**畅购电商系统开发**

**第11天**

传智播客.黑马程序员.深圳

# 学习目标

* 登录页的配置
* 登录成功跳转实现
* 结算页查询实现
* 下单实现
* 变更库存
* 增加积分
* 支付流程介绍
* 微信扫码支付介绍

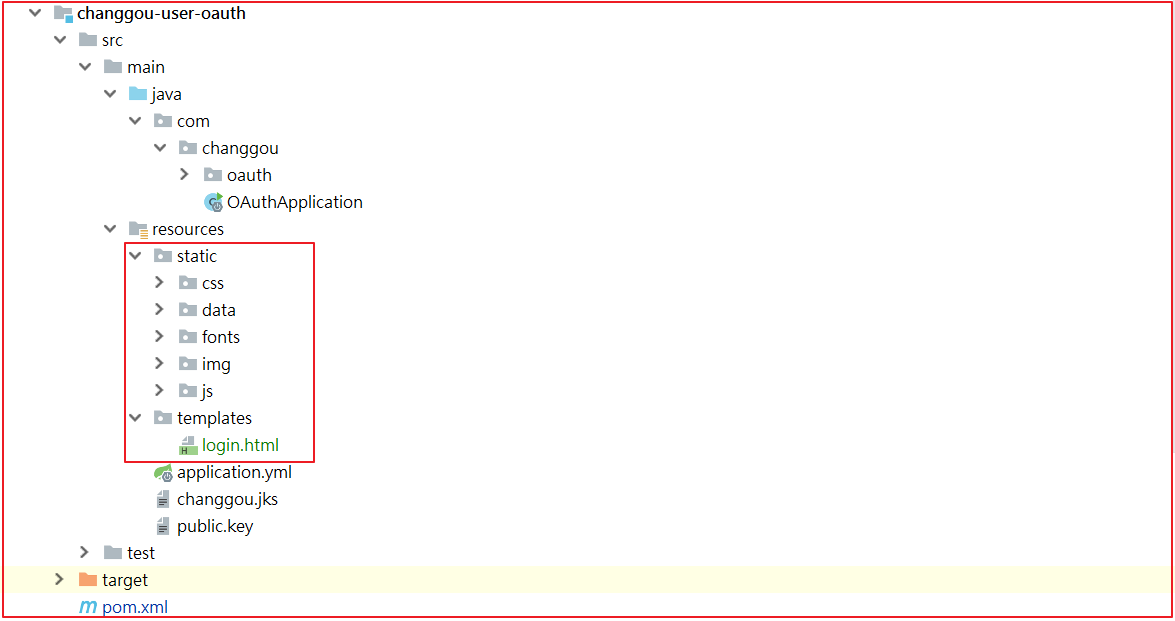
# 登录页面配置

前面使用的都是采用Postman实现登录，接着我们实现一次oauth自定义登录。

## 准备工作

### 静态资源导入

将资料/页面/前端登录相关的静态资源导入到changgou-user-oauth中,如下图。



### 引入thymeleaf

修改changgou-user-oauth，引入thymeleaf模板引擎

*<!--thymeleaf-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  
</**dependency**>

### 登录控制器

修改changgou-user-oauth,编写一个控制器com.changgou.oauth.controller.LoginRedirect，实现登录页跳转，代码如下：

@Controller  
@RequestMapping(**"oauth"**)  
**public class** LoginRedirect {  
  
 */\*\*\*  
 \* 跳转到登录页面  
 \** ***@return*** *\*/* @GetMapping(**"login"**)  
 **public** String login(){  
 **return "login"**;  
 }  
}

### 登录页配置

针对静态资源和登录页面，我们需要实现忽略安全配置，并且要指定登录页面。修改com.changgou.oauth.config.WebSecurityConfig的2个configure方法，代码如下：

**第1个configure配置:**

*/\*\*\*  
 \* 忽略安全拦截的URL  
 \** ***@param web*** *\** ***@throws*** *Exception  
 \*/*@Override  
**public void** configure(WebSecurity web) **throws** Exception {  
 web.ignoring().antMatchers(  
 **"/user/login"**,  
 **"/my/login"**,  
 **"/css/\*\*"**,  
 **"/data/\*\*"**,  
 **"/fonts/\*\*"**,  
 **"/img/\*\*"**,  
 **"/js/\*\*"**,  
 **"/login.html"**,

**"/oauth/login"**, //放行登录跳转  
 **"/user/logout"**);  
}

**第2个configure配置：**

@Override  
**public void** configure(HttpSecurity http) **throws** Exception {  
 http.csrf().disable()  
 .httpBasic() *//启用Http基本身份验证* .and()  
 .formLogin() *//启用表单身份验证* .loginPage(**"/oauth/login"**) *//默认登录页* .loginProcessingUrl(**"/user/login"**) *//默认的登录处理地址* .and()  
 .authorizeRequests() *//限制基于Request请求访问* .anyRequest()  
 .authenticated(); *//其他请求都需要经过验证*}

### 测试

为了方便后续模板页测试，建议关闭thymeleaf的缓存

**thymeleaf**:  
 **cache**: **false**

访问[http://localhost:9001](http://localhost:9001/)，如果样式没有加载成功，记得把相对路径改为绝对路径。



## 登录实现

点击登录按钮，访问之前的登录方法实现登录，我们需要对登录页做一下调整。

### 引入thymeleaf命名空间

修改login.html，引入命名空间

<**html xmlns:th="http://www.thymeleaf.org"**>

### 登录实现

点击登录按钮，使用vue+axios实现登录，我们需要定义脚本访问后台登录方法。

#### 引入js

<**script src="/js/vue.js"**></**script**>  
<**script src="/js/axios.js"**></**script**>

#### 登录脚本

<**script th:inline="javascript"**>  
 **var *app*** = **new Vue**({  
 **el**:**"#profile"**, *// 绑定页面表单相关元素api* **data**:{  
 **msg**:**''**, *//登录提示* **username**:**''**, *//用户名* **password**:**''** *//密码* },  
 **methods**:{  
 *//登录* login:**function** () {  
 **this**.**msg** = **"正在登录..."**;  
 **axios**.post(**"/user/login?username="** + **this**.**username** + **"&password="** + **this**.**password**).then(**function** (response) {  
 **if**(response.**data**.flag){  
 ***app***.**msg** = **"登录成功，即将跳转..."**;  
 }**else**{  
 ***app***.**msg** = response.**data**.**message**;  
 }  
 })  
 }  
 }  
 });  
</**script**>

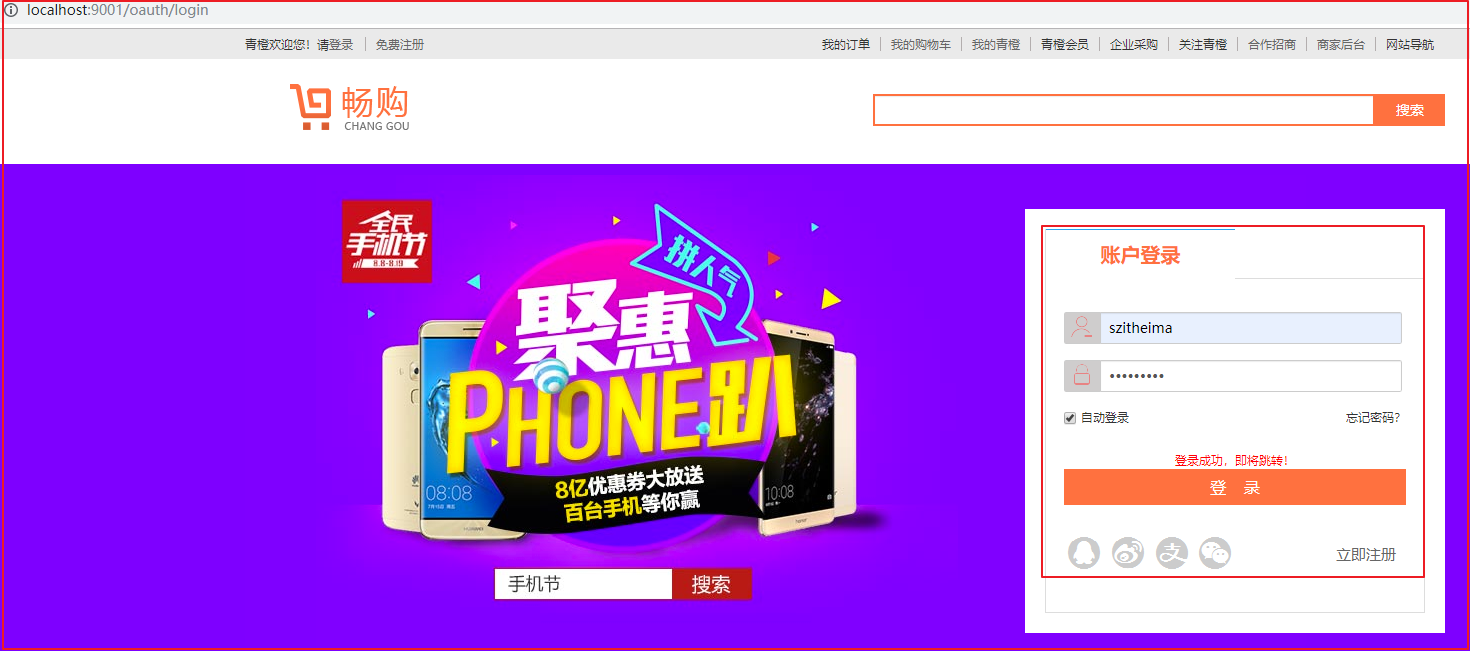
#### 修改控制器，处理登录异常，便于下一步提示

@PostMapping(**"login"**)  
**public** Result<AuthToken> login(String username, String password) {  
 **try** {  
 **if**(StringUtils.*isEmpty*(username)){  
 **throw new** RuntimeException(**"用户名不允许为空"**);  
 }  
 **if**(StringUtils.*isEmpty*(password)){  
 **throw new** RuntimeException(**"密码不允许为空"**);  
 }  
 *//申请令牌* AuthToken authToken = **authService**.login(username,password,**clientId**,**clientSecret**);  
  
 *//用户身份令牌* String access\_token = authToken.getAccessToken();  
 *//将令牌存储到cookie* saveCookie(access\_token);  
  
 **return new** Result<AuthToken>(**true**, StatusCode.***OK***,**"登录成功！"**,authToken);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return new** Result<>(**false**, StatusCode.***LOGINERROR***, **"登录失败"**);  
}

#### 表单修改

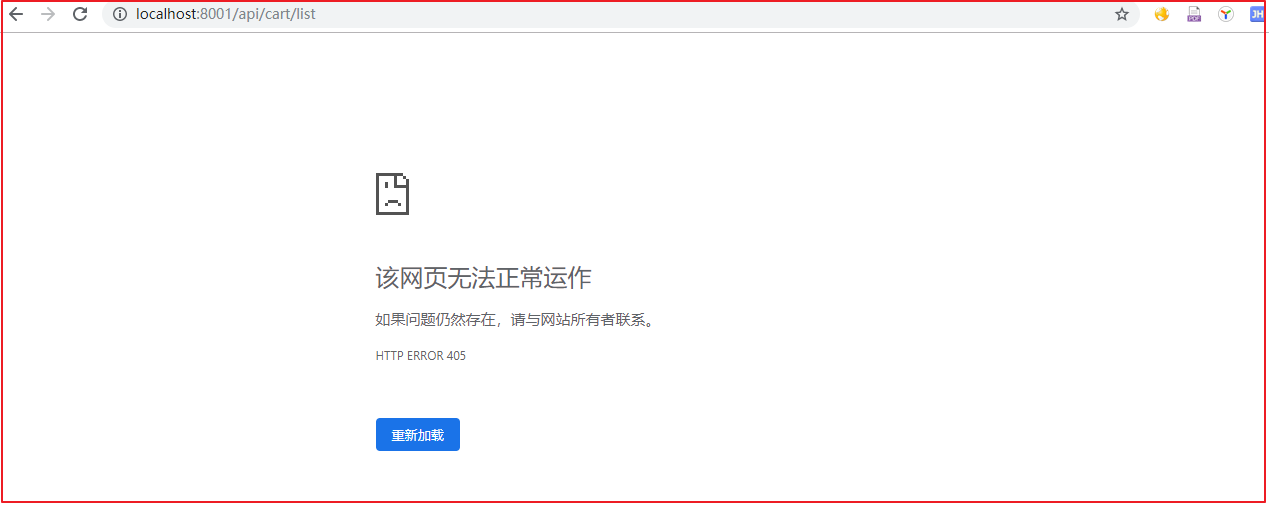
<**form class="sui-form"**>  
 <**div class="input-prepend"**><**span class="add-on loginname"**></**span**>  
 <**input id="inputName" v-model="username" type="text" placeholder="邮箱/用户名/手机号" class="span2 input-xfat"**>  
 </**div**>  
 <**div class="input-prepend"**><**span class="add-on loginpwd"**></**span**>  
 <**input id="inputPassword" v-model="password" type="password" placeholder="请输入密码" class="span2 input-xfat"**>  
 </**div**>  
 <**div class="setting"**>  
 <**label class="checkbox inline"**>  
 <**input name="m1" type="checkbox" value="2" checked=""**>  
 自动登录  
 </**label**>  
 <**span class="forget"**>忘记密码？</**span**>  
 </**div**>  
 <**div class="logined"**>  
 <**div style="text-align**: **center**; **color**: **#ff4103"**>{{msg}}</**div**>  
 <**a class="sui-btn btn-block btn-xlarge btn-danger" @click="login()"**>登**&nbsp;&nbsp;**录</**a**>  
 </**div**>  
</**form**>

#### 测试



## 登录跳转

用户没有登录的时候，我们直接访问购物车，效果如下：



我们可以发现，返回的只是个错误状态码，不方便测试，我们可以重定向到登录页面，让用户登录，我们可以修改网关的头文件，让用户每次没登录的时候，都跳转到登录页面。

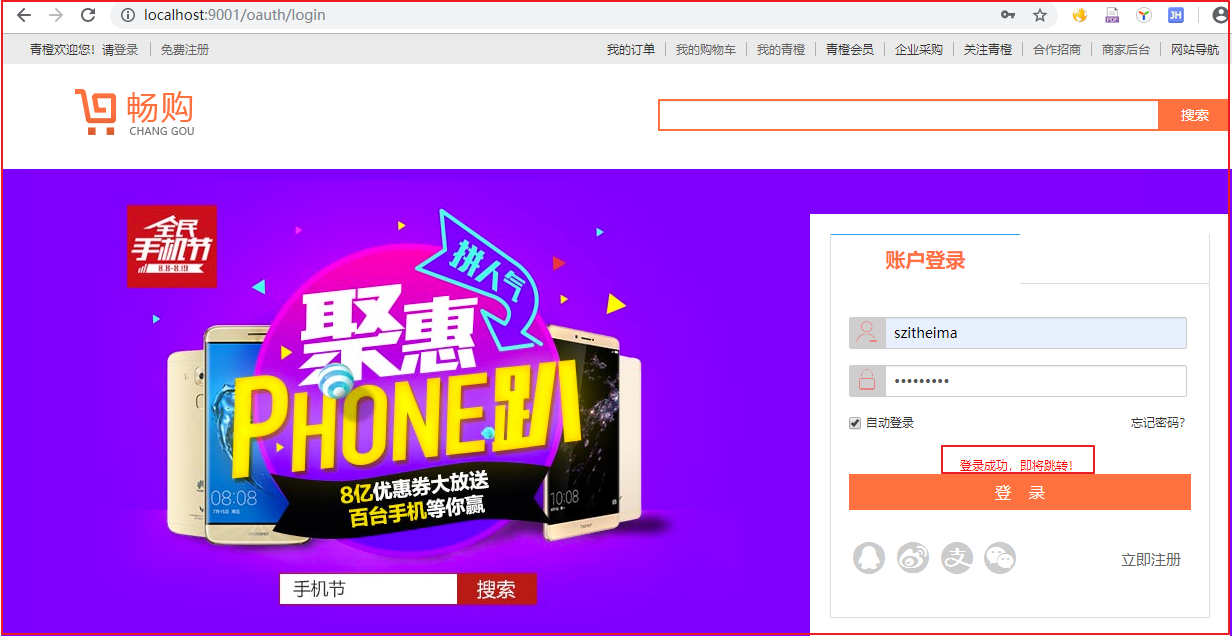
修改changgou-gateway-web的com.changgou.filter.AuthorizeFilter，代码如下：

*//登录url***private static final** String ***USER\_LOGIN\_URL*** = **"http://localhost:9001/oauth/login"**;

*/\*\*  
 \* 拦截请求过滤规则逻辑**\** ***@return*** *\*/*@Override  
**public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  
 *//1、获取Request、Response对象-exchange.get...* ServerHttpRequest request = exchange.getRequest();  
 ServerHttpResponse response = exchange.getResponse();  
 *//2、获取请求的URI-request.getURI().getPath()* String uri = request.getURI().getPath();  
 *//3、如果是登录请求-uri.startsWith，放行-chain.filter* **if**(!URLFilter.*hasAuthorize*(uri)){  
 **return** chain.filter(exchange);  
 }**else**{ *//4、如果是非登录请求  
 //4.1 获取前端传入的令牌-从请求头中获取-request.getHeaders().getFirst* String token = request.getHeaders().getFirst(***AUTHORIZE\_TOKEN***);  
 *//4.2 如果头信息中没有，从请求参数中获取-request.getQueryParams().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 token = request.getQueryParams().getFirst(***AUTHORIZE\_TOKEN***);  
 }  
 *//4.3 如果请求参数中没有，从cookie中获取-request.getCookies().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 HttpCookie first = request.getCookies().getFirst(***AUTHORIZE\_TOKEN***);  
 **if**(first != **null**){  
 token = first.getValue();  
 }  
 }  
 *//4.4 如果以上方式都取不到令牌-返回405错误-response.setStatusCode(405)-return response.setComplete* **if**(StringUtils.*isEmpty*(token)){  
 *//返回405，错误表示方法不允许访问  
 //response.setStatusCode(HttpStatus.METHOD\_NOT\_ALLOWED);* response.setStatusCode(HttpStatus.***SEE\_OTHER***);  
 response.getHeaders().set(**"Location"**, ***USER\_LOGIN\_URL***); *//把请求地址带回去* **return** response.setComplete();  
 }**else**{ *// 4.5 如果获取到了令牌，解析令牌-JwtUtil.parseJWT，放行-chain.filter(exchange)* **try** {  
 *//Claims claims = JwtUtil.parseJWT(token);  
 //4.5.1解析成功-把令牌返回-request.mutate().header(key,value)* request.mutate().header(***AUTHORIZE\_TOKEN***, **"bearer "** + token);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 *//无效的认证* response.setStatusCode(HttpStatus.***UNAUTHORIZED***);  
 **return** response.setComplete();  
 }  
 **return** chain.filter(exchange);  
 }  
 }  
}

此时再测试，就可以跳转到登录页面了。当然，在工作中，这里不能直接跳转到登录页，应该提示状态给页面，让页面根据判断跳转，这里只是为了方便测试。

## 成功登录跳转到原访问页



上面虽然实现了登录跳转，但登录成功后却并没有返回到要访问的购物车页面，我们可以将用户要访问的页面作为参数传递给登录控制器，登录控制器记录下来，每次登录成功后，再跳转记录访问路劲参数指定的页面即可。

### 修改网关携带当前URI

修改changgou-gateway-web的com.changgou.filter.AuthorizeFilter，在之前的URL后面添加FROM参数以及FROM参数的值为request.getURI()，代码如下：

*//4.4 如果以上方式都取不到令牌-返回405错误-response.setStatusCode(405)-return response.setComplete***if**(StringUtils.*isEmpty*(token)){  
 *//返回405，错误表示方法不允许访问  
 //response.setStatusCode(HttpStatus.METHOD\_NOT\_ALLOWED);* response.setStatusCode(HttpStatus.***SEE\_OTHER***);  
 *//拼接url地址* String url = ***USER\_LOGIN\_URL*** + **"?FROM="** + request.getURI();  
 response.getHeaders().set(**"Location"**, url); *//把请求地址带回去* **return** response.setComplete();  
}

### 认证服务器获取FROM参数

修改changgou-user-oauth的com.changgou.oauth.controller.LoginRedirect记录访问来源页，代码如下：

@Controller  
@RequestMapping(**"oauth"**)  
**public class** LoginRedirect {  
  
 */\*\*\*  
 \* 跳转到登录页面  
 \** ***@return*** *\*/* @GetMapping(**"login"**)  
 **public** String login(@RequestParam(value = **"FROM"**,required = **false**)String from, Model model){  
 model.addAttribute(**"from"**, from);  
 **return "login"**;  
 }  
}

修改页面，获取来源页信息，并存到from变量中，登录成功后跳转到该地址。

**data**:{  
 **msg**:**''**, *//登录提示* **username**:**''**, *//用户名* **password**:**''**, *//密码* **from**:[[${from}]] *//需要跳转的地址*},  
**methods**:{  
 *//登录* login:**function** () {  
 **this**.**msg** = **"正在登录..."**;  
 **axios**.post(**"/user/login?username="** + **this**.**username** + **"&password="** + **this**.**password**).then(**function** (response) {  
 **if**(response.**data**.flag){  
 ***app***.**msg** = **"登录成功，即将跳转..."**;  
 *//3秒后跳转之前页页面  
 setTimeout*(**function** () {  
 ***window***.**location**.**href** = ***app***.**from**;  
 }, 3000);  
 }**else**{  
 ***app***.**msg** = response.**data**.**message**;  
 }  
 })  
 }  
}

此时再测试，就可以识别未登录用户，跳转到登录页，然后根据登录状态，如果登录成功，则跳转到来源页。

# 订单结页

## 收件地址分析

用户从购物车页面点击结算，跳转到订单结算页，结算页需要加载用户对应的收件地址，如下图：



表结构分析：

**CREATE** **TABLE** `tb\_address` (

`id` int(11) **NOT** NULL AUTO\_INCREMENT,

`username` varchar(50) **DEFAULT** NULL COMMENT '用户名',

`provinceid` varchar(20) **DEFAULT** NULL COMMENT '省',

`cityid` varchar(20) **DEFAULT** NULL COMMENT '市',

`areaid` varchar(20) **DEFAULT** NULL COMMENT '县/区',

`phone` varchar(20) **DEFAULT** NULL COMMENT '电话',

`address` varchar(200) **DEFAULT** NULL COMMENT '详细地址',

`contact` varchar(50) **DEFAULT** NULL COMMENT '联系人',

`is\_default` varchar(1) **DEFAULT** NULL COMMENT '是否是默认 1默认 0否',

`alias` varchar(50) **DEFAULT** NULL COMMENT '别名',

**PRIMARY** **KEY** (`id`)

) **ENGINE**=**InnoDB** AUTO\_INCREMENT=66 **DEFAULT** **CHARSET**=utf8;

我们可以根据用户登录名去tb\_address表中查询对应的数据。

## 实现用户收件地址查询

### 业务层

**我们可以直接使用生成代码中的findList(Address address)方法，不用再实现。**

### 控制层

修改changgou-service-user微服务，修改com.changgou.user.controller.AddressController，添加根据用户名查询用户收件信息方法，代码如下：

*/\*\*\*\*  
 \* 用户收件地址  
 \*/*@GetMapping(value = **"/user/list"**)  
**public** Result<List<Address>> list() {  
 *//获取用户登录信息* Map<String, String> userMap = TokenDecode.*getUserInfo*();  
 *//获取用户名，构建查询条件* String username = userMap.get(**"username"**);  
 Address where = **new** Address();  
 where.setUsername(username);  
 *//查询用户收件地址* List<Address> addressList = **addressService**.findList(where);  
 **return new** Result(**true**, StatusCode.***OK***, **"查询成功！"**, addressList);  
}

### 测试

访问 <http://localhost:8001/api/address/user/list>



### 添加Feign

测试没问题后，记得提供Feign供其它微服务调用，因为收件人列表展示在订单模块中，用户收件人在service\_user中。

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.user.feign  
 \*/*@FeignClient(name = **"user"**)  
@RequestMapping(**"address"**)  
**public interface** AddressFeign {  
 */\*\*\*\*  
 \* 用户收件地址  
 \*/* @GetMapping(value = **"/user/list"**)  
 **public** Result<List<Address>> list();  
}

## 运送清单

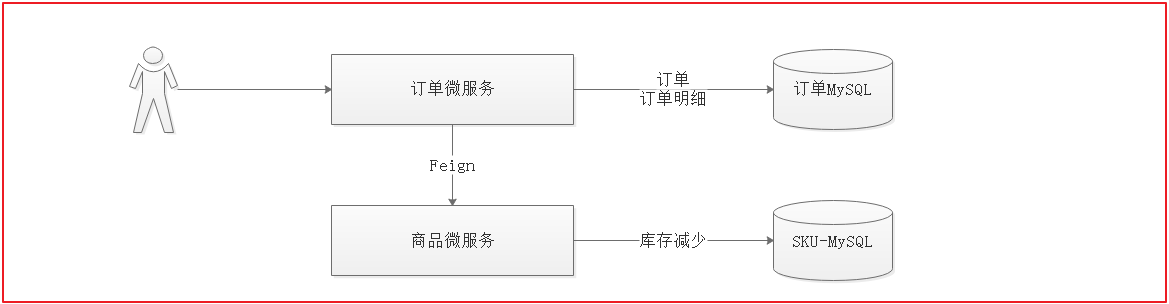


运送清单其实就是购物车列表，直接查询之前的购物车列表即可，这里不做说明了。

# 下单

## 业务分析

点击结算页的时候，会立即创建订单数据，创建订单数据会将数据存入到2张表中，分别是订单表和订单明细表，此处还需要修改商品对应的库存数量。



订单表结构如下：

**CREATE** **TABLE** `tb\_order` (

`id` varchar(50) **COLLATE** utf8\_bin **NOT** NULL COMMENT '订单id',

`total\_num` int(11) **DEFAULT** NULL COMMENT '数量合计',

`total\_money` int(11) **DEFAULT** NULL COMMENT '金额合计',

`pre\_money` int(11) **DEFAULT** NULL COMMENT '优惠金额',

`post\_fee` int(11) **DEFAULT** NULL COMMENT '邮费',

`pay\_money` int(11) **DEFAULT** NULL COMMENT '实付金额',

`pay\_type` varchar(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '支付类型，1、在线支付、0 货到付款',

`create\_time` datetime **DEFAULT** NULL COMMENT '订单创建时间',

`update\_time` datetime **DEFAULT** NULL COMMENT '订单更新时间',

`pay\_time` datetime **DEFAULT** NULL COMMENT '付款时间',

`consign\_time` datetime **DEFAULT** NULL COMMENT '发货时间',

`end\_time` datetime **DEFAULT** NULL COMMENT '交易完成时间',

`close\_time` datetime **DEFAULT** NULL COMMENT '交易关闭时间',

`shipping\_name` varchar(20) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '物流名称',

`shipping\_code` varchar(20) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '物流单号',

`username` varchar(50) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '用户名称',

`buyer\_message` varchar(1000) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '买家留言',

`buyer\_rate` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '是否评价',

`receiver\_contact` varchar(50) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '收货人',

`receiver\_mobile` varchar(12) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '收货人手机',

`receiver\_address` varchar(200) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '收货人地址',

`source\_type` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '订单来源：1:web，2：app，3：微信公众号，4：微信小程序 5 H5手机页面',

`transaction\_id` varchar(30) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '交易流水号',

`order\_status` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '订单状态,0:未完成,1:已完成，2：已退货',

`pay\_status` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '支付状态,0:未支付，1：已支付，2：支付失败',

`consign\_status` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '发货状态,0:未发货，1：已发货，2：已收货',

`is\_delete` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '是否删除',

**PRIMARY** **KEY** (`id`),

**KEY** `create\_time` (`create\_time`),

**KEY** `status` (`order\_status`),

**KEY** `payment\_type` (`pay\_type`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_bin;

订单明细表结构如下：

**CREATE** **TABLE** `tb\_order\_item` (

`id` varchar(50) **COLLATE** utf8\_bin **NOT** NULL COMMENT 'ID',

`category\_id1` int(11) **DEFAULT** NULL COMMENT '1级分类',

`category\_id2` int(11) **DEFAULT** NULL COMMENT '2级分类',

`category\_id3` int(11) **DEFAULT** NULL COMMENT '3级分类',

`spu\_id` varchar(20) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT 'SPU\_ID',

`sku\_id` bigint(20) **NOT** NULL COMMENT 'SKU\_ID',

`order\_id` bigint(20) **NOT** NULL COMMENT '订单ID',

`name` varchar(200) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '商品名称',

`price` int(20) **DEFAULT** NULL COMMENT '单价',

`num` int(10) **DEFAULT** NULL COMMENT '数量',

`money` int(20) **DEFAULT** NULL COMMENT '总金额',

`pay\_money` int(11) **DEFAULT** NULL COMMENT '实付金额',

`image` varchar(200) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '图片地址',

`weight` int(11) **DEFAULT** NULL COMMENT '重量',

`post\_fee` int(11) **DEFAULT** NULL COMMENT '运费',

`is\_return` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '是否退货,0:未退货，1：已退货',

**PRIMARY** **KEY** (`id`),

**KEY** `item\_id` (`sku\_id`),

**KEY** `order\_id` (`order\_id`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_bin;

## 下单实现

下单的时候，先添加订单往tb\_order表中增加数据，再添加订单明细，往tb\_order\_item表中增加数据。

### 创建雪花算法对象

这里先修改changgou-service-order微服务，实现下单操作，这里会生成订单号，我们首先需要在启动类中创建一个IdWorker对象。

在com.changgou.OrderApplication中创建IdWorker，代码如下：

@Bean  
**public** IdWorker getIdWorker(){  
 *//之前在goods模块使用过雪花算法，所以这里数据中心增加1* **return new** IdWorker(0,1);  
}

### 业务层

修改changgou-service-order微服务，修改com.changgou.order.service.impl.OrderServiceImpl的add方法，代码如下：

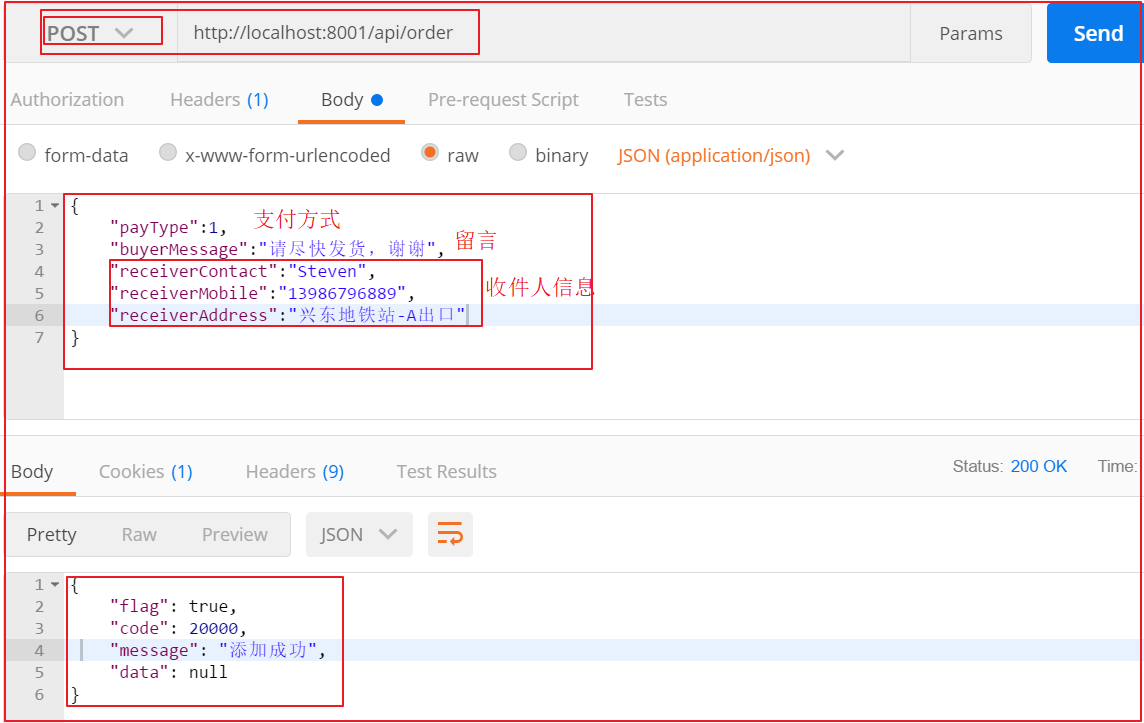
@Autowired  
**private** OrderMapper **orderMapper**;  
@Autowired  
**private** OrderItemMapper **orderItemMapper**;  
@Autowired  
**private** CartService **cartService**;  
@Autowired  
**private** IdWorker **idWorker**;  
@Autowired  
**private** RedisTemplate **redisTemplate**;  
  
*/\*\*  
 \* 增加Order  
 \** ***@param order*** *传入包装的订单相关信息  
 \*/*@Override  
**public void** add(Order order){  
 *//查询出用户的所有购物车* List<OrderItem> orderItems = **cartService**.list(order.getUsername());  
  
 *//基本数据完善* order.setId(**"NO."**+**idWorker**.nextId()); *//订单号* order.setCreateTime(**new** Date());  
 order.setUpdateTime(order.getCreateTime());  
 order.setBuyerRate(**"0"**); *//0:未评价，1：已评价* order.setSourceType(**"1"**); *//来源，1：WEB* order.setOrderStatus(**"0"**); *//0:未完成,1:已完成，2：已退货* order.setPayStatus(**"0"**); *//0:未支付，1：已支付，2：支付失败* order.setConsignStatus(**"0"**); *//0:未发货，1：已发货，2：已收货  
  
 //统计计算* **int** totalMoney = 0; *//总金额* **int** totalPayMoney=0; *//实际支付金额* **int** num = 0; *//总数量  
  
 //添加订单明细* **for** (OrderItem orderItem : orderItems) {  
 orderItem.setId(**"NO."**+**idWorker**.nextId());  
 orderItem.setIsReturn(**"0"**);  
 orderItem.setOrderId(order.getId());  
  
 *//总金额* totalMoney+=orderItem.getMoney();  
 *//实际支付金额* totalPayMoney+=orderItem.getPayMoney();  
 *//总数量* num+=orderItem.getNum();  
 *//保存订单明细* **orderItemMapper**.insertSelective(orderItem);  
 }  
 *//设置统计数据* order.setTotalNum(num);  
 order.setTotalMoney(totalMoney);  
 order.setPayMoney(totalPayMoney);  
 order.setPreMoney(totalMoney-totalPayMoney);  
 *//保存订单* **orderMapper**.insertSelective(order);  
  
 *//清除Redis缓存购物车数据* **redisTemplate**.delete(**"Cart\_"**+order.getUsername());  
}

### 控制层

修改changgou-service-order微服务，修改com.changgou.order.controller.OrderController类，代码如下：

*/\*\*\*  
 \* 新增Order数据  
 \** ***@param order*** *\** ***@return*** *\*/*@PostMapping  
**public** Result add(@RequestBody Order order){  
 *//获取登录名* String username = TokenDecode.*getUserInfo*().get(**"username"**);  
 order.setUsername(username);  
 *//调用OrderService实现添加Order* **orderService**.add(order);  
 **return new** Result(**true**,StatusCode.***OK***,**"添加成功"**);  
}

### 测试



{

"payType":1,

"buyerMessage":"请尽快发货，谢谢",

"receiverContact":"Steven",

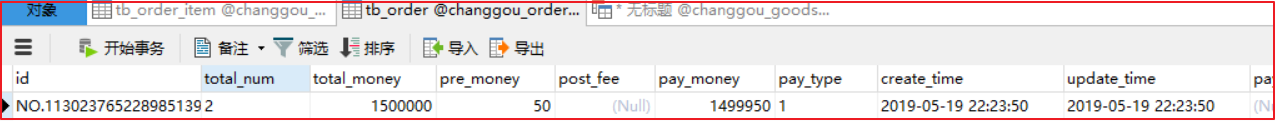
"receiverMobile":"13986796889",

"receiverAddress":"兴东地铁站-A出口"

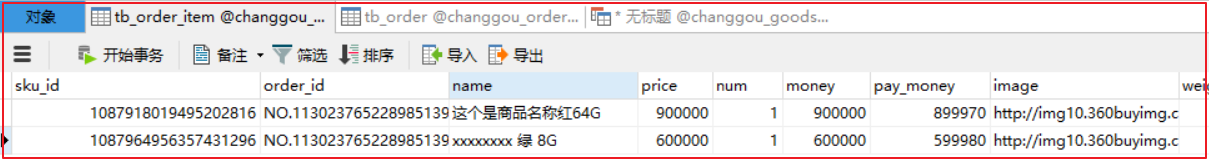
}

保存订单测试，表数据变化如下：

tb\_order表数据：



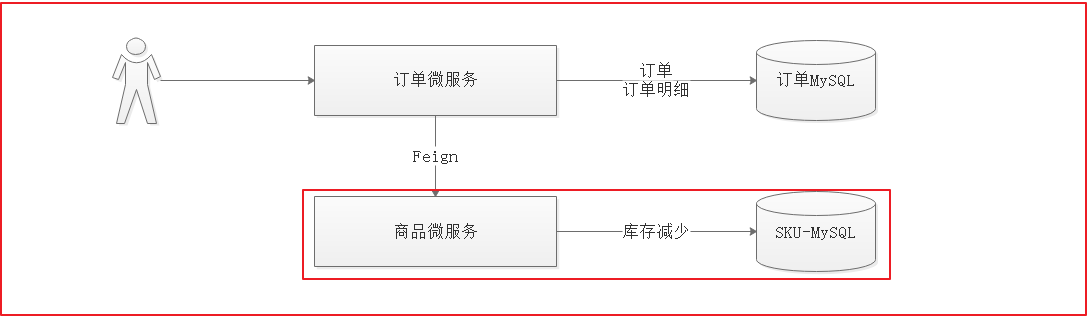
tb\_order\_item表数据：



## 库存变更

### 业务分析

上面操作只实现了下单操作，但对应的库存还没跟着一起减少，我们在下单之后，应该调用商品微服务，将下单的商品库存减少，销量增加。每次订单微服务只需要将用户名传到商品微服务，商品微服务通过用户名到Redis中查询对应的购物车数据，然后执行库存减少，库存减少需要控制当前商品库存>=销售数量。



如何控制库存数量>=销售数量呢？其实可以通过SQL语句实现，每次减少数量的时候，加个条件判断。

where num>=#{num}即可。

update tb\_sku set num = num - 1, sale\_num = sale\_num + 1 where id = 1148477873196113920 and num > 1

订单工程中一会儿需要调用商品微服务api，在订单微服务的pom.xml中添加如下依赖：

<**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_goods\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
</**dependency**>

### 配置容断器模式与Feign拦截器

**要调用其他微服务，需要将头文件中的令牌数据携带到其他微服务中取，所以我们不能使用hystrix的多线程模式，**修改changgou-service-order的applicatin.yml配置，代码如下：

*#hystrix 配置***hystrix**:  
 **command**:  
 **default**:  
 **execution**:  
 **isolation**:  
 **thread**:  
 *# 熔断超时设置，默认为1秒* **timeoutInMilliseconds**: 10000  
 *# 隔离模式使用信号量隔离SEMAPHORE 默认值是线程隔离THREAD* **strategy**: SEMAPHORE

每次还需要使用拦截器添加头文件信息，修改配置类com.changgou.OrderApplication添加拦截器，代码如下：

@Bean  
**public** FeignInterceptor feignInterceptor(){  
 **return new** FeignInterceptor();  
}

### Dao层

修改changgou-service-order微服务的com.changgou.goods.dao.SkuMapper接口，增加库存递减方法,代码如下：

*/\*\*  
 \* 递减库存  
 \** ***@param num*** *扣减数量  
 \** ***@param skuId*** *购买商品skuId  
 \** ***@return*** *\*/*@Update(**"UPDATE tb\_sku SET num = num - #{num},sale\_num = sale\_num + #{num} WHERE id = #{skuId} AND num > #{num}"**)  
**int** decrCount(@Param(**"num"**) Integer num, @Param(**"skuId"**) Long skuId);

### 业务层

修改changgou-service-goods微服务的com.changgou.goods.service.SkuService接口，添加如下方法：

*/\*\*\*  
 \* 库存递减  
 \** ***@param username*** *登录名  
 \*/***void** decrCount(String username);

修改changgou-service-goods微服务的com.changgou.goods.service.impl.SkuServiceImpl实现类，添加一个实现方法，代码如下：

@Autowired  
**private** RedisTemplate **redisTemplate**;  
@Override  
**public void** decrCount(String username) {  
 *//获取购物车数据* List<OrderItem> orderItems = **redisTemplate**.boundHashOps(**"Cart\_"** + username).values();  
  
 *//循环递减* **for** (OrderItem orderItem : orderItems) {  
 *//递减库存* **int** count = **skuMapper**.decrCount(orderItem.getNum(),orderItem.getSkuId());  
 **if**(count<=0){  
 **throw new** RuntimeException(**"库存不足，无法购买！"**);  
 }  
 }  
}

记得把order-api的依赖添加到service-goods中

<**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_order\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
</**dependency**>

在application.yml中配置redis

**spring**:  
 **application**:  
 **name**: goods  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_goods?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 **username**: root  
 **password**: 123456  
 **redis**:  
 **host**: 192.168.211.132

### 控制层

修改changgou-service-goods的com.changgou.goods.controller.SkuController类，添加库存递减方法，代码如下：

*/\*\*\*  
 \* 库存递减  
 \** ***@param username*** *注意，由于我们此方法由order调起，所以此处的参数可以从service-order传入  
 \** ***@return*** *\*/*@PostMapping(value = **"/decr/count/{username}"**)  
**public** Result decrCount(@PathVariable String username){  
 *//库存递减* **skuService**.decrCount(username);  
 **return new** Result(**true**,StatusCode.***OK***,**"库存递减成功！"**);  
}

### 创建feign

同时在changgou-service-goods-api工程添加com.changgou.goods.feign.SkuFeign的实现，代码如下：

*/\*\*\*  
 \* 库存递减  
 \** ***@param username*** *注意，由于我们此方法由order调起，所以此处的参数可以从service-order传入  
 \** ***@return*** *\*/*@PostMapping(value = **"/decr/count/{username}"**)  
**public** Result decrCount(@PathVariable String username);

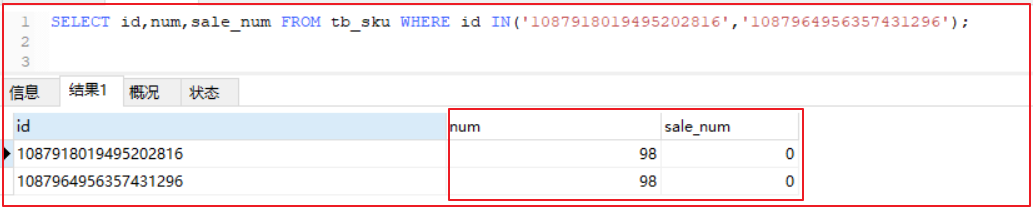
### 调用库存递减

修改changgou-service-order微服务的com.changgou.order.service.impl.OrderServiceImpl类的add方法，增加库存递减的调用。

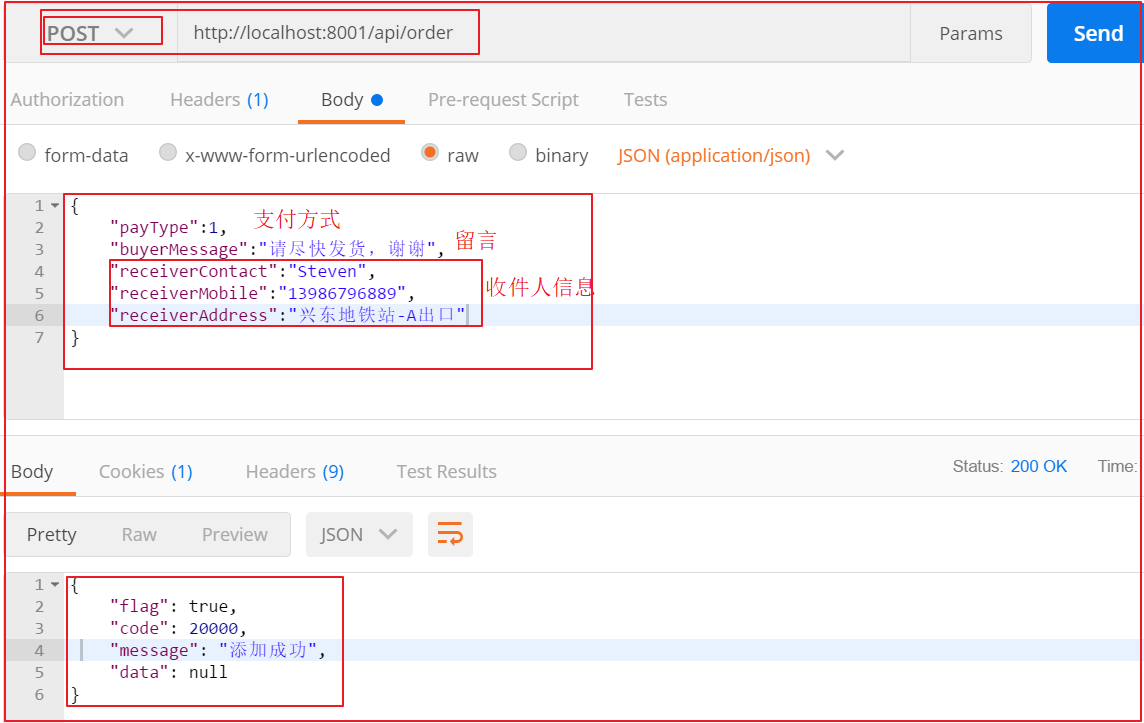
@Autowired  
**private** OrderMapper **orderMapper**;  
@Autowired  
**private** OrderItemMapper **orderItemMapper**;  
@Autowired  
**private** CartService **cartService**;  
@Autowired  
**private** IdWorker **idWorker**;  
@Autowired  
**private** RedisTemplate **redisTemplate**;  
@Autowired  
**private** SkuFeign **skuFeign**;  
  
*/\*\*  
 \* 增加Order  
 \** ***@param order*** *传入包装的订单相关信息  
 \*/*@Override  
**public void** add(Order order){  
 *//查询出用户的所有购物车* List<OrderItem> orderItems = **cartService**.list(order.getUsername());  
  
 *//基本数据完善* order.setId(**"NO."**+**idWorker**.nextId()); *//订单号* order.setCreateTime(**new** Date());  
 order.setUpdateTime(order.getCreateTime());  
 order.setBuyerRate(**"0"**); *//0:未评价，1：已评价* order.setSourceType(**"1"**); *//来源，1：WEB* order.setOrderStatus(**"0"**); *//0:未完成,1:已完成，2：已退货* order.setPayStatus(**"0"**); *//0:未支付，1：已支付，2：支付失败* order.setConsignStatus(**"0"**); *//0:未发货，1：已发货，2：已收货  
  
 //统计计算* **int** totalMoney = 0; *//总金额* **int** totalPayMoney=0; *//实际支付金额* **int** num = 0; *//总数量  
  
 //添加订单明细* **for** (OrderItem orderItem : orderItems) {  
 orderItem.setId(**"NO."**+**idWorker**.nextId());  
 orderItem.setIsReturn(**"0"**);  
 orderItem.setOrderId(order.getId());  
  
 *//总金额* totalMoney+=orderItem.getMoney();  
 *//实际支付金额* totalPayMoney+=orderItem.getPayMoney();  
 *//总数量* num+=orderItem.getNum();  
 *//保存订单明细* **orderItemMapper**.insertSelective(orderItem);  
 }  
 *//设置统计数据* order.setTotalNum(num);  
 order.setTotalMoney(totalMoney);  
 order.setPayMoney(totalPayMoney);  
 order.setPreMoney(totalMoney-totalPayMoney);  
 *//保存订单* **orderMapper**.insertSelective(order);  
   
 *//必须要删除缓存前调用-扣减库存* **skuFeign**.decrCount(order.getUsername());  
  
 *//清除Redis缓存购物车数据* **redisTemplate**.delete(**"Cart\_"**+order.getUsername());  
}

### 测试

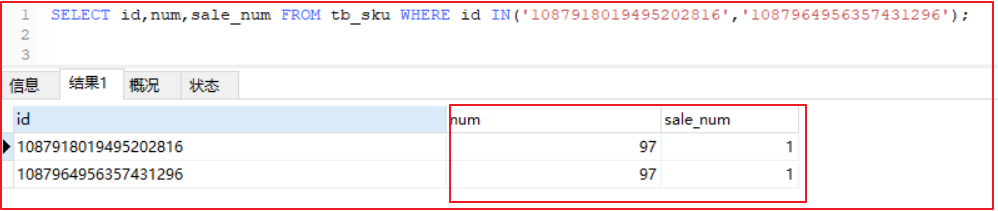
库存减少前，查询数据库Sku数据如下：个数98，销量0



使用Postman执行 <http://localhost:8001/api/order>



执行测试后，剩余库存97，销量1



## 增加积分

比如每次下单完成之后，给用户增加10个积分，支付完成后赠送优惠券，优惠券可用于支付时再次抵扣。我们先完成增加积分功能。如下表：points表示用户积分

**CREATE** **TABLE** `tb\_user` (

`username` varchar(50) **NOT** NULL COMMENT '用户名',

`password` varchar(100) **NOT** NULL COMMENT '密码，加密存储',

`phone` varchar(20) **DEFAULT** NULL COMMENT '注册手机号',

`email` varchar(50) **DEFAULT** NULL COMMENT '注册邮箱',

`created` datetime **NOT** NULL COMMENT '创建时间',

`updated` datetime **NOT** NULL COMMENT '修改时间',

`source\_type` varchar(1) **DEFAULT** NULL COMMENT '会员来源：1:PC，2：H5，3：Android，4：IOS',

`nick\_name` varchar(50) **DEFAULT** NULL COMMENT '昵称',

`name` varchar(50) **DEFAULT** NULL COMMENT '真实姓名',

`status` varchar(1) **DEFAULT** NULL COMMENT '使用状态（1正常 0非正常）',

`head\_pic` varchar(150) **DEFAULT** NULL COMMENT '头像地址',

`qq` varchar(20) **DEFAULT** NULL COMMENT 'QQ号码',

`is\_mobile\_check` varchar(1) **DEFAULT** '0' COMMENT '手机是否验证 （0否 1是）',

`is\_email\_check` varchar(1) **DEFAULT** '0' COMMENT '邮箱是否检测（0否 1是）',

`sex` varchar(1) **DEFAULT** '1' COMMENT '性别，1男，0女',

`user\_level` int(11) **DEFAULT** NULL COMMENT '会员等级',

`points` int(11) **DEFAULT** NULL COMMENT '积分',

`experience\_value` int(11) **DEFAULT** NULL COMMENT '经验值',

`birthday` datetime **DEFAULT** NULL COMMENT '出生年月日',

`last\_login\_time` datetime **DEFAULT** NULL COMMENT '最后登录时间',

**PRIMARY** **KEY** (`username`),

**UNIQUE** **KEY** `username` (`username`) **USING** BTREE

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8 COMMENT='用户表';

### dao层

修改changgou-service-user微服务的com.changgou.user.dao.UserMapper接口，增加用户积分方法，代码如下：

*/\*\*\*  
 \* 增加用户积分  
 \** ***@param username*** *\** ***@param pint*** *\** ***@return*** *\*/*@Update(**"UPDATE tb\_user SET points=points+#{point} WHERE username=#{username}"**)  
**int** addUserPoints(@Param(**"username"**) String username, @Param(**"point"**) Integer pint);

### 业务层

修改changgou-service-user微服务的com.changgou.user.service.UserService接口，代码如下：

*/\*\*\*  
 \* 添加用户积分  
 \** ***@param username*** *\** ***@param pint*** *\** ***@return*** *\*/***int** addUserPoints(String username,Integer pint);

修改changgou-service-user微服务的com.changgou.user.service.impl.UserServiceImpl，增加添加积分方法实现，代码如下：

*/\*\*\*  
 \* 修改用户积分  
 \** ***@param username*** *\** ***@param pint*** *\** ***@return*** *\*/*@Override  
**public int** addUserPoints(String username, Integer pint) {  
 **return userMapper**.addUserPoints(username,pint);  
}

### 控制层

修改changgou-service-user微服务的com.changgou.user.controller.UserController，添加增加用户积分方法，代码如下：

*/\*\*\*  
 \* 增加用户积分  
 \** ***@param points****:要添加的积分  
 \*/*@GetMapping(value = **"/points/add"**)  
**public** Result addPoints(@RequestParam(value = **"points"**) Integer points){  
 *//获取用户名* Map<String, String> userMap = TokenDecode.*getUserInfo*();  
 String username = userMap.get(**"username"**);  
 *//添加积分* **userService**.addUserPoints(username,points);  
 **return new** Result(**true**,StatusCode.***OK***,**"添加积分成功！"**);  
}

### Feign添加

修改changgou-service-user-api工程，修改com.changgou.user.feign.UserFeign，添加增加用户积分方法，代码如下：

*/\*\*\*  
 \* 增加用户积分  
 \** ***@param points****:要添加的积分  
 \*/*@GetMapping(value = **"/points/add"**)  
**public** Result addPoints(@RequestParam(value = **"points"**) Integer points);

### 增加积分调用

修改changgou-service-order，添加changgou-service-user-api的依赖，修改pom.xml,添加如下依赖：

<**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_user\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
</**dependency**>

在增加订单的时候，同时添加用户积分，修改changgou-service-order微服务的com.changgou.order.service.impl.OrderServiceImpl下单方法，增加调用添加积分方法，代码如下：

@Autowired  
**private** UserFeign **userFeign**;  
*/\*\*  
 \* 增加Order  
 \** ***@param order*** *传入包装的订单相关信息  
 \*/*@Override  
**public void** add(Order order){  
 *//省略其它代码………*  
 *//必须要删除缓存前调用-扣减库存* **skuFeign**.decrCount(order.getUsername());  
 *//添加积分* **userFeign**.addPoints(10);  
 *//清除Redis缓存购物车数据* **redisTemplate**.delete(**"Cart\_"**+order.getUsername());  
}

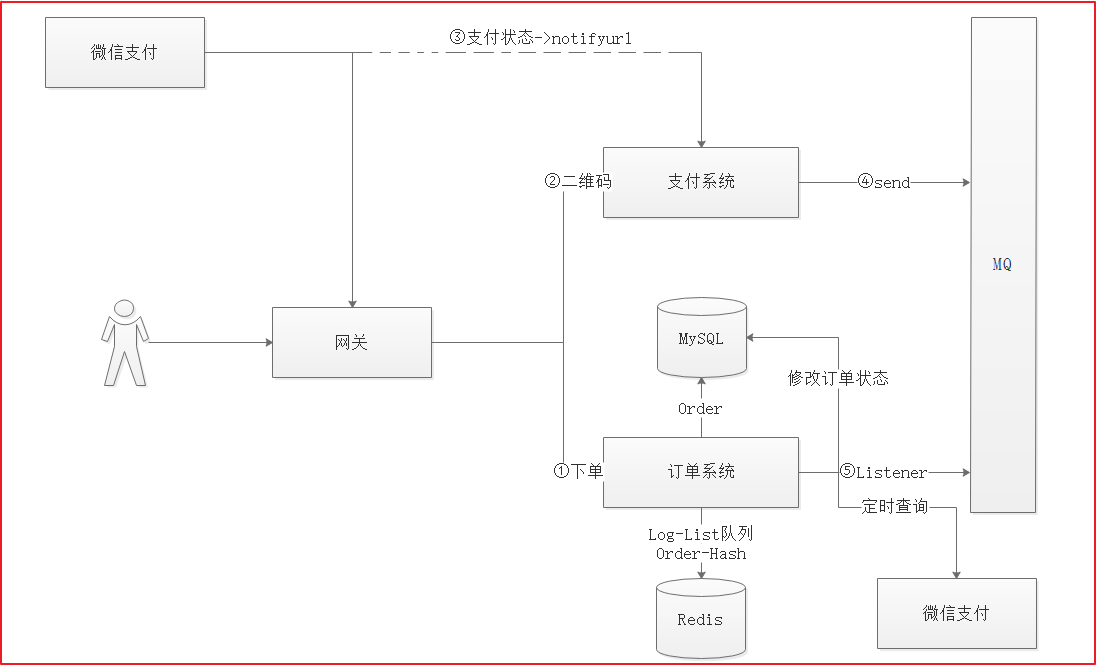
### 启动引导类添加Feign包

修改changgou-service-order的启动类com.changgou.OrderApplication，添加feign的包路径：

@SpringBootApplication  
@EnableEurekaClient  
@MapperScan(basePackages = {**"com.changgou.order.dao"**})  
@EnableFeignClients(basePackages = {**"com.changgou.goods.feign"**,**"com.changgou.user.feign"**})  
**public class** OrderApplication {

# 支付流程分析

## 订单支付分析



如上图，步骤分析如下：

1.用户下单之后，订单数据会存入到MySQL中，同时会将订单对应的支付日志存入到Redis，以队列的方式存储。

2.用户下单后，进入支付页面，支付页面调用支付系统，从微信支付获取二维码数据，并在页面生成支付二维码。

3.用户扫码支付后，微信支付服务器会通调用前预留的回调地址，并携带支付状态信息。

4.支付系统接到支付状态信息后，将支付状态信息发送给RabbitMQ

5.订单系统监听RabbitMQ中的消息获取支付状态，并根据支付状态修改订单状态

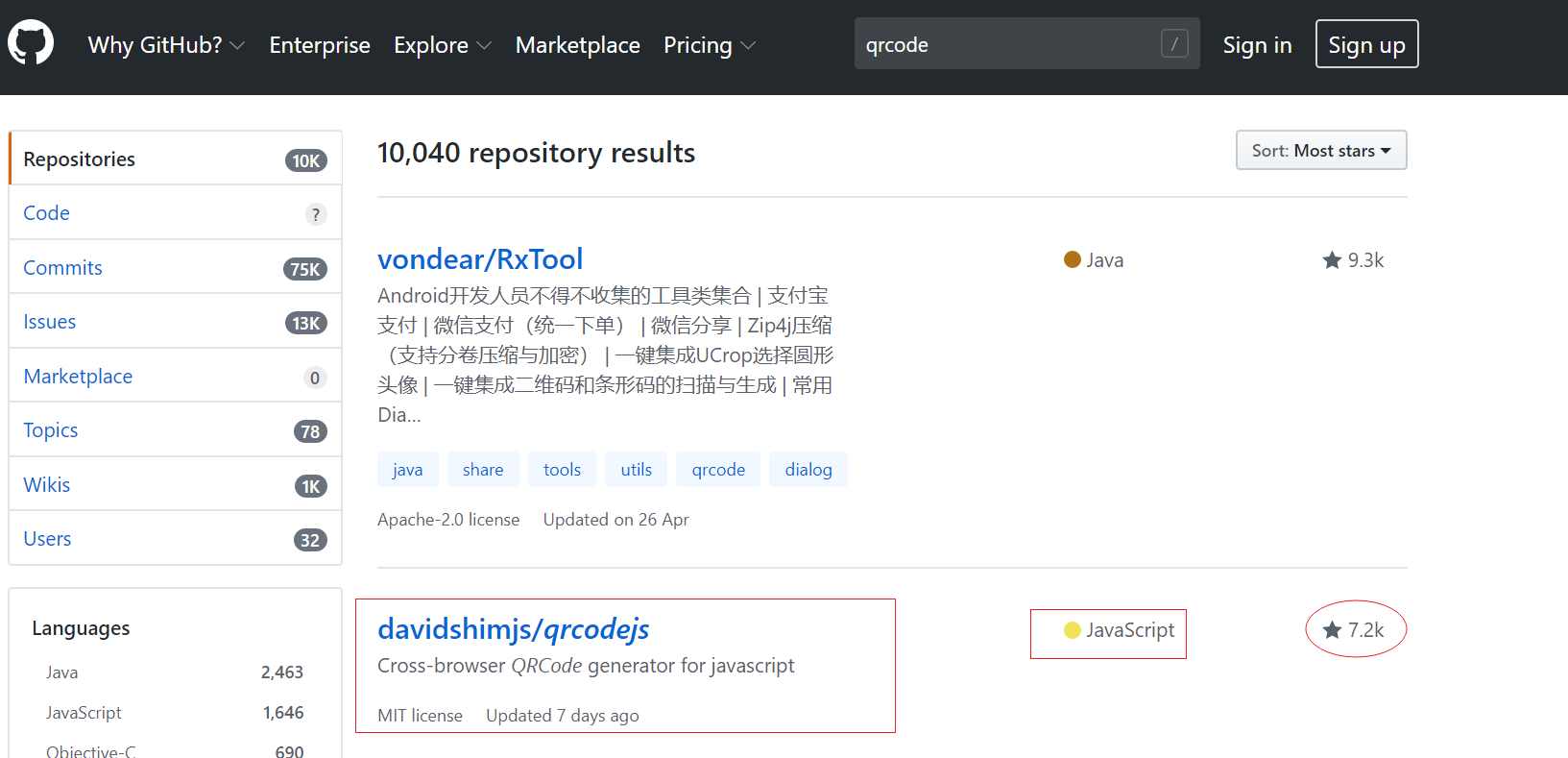
6.为了防止网络问题导致notifyurl没有接到对应数据，定时任务定时获取Redis中队列数据去微信支付接口查询状态，并定时更新对应状态。

## 二维码创建(了解)

今天主要讲微信支付，后面为了看到效果，我们简单说下利用qrcode制作二维码插件。

qrcode是一款基于支持跨浏览器的HTML5 Canvas(画布-图像容器)的纯JS二维码生成插件。通过qrcode.js可以快速生成各种二维码，你可以控制二维码的尺寸颜色与内容。

qrcode在github二维码搜索热度排名中JavaScript语言第一名



qrcode.js二维码插件的可用配置参数如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 默认值 | 描述 |
| text | String | "" | 需要编码为二维码的值 |
| width | Number | 256 | 二维码宽度的尺寸，单位像素。 |
| height | Number | 256 | 二维码宽度的尺寸，单位像素。 |
| colorDark | String | "#000000"-黑色 | 二维码的前景颜色。 |
| colorLight | String | "#ffffff"-白色 | 二维码的背景颜色。 |
| correctLevel | QRCode.CorrectLevel | "H" | 二维码的误差校正级别(L, M, Q, H)。 |

误差校正级别介绍：

L级（低） 7％的码字可以被恢复。

M级（中） 的码字的15％可以被恢复。

Q级（四分）的码字的25％可以被恢复。

H级（高） 的码字的30％可以被恢复。

下面的代码即可生成一张二维码

|  |
| --- |
| <html>  <head>  <title>二维码入门小demo</title>  <script type="text/javascript" src="qrcode.min.js"></script>  </head>  <body>  <div id="qrcode"></div>  <script type="text/javascript">  var qrcode = new QRCode(document.getElementById("qrcode"), {  text: "http://www.itheima.com",  width: 200,  height: 200,  colorDark : "#000000",  colorLight : "#ffffff",  correctLevel : QRCode.CorrectLevel.H  });  </script>  </body>  </html> |

运行效果：



大家掏出手机，扫一下看看是否会看到黑马的官网呢？

# 微信扫码支付简介

## 微信扫码支付申请

微信扫码支付是商户系统按微信支付协议生成支付二维码，用户再用微信“扫一扫”完成支付的模式。该模式适用于PC网站支付、实体店单品或订单支付、媒体广告支付等场景。

申请步骤：（了解）

**第一步：注册公众号（类型须为：服务号）**

请根据营业执照类型选择以下主体注册：[个体工商户](http://kf.qq.com/faq/120911VrYVrA151009JB3i2Q.html)| [企业/公司](http://kf.qq.com/faq/120911VrYVrA151013MfYvYV.html)| [政府](http://kf.qq.com/faq/161220eaAJjE161220IJn6zU.html)| [媒体](http://kf.qq.com/faq/161220IFBJFv161220YnqAbQ.html)| [其他类型](http://kf.qq.com/faq/120911VrYVrA151013nYFZ7Z.html)。

**第二步：认证公众号**

公众号认证后才可申请微信支付，认证费：300元/次。

**第三步：提交资料申请微信支付**

登录公众平台，点击左侧菜单【微信支付】，开始填写资料等待审核，审核时间为1-5个工作日内。

**第四步：开户成功，登录商户平台进行验证**

资料审核通过后，请登录联系人邮箱查收商户号和密码，并登录商户平台填写财付通备付金打的小额资金数额，完成账户验证。

**第五步：在线签署协议**

本协议为线上电子协议，签署后方可进行交易及资金结算，签署完立即生效。

本课程已经提供好“传智播客”的微信支付账号，学员无需申请。

## 开发文档

微信支付接口调用的整体思路：

按API要求组装参数，以XML方式发送（POST）给微信支付接口（URL）,微信支付接口也是以XML方式给予响应。程序根据返回的结果（其中包括支付URL）生成二维码或判断订单状态。

在线微信支付开发文档：

<https://pay.weixin.qq.com/wiki/doc/api/index.html>

我们在本章课程中会用到”统一下单”和”查询订单”两组API

1. appid：微信公众账号或开放平台APP的唯一标识

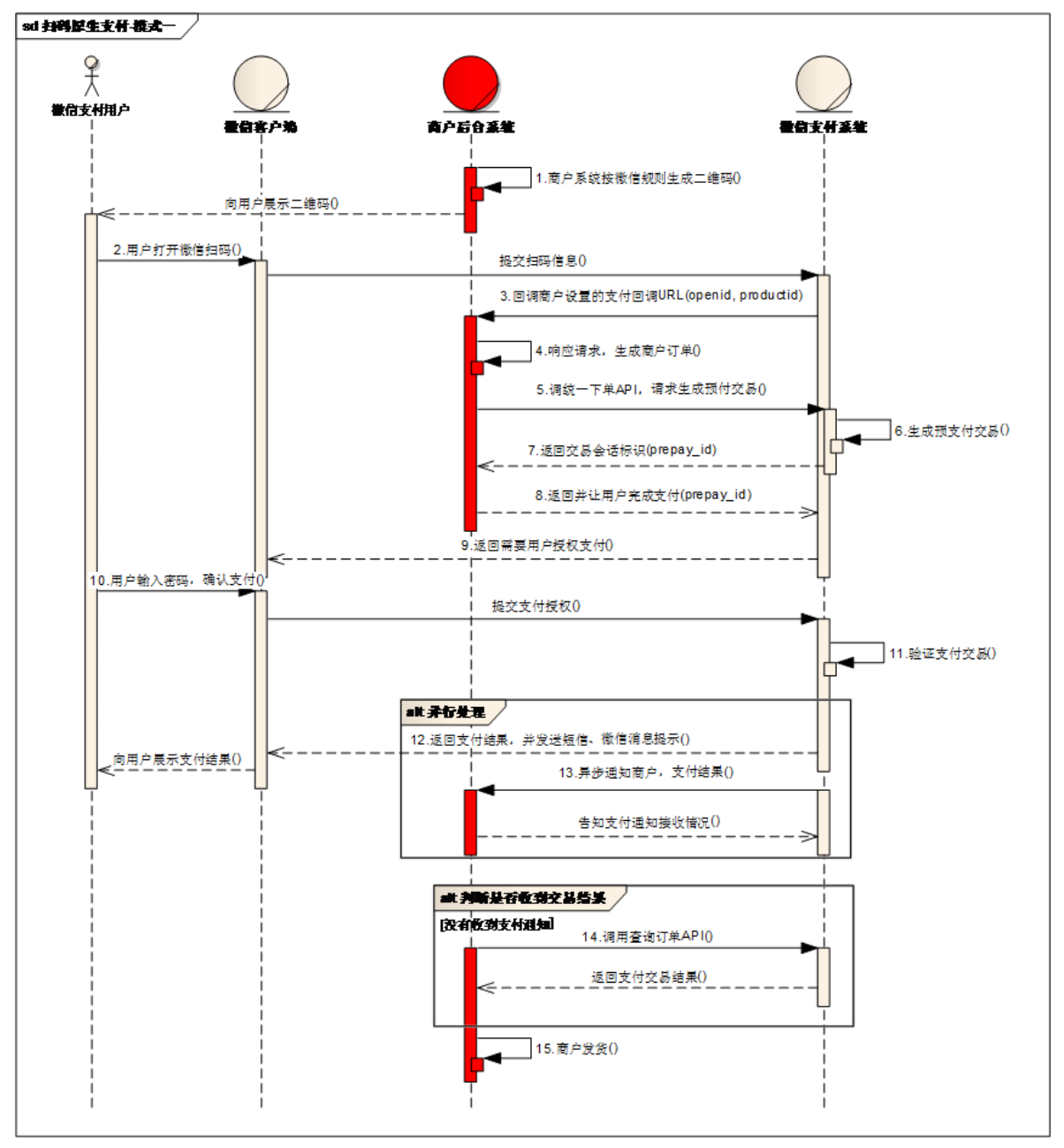
2. mch\_id：商户号 (配置文件中的partner)

3. partnerkey：商户密钥

4. sign:数字签名, 根据微信官方提供的密钥和一套算法生成的一个加密信息, 就是为了保证交易的安全性

## 微信支付模式介绍

### 模式一



业务流程说明：

1.商户后台系统根据微信支付规定格式生成二维码（规则见下文），展示给用户扫码。

2.用户打开微信“扫一扫”扫描二维码，微信客户端将扫码内容发送到微信支付系统。

3.微信支付系统收到客户端请求，发起对商户后台系统支付回调URL的调用。调用请求将带productid和用户的openid等参数，并要求商户系统返回交数据包,详细请见"本节3.1回调数据输入参数"

4.商户后台系统收到微信支付系统的回调请求，根据productid生成商户系统的订单。

5.商户系统调用微信支付【统一下单API】请求下单，获取交易会话标识（prepay\_id）

6.微信支付系统根据商户系统的请求生成预支付交易，并返回交易会话标识（prepay\_id）。

7.商户后台系统得到交易会话标识prepay\_id（2小时内有效）。

8.商户后台系统将prepay\_id返回给微信支付系统。返回数据见"本节3.2回调数据输出参数"

9.微信支付系统根据交易会话标识，发起用户端授权支付流程。

10.用户在微信客户端输入密码，确认支付后，微信客户端提交支付授权。

11.微信支付系统验证后扣款，完成支付交易。

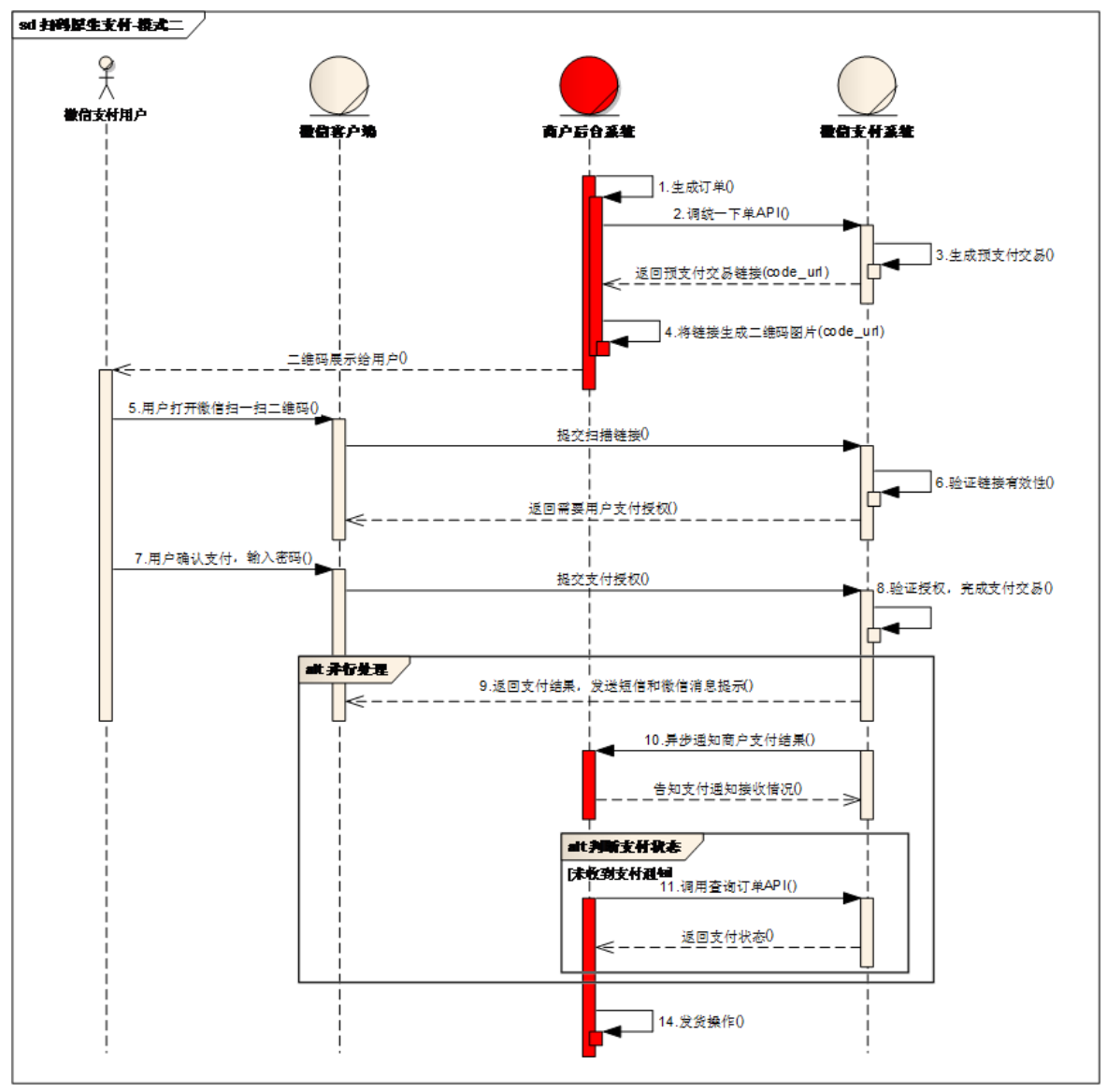
12.微信支付系统完成支付交易后给微信客户端返回交易结果，并将交易结果通过短信、微信消息提示用户。微信客户端展示支付交易结果页面。

13.微信支付系统通过发送异步消息通知商户后台系统支付结果。商户后台系统需回复接收情况，通知微信后台系统不再发送该单的支付通知。

14.未收到支付通知的情况，商户后台系统调用【查询订单API】。

15.商户确认订单已支付后给用户发货。

### 模式二



业务流程说明：

1.商户后台系统根据用户选购的商品生成订单。

2.用户确认支付后调用微信支付【统一下单API】生成预支付交易；

3.微信支付系统收到请求后生成预支付交易单，并返回交易会话的二维码链接code\_url。

4.商户后台系统根据返回的code\_url生成二维码。

5.用户打开微信“扫一扫”扫描二维码，微信客户端将扫码内容发送到微信支付系统。

6.微信支付系统收到客户端请求，验证链接有效性后发起用户支付，要求用户授权。

7.用户在微信客户端输入密码，确认支付后，微信客户端提交授权。

8.微信支付系统根据用户授权完成支付交易。

9.微信支付系统完成支付交易后给微信客户端返回交易结果，并将交易结果通过短信、微信消息提示用户。微信客户端展示支付交易结果页面。

10.微信支付系统通过发送异步消息通知商户后台系统支付结果。商户后台系统需回复接收情况，通知微信后台系统不再发送该单的支付通知。

11.未收到支付通知的情况，商户后台系统调用【查询订单API】。

12.商户确认订单已支付后给用户发货。