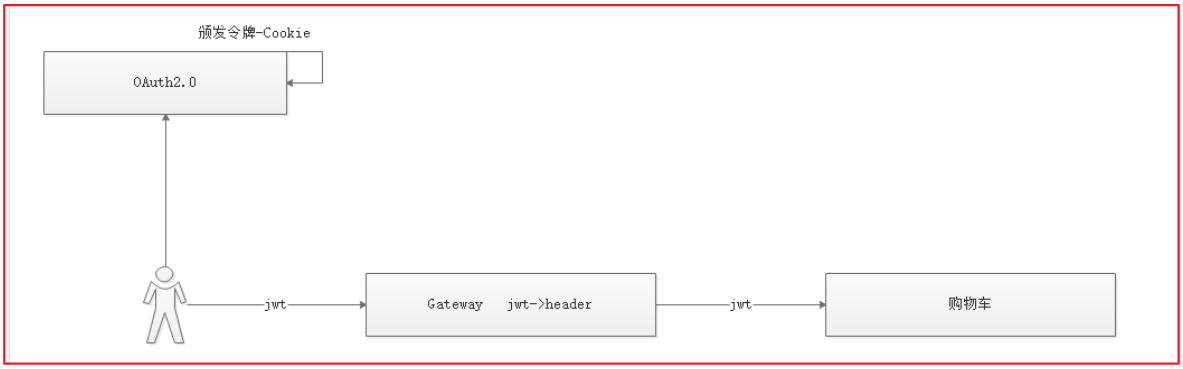
第11章 订单

学习目标

- 登录会话保持
- 购物车配置
- 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

1 登录会话保持

1.1 登录信息Cookie保存



用户登录后，我们将用户信息存入到Cookie中，微服务网关将Cookie中的令牌取出，并添加到请求头中，再继续访问其他微服务，这样就可以保持会话了。

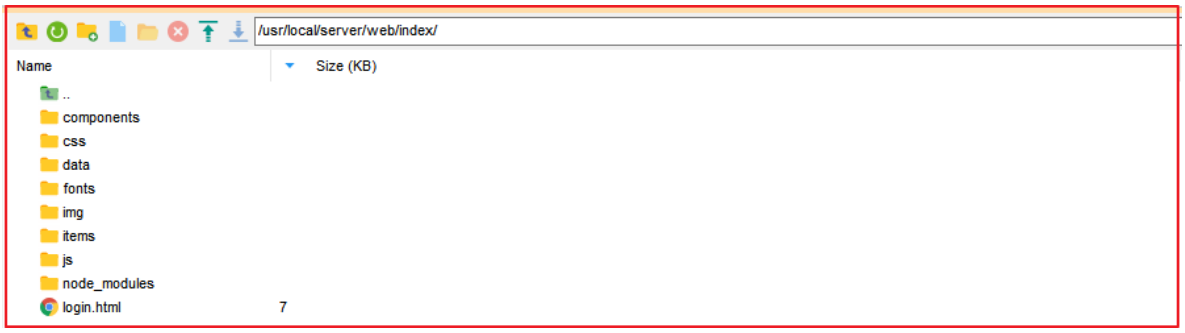
这里需要添加Cookie，在今天资料中有一个工具叫`CookieTools.java`，可以用它实现Cookie的操作。

修改`changgou-user-auth`的登录方法，添加将令牌数据存入到Cookie中的代码，代码如下：

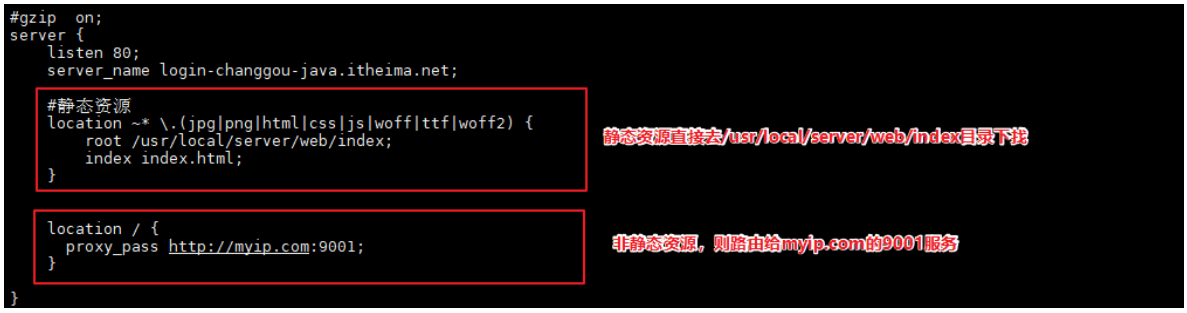
```
/****
 * 登录方法
 */
@PostMapping(value = "/login")
public Result login(String username, String password, HttpServletRequest request, HttpServletResponse response){
    //登录
    AuthToken authToken = authService.login( grantType: "password", username, password, clientId, secret);

    //将令牌信息存入到Cookie中
    CookieTools.setCookie(request, response, cookieName: "Authorization", authToken.getAccessToken());
    //用户名显示
    CookieTools.setCookie(request, response, cookieName: "cuname", username);
    return new Result( flag: true, StatusCode.OK, message: "登录成功！", authToken);
}
```

将资料中的登录页面和样式文件上传到虚拟机的`/usr/local/server/web/index`目录下，并在nginx中配置`login-changgou-java.itheima.net`域名

上传的文件：



修改nginx配置：



上图配置如下：

```
server {
    listen 80;
    server_name login-changgou-java.itheima.net;
    #静态资源
    location ~* \.(jpg|png|html|css|js|woff|ttf|woff2) {
        root /usr/local/server/web/index;
        index index.html;
    }

    location / {
      proxy_pass http://myip.com:9001;
    }
}
```

1.2 微服务网关令牌处理

微服务网关从请求头中和参数中如果都没有获取到令牌，则从Cookie中获取令牌，并将令牌加入到请求头中，再放行，这样其他微服务就可以获取令牌数据了。

```java
@Component
public class AuthorizeFilter implements GlobalFilter,Ordered {

    //令牌头名字
    private static final String AUTHORIZE_TOKEN = "Authorization";

    /**
     * 过滤拦截
     * @param exchange
     * @param chain
     * @return
     */
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        //获取request和response
        ServerHttpRequest request = exchange.getRequest();
        ServerHttpResponse response = exchange.getResponse();

        //获取用户请求的地址
        String path = request.getURI().getPath();

        // /api/user/login放行
        if(path.equals("/api/user/login")){
            //放行
            return chain.filter(exchange);
        }

        //获取用户请求头中的令牌
        String token = request.getHeaders().getFirst(AUTHORIZE_TOKEN); //获取请求头中第1个Authorization参数

        //如果请求头中没有令牌，则有可能用的是参数传入的
        if(StringUtils.isEmpty(token)){
            token = request.getQueryParams().getFirst(AUTHORIZE_TOKEN);//获取请求参数中第1个Authorization
        }

        //如果其他地方都没有令牌，则从Cookie中获取令牌
        if(StringUtils.isEmpty(token)){
            HttpCookie cookie = request.getCookies().getFirst(AUTHORIZE_TOKEN);
            if(cookie==null){
                //状态吗  401
                response.setStatusCode(HttpStatus.UNAUTHORIZED);
                //结束当前请求
                return response.setComplete();
            }
            //获取Cookie中的令牌
            token = cookie.getValue();
            //将令牌存入到请求头中
            request.mutate().header(AUTHORIZE_TOKEN,"bearer "+token);
        }

        return chain.filter(exchange);
    }

    /**
     * 排序
     * @return
     */
    @Override
    public int getOrder() { return 0; }
}
```

2 购物车配置

2.1 购物车配置

购物车对应的域名为`order-changgou-java.itheima.net`,我们可以将购物车静态页面cart.html上传到虚拟机的`/usr/local/server/web/index`目录下，并配置该域名：

修改nginx配置文件：

```
server {
    listen 80;
    server_name order-changgou-java.itheima.net;

    #静态资源
    location ~* \.(jpg|png|html|css|js|woff|ttf|woff2) {
        root /usr/local/server/web/index;
        index index.html;
    }

    location / {
      proxy_pass http://myip.com:8001;
    }
}
```

上图配置如下：

```
server {
    listen 80;
    server_name order-changgou-java.itheima.net;

    #静态资源
    location ~* \.(jpg|png|html|css|js|woff|ttf|woff2) {
        root /usr/local/server/web/index;
        index index.html;
    }

    location / {
      proxy_pass http://myip.com:8001;
    }
}
```

购物车效果如下：`<http://order-changgou-java.itheima.net/cart.html>`

2.2 添加购物车

修改商品详情页的addCart方法，让该方法点击的时候调用添加购物车，调用添加购物车方法需要添加axios,所以我们需要先引入axios。

引入axios

```
<script src="./js/axios.js"></script>
```

修改addCart方法

```
//加入购物车
addCart: function(){
    if(this.sku.id==0){
        alert('该商品已经下架，无法购买！');
        return;
    }

    //下单操作]
    axios.get('http://order-changgou-java.itheima.net/api/cart/add?num='+this.num+'&id='+this.sku.id,{withCredentials:true}).then(function(resp){
        if(resp.data.flag){
            //添加购物车成功！购物车列表查询地址：
            location.href='http://order-changgou-java.itheima.net/cart.html';
        }
    }).catch(function (error) {
        if(error.response!=null){
            if(error.response.status==401){
                //登录
                location.href='http://login-changgou-java.itheima.net/login.html?FROM='+window.location.href;
            }
        }
    });
},
```

此时由于跨域了，并且需要将Cookie提交到后台，这时候需要在微服务网关中配置跨域，在微服务网关的启动类中添加如下代码：

```
/**
 * 配置跨域
 * @return
 */
@Bean
public CorsWebFilter corsFilter() {
    CorsConfiguration config = new CorsConfiguration();
    // cookie跨域
    config.setAllowCredentials(Boolean.TRUE);
    config.addAllowedMethod("*");
    config.addAllowedOrigin("*");
    config.addAllowedHeader("*");

    //跨域解析器
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource(new PathPatternParser());
    source.registerCorsConfiguration("/**", config); //所有请求路径都支持跨域
    return new CorsWebFilter(source);
}
```

3 订单结算页

3.1 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。

3.2 实现用户收件地址查询

3.2.1 代码实现

(1)业务层

业务层接口

修改`changgou-service-user`微服务，需改`com.changgou.user.service.AddressService`接口，添加根据用户名字查询用户收件地址信息，代码如下：

```
/***
 * 收件地址查询
 * @param username
 * @return
 */
List<Address> list(String username);
```

业务层接口实现类

修改`changgou-service-user`微服务，修改`com.changgou.user.service.impl.AddressServiceImpl`类，添加根据用户查询用户收件地址信息实现方法，如下代码：

```
/***
 * 收件地址查询
 * @param username
 * @return
 */
@Override
public List<Address> list(String username) {
    Address address = new Address();
    address.setUsername(username);
    return addressMapper.select(address);
}
```

(2)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();
    String username = userMap.get("username");
    //查询用户收件地址
    List<Address> addressList = addressService.list(username);
    return new Result(true, StatusCode.OK,"查询成功！",addressList);
}
```

3.2.2 测试

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

3.2.3 域名配置

修改虚拟机中的nginx.conf，添加用户域名`user-changgou-java.itheima.net`配置，用于访问收件人地址列表数据。

```
server {
   listen 80;
   server_name user-changgou-java.itheima.net;

   location / {
     proxy_pass http://myip.com:8001;
    }
}
```

上图代码如下：

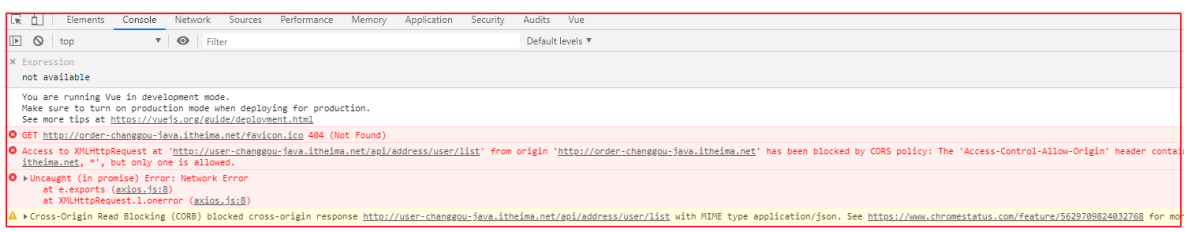
```
server {
   listen 80;
   server_name user-changgou-java.itheima.net;

   location / {
     proxy_pass http://myip.com:8001;
    }
}
```

本地添加`user-changgou-java.itheima.net`的映解析。

```
192.168.211.132 user-changgou-java.itheima.net
```

将页面order.html上传到虚拟机的`/usr/local/server/web/index`中。

访问`<http://order-changgou-java.itheima.net/order.html>`的时候，存在跨域问题，问题如下：



出现上述问题的原因是我们在微服务网关配置了跨域，在`changgou-service-user`微服务网关中也配置了跨域，多出配置了跨域会导致跨域失效，我们只需要配置微服务网关就可以了，需要将`AddressController`中的跨域注解`@CrossOrigin`注释掉。

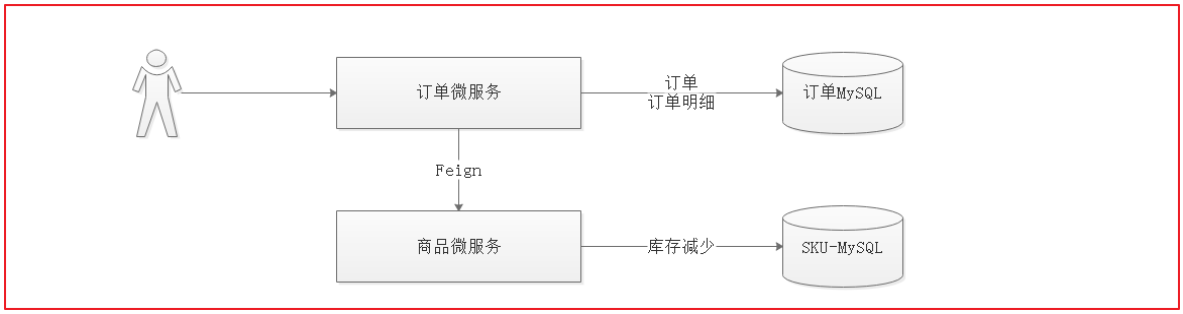
3.2.4 运送清单

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。

4 下单

4.1 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,0，未删除，1表示删除',
  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(60) COLLATE utf8_bin DEFAULT NULL COMMENT 'SPU_ID',
  `sku_id` varchar(60) NOT NULL COMMENT 'SKU_ID',
  `order_id` varchar(60) 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;
```

4.2 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。

4.2.1 代码实现

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 idWorker(){
    return new IdWorker(1,1);
}
```

(1)业务层

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

修改订单微服务添加`com.changgou.order.service.impl.OrderServiceImpl`,代码如下：

```
@Service
public class OrderServiceImpl implements OrderService {

    @Autowired
    private OrderMapper orderMapper;

    @Autowired
    private OrderItemMapper orderItemMapper;

    @Autowired
    private CartService cartService;

    @Autowired
    private IdWorker idWorker;

    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 增加Order
     * 一个订单有多个明细
     *      2张表
     *          1.tb_order表（订单表）
     *          2.tb_order_item表(订单明细表)
     *
```

```
     *       当前用户勾选的商品清单列表
     * @param order
     */
    @Override
    public void add(Order order){
        //1.完善Order
        //id
        order.setId("No"+idWorker.nextId());

        //查询购物车集合
        List<OrderItem> items = redisTemplate.boundHashOps("Cart_" +
order.getUsername()).values();
        int num=0;
        int totalMoney = 0;
        //记录当前商品操作数量信息
        Map<String,Integer> dataMap = new HashMap<String,Integer>();

        //筛选当前勾选的商品
        for (OrderItem orderItem : items) {
            //循环ids，进行匹配
            for (String id : ids) {
                if(orderItem.getSkuId().equals(id)){
                    //2.完善每个OrderItem
                    //完善每个订单ID
                    orderItem.setOrderId(order.getId());
                    //orderItem的id
                    orderItem.setId("No"+idWorker.nextId());
                    //3.循环增加OrderItem
                    orderItemMapper.insertSelective(orderItem);

                    //5.清空勾选的购物车数据
                    redisTemplate.boundHashOps("Cart_" +
order.getUsername()).delete(id);

                    //统计该商品的个数
                    num+=orderItem.getNum();
                    totalMoney+=orderItem.getMoney();

                    //记录操作的数据
                    dataMap.put(id,orderItem.getNum());
                    break;
                }
            }
        }
        //total_num
        order.setTotalNum(num);
        //total_money
        order.setTotalMoney(totalMoney);
        order.setCreateTime(new Date());
        order.setUpdateTime(order.getCreateTime());
        order.setOrderStatus("0");  //0 未完成
        order.setPayStatus("0");    //0 未支付
        order.setConsignStatus("0");//0 未发货
        order.setIsDelete("0");     //0 未删除

        //4.增加Order
        orderMapper.insertSelective(order);
    }
```

```
82  }
```

(2)控制层

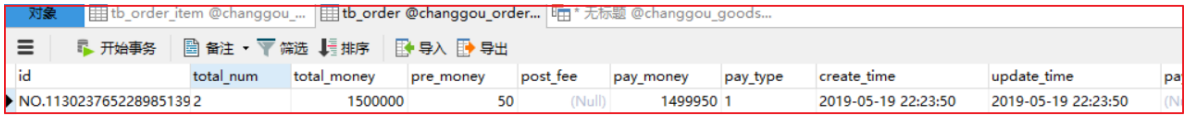
修改 changgou-service-order 微服务，修改 com.changgou.order.controller.OrderController 类，代码如下：

```
/***
 * 新增Order数据
 * @param order
 * @return
 */
@PostMapping
public Result add(@RequestBody Order order){
    //获取用户名
    Map<String, String> userMap = TokenDecode.getUserInfo();
    String username = userMap.get("username");
    //设置购买用户
    order.setUsername(username);
    orderService.add(order);
    return new Result(true,StatusCode.OK,"添加成功");
}
```

4.2.2 测试

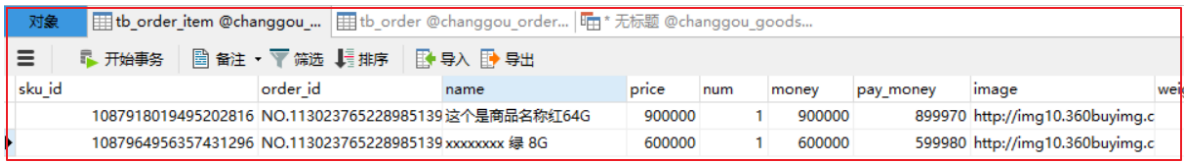
保存订单测试，表数据变化如下：

tb_order表数据：



id	total_num	total_money	pre_money	post_fee	pay_money	pay_type	create_time	update_time
NO.1130237652289851392	2	1500000	50	(Null)	1499950	1	2019-05-19 22:23:50	2019-05-19 22:23:50

tb_order_item表数据：

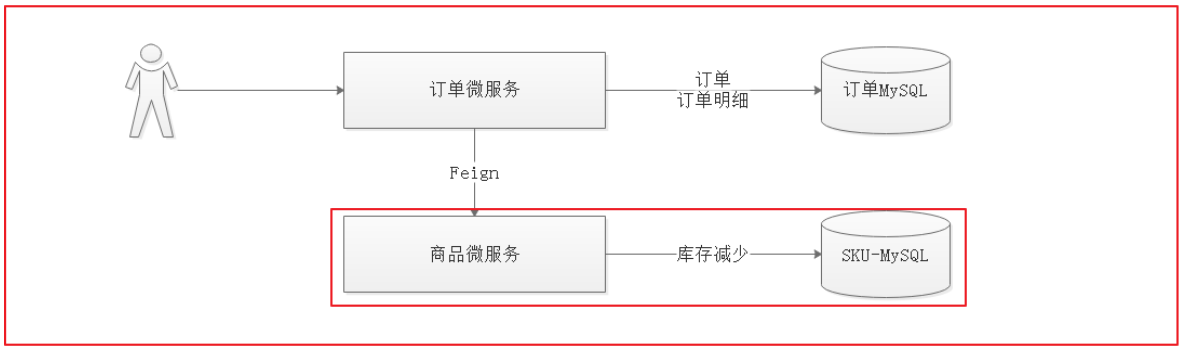


sku_id	order_id	name	price	num	money	pay_money	image
1087918019495202816	NO.1130237652289851392	这个是商品名称红64G	900000	1	900000	899970	http://img10.360buyimg.c
1087964956357431296	NO.1130237652289851392	xxxxxxxx 绿 8G	600000	1	600000	599980	http://img10.360buyimg.c

4.3 库存变更

4.3.1 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。

该工程中一会儿需要查询购物车数据，所以需要引入订单的api，在pom.xml中添加如下依赖：

```
<!--order api 依赖-->
<dependency>
    <groupId>com.changgou</groupId>
    <artifactId>changgou-service-order-api</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

4.3.2 代码实现

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，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();
}
```

(1)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 orderItem
 * @return
 */
@Update("UPDATE tb_sku SET num=num-#{num},sale_num=sale_num+#{num} WHERE id=#{skuId} AND num>=#{num}")
int decrCount(OrderItem orderItem);
```

(2)业务层

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

```
/****
 * 商品库存减少   update table set num=num-? where id=?
 * Map<key,value>  key=id
 *                 value=num
 *
 *                 NO001 = 5
 *                 NO002 = 3
 */
void decrCount(Map<String,Integer> decrData);
```

修改changgou-service-order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;

/****
 * 商品库存减少   update table set num=num-? where id=?
 * Map<key,value>  key=id
 *                 value=num
 *
 *                 NO001 = 5
 *                 NO002 = 3
 */
@Override
public void decrCount(Map<String, Integer> decrData) {
    for (Map.Entry<String, Integer> entry : decrData.entrySet()) {
        //商品ID
        String id = entry.getKey();
        //递减数量
        Object num = entry.getValue();

        //执行递减,使用SQL语句，利用MySQL的行级锁防止超卖现象
        int count =
skuMapper.decrCount(id,Integer.parseInt(num.toString()));
        if(count<=0){
            throw new RuntimeException("库存数量不足，下单失败！");
        }
    }
}
```

(3)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：

```java
/****
 * 库存递减实现
 */
@GetMapping(value = "/decr/count")
public Result decrCount(@RequestParam Map<String,Integer> map){
    //执行递减
    skuService.decrCount(map);
    return new Result(true,StatusCode.OK,"库存递减成功！");
}
```

(4)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：

```java
/***
 * 库存递减实现
 * @param map
 * @return
 */
@GetMapping(value = "/decr/count")
Result decrCount(@RequestParam Map<String,Integer> map);
```

4.3.3 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。

先注入SkuFeign

```java
@Autowired
private SkuFeign skuFeign;
```

再调用库存递减方法

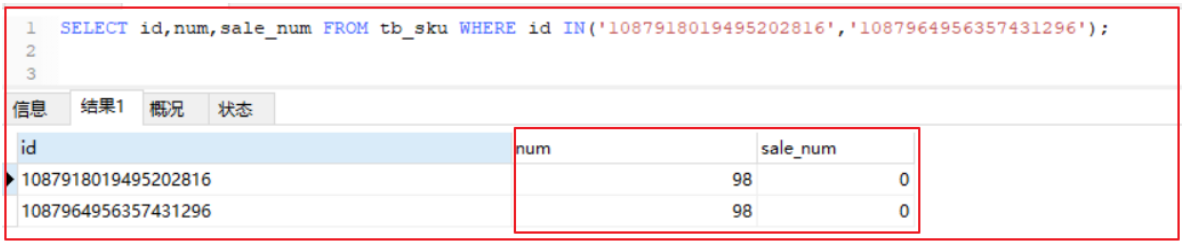
```java
//库存减库存
skuFeign.decrCount(order.getUsername());
```

完整代码如下：

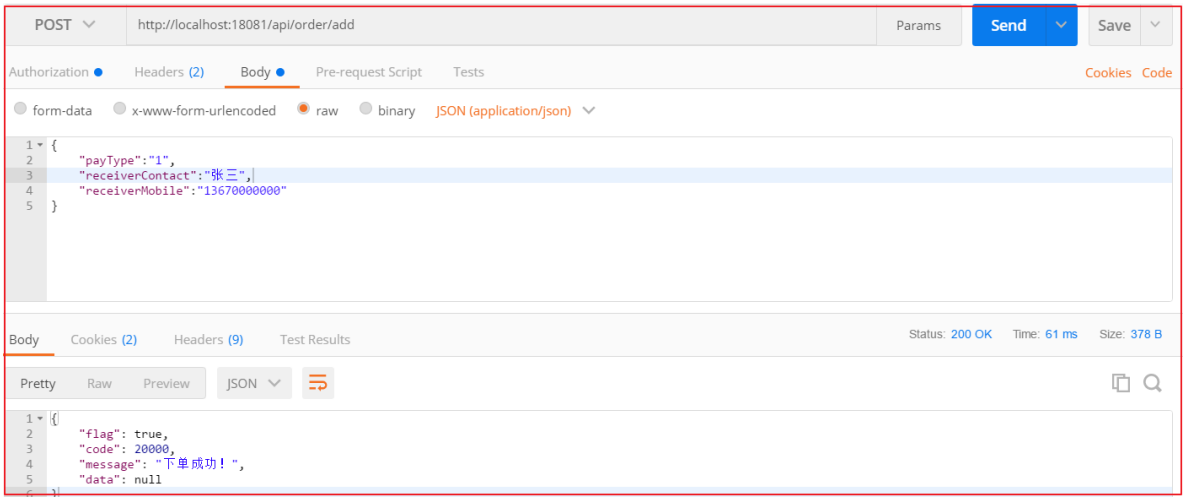
```java
@Override
public void add(Order order,List<String> ids){
    //1.完善Order
    //id
    order.setId("No"+idWorker.nextId());

    //查询购物车集合
    List<OrderItem> items = redisTemplate.boundHashOps( key: "Cart_" + order.getUsername()).values();
    int num=0;
    int totalMoney = 0;
    //记录当前商品操作数量信息
    Map<String,Integer> dataMap = new HashMap<String,Integer>();

    //筛选当前勾选的商品
    for (OrderItem orderItem : items) {
        //循环ids，进行匹配
        for (String id : ids) {
            if(orderItem.getSkuId().equals(id)){
                //2.完善每个OrderItem
                //完善每个订单ID
                orderItem.setOrderId(order.getId());
                //orderItem的id
                orderItem.setId("No"+idWorker.nextId());
                //3.循环增加OrderItem
                orderItemMapper.insertSelective(orderItem);

                //5.清空勾选的购物车数据
                redisTemplate.boundHashOps( key: "Cart_" + order.getUsername()).delete(id);

                //统计该商品的个数
                num+=orderItem.getNum();
                totalMoney+=orderItem.getMoney();

                //记录操作的数据
                dataMap.put(id,orderItem.getNum());
                break;
            }
        }
    }
    //total_num
    order.setTotalNum(num);
    //total_money
    order.setTotalMoney(totalMoney);
    order.setCreateTime(new Date());
    order.setUpdateTime(order.getCreateTime());
    order.setOrderStatus("0");  //0 未完成
    order.setPayStatus("0");    //0 未支付
    order.setConsignStatus("0");//0 未发货
    order.setIsDelete("0");     //0 未删除

    //4.增加Order
    orderMapper.insertSelective(order);

    //5.库存递减
    skuFeign.decrCount(dataMap);
}
```

4.3.4 测试

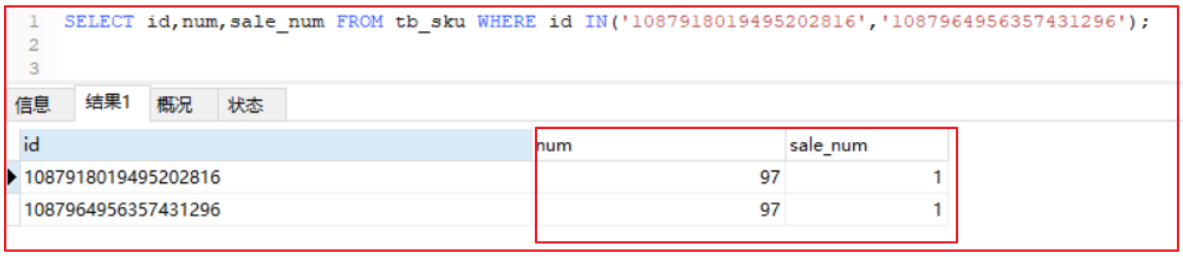
库存减少前，查询数据库Sku数据如下：个数98，销量0



使用Postman执行 http://localhost:18081/api/order/add



执行测试后，剩余库存97，销量1



4.4 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';
```

4.4.1 代码实现

(1)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);
```

(2)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);
}
```

(3)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：

```
@Autowired
private TokenDecode tokenDecode;

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

(4)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：

```
/***
 * 添加用户积分
 * @param points
 * @return
 */
@GetMapping(value = "/points/add")
Result addPoints(@RequestParam(value = "points")Integer points);
```

4.4.2 增加积分调用

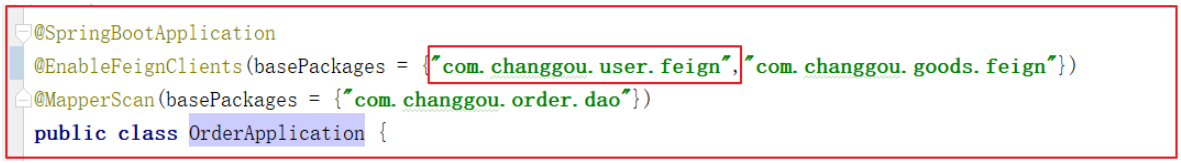
修改changgou-service-order，添加changgou-service-user-api的依赖，修改pom.xml,添加如下依赖：

```
<!--user api 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
 */
@Override
public void add(Order order){
    //查询出用户的所有购物车
    //..略

    //增加用户积分
    userFeign.addPoints(10);

    //清除Redis缓存购物车数据
    redisTemplate.delete( key: "Cart_"+order.getUsername());
}
```

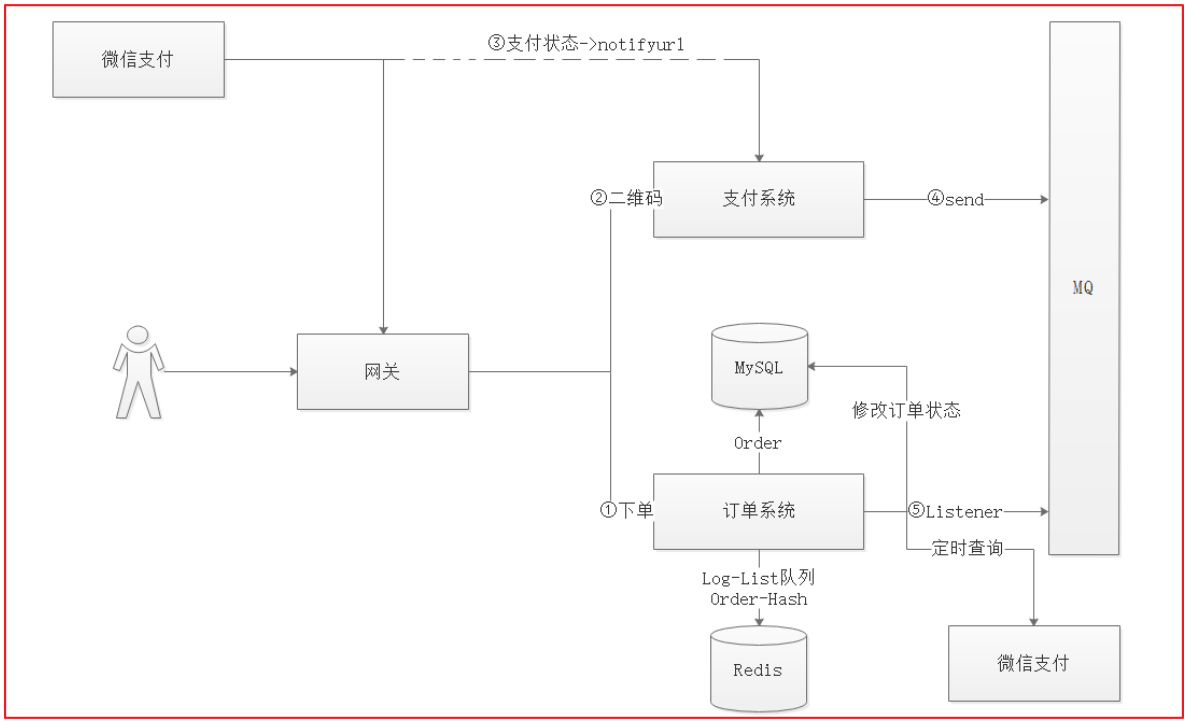
修改changgou-service-order的启动类`com.changgou.OrderApplication`，添加feign的包路径：



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

5 支付流程分析(预习)

5.1 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。
```

5.2 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qrious制作二维码插件。

qrious是一款基于HTML5 Canvas的纯JS二维码生成插件。通过qrious.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，还可以将生成的二维码进行Base64编码。

qrious.js二维码插件的可用配置参数如下：

参数	类型	默认值	描述
background	String	"white"	二维码的背景颜色。
foreground	String	"black"	二维码的前景颜色。
level	String	"L"	二维码的误差校正级别(L, M, Q, H)。
mime	String	"image/png"	二维码输出为图片时的MIME类型。
size	Number	100	二维码的尺寸，单位像素。
value	String	""	需要编码为二维码的值

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>
</head>
<body>
<img id="qrious">
<script src="qrious.js"></script>
<script>
 var qr = new QRious({
        element:document.getElementById('qrious'),
        size:250,
        level:'H',
        value:'http://www.itheima.com'
    });
</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？

6 微信扫码支付简介(预习)

6.1 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| 企业/公司| 政府| 媒体| 其他类型。

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。

6.2 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

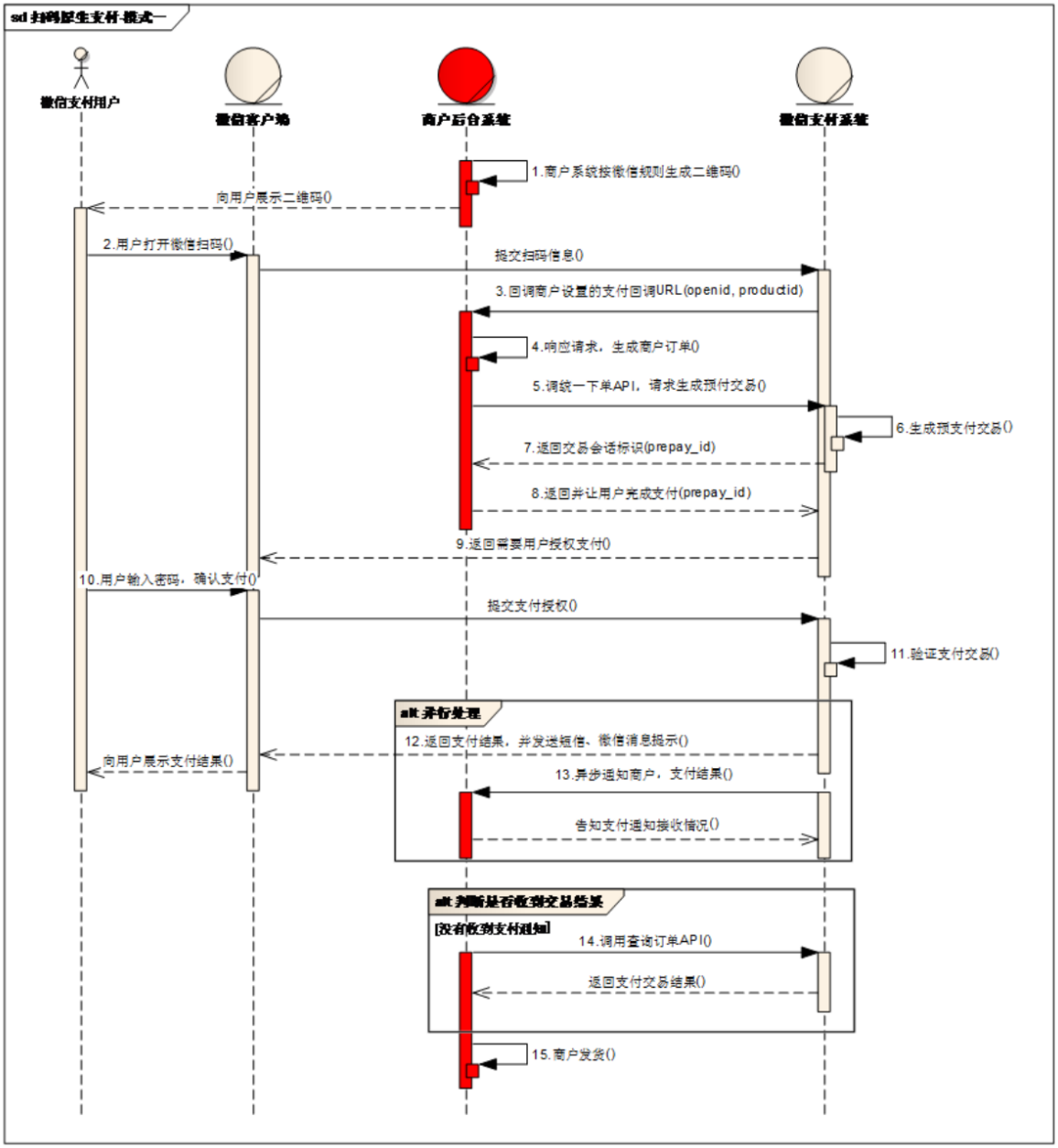
如果你不能联网，请查阅讲义配套资源 （资源\配套软件\微信扫码支付\开发文档）

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
```

6.3 微信支付模式介绍

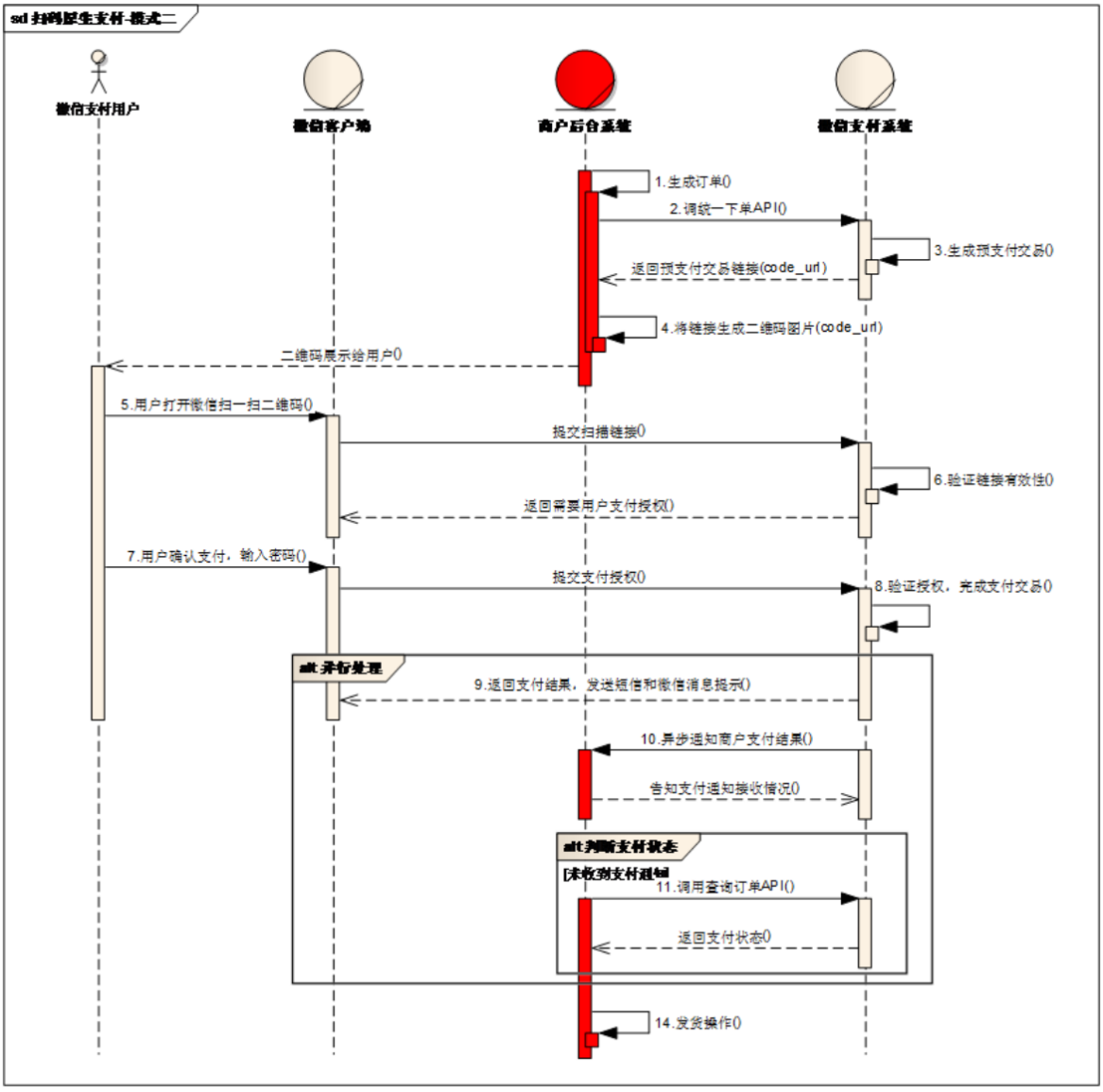
6.3.1 模式一



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。

6.3.2 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。