闪聚支付 第2章 讲义-商户注册&资质申请

1 需求概述

1.1 商户注册

闪聚支付为商户提供聚合支付业务，线下商户和线上商户都可以使用闪聚支付平台。

什么是线下和线上商户？

1）线下场所支付商户

使用线下场所支付的商户是指有实体经营场所的商家，也称为地面商户，一般包含酒店、餐厅、酒吧、美容、 美发、 媒体、 影楼、 家政、 艺廊、 KTV、 会所等。

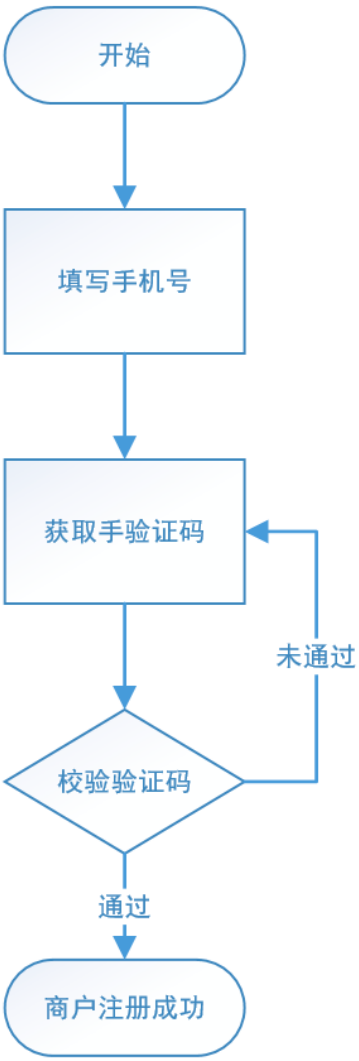
2）线上支付商户

使用线上支付的商户是指通过互联网进行经营服务的商家，常见的有：电商网站、团购网站、旅游网站等。

商户使用闪聚支付平台第一步要在平台进行注册。

商户填写手机号、账号、密码、获取验证码申请注册，注册成功后商户成为闪聚平台的用户，即可使用闪聚支付平台提供的服务。

商户注册的业务流程如下：



1、用户填写手机号、账号、密码等信息

2、点击获取手机验证码

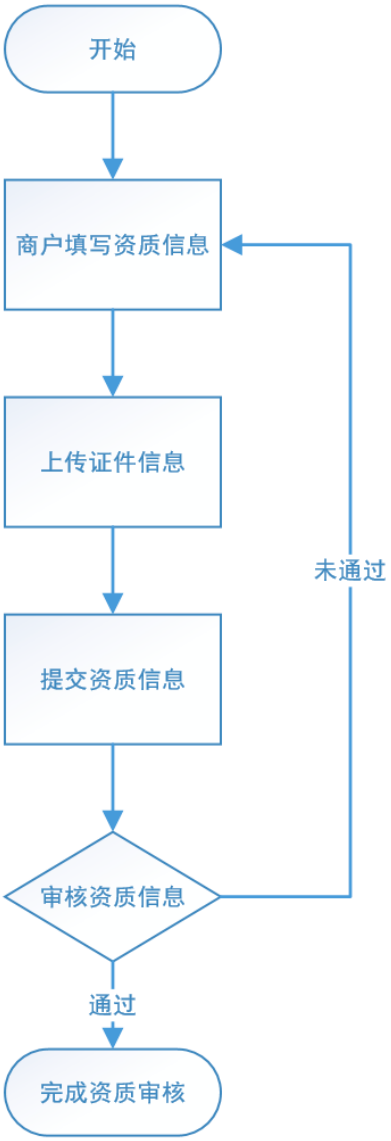
3、输入验证码，点击注册

4、商户注册成功

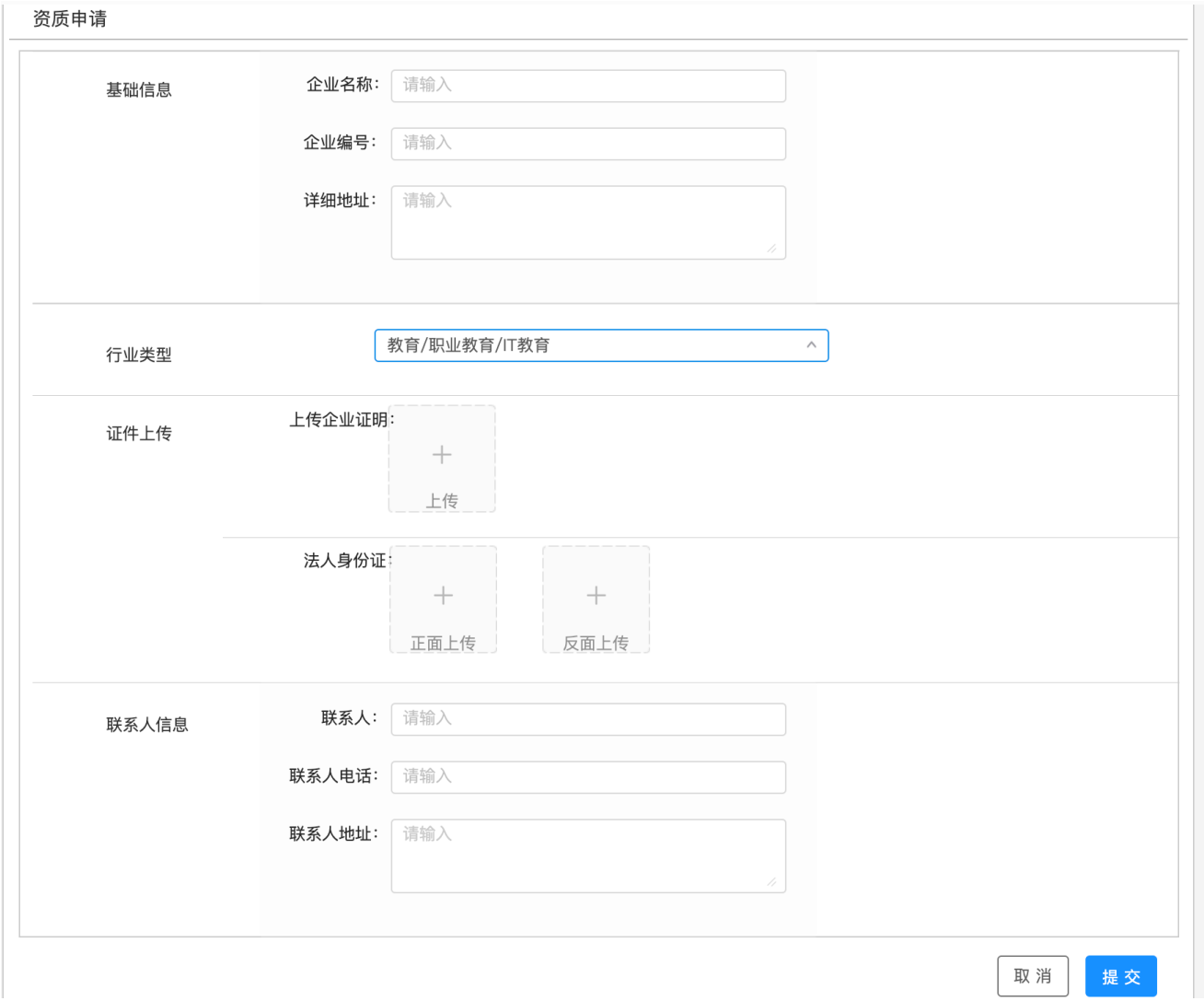
1.2 资质申请

商户在平台注册成功后，需要完善商户信息，将营业执照、申请人身份证以扫描件的形式上传到平台，由平台运营人员对商户资质进行审核，审核通过方可使用平台提供的服务。

资质申请业务流程如下：



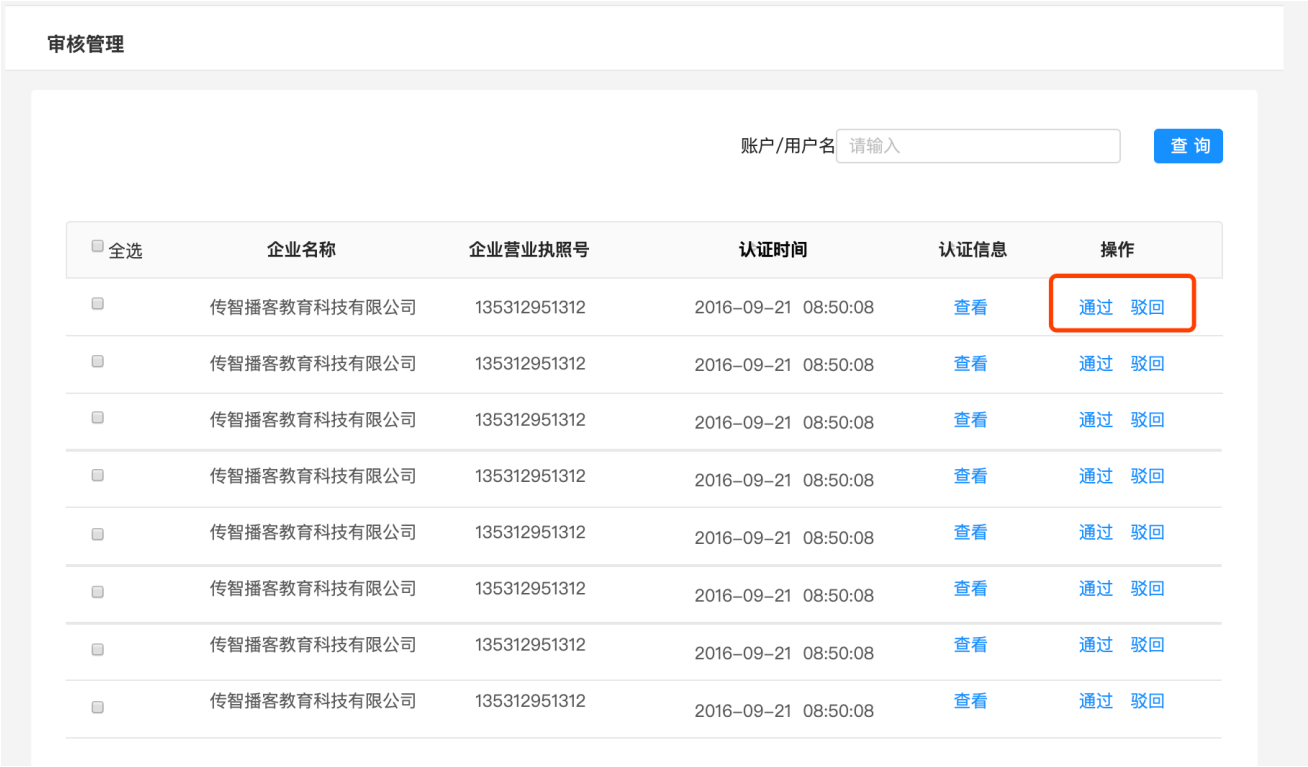
1、商户填写资质信息



2、上传营业执照和法人身份证图片

3、提交资质信息

4、平台运营人员对商户资质信息进行审核



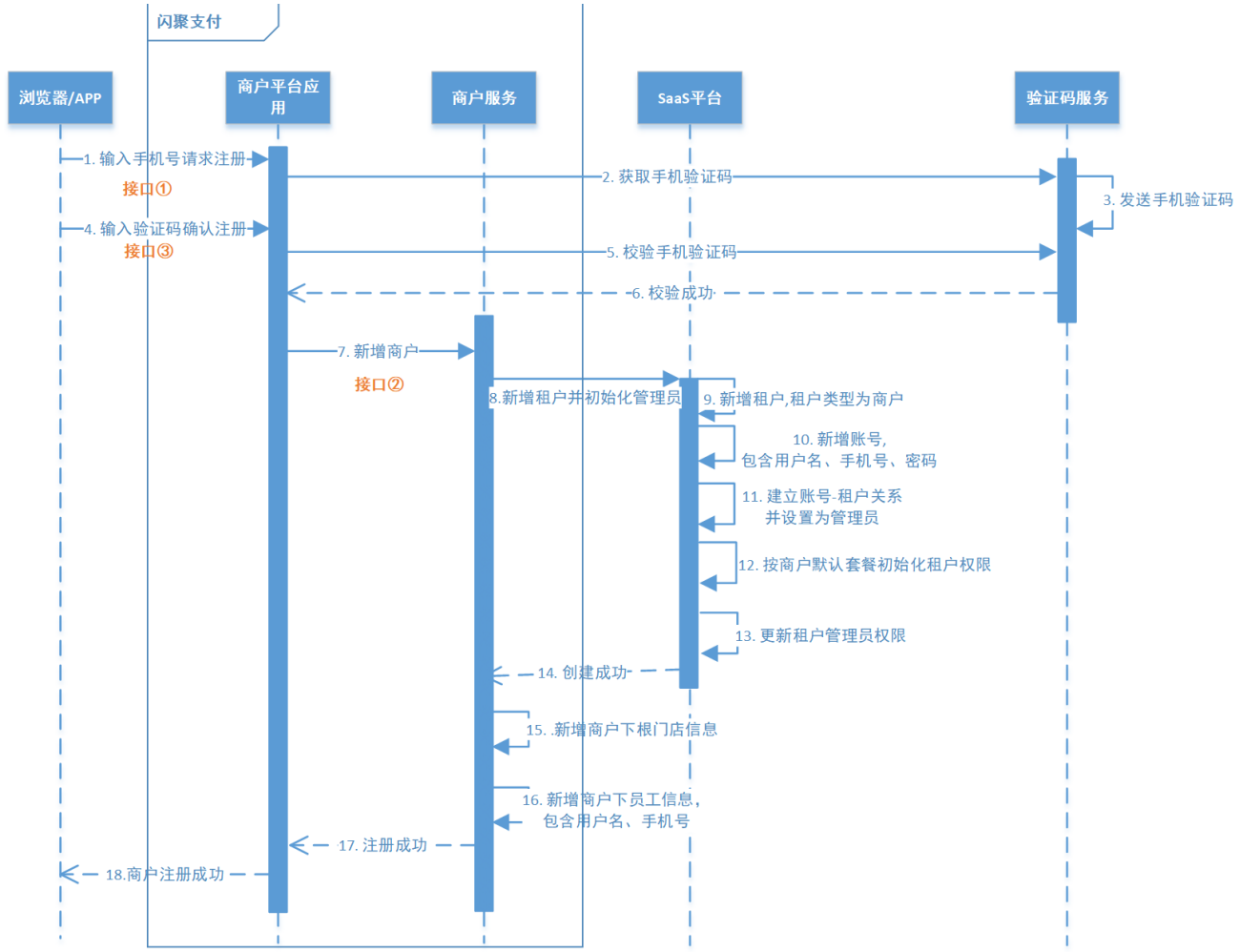
5、审核通过后，完成资质申请

2 商户注册

2.1 需求分析

2.1.1 系统交互流程

商户注册交互流程如下：



商户注册的流程由商户平台应用、商户服务、SaaS平台、验证码服务四个微服务之间进行交互完成，各微服务的职责介绍如下：

1）商户平台应用：此应用主要为商户提供业务功能，包括：商户资质申请、员工管理、门店管理等功能。

2）商户服务： 提供商户管理的相关服务接口，供其它微服务调用，主要为商户平台应用提供接口服务，功能包括：商户基本信息管理、资质申请、商户应用管理、渠道参数配置、商户员工信息管理、商户门店管理等。

3）SaaS平台：闪聚支付项目是一个SaaS平台 ，所谓SaaS平台即多个用户租用平台的业务功能，这样用户即可省去软件系统开发的成本，每个商户就是一个租户，所以又称为多租户系统。

SaaS平台提供租户管理、账号管理、权限管理、资源管理、套餐管理、系统认证授权等功业务功能。在上图商户注册的流程中，商户注册的账号等信息需要写入SaaS平台，由SaaS平台统一管理账号，分配权限，商户统一通过SaaS平台登录闪聚支付。

4）验证码服务：提供获取短信验证码、校验验证码的接口。

商户使用手机号进行注册，平台通过校验手机验证码来确认是否本人在注册。

交互流程如下：

1. 前端请求商户平台应用进行注册
2. 商户平台应用获取短信验证码
3. 前端携带手机验证码、账号、密码等信息请求商户平台应用确认注册
4. 验证码校验通过后请求商户服务新增商户

5. 商户服务请求SaaS平台新增租户并初始化管理员
6. SaaS平台返回创建成功给商户服务商户服务新增商户下根门店信息
7. 商户服务新增商户下员工信息
8. 注册成功

2.1.2 开发步骤

整个商户注册流程比较复杂，本模块采用迭代开发方式，具体开发步骤如下：

1、首先实现商户信息在商户服务注册成功（暂时不与SaaS平台交互）

商户信息只写入商户数据库，暂时不与SaaS平台交互。

2、待商户信息注册成功，资质申请通过、支付参数配置完成再与SaaS平台进行对接。

与SaaS平台交互前需要部署SaaS平台，学习SaaS暴露的接口及认证接口，接通SaaS方可实现用户登录，此部分放在本章节最后实现。

2.2 部署验证码服务

系统中所有验证码相关的功能由验证码服务提供，验证码服务是一个开源的项目，接入了腾讯、阿里等短信接口，本系统只需要和验证码服务接入即可使用腾讯、阿里等短信接口。

参考：“验证码服务使用指南.pdf”部署验证码服务。

2.3 获取短信验证码

根据系统整体交互流程，需要首先获取短信验证码。

2.3.1 RestTemplate技术预研

1、认识RestTemplate

验证码服务对外提供http接口，我们使用的postman和swagger-ui都属于http客户端的一种，使用它们可以调用验证码服务的接口获取验证码。现在我们需要使用Java程序模拟http客户端调用验证码服务的接口获取验证码。

RestTemplate是Spring提供的用于访问RESTful服务的客户端，RestTemplate提供了多种便捷访问远程Http服务的方法,能够大大提高客户端的编写效率。RestTemplate默认依赖JDK提供http连接的能力（HttpURLConnection），也可以通过替换为例如 Apache HttpComponents、Netty或OkHttp等其它HTTP 客户端，OkHttp的性能优越，本项目使用OkHttp，官网：https://square.github.io/okhttp/，github：https://github.com/square/okhttp。

在shanjupay-merchant-application工程中引入依赖：

```
        <!-- okhttp3依赖 -->
         <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
```

在父工程已规范了okhttp的版本

```
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
    <version>3.9.1</version>
</dependency>
```

在MerchantApplicationBootstrap类中添加RestTemplate初始化：

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate(new OkHttp3ClientHttpRequestFactory());
}
```

在test下创建测试程序如下：

使用RestTemplate获取百度的网页内容。

```
package com.shanjupay.merchant;

import com.alibaba.fastjson.JSON;
import lombok.extern.slf4j.Slf4j;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.*;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

import java.util.Map;
@SpringBootTest
@RunWith(SpringRunner.class)
@Slf4j
public class RestTemplateTest {

    @Autowired
    RestTemplate restTemplate;

    //获取网页内容
    @Test
    public void gethtml(){
        String url  = "http://www.baidu.com/";
        ResponseEntity<String> forEntity = restTemplate.getForEntity(url, String.class);
        String body = forEntity.getBody();
        System.out.println(body);
    }
  }
```

通过测试发现可以成功获取百度的网页内容。

网页内容中中文乱码解决方案：

原因：

当RestTemplate默认使用String存储body内容时默认使用ISO_8859_1字符集。

解决：

配置StringHttpMessageConverter 消息转换器，使用utf-8字符集。

修改RestTemplate的定义方法

```
@Bean
public RestTemplate restTemplate() {
    RestTemplate restTemplate = new RestTemplate(new OkHttp3ClientHttpRequestFactory());
    //消息转换器列表
    List<HttpMessageConverter<?>> messageConverters = restTemplate.getMessageConverters();
    //配置消息转换器StringHttpMessageConverter，并设置utf-8
    messageConverters.set(1,
            new StringHttpMessageConverter(StandardCharsets.UTF_8));//支持中文字符集，默认ISO-
8859-1，支持utf-8

    return restTemplate;
}
```

2、使用RestTemplate获取验证码

下边通过RestTemplate请求验证码服务获取验证码。

```
//获取验证码测试方法
@Test
public void testGetSmsCode(){
    String url = "http://localhost:56085/sailing/generate?name=sms&effectiveTime=600";//验证码过
期时间为600秒
        String phone = "13434343434";
        log.info("调用短信微服务发送验证码：url:{}", url);

        //请求体
        Map<String, Object> body = new HashMap();
        body.put("mobile", phone);
        //请求头
        HttpHeaders httpHeaders = new HttpHeaders();
        //设置数据格式为json
        httpHeaders.setContentType(MediaType.APPLICATION_JSON);
        //封装请求参数
        HttpEntity entity = new HttpEntity(body,httpHeaders);

        Map responseMap = null;
        try {
            //post请求
            ResponseEntity<Map> exchange = restTemplate.exchange(url, HttpMethod.POST, entity,
Map.class);
            log.info("调用短信微服务发送验证码: 返回值:{}", JSON.toJSONString(exchange));
```

```
            //获取响应
            responseMap = exchange.getBody();
        } catch (Exception e) {
            log.info(e.getMessage(), e);
        }
        //取出body中的result数据
        if (responseMap != null || responseMap.get("result") != null) {
            Map resultMap = (Map) responseMap.get("result");
            String value = resultMap.get("key").toString();
            System.out.println(value);
        }


}
```

2.3.2 商户平台应用获取验证码(接口①)

2.3.2.1接口定义

在商户平台应用工程定义获取验证码接口：

1、接口交互流程如下：

1）获取手机号

2）向验证码服务请求发送验证码并得到响应

3）响应前端验证码发送结果

2、在MerchantController类中添加getSMSCode接口方法，此接口供前端调：

```
    @ApiOperation("获取手机验证码")
    @ApiImplicitParam(name = "phone", value = "手机号", required = true, dataType = "String",
paramType = "query")
    @GetMapping("/sms")
    public String getSMSCode(@RequestParam String phone) {
        log.info("向手机号:{}发送验证码", phone);
        ...
    }
```

3、Service接口定义如下：

为了方便程序复用在 SmsService 接口中添加sendMsg方法用来获取验证码：

```
package com.shanjupay.merchant.service;

import java.util.Map;

/**
 * <P>
 * 手机短信服务
 * </p>
```

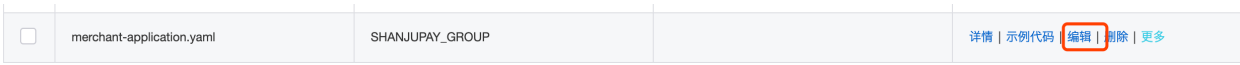
```
 *
 */
public interface SmsService {

    /**
     * 获取短信验证码
     * @param phone
     * @return
     */
    String sendMsg(String phone);

}
```

2.3.2.2 接口实现

1、配置验证码服务的接口地址

在nacos上配置验证码的接口地址

1. 登录nacos，编辑merchant-application.yaml配置：

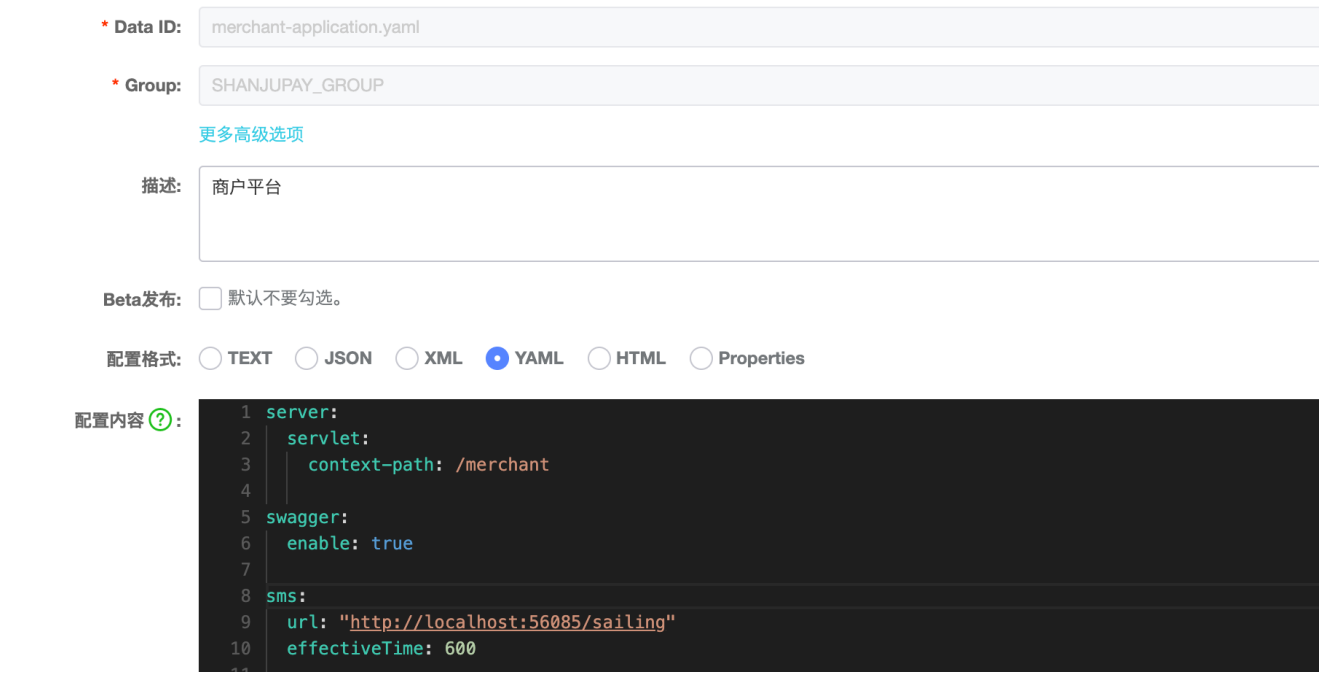


2. 配置如下参数

```
sms:
  url: "http://localhost:56085/sailing"
  effectiveTime: 600
```

url：验证码服务地址

effectiveTime：验证码过期时间



2、编写接口实现方法

编写SmsServiceImpl实现sendMsg方法。

```
@Slf4j
@Service
public class SmsServiceImpl implements SmsService {

@Value("${sms.url}")
private String smsUrl;

@Value("${sms.effectiveTime}")
private String effectiveTime;

@Autowired
private RestTemplate restTemplate;

/**
 * 获取短信验证码
 * @param phone
 * @return
 */
public String sendMsg(String phone) {
    String url = smsUrl + "/generate?name=sms&effectiveTime=" + effectiveTime;//验证码过期时间为
600秒  10分钟
    log.info("调用短信微服务发送验证码：url:{}", url);

    Map<String, Object> body = new HashMap<String, Object>();
    body.add("mobile", phone);
    HttpHeaders httpHeaders = new HttpHeaders();

    httpHeaders.setContentType(MediaType.APPLICATION_JSON);
```

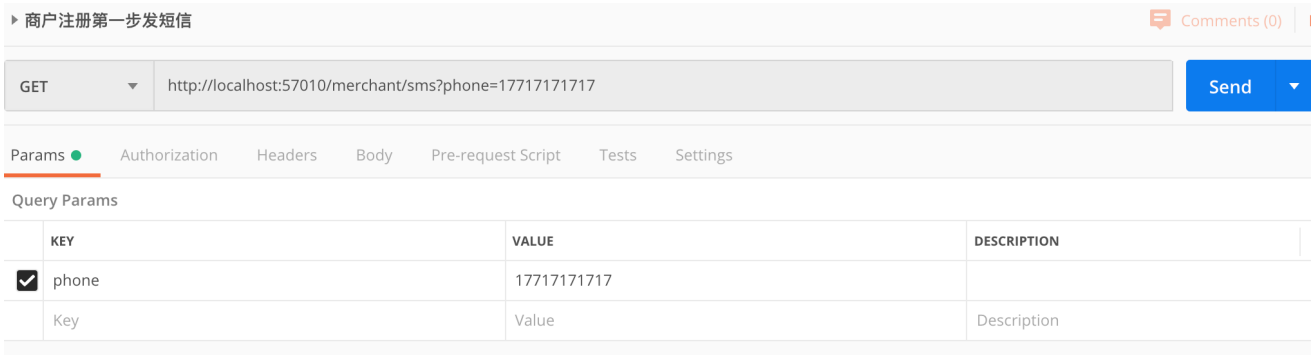
```
        HttpEntity entity = new HttpEntity(body,httpHeaders);

        Map responseMap;
        try {
            ResponseEntity<Map> exchange = restTemplate.exchange(url, HttpMethod.POST, entity,
Map.class);
            log.info("调用短信微服务发送验证码: 返回值:{}", JSON.toJSONString(exchange));
            responseMap = exchange.getBody();
        } catch (Exception e) {
            log.info(e.getMessage(), e);
            throw new RuntimeException("发送验证码出错");
        }

        if (responseMap == null || responseMap.get("result") == null) {
            throw new RuntimeException("发送验证码出错");
        }
        Map resultMap = (Map) responseMap.get("result");
        return resultMap.get("key").toString();
    }


}
```

3、Controller实现方法如下

```
@ApiOperation("获取手机验证码")
@ApiImplicitParam(name = "phone", value = "手机号", required = true, dataType = "String",
paramType = "query")
@GetMapping("/sms")
public String getSMSCode(@RequestParam String phone) {
    log.info("向手机号:{}发送验证码", phone);
    return smsService.sendMsg(phone);
}
```

2.3.2.3 接口测试

使用postman测试获取验证码接口：http://localhost:57010/merchant/sms?phone=17717171717



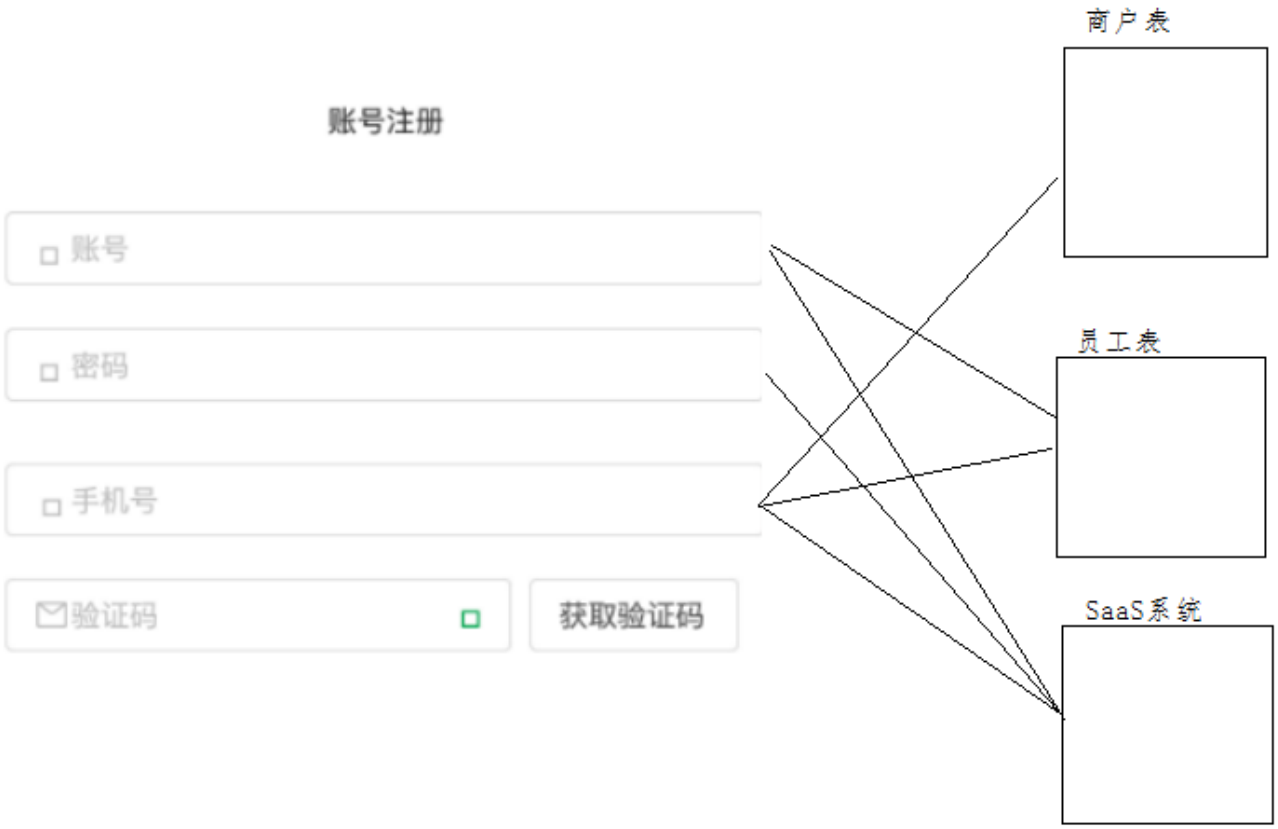
2.4 商户注册

2.4.1 系统设计

商户拿到获取的验证码即可提交注册信息，完成商户注册，本节完成商户注册功能。

商户注册信息向三个地方写入数据 ，如下图：

当前阶段暂时只向商户表写入数据 ，待接入SaaS系统时向员工表、SaaS系统写入。



商户注册表的数据模型如下：

商户表

Field	Type	Comment	
ID	bigint(20) NOT NULL	主键	
MERCHANT_NAME	varchar(50) NULL	商户名称	资质申请时完善
MERCHANT_NO	bigint(20) NULL	企业编号	资质申请时完善
MERCHANT_ADDRESS	varchar(255) NULL	企业地址	资质申请时完善
MERCHANT_TYPE	varchar(50) NULL	商户类型	资质申请时完善
BUSINESS_LICENSES_IMG	varchar(100) NULL	营业执照（企业证明）	资质申请时完善
ID_CARD_FRONT_IMG	varchar(100) NULL	法人身份证正面照片	资质申请时完善
ID_CARD_AFTER_IMG	varchar(100) NULL	法人身份证反面照片	资质申请时完善
USERNAME	varchar(50) NULL	联系人姓名	资质申请时完善
MOBILE	varchar(50) NULL	联系人手机号(关联统一账号)	注册时写入
CONTACTS_ADDRESS	varchar(255) NULL	联系人地址	资质申请时完善
AUDIT_STATUS	varchar(20) NULL	审核状态 0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝	注册成功默认为0
TENANT_ID	bigint(20) NULL	租户ID,关联统一用户	关联SaaS

2.4.2 商户服务注册商户(接口②)

商户服务提供商户注册的服务接口供商户平台应用使用，本节实现商户服务注册商户接口。

2.4.2.1 接口定义

在商户服务定义商户注册接口

1、接口描述如下：

1）校验商户注册的基础数据：账号、密码、手机号

2）添加商户

注意：本节暂不实现对接SaaS系统，仅实现新增一个商户，并将手机号填写到商户表。

3）返回注册结果给商户平台应用

2、定义商户注册DTO

此DTO作为商户注册、资质申请的公共类。

在shanjupay-merchant-api工程的com.shanjupay.merchant.api.dto包下添加MerchantDTO：

```java
package com.shanjupay.merchant.api.dto;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;

@ApiModel(value = "MerchantDTO", description = "商户信息")
@Data
public class MerchantDTO implements Serializable {

    @ApiModelProperty("商户id")
    private Long id;

    @ApiModelProperty("企业名称")
    private String merchantName;

    @ApiModelProperty("企业编号")
    private Long merchantNo;

    @ApiModelProperty("企业地址")
    private String merchantAddress;

    @ApiModelProperty("行业类型")
    private String merchantType;

    @ApiModelProperty("营业执照")
    private String businessLicensesImg;

    @ApiModelProperty("法人身份证正面")
    private String idCardFrontImg;

    @ApiModelProperty("法人身份证反面")
    private String idCardAfterImg;

    @ApiModelProperty("联系人")
    private String username;

    @ApiModelProperty("密码")
    private String password;

    @ApiModelProperty("手机号,关联统一账号")
    private String mobile;

    @ApiModelProperty("联系人地址")
    private String contactsAddress;

    @ApiModelProperty("审核状态,0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝")
    private String auditStatus;

    @ApiModelProperty("租户ID")

    private Long tenantId;
```

```
}
```

DTO可以使用"代码生成工具"自动生成：

修改代码生成工具中的MyBatisPlusGenerator类：

```
public class MyBatisPlusGenerator {
...
private static final Boolean IS_DTO = true;
...
if (IS_DTO) {
    globalConfig.setSwagger2(true);
    globalConfig.setEntityName("%sDTO");
    packageConfig.setEntity("dto");
}
System.out.println("====================== MyBatis Plus Generator ==================");

autoGenerator.execute();
...
```

当生成dto时将IS_DTO设置为true，运行生成工具即可。

3、接口定义如下：

在MerchantService接口类中定义如下接口：

```
/**
 * 商户注册
 * @return
*/
MerchantDTO createMerchant(MerchantDTO merchantDTO) ;
```

2.4.2.2 接口实现

实现MerchantServiceImpl中的createMerchant方法：

```
@org.apache.dubbo.config.annotation.Service
@Slf4j
public class MerchantServiceImpl implements MerchantService {

    @Override
    @Transactional
    public MerchantDTO createMerchant(MerchantDTO merchantDTO) {
        Merchant merchant = new Merchant();
        //设置审核状态0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝
        merchant.setAuditStatus("0");
        //设置手机号
        merchant.setMobile(merchantDTO.getMobile());
        //...
        //保存商户
        merchantMapper.insert(merchant);
```

```
        //将新增商户id返回
        merchantDTO.setId(merchant.getId());
        return merchantDTO;
    }

}
```

2.4.3 商户平台应用注册商户(接口③)

商户平台应用调用商户服务提供的商户注册接口即可完成商户注册业务，本节实现商户平台应用注册商户接口的开发。

2.4.3.1 接口定义

1、接口描述如下：

1）接收商户填写的注册数据

2）商户平台应用校验手机验证码

3）请求商户服务进行商户注册

4）返回注册结果给前端

2、定义商户注册VO：

在com.shanjupay.merchant.vo包下添加VO类型，VO类型负责接收前端请求的数据。

下图是商户注册的用户界面，根据用户界面中的元素定义VO类的属性：


```
package com.shanjupay.merchant.vo;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;

@ApiModel(value = "MerchantRegisterVO", description = "商户注册信息")
@Data
public class MerchantRegisterVO implements Serializable{

    @ApiModelProperty("商户手机号")
    private String mobile;

    @ApiModelProperty("商户用户名")
    private String username;

    @ApiModelProperty("商户密码")
    private String password;

    @ApiModelProperty("验证码的key")
    private String verifiykey;

    @ApiModelProperty("验证码")
    private String verifiyCode;

}
```

3、接口定义如下：

在MerchantController类中定义如下接口：

```
    @ApiOperation("注册商户")
    @ApiImplicitParam(name = "merchantRegister", value = "注册信息", required = true, dataType =
"MerchantRegisterVO", paramType = "body")
    @PostMapping("/merchants/register")
    public MerchantRegisterVO registerMerchant(@RequestBody MerchantRegisterVO merchantRegister)
{
      //校验验证码
      //...
      //注册商户
      //...
      return merchantRegister;
    }
```

2.4.3.2 校验验证码实现

1. 在SmsService中定义校验验证码的service接口

```
    /**
     * 校验验证码，抛出异常则校验无效
     * @param verifiyKey 验证码key
     * @param verifiyCode 验证码
     */
    void checkVerifiyCode(String verifiyKey,String verifiyCode) ;
```

2. 在SmsServiceImpl中定义校验验证码的service 实现方法

```
publicvoid checkVerifiyCode(String verifiyKey,String verifiyCode) {
    //实现校验验证码的逻辑
        String url = smsUrl+"/verify?
name=sms&verificationCode="+verifiyCode+"&verificationKey="+verifiyKey;
        Map responseMap = null;
        try {
            //请求校验验证码
            ResponseEntity<Map> exchange = restTemplate.exchange(url, HttpMethod.POST,
HttpEntity.EMPTY, Map.class);
            responseMap = exchange.getBody();
            log.info("校验验证码，响应内容：{}",JSON.toJSONString(responseMap));
        } catch (Exception e) {
            e.printStackTrace();
            log.info(e.getMessage(),e);
            throw new RuntimeException("验证码错误");
        }

        if(responseMap == null || responseMap.get("result")==null || !(Boolean)
responseMap.get("result")){
            throw new RuntimeException("验证码错误");
        }
}
```

3. 在controller中调用校验验证码

```
@PostMapping("/merchants/register")
    public MerchantRegisterVO registerMerchant(@RequestBody MerchantRegisterVO merchantRegister)
{
        //校验验证码
        smsService.checkVerifiyCode(merchantRegister.getVerifiykey(),
merchantRegister.getVerifiyCode());
        return merchantRegister;
    }
```

2.4.3.3 注册商户实现

在Controller中调用Dubbo接口实现注册商户

```
...
```

```
@Reference
private MerchantService merchantService;

@PostMapping("/merchants/register")
public MerchantRegisterVO registerMerchant(@RequestBody MerchantRegisterVO merchantRegister){
    //校验验证码
    smsService.checkVerifiyCode(merchantRegister.getVerifiykey(),
merchantRegister.getVerifiyCode());
    //注册商户
    MerchantDTO merchantDTO = new MerchantDTO();
    merchantDTO.setUsername(merchantRegister.getUsername());
    merchantDTO.setMobile(merchantRegister.getMobile());
    merchantDTO.setPassword(merchantRegister.getPassword());
    merchantService.createMerchant(merchantDTO);
    return merchantRegister;
}
```

2.4.3.4 接口测试

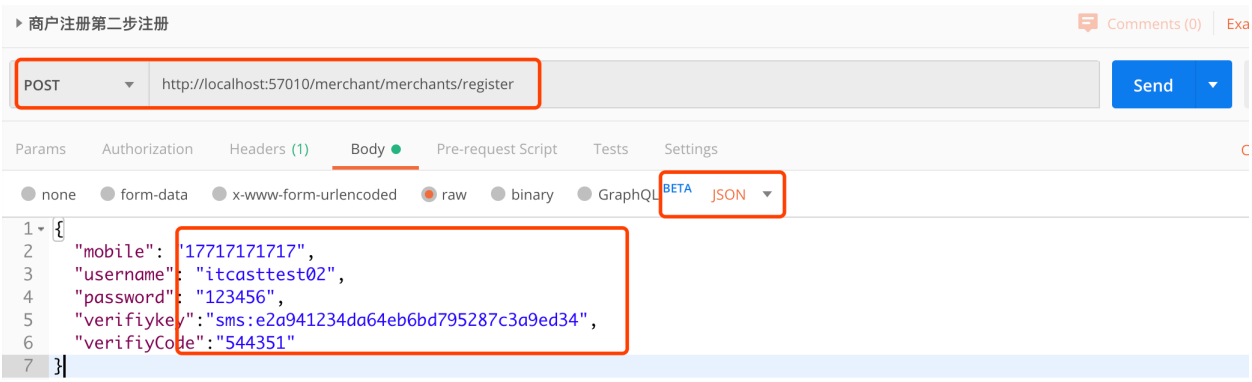
下面使用Postman对注册商户整体流程进行测试：

1. 获取短信验证码

   过程略。

2. 商户注册：http://localhost:57010/merchant/merchants/register，填写注册信息：

```
{
  "mobile": "17717171717",
  "username": "test02",
  "password": "123456",
  "verifiykey":"sms:e2a941234da64eb6bd795287c3a9ed34",
  "verifiyCode":"544351"
}
```



3. 注册成功

2.4.4 MapStruct对象转换

2.4.4.1 数据传输对象

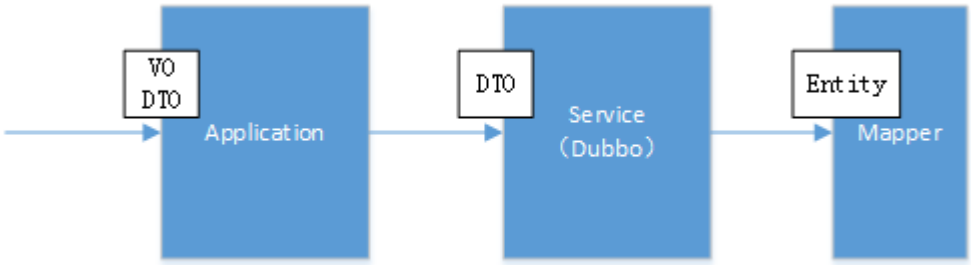
在商户注册开发过程中用于数据传输的对象有MerchantRegisterVO、MerchantDTO、entity(实体类)，它们的用途如下：

1、MerchantRegisterVO用于应用层接收前端请求及响应前端数据。

2、MerchantDTO 用于服务层传入及响应数据。

3、entity(实体类) 用于持久层传入及响应数据。

如下图：



数据传输对象（Data Transfer Object）是系统在交互过程中根据需要及规范将数据封装到数据对象中进行传输。

本项目数据传输对象的规范：

1、应用层

如没有接口参数的特殊要求，应用层使用DTO结尾的对象传输，否则单独定义VO结尾的对象传输。

2、服务层

统一使用DTO结尾的对象传输。

3、持久层

统一使用Entity对象传输。

2.4.4.2 MapStruct

1、数据传输对象转换的繁琐

每层有自己的数据传输对象，当数据流程到该层由需要将数据转成符合要求的格式，比如：当数据由应用层流转到服务层则需要将数据转成DTO格式，当数据由服务层流向持久层则需要将数据转成Entity格式数据

下边的代码数据由服务层流向持久层：

```
public MerchantDTO createMerchant(MerchantDTO merchantDTO) {
        Merchant entity = new Merchant();
        //设置审核状态0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝
        entity.setAuditStatus("0");
        //设置联系人
        entity.setUsername(merchantDTO.getUsername());
        //设置手机号
        entity.setMobile(merchantDTO.getMobile());
        //...
        //保存商户信息

        merchantMapper.insert(entity);
```

```
        //将新增商户的id回写到merchantDTO
        merchantDTO.setId(entity.getId());
        return merchantDTO;
    }
```

上边代码的问题是：由merchantDTO转成entity实现过程繁琐。

2、MapStruct解决数据传输对象转换的繁琐

MapStruct是一个代码生成器，它基于约定优于配置的方法大大简化了Java Bean对象之间的映射转换的实现。MapStruct 使用简单的方法即可完成对象之间的转换，它速度快、类型安全且易于理解。

> 官方地址：https://mapstruct.org/

1）添加依赖

在使用MapStruct的工程添加MapStruct依赖：

```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct-jdk8</artifactId>
</dependency>
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct-processor</artifactId>
    <version>${org.mapstruct.version}</version>
</dependency>
```

2）服务层对象转换

在商户服务工程定义商户对象转换类

定义MerchantConvert转换类，使用@Mapper注解快速实现对象转换

```
package com.shanjupay.merchant.convert;

import com.shanjupay.merchant.api.dto.MerchantDTO;
import com.shanjupay.merchant.entity.Merchant;
import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;

@Mapper
public interface MerchantCovert {

    MerchantCovert INSTANCE = Mappers.getMapper(MerchantCovert.class);

    MerchantDTO entity2dto(Merchant entity);

    Merchant dto2entity(MerchantDTO dto);
}
```

在MerchantCovert中定义测试方法：

```
public static void main(String[] args) {
     //dto转entity
     MerchantDTO merchantDTO = new MerchantDTO();
     merchantDTO.setUsername("测试");
     merchantDTO.setPassword("111");
     Merchant entity = MerchantCovert.INSTANCE.dto2entity(merchantDTO);
     //entity转dto
     entity.setMobile("123444554");
     MerchantDTO merchantDTO1 = MerchantCovert.INSTANCE.entity2dto(entity);
     System.out.println(merchantDTO1);
}
```

List数据也可以转换：

在MerchantCovert中定义list的方法，如下：

```
//list之间的转换
List<MerchantDTO> listentity2dto(List<Merchant> list);
```

测试：

在main方法编写list之间的转换测试

```
//测试list之间的转换
List<Merchant> list_entity = new ArrayList<>();
list_entity.add(entity);
List<MerchantDTO> merchantDTOS = MerchantCovert.INSTANCE.listentity2dto(list_entity);
System.out.println(merchantDTOS);
```

3）应用层对象转换

在商户平台应用工程定义商户对象转换类

```
package com.shanjupay.merchant.convert;

import com.shanjupay.merchant.api.dto.MerchantDTO;
import com.shanjupay.merchant.vo.MerchantRegisterVO;
import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;

@Mapper
public interface MerchantRegisterConvert {

    MerchantRegisterConvert INSTANCE = Mappers.getMapper(MerchantRegisterConvert.class);

    MerchantDTO vo2dto(MerchantRegisterVO vo);

    MerchantRegisterVO dto2vo(MerchantDTO dto);
}
```

2.4.4.3 代码优化

1、优化服务层代码

修改商户服务工程MerchantServiceImpl中的createMerchant方法：

```
public MerchantDTO createMerchant(MerchantDTO merchantDTO) {

    //将dto转成entity
    Merchant entity = MerchantCovert.INSTANCE.dto2entity(merchantDTO);
    //设置审核状态0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝
    entity.setAuditStatus("0");
    //保存商户信息
    merchantMapper.insert(entity);
    //将entity转成 dto
    MerchantDTO merchantDTONew = MerchantCovert.INSTANCE.entity2dto(entity);
    return merchantDTONew;
}
```

2、代码应用层代码

修改商户平台应用工程MerchantController中的registerMerchant方法：

```
@PostMapping("/merchants/register")
public MerchantRegisterVO registerMerchant(@RequestBody MerchantRegisterVO merchantRegister){
    //校验验证码
    smsService.verificationMessageCode(merchantRegister.getVerifiykey(),
merchantRegister.getVerifiyCode());
    //注册商户
    MerchantDTO merchantDTO = MerchantRegisterConvert.INSTANCE.vo2dto(merchantRegister);
    merchantService.createMerchant(merchantDTO);
    return merchantRegister;
}
```

2.4.5 异常处理

2.4.5.1 异常信息格式

系统在交互中难免会有异常发生，前端为了解析异常信息向用户提示特定义了异常信息的返回格式，如下：

1、返回response状态说明

状态码	说明
200	成功
401	没有权限
500	程序错误（需要自定义错误体）

2、自定义错误体

```
{
"errCode": "000000",
"errMessage": "错误说明"
}
```

2.4.5.2 异常处理流程

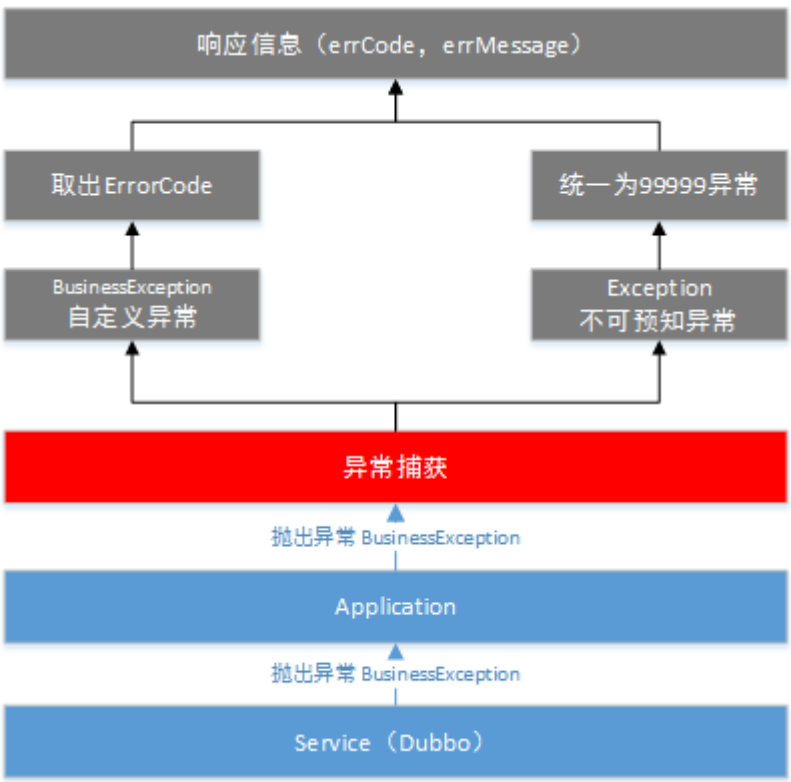
截至目前系统并没有按照前端要求返回异常信息，测试如下：

注册商户时输入一个错误的验证码，返回信息如下：

```
{
    "timestamp": "2019-12-10T10:06:19.936+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "验证码错误",
    "path": "/merchant/merchants/register"
}
```

从上边的返回信息得知，状态码为500符合要求，按前端的规范定义的错误信息要写在“errMessage” 中，显然不符合要求。

系统规范了异常处理流程，如下：



1、在服务层抛出自定义异常类型及不可预知异常类型。

上图中BusinessException为系统的自定义异常类型，程序中在代码显示抛出该异常，此类异常是程序员可预知的。

另一部分是系统无法预知的异常，如：数据库无法连接，服务器宕机等场景下所抛出的异常，此类异常是程序员无法预知的异常。

2、应用层接收到服务层抛出异常继续向上抛出，应用层自己也可以抛出自定义异常类型及不可预知异常类型。

3、统一异常处理器捕获到异常进行解析。

判断如果为自定义异常则直接取出错误代码及错误信息，因为程序员在抛出自定义异常时已将错误代码和异常信息指定。

如果为不可预知的异常则统一定义为99999异常代码。

4、统一异常处理器将异常信息格式为前端要求的格式响应给前端。

服务端统一将异常信息封装在下边的json格式中返回：

```
{
"errCode": "000000",
"errMessage": "错误说明"
}
```

2.4.5.3 自定义业务异常类

1. 在shanjupay-common工程的com.shanjupay.common.domain包下添加业务异常类BusinessException：

```
package com.shanjupay.common.domain;

public class BusinessException extends RuntimeException {

    //错误代码
    private ErrorCode errorCode;

    public BusinessException(ErrorCode errorCode) {
        super();
        this.errorCode = errorCode;
    }
    public BusinessException(){
        super();
    }
```

```
public void setErrorCode(ErrorCode errorCode) {
    this.errorCode = errorCode;
}

public ErrorCode getErrorCode() {
    return errorCode;
}
```

```
}
```

2、定义错误代码

在common工程专门定义了ErrorCode接口及CommonErrorCode通用代码。

从资料文件夹的代码目录获取 CommonErrorCode.java类。

2.4.5.4 自定义业务异常处理器

1、在shanjupay-common工程的com.shanjupay.common.domain包下添加错误响应包装类RestErrorResponse：

```
package com.shanjupay.common.domain;
```

```
import io.swagger.annotations.ApiModel;
```

```
@ApiModel(value = "RestErrorResponse", description = "错误响应参数包装")
@Data
public class RestErrorResponse {

    private String errCode;

    private String errMessage;

    public RestErrorResponse(String errCode,String errMessage){
        this.errCode = errCode;
        this.errMessage= errMessage;
    }
```

```
}
```

2、定义全局异常处理器

全局异常处理器使用ControllerAdvice注解实现，ControllerAdvice是SpringMVC3.2提供的注解，用ControllerAdvice可以方便实现对Controller面向切面编程，具体用法如下：

1、ControllerAdvice和ExceptionHandler注解实现全局异常处理

2、ControllerAdvice和ModelAttribute注解实现全局数据绑定

3、ControllerAdvice生InitBinder注解实现全局数据预处理

今天学习第一种用法，其它用法有兴趣的可自行查阅相关资料。

ControllerAdvice和ExceptionHandler结合可以捕获Controller抛出的异常，根据异常处理流程，Service和持久层最终都会抛给Controller，所以此方案可以实现全局异常捕获，异常被捕获到即可格式为前端要的信息格式响应给前端。

在shanjupay-merchant-application工程的com.shanjupay.merchant.common.intercept添加GlobalExceptionHandler：

```java
package com.shanjupay.merchant.common.intercept;

import com.shanjupay.common.domain.BusinessException;
import com.shanjupay.common.domain.CommonErrorCode;
import com.shanjupay.common.domain.RestErrorResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@ControllerAdvice
public class GlobalExceptionHandler {

        private final static Logger LOGGER =
LoggerFactory.getLogger(GlobalExceptionHandler.class);

        //捕获异常后处理方法
    @ResponseBody
    @ExceptionHandler(value = Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public RestErrorResponse processExcetion(HttpServletRequest request, HttpServletResponse
response,Exception e){

        //如果是自定义异常则直接取出异常信息
        if(e instanceof BusinessException){
            LOGGER.info(e.getMessage(),e);
            BusinessException businessException = (BusinessException) e;
            ErrorCode errorCode = businessException.getErrorCode();
            return new
RestErrorResponse(errorCode.getDesc(),String.valueOf(errorCode.getCode()));

        }
        LOGGER.error("系统异常：",e);
        return new
RestErrorResponse(CommonErrorCode.UNKNOWN.getDesc(),String.valueOf(CommonErrorCode.UNKNOWN.getCod
e()));
    }
```

```
}
```

2.4.5.5 抛出自定义异常

按照异常处理流程，应用层抛出自定义异常由异常处理器进行解析。

1、校验验证码接口抛出 BusinessException

修改商户平台应用工程中SmsServicer的verificationMessageCode接口

```
public void checkVerifyCode(String verifyKey, String verifyCode) throws BusinessException;
```

接口实现中抛出异常自定义异常类型

```
/**
     * 校验验证码，抛出异常则校验无效
     *
     * @param verifyKey  验证码key
     * @param verifyCode 验证码
     */
    @Override
    public void checkVerifyCode(String verifyKey, String verifyCode) throws BusinessException {
        //实现校验验证码的逻辑
        String url = smsUrl+"/verify?
name=sms&verificationCode="+verifyCode+"&verificationKey="+verifyKey;
        Map responseMap = null;
        try {
            //请求校验验证码
            ResponseEntity<Map> exchange = restTemplate.exchange(url, HttpMethod.POST,
HttpEntity.EMPTY, Map.class);
            responseMap = exchange.getBody();
            log.info("校验验证码，响应内容：{}",JSON.toJSONString(responseMap));
        } catch (Exception e) {
            e.printStackTrace();
            log.info(e.getMessage(),e);
            throw new BusinessException(CommonErrorCode.E_100102);
//            throw new RuntimeException("验证码错误");
        }

        if(responseMap == null || responseMap.get("result")==null || !(Boolean)
responseMap.get("result")){
            throw new BusinessException(CommonErrorCode.E_100102);
//            throw new RuntimeException("验证码错误");
        }

    }
```

2、测试

请求商户注册，输出一个错误的验证码，返回信息如下：

```
{
    "errCode": "100102",
    "errMessage": "验证码错误"
}
```

3、测试不可预知异常

故意在Controller中制造异常，测试是否抛出未知错误异常。

代码如下：

```
@PostMapping("/merchants/register")
public MerchantRegisterVO registerMerchant(@RequestBody MerchantRegisterVO merchantRegister){
    int i=1/0;//故意制造异常
    ....
```

请商户注册，返回信息如下：

```
{
    "errMessage": "未知错误",
    "errCode": "999999"
}
```

2.4.6 校验商户手机号

2.4.6.1 实现思路

校验商户手机号的唯一性，根据商户的手机号查询商户表，如果存在记录则说明已有相同的手机号重复，手机号不唯一则抛出异常自定义异常。

2.4.6.2 完善代码

1、修改商户服务注册商户接口，添加抛出异常声明

```
/**
 * 商户注册
 * @return
 */
MerchantDTO createMerchant(MerchantDTO merchantDTO) throws BusinessException;
```

2、修改商户服务注册商户接口实现方法

```
@Transactional
public MerchantDTO createMerchant(MerchantDTO merchantDTO) throws BusinessException{
    // 1.校验
    if (merchantDTO == null) {
        throw new BusinessException(CommonErrorCode.E_100108);
    }
    //手机号非空校验
```

```
    if (StringUtils.isBlank(merchantDTO.getMobile())) {
        throw new BusinessException(CommonErrorCode.E_100112);
    }
    //校验手机号的合法性
    if (!PhoneUtil.isMatches(merchantDTO.getMobile())) {
        throw new BusinessException(CommonErrorCode.E_100109);
    }
    //联系人非空校验
    if (StringUtils.isBlank(merchantDTO.getUsername())) {
        throw new BusinessException(CommonErrorCode.E_100110);
    }
    //密码非空校验
    if (StringUtils.isBlank(merchantDTO.getPassword())) {
        throw new BusinessException(CommonErrorCode.E_100111);
    }

    //校验商户手机号的唯一性,根据商户的手机号查询商户表，如果存在记录则说明已有相同的手机号重复
    LambdaQueryWrapper<Merchant> lambdaQryWrapper = new LambdaQueryWrapper<Merchant>()
            .eq(Merchant::getMobile,merchantDTO.getMobile());
    Integer count = merchantMapper.selectCount(lambdaQryWrapper);
    if(count>0){
        throw new BusinessException(CommonErrorCode.E_100113);
    }
    //将dto转成entity
    Merchant entity = MerchantCovert.INSTANCE.dto2entity(merchantDTO);
    //设置审核状态0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝
    entity.setAuditStatus("0");
    //保存商户信息
    merchantMapper.insert(entity);
    //将entity转成 dto
    MerchantDTO merchantDTONew = MerchantCovert.INSTANCE.entity2dto(entity);
    return merchantDTONew;
}
```

3、修改商户应用平台注册商户接口

添加对注册信息的非空校验。

```
@PostMapping("/merchants/register")
public MerchantRegisterVO registerMerchant(@RequestBody MerchantRegisterVO merchantRegister){
    // 1.校验
    if (merchantRegister == null) {
        throw new BusinessException(CommonErrorCode.E_100108);
    }
    //手机号非空校验
    if (StringUtils.isBlank(merchantRegister.getMobile())) {
        throw new BusinessException(CommonErrorCode.E_100112);
    }
    //校验手机号的合法性
    if (!PhoneUtil.isMatches(merchantRegister.getMobile())) {
        throw new BusinessException(CommonErrorCode.E_100109);
```

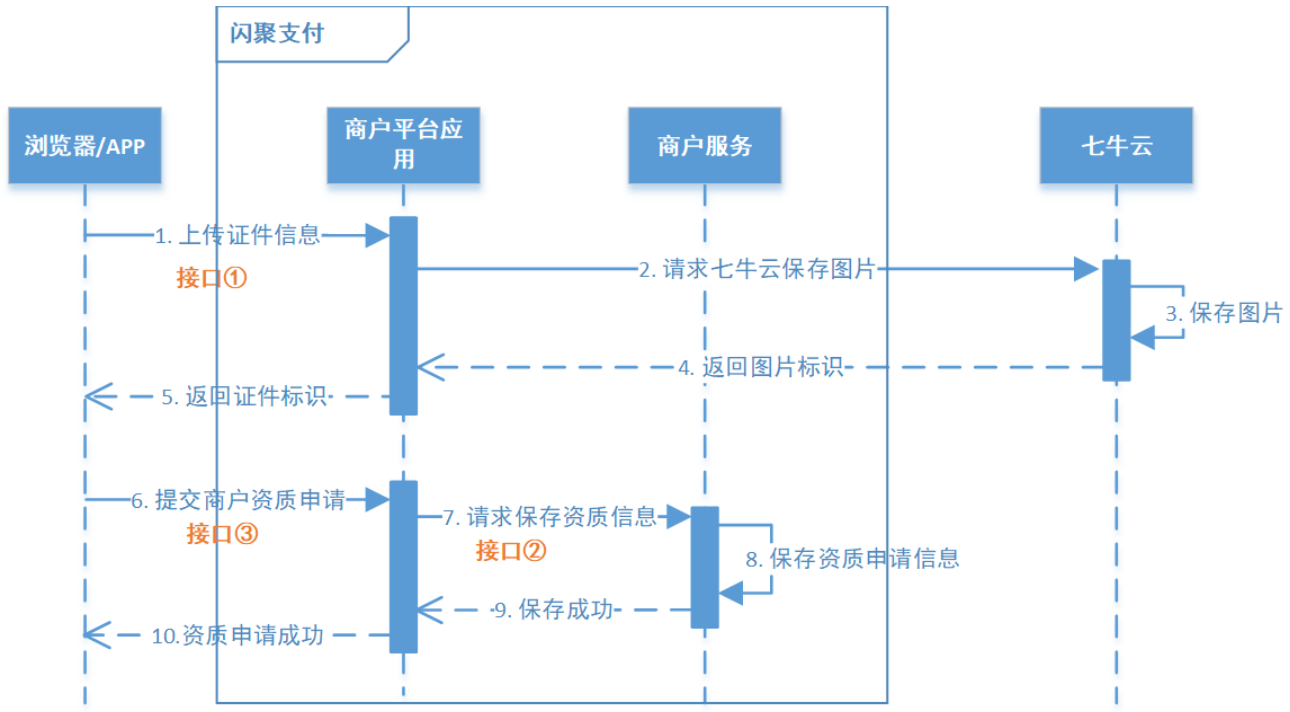
```
    }
    //联系人非空校验
    if (StringUtils.isBlank(merchantRegister.getUsername())) {
        throw new BusinessException(CommonErrorCode.E_100110);
    }
    //密码非空校验
    if (StringUtils.isBlank(merchantRegister.getPassword())) {
        throw new BusinessException(CommonErrorCode.E_100111);
    }
    //验证码非空校验
    if (StringUtils.isBlank(merchantRegister.getVerifiyCode()) ||
StringUtils.isBlank(merchantRegister.getVerifiykey())) {
        throw new BusinessException(CommonErrorCode.E_100103);
    }

    //校验验证码
    smsService.verificationMessageCode(merchantRegister.getVerifiykey(),
merchantRegister.getVerifiyCode());
    //注册商户
    MerchantDTO merchantDTO = MerchantRegisterConvert.INSTANCE.vo2dto(merchantRegister);
    merchantService.createMerchant(merchantDTO);
    return merchantRegister;
}
```

3 商户资质申请

3.1 需求分析

3.1.1 系统交互流程

商户资质申请交互流程如下：



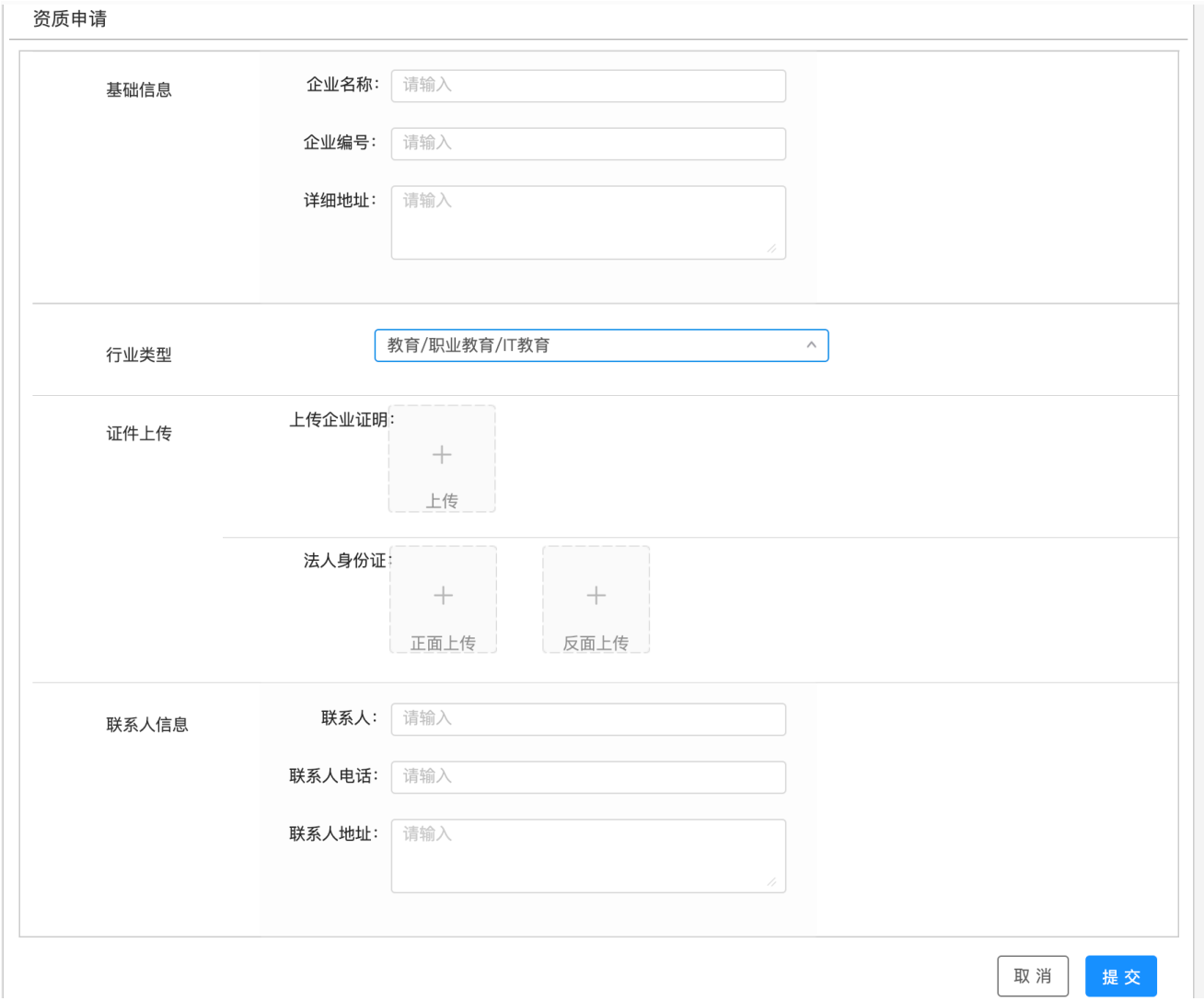
七牛云：

七牛云是国内一家云服务提供商，本系统与七牛云对接将商户资质照片存储在七牛云。

交互流程如下：

1. 前端上传证件照片，请求商户平台应用。
2. 商户平台应用请求七牛云上传图片。
3. 上传成功返回图片标识给前端。
4. 前端携带证件图片标识和资质申请信息提交到商户平台应用。
5. 请求商户服务保存资质申请。
6. 保存成功返回给前端。

商户资质申请界面如下：



3.1.2 资质信息存储

商户资质 信息存储在商户表，上传的资质证件照片存储Url绝对路径。

关于资质 申请状态 说明如下 ：

1、提交资质申请，审核状态 为1（已申请待审核）

2、资质审核后，审核状态 为2（审核通过 ）或3（审核不通过 ）。

商户表

	Field	Type	Comment	
🔑	ID	bigint(20) NOT NULL	主键	
	MERCHANT_NAME	varchar(50) NULL	商户名称	资质申请时完善
	MERCHANT_NO	bigint(20) NULL	企业编号	资质申请时完善
	MERCHANT_ADDRESS	varchar(255) NULL	企业地址	资质申请时完善
	MERCHANT_TYPE	varchar(50) NULL	商户类型	资质申请时完善
	BUSINESS_LICENSES_IMG	varchar(100) NULL	营业执照（企业证明）	资质申请时完善
	ID_CARD_FRONT_IMG	varchar(100) NULL	法人身份证正面照片	资质申请时完善
	ID_CARD_AFTER_IMG	varchar(100) NULL	法人身份证反面照片	资质申请时完善
	USERNAME	varchar(50) NULL	联系人姓名	资质申请时完善
	MOBILE	varchar(50) NULL	联系人手机号(关联统一账号)	注册时写入
	CONTACTS_ADDRESS	varchar(255) NULL	联系人地址	资质申请时完善
	AUDIT_STATUS	varchar(20) NULL	审核状态 0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝	注册成功默认为0
	TENANT_ID	bigint(20) NULL	租户ID,关联统一用户	关联SaaS

3.2 七牛云

商户上传证件照片要与七牛云对接，参考“七牛云使用指南.pdf”完成对接。

3.3 上传证件

3.3.1 商户平台应用证件上传(接口①)

3.3.1.1 接口定义

1、接口描述

1）前端携带证件信息请求商户平台应用

2）商户平台应用请求七牛云服务上传证件图片

3）七牛云返回图片地址给前端

2、接口定义如下：

定义FileService负责文件上传至七牛云：

```
package com.shanjupay.merchant.service;

import org.springframework.web.multipart.MultipartFile;

/**
 * <P>
 * 文件服务
 * </p>
 *
 */
public interface FileService {
    /**
     * 上传文件
     * @param bytes  文件字节
     * @param fileName 文件名称
     * @return 文件下载路径
     * @throws BatchUpdateException
     */
    public String upload(byte[] bytes,String fileName) throws BatchUpdateException;
}
```

在MerchantController定义upload负责接收前端上传证件的请求。

```
    @ApiOperation("证件上传")
    @PostMapping("/upload")
    public String upload(@ApiParam(value = "上传的文件", required = true) @RequestParam("file")
MultipartFile file) {
        return null;
    }
```

3.3.1.2 接口实现

使用提供的七牛云工具类完成图片上传到七牛云

1、在shanjupay-common工程引入

```
<!-- 七牛云SDK -->
<dependency>
  <groupId>com.qiniu</groupId>
  <artifactId>qiniu-java-sdk</artifactId>
</dependency>
```

2、编写工具类

在common工程中编写七牛云上传工具类，此工具类被FileService调用。

```
public class QiniuUtils {

    private static final Logger LOGGER = LoggerFactory.getLogger(QiniuUtils.class);
```

```
    //工具方法，上传文件
    public static void upload2Qiniu(String accessKey, String secretKey, String bucket, byte[]
bytes,
                                    String fileName){
        //构造一个带指定 Region 对象的配置类
        Configuration cfg = new Configuration(Region.huanan());
//...其他参数参考类注释
        UploadManager uploadManager = new UploadManager(cfg);
//默认不指定key的情况下，以文件内容的hash值作为文件名，这里建议由自己来控制文件名
        String key = fileName;
            //通常这里得到文件的字节数组
            Auth auth = Auth.create(accessKey, secretKey);
            String upToken = auth.uploadToken(bucket);
            try {
                Response response = uploadManager.put(bytes, key, upToken);
                //解析上传成功的结果
                DefaultPutRet putRet = new Gson().fromJson(response.bodyString(),
DefaultPutRet.class);
                System.out.println(putRet.key);
                System.out.println(putRet.hash);
            } catch (QiniuException ex) {
                Response r = ex.response;
                LOGGER.error(r.toString());
                try {
                    LOGGER.error(r.bodyString());
                } catch (QiniuException e) {
                    e.printStackTrace();
                }
                throw new RuntimeException(r.toString());
            }
    }
}
```

2、FileServiceImpl

在nacos配置七牛云上传参数：

```
oss:
  qiniu:
    url: ""
    accessKey: ""
    secretKey: ""
    bucket: ""
```

FileServiceImpl上传证件实现方法如下：

```
package com.shanjupay.merchant.service;

import com.shanjupay.common.domain.BusinessException;
import com.shanjupay.common.domain.CommonErrorCode;
import com.shanjupay.common.util.QiniuUtils;

import lombok.extern.slf4j.Slf4j;
```

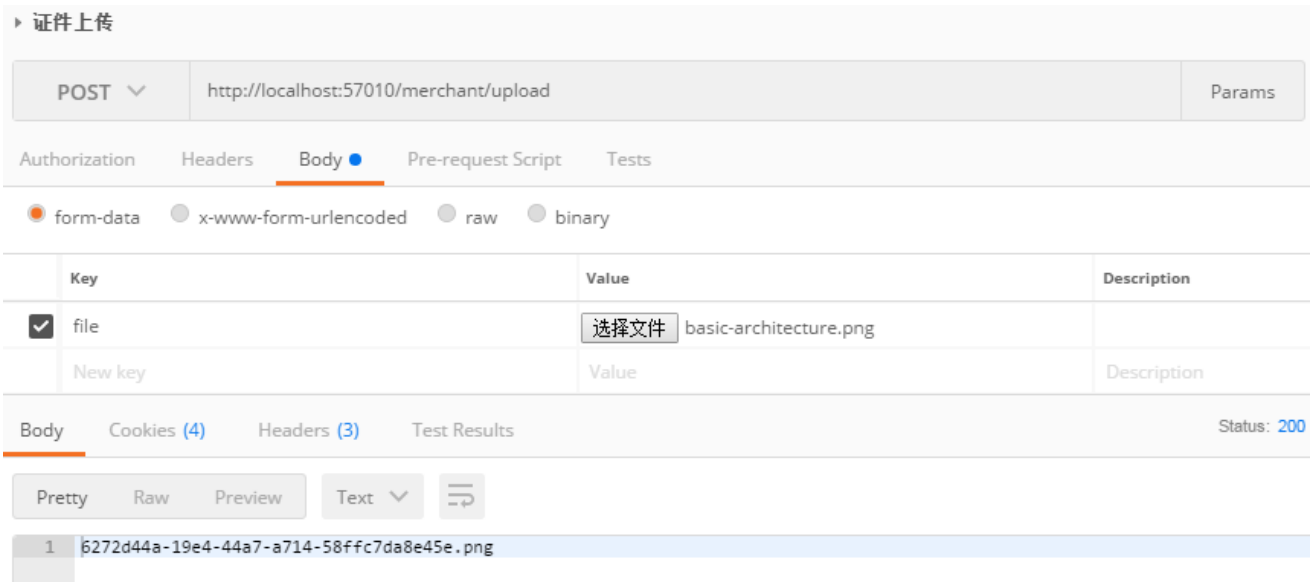
```
import org.apache.commons.lang.StringUtils;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.util.UUID;

@Slf4j
@Service
public class FileServiceImpl implements FileService {

    @Value("${oss.qiniu.url}")
    private String qiniuUrl;
    @Value("${oss.qiniu.accessKey}")
    private String accessKey;
    @Value("${oss.qiniu.secretKey}")
    private String secretKey;
    @Value("${oss.qiniu.bucket}")
    private String bucket;

      @Override
    public String upload(byte[] bytes,String fileName) throws BatchUpdateException {

        try {
            QiniuUtils.upload2Qiniu(accessKey,secretKey,bucket,bytes,fileName);
        } catch (Exception e) {
            e.printStackTrace();
            throw new BusinessException(CommonErrorCode.E_100106);
        }
        //返回文件名称
        return qiniuUrl+fileName;
    }

   }
```

3、MerchantController

```
  @Autowired
  private FileService fileService;

    @ApiOperation("证件上传")
    @PostMapping("/upload")
    public String upload(@ApiParam(value = "上传的文件", required = true) @RequestParam("file")
MultipartFile file) {
         //原始文件名称
        String originalFilename = file.getOriginalFilename();
        //文件后缀
        String suffix = originalFilename.substring(originalFilename.lastIndexOf(".")-1);
        //文件名称
        String fileName = UUID.randomUUID().toString()+suffix;

        //上传文件，返回文件下载url
```

```
        String fileurl = fileService.upload(file.getBytes(), fileName);
        return fileurl;
    }
```

3.3.1.3 接口测试



2、上传成功，登录七牛云查询图片是否上传成功

3.4 资质申请

3.4.1 商户服务资质申请(接口②)

3.4.1.1 接口定义

1、接口描述

1）接收资质申请信息，更新商户信息及审核状态（待审核）

2）返回结果

2、接口定义如下：

在 MerchantService中定义applyMerchant接口

```
/**
     * 资质申请
     * @param merchantId 商户id
     * @param merchantDTO 资质申请信息
     * @throws BusinessException
     */
    public void applyMerchant(Long merchantId,MerchantDTO merchantDTO) throws BusinessException;
```

3.4.1.2 接口实现

实现MerchantServiceImpl 中的applyMerchant方法

```
    @Override
    @Transactional
    public void applyMerchant(Long merchantId,MerchantDTO merchantDTO) throws BusinessException{
        //接收资质申请信息，更新到商户表
        if(merchantDTO == null || merchantId == null){
            throw new BusinessException(CommonErrorCode.E_100108);
        }
        //根据id查询商户
        Merchant merchant = merchantMapper.selectById(merchantId);
        if(merchant == null){
            throw new BusinessException(CommonErrorCode.E_200002);
        }
        Merchant merchant_update = MerchantCovert.INSTANCE.dto2entity(merchantDTO);
        merchant_update.setAuditStatus("1");//已申请待审核
        merchant_update.setTenantId(merchant.getTenantId());//租户id
        //更新
        merchantMapper.updateById(merchant_update);
    }
```

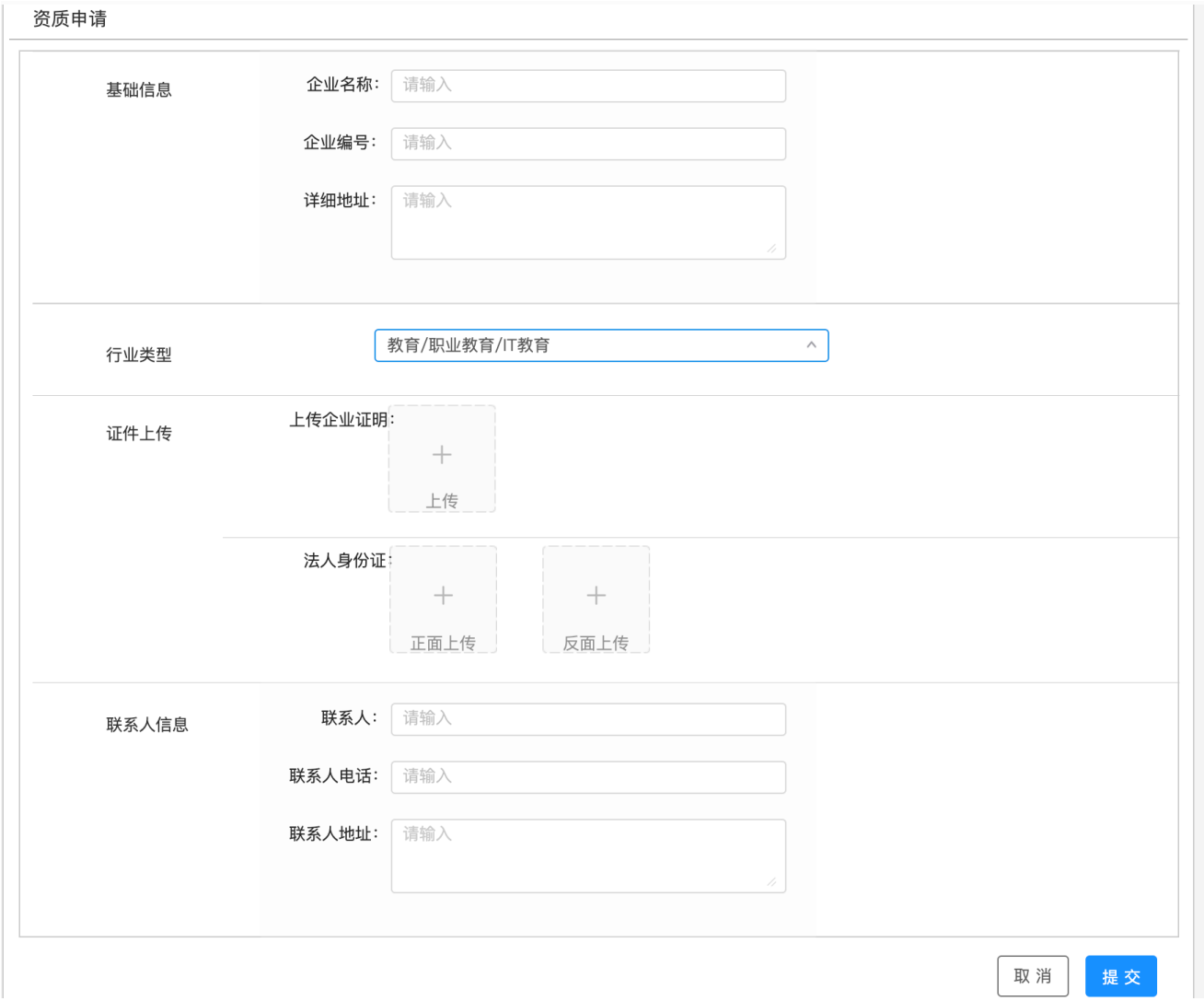
3.2.4 商户平台应用资质申请(接口③)

3.2.4.1 接口定义

1、接口描述

1）商户登录闪聚支付平台

2）商户上传证件，填写资质信息



3）请求商户平台应用进行资质申请

4）商户平台应用请求商户服务完成资质申请

5）返回结果

2、接口定义如下：

根据原型编写商户资质申请VO：MerchantDetailVO

```
package com.shanjupay.merchant.vo;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;

@ApiModel(value = "MerchantDetailVO", description = "商户资质申请信息")
@Data
```

```
public class MerchantDetailVO implements Serializable {

    @ApiModelProperty("企业名称")
    private String merchantName;

    @ApiModelProperty("企业编号")
    private String merchantNo;

    @ApiModelProperty("企业地址")
    private String merchantAddress;

    @ApiModelProperty("行业类型")
    private String merchantType;

    @ApiModelProperty("营业执照")
    private String businessLicensesImg;

    @ApiModelProperty("法人身份证正面")
    private String idCardFrontImg;

    @ApiModelProperty("法人身份证反面")
    private String idCardAfterImg;

    @ApiModelProperty("联系人")
    private String username;

    @ApiModelProperty("联系人地址")
    private String contactsAddress;

}
```

在MerchantController中定义saveMerchant

```
    @ApiOperation("商户资质申请")
    @ApiImplicitParams({
            @ApiImplicitParam(name = "merchantInfo", value = "商户认证资料", required = true,
dataType = "MerchantDetailVO", paramType = "body")
    })
    @PostMapping("/my/merchants/save")
    public void saveMerchant(@RequestBody MerchantDetailVO merchantInfo) {

  }
```

3.2.4.2 获取商户身份

1、商户登录临时方案

因前期未实现登录功能，故目前手动指定的商户ID生成Token（用户登录后的身份令牌），将Token配置在前端，前端拥有了token则说明该商户Id对应的商户登录成功。

商户登录及身份解析流程如下：

1）前端携带token访问商户平台应用。

2）商户平台应用解析token取出商户id

2、生成token

拷贝“资料”-->"代码"下的TokenTemp.java到商户平台应用的test下。

代码如下：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class TokenTemp {

    @Autowired
    MerchantService merchantService;

    @Test
    public void createTestToken() {
        Long merchantId = 1209826678635798530L;//填写用于测试的商户id
        MerchantDTO merchantDTO = merchantService.queryMerchantById(merchantId);
        JSONObject token = new JSONObject();
        token.put("mobile", merchantDTO.getMobile());
        token.put("user_name", merchantDTO.getUsername());
        token.put("merchantId", merchantId);

        String jwt_token = "Bearer " +
EncryptUtil.encodeBase64(JSON.toJSONString(token).getBytes());
        System.out.println(jwt_token);
    }
}
```

向merchantId中设置商户id，运行此测试方法。（运行测试之前停止商户平台应用服务）

3、暂时使用工具类从请求中获取Token并解析

从“资料”--》“代码”文件夹拷贝“util（模拟token）”目录下的SecurityUtil及相关类到商户平台应用工程的util包下

3.2.4.3 资质申请实现

1）编写对象转换类

```
@Mapper
public interface MerchantDetailConvert {

    MerchantDetailConvert INSTANCE = Mappers.getMapper(MerchantDetailConvert.class);

    MerchantDTO vo2dto(MerchantDetailVO vo);

    MerchantDetailVO dto2vo(MerchantDTO dto);
}
```

2）编写MerchantController中的saveMerchant方法

前端携带Token请求此方法，在此方法中需要解析token获取当前商户的Id。

```
    @ApiOperation(value="资质申请")
    @PostMapping("/my/merchants/save")
    @ApiImplicitParam(value = "资质申请信息",name = "merchantDetailVO",required = true,dataType =
"MerchantDetailVO",paramType = "body")
    public void saveMerchant(@RequestBody MerchantDetailVO merchantDetailVO){
        //解析token得到商户id
        Long merchantId = SecurityUtil.getMerchantId();
        MerchantDTO merchantDTO = MerchantDetailConvert.INSTANCE.vo2dto(merchantDetailVO);
        //资质申请
        merchantService.applyMerchant(merchantId,merchantDTO);
    }
```

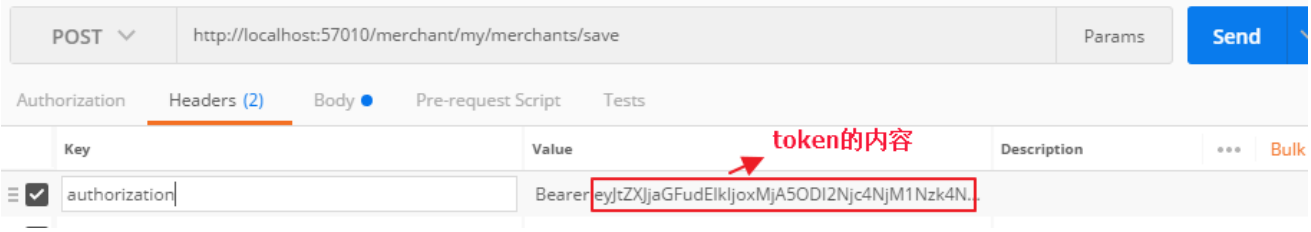
3.2.4.2 接口测试

1、生成token

运行createTestToken测试方法生成临时token。

在Header中添加 ：key:authorization value：token

例子如下 ：



注意：token内容前边固定添加 “Bearer ”(后边一个空格)

2、上传证件，获取证件标识

参考证件上传测试。

3、资质申请

请求数据：

```
{
  "merchantName": "学生餐厅",
  "merchantNo": "32321321312",
  "merchantType": "餐饮",
  "merchantAddress": "郑州梧桐创业大厦",
  "contactsAddress": "郑州梧桐街",
  "businessLicensesImg": "6272d44a-19e4-44a7-a714-58ffc7da8e45e.png",
  "idCardAfterImg": "6272d44a-19e4-44a7-a714-58ffc7da8e45e.png",
  "idCardFrontImg": "6272d44a-19e4-44a7-a714-58ffc7da8e45e.png",
  "username":"张先生"
}
```


测试截图：