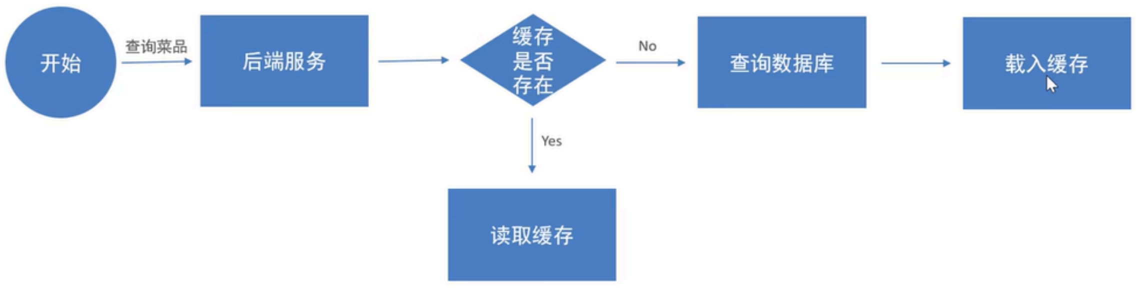
苍穹外卖day7

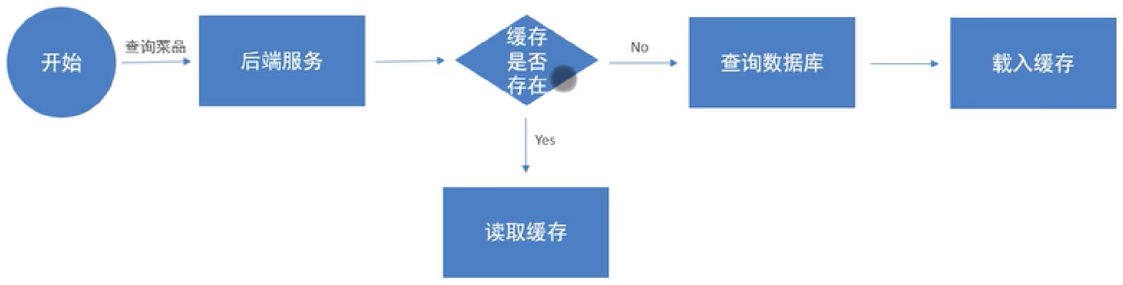
1. 缓存菜品

通过redis来缓存菜品数据，减少数据库查询操作



缓存逻辑分析：

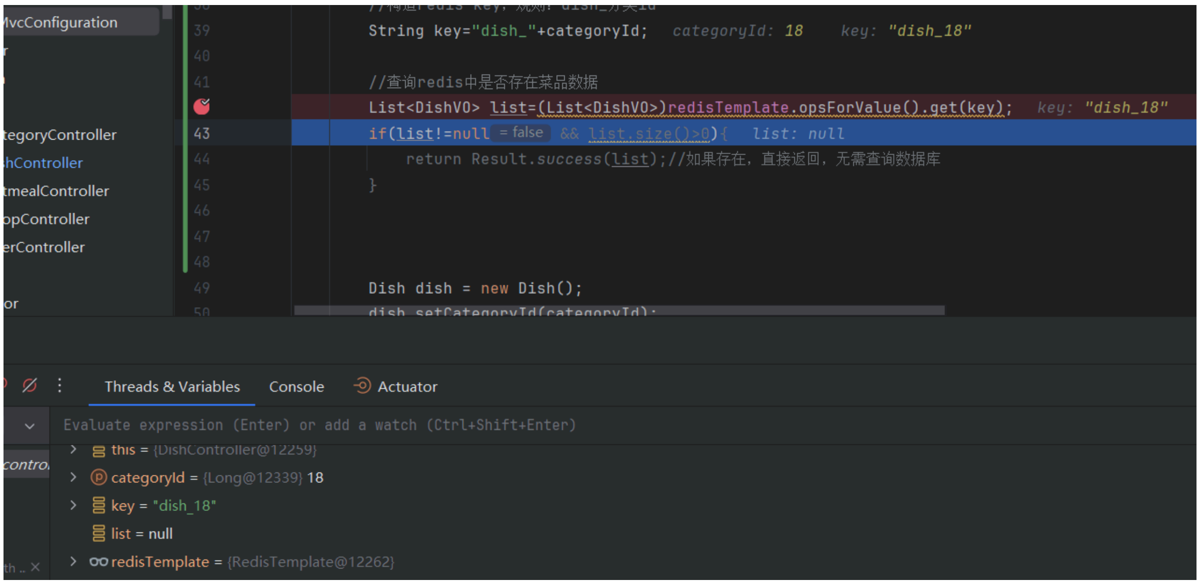
- 每个分类下的菜品保存一份缓存数据
- 数据库中菜品数据有变更时清理缓存数据



com/sky/controller/user/DishController.java

```
    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 根据分类id查询菜品
     *
     * @param categoryId
     * @return
     */
    @GetMapping("/list")
```

```java
    @ApiOperation("根据分类id查询菜品")
    public Result<List<DishVO>> list(Long categoryId) {
        //构造redis key，规则：dish_分类id
        String key="dish_"+categoryId;

        //查询redis中是否存在菜品数据
        List<DishVO> list=(List<DishVO>)redisTemplate.opsForValue().get(key);
        if(list!=null && list.size()>0){
            return Result.success(list);//如果存在，直接返回，无需查询数据库
        }



        Dish dish = new Dish();
        dish.setCategoryId(categoryId);
        dish.setStatus(StatusConstant.ENABLE);//查询起售中的菜品


        //如果不存在，查询数据库，将查询到的数据放入redis中
        list = dishService.listWithFlavor(dish);
        redisTemplate.opsForValue().set(key,list);//放到redis中

        return Result.success(list);
    }
```



```
        String key="dish_"+categoryId;   categoryId: 18    key: "dish_18"

        //查询redis中是否存在菜品数据
        List<DishVO> list=(List<DishVO>)redisTemplate.opsForValue().get(key);   key: "dish_18"
        if(list!=null = false && list.size()>0){   list: null
            return Result.success(list);//如果存在，直接返回，无需查询数据库
        }



        Dish dish = new Dish();
```

```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> this = {DishController@12259}
> categoryId = {Long@12339} 18
> key = "dish_18"
  list = null
> redisTemplate = {RedisTemplate@12262}
```

2. 解决缓存一致性问题和启用禁用菜品

即什么时候清理缓存数据

需要改造管理端的代码：

- 新增菜品
- 修改菜品

- 批量删除菜品
- 起售、停售菜品

com/sky/controller/admin/DishController.java在对应位置添加代码

```java
    @Autowired
    private RedisTemplate redisTemplate;

    public Result save(@RequestBody DishDTO dishDTO){
        //清理缓存数据
        String key="dish_"+ dishDTO.getCategoryId();
        cleanCache(key);
    }

    public Result delete(@RequestParam List<Long> ids){
        //将所有的菜品缓存数据清理掉，所有以dish_*开头的key
        cleanCache("dish_*");
    }

    public Result update(@RequestBody DishDTO dishDTO){
        //将所有的菜品缓存数据清理掉，所有以dish_*开头的key
        cleanCache("dish_*");
    }

    /**
     * 菜品起售停售
     * @return
     */
    @PostMapping("/status/{status}")
    @ApiOperation("菜品起售停售")
    public Result<String> startOrStop(@PathVariable Integer status,Long id){
        dishService.startOrStop(status,id);

        //将所有的菜品缓存数据清理掉，所有以dish_*开头的key
        cleanCache("dish_*");
        return Result.success();
    }

    /**
     * 清理缓存数据
     * @param pattern
     */
    private void cleanCache(String pattern){
        Set keys = redisTemplate.keys(pattern);
        redisTemplate.delete(keys);
    }
```

DishService.java

```java
    /**
     * 菜品起售停售
     * @param status
     * @param id
     */
    void startOrStop(Integer status, Long id);
```

DishServiceImpl.java

```java
    @Autowired
    private SetmealMapper setmealMapper;

    /**
     * 启用禁用菜品
     * @param status
     * @param id
     */
    @Override
    public void startOrStop(Integer status, Long id) {
        Dish dish = new Dish();
        dish.setId(id);
        dish.setStatus(status);
        dishMapper.update(dish);

        // 如果是禁用，需要将套餐中的菜品也禁用
        if (status == StatusConstant.DISABLE) {
            List<Long> dishIds = new ArrayList<>();
            dishIds.add(id);
            List<Long> setmealIds =
setmealDishMapper.getSetmealIdByDishIds(dishIds);//获取菜品对应的套餐id
            if (setmealIds != null && setmealIds.size() > 0) {
                for (Long setmealId : setmealIds) {
                    Setmeal setmeal = Setmeal.builder()
                            .id(setmealId)
                            .status(StatusConstant.DISABLE)
                            .build();
                    setmealMapper.update(setmeal);
                }
            }
        }
    }
```

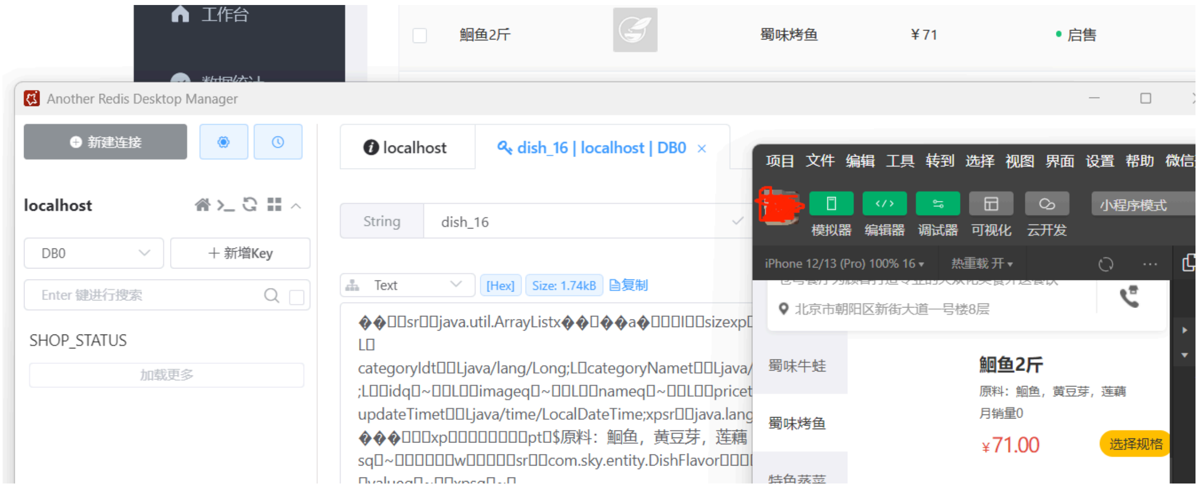
SetmealDishMapper.java

```java
    /**
     * 根据菜品id查询套餐id
     *
     * @param dishIds
     * @return
     */
    List<Long> getSetmealIdByDishIds(List<Long> dishIds);
```

SetmealDishMapper.xml

```xml
    <select id="getSetmealIdByDishIds" resultType="java.lang.Long">
        select  setmeal_id from setmeal_dish where dish_id  in
        <foreach collection="dishIds" item="dishId" open="(" separator=","
close=")">
            #{dishId}
        </foreach>
    </select>
```

可以看到管理端修改价格后，会清理缓存。



[redacted]

```
��□□sr□□java.util.ArrayListx��□��a�□□□I□□sizexp□
L□
categoryIdt□□Ljava/lang/Long;L□categoryNamet□□Ljava/
;L□□idq□~□□L□□imageq□~□□L□□nameq□~□□L□□pricet
updateTimet□□Ljava/time/LocalDateTime;xpsr□□java.lang
���□□□xp□□□□□□□□□pt□$原料：鲫鱼，黄豆芽，莲藕
sq□~□□□□□□w□□□□□sr□□com.sky.entity.DishFlavor□□□
```

3. Spring Cache入门案例

Spring Cache是一个框架，实现了基于注解的缓存功能，只需要简单地加一个注解，就能实现缓存功能。

Spring Cache提供了一层抽象，底层可以切换不同的缓存实现，例如：

- EHCache
- Caffeine
- Redis

常用注解：

注解	说明
@EnableCaching	开启缓存注解功能，通常加在启动类上
@Cacheable	在方法执行前先查询缓存中是否有数据，如果有数据，则直接返回缓存数据；如果没有缓存数据，调用方法并将方法返回值放到缓存中
@CachePut	将方法的返回值放到缓存中
@CacheEvict	将一条或多条数据从缓存中删除

idea将工程`C:\baidunetdiskdownload\资料\day07\springcache-demo`复制到非中文路径并打开。

application.yml修改密码

```yaml
spring:
  datasource:
    druid: #数据库连接池
      driver-class-name: com.mysql.cj.jdbc.Driver
      url: jdbc:mysql://localhost:3306/spring_cache_demo?serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8&zeroDateTimeBehavior=convertToNull&useSSL=false&allowPublicKeyRetrieval=true
      username: root
      password: 123456
```

navicat新建数据库`spring_cache_demo`。

```sql
create DATABASE if not EXISTS spring_cache_demo;
use spring_cache_demo;
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `name` varchar(45) DEFAULT NULL,
  `age` int DEFAULT NULL,
  PRIMARY KEY (`id`)
);
```

CacheDemoApplication.java

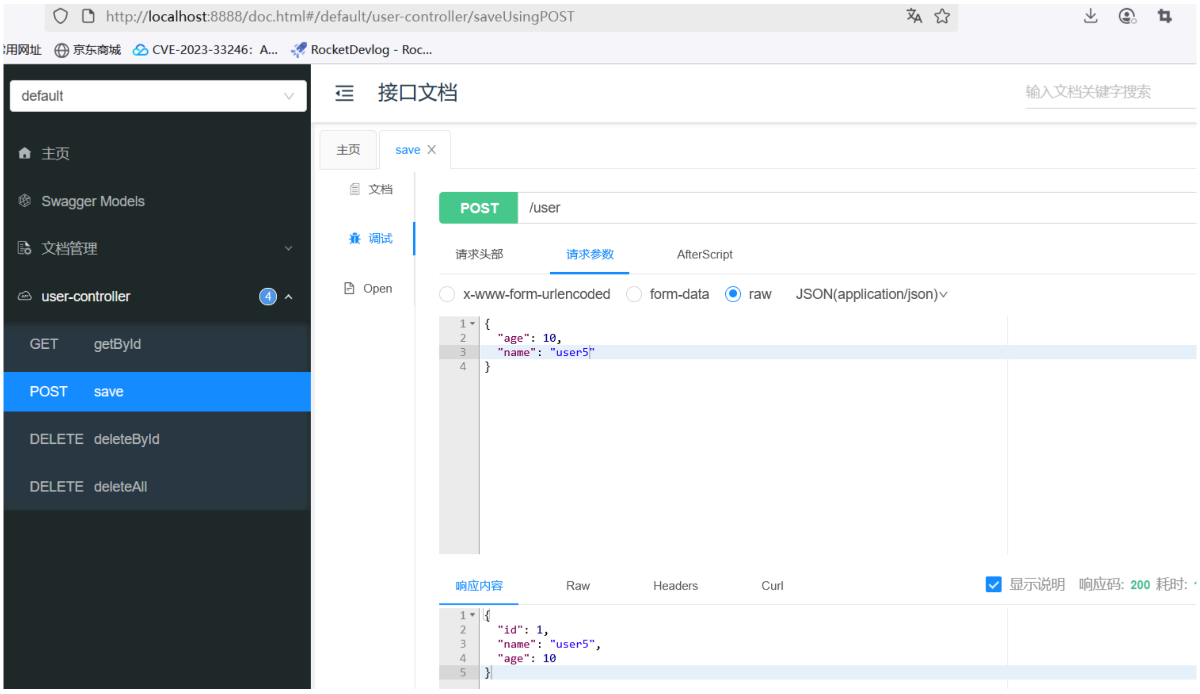
```java
@Slf4j
@SpringBootApplication
@EnableCaching //开启缓存注解功能
public class CacheDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(CacheDemoApplication.class,args);
        log.info("项目启动成功...");
    }
}
```

UserController.java添加

```java
    @PostMapping
    //@CachePut(cacheNames = "userCache",key="#user.id") //SpEL表达式
    //如果使用Spring Cache缓存数据，key的生成：userCache::xxx
    @CachePut(cacheNames = "userCache",key="#result.id")//取返回值的id属性，"."称为对象导航
    //@CachePut(cacheNames = "userCache",key="#p0.id") //取save函数第一个参数的id属性
    //@CachePut(cacheNames = "userCache",key="#a0.id") //取save函数第一个参数的id属性
    //insert操作之后，将id值赋值给user
    //而@Cacheput操作在mysql insert操作之后，所以返回值的user和参数里的user此时是一样的
    public User save(@RequestBody User user){
        userMapper.insert(user);
        return user;
    }

    @DeleteMapping
    @CacheEvict(cacheNames = "userCache",key="#id")
    public void deleteById(Long id){
        userMapper.deleteById(id);
    }

    @DeleteMapping("/delAll")
    @CacheEvict(cacheNames = "userCache", allEntries=true)
    public void deleteAll(){
        userMapper.deleteAll();
    }

    @GetMapping
    @Cacheable(cacheNames = "userCache",key="#id") //key的生成：userCache::xxx
    //在方法执行前先查询缓存中是否有数据，如果有数据，则直接返回缓存数据；如果没有缓存数据，调用方法并将方法返回值放到缓存中
    public User getById(Long id){
        User user = userMapper.getById(id);
        return user;
    }
```

maven刷新清理编译一下

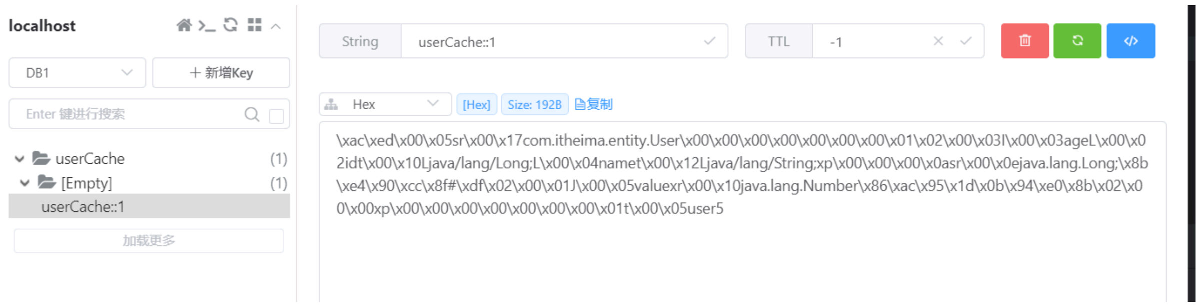
访问http://localhost:8888/doc.html



```
{
  "age": 10,
  "name": "user5"
}
```

```
{
  "id": 1,
  "name": "user5",
  "age": 10
}
```

因为有两个`:`，所以中间会有个Empty文件夹



```
\xac\xed\x00\x05sr\x00\x17com.itheima.entity.User\x00\x00\x00\x00\x00\x00\x00\x01\x02\x00\x03I\x00\x03ageL\x00\x02idt\x00\x10Ljava/lang/Long;L\x00\x04namet\x00\x12Ljava/lang/String;xp\x00\x00\x00\x0asr\x00\x0ejava.lang.Long;\x8b\xe4\x90\xcc\x8f#\xdf\x02\x00\x01J\x00\x05valuexr\x00\x10java.lang.Number\x86\xac\x95\x1d\x0b\x94\xe0\x8b\x02\x00\x00xp\x00\x00\x00\x00\x00\x00\x00\x01t\x00\x05user5
```

4. 缓存套餐

缓存套餐实现思路：

- 导入Spring Cache和Redis相关maven坐标
- 在启动类上加入@EnableCaching注解，开启缓存注解功能
- 在用户端接口SetmealController的list方法上加入@Cacheable注解
- 在管理端接口SetmealController的save、delete、update、startOrStop等方法上加入CacheEvict注解

SkyApplication.java

```java
@EnableCaching
public class SkyApplication {
```

com/sky/controller/user/SetmealController.java

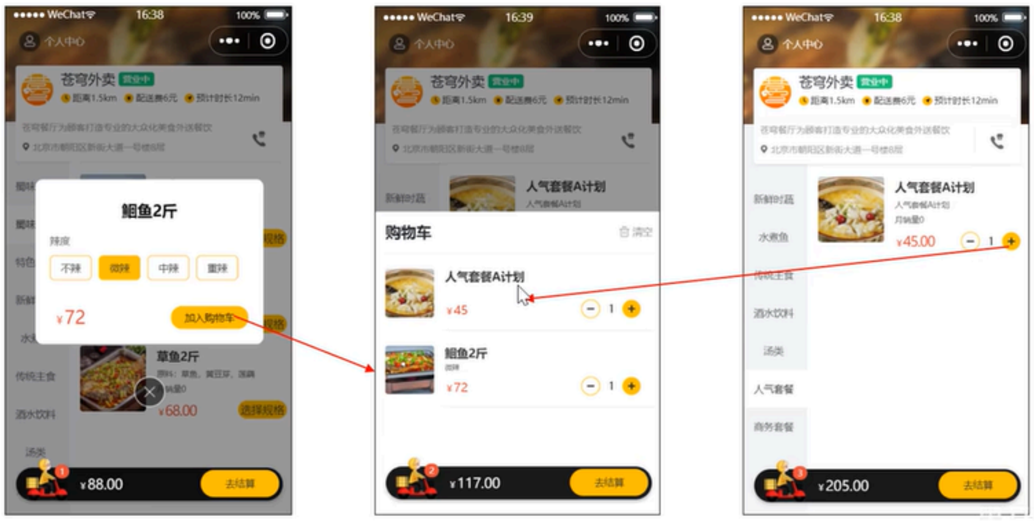
```java
    @Cacheable(cacheNames = "setmealCache",key="#categoryId") //key: setmealCache::xxx
    public Result<List<Setmeal>> list(Long categoryId) {
```

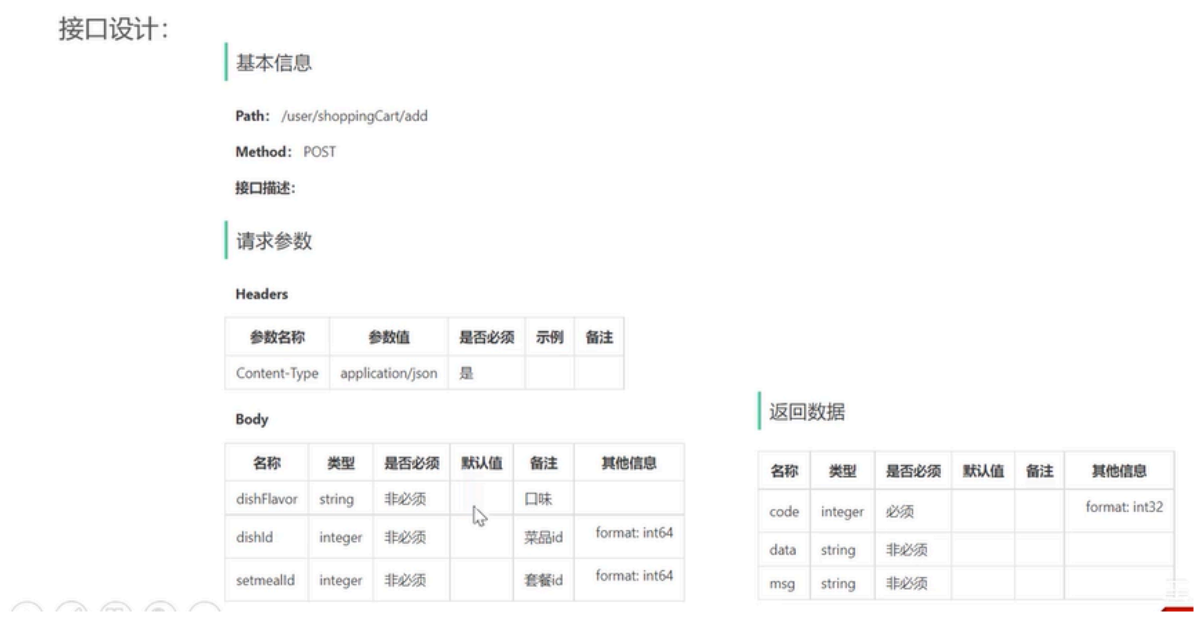
com/sky/controller/admin/SetmealController.java

```java
    @CacheEvict(cacheNames = "setmealCache",key="#setmealDTO.categoryId") //删除缓存
    public Result save(@RequestBody SetmealDTO setmealDTO) {

    @CacheEvict(cacheNames = "setmealCache",allEntries = true)
    public Result delete(@RequestParam List<Long> ids){

    @CacheEvict(cacheNames = "setmealCache",allEntries = true)
    public Result update(@RequestBody SetmealDTO setmealDTO) {

    @CacheEvict(cacheNames = "setmealCache",allEntries = true)
    public Result startOrStop(@PathVariable Integer status, Long id) {
```

5. 添加购物车

产品原型：




接口设计:

基本信息

Path: /user/shoppingCart/add

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	其他信息
dishFlavor	string	非必须		口味	
dishId	integer	非必须		菜品id	format: int64
setmealId	integer	非必须		套餐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	string	非必须			
msg	string	非必须			

数据库设计

- 作用：暂时存放所选商品的地方
- 选的什么商品
- 每个商品都买了几个
- 不同用户的购物车需要区分开

详见shopping_cart表

sky-server/src/main/java/com/sky/controller/user/ShoppingCartController.java

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

import com.sky.dto.ShoppingCartDTO;
import com.sky.result.Result;
import com.sky.service.ShoppingCartService;
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.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/user/shoppingCart")
@Slf4j
@Api(tags="C端购物车相关接口")
public class ShoppingCartController {

    @Autowired //注入业务层接口
    private ShoppingCartService shoppingCartService;


    /**
     * 添加购物车
     * @param shoppingCartDTO
     * @return
     */
    @PostMapping("/add")
    @ApiOperation("添加购物车")
    public Result add(@RequestBody ShoppingCartDTO shoppingCartDTO) {
        log.info("添加购物车，商品信息为：{}", shoppingCartDTO);
        shoppingCartService.addShoppingCart(shoppingCartDTO);
        return Result.success();
    }
}

```

sky-server/src/main/java/com/sky/service/ShoppingCartService.java

```java
package com.sky.service;

import com.sky.dto.ShoppingCartDTO;

public interface ShoppingCartService {

    /**
     * 添加购物车
     * @param shoppingCartDTO
     */
    void addShoppingCart(ShoppingCartDTO shoppingCartDTO);

}

```

sky-server/src/main/java/com/sky/service/impl/ShoppingCartServiceImpl.java

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

import com.sky.context.BaseContext;
import com.sky.dto.ShoppingCartDTO;
import com.sky.entity.Dish;
import com.sky.entity.Setmeal;
import com.sky.entity.ShoppingCart;
import com.sky.mapper.DishMapper;
import com.sky.mapper.SetmealMapper;
import com.sky.mapper.ShoppingCartMapper;
import com.sky.service.ShoppingCartService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
@Slf4j
public class ShoppingCartServiceImpl implements ShoppingCartService {

    //注入mapper层接口
    @Autowired
    private ShoppingCartMapper shoppingCartMapper;

    @Autowired
    private DishMapper dishMapper;

    @Autowired
    private SetmealMapper setmealMapper;

    /**
     * 添加购物车
     * @param shoppingCartDTO
     */
    public void addShoppingCart(ShoppingCartDTO shoppingCartDTO) {
        //判断当前加入到购物车中的商品是否已经存在
        ShoppingCart shoppingCart = new ShoppingCart();
        BeanUtils.copyProperties(shoppingCartDTO, shoppingCart);
        Long userId = BaseContext.getCurrentId();
        shoppingCart.setUserId(userId);

        List<ShoppingCart> list=shoppingCartMapper.list(shoppingCart);

        //如果已经存在了，只需要将数量加一
        if(list!=null&&list.size()>0){
            ShoppingCart cart=list.get(0);
            cart.setNumber(cart.getNumber()+1);
            shoppingCartMapper.updateNumberById(cart);
        }else{
            //如果不存在，需要插入一条购物车数据

            //判断本次添加到购物车的是菜品还是套餐
            Long dishId=shoppingCartDTO.getDishId();
            if(dishId!=null){
                //添加的是菜品
                Dish dish=dishMapper.getById(dishId);
                shoppingCart.setName(dish.getName());
                shoppingCart.setImage(dish.getImage());
                shoppingCart.setAmount(dish.getPrice());

            }else{
                //添加的是套餐
                Long setmealId=shoppingCartDTO.getSetmealId();
                Setmeal setmeal=setmealMapper.getById(setmealId);
                shoppingCart.setName(setmeal.getName());
                shoppingCart.setImage(setmeal.getImage());
                shoppingCart.setAmount(setmeal.getPrice());

            }
            shoppingCart.setNumber(1);
            shoppingCart.setCreateTime(LocalDateTime.now());
            shoppingCartMapper.insert(shoppingCart);
        }
    }
}

```

sky-server/src/main/java/com/sky/mapper/ShoppingCartMapper.java

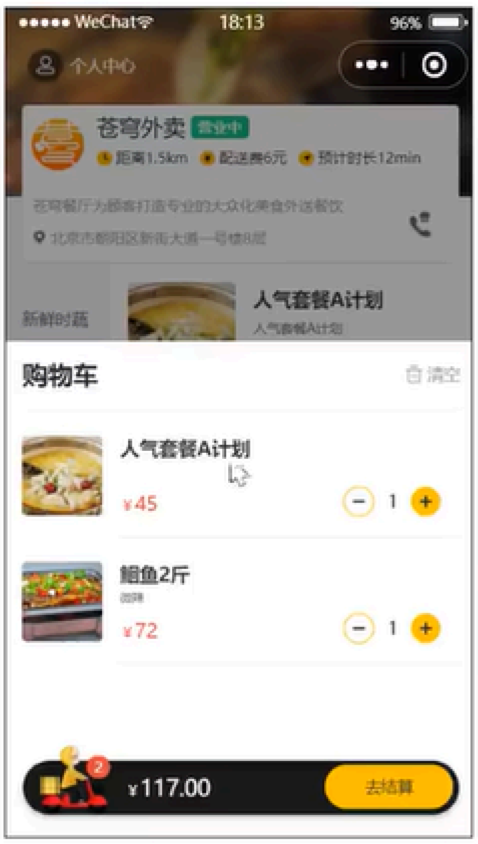
```java
package com.sky.mapper;

import com.sky.entity.ShoppingCart;
import com.sky.entity.User;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;

import java.util.List;

@Mapper
public interface ShoppingCartMapper {

    /**
     * 动态条件查询
     * @param shoppingCart
     * @return
     */
    List<ShoppingCart> list(ShoppingCart shoppingCart);

    /**
     * 根据id修改商品数量
     * @param shoppingCart
     */
    @Update("update shopping_cart set number= #{number} where id=#{id}")
    void updateNumberById(ShoppingCart shoppingCart);

    /**
     * 插入购物车数据
     * @param shoppingCart
     */
    @Insert("insert into shopping_cart (user_id, dish_id, setmeal_id, name, image, amount, number, dish_flavor, create_time) "
            +
            "values " +
            "(#{userId}, #{dishId}, #{setmealId}, #{name}, #{image}, #{amount}, #{number}, #{dishFlavor}, #{createTime})")
    void insert(ShoppingCart shoppingCart);
}

```

sky-server/src/main/resources/mapper/ShoppingCartMapper.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.ShoppingCartMapper">



    <select id="list" resultType="com.sky.entity.ShoppingCart">
        select * from shopping_cart
        <where>
            <if test="userId != null">
                and user_id = #{userId}
            </if>
            <if test="setmealId != null">
                and setmeal_id = #{setmealId}
            </if>
            <if test="dishId != null">
                and dish_id = #{dishId}
            </if>
            <if test="dishFlavor != null">
                and dish_flavor = #{dishFlavor}
            </if>
        </where>
    </select>
</mapper>
```

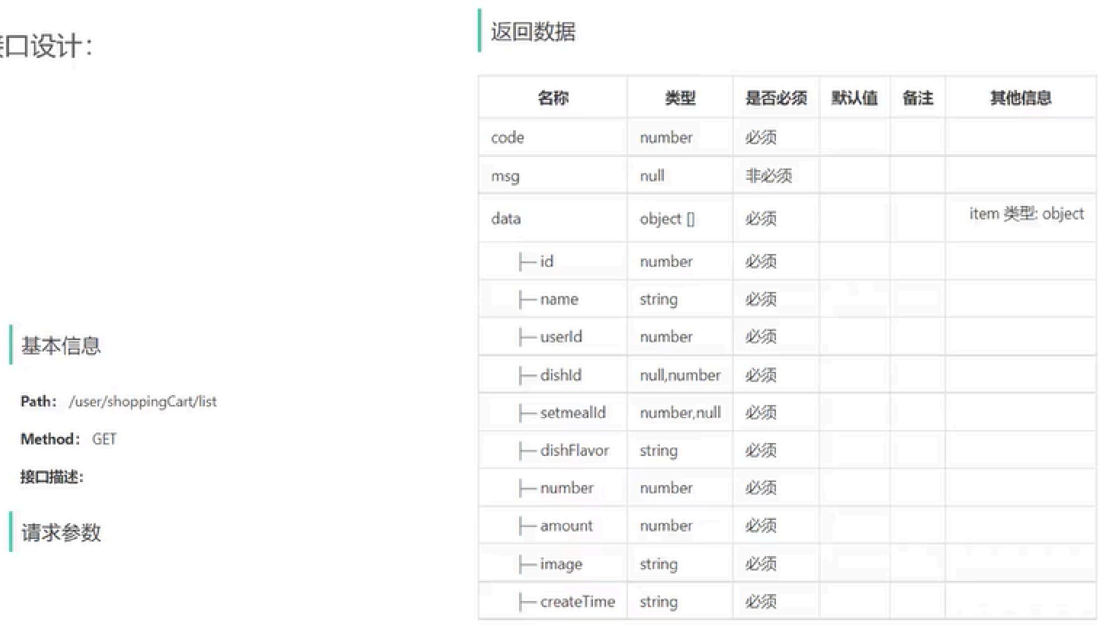
6. 查看购物车

产品原型:



需求分析和设计

接口设计:



基本信息

Path: /user/shoppingCart/list

Method: GET

接口描述:

请求参数

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			
msg	null	非必须			
data	object []	必须			item 类型: object
├─ id	number	必须			
├─ name	string	必须			
├─ userId	number	必须			
├─ dishId	null,number	必须			
├─ setmealId	number,null	必须			
├─ dishFlavor	string	必须			
├─ number	number	必须			
├─ amount	number	必须			
├─ image	string	必须			
├─ createTime	string	必须			

ShoppingCartController.java

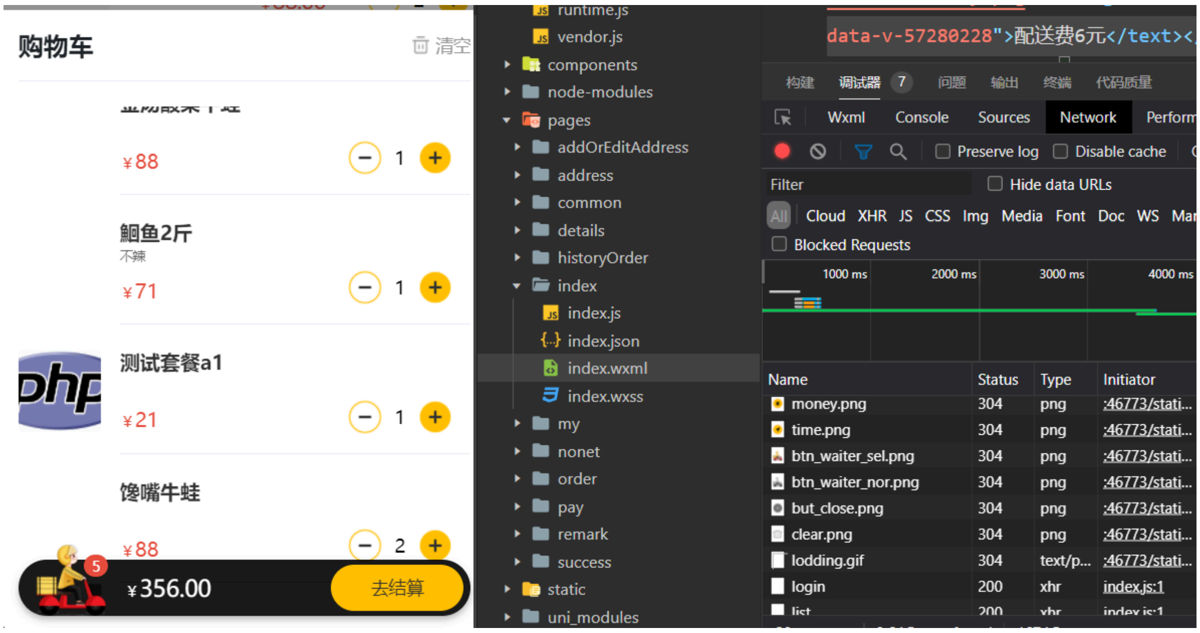
```java
    /**
     * 查看购物车
     * @return
     */
    @GetMapping("/list")
    @ApiOperation("查看购物车")
    public Result<List<ShoppingCart>> list(){
        List<ShoppingCart> list= shoppingCartService.showShoppingCart();
        return Result.success(list);
    }
```

ShoppingCartService.java

```java
    /**
     * 查看购物车
     * @return
     */
    List<ShoppingCart> showShoppingCart();
```

ShoppingCartServiceImpl.java

```java
    /**
     * 查看购物车
     * @return
     */
    public List<ShoppingCart> showShoppingCart() {
        //获取当前微信id
        Long userID = BaseContext.getCurrentId();
        ShoppingCart shoppingCart=ShoppingCart.builder()
                .userId(userID)
                .build();
        List<ShoppingCart> list=shoppingCartMapper.list(shoppingCart);
        return list;
    }
```


购物车 清空

¥88 1

鲷鱼2斤
不辣
¥71 1

测试套餐a1
¥21 1

馋嘴牛蛙
¥88 2

¥356.00 去结算

```
runtime.js
vendor.js
▸ components
▸ node-modules
▾ pages
  ▸ addOrEditAddress
  ▸ address
  ▸ common
  ▸ details
  ▸ historyOrder
  ▾ index
      index.js
      index.json
      index.wxml
      index.wxss
  ▸ my
  ▸ nonet
  ▸ order
  ▸ pay
  ▸ remark
  ▸ success
▸ static
▸ uni_modules
```

```
data-v-57280228">配送费6元</text><
```

构建 调试器 7 问题 输出 终端 代码质量

Wxml Console Sources Network Perform

Preserve log Disable cache

Filter Hide data URLs

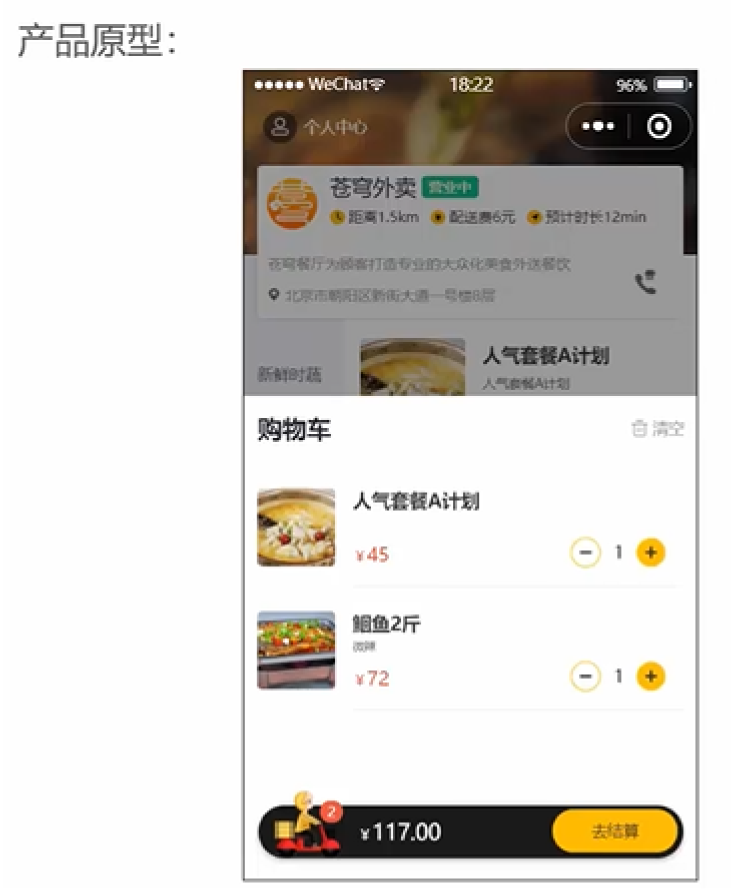
All Cloud XHR JS CSS Img Media Font Doc WS Mar

Blocked Requests

1000 ms 2000 ms 3000 ms 4000 ms

Name	Status	Type	Initiator
money.png	304	png	:46773/stati...
time.png	304	png	:46773/stati...
btn_waiter_sel.png	304	png	:46773/stati...
btn_waiter_nor.png	304	png	:46773/stati...
but_close.png	304	png	:46773/stati...
clear.png	304	png	:46773/stati...
lodding.gif	304	text/p...	:46773/stati...
login	200	xhr	index.js:1
list	200	xhr	index.js:1

7. 清空购物车


产品原型：

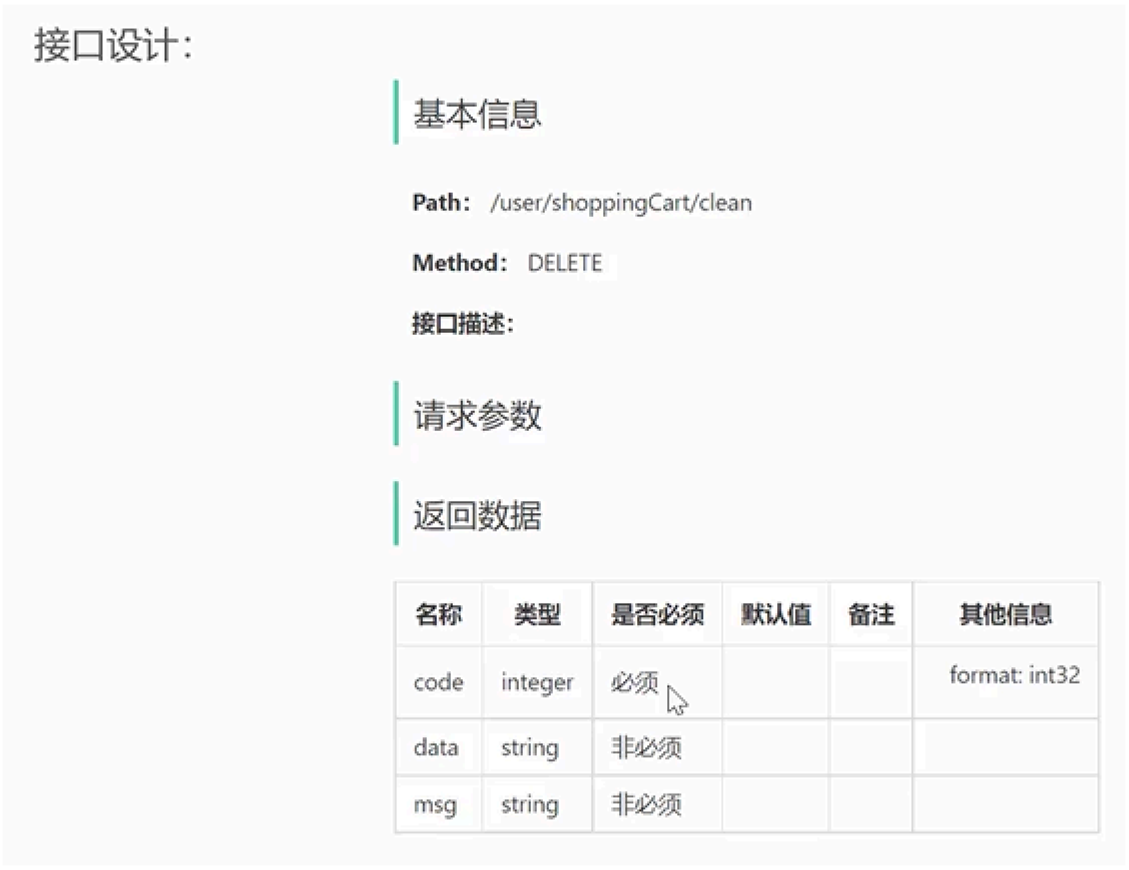
WeChat 18:22 96%
个人中心
苍穹外卖 营业中
距离1.5km 配送费6元 预计时长12min
苍穹餐厅为顾客打造专业的大众化美食外送餐饮
北京市朝阳区新街大道一号楼8层
新鲜时蔬
人气套餐A计划
人气套餐A计划

购物车 清空

人气套餐A计划
¥45 1

鲷鱼2斤
微辣
¥72 1

¥117.00 去结算



接口设计：

基本信息

Path： /user/shoppingCart/clean

Method： DELETE

接口描述：

请求参数

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	string	非必须			
msg	string	非必须			

ShoppingCartController.java

```java
    /**
     * 清空购物车
     * @return
     */
    @DeleteMapping("/clean")
    @ApiOperation("清空购物车")
    public Result clean(){
        shoppingCartService.cleanShoppingCart();
        return Result.success();
    }
```

ShoppingCartService.java

```java
    /**
     * 清空购物车
     */
    void cleanShoppingCart();
```

ShoppingCartServiceImpl.java

```java
    /**
     * 清空购物车
     */
    public void cleanShoppingCart() {
        Long userID = BaseContext.getCurrentId();
        shoppingCartMapper.deleteByUserId(userID);
    }
```

ShoppingCartMapper.java

```java
    /**
     * 根据用户id清空购物车
     * @param userID
     */
    @Delete("delete  from shopping_cart where user_id=#{userId}")
    void deleteByUserId(Long userID);
```

8. 删除购物车

基本信息

Path： /user/shoppingCart/sub

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	其他信息
dishId	number	非必须		菜品id	
dishFlavor	string	非必须		口味	
setmealId	number	非必须		套餐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			
msg	string	非必须			
data	string	非必须			

ShoppingCartController.java

```java
    /**
     * 减少或删除购物车某一商品
     * @param shoppingCartDTO
     * @return
     */
    @PostMapping("/sub")
    @ApiOperation("减少或删除购物车某一商品")
    public Result sub(@RequestBody ShoppingCartDTO shoppingCartDTO){
        log.info("减少或删除购物车某一商品：{}", shoppingCartDTO);
        shoppingCartService.subShoppingCart(shoppingCartDTO);
        return Result.success();
    }
```

ShoppingCartService.java

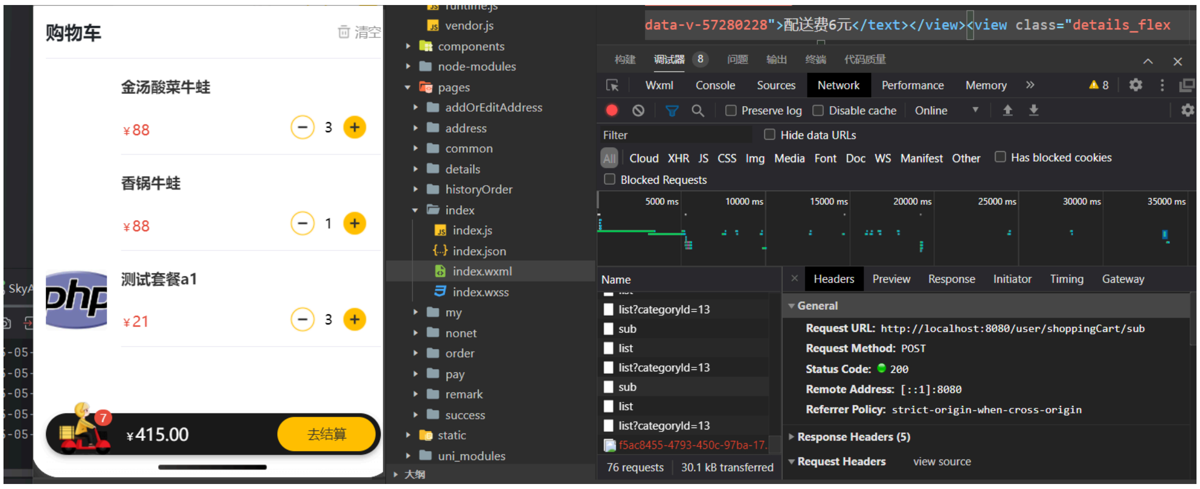
```java
    /**
     * 减少或删除购物车某一商品
     * @param shoppingCartDTO
     */
    void subShoppingCart(ShoppingCartDTO shoppingCartDTO);
```

ShoppingCartServiceImpl.java

```java
    /**
     * 减少或删除购物车某一商品
     * @param shoppingCartDTO
     */
    @Override
    public void subShoppingCart(ShoppingCartDTO shoppingCartDTO) {

        ShoppingCart shoppingCart = new ShoppingCart();
        BeanUtils.copyProperties(shoppingCartDTO, shoppingCart);
        Long userId = BaseContext.getCurrentId();
        shoppingCart.setUserId(userId);
        List<ShoppingCart> list = shoppingCartMapper.list(shoppingCart);
        if (list != null && list.size()>0) {
            ShoppingCart cart=list.get(0);

            if (cart.getNumber()==1) {//删除菜品或套餐
                shoppingCartMapper.delete(cart);
            }else{//修改菜品或套餐
                cart.setNumber(cart.getNumber()-1);
                shoppingCartMapper.updateNumberById(cart);
            }
        }

    }
```

ShoppingCartMapper.java

```java
    /**
     * 删除购物车某一商品
     * @param shoppingCart
     */
    @Delete("delete from shopping_cart where id=#{id}")
    void delete(ShoppingCart shoppingCart);
```



```
data-v-57280228">配送费6元</text></view><view class="details_flex
```

General
Request URL: http://localhost:8080/user/shoppingCart/sub
Request Method: POST
Status Code: 200
Remote Address: [::1]:8080
Referrer Policy: strict-origin-when-cross-origin
Response Headers (5)
Request Headers   view source

如果前端有问题，可用以下方法获取打包前的vue文件

Webpack打包Vue后的JS文件还原主要依赖源码映射文件（Source Map）和反编译工具

```
bashCopy Code# 安装反编译工具reverse-sourcemap
npm install -g reverse-sourcemap

bashCopy Code# 执行反编译生成原始目录结构
reverse-sourcemap --output-dir ./source-code app.xxxx.js.map
```