安卓 App 详细设计说明书

小组成员：孙浩然

专业班级：计科 1802

日期：2021 年 1 月 10 日

目录

1. 说明

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 以后后台服务模块的开发，要求后台使用 JavaEE 框架实现。

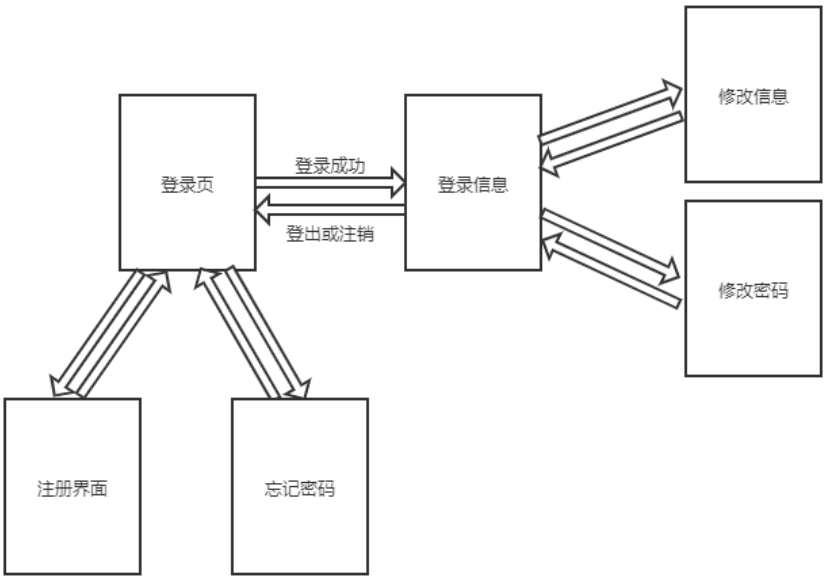
1.2 用途

编程新技术实务课程最终课程设计，通过 Java 连接数据库，结合 J2EE 和 Android，开发部署一个 App，包含基础的登录注册等功能。结合实验 1：数据库操作，实验 2：配置 Tomcat 并在其上部署网页端的注册登录模块，实验 3：简易安卓拨号器的实现，通过之前三个实验所学习到的工具和知识，结合起来能够完成本次实验。

1.3 预期功能

开发一个可用的安卓 App, 在登录界面可以进行账号注册, 忘记密码的处理, 以及登录功能。登录成功进入到用户信息界面，显示当前登录用户的基础信息。在用户的信息页可以进行登出操作，修改个人信息操作，修改密码，以及注销当前账户。

1.4 设计概要



大致逻辑如上图所示，通过页面上不同的 BUTTON 的点击，来触发提交或者是跳转功能，通过到达相应的页面，进而执行相关的功能。

1.5 实验环境

服务器环境：
Ubuntu 18.04.4 LTS

数据库：
mysql Ver 14.14 Distrib 5.7.31, for Linux (x86_64) using EditLine wrapper

Java 版本：
openjdk 11.0.8 2020-07-14
OpenJDK Runtime Environment (build 11.0.8+10-post-Ubuntu-0ubuntu118.04.1)
OpenJDK 64-Bit Server VM (build 11.0.8+10-post-Ubuntu-0ubuntu118.04.1, mixed mode, sharing)

Tomcat 版本：
Apache Tomcat/9.0.39

安卓 SDK 版本:
安卓 8.1 SdkVersion 27

项目地址：
https://github.com/Moyu-42/AndroidApp

项目 Demo（包含网页端设计以及 App 下载）：
Moyu's World

2. 服务端设计

2.1 整体架构

服务端主要处理客户端发来的数据，接收之后发送到数据库，并向客户端返回相应的信息。

在服务端的实现中，主要分为四层。Bean 层负责基本的数据结构的实现，以及对数据库的连接操作；Dao 层实现了对于用户 User 和 Person 的方法的实现；Service 层封装了对应的底层接口；Servlet 层主要负责接受发送到服务器的信息进行处理，并传回。

在 Servlet 层中，每一个 servlet 接口对应于上面设计概要中的每一个模块，负责处理相关的信息，通过接受客户端发送的 http 请求，对其内容进行相应的处理，并通过 JSON 的数据格式将处理的结果传回客户端，从而实现交互。

2.2 模块设计

2.2.1 Bean 层

1. **功能描述**
   负责构造底层的数据结构，负责对数据库进行操作，封装相应的接口。
2. **封装接口**
   1) Database.java

方法	描述
Database()	初始化数据库，获取一个新的链接
update()	用于更新或插入数据库的数据,返回布尔值表示成功与否
query()	用于查询对应的表项的数据，返回一个 List<Map<>>
close()	将链接等关闭

   2) User.java

方法	描述
User()	初始化一个 User，其 username 和 password 均为空
User(String username, String password)	按照传入的 username 和 password 初始化一个 User
setUsername(String username_)	将对应的 User 的 username 设置为传入值
setPassword(String password_)	将对应的 User 的 password 设置为传入值
getUsername()	返回 String 类型，获得当前 User 的 username
getPassword()	返回 String 类型，获得当前 User 的 password

   3) Person.java

方法	描述
Person(Params...)	初始化 Person，由于这里提供的接口较多，所以不再一一列举。可以初始化为空，或者初始化一定的信息，但是 Age 和 Teleno 可以保留为空
setUsername(String username_)	将对应的 Person 的 username 设置为传入值
setName(String name_)	将对应的 Person 的 name 设置为传入值
setAge(Integer age_)	将对应的 Person 的 age 设置为传入值
setTeleno(String teleno_)	将对应的 Person 的 teleno 设置为传入值
getUsername()	返回 String 类型，获得当前 Person 的 username
getName()	返回 String 类型，获得当前 Person 的 name
getAge()	返回 Integer 类型，获得当前 Person 的 age
getTeleno()	返回 String 类型，获得当前 Person 的 teleno

4) MD5.java

方法	描述
stringtoMD5(String plainText)	将传入的字符串进行 MD5 加密，返回加密后的字符串

3. 具体实现

1) Database.java（用于连接数据库）

通过从 JSON 配置文件（settings.json）读取数据库的配置，从而使用 JDBC 的方法连接到对应的数据库。

多次使用 JDBC 库支持的 `prepareStatement()`方法将操作发送到数据库，并且对增改查四个操作进行封装，作为最底层的接口提供给后续 Dao 层使用。

增改：

```
public boolean update(String sql, Object... params) {
    boolean flag = false;
    int result;
    try {
        pstmt = conn.prepareStatement(sql);
        if (params != null) {
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
        }
        result = pstmt.executeUpdate();
        flag = result > 0 ? true : false;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return flag;
}
```

查询：

```
public List<Map<String, Object>> query(String sql, Object... params) {
    List<Map<String, Object>> ans = new ArrayList<Map<String, Object>>();
    ResultSet resultset = null;
    ResultSetMetaData rsmd = null;
    try {
        pstmt = conn.prepareStatement(sql);
        if (params != null) {
            for (int i = 0; i < params.length; i++) {
                pstmt.setObject(i + 1, params[i]);
            }
        }
```

```
            resultset = pstmt.executeQuery();
            rsmd = resultset.getMetaData();

            while (resultset.next()) {
                Map<String, Object> map = new HashMap<String, Object>();
                for (int i = 0; i < rsmd.getColumnCount(); i++) {
                    String column_label = rsmd.getColumnLabel(i + 1);
                    Object column_object = resultset.getObject(column_label);
                    map.put(column_label, column_object);
                }
                ans.add(map);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            if (resultset != null) {
                try {
                    resultset.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
        return ans;
    }
```

关闭链接:

```
    public void close() {
        if (pstmt != null) {
            try {
                pstmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

2) User.java

包含 User 对应的信息，主要有：

```
public class User {
    private String Username;
    private String Password;
    ……
}
```

以及对应的 set，get 方法。

Username 作为主键，存放到数据库的对应表中

3) Person.java

包含 Person 对应的信息：

```
public class Person {
    private String Username;
    private String Name;
    private Integer Age;
    private String Teleno;
    ……
}
```

以及对应的 set，get 方法。

这里 Username 作为外键与 User 中的 Username 相连，对 Person 进行新增时，也会涉及到 User 表中对应的数据的新增；对 User 表中数据删除时，也会涉及到对 Person 表中相应的数据的删除。

其中，Name 作为 Person 表的主键存在

4) MD5.java

主要用于对传入数据库的密码进行加密，以提高安全性。

2.2.2 Dao 层

1. **功能概述**

   实现数据的持久化操作，如增删改查

2. **封装接口**

   1) UserOpt.java

方法	**描述**
insert(User user)	把传入的 User 插入或更新到数据库中
delete(User user)	将传入的 User 按照 username 删除
query ()	以 List<User>格式返回数据库的 User 表的全部表项
search(User user)	返回布尔值，判断传入的 User 是否存在，存在返回 True，否则为 False

2) PersonOpt.java

方法	描述
insert(Person p)	把传入的 Person 插入或更新到数据库中
delete(Person p)	将传入的 Person 按照 username 删除
query ()	以 List<Person>格式返回数据库的 Person 表的全部表项
search(Person p, int opt)	返回布尔值，判断传入的 Person 是否存在，存在返回 True，否则为 False。如果 opt 为 1，那么判断依据是 Person 的 username，否则按照 Person 的 name 来进行判断。

3. 具体实现

1) UserOpt.java

主要涉及到了对 User 的增加、删除、查询操作，此外，还支持数据库的日志系统，每次进行操作都会留下相应的日志。

增加：

```
public int insert(User user) {
    String sql = "select * from users where Username = ?";
    List<Map<String, Object>> uList = new ArrayList<Map<String, Object>>();
    uList = db.query(sql, user.getUsername());
    db.getLog(4);
    if (uList.isEmpty()) {
        sql = "insert into users(Username, Password) values(?, ?)";
        Object[] obj = { user.getUsername(), user.getPassword() };
        db.update(sql, obj);
    }
    else {
        sql = "update users set Password = ? where Username = ?";
        Object[] obj = { user.getPassword(), user.getUsername() };
        db.update(sql, obj);
    }
    db.getLog(2);
    return 1;
}
```

删除：

```
public int delete(User user) {
    String sql;
    sql = "delete from users where Username like ?";
    db.update(sql, user.getUsername());
    db.getLog(3);
    return 1;
}
```

查询：

这里的查询分为查询得到所有的 User 用户，以及查询 Username 对应的用户是否存在于数据库中。

查询全部用户：

```
    public List<User> query() {
        String sql = "select * from users";
        List<Map<String, Object>> ret = db.query(sql);
        List<User> ans = new ArrayList<User>();
        for (Map<String, Object> map : ret) {
            User user = new User((String) map.get("Username"), (String) map.get("Pas
sword"));
            ans.add(user);
        }
        db.getLog(4);
        return ans;
    }
```

查询对应的用户是否存在

```
    public boolean search(User user) {
        String sql = "select * from users where Username = ?";
        List<Map<String, Object>> uList = new ArrayList<>();
        uList = db.query(sql, user.getUsername());
        db.getLog(4);
        if (uList.isEmpty()) {
            return false;
        }
        else return true;
    }
```

2) PersonOpt.java

对于 Person 的操作和 User 相似，都是增删查三类方法。只不过对于增加的方法，要先判断 User 表存不存在相对应的 User，不存在也要向其中插入一个：

```
    public int insert(Person p) {
        String sql = "select * from person where Username = ?";
        List<Map<String, Object>> pList = new ArrayList<Map<String, Object>>();
        pList = db.query(sql, p.getUsername());
        if (pList.isEmpty()) {
            String sql_user = "select * from users where Username = ?";
            List<Map<String, Object>> uList = new ArrayList<Map<String, Object>>();
            uList = db.query(sql_user, p.getUsername());
            db.getLog(4);
            if (uList.isEmpty()) {
                String sql_user_insert = "insert into users(Username, Password) valu
es(?, ?)";
                Object[] obj_user = {p.getUsername(), "888888"};
```

```
                db.update(sql_user_insert, obj_user);
                db.getLog(2);
                String sql_person_insert = "insert into person(Username, Name, Age,
Teleno) values(?, ?, ?, ?)";
                Object[] obj_person = {p.getUsername(), p.getName(), p.getAge(), p.g
etTeleno()};
                db.update(sql_person_insert, obj_person);
            }else {
                String sql_person_insert = "insert into person(Username, Name, Age,
Teleno) values(?, ?, ?, ?)";
                Object[] obj_person = {p.getUsername(), p.getName(), p.getAge(), p.g
etTeleno()};
                db.update(sql_person_insert, obj_person);
            }
        }else {
            String sql_ = "update person set Name = ?, Age = ?, Teleno = ? where Use
rname = ?";
            Object[] obj = {p.getName(), p.getAge(), p.getTeleno(), p.getUsername()}
;
            db.update(sql_, obj);
        }
        db.getLog(6);
        return 1;
    }
```

其余部分则是和 UserOpt 类似，只不过将操作的主体换为了 Person

2.2.3 Service 层

1. **功能概述**
   将底层的 UserOpt 和 PersonOpt 进行封装，只暴露接口为后续程序提供使用
2. **封装接口**
   1) UserService.java

方法	描述
addUser(User user)	封装 userOpt 的 insert 方法
delUser (User user)	封装 userOpt 的 delete 方法
getQuery ()	封装 userOpt 的 query 方法
search(User user)	封装 userOpt 的 search 方法
login_varify(User user)	结合 userOpt 的 query 方法，用来提供给之后的注册界面，判断 username 和 password 是否匹配。

   2) PersonService.java

方法	描述
addPerson(Person person)	封装 personOpt 的 insert 方法
deletePerson(Person person)	封装 personOpt 的 delete 方法
getQuery()	封装 personOpt 的 query 方法
search(Person person)	封装 personOpt 的 search 方法

2.2.4 Servlet 层

1. **功能概述**

   是服务端与客户端交互的接口，客户端通过 http 将请求发送到服务端对应的 Servlet，经过处理后将信息返回给客户端。

   主要包括

   LoginServlet：处理登录界面的登录请求

   RegisterServlet：处理注册用户的请求

   GetLogInfoServlet：服务于安卓端，用来获取当前登录用户的信息

   GetInfoServlet：服务于 Web 端，获取登录用户的信息

   ModifyInfoServlet：用来修改用户的个人信息

   ChangePasswdServlet：用于处理忘记密码或者是修改密码的请求

   DeleteServlet：用来处理注销当前用户的请求

   DownloadServlet：服务于 Web 端，用来下载 App

2. **具体实现**

   以 LoginServlet.java 为例

```
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws S
ervletException, IOException {
        resp.setCharacterEncoding("UTF-8");
        req.setCharacterEncoding("utf-8");
        resp.setContentType("text/jsp; charset=utf-8");

        HttpSession sc = req.getSession();
        Database db;
        if (sc.getAttribute("database") == null) {
            db = new Database();
        }
        else db = (Database)sc.getAttribute("database");
        try(PrintWriter out = resp.getWriter()) {
            String username = req.getParameter("username").trim();
            String passwd = req.getParameter("password").trim();
            MD5 md5 = new MD5();
            passwd = md5.stringToMD5(passwd);

            UserService userService = new UserService(db);
```

```
            User user = new User(username, passwd);

            int result = userService.login_varify(user);
            String Result = new String();
            if (result == 0) {
                Result = "登录成功!";
            }else if (result == 1) {
                Result = "用户名不存在!请先注册";
            }else if (result == 2) {
                Result = "密码与用户名不匹配!";
            }

            JSONObject jsonObject = new JSONObject();
            jsonObject.put("Result", Result);
            out.write(jsonObject.toString());
        }
        if (sc.getAttribute("database") == null) {
            db.close();
        }
    }
```

首先从 request 中获取传输到服务端的数据
之后调用 service 层提供的接口，获得相应的操作结果
根据得到的结果设置返回信息，封装到 JSON 内，传输回客户端

3. 客户端设计

3.1 基本架构

主要包括两个部分：Web 端和 Android App 端。两个客户端均包含在“概要设计”中提到的部分，都支持完整的从注册到登录再到修改信息的功能。其中，Web 端支持对 App 应用的下载，Android 端提供了密码可见性操作以及选择性清空个人信息条目的选项，相较于 Web 端更加完善。

3.1.1 Android App 端

对于每一个设计界面，包括一个.xml 界面以及对应的.java 文件来处理页面的逻辑，以及向服务端发送信息并接受反馈，实现相应的功能。
界面包括：

.xml 页面文件	功能	.java 文件
activity_login.xml	主界面	MainActivity.java
loginfo.xml	用户登录信息界面	Loginfo.java

register.xml	注册新用户界面	Register.java
forgetpassword.xml	忘记密码界面	ForgetPassword.java
changepassword.xml	修改密码界面	ChangePassword.java
modify.xml	修改个人信息界面	Modify.java

3.1.2 Web 端

Web 端则是对于每个界面对应于一个.jsp 网页文件，通过 JavaScript 来处理页面的逻辑、数据校验以及表单的提交。
界面包括:

.jsp 页面	功能
index.jsp	主界面
login_info.jsp	登陆信息页面
register.jsp	注册界面
forget.jsp	忘记密码界面
change_password.jsp	修改密码界面
modify.jsp	修改个人信息界面

包含的 js 文件即相关的功能如下:

.js 文件	功能
login&forget.js	处理登录和忘记密码页面的操作
login_info.js	处理个人信息页面的注销操作
register.js	处理注册新用户的操作
modify.js	处理修改个人信息以及修改密码的操作
validator.js	对应于所有的表单数据校验

3.2 模块设计

3.2.1 Android App

需要为每个界面搭建显示界面：.xml 文件

1. **主界面**
   主要包括两个输入栏，输入 username 和 password，对于 password，提供了密码可见性的控制按钮。
   还包括三个 button 控件，分别为登录、注册、忘记密码。
2. **用户登录信息界面**
   在登录信息界面主要以文本的形式显示个人信息，至上向下依次为：当前登录用户的 username，用户的 Name，年龄 Age 以及电话号码 Telephone Number。对于 Age 和 Telephone Number，如果为空则不显示相关信息。对于第二行 Name 行，显示文本:

“Hello xxx !”，其中 xxx 为当前用户的 name。
此外还有四个 button，分别为登出，修改信息、修改密码以及注销账户。

3. **注册新用户界面**
   注册界面则需要输入 User 表和 Person 表对应的信息，自上向下依次为：username，password，还有重复输入 password，Name，Age，Telephone Number。对于 Age 和 Telephone Number 为非必填项，可以选择留空。对于两个输入的 password 要保证其一致，并且都提供了显示或隐藏的按钮。
   底部包括提交和取消两个按钮
4. **忘记密码界面**
   忘记密码界面则包括 username 的输入栏以及两次输入新 password 的输入栏，同样包括提交和取消两个按钮
5. **修改密码界面**
   修改密码的界面和忘记密码相比不再需要输入 username 栏，因为只有登录之后才会有修改密码的选项，此时已经知道了对应用户的 username。
   同样有提交取消两个按钮
6. **修改个人信息界面**
   修改个人信息则提供了三个输入栏：修改 name，修改 age，修改 telephone number。对于 age 和 telephone number，提供了切换按钮，如果当前的相对应的信息不为空，那么激活对应的按钮，可以在提交时将相关的信息清空；否则，不进行 age 或 telephone number 的输入则会保持原本的信息不改变。
   同样有提交取消两个按钮

对于每个界面，都需要一个.java 文件来处理相关的后台逻辑。

1. **MainActivity.java**
   处理按钮点击时的逻辑。对于登录按钮，获取当前的输入 username 和 password 之后，将其提交到服务端对应的 servlet（loginServlet），并获取传输会的 JSON 数据，取出信息后显示在界面上，并根据信息判断是否要跳转到用户登录信息页面。如果要跳转，则要将 username 放入 Bundle 内发送到对应的页面。
   对于注册和忘记密码的点击，则是直接跳转到对应页面。
2. **Loginfo.java**
   创建用户登录界面时，要根据发送到该页面的 username 来获取用户信息，将该 username 发送到 getLogInfoServlet ，获得传回的 User 的详细信息，并在页面上进行展示。
   如果点击了注销账户按钮，那么则会把当前的 username 发送到 deleteServlet ，将当前用户进行删除，删除之后跳转回主界面。
   点击登出按钮会跳转到主界面，点击修改密码或者修改个人信息按钮则会将 username 放入 Bundle 内发送到相应的页面并且跳转。
3. **Register.java**
   注册界面则在点击提交时，对输入的信息进行校验，要保证 username，password，name 不能为空，并且两次输入的 password 要相同，username 和 name 都不存在于数据库中。
   如果填入了 age 和 telephone number，那么要保证其为数字，且 age 要在一定的范围内，不能为负数，telephone number 要为 11 位。通过上面的校验规则之后，就会将输入的数据传输到 registerServlet ，接受传输回的消息并且跳转到主界面。
   点击取消则直接跳转到主界面。
4. **ForgetPassword.java**

对于忘记密码的操作，首先 username 和两次输入的 password 要符合校验规则，username 还要是能在数据库中查找到匹配项的；对于 password，新的 password 要保证不与上次使用的 password 相同。提交后发送到 changePasswdServlet ，并且跳转到主界面。

取消也是跳转到主界面。

5. **ChangePassword.java**

   在修改密码部分，password 的校验规则和忘记密码相同，通过校验后会将数据提交到服务端 changePasswdServlet 并且传回修改是否成功的信息，修改成功则是跳转回用户登录信息界面。

   点击取消也会跳转回用户登录信息界面。

6. **Modify.java**

   对于修改信息的校验，首先 name 不能与其他用户的 name 重复，但是可以为空，为空表示不对 name 进行修改。对于 age 和 telephone number 则可以为空，但是要保证上面三个输入框都不为空才可以进行提交。对于 age 和 telephone number，如果激活了清 0 按钮，那么当其输入框没有输入时，是可以将原本的对应的数据进行清空处理。校验通过之后提交到 modifyInfoServlet ，并且返回用户登录信息界面。

   点击取消返回用户登录信息界面。

3.2.2 Web

对于 Web 端每个界面的交互信息都和 Android 端的相同，这里就不再进行赘述。

对于每个 js 文件，起到的作用是处理 button 按钮的跳转以及数据的校验。

1. **login&forget.js**

   主要处理主界面和忘记密码界面的表单提交处理，当校验通过之后，会进行确认的弹出框提示。

   如果在登录界面，会将数据发送到 loginServlet ，并且登录成功将 username 发送到 getInfoServlet ，从这个 servlet 得到 username 对应的用户数据信息，保存到 session 内，在跳转到用户登录信息界面，通过 EL 表达式从 session 中取出数据并且显示。

   如果在忘记密码界面就会发送到 changePasswdServlet。

2. **login_info.js**

   在这个 js 文件中则是处理了删除用户按钮的作用，将 username 提交到 deleteServlet 并且跳转回主界面。

3. **register.js**

   这里处理了提交按钮，如果校验通过之后，那么会把数据提交到 registerServlet ，并且跳转回主界面。

4. **modify.js**

   处理了在点击提交按钮之后，将数据发送到 modifyServlet 的逻辑。

5. **validator.js**

   数据校验部分就与上面的 Android App 端相同。

与服务端交互

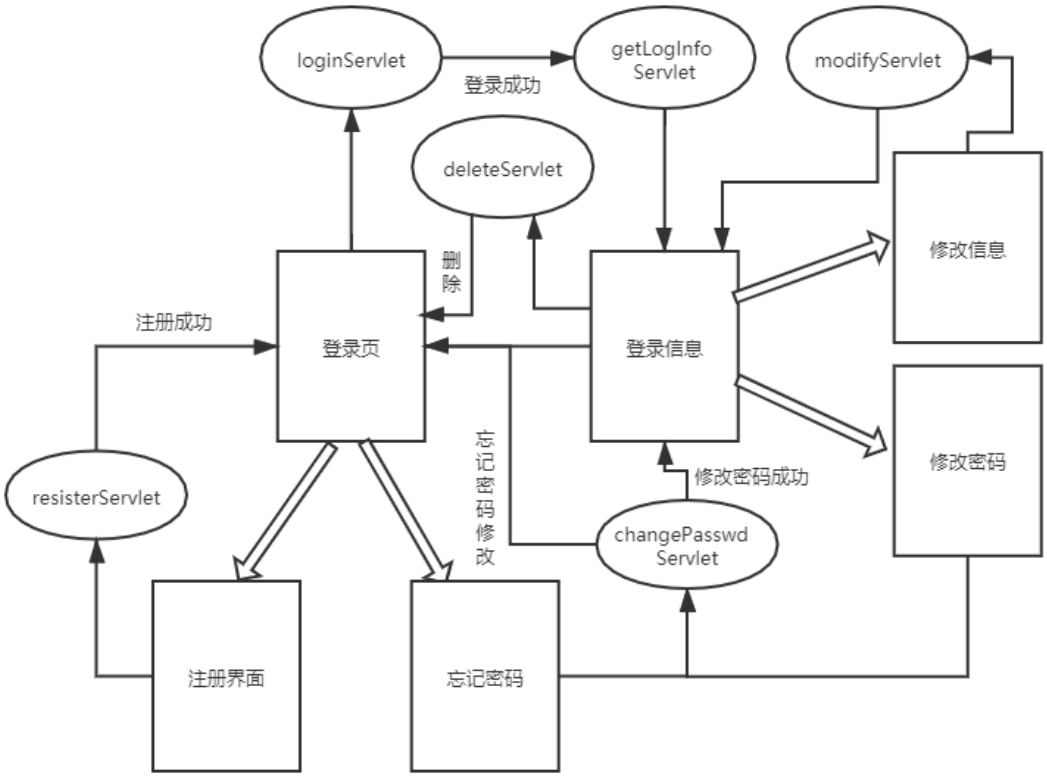
在于服务端交互时，通过 http 将数据发送到了服务端的 servlet。这里使用了 volley 框架，以主界面的登录按钮的提交为例：

```
    public void loginRequest(final String username, final String password) {
        String url = "http://49.234.84.130:8080/Exp4/loginServlet";
        String tag = "Login";
        RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
        requestQueue.cancelAll(tag);
        final StringRequest request = new StringRequest(Request.Method.POST, url,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        try {
                            JSONObject jsonObject = (JSONObject) new JSONObject(response);
                            String result = jsonObject.getString("Result");
                            System.out.println(result);
                            Toast.makeText(MainActivity.this, result, Toast.LENGTH_SHORT).show
();
                            if (result.equals("登录成功!")) {
                                Intent intent = new Intent(MainActivity.this, Loginfo.class);
                                Bundle bundle = new Bundle();
                                bundle.putString("username", username);
                                intent.putExtras(bundle);
                                startActivity(intent);
                            }
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    }
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                error.printStackTrace();
            }
        }) {
            @Override
            protected Map<String, String> getParams() throws AuthFailureError {
                Map<String, String> params = new HashMap<>();
                params.put("username", username);
                params.put("password", password);
                System.out.println(username);
                System.out.println(password);
```

```
                return params;
            }
        };
        request.setTag(tag);
        requestQueue.add(request);
    }
```

在设置好对应的 url 和 tag 之后，开始发送请求。这里的 url 即 Servlet 配置的 url 路径。通过 Map 的形式经过 getParams 方法传递数据，将其发送到 Servlet 中，并在 Servlet 对其处理之后传回，在 onResponse 方法中得到发送回的 JSON 格式数据。从中取出需要的信息并且执行后续的操作。

运行流程图



结果

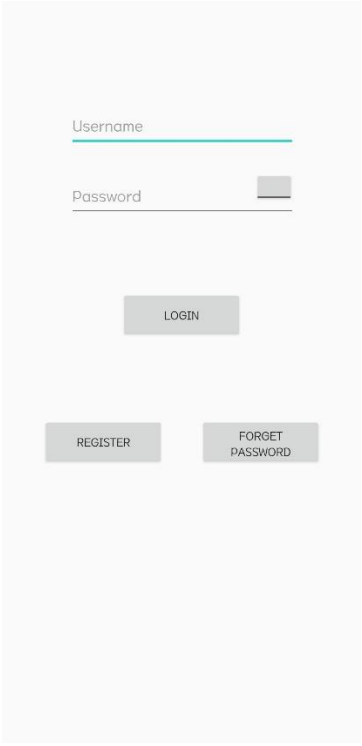


图 1

图 2

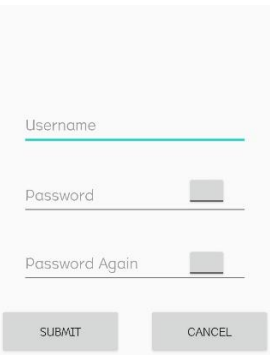


图 3

图 4

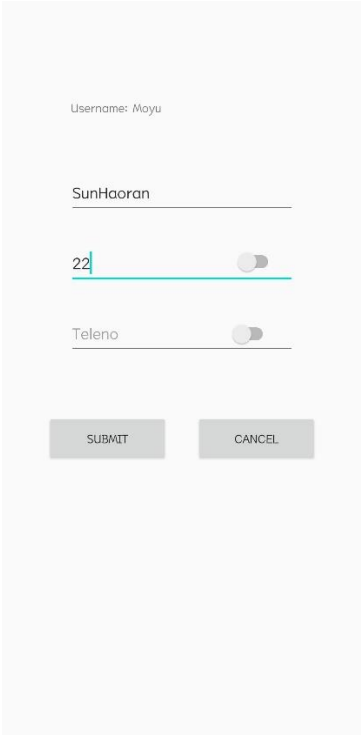


图 5

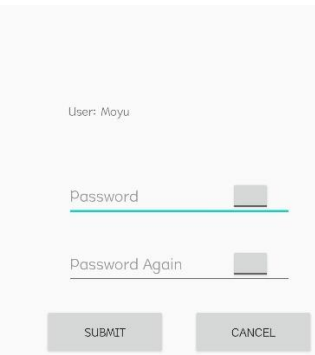


图 6

图 7

图 8

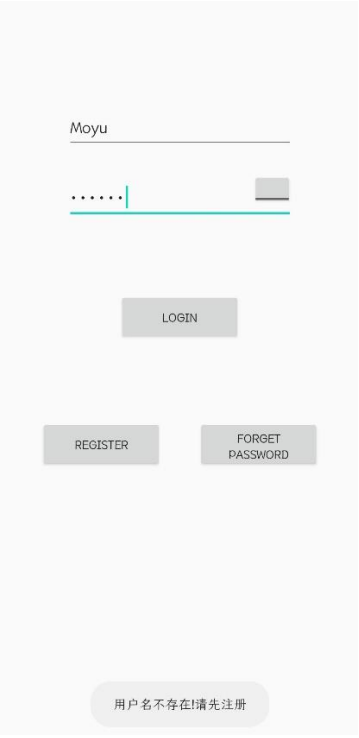


图 9

图 1 为主界面
图 2 为注册界面，注册了新的用户：username 为 Moyu，password 为 123456，Name 为 Shr
图 3 为忘记密码界面
图 4 为注册后登录成功界面，显示了个人信息
图 5 为修改个人信息界面，将 name 修改为了 SunHaoran，年龄修改为了 22
图 6 为修改密码界面，可以看到最上方为当前用户的 username
图 7 为修改完成后的个人信息界面
图 8 为删除用户的确定操作
图 9 为删除后再次登录，用户已经不存在。

开发人员名单与致谢

服务端:	孙浩然
Android App 客户端:	孙浩然
Web 客户端:	孙浩然
测试人员:	孙浩然
整体架构设计:	孙浩然
UI 设计:	孙浩然
设计文档编写:	孙浩然
服务器环境配置:	孙浩然