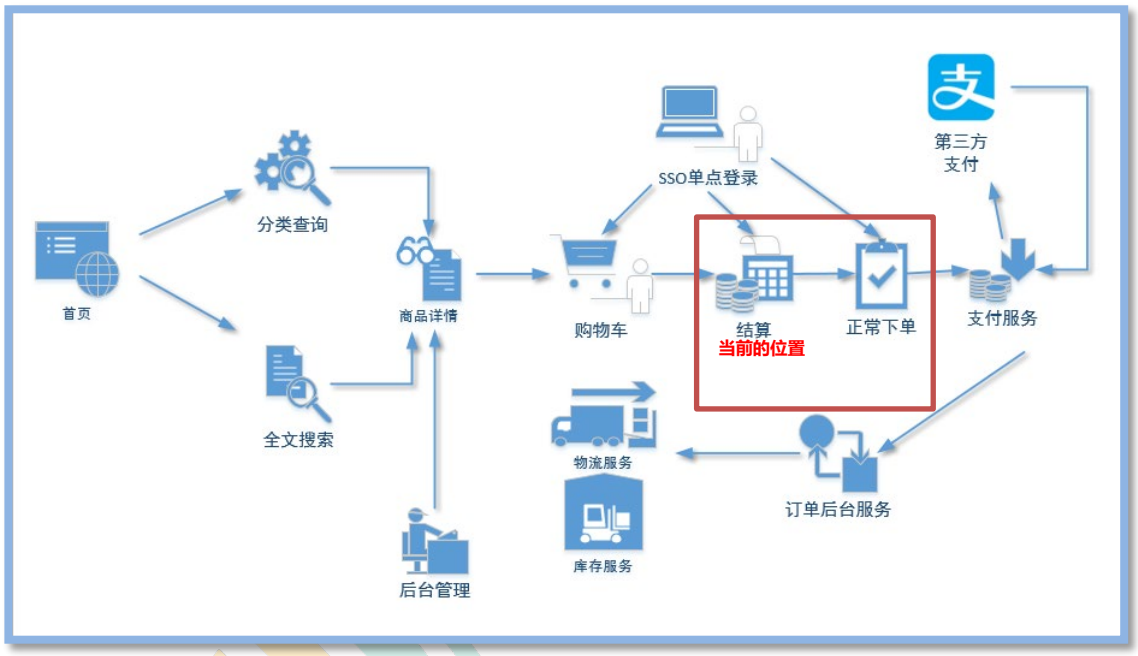
尚品汇商城

一、业务介绍

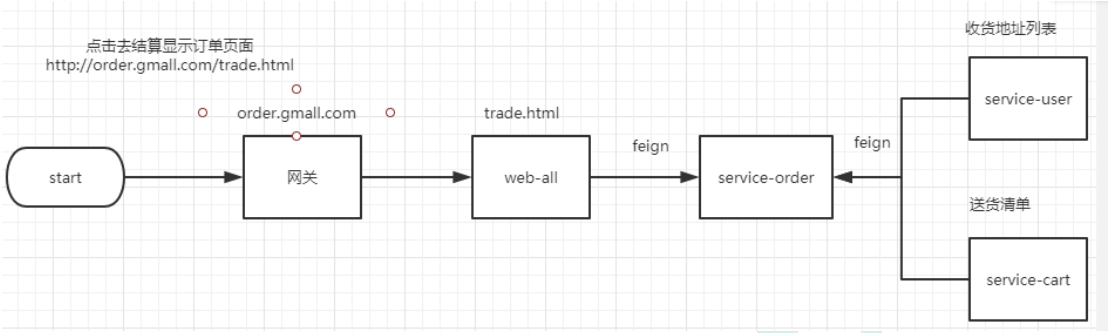


订单业务在整个电商平台中处于核心位置，也是比较复杂的一块业务。是把“物”变为“钱”的一个中转站。

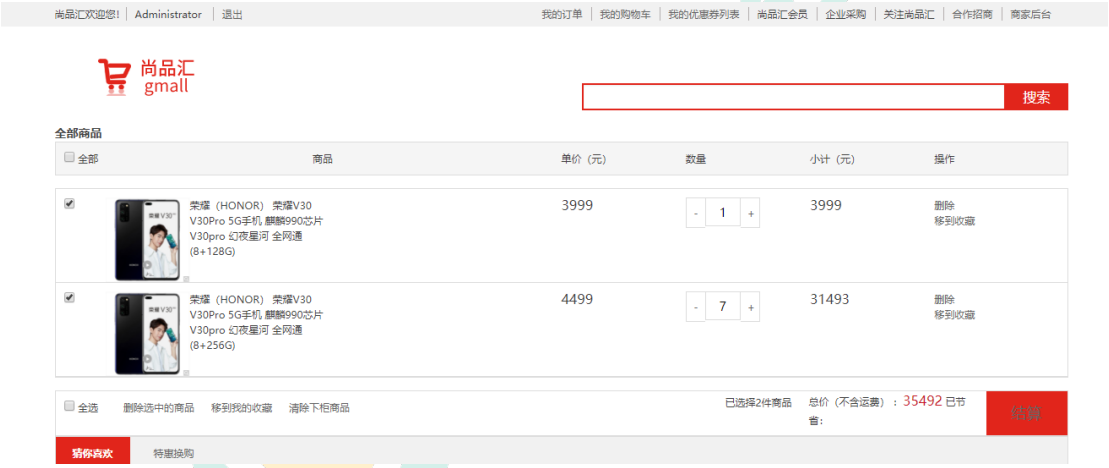
整个订单模块一共分四部分组成：

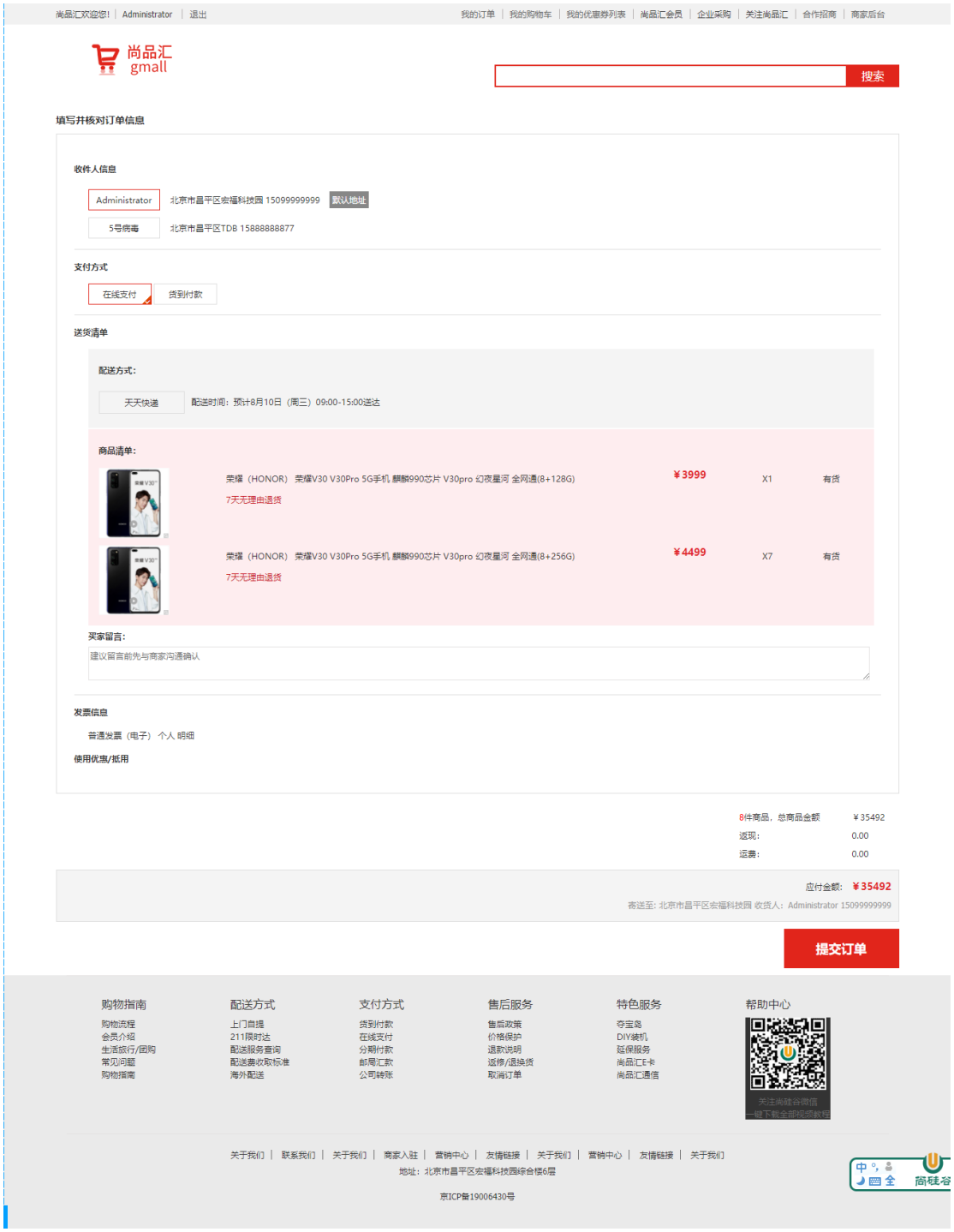
1. 结算
2. 下单
3. 对接支付服务
4. 对接库存管理系统

二、结算页



入口：购物车点击计算按钮，结算必须要登录！





分析页面需要的数据：

1、需要用户地址信息

2、购物车中选择的商品列表

3、用户地址信息在 service-user 模块，购物车信息在 service-cart 模块，所以我们要在相应模块提供 api 接口，通过 feign client 调用获取数据

2.1 在 service-user 模块获取地址列表

2.1.1 添加 mapper

```
package com.atguigu.gmall.user.mapper;


@Mapper
public interface UserAddressMapper extends BaseMapper<UserAddress> {
}
```

2.1.2 编写接口与实现类

```
package com.atguigu.gmall.user.service;


public interface UserAddressService {

    /**
     * 根据用户Id 查询用户的收货地址列表!
     * @param userId
     * @return
     */
    List<UserAddress> findUserAddressListByUserId(String userId);
}
```

实现类

```
package com.atguigu.gmall.user.service.impl;


@Service
public class UserAddressServiceImpl implements UserAddressService {

    @Autowired
    private UserAddressMapper userAddressMapper;

    @Override
    public    List<UserAddress>    findUserAddressListByUserId(String
userId) {
        // 操作哪个数据库表，则就使用哪个表对应的mapper!
    // new Example() ; 你操作的哪个表，则对应的传入表的实体类!
    // select * from userAddress where userId = ? ;
        QueryWrapper<UserAddress>       queryWrapper       =       new
QueryWrapper<>();
        queryWrapper.eq("user_id", userId);
```

```
        List<UserAddress>              userAddressList              =
userAddressMapper.selectList(queryWrapper);
        return userAddressList;
    }
}
```

2.1.3 编写控制器

Controller

```
package com.atguigu.gmall.user.controller;

@RestController
@RequestMapping("/api/user")
public class UserApiController {

    @Autowired
    private UserAddressService userAddressService;

    /**
     * 获取用户地址
   * @param userId
     * @return
     */
    @GetMapping("inner/findUserAddressListByUserId/{userId}")
    public                                          List<UserAddress>
findUserAddressListByUserId(@PathVariable("userId") String userId){
        return
userAddressService.findUserAddressListByUserId(userId);
    }

}
```

2.1.3 在 service-user-client 暴露接口

接口类

```
package com.atguigu.gmall.user.client;


@FeignClient(value = "service-user", fallback = UserDegradeFeignClient.class)
public interface UserFeignClient {


@GetMapping("/api/user/inner/findUserAddressListByUserId/{userId}")
    List<UserAddress> findUserAddressListByUserId(@PathVariable(value = "userId") String userId);
}
```

```
package com.atguigu.gmall.user.client.impl;


@Component
public class UserDegradeFeignClient implements UserFeignClient {


    @Override
    public List<UserAddress> findUserAddressListByUserId(String userId) {
        return null;
    }
}
```

2.2 在 service-cart 模块获取选中商品数据

2.2.1 添加接口与实现类

```
CartService 接口

/**
 * 根据用户 Id 查询购物车列表
 *
 * @param userId
```

```
 * @return
 */
List<CartInfo> getCartCheckedList(String userId);
```

实现类

```
@Override
public List<CartInfo> getCartCheckedList(String userId) {
    //  获取的选中的购物车列表!
  String cartKey = this.getCartKey(userId);
    //  获取到购物车集合数据:
  List<CartInfo>                                cartInfoList                =
this.redisTemplate.opsForHash().values(cartKey);
    List<CartInfo> cartInfos = cartInfoList.stream().filter(cartInfo -> {
        //  再次确认一下最新价格

cartInfo.setSkuPrice(productFeignClient.getSkuPrice(cartInfo.getSkuId()));
        return cartInfo.getIsChecked().intValue() == 1;
    }).collect(Collectors.toList());

    //  返回数据
  return cartInfos;
}
```

2.2.2 编写控制器

```
/**
 * 根据用户Id 查询购物车列表
 *
 * @param userId
 * @return
 */
@GetMapping("getCartCheckedList/{userId}")
public   List<CartInfo>   getCartCheckedList(@PathVariable(value   =
"userId") String userId) {
    return cartService.getCartCheckedList(userId);
}
```

2.2.3 创建 service-cart-client

修改配置 pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>com.atguigu.gmall</groupId>
      <artifactId>service-client</artifactId>
      <version>1.0</version>
   </parent>

   <artifactId>service-cart-client</artifactId>
   <version>1.0</version>

   <packaging>jar</packaging>
   <name>service-cart-client</name>
   <description>service-cart-client</description>

</project>
```

```java
package com.atguigu.gmall.cart.client;



@FeignClient(value = "service-cart",fallback = CartDegradeFeignClient.class)
public interface CartFeignClient {
    //  获取选中购物车列表！
  @GetMapping("/api/cart/getCartCheckedList/{userId}")
    public List<CartInfo> getCartCheckedList(@PathVariable String userId);
}
```

```java
package com.atguigu.gmall.cart.client.impl;

@Component
public class CartDegradeFeignClient implements CartFeignClient {

    @Override
    public List<CartInfo> getCartCheckedList(String userId) {
        return null;
    }
}
```

2.3 搭建 service-order 模块

2.3.1 搭建 service-order

搭建方式如 service-cart

2.3.2 修改 pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>com.atguigu.gmall</groupId>
      <artifactId>service</artifactId>
      <version>1.0</version>
   </parent>

   <version>1.0</version>
   <artifactId>service-order</artifactId>
   <packaging>jar</packaging>
   <name>service-order</name>
   <description>service-order</description>

   <dependencies>
      <dependency>
         <groupId>com.atguigu.gmall</groupId>
         <artifactId>service-product-client</artifactId>
         <version>1.0</version>
      </dependency>
      <dependency>
         <groupId>com.atguigu.gmall</groupId>
         <artifactId>service-cart-client</artifactId>
         <version>1.0</version>
      </dependency>
      <dependency>
         <groupId>com.atguigu.gmall</groupId>
         <artifactId>service-user-client</artifactId>
         <version>1.0</version>
      </dependency>
   </dependencies>

   <build>
      <finalName>service-order</finalName>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

2.3.3 添加配置

bootstrap.properties

```
spring.application.name=service-order
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
spring.cloud.nacos.config.shared-configs[0].data-id=common.yaml
```

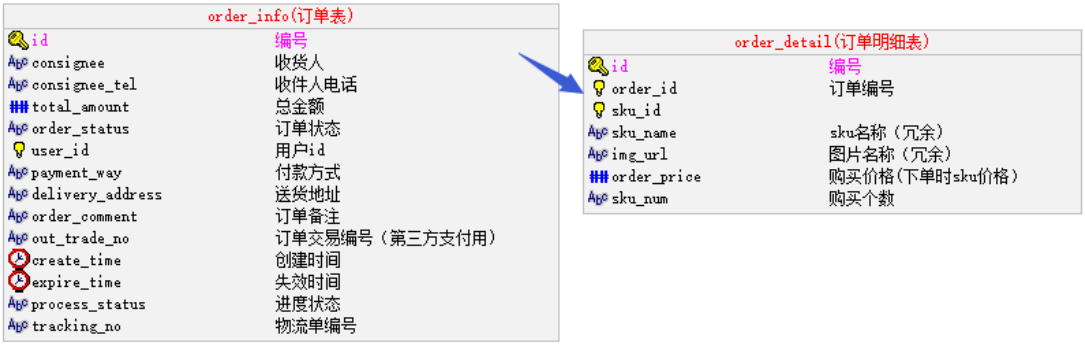
启动类

```
package com.atguigu.gmall.order;

@SpringBootApplication
@ComponentScan(basePackages = "com.atguigu.gmall")
@EnableDiscoveryClient
@EnableFeignClients(basePackages = "com.atguigu.gmall")
public class ServiceOrderApplication {
    public static void main(String[] args) {

        SpringApplication.run(ServiceOrderApplication.class,args);
    }
}
```

2.3.4 订单的数据结构

orderInfo：订单表

orderDetail ： 订单明细



id	主键。自动生成
consignee	收货人名称。页面获取
consignee_tel	收货人电话。页面获取
deliveryAddress	收货地址。页面获取
total_amount	总金额。计算
order_status	订单状态，用于显示给用户查看。设定初始值。
userId	用户 Id。从拦截器已放到请求属性中。
payment_way	支付方式（网上支付、货到付款）。页面获取
orderComment	订单状态。页面获取
out_trade_no	第三方支付编号。按规则生成
create_time	创建时间。设当前时间
expire_time	默认当前时间+1 天
process_status	订单进度状态，程序控制、 后台管理查看。设定初始值，
tracking_no	物流编号,初始为空，发货后补充
parent_order_id	拆单时产生，默认为空

id	主键，自动生成

order_id	订单编号，主表保存后给从表
sku_id	商品 id 页面传递
sku_name	商品名称，后台添加
img_url	图片路径，后台添加
order_price	商品单价，从页面中获取，并验价。
sku_num	商品个数，从页面中获取

添加实体

```
package com.atguigu.gmall.model.order;


@Data
@ApiModel(description = "订单信息")
@TableName("order_info")
public class OrderInfo extends BaseEntity {

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "收货人")
    @TableField("consignee")
    private String consignee;

    @ApiModelProperty(value = "收件人电话")
    @TableField("consignee_tel")
    private String consigneeTel;

    @ApiModelProperty(value = "总金额")
    @TableField("total_amount")
    private BigDecimal totalAmount;

    @ApiModelProperty(value = "订单状态")
    @TableField("order_status")
    private String orderStatus;

    @ApiModelProperty(value = "用户 id")
    @TableField("user_id")
    private Long userId;

    @ApiModelProperty(value = "付款方式")
    @TableField("payment_way")
    private String paymentWay;

    @ApiModelProperty(value = "送货地址")
```

```
    @TableField("delivery_address")
    private String deliveryAddress;

    @ApiModelProperty(value = "订单备注")
    @TableField("order_comment")
    private String orderComment;

    @ApiModelProperty(value = "订单交易编号（第三方支付用)")
    @TableField("out_trade_no")
    private String outTradeNo;

    @ApiModelProperty(value = "订单描述(第三方支付用)")
    @TableField("trade_body")
    private String tradeBody;

    @ApiModelProperty(value = "创建时间")
    @TableField("create_time")

   @JsonFormat(locale="zh",   timezone="GMT+8",   pattern="yyyy-MM-dd
HH:mm:ss")

    private Date createTime;

    @ApiModelProperty(value = "失效时间")
    @TableField("expire_time")

   @JsonFormat(locale="zh",   timezone="GMT+8",   pattern="yyyy-MM-dd
HH:mm:ss")

    private Date expireTime;

    @ApiModelProperty(value = "进度状态")
    @TableField("process_status")
    private String processStatus;

    @ApiModelProperty(value = "物流单编号")
    @TableField("tracking_no")
    private String trackingNo;

    @ApiModelProperty(value = "父订单编号")
    @TableField("parent_order_id")
    private Long parentOrderId;

    @ApiModelProperty(value = "图片路径")
    @TableField("img_url")
    private String imgUrl;

    @TableField(exist = false)
    private List<OrderDetail> orderDetailList;

    @TableField(exist = false)
    private String wareId;
```

```
    // 计算总价格
  public void sumTotalAmount(){
        BigDecimal totalAmount=new BigDecimal("0");
        for (OrderDetail orderDetail : orderDetailList) {
           totalAmount=
totalAmount.add(orderDetail.getOrderPrice().multiply(new
BigDecimal(orderDetail.getSkuNum())));
        }
        this.totalAmount=  totalAmount;
    }

}
```

```
package com.atguigu.gmall.model.order;

@Data
@ApiModel(description = "订单明细")
@TableName("order_detail")
public class OrderDetail extends BaseEntity {

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "订单编号")
    @TableField("order_id")
    private Long orderId;

    @ApiModelProperty(value = "sku_id")
    @TableField("sku_id")
    private Long skuId;

    @ApiModelProperty(value = "sku 名称（冗余)")
    @TableField("sku_name")
    private String skuName;

    @ApiModelProperty(value = "图片名称（冗余)")
    @TableField("img_url")
    private String imgUrl;

    @ApiModelProperty(value = "购买价格(下单时 sku 价格）")
    @TableField("order_price")
    private BigDecimal orderPrice;

    @ApiModelProperty(value = "购买个数")
    @TableField("sku_num")
    private Integer skuNum;

    // 是否有足够的库存！
    @TableField(exist = false)
    private String hasStock;

}
```

其中 hasStock 是一个非持久化属性，用户传递【是否还有库存】的标志。

如果商品在库存中有足够数据，suceess = “1”，fail= “0”

2.3.5 接口封装 OrderApiController

```
package com.atguigu.gmall.order.controller;

@RestController
@RequestMapping("api/order")
public class OrderApiController {

    @Autowired
    private UserFeignClient userFeignClient;

    @Autowired
    private CartFeignClient cartFeignClient;


    /**
     * 确认订单
   * @param request
     * @return
     */
    @GetMapping("auth/trade")
    public   Result<Map<String,   Object>>   trade(HttpServletRequest
request) {
        // 获取到用户Id
        String userId = AuthContextHolder.getUserId(request);

        //获取用户地址
    List<UserAddress>                    userAddressList                =
userFeignClient.findUserAddressListByUserId(userId);

        // 渲染送货清单
    // 先得到用户想要购买的商品！
    List<CartInfo>                       cartInfoList                   =
cartFeignClient.getCartCheckedList(userId);
        // 声明一个集合来存储订单明细
    ArrayList<OrderDetail> detailArrayList = new ArrayList<>();
        for (CartInfo cartInfo : cartInfoList) {
            OrderDetail orderDetail = new OrderDetail();
            orderDetail.setSkuId(cartInfo.getSkuId());
            orderDetail.setSkuName(cartInfo.getSkuName());
            orderDetail.setImgUrl(cartInfo.getImgUrl());
            orderDetail.setSkuNum(cartInfo.getSkuNum());
            orderDetail.setOrderPrice(cartInfo.getSkuPrice());

            // 添加到集合
```

```
        detailArrayList.add(orderDetail);
          }
          // 计算总金额
    OrderInfo orderInfo = new OrderInfo();
        orderInfo.setOrderDetailList(detailArrayList);
        orderInfo.sumTotalAmount();

        Map<String, Object> result = new HashMap<>();
        result.put("userAddressList", userAddressList);
        result.put("detailArrayList", detailArrayList);
        // 保存总金额
        result.put("totalNum", detailArrayList.size());
        result.put("totalAmount", orderInfo.getTotalAmount());

        return Result.ok(result);
    }

}
```

说明：接口已经封装，接下来暴露接口，提供给 web-all 模块前端展示数据

2.4 结算页面

2.4.1 搭建 service-order-client 模块

1，搭建过程同 service-cart-client

2，pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>com.atguigu.gmall</groupId>
      <artifactId>service-client</artifactId>
      <version>1.0</version>
   </parent>

   <version>1.0</version>
   <artifactId>service-order-client</artifactId>
   <packaging>jar</packaging>
   <name>service-order-client</name>
   <description>service-order-client</description>
```

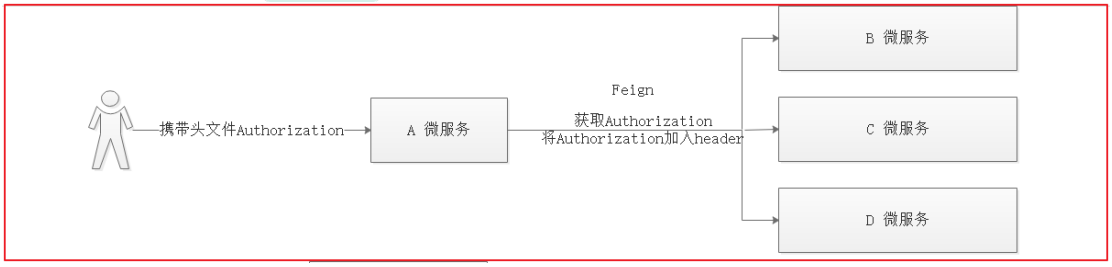
```
</project>
```

2.4.2 在 service-order-client 暴露接口

```
package com.atguigu.gmall.order.client;


@FeignClient(value = "service-order", fallback =
OrderDegradeFeignClient.class)
public interface OrderFeignClient {
    @GetMapping("/api/order/auth/trade")
    Result<Map<String, Object>> trade();
}
```

```
package com.atguigu.gmall.order.client.impl;


@Component
public class OrderDegradeFeignClient implements OrderFeignClient {

    @Override
    public Result<Map<String, Object>> trade() {
        return Result.fail();
    }
}
```

微服务之间用户信息传递



如上图：因为微服务之间**并没有传递头文件**，所以我们可以定义一个拦截器，每次微服务调用之前都先检查下头文件，将请求的头文件中的用户信息再放入到 header 中，再调用其他微服务即可。

在 web-util 中添加拦截器

```
package com.atguigu.gmall.common.interceptor;

import javax.servlet.http.HttpServletRequest;

@Component
public class FeignInterceptor implements RequestInterceptor {

    public void apply(RequestTemplate requestTemplate){
        ServletRequestAttributes            attributes            =
(ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = attributes.getRequest();

        requestTemplate.header("userTempId",
request.getHeader("userTempId"));
        requestTemplate.header("userId",
request.getHeader("userId"));
    }
}
```

2.4.3 配置网关

```
- id: web-order
  uri: lb://web-all
  predicates:
  - Host=order.gmall.com
- id: service-order
  uri: lb://service-order
  predicates:
  - Path=/*/order/**
```

2.4.4 在 web-all 模块中添加依赖

```
      <dependency>
          <groupId>com.atguigu.gmall</groupId>
          <artifactId>service-order-client</artifactId>
          <version>1.0</version>
      </dependency>
```

2.4.5 在 web-all 中添加控制器

```
package com.atguigu.gmall.all.controller;


@Controller
public class OrderController {

    @Autowired
    private OrderFeignClient orderFeignClient;

    /**
     * 确认订单
   * @param model
     * @return
     */
    @GetMapping("trade.html")
    public String trade(Model model) {
        Result<Map<String,         Object>>         result         =
orderFeignClient.trade();

        model.addAllAttributes(result.getData());
        return "order/trade";
    }
}
```

三、下订单

3.1 下单功能分析：

1. 保存单据前要做记录：验库存，验价格
2. 保存单据: orderInfo orderDetail。
3. 保存以后把购物车中的商品删除。{不删！}
4. 重定向到支付页面。

3.2 添加 mapper

Mapper

```
package com.atguigu.gmall.order.mapper;
```

```
@Mapper
public interface OrderInfoMapper extends BaseMapper<OrderInfo> {

}
```

```
package com.atguigu.gmall.order.mapper;


@Mapper
public interface OrderDetailMapper extends BaseMapper<OrderDetail> {
}
```

3.3 添加接口与实现类

```
package com.atguigu.gmall.order.service;

public interface OrderService extends IService<OrderInfo> {

    /**
     * 保存订单
   * @param orderInfo
     * @return
     */
    Long saveOrderInfo(OrderInfo orderInfo);

}
```

实现类

```
package com.atguigu.gmall.order.service.impl;


@Service
public class OrderServiceImpl extends ServiceImpl<OrderInfoMapper,
OrderInfo> implements OrderService {

    @Autowired
    private OrderInfoMapper orderInfoMapper;

    @Autowired
    private OrderDetailMapper orderDetailMapper;


@Override
@Transactional
public Long saveOrderInfo(OrderInfo orderInfo) {
    orderInfo.sumTotalAmount();
    orderInfo.setOrderStatus(OrderStatus.UNPAID.name());
```

```
    String outTradeNo = "ATGUIGU" + System.currentTimeMillis() + "" + new
Random().nextInt(1000);
    orderInfo.setOutTradeNo(outTradeNo);
    orderInfo.setCreateTime(new Date());
    // 定义为1天
  Calendar calendar = Calendar.getInstance();
    calendar.add(Calendar.DATE, 1);
    orderInfo.setExpireTime(calendar.getTime());

    orderInfo.setProcessStatus(ProcessStatus.UNPAID.name());
    // 获取订单明细
   List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
    StringBuffer tradeBody = new StringBuffer();
    for (OrderDetail orderDetail : orderDetailList) {
        tradeBody.append(orderDetail.getSkuName()+" ");
    }
    if (tradeBody.toString().length()>100){
        orderInfo.setTradeBody(tradeBody.toString().substring(0,100));
    }else {
        orderInfo.setTradeBody(tradeBody.toString());
    }


    orderInfoMapper.insert(orderInfo);

    for (OrderDetail orderDetail : orderDetailList) {
        orderDetail.setOrderId(orderInfo.getId());
        orderDetailMapper.insert(orderDetail);
    }
    return orderInfo.getId();
  }
}
```

3.4 编写控制器

```
@Autowired
private OrderService orderService;

    /**
     * 提交订单
   * @param orderInfo
     * @param request
     * @return
     */
    @PostMapping("auth/submitOrder")
    public  Result  submitOrder(@RequestBody  OrderInfo  orderInfo,
HttpServletRequest request) {
        // 获取到用户Id
        String userId = AuthContextHolder.getUserId(request);
        orderInfo.setUserId(Long.parseLong(userId));

        // 验证通过，保存订单！
    Long orderId = orderService.saveOrderInfo(orderInfo);
```

```
        return Result.ok(orderId);
    }
```

3.5 如何解决用户利用浏览器回退重复提交订单？

在进入结算页面时，生成一个结算流水号，然后保存到结算页面的隐藏元素中，每次用户提交都检查该流水号与页面提交的是否相符，订单保存以后把后台的流水号删除掉。那么第二次用户用同一个页面提交的话流水号就会匹配失败，无法重复保存订单。

3.5.1 修改结算页增加流水号的生成。

OrderService 接口

```
/**
 * 生产流水号
 * @param userId
 * @return
 */
String getTradeNo(String userId);

/**
 * 比较流水号
 * @param userId 获取缓存中的流水号
 * @param tradeCodeNo    页面传递过来的流水号
 * @return
 */
boolean checkTradeCode(String userId, String tradeCodeNo);


/**
 * 删除流水号
 * @param userId
 */
void deleteTradeNo(String userId);
```

实现类

```
@Autowired
private RedisTemplate redisTemplate;



@Override
public String getTradeNo(String userId) {
```

```
    // 定义key
    String tradeNoKey = "user:" + userId + ":tradeCode";
    // 定义一个流水号
  String tradeNo = UUID.randomUUID().toString().replace("-", "");
    redisTemplate.opsForValue().set(tradeNoKey, tradeNo);
    return tradeNo;
}

@Override
public boolean checkTradeCode(String userId, String tradeCodeNo) {
    // 定义key
    String tradeNoKey = "user:" + userId + ":tradeCode";
    String             redisTradeNo             =             (String)
redisTemplate.opsForValue().get(tradeNoKey);
    return tradeCodeNo.equals(redisTradeNo);
}

@Override
public void deleteTradeNo(String userId) {
    // 定义key
    String tradeNoKey = "user:" + userId + ":tradeCode";
    // 删除数据
  redisTemplate.delete(tradeNoKey);
}
```

3.5.2 在 OrderController 类 trade 方法添加交易流水号

```
// 获取流水号
String tradeNo = orderService.getTradeNo(userId);
result.put("tradeNo", tr       adeNo);
```

3.5.3 在 OrderApiController 控制器中实现

```
// 获取前台页面的流水号
String tradeNo = request.getParameter("tradeNo");

// 调用服务层的比较方法
boolean flag = orderService.checkTradeCode(userId, tradeNo);
if (!flag) {
    // 比较失败!
  return Result.fail().message("不能重复提交订单！");
}

//  删除流水号
```

```
orderService.deleteTradeNo(userId);
```

3.6 验库存与验证价格

通过 restful 接口查询商品是否有库存

一般电商系统的商品库存，都不由电商系统本身来管理，由另外一套仓库管理系统，或者进销存系统来管理，电商系统通过第三方接口调用该系统。

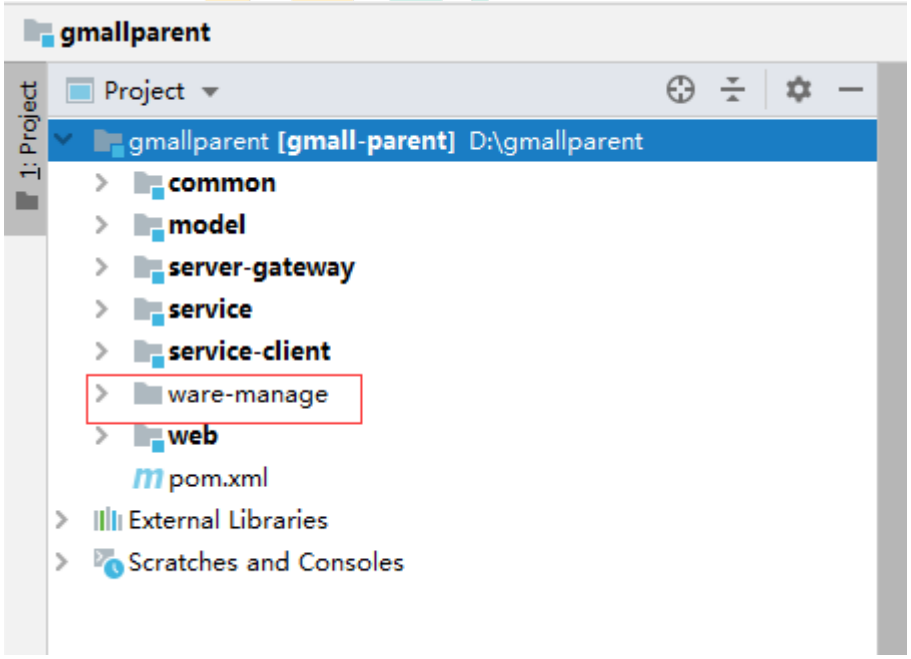
由于库管系统可能是异构的系统，所以不在微服务体系之内。只支持 restful 风格的 webservice 调用和消息队列的调用。

详见《库存管理系统》

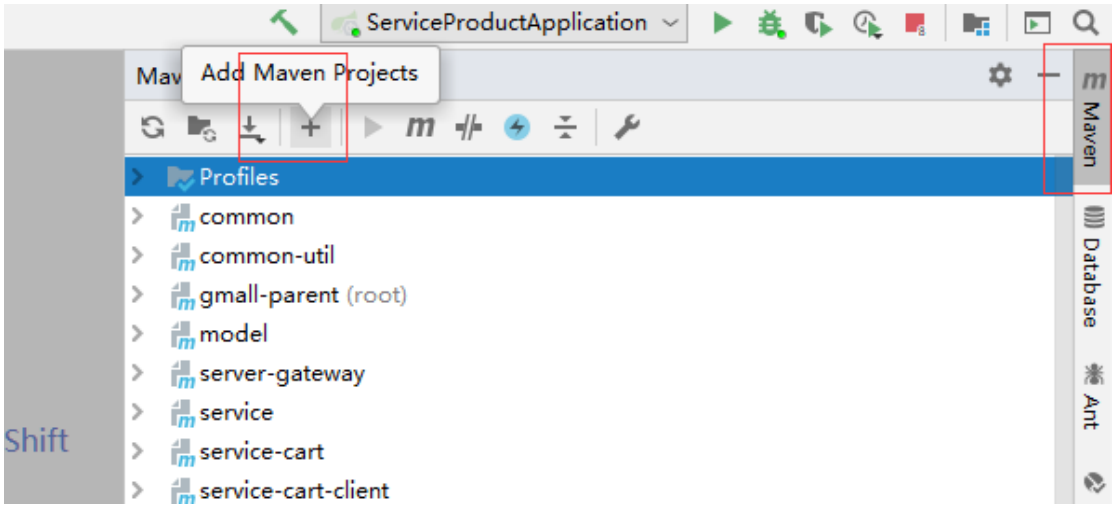
根据手册中的接口文档，编写调用代码。

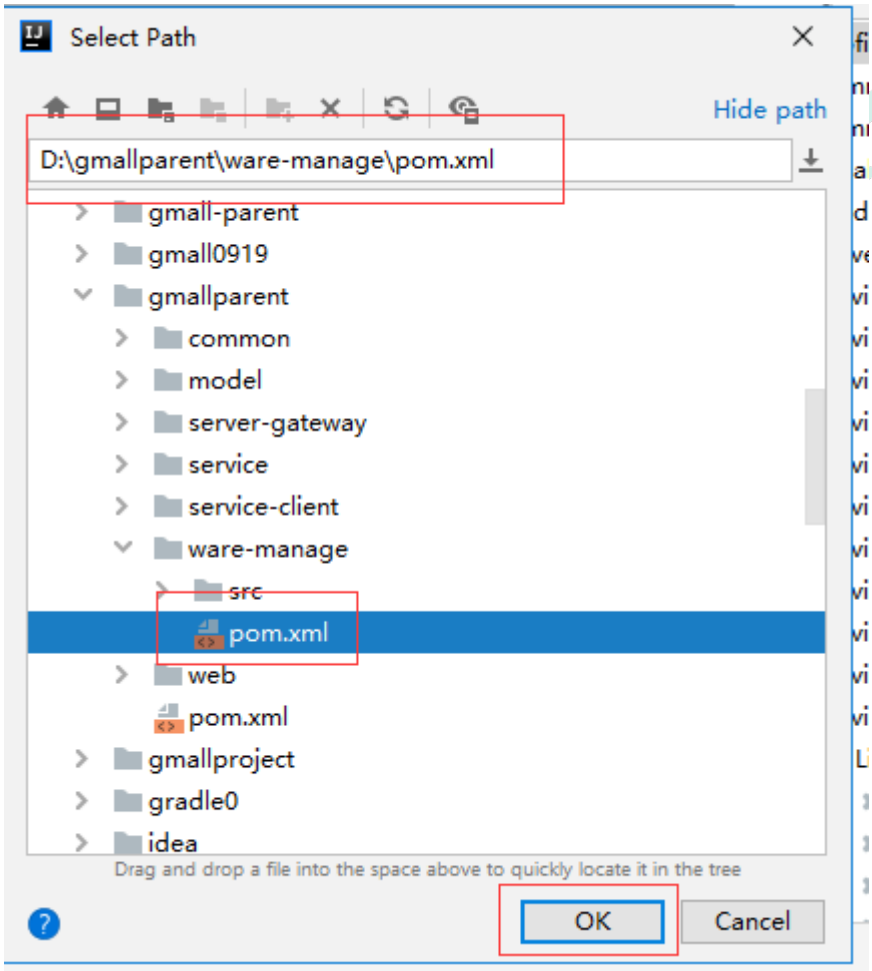
3.6.1 导入项目 ware-manage 项目

详情查看库存管理系统文档，把资料中的 ware-manage 项目直接放入到项目模块目录下。

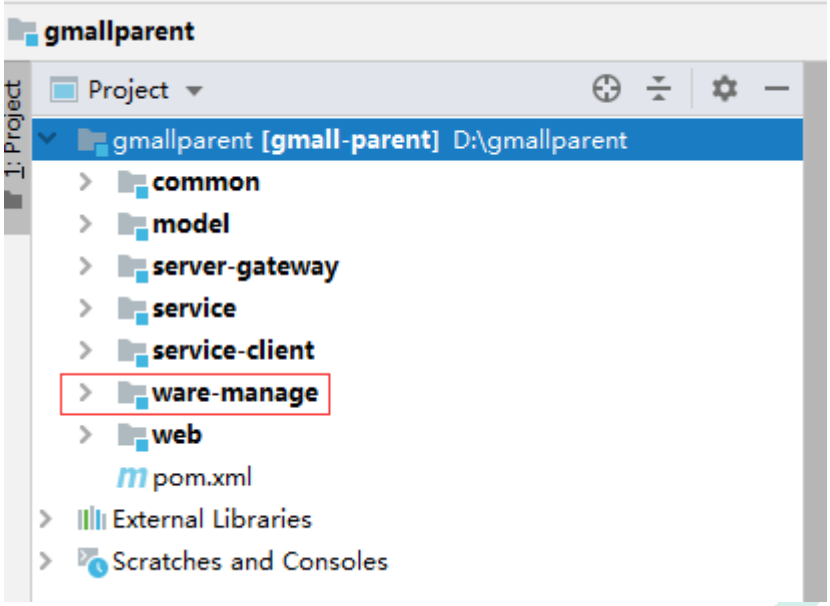


打开库存项目


ServiceProductApplication
Add Maven Projects
Profiles
common
common-util
gmall-parent (root)
model
server-gateway
service
service-cart
service-cart-client
Shift
Maven
Database
Ant


Select Path
Hide path
D:\gmallparent\ware-manage\pom.xml
gmall-parent
gmall0919
gmallparent
common
model
server-gateway
service
service-client
ware-manage
src
pom.xml
web
pom.xml
gmallproject
gradle0
idea
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel

点击 OK.



填写库存信息！

可以访问库存系统：http://localhost:9001/index

验证库存：

1、查询库存

接口	http://localhost:9001/hasStock
请求参数	skuId： 商品 skuId num： 商品数量
请求方式	get
例：	/hasStock?skuId=10221&num=2
返回值	0：无库存 1：有库存

添加商品库存的时候，商品的库存应该选择同一个库存！

17	18	17		2	1000	荣耀V30 PRO 李现同款 DXO122分 5G双模 麒麟990 5GSOC7	<null>
18	19	21		2	100	小米cc9手机 深蓝星球（蓝色） 6G+128G 全网通	<null>

3.6.2 查询仓库数量，进行校验

现在验证库存数量方法

在 orderService 接口中定义验库存接口

```
/**
 * 验证库存
 * @param skuId
 * @param skuNum
 * @return
 */
boolean checkStock(Long skuId, Integer skuNum);
```

实现类 OrderServiceImpl

```
@Value("${ware.url}")
private String WARE_URL;


@Override
public boolean checkStock(Long skuId, Integer skuNum) {
    // 远程调用 http://localhost:9001/hasStock?skuId=10221&num=2
    String result = HttpClientUtil.doGet(WARE_URL +
"/hasStock?skuId=" + skuId + "&num=" + skuNum);
    return "1".equals(result);
}
```

3.6.3 submitOrder 中增加 方法

```
@Autowired
private ProductFeignClient productFeignClient;



/**
 * 提交订单
 * @param orderInfo
 * @param request
```

```
 * @return
 */
@PostMapping("auth/submitOrder")
public Result submitOrder(@RequestBody OrderInfo orderInfo,
HttpServletRequest request) {
    // 获取到用户 Id
    String userId = AuthContextHolder.getUserId(request);
    orderInfo.setUserId(Long.parseLong(userId));

    // 获取前台页面的流水号
    String tradeNo = request.getParameter("tradeNo");

    // 调用服务层的比较方法
    boolean flag = orderService.checkTradeCode(userId, tradeNo);
    if (!flag) {
        // 比较失败!
        return Result.fail().message("不能重复提交订单！");
    }
    // 删除流水号
    orderService.deleteTradeNo(userId);


    // 验证库存:
    List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
    for (OrderDetail orderDetail : orderDetailList) {
        // 验证库存:
        boolean result = orderService.checkStock(orderDetail.getSkuId(),
orderDetail.getSkuNum());
        if (!result) {
            return Result.fail().message(orderDetail.getSkuName() + "库存不
足！");
        }
        // 验证价格:
        BigDecimal skuPrice =
productFeignClient.getSkuPrice(orderDetail.getSkuId());
        if (orderDetail.getOrderPrice().compareTo(skuPrice) != 0) {
            // 重新查询价格!
            // 设置最新的价格
List<CartInfo> cartCheckedList =
this.cartFeignClient.getCartCheckedList(userId);
// 写入缓存:
cartCheckedList.forEach(cartInfo -> {
    this.redisTemplate.opsForHash().put(RedisConst.USER_KEY_PREFIX +
userId + RedisConst.USER_CART_KEY_SUFFIX,
cartInfo.getSkuId().toString(), cartInfo);
});
            return Result.fail().message(orderDetail.getSkuName() + "价格有
变动！");
        }
    }
```

```
    // 验证通过，保存订单！
    Long orderId = orderService.saveOrderInfo(orderInfo);
    return Result.ok(orderId);
}
```

3.6.4 优化下单

下单我们要校验库存与价格，请求比较多，时间比较长，我们可以通过异步编排的形式减少请求时间，异步编排我们前面已经学习过了，接下来怎么做呢

3.6.4.1 引入线程类

将 service-item 下面的 config 线程池配置类 copy 过来

```
package com.atguigu.gmall.order.config;


@Configuration
public class ThreadPoolConfig {

    /**
     *
     * public ThreadPoolExecutor(int corePoolSize,
     *                                int maximumPoolSize,
     *                                long keepAliveTime,
     *                                TimeUnit unit,
     *                                         BlockingQueue<Runnable>
workQueue,
     *                                ThreadFactory threadFactory,
     *                                        RejectedExecutionHandler
handler)
     * 构造函数的参数含义如下：
     *
     * corePoolSize:指定了线程池中的线程数量，它的数量决定了添加的任务是开辟新的线程去执行，还是放到workQueue 任务队列中去；
     * maximumPoolSize:指定了线程池中的最大线程数量，这个参数会根据你使用的workQueue 任务队列的类型，决定线程池会开辟的最大线程数量；
     * keepAliveTime:当线程池中空闲线程数量超过corePoolSize 时，多余的线程会在多长时间内被销毁；
     * unit:keepAliveTime 的单位
     * workQueue:任务队列，被添加到线程池中，但尚未被执行的任务；它一般分为直接提交队列、有界任务队列、无界任务队列、优先任务队列几种；
     * threadFactory:线程工厂，用于创建线程，一般用默认即可；
     * handler:拒绝策略；当任务太多来不及处理时，如何拒绝任务；
```

```
     * @return
     */
    @Bean
    public ThreadPoolExecutor threadPoolExecutor(){

        return new ThreadPoolExecutor(50, 500, 30, TimeUnit.SECONDS,
new ArrayBlockingQueue<>(10000));
    }
}
```

3.6.4.2 调整下单类

```
@Autowired
private ThreadPoolExecutor threadPoolExecutor;
```

```
/**
 * 提交订单
 * @param orderInfo
 * @param request
 * @return
 */
@PostMapping("auth/submitOrder")
public Result submitOrder(@RequestBody OrderInfo orderInfo,
HttpServletRequest request) {
    // 获取到用户Id
    String userId = AuthContextHolder.getUserId(request);
    orderInfo.setUserId(Long.parseLong(userId));

    // 获取前台页面的流水号
    String tradeNo = request.getParameter("tradeNo");

    // 调用服务层的比较方法
    boolean flag = orderService.checkTradeCode(userId, tradeNo);
    if (!flag) {
        // 比较失败!
        return Result.fail().message("不能重复提交订单！");
    }

    // 删除流水号
    orderService.deleteTradeNo(userId);

    List<String> errorList = new ArrayList<>();
    List<CompletableFuture> futureList = new ArrayList<>();
    // 验证库存:
    List<OrderDetail> orderDetailList =
orderInfo.getOrderDetailList();
    for (OrderDetail orderDetail : orderDetailList) {
        CompletableFuture<Void> checkStockCompletableFuture =
CompletableFuture.runAsync(() -> {
            // 验证库存:
```

```
            boolean result = orderService.checkStock(orderDetail.getSkuId(), orderDetail.getSkuNum());
            if (!result) {
                errorList.add(orderDetail.getSkuName() + "库存不足！");
            }
        }, threadPoolExecutor);
        futureList.add(checkStockCompletableFuture);

        CompletableFuture<Void> checkPriceCompletableFuture = CompletableFuture.runAsync(() -> {
            // 验证价格:
            BigDecimal skuPrice = productFeignClient.getSkuPrice(orderDetail.getSkuId());
            if (orderDetail.getOrderPrice().compareTo(skuPrice) != 0) {
                // 重新查询价格!
List<CartInfo> cartCheckedList = this.cartFeignClient.getCartCheckedList(userId);
//  写入缓存:
cartCheckedList.forEach(cartInfo -> {
    this.redisTemplate.opsForHash().put(RedisConst.USER_KEY_PREFIX + userId + RedisConst.USER_CART_KEY_SUFFIX, cartInfo.getSkuId().toString(), cartInfo);
});

                errorList.add(orderDetail.getSkuName() + "价格有变动！");
            }
        }, threadPoolExecutor);
        futureList.add(checkPriceCompletableFuture);
    }

    //合并线程
    CompletableFuture.allOf(futureList.toArray(new CompletableFuture[futureList.size()])).join();
    if(errorList.size() > 0) {
        return Result.fail().message(StringUtils.join(errorList, ","));
    }

    // 验证通过，保存订单!
    Long orderId = orderService.saveOrderInfo(orderInfo);
    return Result.ok(orderId);
}
```

四、我的订单

4.1 添加 service 接口

在 OrderService 类添加接口

```
IPage<OrderInfo> getPage(Page<OrderInfo> pageParam, String userId);
```

4.2 添加 service 接口实现

在 OrderServiceImpl 类添加接口实现

```
@Override
public IPage<OrderInfo> getPage(Page<OrderInfo> pageParam, String userId) {
    IPage<OrderInfo> page = orderInfoMapper.selectPageByUserId(pageParam,
userId);
    page.getRecords().stream().forEach(item -> {

item.setOrderStatusName(OrderStatus.getStatusNameByStatus(item.getOrderStat
us()));
    });
    return page;
}
```

4.3 添加 mapper 接口

1，在 OrderInfoMapper 类添加接口

```
IPage<OrderInfo> selectPageByUserId(Page<OrderInfo> page,
@Param("userId")String userId);
```

2，添加接口对应的 xml 文件方法

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.order.mapper.OrderInfoMapper">

   <resultMap id="orderInfoMap" type="com.atguigu.gmall.model.order.OrderInfo" autoMapping="true">
        <id property="id" column="id"></id>
        <!---一对多-->
        <collection property="orderDetailList" ofType="com.atguigu.gmall.model.order.OrderDetail" autoMapping="true"
                column="{orderId = id}"
                select="selectOrderDetailByOrderId">
        </collection>
   </resultMap>

   <!-- 用于 select 查询公用抽取的列 -->
   <sql id="orderColumns">
id,consignee,consignee_tel,total_amount,order_status,user_id,payment_way,delivery_address,order_comment,out_trade_no,trade_body,create_time,expire_time,process_status,tracking_no,parent_order_id,img_url
   </sql>

   <sql id="orderDetailColumns">

id,order_id,sku_id,sku_name,img_url,order_price,sku_num,create_time,source_type,source_id,split_total_amount,split_activity_amount,split_coupon_amount
   </sql>

    <select id="selectPageByUserId" resultMap="orderInfoMap">
       select <include refid="orderColumns" />
       from order_info
      where user_id = #{userId}
      and order_status not in('CLOSED', 'SPLIT')
      and is_deleted = 0
      order by id desc
    </select>

   <select id="selectOrderDetailByOrderId" resultType="com.atguigu.gmall.model.order.OrderDetail">
      select <include refid="orderDetailColumns" />
      from order_detail
      where order_id = #{orderId}
      and is_deleted = 0
      order by id desc
   </select>

</mapper>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.order.mapper.OrderInfoMapper">
    <!--配置返回结果集映射-->
    <resultMap id="OrderInfoMap" type="com.atguigu.gmall.model.order.OrderInfo" autoMapping="true">
        <!--主键映射-->
        <id property="id" column="id"></id>
        <!--配置 1: n -->
        <collection property="orderDetailList" ofType="com.atguigu.gmall.model.order.OrderDetail" autoMapping="true">
            <id property="id" column="detail_id"></id>
        </collection>
    </resultMap>

    <select id="selectPageByuserId" resultMap="OrderInfoMap">
       SELECT
            oi.id,
            oi.consignee,
            oi.consignee_tel,
            oi.total_amount,
            oi.order_status,
            oi.user_id,
            oi.payment_way,
            oi.delivery_address,
            oi.order_comment,
            oi.out_trade_no,
            oi.trade_body,
            oi.create_time,
            oi.expire_time,
            oi.process_status,
            od.id detail_id,
            od.order_id,
            od.sku_id,
            od.sku_name,
            od.img_url,
            od.order_price,
            od.sku_num,
            od.create_time
        FROM
            order_info oi
        INNER JOIN order_detail od ON od.order_id = oi.id
        WHERE
            user_id = #{userId}
        AND oi.order_status NOT IN ('CLOSED', 'SPLIT')
        ORDER BY
            oi.id DESC
    </select>

</mapper>
```

4.4 添加 controller 接口

在 OrderApiController 类添加接口

```
@ApiOperation("我的订单")
@GetMapping("auth/{page}/{limit}")
public Result<IPage<OrderInfo>> index(
        @ApiParam(name = "page", value = "当前页码", required = true)
        @PathVariable Long page,

        @ApiParam(name = "limit", value = "每页记录数", required = true)
        @PathVariable Long limit,
        HttpServletRequest request) {
    // 获取到用户 Id
    String userId = AuthContextHolder.getUserId(request);

    Page<OrderInfo> pageParam = new Page<>(page, limit);
    IPage<OrderInfo> pageModel = orderService.getPage(pageParam, userId);
    return Result.ok(pageModel);
}
```

4.5 web-all 中添加控制器

```
/**
 * 我的订单
 * @return
 */
@GetMapping("myOrder.html")
public String myOrder() {
    return "order/myOrder";
}
```