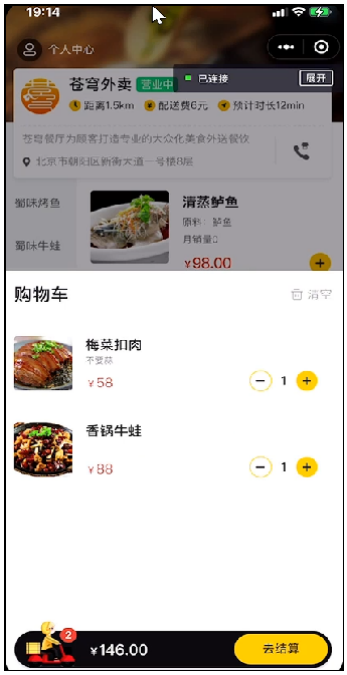
今日内容

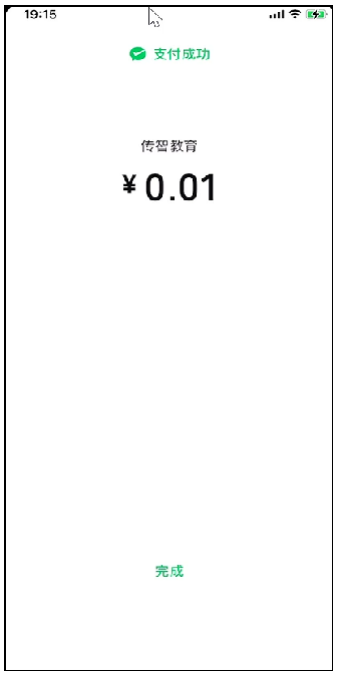
- 导入地址簿功能代码
- 用户下单
- 订单支付

功能实现：**用户下单、订单支付**

用户下单效果图：



订单支付效果图：



一、地址簿相关功能

1、需求分析和设计

1.1、产品原型

地址簿，指的是消费者用户的地址信息，用户登录成功后可以维护自己的地址信息。同一个用户可以有多个地址信息，但是只能有一个**默认地址**。

效果图：

对于地址簿管理，我们需要实现以下几个功能：

- 查询地址列表
- 新增地址
- 修改地址
- 删除地址
- 设置默认地址
- 查询默认地址

1.2、接口设计

根据上述原型图先**粗粒度**设计接口，共包含 7 个接口。

接口设计：

- 新增地址
- 查询登录用户所有地址

- 查询默认地址
- 根据 id 修改地址
- 根据 id 删除地址
- 根据 id 查询地址
- 设置默认地址

接下来**细粒度**分析每个接口，明确每个接口的请求方式、请求路径、传入参数和返回值。

1). 新增地址

基本信息

Path： /user/addressBook

Method： POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注
cityCode	string	非必须		
cityName	string	非必须		
consignee	string	非必须		
detail	string	必须		详细地址
districtCode	string	非必须		
districtName	string	非必须		
id	integer	非必须		
isDefault	integer	非必须		
label	string	非必须		
phone	string	必须		手机号
provinceCode	string	非必须		
provinceName	string	非必须		
sex	string	必须		
userId	integer	非必须		

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

2). 查询登录用户所有地址

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
├─ id	number	必须			
├─ userId	number	必须			
├─ consignee	string	必须			
├─ phone	string	必须			
├─ sex	string	必须			
├─ provinceCode	string	必须			
├─ provinceName	string	必须			
├─ cityCode	string	必须			
├─ cityName	string	必须			
├─ districtCode	string	必须			
├─ districtName	string	必须			
├─ detail	string	必须			
├─ label	string	必须			
├─ isDefault	number	必须			
msg	string	非必须			

基本信息

Path： /user/addressBook/list

Method： GET

接口描述：

请求参数

3). 查询默认地址

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
├─ cityCode	string	非必须			
├─ cityName	string	非必须			
├─ consignee	string	非必须			
├─ detail	string	非必须			
├─ districtCode	string	非必须			
├─ districtName	string	非必须			
├─ id	integer	非必须			format: int64
├─ isDefault	integer	非必须			format: int32
├─ label	string	非必须			
├─ phone	string	非必须			
├─ provinceCode	string	非必须			
├─ provinceName	string	非必须			
├─ sex	string	非必须			
├─ userId	integer	非必须			format: int64
msg	string	非必须			

基本信息

Path： /user/addressBook/default

Method： GET

接口描述：

请求参数

4). 修改地址

Body

名称	类型	是否必须	默认值	备注	其他信息
cityCode	string	非必须			
cityName	string	非必须			
consignee	string	非必须			
detail	string	必须		详细地址	
districtCode	string	非必须			
districtName	string	非必须			
id	integer	必须		主键值	format: int64
isDefault	integer	非必须			format: int32
label	string	非必须			
phone	string	必须		手机号	
provinceCode	string	非必须			
provinceName	string	非必须			
sex	string	必须			
userId	integer	非必须			format: int64

基本信息

Path： /user/addressBook

Method： PUT

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

5). 根据 id 删除地址

基本信息

Path： /user/addressBook

Method： DELETE

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
id	是	101	地址id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

6). 根据 id 查询地址

基本信息

Path： /user/addressBook/{id}

Method： GET

接口描述：

请求参数

路径参数

参数名称	示例	备注
id	101	地址id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
data	object	必须			
├─ id	number	非必须			
├─ phone	string	非必须			
├─ consignee	string	非必须			
├─ userId	number	非必须			
├─ cityCode	string	非必须			
├─ provinceName	string	非必须			
├─ provinceCode	string	非必须			
├─ sex	string	非必须			
├─ districtName	string	非必须			
├─ districtCode	string	非必须			
├─ cityName	string	非必须			
├─ isDefault	number	非必须			
├─ label	string	非必须			
├─ detail	string	非必须			
msg	string	非必须			

7). 设置默认地址

基本信息

Path： /user/addressBook/default

Method： PUT

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	必须		地址id	format: int64

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

1.3、表设计

用户的地址信息会存储在 address_book 表，即地址簿表中。具体表结构如下：

字段名	数据类型	说明	备注
id	bigint	主键	自增
user_id	bigint	用户 id	逻辑外键
consignee	varchar(50)	收货人	
sex	varchar(2)	性别	
phone	varchar(11)	手机号	
province_code	varchar(12)	省份编码	
province_name	varchar(32)	省份名称	
city_code	varchar(12)	城市编码	
city_name	varchar(32)	城市名称	
district_code	varchar(12)	区县编码	
district_name	varchar(32)	区县名称	
detail	varchar(200)	详细地址信息	具体到门牌号
label	varchar(100)	标签	公司、家、学校
is_default	tinyint(1)	是否默认地址	1：是，0：否

这里面有一个字段 is_default，实际上我们在设置默认地址时，只需要更新这个字段就可以了。

2、代码开发

进入到 sky-server 模块中

2.1、Controller 层

```
package com.sky.controller.user;

import com.sky.context.BaseContext;
import com.sky.entity.AddressBook;
import com.sky.result.Result;
import com.sky.service.AddressBookService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/user/addressBook")
```

```java
@Api(tags = "C端地址簿接口")
@Slf4j
public class AddressBookController {
    @Autowired
    private AddressBookService addressBookService;

    /**
     * 查询当前登录用户的所有地址信息
     * @return
     */
    @GetMapping("/list")
    @ApiOperation("查询当前登录用户的所有地址信息")
    public Result<List<AddressBook>> list() {
        Long userId = BaseContext.getCurrentId();
        log.info("查询当前登录用户：{}的所有地址信息",userId);
        AddressBook addressBook = new AddressBook();
        addressBook.setUserId(userId);
        List<AddressBook> list = addressBookService.list(addressBook);
        return Result.success(list);
    }

    /**
     * 新增地址
     * @param addressBook
     * @return
     */
    @PostMapping
    @ApiOperation("新增地址")
    public Result save(@RequestBody AddressBook addressBook) {
        log.info("新增地址：{}",addressBook);
        addressBookService.save(addressBook);
        return Result.success();
    }

    /**
     * 根据id查询地址
     * @param addressBook
     * @return
     */
    @GetMapping("/{id}")
    @ApiOperation("根据id查询地址")
    public Result<AddressBook> getById(@PathVariable Long id) {
        log.info("根据id查询地址：{}",id);
        AddressBook addressBook = addressBookService.getById(id);
        return Result.success(addressBook);
    }

    /**
     * 根据id修改地址
     * @param addressBook
     * @return
     */
```

```
    @PutMapping
    @ApiOperation("根据id修改地址")
    public Result update(@RequestBody AddressBook addressBook) {
        log.info("根据id修改地址：{}",addressBook);
        addressBookService.update(addressBook);
        return Result.success();
    }

    /**
     * 设置默认地址
     * @param addressBook
     * @return
     */
    @PutMapping("/default")
    @ApiOperation("设置默认地址")
    public Result setDefault(@RequestBody AddressBook addressBook) {
        log.info("设置默认地址：{}",addressBook);
        addressBookService.setDefault(addressBook);
        return Result.success();
    }

    /**
     * 根据id删除地址
     * @param id
     * @return
     */
    @DeleteMapping
    @ApiOperation("根据id删除地址")
    public Result deleteById(Long id) {
        log.info("根据id删除地址：{}",id);
        addressBookService.deleteById(id);
        return Result.success();
    }

    /**
     * 查询默认地址
     */
    @GetMapping("default")
    @ApiOperation("查询默认地址")
    public Result<AddressBook> getDefault() {
        //SQL:select * from address_book where user_id = ? and is_default = 1
        AddressBook addressBook = new AddressBook();
        addressBook.setIsDefault(1);
        addressBook.setUserId(BaseContext.getCurrentId());
        List<AddressBook> list = addressBookService.list(addressBook);

        if (list != null && list.size() == 1) {
            return Result.success(list.get(0));
        }

        return Result.error("没有查询到默认地址");
    }
```

```
119     }
```

2.2、Service 层

创建 AddressBookService.java

```
1   package com.sky.service;
2
3   import com.sky.entity.AddressBook;
4   import java.util.List;
5
6   public interface AddressBookService {
7       List<AddressBook> list(AddressBook addressBook);
8
9       void save(AddressBook addressBook);
10
11      AddressBook getById(Long id);
12
13      void update(AddressBook addressBook);
14
15      void setDefault(AddressBook addressBook);
16
17      void deleteById(Long id);
18  }
```

创建 AddressBookServiceImpl.java

```
1   package com.sky.service.impl;
2
3   import com.sky.context.BaseContext;
4   import com.sky.entity.AddressBook;
5   import com.sky.mapper.AddressBookMapper;
6   import com.sky.service.AddressBookService;
7   import lombok.extern.slf4j.Slf4j;
8   import org.springframework.beans.factory.annotation.Autowired;
9   import org.springframework.stereotype.Service;
10  import org.springframework.transaction.annotation.Transactional;
11  import java.util.List;
12
13  @Service
14  @Slf4j
15  public class AddressBookServiceImpl implements AddressBookService {
16      @Autowired
17      private AddressBookMapper addressBookMapper;
18
19      /**
20       * 条件查询
21       * @param addressBook
22       * @return
23       */
24      public List<AddressBook> list(AddressBook addressBook) {
25          return addressBookMapper.list(addressBook);
```

```
    }

    /**
     * 新增地址
     * @param addressBook
     */
    public void save(AddressBook addressBook) {
        addressBook.setUserId(BaseContext.getCurrentId());
        addressBook.setIsDefault(0);
        addressBookMapper.insert(addressBook);
    }

    /**
     * 根据id查询地址
     * @param id
     * @return
     */
    public AddressBook getById(Long id) {
        AddressBook addressBook = addressBookMapper.getById(id);
        return addressBook;
    }

    /**
     * 根据id修改地址
     * @param addressBook
     */
    public void update(AddressBook addressBook) {
        addressBookMapper.update(addressBook);
    }

    /**
     * 设置默认地址
     * @param addressBook
     */
    @Transactional
    public void setDefault(AddressBook addressBook) {
        //1、将当前用户的所有地址修改为非默认地址 update address_book set is_default = ? where user_id = ?
        addressBook.setIsDefault(0);
        addressBook.setUserId(BaseContext.getCurrentId());
        addressBookMapper.updateIsDefaultByUserId(addressBook);

        //2、将当前地址改为默认地址 update address_book set is_default = ? where id = ?
        addressBook.setIsDefault(1);
        addressBookMapper.update(addressBook);
    }

    /**
     * 根据id删除地址
     * @param id
     */
    public void deleteById(Long id) {
```

```
            addressBookMapper.deleteById(id);
        }
    }
```

2.3、Mapper 层

创建 AddressBookMapper.java

```java
package com.sky.mapper;

import com.sky.entity.AddressBook;
import org.apache.ibatis.annotations.*;
import java.util.List;

@Mapper
public interface AddressBookMapper {
    /**
     * 条件查询
     * @param addressBook
     * @return
     */
    List<AddressBook> list(AddressBook addressBook);

    /**
     * 新增地址
     * @param addressBook
     */
    @Insert("insert into address_book " +
            "(user_id,consignee,phone,sex,province_code,province_name,city_code,city_name," +
            "district_code,district_name,detail,label,is_default)" +
            "values (#{userId},#{consignee},#{phone},#{sex},#{provinceCode},#{provinceName}," +
            "#{cityCode},#{cityName},#{districtCode},#{districtName},#{detail},#{label},#{isDefault})")
    void insert(AddressBook addressBook);

    /**
     * 根据id查询地址
     * @param id
     * @return
     */
    @Select("select * from address_book where id = #{id}")
    AddressBook getById(Long id);

    /**
     * 根据id修改地址
     * @param addressBook
     */
    void update(AddressBook addressBook);

    /**
```

```
     * 根据用户id修改是否默认地址
     * @param addressBook
     */
    @Update("update address_book set is_default = #{isDefault} where user_id = #{userId}")
    void updateIsDefaultByUserId(AddressBook addressBook);

    /**
     * 根据id删除地址
     * @param id
     */
    @Delete("delete from address_book where id = #{id}")
    void deleteById(Long id);
}
```

创建 AddressBookMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace="com.sky.mapper.AddressBookMapper">
    <select id="list" parameterType="AddressBook" resultType="AddressBook">
        select * from address_book
        <where>
            <if test="userId != null">
                and user_id = #{userId}
            </if>
            <if test="phone != null">
                and phone = #{phone}
            </if>
            <if test="isDefault != null">
                and is_default = #{isDefault}
            </if>
        </where>
    </select>

    <update id="update" parameterType="addressBook">
        update address_book
        <set>
            <if test="consignee != null">
                consignee = #{consignee},
            </if>
            <if test="sex != null">
                sex = #{sex},
            </if>
            <if test="phone != null">
                phone = #{phone},
            </if>
            <if test="detail != null">
                detail = #{detail},
            </if>
            <if test="label != null">
```

```
                label = #{label},
            </if>
            <if test="isDefault != null">
                is_default = #{isDefault},
            </if>
        </set>
        where id = #{id}
    </update>
</mapper>
```

3、功能测试

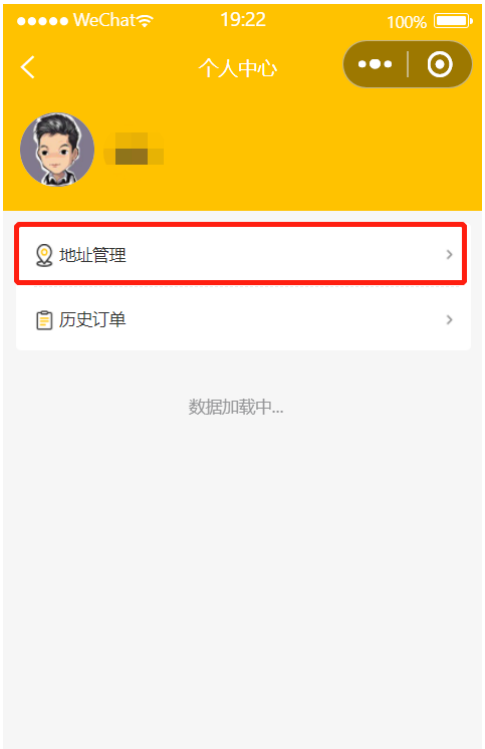
可以通过如下方式进行测试：

- 查看控制台 sql 和数据库中的数据变化
- Swagger 接口文档测试
- 前后端联调

我们直接使用**前后端联调**测试：

启动后台服务，编译小程序

登录进入首页 —> 进入个人中心 —> 进入地址管理



1). 新增收货地址

添加两条收货地址：

查看收货地址：

查看数据库：

consignee	sex	phone	province_code	province_name	city_code	city_name	district_code	district_name	detail	label	is_default
张三	0	13812345678	11	北京市	1101	市辖区	110102	西城区	xx单元xx层xx号	1	0
李四	0	13812345670	11	北京市	1101	市辖区	110105	朝阳区	xx单元xx层xx号	2	0

2). 设置默认收货地址

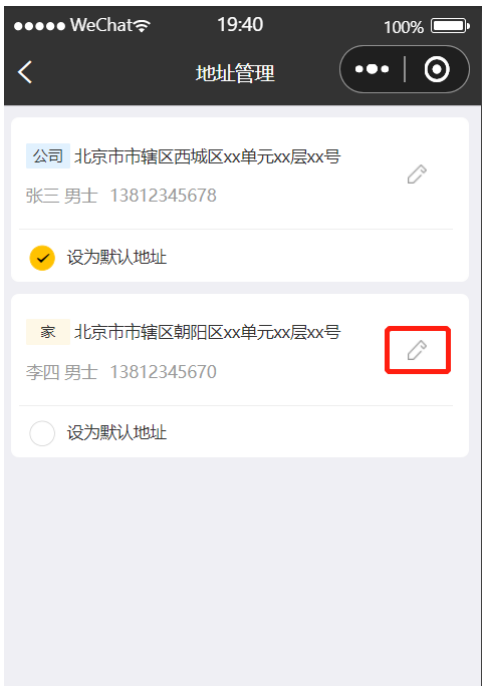
设置默认地址：

查看数据库：

consignee	sex	phone	province_code	province_name	city_code	city_name	district_code	district_name	detail	label	is_default
张三	0	13812345678	11	北京市	1101	市辖区	110102	西城区	xx单元xx层xx号	1	1
李四	0	13812345670	11	北京市	1101	市辖区	110105	朝阳区	xx单元xx层xx号	2	0

3). 删除收货地址

进行编辑：

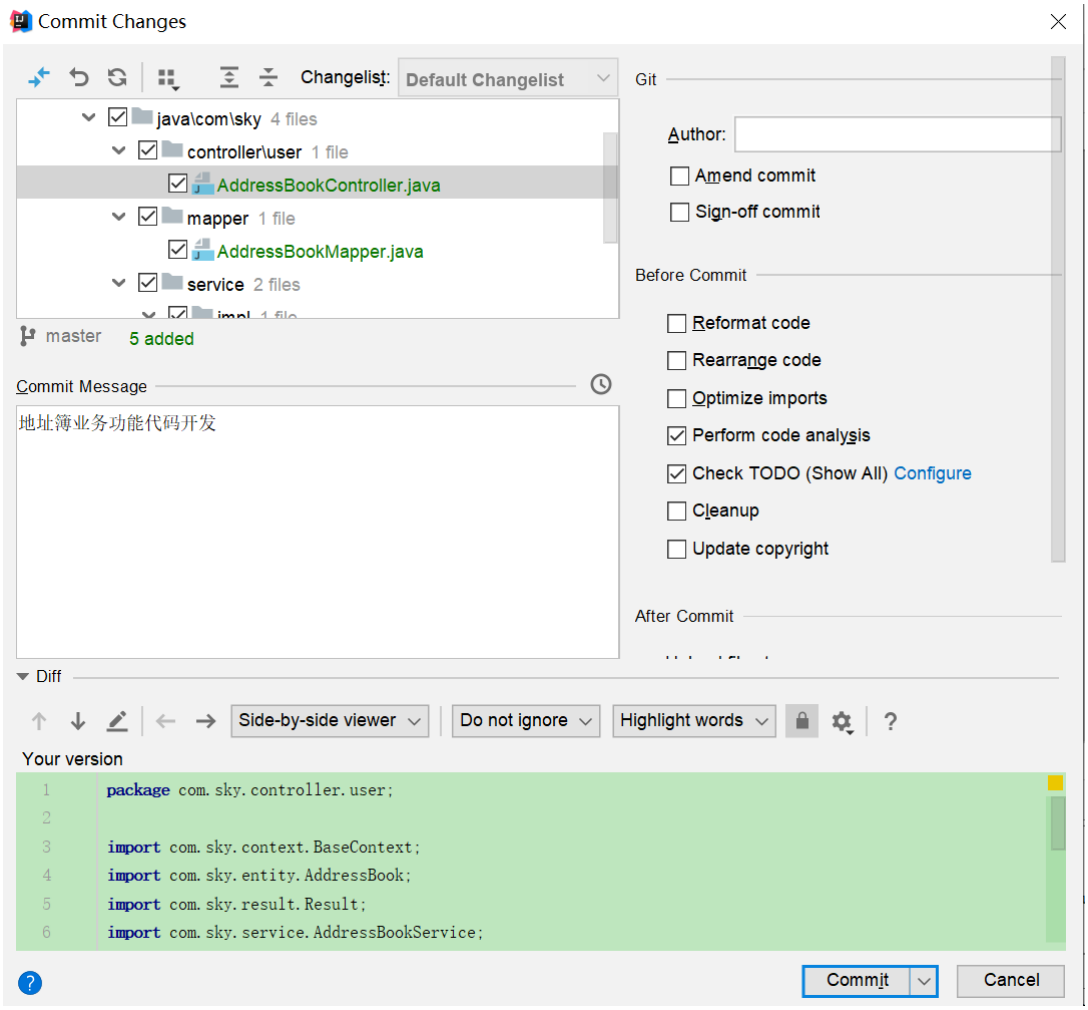


删除地址：

查看数据库：

consignee	sex	phone	province_code	province_name	city_code	city_name	district_code	district_name	detail	label	is_default
张三	0	13812345678	11	北京市	1101	市辖区	110102	西城区	xx单元xx层xx号	1	1

4、代码提交



后续步骤和其它功能代码提交一致，不再赘述。

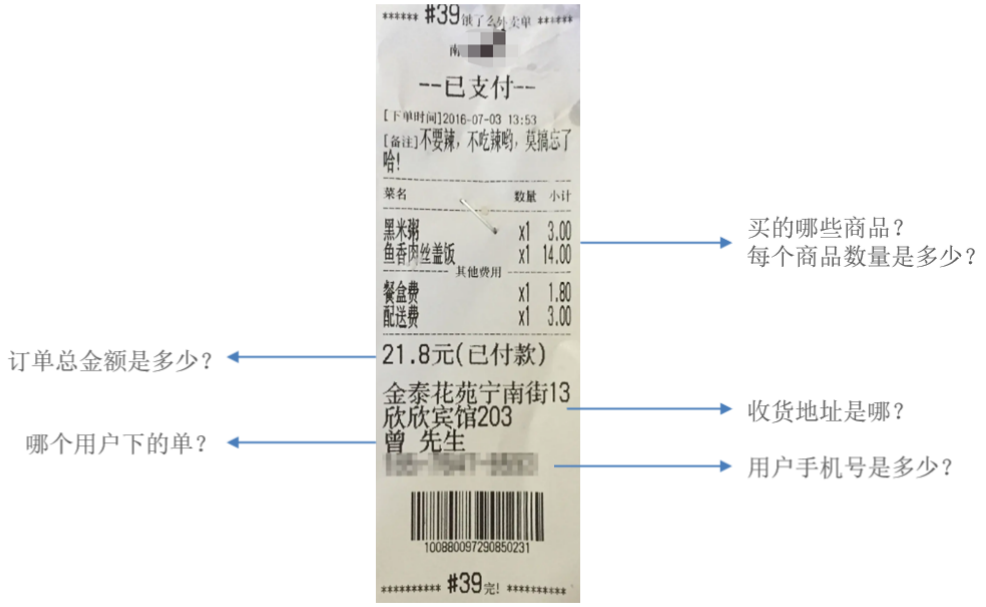
二、用户下单

1、需求分析和设计

1.1、产品原型

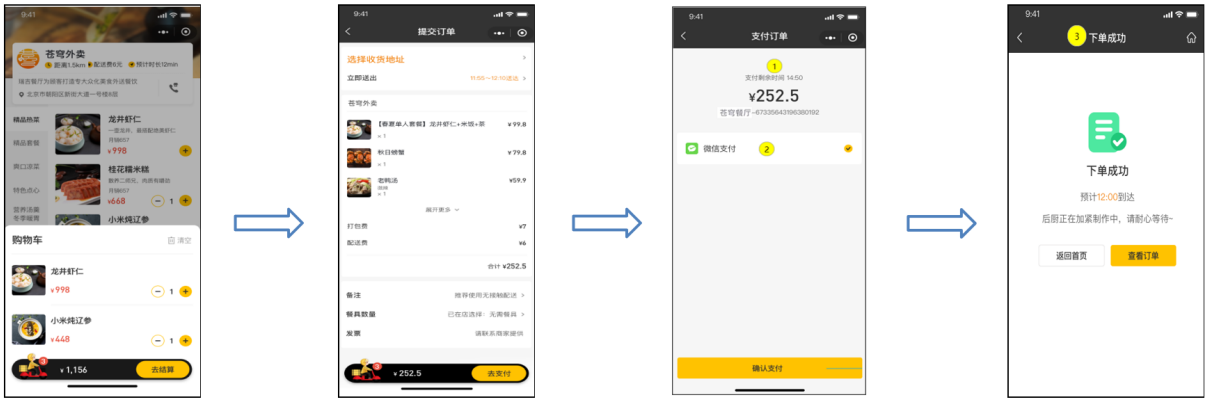
用户下单业务说明：

在电商系统中，用户是通过下单的方式通知商家，用户已经购买了商品，需要商家进行备货和发货。用户下单后会产生订单相关数据，订单数据需要能够体现如下信息：



用户将菜品或者套餐加入购物车后，可以点击购物车中的 "去结算" 按钮，页面跳转到订单确认页面，点击 "去支付" 按钮则完成下单操作。

用户点餐业务流程（效果图）：



购物车页面　　订单提交页面　　订单支付页面　　下单成功页面

1.2、接口设计

接口分析：

请求方式：POST

请求路径：/user/order/submit

参数：

- 地址簿id
- 配送状态（立即送出、选择送出时间）
- 打包费
- 总金额
- 备注
- 餐具数量

返回数据：

- 下单时间
- 订单总金额
- 订单号
- 订单id

接口设计：

基本信息

Path： /user/order/submit

Method： POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注
addressBookId	integer	必须		地址簿id
amount	number	必须		总金额
deliveryStatus	integer	必须		配送状态： 1立即送出 0选择具体时间
estimatedDeliveryTime	string	必须		预计送达时间
packAmount	integer	必须		打包费
payMethod	integer	必须		付款方式
remark	string	必须		备注
tablewareNumber	integer	必须		餐具数量
tablewareStatus	integer	必须		餐具数量状态 1按餐量提供 0选择具体数量

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
├─id	integer	必须		订单id
├─orderAmount	number	必须		订单金额
├─orderNumber	string	必须		订单号
├─orderTime	string	必须		下单时间
msg	string	非必须		

1.3、表设计

用户下单业务对应的数据表为 orders 表和 order_detail 表（一对多关系，一个订单关联多个订单明细）：

表名	含义	说明
orders	订单表	主要存储订单的基本信息（如：订单号、状态、金额、支付方式、下单用户、收件地址等）
order_detail	订单明细表	主要存储订单详情信息（如：该订单关联的套餐及菜品的信息）

具体的表结构如下：

1). orders 订单表

字段名	数据类型	说明	备注
id	bigint	主键	自增
number	varchar(50)	订单号	
status	int	订单状态	1：待付款，2：待接单，3：已接单，4：派送中，5：已完成，6：已取消
user_id	bigint	用户 id	逻辑外键
address_book_id	bigint	地址 id	逻辑外键
order_time	datetime	下单时间	
checkout_time	datetime	付款时间	
pay_method	int	支付方式	1：微信支付，2：支付宝支付
pay_status	tinyint	支付状态	0：未支付，1：已支付，2：退款
amount	decimal(10,2)	订单金额	
remark	varchar(100)	备注信息	
phone	varchar(11)	手机号	冗余字段
address	varchar(255)	详细地址信息	冗余字段
consignee	varchar(32)	收货人	冗余字段
cancel_reason	varchar(255)	订单取消原因	
rejection_reason	varchar(255)	拒单原因	
cancel_time	datetime	订单取消时间	
estimated_delivery_time	datetime	预计送达时间	
delivery_status	tinyint	配送状态	1：立即送出，0：选择具体时间

字段名	数据类型	说明	备注
delivery_time	datetime	送达时间	
pack_amount	int	打包费	
tableware_number	int	餐具数量	
tableware_status	tinyint	餐具数量状态	1：按餐量提供，0：选择具体数量

2). order_detail订单明细表

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	商品名称	冗余字段
image	varchar(255)	商品图片路径	冗余字段
order_id	bigint	订单 id	逻辑外键
dish_id	bigint	菜品 id	逻辑外键
setmeal_id	bigint	套餐 id	逻辑外键
dish_flavor	varchar(50)	菜品口味	
number	int	商品数量	
amount	decimal(10,2)	商品单价	

说明：用户提交订单时，需要往订单表 orders 中插入一条记录，并且需要往 order_detail 中插入一条或多条记录。

2、代码开发

2.1、DTO 设计

根据用户下单接口的参数设计 DTO：

Body

名称	类型	是否必须	默认值	备注
addressBookId	integer	必须		地址簿id
amount	number	必须		总金额
deliveryStatus	integer	必须		配送状态： 1立即送出 0选择具体时间
estimatedDeliveryTime	string	必须		预计送达时间
packAmount	integer	必须		打包费
payMethod	integer	必须		付款方式
remark	string	必须		备注
tablewareNumber	integer	必须		餐具数量
tablewareStatus	integer	必须		餐具数量状态 1按餐量提供 0选择具体数量

在 sky-pojo 模块，OrdersSubmitDTO.java 已定义

```java
package com.sky.dto;

import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;
import java.io.Serializable;
import java.math.BigDecimal;
import java.time.LocalDateTime;

@Data
public class OrdersSubmitDTO implements Serializable {
    //地址簿id
    private Long addressBookId;
    //付款方式
    private int payMethod;
    //备注
    private String remark;
    //预计送达时间
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime estimatedDeliveryTime;
    //配送状态  1立即送出  0选择具体时间
    private Integer deliveryStatus;
    //餐具数量
    private Integer tablewareNumber;
    //餐具数量状态  1按餐量提供  0选择具体数量
    private Integer tablewareStatus;
    //打包费
    private Integer packAmount;
    //总金额
    private BigDecimal amount;
}
```

2.2、VO 设计

根据用户下单接口的返回结果设计 VO：

> 返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
├─id	integer	必须		订单id	format: int64
├─orderAmount	number	必须		订单金额	
├─orderNumber	string	必须		订单号	
├─orderTime	string	必须		下单时间	format: date-time
msg	string	非必须			

在 sky-pojo 模块，OrderSubmitVO.java 已定义

```
1   package com.sky.vo;
2
3   import lombok.AllArgsConstructor;
4   import lombok.Builder;
5   import lombok.Data;
6   import lombok.NoArgsConstructor;
7   import java.io.Serializable;
8   import java.math.BigDecimal;
9   import java.time.LocalDateTime;
10
11  @Data
12  @Builder
13  @NoArgsConstructor
14  @AllArgsConstructor
15  public class OrderSubmitVO implements Serializable {
16      //订单id
17      private Long id;
18      //订单号
19      private String orderNumber;
20      //订单金额
21      private BigDecimal orderAmount;
22      //下单时间
23      private LocalDateTime orderTime;
24  }
```

2.3、Controller 层

创建 OrderController 并提供用户下单方法：

```
1   package com.sky.controller.user;
2
3   import com.sky.dto.OrdersPaymentDTO;
```

```java
import com.sky.dto.OrdersSubmitDTO;
import com.sky.result.PageResult;
import com.sky.result.Result;
import com.sky.service.OrderService;
import com.sky.vo.OrderPaymentVO;
import com.sky.vo.OrderSubmitVO;
import com.sky.vo.OrderVO;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

/**
 * 订单
 */
@RestController("userOrderController")
@RequestMapping("/user/order")
@Slf4j
@Api(tags = "C端-订单接口")
public class OrderController {
    @Autowired
    private OrderService orderService;

    /**
     * 用户下单
     * @param ordersSubmitDTO
     * @return
     */
    @PostMapping("/submit")
    @ApiOperation("用户下单")
    public Result<OrderSubmitVO> submit(@RequestBody OrdersSubmitDTO ordersSubmitDTO)
{
        log.info("用户下单，参数为：{}", ordersSubmitDTO);
        OrderSubmitVO orderSubmitVO = orderService.submitOrder(ordersSubmitDTO);
        return Result.success(orderSubmitVO);
    }
}
```

2.4、Service 层接口

创建 OrderService 接口，并声明用户下单方法：

```java
package com.sky.service;

import com.sky.dto.*;
import com.sky.vo.OrderSubmitVO;

public interface OrderService {
    /**
     * 用户下单
     * @param ordersSubmitDTO
     * @return
     */
    OrderSubmitVO submitOrder(OrdersSubmitDTO ordersSubmitDTO);
}
```

2.5、Service 层实现类

创建 OrderServiceImpl 实现 OrderService 接口：

```java
package com.sky.service.impl;

/**
 * 订单
 */
@Service
@Slf4j
public class OrderServiceImpl implements OrderService {
    @Autowired
    private OrderMapper orderMapper;
    @Autowired
    private OrderDetailMapper orderDetailMapper;
    @Autowired
    private ShoppingCartMapper shoppingCartMapper;
    @Autowired
    private AddressBookMapper addressBookMapper;

    /**
     * 用户下单
     * @param ordersSubmitDTO
     * @return
     */
    @Transactional
    public OrderSubmitVO submitOrder(OrdersSubmitDTO ordersSubmitDTO) {
        //异常情况的处理（收货地址为空、超出配送范围、购物车为空）
        AddressBook addressBook =
addressBookMapper.getById(ordersSubmitDTO.getAddressBookId());
        if (addressBook == null) {
            //抛出业务异常
            throw new
AddressBookBusinessException(MessageConstant.ADDRESS_BOOK_IS_NULL);
        }

        Long userId = BaseContext.getCurrentId();
```

```
33          ShoppingCart shoppingCart = new ShoppingCart();
34          shoppingCart.setUserId(userId);
35
36          //查询当前用户的购物车数据
37          List<ShoppingCart> shoppingCartList = shoppingCartMapper.list(shoppingCart);
38          if (shoppingCartList == null || shoppingCartList.size() == 0) {
39              //抛出业务异常
40              throw new
    ShoppingCartBusinessException(MessageConstant.SHOPPING_CART_IS_NULL);
41          }
42
43          //构造订单数据
44          Orders order = new Orders();
45          BeanUtils.copyProperties(ordersSubmitDTO,order);
46          order.setPhone(addressBook.getPhone());
47          order.setAddress(addressBook.getDetail());
48          order.setConsignee(addressBook.getConsignee());
49          order.setNumber(String.valueOf(System.currentTimeMillis()));
50          order.setUserId(userId);
51          order.setStatus(Orders.PENDING_PAYMENT);
52          order.setPayStatus(Orders.UN_PAID);
53          order.setOrderTime(LocalDateTime.now());
54
55          //向订单表插入1条数据
56          orderMapper.insert(order);
57
58          //订单明细数据
59          List<OrderDetail> orderDetailList = new ArrayList<>();
60          for (ShoppingCart cart : shoppingCartList) {
61              OrderDetail orderDetail = new OrderDetail();
62              BeanUtils.copyProperties(cart, orderDetail);
63              orderDetail.setOrderId(order.getId());
64              orderDetailList.add(orderDetail);
65          }
66
67          //向明细表插入n条数据
68          orderDetailMapper.insertBatch(orderDetailList);
69
70          //清空当前用户的购物车数据
71          shoppingCartMapper.deleteByUserId(userId);
72
73          //封装VO返回结果
74          OrderSubmitVO orderSubmitVO = OrderSubmitVO.builder()
75                  .id(order.getId())
76                  .orderNumber(order.getNumber())
77                  .orderAmount(order.getAmount())
78                  .orderTime(order.getOrderTime())
79                  .build();
80
81          return orderSubmitVO;
82      }
83  }
```

2.6、Mapper层

创建 OrderMapper 接口和对应的 xml 映射文件：

OrderMapper.java

```java
package com.sky.mapper;

@Mapper
public interface OrderMapper {
    /**
     * 插入订单数据
     * @param order
     */
    void insert(Orders orders);
}
```

OrderMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace="com.sky.mapper.OrderMapper">
    <insert id="insert" parameterType="Orders" useGeneratedKeys="true"
keyProperty="id">
        insert into orders
        (number, status, user_id, address_book_id, order_time, checkout_time,
pay_method, pay_status, amount, remark,
         phone, address, consignee, estimated_delivery_time, delivery_status,
pack_amount, tableware_number,
         tableware_status)
        values (#{number}, #{status}, #{userId}, #{addressBookId}, #{orderTime}, #
{checkoutTime}, #{payMethod},
                #{payStatus}, #{amount}, #{remark}, #{phone}, #{address}, #
{consignee},
                #{estimatedDeliveryTime}, #{deliveryStatus}, #{packAmount}, #
{tablewareNumber}, #{tablewareStatus})
    </insert>
</mapper>
```

创建 OrderDetailMapper 接口和对应的 xml 映射文件：

OrderDetailMapper.java

```java
package com.sky.mapper;

import com.sky.entity.OrderDetail;
import java.util.List;

@Mapper
public interface OrderDetailMapper {
    /**
     * 批量插入订单明细数据
     * @param orderDetails
     */
    void insertBatch(List<OrderDetail> orderDetails);
}
```

OrderDetailMapper.xml

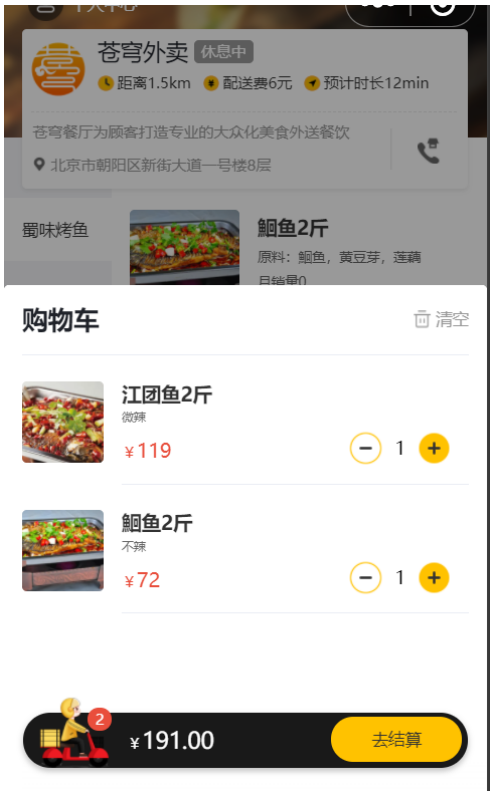
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace="com.sky.mapper.OrderDetailMapper">
    <insert id="insertBatch" parameterType="list">
        insert into order_detail
        (name, order_id, dish_id, setmeal_id, dish_flavor, number, amount, image)
        values
        <foreach collection="orderDetails" item="od" separator=",">
            (#{od.name},#{od.orderId},#{od.dishId},#{od.setmealId},#{od.dishFlavor},
             #{od.number},#{od.amount},#{od.image})
        </foreach>
    </insert>
</mapper>
```

3、功能测试

登录小程序，完成下单操作

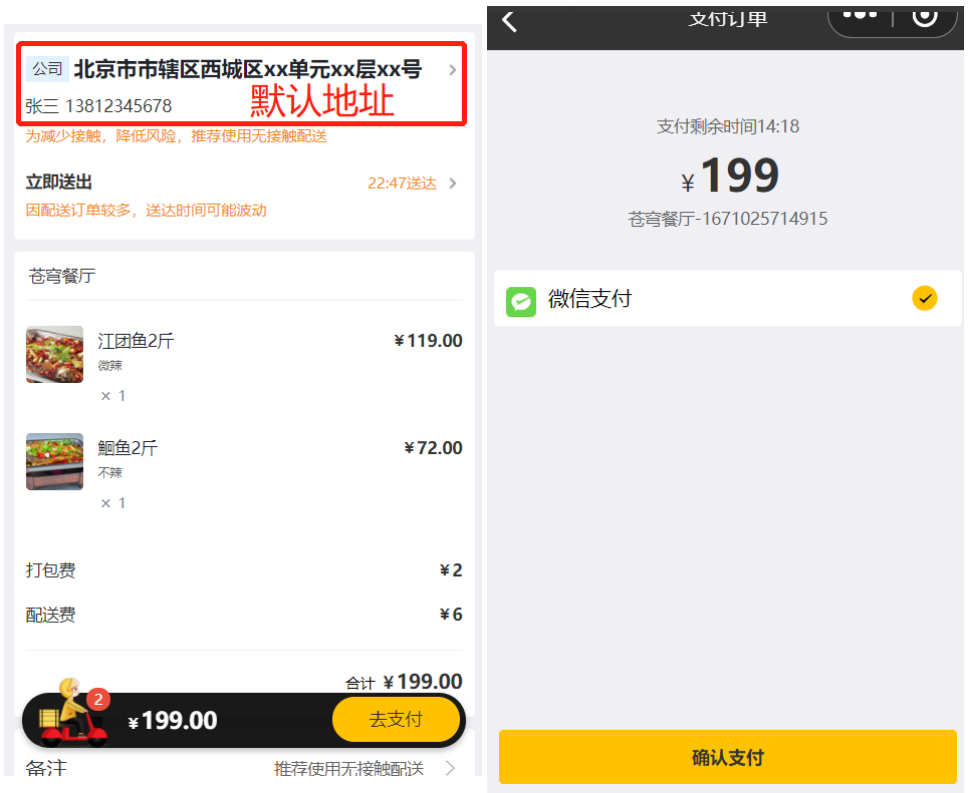
下单操作时，同时会删除购物车中的数据



查看 shopping_cart 表：

id	name	image	user_id	dish_id	setmeal_id	dish_flavor
10	鲳鱼2斤	https://sky-itcast.oss-cn-beijing.aliyuncs.com/8cfcc576-4b66-4a09-a	4	67	(NULL)	不辣
11	江团鱼2斤	https://sky-itcast.oss-cn-beijing.aliyuncs.com/a101a1e9-8f8b-47b2-a	4	66	(NULL)	微辣

去结算 —> 去支付



查看 orders 表：

number	status	user_id	address_book_id	order_time	checkout_time	pay_method	pay_status	amount	remark	phone	address
1671025714915	1	4	2	2022-12-14 21:48:35	(NULL)	1	0	199.00		13812345678	xx单元xx层xx号

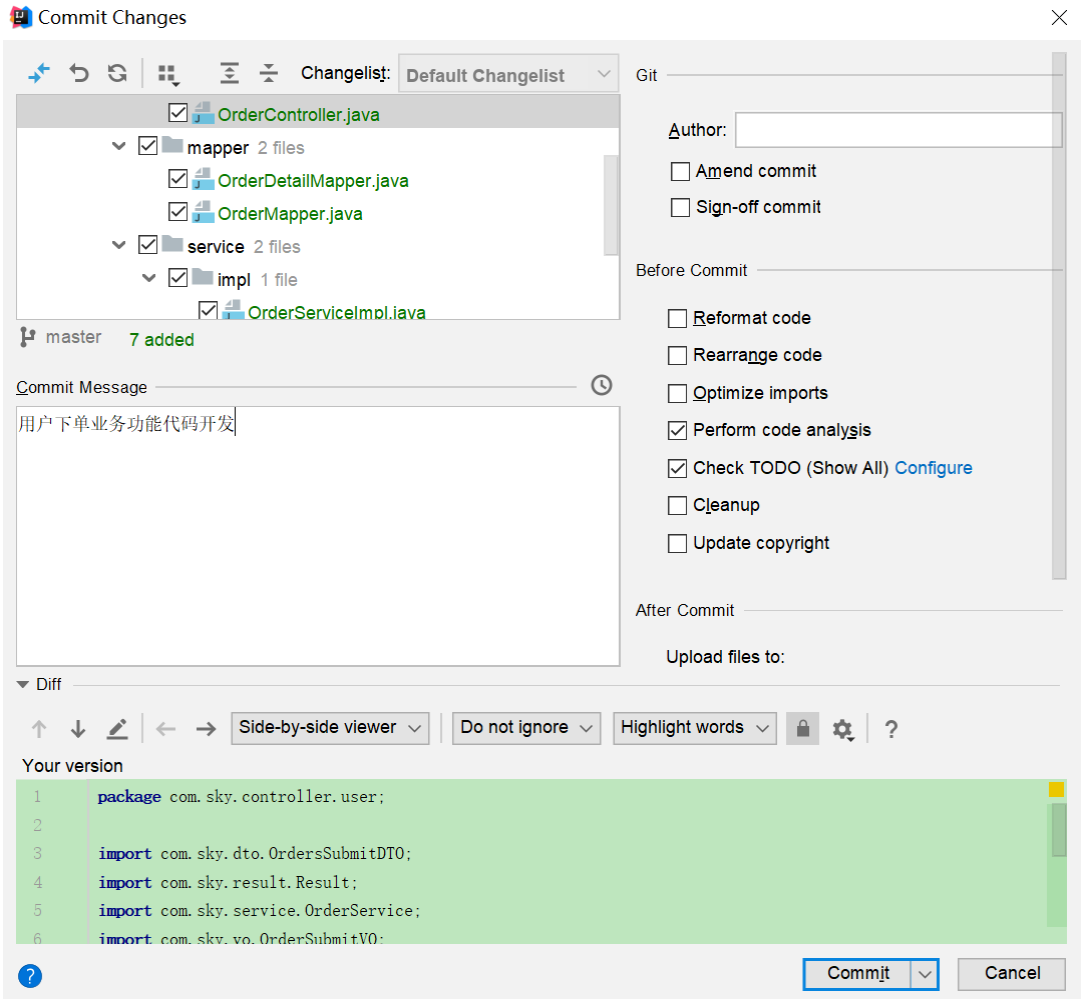
查看 order_detail 表：

id	name	image	order_id	dish_id	setmeal_id	dish_flavor	number	amount
5	江团鱼2斤	https://sky-itcast.oss-cn-beijing.aliyuncs.com/a101a1e9-8f8b-47b2-a	4	66	(NULL)	微辣	1	119.00
6	鲳鱼2斤	https://sky-itcast.oss-cn-beijing.aliyuncs.com/8cfcc576-4b66-4a09-a	4	67	(NULL)	不辣	1	72.00

同时，购物车表中数据删除：

id	name	image	user_id	dish_id	setmeal_id	dish_flavor	number	amount	create_time
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	1	(NULL)	(NULL)

4、代码提交



后续步骤和其它功能代码提交一致，不再赘述。

三、订单支付

1、微信支付介绍

前面的课程已经实现了用户下单，那接下来就是订单支付，就是完成付款功能。支付大家应该都不陌生了，在现实生活中经常购买商品并且使用支付功能来付款，在付款的时候可能使用比较多的就是微信支付和支付宝支付了。在苍穹外卖项目中，选择的就是**微信支付**这种支付方式。

要实现微信支付就需要注册微信支付的一个商户号，这个商户号是必须要有一家企业并且有正规的营业执照。只有具备了这些资质之后，才可以去注册商户号，才能开通支付权限。

个人不具备这种资质，所以我们在学习微信支付时，最重要的是了解微信支付的流程，并且能够阅读微信官方提供的接口文档，能够和第三方支付平台对接起来就可以了。

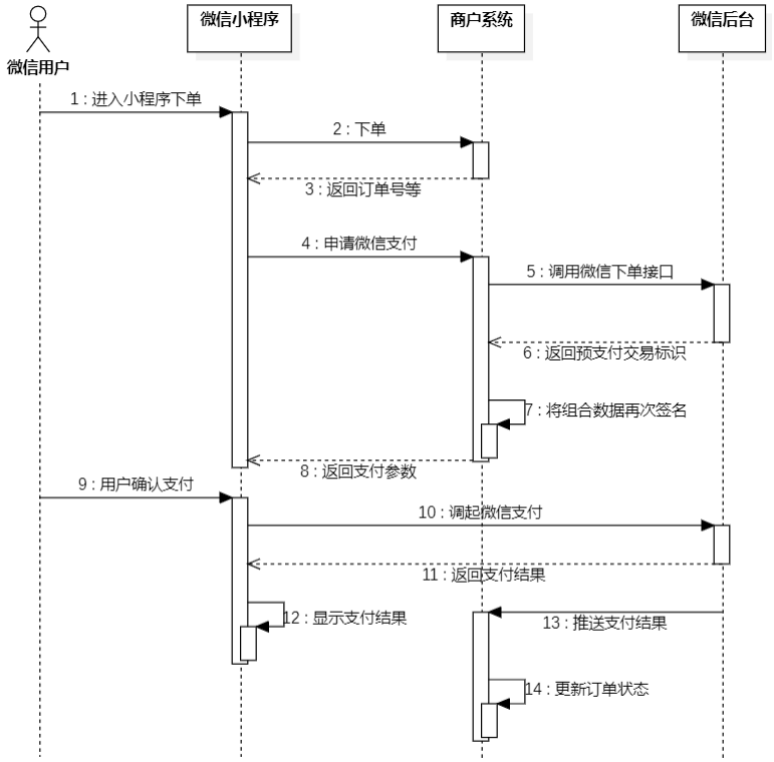
微信支付产品：

本项目选择**小程序支付**

参考： https://pay.weixin.qq.com/static/product/product_index.shtml

微信支付接入流程：

微信小程序支付时序图：



微信支付相关接口：

JSAPI下单：商户系统调用该接口在微信支付服务后台生成预支付交易单（对应时序图的第 5 步）

适用对象： 直连商户

请求URL： https://api.mch.weixin.qq.com/v3/pay/transactions/jsapi

请求方式： POST

```
{
  "mchid": "1900006XXX",
  "out_trade_no": "1217752501201407033233368318",
  "appid": "wxdace645e0bc2cXXX",
  "description": "Image形象店-深圳腾大-QQ公仔",
  "notify_url": "https://www.weixin.qq.com/wxpay/pay.php",
  "amount": {
    "total": 1,
    "currency": "CNY"
  },
  "payer": {
    "openid": "o4GgauInH_RCEdvrrNGrntXDuXXX"
  }
}
```

返回参数

参数名	变量	类型[长度限制]	必填	描述
预支付交易会话标识	prepay_id	string[1,64]	是	预支付交易会话标识。用于后续接口调用中使用，该值有效期为2小时 示例值：wx201410272009395522657a690389285100

微信小程序调起支付：通过 JSAPI 下单接口获取到发起支付的必要参数 prepay_id，然后使用微信支付提供的小程序方法调起小程序支付（对应时序图的第 10 步）

接口名称：wx.requestPayment，详见小程序API文档

Object参数说明：

参数名	变量	类型[长度限制]	必填	描述
时间戳	timeStamp	string[1,32]	是	当前的时间，其他详见时间戳规则。 示例值：1414561699
随机字符串	nonceStr	string[1,32]	是	随机字符串，不长于32位。 示例值：5K8264ILTKCH16CQ2502SI8ZNMTM67VS
订单详情扩展字符串	package	string[1,128]	是	小程序下单接口返回的prepay_id参数值，提交格式如：prepay_id=*** 示例值：prepay_id=wx201410272009395522657a690389285100
签名方式	signType	string[1,32]	是	签名类型，默认为RSA，仅支持RSA。 示例值：RSA
签名	paySign	string[1,512]	是	签名，使用字段appId、timeStamp、nonceStr、package计算得出的签名值 示例值：oR9d8PuhnIc+YZ8cBHFCwfgpaK9gd7vaRvkYD7rthRAZ\/X+QBhcCYL21N7cHCTUxbQ+EAt6Uy+lwSN22f5YZvI45MLko8Pfso0jm46v5hqcVwrk6uddkGuT+Cdvu4WBqDzaDjnNa5UK3GfE1Wfl2gHxIIY5lLdUgWFts17D4WuolLLkiFZV+JSHMvH7ealdT9N5GBovBwu5yYKUR7skR8Fu+LozcSqQixnlEZUfyE55feLOQTUYzLmR9pNtPbPsu6WVhbNHMS3Ss2+AehHvz+n64GDmXxbX++IOBvm2olHu3PsOUGRwhudhVf7UcGcunXt8cqNjKNqZLhLw4jq\/xDg== 请求示例

```
wx.requestPayment
(
  {
    "timeStamp": "1414561699",
    "nonceStr": "5K8264ILTKCH16CQ2502SI8ZNMTM67VS",
    "package": "prepay_id=wx201410272009395522657a690389285100",
    "signType": "RSA",
    "paySign":
"oR9d8PuhnIc+YZ8cBHFCwfgpaK9gd7vaRvkYD7rthRAZ\/X+QBhcCYL21N7cHCTUxbQ+EAt6Uy+lwSN22f5YZvI45MLko8Pfso0jm46v5hqcVwrk6uddkGuT+Cdvu4WBqDzaDjnNa5UK3GfE1Wfl2gHxIIY5lLdUgWFts17D4WuolLLkiFZV+JSHMvH7ealdT9N5GBovBwu5yYKUR7skR8Fu+LozcSqQixnlEZUfyE55feLOQTUYzLmR9pNtPbPsu6WVhbNHMS3Ss2+AehHvz+n64GDmXxbX++IOBvm2olHu3PsOUGRwhudhVf7UcGcunXt8cqNjKNqZLhLw4jq\/xDg==",
    "success":function(res){},
    "fail":function(res){},
    "complete":function(res){}
  }
)
```

2、微信支付准备工作

2.1、如何保证数据安全？

完成微信支付有两个关键的步骤：

第一个就是需要在商户系统当中调用微信后台的一个下单接口，就是生成预支付交易单。

第二个就是支付成功之后微信后台会给推送消息。

这两个接口数据的安全性，要求其实是非常高的。

解决：微信提供的方式就是对数据进行加密、解密、签名多种方式。要完成数据加密解密，需要提前准备相应的一些文件，其实就是一些证书。

获取微信支付平台证书、商户私钥文件：

apiclient_key.pem
wechatpay_166D96F876F45C7D07CE98952A96EC980368ACFC.pem

在后续程序开发过程中，就会使用到这两个文件，需要提前把这两个文件准备好。

2.2、如何调用到商户系统？

微信后台会调用到商户系统给推送支付的结果，在这里我们就会遇到一个问题，就是微信后台怎么就能调用到我们这个商户系统呢？因为这个调用过程，其实本质上也是一个 HTTP 请求。

目前，商户系统它的 ip 地址就是当前自己电脑的 ip 地址，只是一个局域网内的 ip 地址，微信后台无法调用到。

解决：内网穿透。通过 cpolar **软件**可以获得一个临时域名，而这个临时域名是一个公网 ip，这样，微信后台就可以请求到商户系统了。

cpolar 软件的使用：

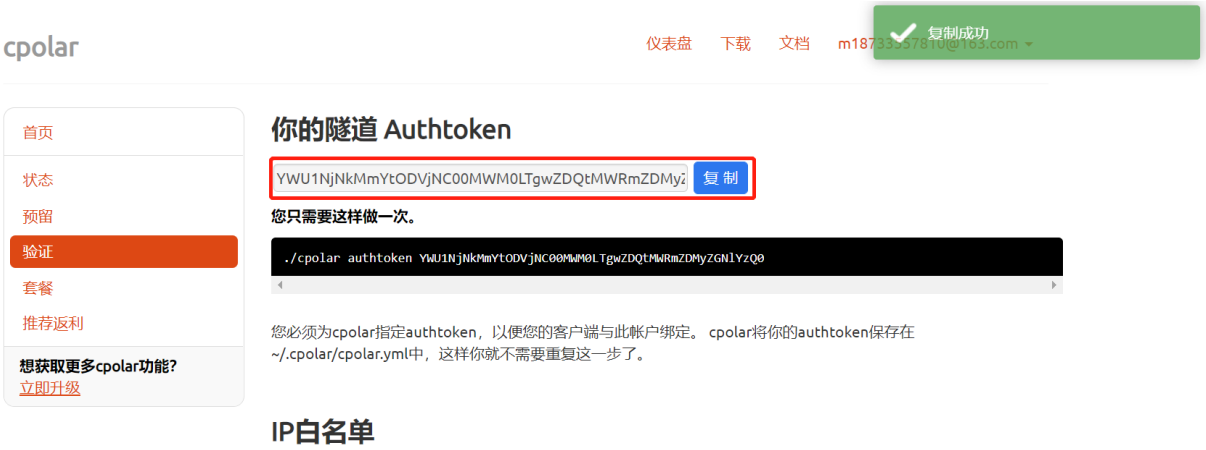
1). **下载与安装**

下载地址： https://dashboard.cpolar.com/get-started

安装过程中，一直下一步即可，不再演示。

2). cpolar 指定 authtoken

复制 authtoken：

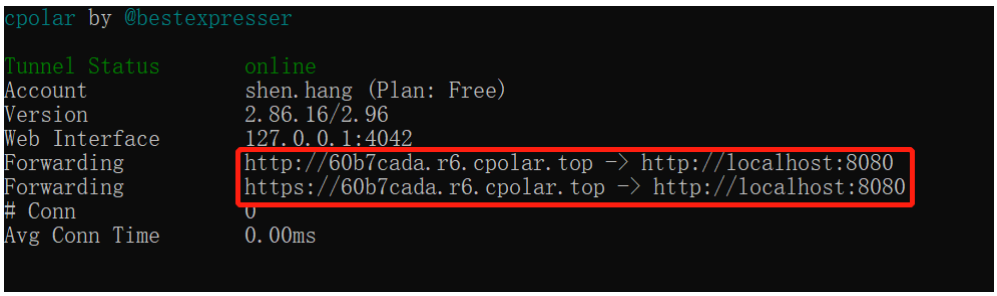


执行命令：

3). 获取临时域名

执行命令：

获取域名：



4). 验证临时域名有效性

访问接口文档

使用 localhost:8080 访问

使用临时域名访问

证明临时域名生效。

3、代码导入

3.1、微信支付相关配置

application-dev.yml

```yaml
sky:
  wechat:
    appid: wxcd2e39f677fd30ba
    secret: 84fbfdf5ea288f0c432d829599083637
    mchid : 1561414331
    mchSerialNo: 4B3B3DC35414AD50B1B755BAF8DE9CC7CF407606
    privateKeyFilePath: D:\apiclient_key.pem
    apiV3Key: CZBK51236435wxpay435434323FFDuv3
    weChatPayCertFilePath: D:\wechatpay_166D96F876F45C7D07CE98952A96EC980368ACFC.pem
    notifyUrl: https://www.weixin.qq.com/wxpay/pay.php
    refundNotifyUrl: https://www.weixin.qq.com/wxpay/pay.php
```

application.yml

```yaml
sky:
  wechat:
    appid: ${sky.wechat.appid}
    secret: ${sky.wechat.secret}
    mchid : ${sky.wechat.mchid}
    mchSerialNo: ${sky.wechat.mchSerialNo}
    privateKeyFilePath: ${sky.wechat.privateKeyFilePath}
    apiV3Key: ${sky.wechat.apiV3Key}
    weChatPayCertFilePath: ${sky.wechat.weChatPayCertFilePath}
    notifyUrl: ${sky.wechat.notifyUrl}
    refundNotifyUrl: ${sky.wechat.refundNotifyUrl}
```

WeChatProperties.java：读取配置（已定义）

```java
package com.sky.properties;

import lombok.Data;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "sky.wechat")
@Data
public class WeChatProperties {

    private String appid; //小程序的appid
    private String secret; //小程序的秘钥
    private String mchid; //商户号
    private String mchSerialNo; //商户API证书的证书序列号
    private String privateKeyFilePath; //商户私钥文件
    private String apiV3Key; //证书解密的密钥
    private String weChatPayCertFilePath; //平台证书
    private String notifyUrl; //支付成功的回调地址
    private String refundNotifyUrl; //退款成功的回调地址
}
```

3.2、Mapper层

在 OrderMapper.java 中添加 getByNumberAndUserId 和 update 两个方法

```java
/**
 * 根据订单号和用户id查询订单
 * @param orderNumber
 * @param userId
 */
@Select("select * from orders where number = #{orderNumber} and user_id= #{userId}")
Orders getByNumberAndUserId(String orderNumber, Long userId);

/**
 * 修改订单信息
 * @param orders
 */
void update(Orders orders);
```

在 OrderMapper.xml 中添加

```xml
<update id="update" parameterType="com.sky.entity.Orders">
    update orders
    <set>
        <if test="cancelReason != null and cancelReason!='' ">
            cancel_reason=#{cancelReason},
        </if>
        <if test="rejectionReason != null and rejectionReason!='' ">
            rejection_reason=#{rejectionReason},
        </if>
        <if test="cancelTime != null">
            cancel_time=#{cancelTime},
        </if>
        <if test="payStatus != null">
            pay_status=#{payStatus},
        </if>
        <if test="payMethod != null">
            pay_method=#{payMethod},
        </if>
        <if test="checkoutTime != null">
            checkout_time=#{checkoutTime},
        </if>
        <if test="status != null">
            status = #{status},
        </if>
        <if test="deliveryTime != null">
            delivery_time = #{deliveryTime}
        </if>
    </set>
    where id = #{id}
</update>
```

3.3、Service层

在 OrderService.java 中添加 payment 和 paySuccess 两个方法定义

```
/**
 * 订单支付
 * @param ordersPaymentDTO
 * @return
 */
OrderPaymentVO payment(OrdersPaymentDTO ordersPaymentDTO) throws Exception;

/**
 * 支付成功，修改订单状态
 * @param outTradeNo
 */
void paySuccess(String outTradeNo);
```

在 OrderServiceImpl.java 中实现 payment 和 paySuccess 两个方法

```
@Autowired
private UserMapper userMapper;
@Autowired
private WeChatPayUtil weChatPayUtil;

/**
 * 订单支付
 * @param ordersPaymentDTO
 * @return
 */
public OrderPaymentVO payment(OrdersPaymentDTO ordersPaymentDTO) throws Exception {
    // 当前登录用户id
    Long userId = BaseContext.getCurrentId();
    User user = userMapper.getById(userId);

    //调用微信支付接口，生成预支付交易单
    JSONObject jsonObject = weChatPayUtil.pay(
        ordersPaymentDTO.getOrderNumber(), //商户订单号
        new BigDecimal(0.01), //支付金额，单位 元
        "苍穹外卖订单", //商品描述
        user.getOpenid() //微信用户的openid
    );

    if (jsonObject.getString("code") != null &&
jsonObject.getString("code").equals("ORDERPAID")) {
        throw new OrderBusinessException("该订单已支付");
    }

    OrderPaymentVO vo = jsonObject.toJavaObject(OrderPaymentVO.class);
    vo.setPackageStr(jsonObject.getString("package"));

    return vo;
}

```

```java
/**
 * 支付成功，修改订单状态
 * @param outTradeNo
 */
public void paySuccess(String outTradeNo) {
    // 当前登录用户id
    Long userId = BaseContext.getCurrentId();

    // 根据订单号查询当前用户的订单
    Orders ordersDB = orderMapper.getByNumberAndUserId(outTradeNo, userId);

    // 根据订单id更新订单的状态、支付方式、支付状态、结账时间
    Orders orders = Orders.builder()
        .id(ordersDB.getId())
        .status(Orders.TO_BE_CONFIRMED)
        .payStatus(Orders.PAID)
        .checkoutTime(LocalDateTime.now())
        .build();

    orderMapper.update(orders);
}
```

3.4、Controller层

在 OrderController.java 中添加 payment 方法

```java
/**
 * 订单支付
 * @param ordersPaymentDTO
 * @return
 */
@PutMapping("/payment")
@ApiOperation("订单支付")
public Result<OrderPaymentVO> payment(@RequestBody OrdersPaymentDTO ordersPaymentDTO)
throws Exception {
    log.info("订单支付：{}", ordersPaymentDTO);
    OrderPaymentVO orderPaymentVO = orderService.payment(ordersPaymentDTO);
    log.info("生成预支付交易单：{}", orderPaymentVO);
    return Result.success(orderPaymentVO);
}
```

PayNotifyController.java

```java
package com.sky.controller.notify;

import com.alibaba.druid.support.json.JSONUtils;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.sky.annotation.IgnoreToken;
import com.sky.properties.WeChatProperties;
import com.sky.service.OrderService;
import com.wechat.pay.contrib.apache.httpclient.util.AesUtil;
```

```
import lombok.extern.slf4j.Slf4j;
import org.apache.http.entity.ContentType;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;

/**
 * 支付回调相关接口
 */
@RestController
@RequestMapping("/notify")
@Slf4j
public class PayNotifyController {
    @Autowired
    private OrderService orderService;
    @Autowired
    private WeChatProperties weChatProperties;

    /**
     * 支付成功回调
     * @param request
     */
    @RequestMapping("/paySuccess")
    public void paySuccessNotify(HttpServletRequest request, HttpServletResponse response) throws Exception {
        //读取数据
        String body = readData(request);
        log.info("支付成功回调：{}", body);

        //数据解密
        String plainText = decryptData(body);
        log.info("解密后的文本：{}", plainText);

        JSONObject jsonObject = JSON.parseObject(plainText);
        String outTradeNo = jsonObject.getString("out_trade_no");//商户平台订单号
        String transactionId = jsonObject.getString("transaction_id");//微信支付交易号

        log.info("商户平台订单号：{}", outTradeNo);
        log.info("微信支付交易号：{}", transactionId);

        //业务处理，修改订单状态、来单提醒
        orderService.paySuccess(outTradeNo);

        //给微信响应
        responseToWeixin(response);
    }

```

```
61      /**
62       * 读取数据
63       * @param request
64       * @return
65       * @throws Exception
66       */
67      private String readData(HttpServletRequest request) throws Exception {
68          BufferedReader reader = request.getReader();
69          StringBuilder result = new StringBuilder();
70          String line = null;
71          while ((line = reader.readLine()) != null) {
72              if (result.length() > 0) {
73                  result.append("\n");
74              }
75              result.append(line);
76          }
77          return result.toString();
78      }
79
80      /**
81       * 数据解密
82       * @param body
83       * @return
84       * @throws Exception
85       */
86      private String decryptData(String body) throws Exception {
87          JSONObject resultObject = JSON.parseObject(body);
88          JSONObject resource = resultObject.getJSONObject("resource");
89          String ciphertext = resource.getString("ciphertext");
90          String nonce = resource.getString("nonce");
91          String associatedData = resource.getString("associated_data");
92
93          AesUtil aesUtil = new
    AesUtil(weChatProperties.getApiV3Key().getBytes(StandardCharsets.UTF_8));
94          //密文解密
95          String plainText =
    aesUtil.decryptToString(associatedData.getBytes(StandardCharsets.UTF_8),
96                  nonce.getBytes(StandardCharsets.UTF_8),
97                  ciphertext);
98
99          return plainText;
100     }
101
102     /**
103      * 给微信响应
104      * @param response
105      */
106     private void responseToWeixin(HttpServletResponse response) throws Exception{
107         response.setStatus(200);
108         HashMap<Object, Object> map = new HashMap<>();
109         map.put("code", "SUCCESS");
110         map.put("message", "SUCCESS");
```

```
111             response.setHeader("Content-type", ContentType.APPLICATION_JSON.toString());
112 
        response.getOutputStream().write(JSONUtils.toJSONString(map).getBytes(StandardCharse
        ts.UTF_8));
113             response.flushBuffer();
114         }
115     }
```

4、功能测试

测试过程中，可通过断点方式查看后台每一步执行情况。

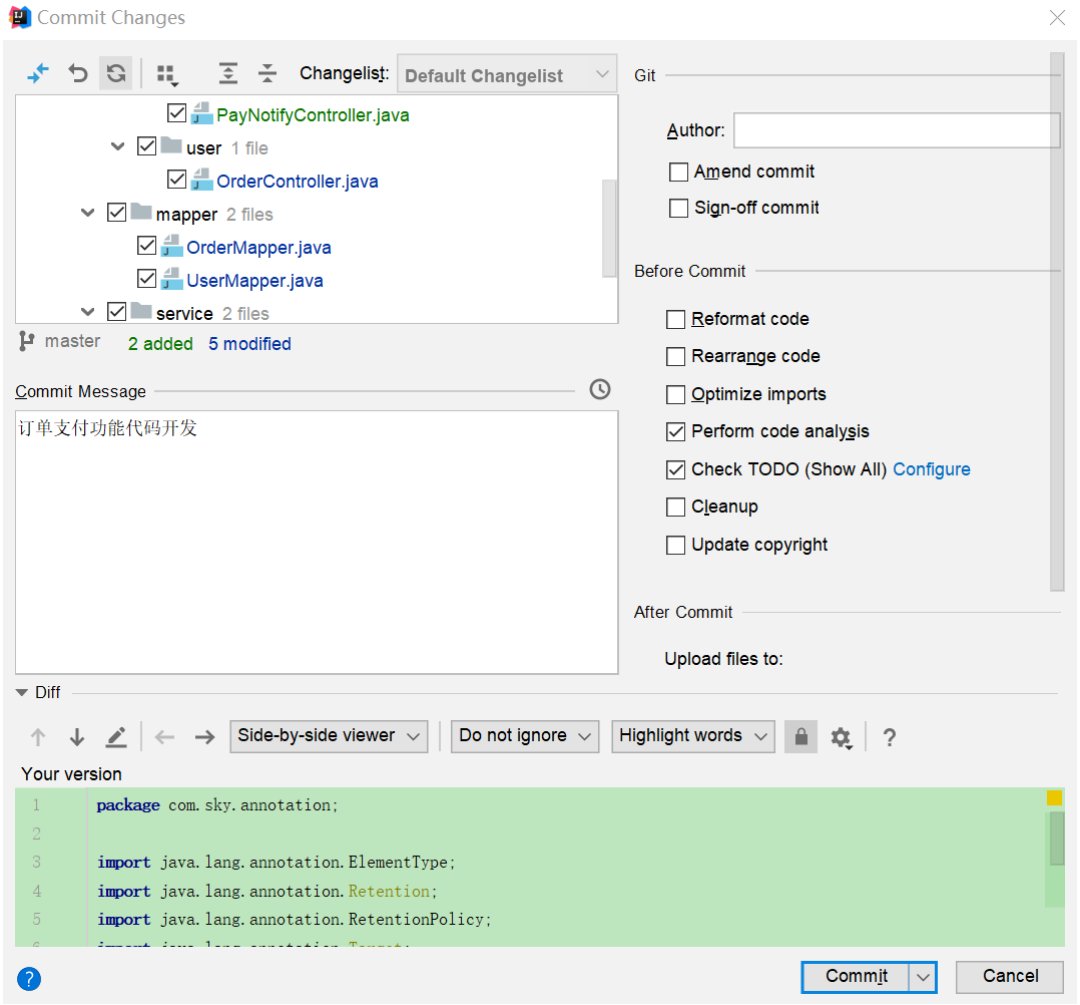
下单：

去支付：

确认支付：

进行扫码支付即可。

5、代码提交



后续步骤和其它功能代码提交一致，不再赘述。