HUNAN UNIVERSITY

编程新技术实务

实 验 四 ：	系统登录/注册模块(Android app)的开发
小组成员：	李俊哲 牟可心 廖晨锦
专业班级：	计科 1901
完成日期：	2021 年 1 月 5 日

目录

1 引言.....3
1.1 实验目的.....3
1.2 实验对应知识点.....3
1.3 实验前任务.....3
1.4 实验内容.....3
1.5 实验环境.....4
1.6 术语与缩写解释.....4
2 模块汇总.....4
2.1 模块汇总表.....4
2.2 模块内容.....5
2.3 模块关系图.....6
3 子系统模块设计.....7
3.1 客户端——UI 模块.....7
3.2 客户端——活动模块.....8
3.3 客户端——属性模块.....14
3.4 服务端——application.yml.....14
3.5 服务端——bean 层.....15
3.6 服务端——mapper 层.....17
3.7 服务端——service 层.....18
3.8 服务端——controller 层.....22
3.9 服务端——具体的实现过程举例：find.....24
4 程序运行流程演示.....24
登录界面.....24
注册界面.....25
注册成功界面.....25
忘记密码界面.....26
收到的验证码.....27
修改密码界面.....27
修改密码成功界面.....28
欢迎界面.....28

1 引言

1.1 实验目的

通过使用 Android API 进行系统注册模块的开发，包括前台的 Android 原生 app 以及后台服务模块的开发，要求后台使用 JavaEE 框架实现。

1.2 实验对应知识点

Android 的应用编程框架以及 Android API 的使用，JavaEE 框架的应用开发。

1.3 实验前任务

Java 编程、Android API 的编程、JavaEE 中 JSP、Servlet 的编程。

1.4 实验内容

一、Android app 的开发

对于 Android app，本组实现了如下界面：

（1）登录界面：包含用户名、密码的文字标识以及相应的输入栏，登录以及注册的按钮。当输入用户名以及密码后，点击登录按钮，则交数据提交至后台进行验证，如通过验证则跳转至欢迎界面，否则跳转回登录界面，并提示用户的验证错误原因；当用户点击注册按钮则跳转至注册界面；当用户点击忘记密码文字时若已输入已存在的用户名则跳转至忘记密码界面，否则停留在登录界面。

（2）注册界面：包含用户名、密码以及确认密码的文字标识以及相应的输入栏，提交以及取消按钮。当输入相关信息后，首先验证输入的信息是否符合要求（用户名至少 5 位，最多 10 位，以英文字母开头，只允许包含英文字母、数字以及_，同时必须至少有一个大写英文字母；密码为 6-12 位，只允许包含英文字母、数字和_，同时要求确认密码必须与密码一致），如不符合要求则在界面内提示错误，只有符合要求才提交给后台进行注册操作。如注册成功则跳转至欢迎界面，否则跳转回注册界面并提示用户的注册错误原因；当用户点击取消按钮则返回登录界面。

（3）忘记密码界面：包含手机号、验证码的文字标识以及相应的输入栏，获取验证码、提交及返回按钮。当输入手机号后，点击获取验证码，弹出相关提示，确认后手机号会收到短信验证码。当验证码正确并且输入的手机号与登录界面输入的用户名相匹配时，点击提交按钮才能跳转至修改密码界面，否则跳转回忘记密码界面并提示错误原因。当用户点击返回按钮则返回登录界面。

（4）修改密码界面：包含密码、确认密码的文字标识以及相应的输入栏，密码为 6-12 位，只允许包含英文字母、数字和_，同时要求确认密码必须与密码一致。如不符合要求则在界面内提示错误，只有符合要求才提交给后台进行修改密码操作。如修改成功则跳转至登录界面，否则跳转回修改密码界面并提示出错原因。当用户点击返回按钮则返回登录界面。

（5）欢迎界面：显示对用户的欢迎信息，其中包括用户的用户名、姓名等，点击查看

信息按钮则输出详细信息，点击返回按钮则返回登录界面。

二、后台服务的开发

对于 Android app 调用的后台服务而言，要求使用 JavaEE 中的 JSP 或者 Servlet 进行编写，响应信息可以使用 XML 或 JSON 等方式进行封装，封装的信息由 Android app 再进行解析处理。

对于注册信息的存储，要求使用 Mysql 数据库进行存放，其中用户名作为表的主键进行存储。

1.5 实验环境

相关工具	解释	版本
IntelliJ Idea	服务端开发工具	2021.2.2
Android Studio	客户端开发工具	2020.3.1
阿里云	服务器	轻量级 1 核 2GiB

1.6 术语与缩写解释

缩写、术语	解释
JDK	Java 语言的软件开发工具包，Java Development Kit
Android SDK	安卓软件开发工具包，Android Software Development Tools
Mysql	数据库
SPP	精简并行过程，Simplified Parallel Process
SD	系统设计，System Design
ADT	安卓开发工具，Android Development Tools
HTTP	超文本传送协议，Hypertext transfer protocol
JSON	一种轻量级的数据交换格式，JavaScript Object Notation
@Override	表示子类重写了父类的方法（编译器会校验写的方法在父类中是否存在）
OkHttp3	当前主流的网络请求的开源框架
URL	在 WWW 上，每一信息资源都有统一的且在网上唯一的地址，该地址就叫 URL（Uniform Resource Locator,统一资源定位器），它是 WWW 的统一资源定位标志，就是指网络地址。
URI	URI，统一资源标志符(Uniform Resource Identifier，URI)，表示的是 web 上每一种可用的资源，如 HTML 文档、图像、视频片段、程序等都由一个 URI 进行标识的。

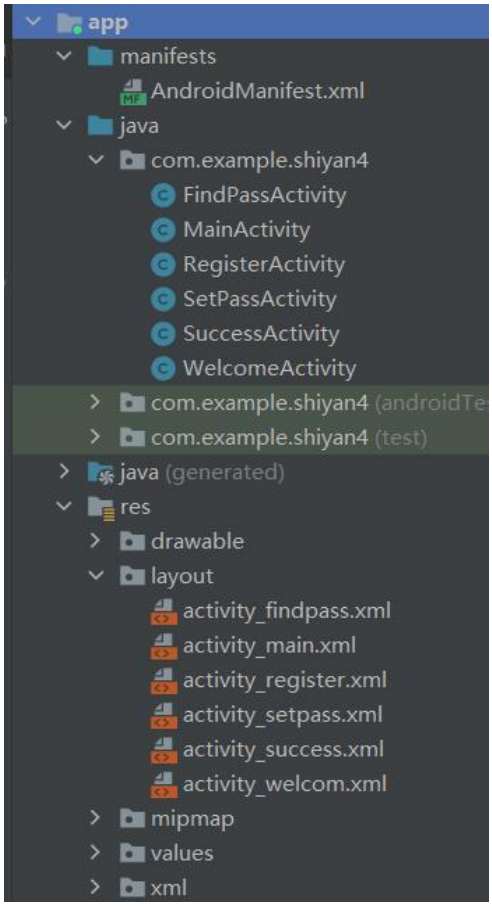
2 模块汇总

2.1 模块汇总表

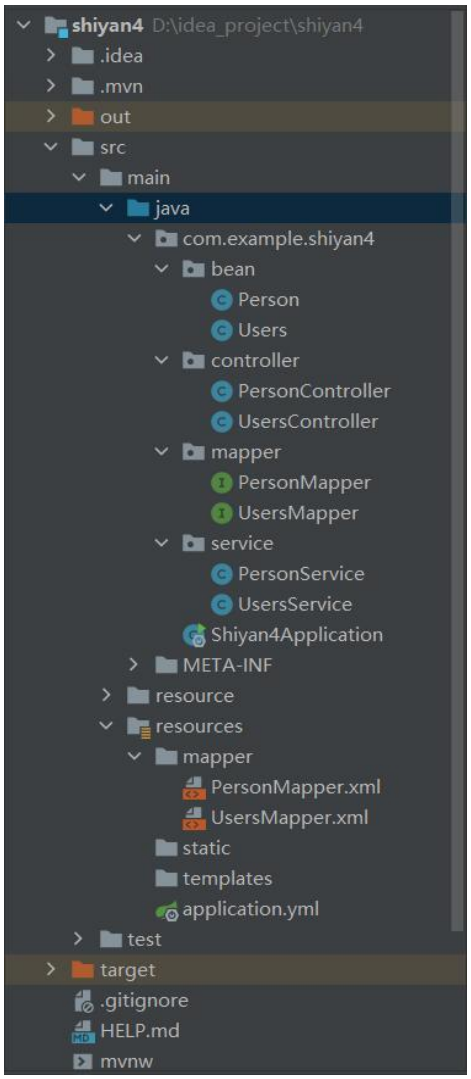
客户端 Client	
模块名称	功能简述
UI 模块(layout)	作为视图层展现给用户
活动模块(main/java)	用户操作的执行代码，其中包括客户端网络通信编程
属性模块(drawable)	存储数据和属性以及需要调用的图片等文件
服务端 Server	
模块名称	功能简述
application.yml	配置文件，方便修改连接数据库的相关信息
bean	封装数据，是数据库中 users 和 person 的映射
controller	实现数据的持久化操作，如增删改查
service	业务逻辑的实现，主要调用 mapper 层的方法
mapper	连接层，实现数据库的连接和配置文件的读取
具体实现过程举例	通过 find 操作解释不同功能的具体实现过程

2.2 模块内容

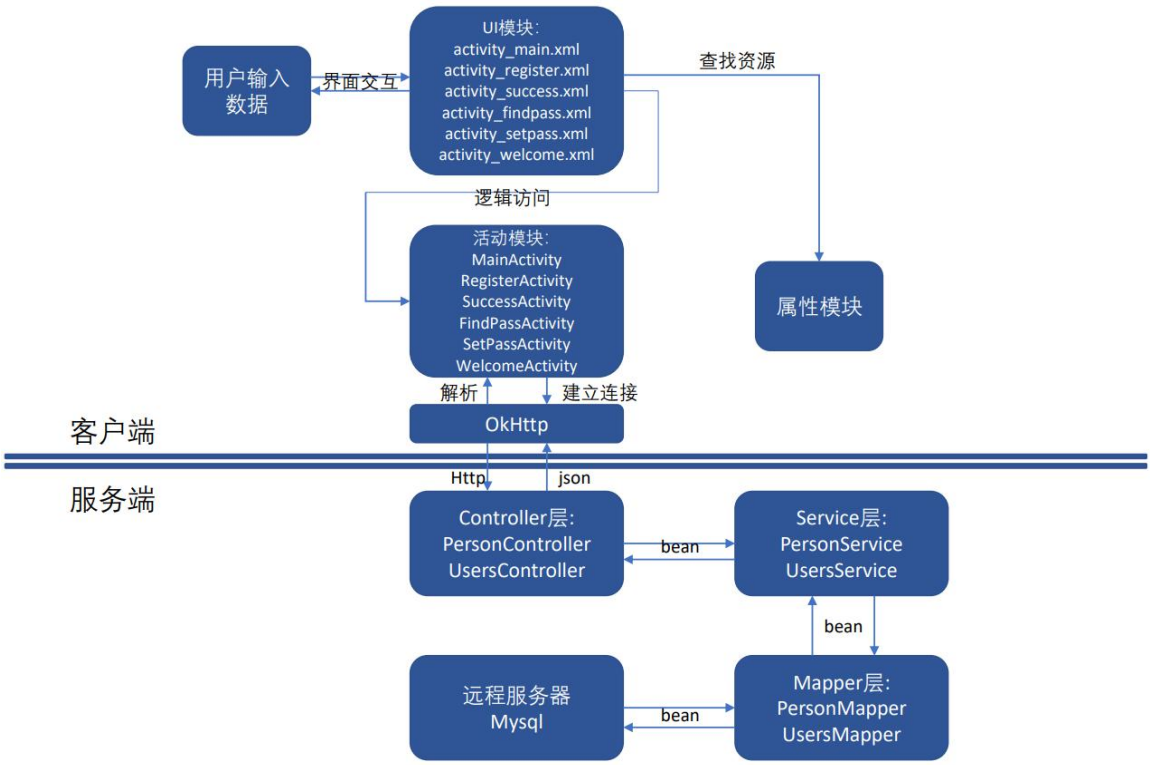
客户端：



服务端：


shiyan4 D:\idea_project\shiyan4
.idea
.mvn
out
src
main
java
com.example.shiyan4
bean
Person
Users
controller
PersonController
UsersController
mapper
PersonMapper
UsersMapper
service
PersonService
UsersService
Shiyan4Application
META-INF
resource
resources
mapper
PersonMapper.xml
UsersMapper.xml
static
templates
application.yml
test
target
.gitignore
HELP.md
mvnw

2.3 模块关系图



3 子系统模块设计

3.1 客户端——UI 模块

模块名称	UI 模块
功能描述	直接显示在屏幕上的视图，可以在其中放置多个控件并且设置其布局方式使其呈现在用户面前。
接口与属性	View 和 ViewGroup 对象创建的用户见面元素是可以和用户互动的，VirwGroup 对象是为了定义布局的接口而保存其他的 View Android 提供一个 View 和 ViewGroup 子类的集合，这个集合能提供相同的输入控制（例如按钮和文本框）和各种各样的布局模式（例如一个线性或者相对布局）。
数据结构与算法	在本设计中，包含五个 XML 文件： **一、登录界面（activity_main.xml）** 主要包含了两个输入栏（EditText），每个输入栏提供了 hint，一个为“用户名”，另一个为“密码”。用户名栏的右侧提供了一个清空按钮，实现了一键清空输入栏的功能。密码栏的右侧提供了一个状态按钮，实现了查看/隐藏密码功能。输入栏的上方显示了一个表示登陆的图标。输入栏的下方提供了注册、登陆按钮。注册、登陆按钮下一行有忘记密码的提示信息，点击忘记密码按钮并且用户名栏有已存在的用户名时可进入忘记密码界面。点击按钮跳转页面是通过 onclick 属性绑定相关的函数，图标、输

	入栏的布局通过使用 constraintlayout 模块来实现，即可拖动模块。 （添加功能：增加了一个限制功能，即对应每个输入栏可以输入的内容 Type 不同，输入时的键盘功能也不同。例如电话号码的输入栏只可以输入数字，而用户名的输入栏可以输入字母、合法符号以及数字。） **二、注册界面（activity_register.xml）** 主要是六个输入栏，都有给出提示信息，分别为用户名、姓名、年龄、电话号码、密码、确认密码栏。输入栏上方是一个表示注册信息的图标。输入栏下方是取消、注册按钮，同样的是利用 onclick 属性绑定相关的函数。 **三、注册/修改密码成功界面（activity_success.xml）** 主要显示一个注册成功（修改成功）的图标，一个“注册成功！”（“修改成功”）的提示文字，以及一个返回登陆的按钮，同样的是利用 onclick 属性绑定相关的函数。 **四、忘记密码界面（activity_findpass.xml）** 主要是三个输入栏，第一个输入栏给出“用户名：”的提示信息，对应用户名的输入；第二个输入栏给出“电话号码：”的提示信息，对应电话号码的输入；第三个输入栏给出“验证码：”的提示信息，对应验证码的输入，验证码栏的右侧提供了一个按钮用来获取验证码。输入栏的上方是图标提示。输入栏下方是提交和取消按钮。 **五、修改密码界面（activity_setpass.xml）** 主要是两个输入栏，都有给出提示信息，分别为密码栏和确认密码栏。输入栏上方是一个表示修改信息的图标。输入栏下方是取消、确认按钮，同样的是利用 onclick 属性绑定相关的函数。 **六、欢迎界面（activity_welcome.xml）** 主要是显示一个欢迎的图标，用户名+“Welcome!”的提示文字。
补充说明	都采用线性布局。 具体按钮绑定的函数，在活动模块中继续讨论。

3.2 客户端——活动模块

模块名称	**活动模块**
功能描述	实现 Android 的客户端与服务端的通信功能。包括异步线程处理，访问服务端，用 http 实现客户端和服务器端的通信等。
接口与属性	Android 目前提供两种 http 通信接口： 标准 Java 接口(java.net) ----HttpURLConnection，可以实现简单的基于 URL 请求、响应功能； Apache 接口(org.appache.http)----HttpClient，相对更大更全能，但是

	速度相对较慢。 几乎在任何项目的代码中我们都能看到这两个类的身影，使用率非常高。不过 HttpURLConnection 和 HttpClient 的用法还是稍微有些复杂的，如果不进行适当封装的话，很容易就会写出不少重复代码。于是乎，一些 Android 网络通信框架也就应运而生，比如 AsyncHttpClient、Volley 等。**本次实验采用的是 OkHttp3 框架，使用 formdata 数据的格式传递参数**。例如：http://47.98.101.147:8081/person/find?username=Alfred 同过接口后面加上”?”，然后 param1=value1，param2=value2.....，其传输的时候就是采用 json 键值对的形式，获取到的内容也是 json 键值对，需要对其进行解析。
数据结构 与算法	HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。 **HTTP 协议的主要特点：** 1. 支持 C/S（客户/服务器）模式。 2. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST，每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。 3. 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。 4. 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。 5. 无状态：HTTP 协议是无状态协议，无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。 **HTTP URL 的格式如下：** http://host[":"port][abs_path] http 表示要通过 HTTP 协议来定位网络资源；host 表示合法的 Internet 主机域名或者 IP 地址；port 指定一个端口号，为空则使用默认端口 80；abs_path 指定请求资源的 URI（Web 上任意的可用资源）。 HTTP 有两种报文分别是请求报文和响应报文，让我们先来看看请求报文。 **HTTP 的请求报文：** 先来看看请求报文的一般格式：

通常来说一个HTTP请求报文由请求行、请求报头、空行、和请求数据4个部分组成。

请求行：

请求行由请求方法，URI字段和HTTP协议的版本组成，格式如下：

Method Request-URI HTTP-Version CRLF

其中 Method 表示请求方法；Request-URI 是一个统一资源标识符；HTTP-Version 表示请求的 HTTP 协议版本；CRLF 表示回车和换行（除了作为结尾的 CRLF 外，不允许出现单独的 CR 或 LF 字符）。

HTTP 请求方法有 8 种：

分别是 GET、POST、DELETE、PUT、HEAD、TRACE、CONNECT 、OPTIONS。其中 PUT、DELETE、POST、GET 分别对应着增删改查，对于移动开发最常用的就是 POST 和 GET 了。

GET：请求获取 Request-URI 所标识的资源

POST：在 Request-URI 所标识的资源后附加新的数据

HEAD：请求获取由 Request-URI 所标识的资源的响应消息报头

PUT： 请求服务器存储一个资源，并用 Request-URI 作为其标识

DELETE ：请求服务器删除 Request-URI 所标识的资源

TRACE ： 请求服务器回送收到的请求信息，主要用于测试或诊断

CONNECT： HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

OPTIONS ：请求查询服务器的性能，或者查询与资源相关的选项和需求

例如我去访问百度请求行是：

GET http://www.baidu.com HTTP/1.1

请求报头：

在请求行之后会有 0 个或者多个请求报头，每个请求报头都包含一个名字和一个值，它们之间用“:”分割。请求头部会以一个空行，发送回车符和换行符，通知服务器以下不会有请求头。关于请求报头，会在后面的消息报头一节做统一的解释。

请求数据：

请求数据不在 GET 方法中使用，而是在 POST 方法中使用。POST 方法适用于需要客户填写表单的场合，与请求数据相关的最常用的请求头是 Content-Type 和 Content-Length。

具体方法：

主要使用到两种 http 请求方法：Get 与 Post 方法。

Post 请求可以向服务器传送数据，而且数据放在 HTML HEADER 内一起传送到服务端 URL 地址，数据对用户不可见。而 Get 是把参数数据队列加到提交的 URL 中，值和表单内各个字段一一对应。

Get 方式：Get 机制用的是在 URL 地址里面通过“?”号间隔，然后以 name=value 的形式给客户端传递参数。所以首先要在 Android 工程下的 onCreate 方法定义好其 URL 地址以及要传递的参数，然后通过 URL 打开一个 HttpURLConnection 链接，此链接可以获得 InputStream 字节流对象，也是往服务端输出和从服务端返回数据的重要过程，而若服务端往 andorid 返回信息时候，就可以通过 InputStreamReader 作转换，将返回来的数据用 BufferReader 显示出来。

Post 方式：Post 传输方式不在 URL 里传递，也解决了 get 传输量小、容易篡改及不安全等一系列不足。主要是通过对 HttpURLConnection 的设置，让其支持 post 传输方式，然后在通过相关属性传递参数（若需要传递中文字符，则可以通过 URLEncoder 编码，而在获取端采用 URLDecoder 解码即可）

具体实现：

活动模块由两部分组成，即控制部分和连接部分。

控制部分实现的功能主要是当用户点击不同的按钮，或是在文本框中输入内容时，产生不同的反馈。即通过对不同函数的调用来实现整个程序的逻辑。

连接部分主要是通过 OkHttp 库将逻辑产生的 http 请求从前端发送给后端，后端再返回一个 json 数据。注意，此处需要新建一个线程来实现请求的发送，然后再将信息通过新建立的子进程传递到主进程，原因是子进程不能直接对用户活动进行反馈。

以 MainActivity 为例，当用户在登录界面的“用户名：”处输入了一个用户名，在“密码：”处输入了一个密码，并点击了登录按钮，则客户端正确的处理逻辑应该是：

第一步：

处理点击事件。

```
@Override
public void onClick(View v) {
    try {
        switch (v.getId()){
            case R.id.button1://登录按钮

verify(String.valueOf(et_username.getText()),String.valueOf(et_pass.getText()));
                break;
        }
```

```
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

第二步：

新创建一个子进程，用来判断账号与密码是否匹配。

```
public void verify(String username,String pass){
    new Thread(){
        @Override
        public void run() {
            try {
                super.run();
                //get 请求
                String url =
"http://47.98.101.147:8081/users/find?username="+username;
                JSONObject jsonresult = get(url);
                //向主线程传递信息 true or false
                Message message = Message.obtain();
                message.what =VAL;
                if(jsonresult == null){
                    message.obj = NOPERSON;
                } else if (!pass.equals(jsonresult.getString("pass"))){
                    message.obj = WRONGPASS;
                } else{
                    message.obj = LOGINSUCCESS;
                }
                handler.sendMessage(message);
            }catch (Exception e){
                e.printStackTrace();
            }
        }
    }.start();
}
```

第三步：

处理 get 请求。

```
public JSONObject get(String url){
    try {
        Request request = new Request.Builder().url(url).build();
        Response response = client.newCall(request).execute();
        JSONObject jsonObject = new JSONObject(response.body().string());
        System.out.println("jsonObject:"+jsonObject);
        return jsonObject;
```

```
    }catch (Exception e){
        e.printStackTrace();
        return null;
    }
}
```

第四步：

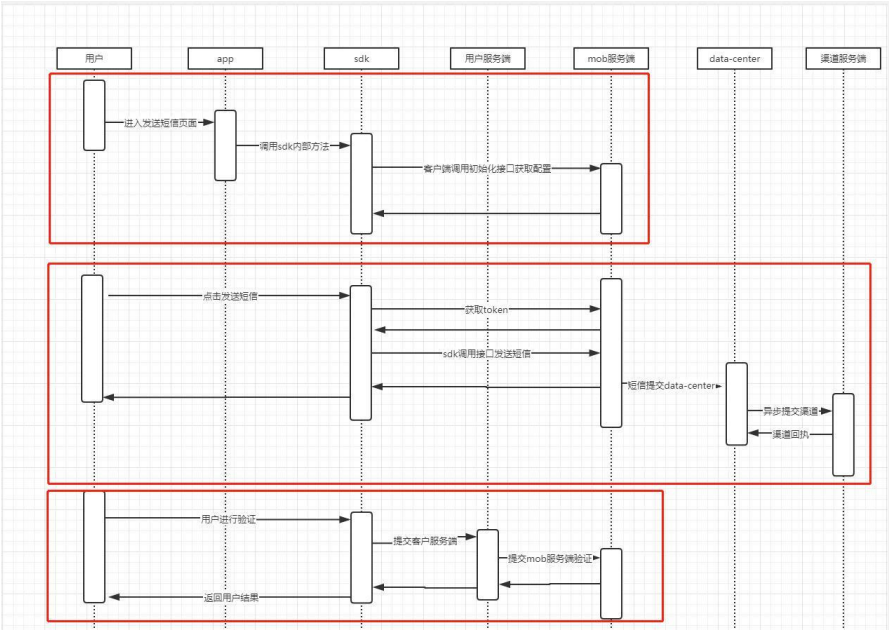
处理 message。

```
private Handler handler = new Handler(){
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        switch (msg.what){
            case VAL:
                int flag = (int) msg.obj;
                if(flag == NOPERSON){
                    Toast toast = Toast.makeText(MainActivity.this,"用户未注册
",Toast.LENGTH_LONG);
                    toast.setGravity(Gravity.TOP,0,150);
                    toast.show();
                } else if(flag == WRONGPASS){
                    Toast toast = Toast.makeText(MainActivity.this,"密码错误
",Toast.LENGTH_LONG);
                    toast.setGravity(Gravity.TOP,0,150);
                    toast.show();
                    et_pass.setText("");
                } else if(flag == LOGINSUCCESS){//登录成功，获取 name，并跳转
页面
                    new Thread(){
                        @Override
                        public void run() {
                            try {
                                super.run();
                                System.out.println("登录成功");
                                JSONObject jsonperson =
get("http://47.98.101.147:8081/person/find?username="+String.valueOf(et_usernam
e.getText()));
                                String name = jsonperson.getString("name");
                                Intent intent = new
Intent(MainActivity.this,WelcomeActivity.class);
                                intent.putExtra("name",name);
                                startActivity(intent);
                            }catch (Exception e){
                                e.printStackTrace();
```

```
                                }
                            }
                        }.start();

                    }
                    break;
                }
            }
        };
```

短信验证码部分的框架：



3.3 客户端——属性模块

模块名称	**属性模块**
功能描述	R.java 文件自动生成，用来定义 Android 程序中所有各类型的资源的索引。（它是只读的，开发人员不对其修改）。
接口与属性	android 工程所有资源信息(组件、图片、字符等等)都是由 HashMap<Integer,Object>来存储的 key 值就是 R.java 中的静态变量值，value 就是相对应的各种对象信息(组件、图片、字符等等)。
数据结构与算法	R.java 文件中默认有 attr、drawable、layout、string 等四个静态内部类，每个静态内部类分别对应着一种资源，如 layout 静态内部类对应 layout 中的界面文件，其中每个静态内部类中的静态常量分别定义一条资源标识符，如 public static final int main=0x7f030000;对应的是 layout 目录下的 main.xml 文件。
补充说明	R.java 及本地资源文件，使用“@+”声明的资源，系统会自动在 R.java

	中创建。

3.4 服务端——application.yml

模块名称	配置文件
功能描述	将一些信息，包括服务器的端口号 8081、mysql 的驱动、mysql 连接的 url 以及 mysql 的账号和密码，保存在配置文件中，方便修改。
接口与属性	采用的是 yml 配置文件，基本形式是 key-value 键值对的形式。

数据结构与算法	server:
	  port: 8081 # 服务器的端口
	spring:
	  # 存储的信息
	  datasource:
	  driver-class-name: com.mysql.cj.jdbc.Driver
	  url:
	jdbc:mysql://localhost:3306/shiyan2?characterEncoding=utf-8&useSSL=false&serverTimezone=UTC
	  username: root
	  password: gfacljz001028
	
	# mybatis
	mybatis:
	  # mapper
	  mapper-locations: classpath:mapper/*.xml

3.5 服务端——bean 层

模块名称	bean
功能描述	封装数据，是数据库中 users 和 person 的映射的实体
接口与属性	含有本身的字段属性，同时有相应的 get 和 set 方法。 对于 person，在构造函数中将 teleno、age 分别初始化为“”和-1

数据结构与算法	public class Users {
	  private String username;# 用户名
	  private String pass;# 密码
	  #构造函数、get、set 方法...
	  public String getUsername() {
	  return username;
	  }
	
	  public void setUsername(String username) {
	  this.username = username;
	  }

```
    public String getPass() {
        return pass;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    public Users(String username, String pass) {
        this.username = username;
        this.pass = pass;
    }
}


public class Person {
    private String username;# 用户名
    private String name;# 姓名
    private String teleno;# 电话号码
    #构造函数、get、set 方法...
    private int age;# 年龄
    {
        teleno = "";
        age = -1;
    }
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getTeleno() {
        return teleno;
    }
```

```
    public void setTeleno(String teleno) {
        this.teleno = teleno;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Person(String username, String name,int age,String teleno) {
        this.username = username;
        this.name = name;
        this.teleno = teleno;
        this.age = age;
    }
}
```

3.6 服务端——mapper 层

模块名称	mapper
功能描述	mapper 层是连接层，负责服务器和数据库的连接，可以对数据库进行直接操作，本实验中实现了增删改查操作。在 UsersMapper 和 PersonMapper 中对增删改查操作进行了接口的定义。然后再 UsersMapper.xml 和 PersonMapper.xml 中进行了实现。
接口与属性	Users 具体体现为 addUsers、deleteUsers、updateUsers、和 findByUsername、findAll，对应增删改查操作。 Person 具体体现为 addPerson_Four、addPerson_Three_age、addPerson_Three_teleno、addPerson_Two，deletePerson，updatePerson，findByUsername、findAll，对应增删改查操作。其中"增"允许两个参数、三个参数和四个参数的增，例如 addPerson_Two 对应的是两个参数:username 和 name，addPerson_Three_age 对应的是三个参数:username、name、age。
数据结构与算法	public interface UsersMapper { /** * 全部用户查询 * @return */ List<Users> findAll();

```
    /**
     * 新增用户
     * @param users
     */
    void addUsers(Users users);

    /**
     * 修改用户
     * @param users
     */
    void updateUsers(Users users);

    /**
     * 删除用户
     * @param username
     */
    void deleteUsers(String username);

    /**
     * 指定 username 用户查询
     * @param username
     * @return
     */
    Users findByUsername(String username);


}


public interface PersonMapper {
    List<Person> findAll();
    void addPerson_Four(Person person);
    void addPerson_Three_age(Person person);
    void addPerson_Three_teleno(Person person);
    void addPerson_Two(Person person);
    void updatePerson(Person person);
    void deletePerson(String username);
    Person findByUsername(String username);
}
```

usersmapper.xml 实现

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/schema/mybatis-3-mapper.dtd">
<mapper namespace="com.example.shiyan4.mapper.UsersMapper">
```

```
    <select id="findAll" resultType="com.example.shiyan4.bean.Users">
        SELECT * FROM users
    </select>
    <insert id="addUsers" parameterType="com.example.shiyan4.bean.Users"
useGeneratedKeys="true" keyProperty="username" >
        INSERT INTO users (username,pass) VALUES
(#{username},#{pass})
    </insert>
    <update id="updateUsers"
parameterType="com.example.shiyan4.bean.Users">
        UPDATE users SET username=#{username}, pass=#{pass} WHERE
username=#{username}
    </update>
    <delete id="deleteUsers" parameterType="String">
        DELETE FROM users WHERE username=#{username}
    </delete>
    <select id="findByUsername" parameterType="String"
resultType="com.example.shiyan4.bean.Users">
        SELECT username,pass FROM users WHERE
username=#{username}
    </select>
</mapper>
```

personmapper.xml 实现：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/schema/mybatis-3-mapper.dtd">
<mapper namespace="com.example.shiyan4.mapper.PersonMapper">
    <select id="findAll" resultType="com.example.shiyan4.bean.Person">
        SELECT * FROM person
    </select>
    <insert id="addPerson_Four"
parameterType="com.example.shiyan4.bean.Person"
useGeneratedKeys="true" keyProperty="username">
        INSERT INTO
person(username,name,age,teleno)VALUES(#{username},#{name},#{age},#{te
leno})
    </insert>
    <insert id="addPerson_Three_age"
parameterType="com.example.shiyan4.bean.Person"
useGeneratedKeys="true" keyProperty="username">
        INSERT INTO
person(username,name,age,teleno)VALUES(#{username},#{name},#{age},#{te
leno})
    </insert>
```

```
    <insert id="addPerson_Three_teleno"
parameterType="com.example.shiyan4.bean.Person"
useGeneratedKeys="true" keyProperty="username">
        INSERT INTO
person(username,name,age,teleno)VALUES(#{username},#{name},#{age},#{te
leno})
    </insert>
    <insert id="addPerson_Two"
parameterType="com.example.shiyan4.bean.Person"
useGeneratedKeys="true" keyProperty="username">
        INSERT INTO
person(username,name,age,teleno)VALUES(#{username},#{name},#{age},#{te
leno})
    </insert>
    <update id="updatePerson"
parameterType="com.example.shiyan4.bean.Person">
        UPDATE person SET username=#{username}, name=#{name},
age=#{age}, teleno=#{teleno} WHERE username=#{username}
    </update>
    <delete id="deletePerson" parameterType="String">
        DELETE FROM person WHERE username=#{username}
    </delete>
    <select id="findByUsername" parameterType="String"
resultType="com.example.shiyan4.bean.Person">
        SELECT username,name,age,teleno FROM person WHERE
username=#{username}
    </select>
</mapper>
```

3.7 服务端——service 层

模块名称	service
功能描述	服务层，主要是业务逻辑的实现，需要调用 mapper 层的方法
接口与属性	同样是增删改查操作，只不过进行了逻辑上的实现。例如 addOrUpdatePerson 方法，再增加 person 表时，一并增加 users 表。
数据结构与算法	

```
public class UsersService {
    @Autowired
    private UsersMapper usersMapper;

    /**
     * 查找全部
     * @return
     */
    public List<Users> findAll(){
```

```
        return usersMapper.findAll();
    }

    /**
     * 增
     * @param users
     */
    public void addUsers(Users users){
        usersMapper.addUsers(users);
    }

    /**
     * 改
     * @param users
     */
    public void updateUsers(Users users){
        usersMapper.updateUsers(users);
    }

    /**
     * 删
     * @param username
     */
    public void deleteUsers(String username){
        System.out.println("数据库：users 表删除数据  "+username);
        usersMapper.deleteUsers(username);
    }

    /**
     * 查
     * @param username
     * @return
     */
    public Users findByUsername(String username){
        return usersMapper.findByUsername(username);
    }
}
```

```
public class PersonService {
    @Autowired
    private PersonMapper personMapper;
    @Autowired
    private UsersService usersService;
```

```
    /**
     * 查找全部
     * @return
     */
    public List<Person> findAll(){
        return personMapper.findAll();
    }

    /**
     * 增（如果有则修改）
     * Users 表中一并增加（如果有则修改）
     * @param person
     */
    public void addOrUpdatePerson(Person person){
        Person person1 =
personMapper.findByUsername(person.getUsername());
        if (person1 == null){
            // person 表中没有这个 username，在 person 和 users 表中都执行
添加
            System.out.println("数据库：Person 表新增数据");
            System.out.printf("username: %s\t",person.getUsername());
            System.out.printf("name: %s\t",person.getName());
            if (person.getAge() == -1){
                System.out.print("age: \t");
            }else{
                System.out.printf("age: %d\t",person.getAge());
            }
            System.out.printf("teleno: %s\n",person.getTeleno());

            // 针对不同数据调用不同的函数
            if (!person.getTeleno().equals("") && person.getAge() != -1){
                personMapper.addPerson_Four(person);
            }
            else if(!person.getTeleno().equals("") && person.getAge() == -1){
                personMapper.addPerson_Three_teleno(person);
            }
            else if(person.getTeleno().equals("") && person.getAge() != -1){
                personMapper.addPerson_Three_age(person);
            }
            else{
                personMapper.addPerson_Two(person);
            }
            //处理 User 表
            System.out.println("数据库：Users 表新增数据");
```

```
            System.out.printf("username: %s\t",person.getUsername());
            System.out.println("pass: 88888888");
            Users users = new Users(person.getUsername(),"88888888");
            usersService.addUsers(users);
        } else{
            // person 表中有这个 username，执行修改
            System.out.println("数据库：Person 表修改数据");
            System.out.printf("username: %s\t",person.getUsername());
            System.out.printf("name: %s\t",person.getName());
            if (person.getAge() == -1){
                System.out.print("age: \t");
            }else{
                System.out.printf("age: %d\t",person.getAge());
            }
            System.out.printf("teleno: %s\n",person.getTeleno());
            personMapper.updatePerson(person);
        }
    }

    /**
     * 改
     * @param person
     */
    public void updatePerson(Person person){
        personMapper.updatePerson(person);
    }

    /**
     * 删
     * @param username
     */
    public void deletePerson(String username){
        personMapper.deletePerson(username);
    }

    /**
     * 查
     * @param username
     * @return
     */
    public Person findByUsername(String username){
        return personMapper.findByUsername(username);
    }
```

	}

3.8 服务端——controller 层

模块名称	controller
功能描述	控制层，主要是针对不同的接口进行不同的返回。Get 方法通常是调用服务层 service，然后再通过 service 层调用 mapper 层，返回数据库中的数据。而 Post 方法则是只需要对数据库进行操作，返回 void。
接口与属性	有如下接口： @RequestMapping(value = "users/findall", method = RequestMethod.*GET*) @RequestMapping(value = "users/add",method = RequestMethod.*POST*) @RequestMapping(value = "users/update",method = RequestMethod.*POST*) @RequestMapping(value = "users/delete",method = RequestMethod.*POST*) @RequestMapping(value = "users/find",method = RequestMethod.*GET*) @RequestMapping(value = "person/findall", method = RequestMethod.*GET*) @RequestMapping(value = "person/add4", method = RequestMethod.*POST*) @RequestMapping(value="person/add3_teleno",method=RequestMethod.*POST*) @RequestMapping(value = "person/add3_age",method = RequestMethod.*POST*) @RequestMapping(value = "person/add2",method = RequestMethod.*POST*) @RequestMapping(value = "person/delete",method = RequestMethod.*POST*) @RequestMapping(value = "person/find",method = RequestMethod.*GET*)

数据结构与算法	public class UsersController {
	@Autowired
	private UsersService usersService;
	
	@RequestMapping(value = "users/findall", method = RequestMethod.*GET*)
	public List<Users> getUsersAll(){
	return usersService.findAll();
	}
	@RequestMapping(value = "users/add",method = RequestMethod.*POST*)
	public void addUsers(@RequestParam String username,@RequestParam String pass){
	Users users = new Users(username,pass);
	usersService.addUsers(users);
	}
	@RequestMapping(value = "users/update",method = RequestMethod.*POST*)
	public void updateUsers(@RequestParam String username,@RequestParam String pass){
	Users users = new Users(username,pass);
	usersService.updateUsers(users);
	}
	@RequestMapping(value = "users/delete",method =

```
RequestMethod.POST)
    public void deleteUsers(@RequestParam String username){
        usersService.deleteUsers(username);
    }
    @RequestMapping(value = "users/find",method = RequestMethod.GET)
    public Users getUsers(@RequestParam String username){
        return usersService.findByUsername(username);
    }
}
```

```
public class PersonController {
    @Autowired
    private PersonService personService;

    @RequestMapping(value = "person/findall", method =
RequestMethod.GET)
    public List<Person> getPersonAll(){
        return personService.findAll();
    }
    @RequestMapping(value = "person/add4", method =
RequestMethod.POST)
    public void addPerson(@RequestParam String
username,@RequestParam String name,@RequestParam int
age,@RequestParam String teleno){
        Person person = new Person(username,name,age,teleno);
        personService.addOrUpdatePerson(person);
    }
    @RequestMapping(value = "person/add3_teleno",method =
RequestMethod.POST)
    public void addPerson(@RequestParam String
username,@RequestParam String name,@RequestParam String teleno){
        Person person = new Person(username,name,-1,teleno);
        personService.addOrUpdatePerson(person);
    }
    @RequestMapping(value = "person/add3_age",method =
RequestMethod.POST)
    public void addPerson(@RequestParam String
username,@RequestParam String name,@RequestParam int age){
        Person person = new Person(username,name,age,"");
        personService.addOrUpdatePerson(person);
    }
    @RequestMapping(value = "person/add2",method =
RequestMethod.POST)
    public void addPerson(@RequestParam String
```

```
username,@RequestParam String name){
        Person person = new Person(username,name,-1,"");
        personService.addOrUpdatePerson(person);
    }
    @RequestMapping(value = "person/delete",method =
RequestMethod.POST)
    public void deletePerson(@RequestParam String username){
        personService.deletePerson(username);
    }
    @RequestMapping(value = "person/find",method = RequestMethod.GET)
    public Person getPerson(@RequestParam String username){
        return personService.findByUsername(username);
    }


}
```

3.9 服务端——具体的实现过程举例：find

模块名称	具体的实现过程举例：find
功能描述	如何实现查找操作
接口与属性	getUsers 方法

数据结构与算法	以 find 为例：
	首先在控制层 controller 中找到 find 操作对应的函数 getUsers()；
	`@RequestMapping(value = "users/find",method = RequestMethod.GET)`
	`public Users getUsers(@RequestParam String username){`
	`  return usersService.findByUsername(username);`
	`}`
	我们发现 gerUsers（）函数调用了服务层 service 中对应 find 操作的函数 findByUsername（）；
	`/**`
	` * 查`
	` * @param username`
	` * @return`
	` */`
	`public Users findByUsername(String username){`
	`  return usersMapper.findByUsername(username);`
	`}`

	接着可以看到服务层 service 中的 findByUsername（）函数调用了连接层 mapper 中对应 find 操作的函数 finfByUsername（）；
	`/**`
	` * 指定 username 用户查询`
	` * @param username`
	` * @return`
	` */`
	`Users findByUsername(String username);`
	最后发现连接层 mapper 中的 findByUsername()函数使用了实体 bean 层中定义的 username 变量。
	步骤大致为：
	1、设置相应内容类型
	2、获取参数
	3、调用 service 层函数，完成相关操作，得到返回结果
	4、将结果封装到 JSONObject 中，返回

4 程序运行流程演示

登录界面：

注册界面：

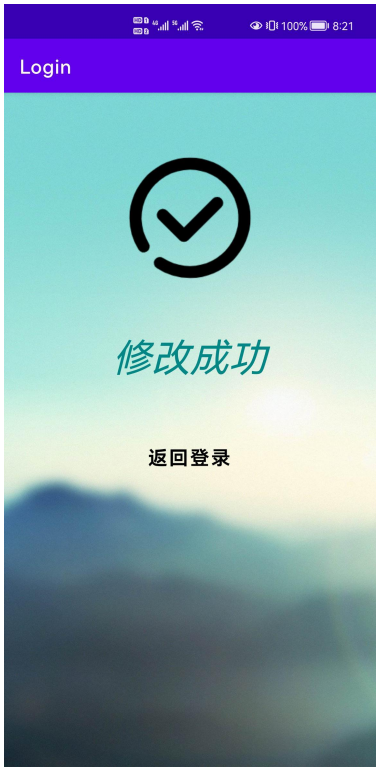
注册成功界面：

忘记密码界面：

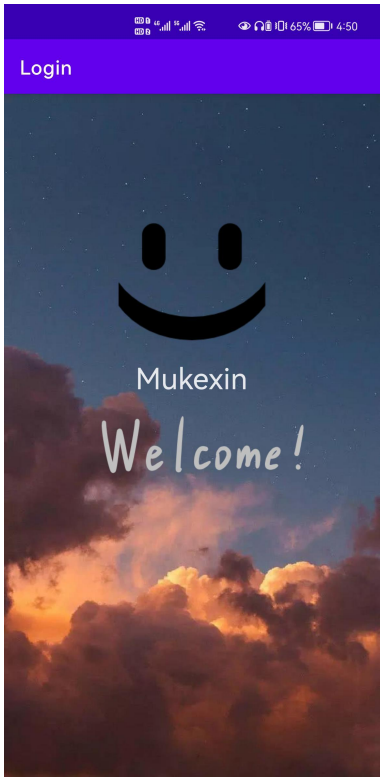
收到的验证码：

修改密码界面：

修改密码成功界面：



欢迎界面：


65% 4:50
Login
Mukexin
Welcome!