |
| appid=wx8397f8696b538317  partner=1473426802  partnerkey=8A627A4578ACE384017C997F12D68B23  notifyurl=http://a31ef7db.ngrok.io/WeChatPay/WeChatPayNotify(我们不用这个回调，所以错的也无所谓) |

appid： 微信公众账号或开放平台APP的唯一标识

partner：财付通平台的商户账号

partnerkey：财付通平台的商户密钥

notifyurl: 回调地址

1. mall-cart-web依赖工程mall-pay-service
2. 将二维码插件QRious 拷贝到mall-cart-web 的plugins目录中

使用微信扫码支付，应该申请微信扫码支付，需要微信公众号中的服务号，还必须是企业，个体，媒体，政府等。提交资料进行审核，每次审核300。等待1-5天审核，审核成功，

开户，在线签署协议。

申请之后，我们会获得公众号，商户号，秘钥，数字签名。

首先创建支付配置类,weixinpay.properties把这四项放进去其中商户ID是微信分配的,商户秘钥是我们自己定义的;

接着有参构造一个支付类WXpay,weixinpay.properties为构造支付类的参数;然后new一个HashMap,对数据进行装配,也就是订单信息,把一些必填的字段put进去,（包括公众号ID,商户号,随机字符串(nonce\_str),签名(sign)( 签名类型，默认为MD5，支持HMAC-SHA256和MD5。),商品描述(body),商户订单号(out\_trade\_no),标价金额(total\_fee),终端IP(spbill\_create\_ip),通知地址(notify\_url),交易类型trade\_type(JSAPI是公众号支付,NATIVE是扫码支付,APP是APP支付);）

（调用WXPayUtil类把map转化成xml形式，通过我们封装的httpClient，把数据提交给API的url地址，并等待微信支付系统的返回。返回的结果从xml形式重新转化成map形式，进行数据操作。）

String xmlParam = WXPayUtil.*generateSignedXml*(param, partnerkey);

System.***out***.println(xmlParam);

HttpClient client=**new** HttpClient("https://api.mch.weixin.qq.com/pay/unifiedorder");

String result = client.getContent();

System.***out***.println(result);

Map<String, String> resultMap = WXPayUtil.*xmlToMap*(result);

Map<String, String> map=**new** HashMap<>();

map.put("code\_url", resultMap.get("code\_url"));//支付地址

map.put("total\_fee", total\_fee);//总金额

map.put("out\_trade\_no",out\_trade\_no);//订单号

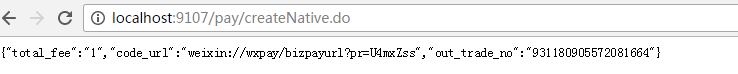
第四步 将链接变成二维码图片:这里我用的是二维码生成插件qrious,订单号通过分布式ID生成器生成，将生成的二维码展示给用户;

**public** Map createNative(){

IdWorker idworker=**new** IdWorker();

**return** weixinPayService.createNative(idworker.nextId()+"","1");

}



第五步 用户打开微信扫描二维码,提交扫描链接到微信支付系统;

第六步 支付系统验证链接的有效性,返回需要用户输入密码或者指纹;

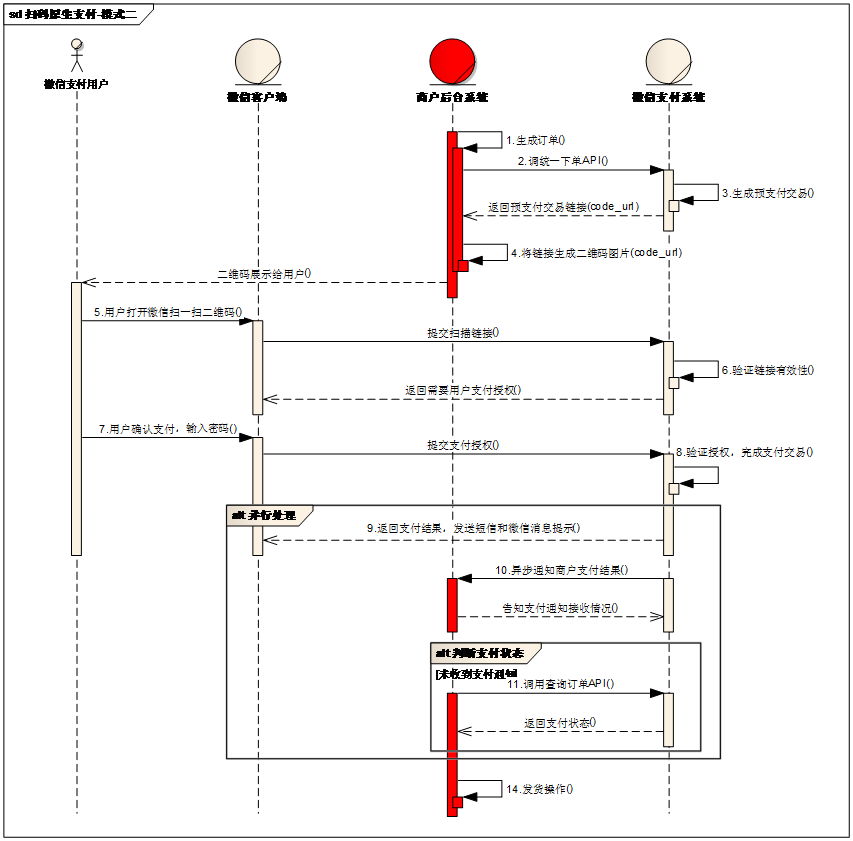
第七步 用户确认支付,输入密码,提交支付授权到支付系统;

第八步 系统验证授权,完成支付;这时走一个异步通知,以发短信或者微信消息提示的方式返回支付结果给微信客户端,同时也通知商户支付结果,

第九步 我们后台系统再告知支付通知接收情况给支付系统;如果我们后台没有接收到支付通知,则调用查询订单的API,支付系统再返回给我们一个支付状态;最后发货

**如何确认支付成功和金额正确的：**

通过查询订单API进行比对，发现问题，进行修改



### 检测支付状态

（检测微信的支付状态

调用微信的查询API，进行不断的查询，查询结果的trade\_state为SUCCESS的时候，说明支付成功，

跳转到成功页面，否则跳转到失败页面。在我们不断查询的过程中，不能让查询永远运行，我们可以通过计数的方式，设置一个时间范围，比如15分钟。如果，用户在15分钟内支付，也就是trade\_state15分钟之内不为SUCCESS，我们就结束查询。如果支付成功，跳转到成功页面，我们跳转之前添加?#money=参数，然后使用angluarJS的$location.search()去接收参数，进行显示。）

当用户支付成功后跳转到成功页面,当返回异常时跳转到错误页面

我们通过HttpClient工具类实现对远程支付接口的调用。

接口链接：https://api.mch.weixin.qq.com/pay/orderquery

定义接口查询支付状态**public** Map queryPayStatus(String out\_trade\_no);根据订单号查询订单，具体参数参见“查询订单”API,（有 appid//公众账号ID， partner商户号，out\_trade\_no订单号，nonce\_str//随机字符串）然后设置签名param加上partnerke返回一个签名，根据签名发送请求进行访问

我们在controller方法中轮询调用查询订单Thread.*sleep*(3000);//间隔三秒

（间隔3秒），查询结果的trade\_state为SUCCESS的时候，说明支付成功，

跳转到成功页面，

**if**(map.get("trade\_state").eiyquals("SUCCESS")){//如果成功

result=**new** Result(**true**, "支付成功");

**break**;

前端代码收到结果后跳转到成功页面。否则跳转到失败页面

#### 问题分析

如果用户到了二维码页面一直未支付，或是关掉了支付页面，我们的代码会一直循环调用微信接口，这样会对程序造成很大的压力。所以我们要加一个时间限制或是循环次数限制，当超过时间或次数时，跳出循环。

//为了不让循环无休止地运行，我们定义一个循环变量，如果这个变量超过了这个值则退出循环，设置时间为5分钟

**int** x=0;

x++;

**if**(x>=100){

result=**new** Result(**false**, "二维码超时");

**break**;

}

## 申请退款你是怎么做的

创建支付配置类MyConfig,作为参数有参构造支付类WXPay,然后new一个map,进行数据装配,将out\_trade\_no(商户订单号), out\_refund\_no(退款单号)( 这个退款单号是商户系统内部的退款单号，商户系统内部唯一 ，同一退款单号多次请求只退一笔), total\_fee(订单金额), refund\_fee(退款金额)

put进去,调用支付类的wxPay.fillRequestData方法,然后调用wxPay.refund方法,就可以完成退款功能,**需要注意的是,退款功能需要双向证书**,

public MyConfig() throws Exception {

String certPath = "/path/to/apiclient\_cert.p12";

File file = new File(certPath);

InputStream certStream = new FileInputStream(file);

this.certData = new byte[(int) file.length()];

certStream.read(this.certData);

certStream.close();

}

## 发送现金红包是怎么做的

还是老套路,new一个配置类MyConfig,作为参数有参构造支付类WXpay,然后new一个map,进行数据装配,将必填的请求参数put进去,有随机字符串nonce\_str(调用UUID的随机生成方法,转换成String类型,将”-”以””代替),商户订单号mch\_billno,商户号mch\_id,公众号appid(wxappi),商户名称send\_name,用户openid(re\_openid),红包金额total\_amount,红包发放总人数total\_num,红包祝福语wishing,Ip地址,client\_ip,活动名称act\_name,备注remark,**场景值id(红包金额大于200时必传)**,签名sign(**签名的生成算法**: 将所有的参数按照字典序排序, 使用URL键值对的格式（即key1=value1&key2=value2…）拼接成字符串stringA,然后拼接上商户秘钥key得到新的字符串stringB,对stringB进行MD5加密,再将加密后的字符串全部转换为大写,得到最终要发送的数据),然后调用requestWithCert方法就ok了

## 查看统一对账单是怎么做的

#### 支付日志

（当用户下单的时候，把支付的信息，以日志形式保存到tb\_pay\_log表里。

默认的支付状态是未支付。并把支付日志对象保存到redis中。

在用户跳转到支付页面的时候，从redis中获取支付对象中的支付订单号进行显示。

在用户完成支付的时候，修改订单的支付状态，修改支付日志表的支付状态，并删除redis中的支付对象。）

可以在main方法里进行测试,new一个MyConfig配置类,将myConfig作为参数有参构造一个支付类WXpay,然后new一个map,将”bill\_date”账单日期,”bill\_type”账单类型put进去;账单日期的类型为年月日,需要注意的是,微信在第二天的9点开始启动生成前一天的对账单,所以我们最好在10以点后再获取对账单;账单类型有ALL,SUCCESS,FRFUND, RECHARGE\_REFUND;然后调用wxPay.downloadBill(data)的方法,就可以得到对账单了

成功时，数据以文本表格的方式返回，第一行为表头，后面各行为对应的字段内容，字段内容跟查询订单或退款结果一致。

第一行为表头，根据请求下载的对账单类型不同而不同(由bill\_type决定),目前有： 当日所有订单, 当日成功支付的订单, 当日退款的订单

从第二行起，为数据记录，各参数以逗号分隔，参数前增加`符号，字段顺序与表头一致。这里可以用POI导出到excel表格,(我知道怎么做,但是没有做过这部分,因为公司的账单不是对每个人都公开的)