万信金融 第2章 讲义-开户

1 前端环境搭建

本项目不涉及前端开发的知识，课件中会把项目所用到的前端工程直接发给大家使用，但是本项目会讲解企业中前后端开发的具体流程和集成测试。

前端工程使用了uni-app框架， uni-app 是一个使用 Vue.js 开发跨平台应用的前端框架，开发者编写一套代码，可编译到iOS、 Android、 H5、小程序等多个平台上运行。

1.1 安装HBuilder X

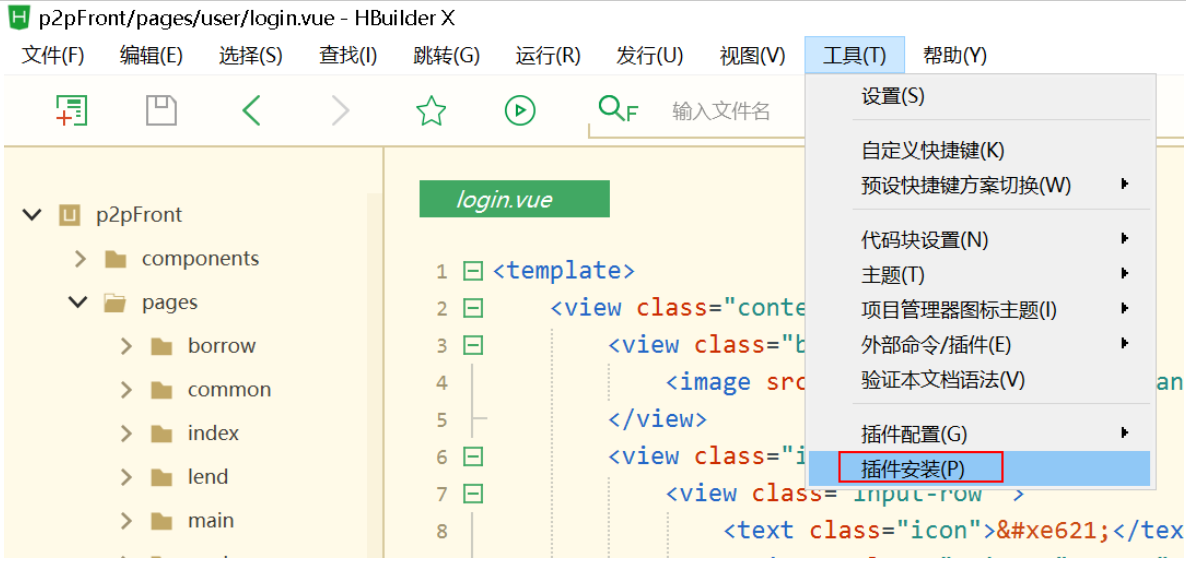
1) 请自行在本机安装谷歌的chrome浏览器

2) 请自行在本机安装HBuilder X (找到压缩包，解压后双击运行HBuilderX.exe即可)

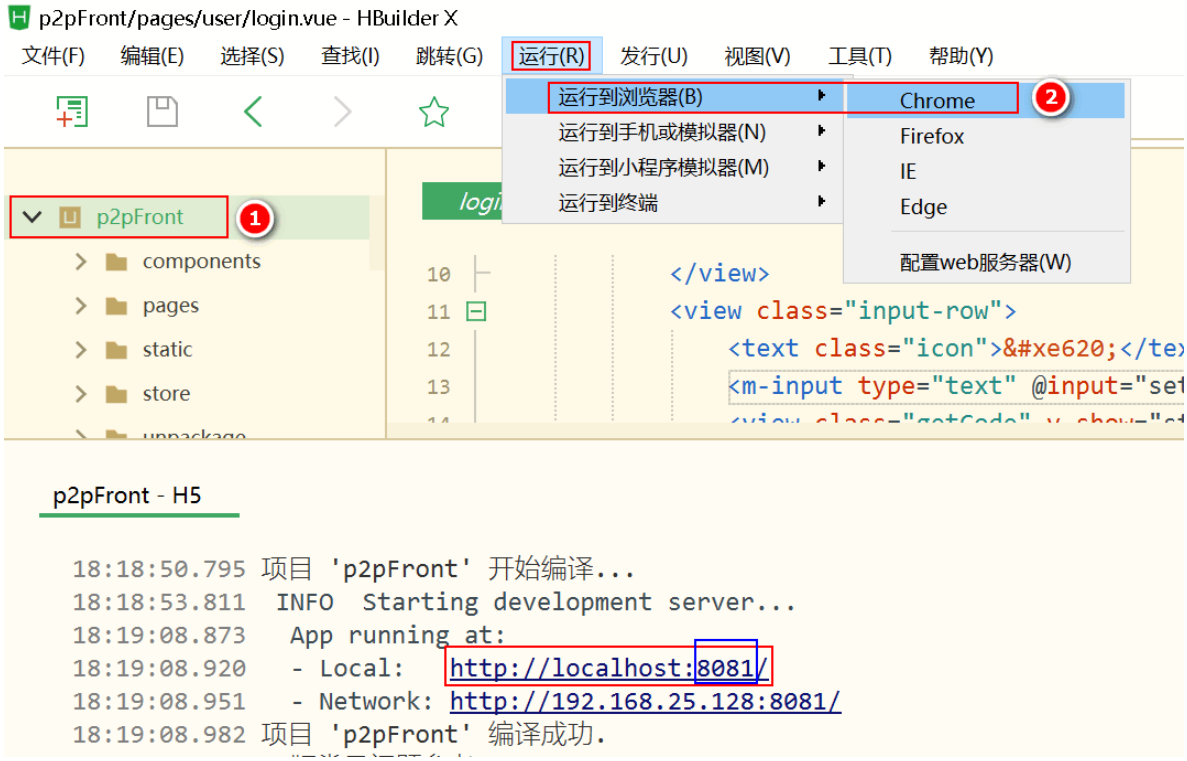
1.2 配置并运行前端工程

1) 用HBuilder X打开老师提供的前端工程wanxinp2p_front

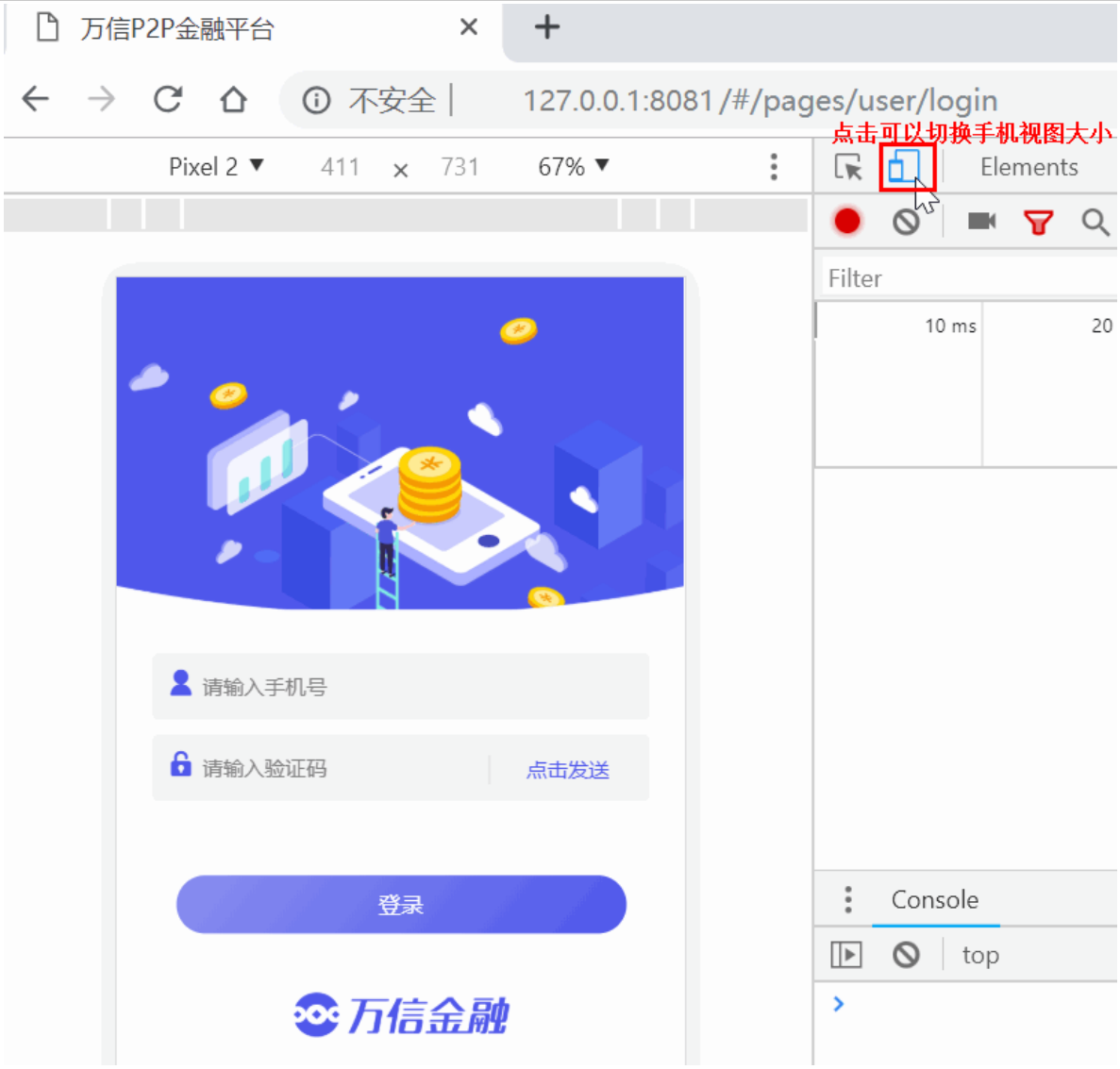
2) 参考下面两图安装一个插件



3) 参考下图，先鼠标点击项目名，然后运行



4）在浏览器里通过上图中的地址访问前端，并按下F12调出开发者工具



2 前后端开发步骤与编码规范

本项目是基于前后端分离的架构进行开发，前后端分离架构总体上包括前端和服务端，通常是多人或多团队协作并行开发，开发步骤如下：

1、需求分析

梳理用户的需求，分析业务流程

2、接口定义

根据需求分析定义接口

3、服务端和前端并行开发

前后端开发人员依据接口进行开发。

服务端开发人员根据接口实现业务功能，每个功能开发完毕后，需要通过Swagger或Postman进行功能测试。前端开发人员制作用户操作界面，然后请求服务端接口完成业务处理。

4、前后端集成测试

服务端业务功能开发完毕后，一般会先通过Swagger或Postman等工具进行测试，然后再和前端进行集成测试。

每个公司都有自己的编码规范，例如：阿里巴巴就公开了自己的《Java开发手册》，包含编程规约、异常日志、单元测试、安全规约、MySQL 数据库、工程结构、设计规约七个维度的内容。 为了码出高效， 撸出质量，咱们针对P2P项目也制定了一个编码规范，请查阅资料文件夹中的"万信金融p2p项目开发规范.pdf", 特别是里面的第八项。

3 需求概述

3.1 什么是开户

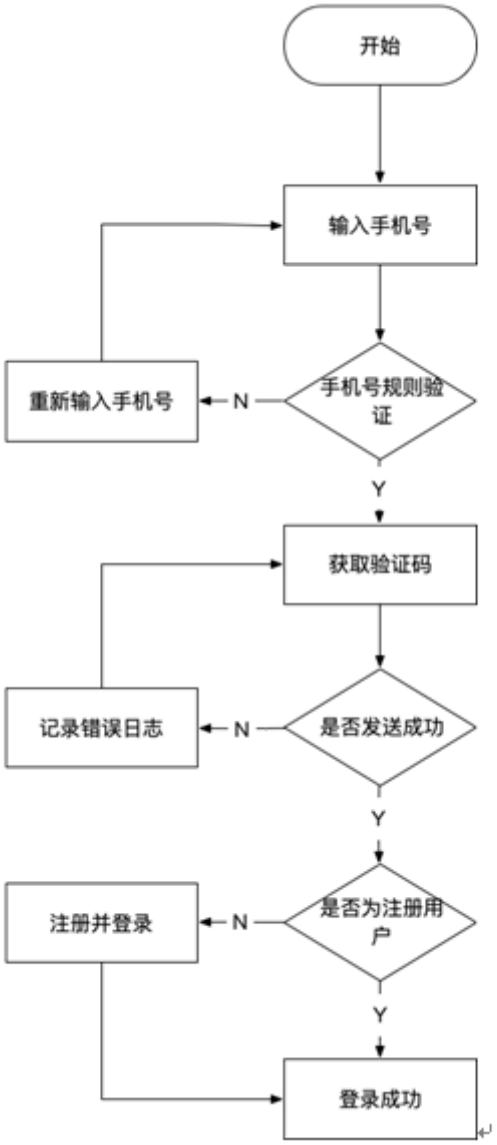
开户是指借款用户和投资用户在交易前都需要在银行存管系统开通个人存管账户，在开户前借款用户和投资用户还需要在万信金融平台注册为平台的用户，本章节讲解从用户注册到用户开户的整个过程。

在开户流程中银行存管系统是一个很重要的系统，它是当前P2P平台最常见的一种模式，为了保证资金不流向P2P平台，由银行存管系统去管理借款用户和投资用户的资金，P2P平台与银行存管系统进行接口交互为借款用户和投资用户搭建交易的桥梁，它们之间的关系如下：

3.2 业务流程

3.2.1 用户注册与登录

注册与登录流程如下：



1、用户打开客户端界面

2、输入手机号，点击“点击发送”按钮， 系统向手机发送验证码

3、手机收到验证码，输入验证码

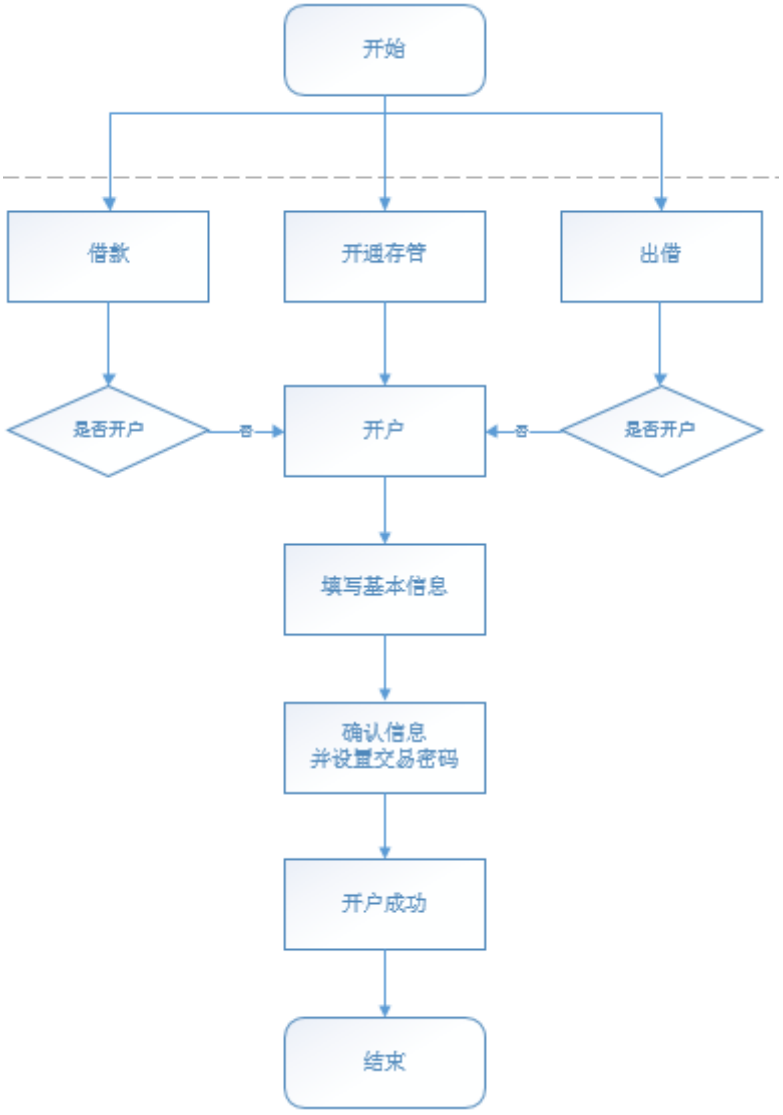
4、点击登录

4.1 如果你是没有注册过的用户，则先自动注册用户并登录成功

4.2 如果你是已经注册过的用户，则直接进行登录

3.2.2 用户开户

用户开户流程如下：



1、进入开户界面

借款人或投资人在平台交易前平台会校验是否开户，如果未开户自动进入开户界面；

借款人或投资人也可以在首页点击“开通存管”

2、填写开户信息

在开户界面填写开户信息

3、确认信息并设置交易密码

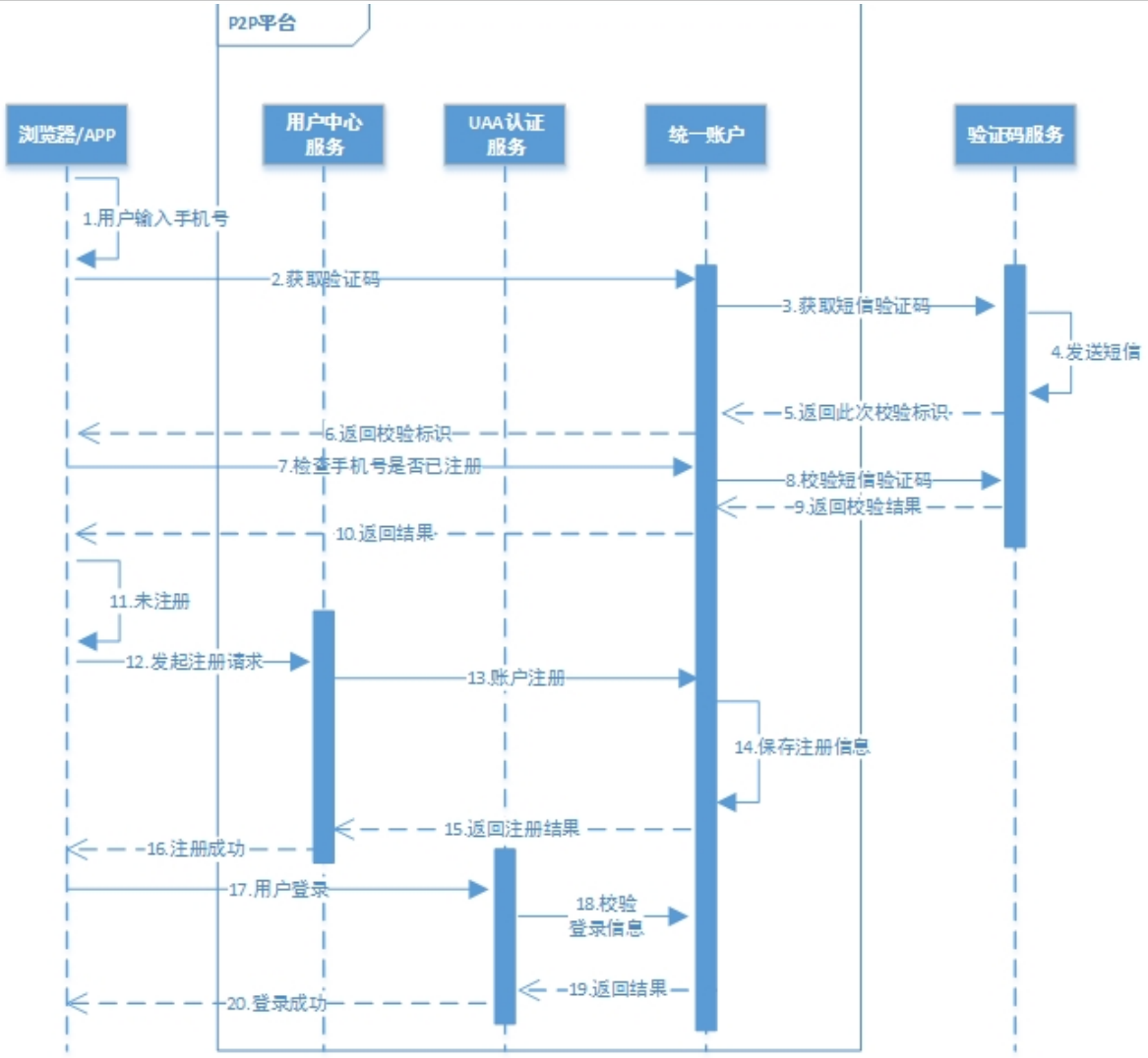
填写信息完成后点击“确认协议并注册 ”，确认开户是否正确，同时需要设置交易密码。

4、开户成功

4 用户注册

4.1 需求分析

用户注册功能交互流程如下：



用户注册功能具体是在网关服务、用户中心服务、统一账户服务、验证码服务几个微服务之间进行交互，各个微服务介绍如下：

- 网关服务：前端发送的所有请求都必须经过网关服务，才能到达后端微服务，网关是一个无处不在的服务，用来保护后端微服务，只有经过它过滤、认证和鉴权才能访问后端微服务。
- 用户中心服务：为借款人和投资人提供用户信息管理服务，包括：注册、开户、充值、提现等。
- 统一账户服务：对借款人和投资人的登录平台账号进行管理，包括：注册账号、账号权限管理等。
- 验证码服务：提供短信、邮件、图片等各种验证码的生成的校验服务。

用户注册功能的具体交互流程：

1）前端请求统一账户服务获取短信验证码

2）前端校验手机号是否存在，校验验证码是否正确，如果不存在则说明未注册

3）前端发起注册请求，请求用户中心服务

4）用户中心服务请求统一账户服务保存注册信息

5）用户中心服务保存用户信息

6）注册成功

4.2 搭建统一账户服务

由于基础工程中并没有提供统一账户服务，所以接下来我们需要搭建统一账户服务(wanxinp2p-account-service)并编码实现相关功能。

1)创建wanxinp2p-account-service（参考基础工程wanxinp2p-consumer-service）

2) 在Apollo中为统一账户服务新建一个配置项目account-service

3) 为account-service添加配置(参考consumer-service)

4) 创建AccountController，编写hello方法，启动服务进行测试

4.3 部署验证码微服务

参考：“验证码服务使用指南.pdf”

4.4 注册1：实现发送短信验证码

4.4.1 接口定义

在统一账号服务中定义获取验证码接口：

1、接口描述如下：

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（成功或失败）

2、接口定义如下：

在wanxinp2p-api工程中新建AccountAPI接口，在该接口中定义getSMSCode方法：

```
/**
 * 获取手机验证码
 * @param mobile 手机号
 * @return
 */
RestResponse getSMSCode(String mobile);
```

在wanxinp2p-account-service工程中新建AccountController类，并实现AccountAPI接口：

```
@Slf4j
@Api(value = "统一账号服务", tags = "Account", description = "统一账号服务API")
@RestController
public class AccountController implements AccountAPI {
    @ApiOperation("获取手机验证码")
    @ApiImplicitParam(name = "mobile", value = "手机号", dataType = "String")
    @GetMapping("/sms/{mobile}")
    public RestResponse getSMSCode(@PathVariable String mobile) {
    }
}
```

4.4.2 功能实现

1）SmsService

在wanxinp2p-account-service工程中新建SmsService类，调用验证码服务发送验证码。

```
@Service
public class SmsService {
```

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

    @Value("${sms.enable}")
    private Boolean smsEnable;


    /**
     * 获取短信验证码
     * @param mobile
     * @return
     */
    public RestResponse getSmsCode(String mobile) {
        if (smsEnable) {
            return OkHttpUtil.post(smsURL + "/generate?effectiveTime=300&name=sms",
                                                    "{\"mobile\":" + mobile +
"}");
        }
        return RestResponse.success();
    }
}
```

2）AccountService

在wanxinp2p-account-service工程中新建AccountService接口，定义getSMSCode方法：

```
public interface AccountService{

    /**
     * 获取手机验证码
     * @param mobile 手机号
     * @return
     */
    RestResponse getSMSCode(String mobile);
}
```

定义该接口的实现类AccountServiceImpl，注入SmsService并实现获取短信验证码的功能：

```
@Slf4j
@Service
public class AccountServiceImpl implements AccountService {

    @Autowired
    private SmsService smsService;

    @Override
    public RestResponse getSMSCode(String mobile) {
        return smsService.getSmsCode(mobile);
    }
}
```

3）AccountController

补充controller代码，注入AccountService获得短信验证码：

```
@Slf4j
```

```
@Api(value = "统一账号服务", tags = "Account", description = "统一账号服务API")
@RestController
public class AccountController implements AccountAPI {

    @Autowired
    private AccountService accountService;

    @ApiOperation("获取手机验证码")
    @ApiImplicitParam(name = "mobile", value = "手机号", dataType = "String")
    @GetMapping("/sms/{mobile}")
    @Override
    public RestResponse getSMSCode(@PathVariable String mobile) {
        return accountService.getSMSCode(mobile);
    }
}
```

4）功能测试

启动wanxinp2p-account-service微服务，使用Swagger或Postman进行功能测试

4.5 注册2：校验手机号和验证码

4.5.1 Mybatis-Plus入门

用户注册功能涉及到操作数据库，本项目的数据访问层采用的是Mybatis-Plus框架，请参考"Mybatis-Plus讲义.pdf"。

4.5.2 接口定义

在统一账号服务中定义校验手机号接口：

1、接口描述如下：

1）校验验证码是否正确

2）如果验证码正确则校验手机号是否存在

2、接口定义如下：

在wanxinp2p-api工程中的AccountAPI接口里面定义checkMobile方法：

```
/**
 * 校验手机号和验证码
 * @param mobile 手机号
 * @param key 校验标识
 * @param code 验证码
 * @return
 */
RestResponse<Integer> checkMobile(String mobile, String key, String code);
```

在wanxinp2p-account-service工程中的AccountController类里面实现checkMobile方法：

```
@ApiOperation("校验手机号和验证码")
@ApiImplicitParams({@ApiImplicitParam(name = "mobile", value = "手机号", required
= true,
                                      dataType = "String"),
@ApiImplicitParam(name = "key", value = "校验标识", required = true, dataType =
"String"),
@ApiImplicitParam(name = "code", value = "验证码", required = true, dataType =
"String")})
@GetMapping(value = "/mobiles/{mobile}/key/{key}/code/{code}")
@Override
public RestResponse<Integer> checkMobile(@PathVariable String mobile,
                                         @PathVariable String key,
                                         @PathVariable String code) {
    return null;
}
```

4.5.3 功能实现

1）SmsService

在SmsService类中定义verifySmsCode方法，调用验证码服务实现校验功能：

```
/**
 * 校验验证码
 * @param key 校验标识  redis中的键
 * @param code 短信验证码
 */
public void verifySmsCode(String key, String code) {
   if (smsEnable) {
      StringBuilder params = new StringBuilder("/verify?name=sms");
      params.append("&verificationKey=").append(key);
      params.append("&verificationCode=").append(code);
      RestResponse smsResponse = OkHttpUtil.post(smsURL + params, "");
      if (smsResponse.getCode() != CommonErrorCode.SUCCESS.getCode() ||
          smsResponse.getResult().toString().equalsIgnoreCase("false")) {
         throw new BusinessException(AccountErrorCode.E_140152);
      }
   }
}
```

2）引入MP相关的依赖

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-typehandlers-jsr310</artifactId>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

3）修改Apollo上关于MP的配置：mybatis-plus.typeAliasesPackage 和 mybatis-plus.mapper-locations

在启动类上添加@MapperScan("cn.itcast.wanxinp2p.account.mapper") //设置mapper接口的扫描包

4）实体类

新建一个实体类Account:

```
@Data
@TableName("account")
public class Account implements Serializable {

    private static final long serialVersionUID = 1L;

    /**
     * 主键
     */
    @TableId("ID")
    private Long id;

    /**
     * 用户名
     */
    @TableField("USERNAME")
    private String username;

    /**
     * 手机号
     */
    @TableField("MOBILE")
    private String mobile;

    /**
     * 密码
     */
    @TableField("PASSWORD")
    private String password;

    /**
     * 加密盐
     */
    @TableField("SALT")
    private String salt;

    /**
     * 账号状态
     */
    @TableField("STATUS")
    private Integer status;

   /**
    * 域(c：c端用户；b：b端用户)
    */
   @TableField("DOMAIN")
```

```
    private String domain;

}
```

5）Mapper

新建一个Mapper接口，继承Mybatis-Plus的BaseMapper接口：

```
public interface AccountMapper extends BaseMapper<Account> {

}
```

新建一个Mapper配置文件：

```
<?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="cn.itcast.wanxinp2p.account.mapper.AccountMapper">

</mapper>
```

6）AccountService

在AccountService接口中定义checkMobile方法：

```
/**
 * 校验手机号和验证码
 * @param mobile
 * @param key
 * @param code
 * @return
 */
Integer checkMobile(String mobile, String key, String code);
```

注意：要修改该接口去继承Mybatis-Plus的IService接口

在AccountServiceImpl类中实现checkMobile方法：

```
@Override
public Integer checkMobile(String mobile, String key, String code) {
   smsService.verifySmsCode(key,code);
        QueryWrapper<Account> wrapper=new QueryWrapper<>();
        //wrapper.eq("mobile",mobile);
        wrapper.lambda().eq(Account::getMobile,mobile);
        int count=count(wrapper);
        return count > 0 ? 1 : 0;
}
```

注意：要修改该实现类去继承Mybatis-Plus的ServiceImpl

7）AccountController

补充controller代码，调用accountService实现校验功能：

```
@ApiOperation("校验手机号和验证码")
@ApiImplicitParams({@ApiImplicitParam(name = "mobile", value = "手机号", required
= true,
                                      dataType = "String"),
@ApiImplicitParam(name = "key", value = "校验标识", required = true, dataType =
"String"),
@ApiImplicitParam(name = "code", value = "验证码", required = true, dataType =
"String")})
@GetMapping(value = "/mobiles/{mobile}/key/{key}/code/{code}")
@Override
public RestResponse<Integer> checkMobile(@PathVariable String mobile,
@PathVariable String key,
                                         @PathVariable String code) {
   return RestResponse.success(accountService.checkMobile(mobile, key, code));
}
```

8）功能测试

重启相关服务，使用Swagger或Postman进行功能测试

备注：

1. **如果在测试过程中遇到"java.sql.SQLException: The server time zone..."错误，请自行参考资料文件夹中**

的"mysql时区报错.txt"进行解决。

2. **请把资料文件夹中的“GlobalExceptionHandler.java”复制到account微服务工程中，以实现全局异常处理**

4.6 注册3：保存账号信息

如果手机号是不存在的，证明这是新用户，接下来需要保存用户账号相关信息，从而最终实现注册功能。这里有两个微服务都需要进行账号信息保存，一个是Consumer微服务(用户中心)，一个是Account微服务(统一账户)，Consumer微服务会把数据保存到p2p_consumer数据库的consumer表中，Account微服务会把数据保存到p2p_account数据库的account表中。

此处我们采用了用户、账号分离设计，这样设计的好处是，当用户的业务信息发生变化时，不会影响到认证、授权等系统机制的运行，但是需要做用户和账号的数据同步工作。

前端页面发起的请求会先到达Consumer微服务，然后远程调用Account微服务，从而分别实现各自的信息保存，详情请参考“注册登录流程图”。

4.6.1 统一账号服务实现保存功能

4.6.1.1 接口定义

1、接口描述如下：

保存账号信息

2、接口定义如下：

在AccountAPI接口中定义register方法：

```
/**
 * 用户注册
 * @param accountRegisterDTO
 * @return
 */
RestResponse<AccountDTO> register(AccountRegisterDTO accountRegisterDTO);
```

在AccountController类中实现register方法：

```
@ApiOperation("用户注册")
@ApiImplicitParam(name = "accountRegisterDTO", value = "账户注册信息", required =
true,
                  dataType = "AccountRegisterDTO", paramType = "body")
@PostMapping(value = "/1/accounts")
public RestResponse<AccountDTO> register(@RequestBody AccountRegisterDTO
accountRegisterDTO) {
}
```

4.6.1.2 功能实现

1）AccountService

在AccountService接口中定义register方法：

```
/**
 * 账户注册
 * @param registerDTO 注册信息
 * @return
 */
AccountDTO register(AccountRegisterDTO registerDTO);
```

在AccountServiceImpl类中实现register方法，通过Mybatis-Plus实现数据保存

```
/**
 * 账户注册
 * @param registerDTO 注册信息
 * @return
 */
@Override
//这里需要注意DTO和DO的区别，请参考“万信金融p2p项目开发规范.pdf”
public AccountDTO register(AccountRegisterDTO registerDTO) {
        Account account = new Account();
        account.setUsername(registerDTO.getUsername());
        account.setMobile(registerDTO.getMobile());
        account.setPassword(PasswordUtil.generate(registerDTO.getPassword()));
        if (smsEnable) {
            account.setPassword(PasswordUtil.generate(account.getMobile()));
        }
        account.setDomain("c");
        account.setStatus(StatusCode.STATUS_OUT.getCode());
        save(account);
        return convertAccountEntityToDTO(account);
}
/**
 * entity转为dto
```

```
     * @param entity
     * @return
     */
    private AccountDTO convertAccountEntityToDTO(Account entity) {
        if (entity == null) {
            return null;
        }
        AccountDTO dto = new AccountDTO();
        BeanUtils.copyProperties(entity, dto);
        return dto;
    }
```

2）AccountController

补充controller代码，调用accountService实现注册功能

```
@ApiOperation("用户注册")
@ApiImplicitParam(name = "accountRegisterDTO", value = "账户注册信息", required =
true,
                  dataType = "AccountRegisterDTO", paramType = "body")
@PostMapping(value = "/1/accounts")
@Override
public RestResponse<AccountDTO> register(@RequestBody AccountRegisterDTO
accountRegisterDTO) {
   return RestResponse.success(accountService.register(accountRegisterDTO));
}
```

由于在注册功能中，Account服务会被Consumer服务调用，所以等到下面Consumer服务代码编写完毕后统一进行测试。

4.6.2 用户中心服务实现保存功能

4.6.2.1 接口定义

1、接口描述如下：

1）在用户中心保存用户信息

2）请求统一账号服务实现账户信息保存

2、接口定义如下：

在wanxinp2p-api工程中新建ConsumerApi接口，在该接口中定义register方法

```
public interface ConsumerApi {
   /**
    * 用户注册
    * @param consumerRegisterDTO
    * @return
    */
   RestResponse register(ConsumerRegisterDTO consumerRegisterDTO);
}
```

在ConsumerController类中实现register方法：

```
@ApiOperation("用户注册")
@ApiImplicitParam(name = "consumerRegisterDTO", value = "用户注册", required =
true,
                  dataType = "AccountRegisterDTO", paramType = "body")
@PostMapping(value = "/consumers")
public RestResponse register(@RequestBody ConsumerRegisterDTO
consumerRegisterDTO){

}
```

4.6.2.2 功能实现

1）由于涉及到要在Consumer微服务中操作数据库，所以需要先把MP的环境搞定，请参考“注册2：校验手机号和验证码”章节内容。

2）新建一个实体类Consumer:

```
@Data
@TableName("consumer")
public class Consumer implements Serializable {

    private static final long serialVersionUID = 1L;

    /**
     * 主键
     */
    @TableId(value = "ID", type = IdType.AUTO)
    private Long id;

    /**
     * 用户名
     */
    @TableField("USERNAME")
    private String username;

    /**
     * 真实姓名
     */
    @TableField("FULLNAME")
    private String fullname;

    /**
     * 身份证号
     */
    @TableField("ID_NUMBER")
    private String idNumber;

    /**
     * 用户编码,生成唯一,用户在存管系统标识
     */
    @TableField("USER_NO")
    private String userNo;

    /**
     * 平台预留手机号
     */
    @TableField("MOBILE")
```

```
    private String mobile;

    /**
     * 用户类型,个人or企业，预留
     */
    @TableField("USER_TYPE")
    private String userType;

    /**
     * 用户角色.借款人or投资人
     */
    @TableField("ROLE")
    private String role;

    /**
     * 存管授权列表
     */
    @TableField("AUTH_LIST")
    private String authList;

    /**
     * 是否已绑定银行卡
     */
    @TableField("IS_BIND_CARD")
    private Integer isBindCard;

    /**
     * 可用状态
     */
    @TableField("STATUS")
    private Integer status;

    /**
     * 可贷额度
     */
    @TableField("LOAN_AMOUNT")
    private BigDecimal loanAmount;

    /**
     * 请求流水号
     */
    @TableField("REQUEST_NO")
    private String requestNo;

}
```

3）Mapper

新建一个Mapper接口，继承Mybatis-Plus的BaseMapper：

```
public interface ConsumerMapper extends BaseMapper<Consumer> {

}
```

新建一个Mapper配置文件：

```
<?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="cn.itcast.wanxinp2p.consumer.mapper.ConsumerMapper">

</mapper>
```

4）ConsumerService

新建ConsumerService接口，并定义checkMobile和register方法

```
public interface ConsumerService extends IService<Consumer> {
    /**
     * 检测用户是否存在
     * @param mobile
     * @return
     */
    Integer checkMobile(String mobile);
    /**
     * 用户注册
     * @param consumerRegisterDTO
     * @return
     */
    void register(ConsumerRegisterDTO consumerRegisterDTO);
}
```

新建ConsumerServiceImpl类实现该接口的checkMobile和register方法

```
@Service
public class ConsumerServiceImpl extends ServiceImpl<ConsumerMapper, Consumer>
implements ConsumerService {
    @Override
    public Integer checkMobile(String mobile) {
        return getByMobile(mobile, false) != null ? 1 : 0;
    }

    /**
     * 根据手机号获取用户信息
     * @param mobile 手机号
     * @param throwEx 不存在是否抛出异常
     * @return
     */
    private ConsumerDTO getByMobile(String mobile, Boolean throwEx) {
        Consumer entity = getOne(new QueryWrapper<Consumer>().lambda()
                                                .eq(Consumer::getMobile,  mobile));
        return convertConsumerEntityToDTO(entity);
    }

    @Override
    public void register(ConsumerRegisterDTO consumerRegisterDTO) {
       //检测是否已注册
       if (checkMobile(consumerRegisterDTO.getMobile()) == 1) {
          throw new BusinessException(ConsumerErrorCode.E_140107);
       }

       Consumer consumer=new Consumer();
```

```
        BeanUtils.copyProperties(consumerRegisterDTO, consumer);
        consumer.setUsername(CodeNoUtil.getNo(CodePrefixCode.CODE_NO_PREFIX));

 consumer.setUserNo(CodeNoUtil.getNo(CodePrefixCode.CODE_REQUEST_PREFIX));
        consumer.setIsBindCard(0);
       //保存用户信息
       save(consumer);
    }

    /**
     * entity转为dto
     * @param entity
     * @return
     */
    private ConsumerDTO convertConsumerEntityToDTO(Consumer entity) {
        if (entity == null) {
            return null;
        }
        ConsumerDTO dto = new ConsumerDTO();
        BeanUtils.copyProperties(entity, dto);
        return dto;
    }
}
```

5）ConsumerController

补充controller代码，调用consumerService实现注册功能

```
@Override
@ApiOperation("用户注册")
@ApiImplicitParam(name = "consumerRegisterDTO", value = "注册信息", required =
true,
                  dataType = "AccountRegisterDTO", paramType = "body")
@PostMapping(value = "/consumers")
public RestResponse register(@RequestBody ConsumerRegisterDTO
consumerRegisterDTO) {
   consumerService.register(consumerRegisterDTO);
   return RestResponse.success();
}
```

6）远程调用统一账户中心微服务

1. 在agent包中创建代理AccountApiAgent

```
@FeignClient(value = "account-service")
public interface AccountApiAgent {
    /**
     * 用户注册
     * @param accountRegisterDTO
     * @return
     */
    @PostMapping(value = "/account/1/accounts")
    RestResponse<AccountDTO> register(@RequestBody AccountRegisterDTO
                                                          accountRegisterDTO);

}
```

2. 补充service代码，发起远程调用

```
... ...
save(consumer);
AccountRegisterDTO accountRegisterDTO = new AccountRegisterDTO();
BeanUtils.copyProperties(consumerRegisterDTO, accountRegisterDTO);
RestResponse<AccountDTO> restResponse = accountApiAgent
                                        .register(accountRegisterDTO);
if (restResponse.getCode() != CommonErrorCode.SUCCESS.getCode()) {
  throw new BusinessException(ConsumerErrorCode.E_140106);
}
```

3. 启动类上添加@EnableFeignClients(basePackages = {"cn.itcast.wanxinp2p.consumer.agent"})

7）功能测试

启动相关服务，使用Swagger或Postman进行功能测试

请把资料文件夹中的"GlobalExceptionHandler.java"复制到consumer微服务工程中，以实现全局异常处理

4.7 前后端集成测试

1. 在Apollo中对网关服务增加路由配置

2. 启动前端工程
3. 启动Apollo
4. 后端需要启动：
   - wanxinp2p-discover-server微服务
   - wanxinp2p-gateway-server微服务
   - wanxinp2p-account-service微服务
   - wanxinp2p-consumer-service微服务
   - 短信验证码服务（依赖redis）
5. 在浏览器中访问登录页面

注意：所有前端到后端的请求都经过网关微服务(gateway-server)，它是无处不在的，但是很多图中并没有把网关体现出来，并不代表它不存在，而是因为它一直都存着，是一个基础组件，没必要在每个图中都画出来。

5 用户登录

5.1 传统实现方式

注册功能搞定后，紧接着就该实现登录功能了。这个功能对于大家来说应该是很熟悉的，业务熟悉，代码也熟悉，传统实现方式的思路：查询数据库，确定账号和密码是否存在(正确)。

1. 接口定义

   在AccountAPI接口中定义登录方法login：

```
/**
     * 用户登录
     * @param accountLoginDTO  封装登录请求数据
     * @return
*/
RestResponse<AccountDTO> login(AccountLoginDTO accountLoginDTO);
```

   在AccountController中实现login方法：

```
@ApiOperation("用户登录")
@ApiImplicitParam(name = "accountLoginDTO",  value = "登录信息",  required =
true,
                                    dataType = "AccountLoginDTO",  paramType
= "body")
@PostMapping(value = "/1/accounts/session")
@Override
public RestResponse<AccountDTO> login(@RequestBody AccountLoginDTO
accountLoginDTO) {
   return null;
}
```

2. 功能实现

   在AccountService中新增登录接口login：

```
/**
 登录功能
 @param accountLoginDTO 封装登录请求数据
 @return 用户及权限信息
*/
AccountDTO login(AccountLoginDTO accountLoginDTO);
```

   在AccountServiceImpl类中实现login方法：

```
@Override
public AccountDTO login(AccountLoginDTO accountLoginDTO) {
   Account account = null;
   if (accountLoginDTO.getDomain().equalsIgnoreCase("c")) {
      account = getAccountByMobile(accountLoginDTO.getMobile());//获取c端用户
   } else {
      account = getAccountByUsername(accountLoginDTO.getUsername());//获取b端
用户
   }
   if (account == null) {
```

```
            throw new BusinessException(AccountErrorCode.E_130104); // 用户不存在
        }
        AccountDTO accountDTO = convertAccountEntityToDTO(account);
        if (smsEnable) {// 如果smsEnable=true，说明是短信验证码登录，不做密码校验
            return accountDTO;
        }   //验证密码
        if (PasswordUtil.verify(accountLoginDTO.getPassword(),
account.getPassword())) {
            return accountDTO;
        }
        throw new BusinessException(AccountErrorCode.E_130105);
    }

    /**
         根据手机获取账户信息
         @param mobile 手机号
         @return 账户实体
    */
    private Account getAccountByMobile(String mobile) {
        return getOne(new QueryWrapper<Account>().lambda()
                                            .eq(Account::getMobile,  mobile));
    }

    /**
         根据用户名获取账户信息
         @param username 用户名
         @return 账户实体
    */
    private Account getAccountByUsername(String username) {
        return getOne(new QueryWrapper<Account>().lambda()
                                          .eq(Account::getUsername,  username));
    }
    /**
         entity转为dto
         @param entity对象
         @return dto对象
    */
    private AccountDTO convertAccountEntityToDTO(Account entity) {
        if (entity == null) {
            return null;
        }
        AccountDTO dto = new AccountDTO();
        BeanUtils.copyProperties(entity, dto);
        return dto;
    }
```

3. 完善AccountController代码，调用AccountService完成登录功能：

```
@ApiOperation("用户登录")
@ApiImplicitParam(name = "accountLoginDTO", value = "登录信息", required =
true,
                                    dataType = "AccountLoginDTO", paramType
= "body")
@PostMapping(value = "/l/accounts/session")
@Override
public RestResponse<AccountDTO> login(@RequestBody AccountLoginDTO
accountLoginDTO) {
   return RestResponse.success(accountService.login(accountLoginDTO));
}
```

5.2 问题思考

传统登录实现方式在应付分布式、微服务场景时存在的问题：

1. 每个微服务都要进行登录校验，十分麻烦，我们需要的是单点登录
2. 会话保持问题
3. 认证方式单一，无法适应各种认证场景(扫码，指纹...)，毫无扩展性
4. ... ...

P2P平台作为网络贷款平台，采用了前后端分离、分布式、微服务等架构，这就决定了传统的登录实现方式在这里无法胜任。为了解决这个问题，我们要在P2P平台引入独立的UAA服务。 UAA全称是User Account and Authentication，简称为认证服务，UAA服务使用Spring Security+Oauth2+JWT技术栈实现，结合前面的网关服务(gateway)即可搞定P2P平台的认证和授权业务功能。