

# 移动互联网技术及应用

## 大作业报告

题目： Zigzag 定位签到系统的设计与实现

类型： 应用系统设计实现

| 姓名         | 班级         | 学号         |
|------------|------------|------------|
| 阿卜杜萨拉木·萨比克 | 2021211313 | 2021211363 |

2024.5.30

## 目录

|                                 |    |
|---------------------------------|----|
| 1. 相关技术.....                    | 3  |
| 1.1 位置获取.....                   | 3  |
| 1.1.1. Android 位置服务.....        | 3  |
| 1.1.2 LocationManager 类.....    | 3  |
| 1.1.3 LocationProvider 提供者..... | 3  |
| 1.1.4 获取位置信息.....               | 3  |
| 1.1.5 位置监听器.....                | 3  |
| 1.1.6 Geocoder 地理编码器.....       | 3  |
| 1.2 本地数据库.....                  | 3  |
| 1.2.1 数据结构.....                 | 4  |
| 1.2.2 用户数据初始化.....              | 4  |
| 1.2.3 用户管理功能.....               | 4  |
| 1.3 远程数据库.....                  | 5  |
| 1.4 Flask 后端服务器.....            | 6  |
| 1.5.1 Flask 框架.....             | 6  |
| 1.5.2. 跨域资源共享 (CORS).....       | 6  |
| 1.5.3. JSON 处理.....             | 6  |
| 1.5.4. HTTP 请求处理.....           | 6  |
| 1.5.5. 请求前后钩子.....              | 7  |
| 1.5.6. 地理位置计算.....              | 7  |
| 1.5.7. 数据处理逻辑.....              | 8  |
| 1.5 网络请求与响应处理.....              | 9  |
| 1.3.1 网络请求.....                 | 9  |
| 1.3.2 异步任务.....                 | 9  |
| 1.3.3 响应处理.....                 | 9  |
| 1.6 动态配置服务器.....                | 12 |
| 2. 系统功能需求.....                  | 12 |
| 2.1 用户需求分析.....                 | 12 |
| 2.2 系统功能.....                   | 13 |
| 3. 系统设计与实现.....                 | 13 |
| 3.1 总体设计.....                   | 13 |
| 3.1.1 架构设计.....                 | 13 |
| 3.1.2 技术选型.....                 | 14 |
| 3.1.3 功能划分.....                 | 14 |
| 3.1.4 界面设计.....                 | 16 |
| 3.2 系统组成.....                   | 19 |
| 3.3 各模块设计.....                  | 24 |
| 3.3.1 理论设计.....                 | 24 |
| 3.3.2 模块程序设计.....               | 25 |
| 3.4 关键代码解释.....                 | 29 |
| 4. 系统可能的扩展.....                 | 45 |
| 5. 总结体会.....                    | 47 |

# 1. 相关技术

## 1.1 位置获取

在本项目中，使用了 Android 平台提供的位置获取技术，通过定位用户的设备来获取其当前的地理位置信息。这里我们将详细介绍所使用的位置获取技术及其功能。

### Android 位置服务

Android 平台提供了位置服务（Location Service），它是一种系统级的服务，用于获取设备的地理位置信息。在本项目中，我们使用了 Android 的位置服务来实现位置的获取和监控。

### LocationManager 类

LocationManager 类是 Android 位置服务的核心组件之一，它用于管理位置提供者和位置更新的请求。通过 LocationManager，我们可以实现对位置信息的获取、监听和控制。

### LocationProvider 提供者

在 Android 中，位置信息可以通过不同的提供者来获取，包括 GPS、网络等。在本项目中，我们可以通过 LocationManager 来选择 GPS 和网络位置提供者，并根据需求获取位置信息。

### 获取位置信息

通过 LocationManager.getLastKnownLocation() 方法，我们可以获取设备的最后已知位置信息。此外，我们还可以通过请求位置更新来获取实时的位置信息，以实现位置的动态监控和追踪。

### 位置监听器

通过注册位置监听器，我们可以监听位置信息的变化，包括位置状态的改变、位置提供者的启用和禁用以及位置的实时更新。这样可以实现对位置变化的及时响应和处理。

### Geocoder 地理编码器

Geocoder 是 Android 提供的一个类，用于将地理位置的经纬度坐标转换为具体的地理位置信息，例如地址、城市等。在本项目中，我们使用了 Geocoder 来实现位置信息的解析和显示。

## 1.2 本地数据库

在本项目中，为了管理用户登录/注册，使用了一个基于 Java HashMap 实现的本地数据库辅助类 UserList。该类提供了一系列静态方法，用于用户数据的添加、删除、查询和验证。当用户在首页进行注册/登录操作时会调用该类的方法进行创建和验证用户的操作。这种实现方式简单高效，但是每次重启应用时都需要重新注册/登录。以下是对该类的详细概述：

### 1.2.1 数据结构

`UserList` 类中使用了一个静态的 `HashMap<String, String>` 对象来存储用户数据, 其中键(`String`) 代表用户名, 值(`String`) 代表用户密码。通过 `HashMap`, 可以高效地实现用户数据的存取和管理。

### 1.2.2 用户数据初始化

在类加载时, 会调用静态代码块 `initializeUsers()` 来初始化用户数据。在初始化过程中, 可以预先添加一些默认用户。例如:

```
private static void initializeUsers() {  
    addUser("salmjohn", "12345");  
    // 可以添加更多用户数据...  
}
```

### 1.2.3 用户管理功能

`UserList` 类提供了一些静态方法来管理用户数据, 包括添加用户、删除用户、检查用户名是否存在, 以及验证用户名和密码。以下是主要的功能介绍:

#### 添加用户

`addUser(String username, String password)` 方法用于向数据库中添加新用户。在添加用户之前, 会先检查用户名是否已存在, 以避免重复添加。

```
public static void addUser(String username, String password) {  
    if (!isUsernameExists(username)) {  
        userList.put(username, password);  
        System.out.println("用户 " + username + " 添加成功!");  
    } else {  
        System.out.println("用户名 " + username + " 已存在, 添加失败!");  
    }  
}
```

#### 删除用户

`removeUser(String username)` 方法用于从数据库中删除指定用户名的用户。在删除用户之前, 会先检查用户名是否存在, 以确保删除操作的有效性。

```
public static void removeUser(String username) {  
    if (isUsernameExists(username)) {  
        userList.remove(username);  
        System.out.println("用户 " + username + " 删除成功!");  
    } else {  
        System.out.println("用户名 " + username + " 不存在, 删除失败!");  
    }  
}
```

### 检查用户名是否存在

isUsernameExists(String username)方法用于检查指定用户名是否存在于数据库中。

```
public static boolean isUsernameExists(String username) {  
    return userList.containsKey(username);  
}
```

### 验证用户名和密码

isUserCredentialsCorrect(String username, String password)方法用于验证提供的用户名和密码是否匹配。

```
public static boolean isUserCredentialsCorrect(String username, String password) {  
    String storedPassword = userList.get(username);  
    return storedPassword != null && storedPassword.equals(password);  
}
```

## 1.3 远程数据库

本项目是基于定位服务的签到系统，管理员可以确定签到位置，用户可以进行签到，所以签到位置信息和用户签到信息需要保存在服务器。我用的是 json 文件保存，当有新的请求时服务器会更新两个信息文件。具体更新过程如下：

从文件读取数据并初始化全局变量

```
def load_data_from_file(classroom_file='place_info.json', sign_file='sign_info.json'):  
    global classroom_info, sign_info  
    try:  
        with open(classroom_file, 'r') as f:  
            classroom_info = json.load(f)    直接赋值而不是 update  
        with open(sign_file, 'r') as f:  
            sign_info = json.load(f)    直接赋值而不是 extend  
    except FileNotFoundError:  
        如果文件不存在，创建一个空文件  
        with open(classroom_file, 'w') as f:  
            pass    创建空文件  
        with open(sign_file, 'w') as f:  
            pass    创建空文件
```

将全局变量写回文件

```
def save_data_to_file(classroom_file='place_info.json', sign_file='sign_info.json'):  
    with open(classroom_file, 'w') as f:  
        json.dump(classroom_info, f)  
    with open(sign_file, 'w') as f:  
        json.dump(sign_info, f)
```

在收到请求时加载数据

```
@app.before_request
def load_data():
    load_data_from_file()
```

在应用上下文结束时保存数据

```
@app.teardown_appcontext
def save_data(exception=None):
    save_data_to_file()
```

每次收到请求时先从文件取出签到信息和位置信息初始化全局变量。用户的请求可能会改变全局变量的值。完成客户端请求后把全局变量写回文件，保证了信息的实时性。

## 1.4 Flask 后端服务器

本项目我用 python Flask 框架写了 app 后端服务器。这个 Flask 后端服务器利用 Flask 框架处理 HTTP 请求，通过 CORS 支持跨域请求，通过 JSON 文件持久化数据，并使用地理计算方法来处理位置相关的业务逻辑。通过我的测试，该系统能够有效地记录和管理教室和学生的签到信息。该 flask 程序目前已经上服务器运行，保证了 APP 长时间正常运行。

### 1.5.1 Flask 框架

Flask 是一个轻量级的 WSGI Web 应用框架，它被设计为易于使用且高度可扩展。该代码使用 Flask 来创建和运行 Web 服务器，并处理 HTTP 请求和响应。

### 1.5.2. 跨域资源共享 (CORS)

使用 flask\_cors 库来处理跨域资源共享（CORS），允许前端应用程序从不同的源访问后端资源。通过 CORS(app) 来启用 CORS 支持。这个是为了在开发过程中进行实时的测试。

### 1.5.3. JSON 处理

通过 json 库来读取和写入 JSON 文件，用于存储和加载教室和签到信息。Flask 的 request 对象用于获取请求体中的 JSON 数据，jsonify 函数用于返回 JSON 格式的响应。

### 1.5.4. HTTP 请求处理

定义了多个路由来处理不同的 HTTP 请求：

| 请求类型 | 路由             | 功能              |
|------|----------------|-----------------|
| GET  | /              | 返回一个简单的 HTML 页面 |
| POST | /set_position  | 设置新的签到位置信息      |
| POST | /sign          | 学生签到            |
| GET  | /get_sign_info | 获取所有学生的签到信息     |

|      |                    |             |
|------|--------------------|-------------|
| GET  | /get_place_info    | 获取所有签到位置的信息 |
| POST | /delete_sign_info  | 清除所有签到信息    |
| POST | /delete_place_info | 清除所有教室位置信息  |

Post 请求参数:

| 路由                      | 请求参数  | 服务器返回   |
|-------------------------|---|---|
| POST /sign              | <pre>{   "latitude": "float",   "longitude": "float",   "altitude": "float",   "student_id": "string",   "student_name": "string" }</pre> | jsonify({'position':string})                  |
| POST /set_position      | <pre>{   "name": "string",   "latitude": "float",   "longitude": "float",   "altitude": "float",   "classroom_name": "string" }</pre>     | return "1"                                    |
| POST /delete_sign_info  | <pre>{   "name": "007" }</pre>  | return "1"<br>或<br>return "Unauthorized", 403 |
| POST /delete_place_info | <pre>{   "name": "007" }</pre>  | return "1"<br>或<br>return "Unauthorized", 403 |

#### 1.5.5. 请求前后钩子

使用 `@app.before_request` 和 `@app.teardown_appcontext` 钩子在每次请求前后执行数据加载和保存操作:

`load_data` 函数在每次请求前加载教室和签到信息。

`save_data` 函数在请求结束时保存教室和签到信息。

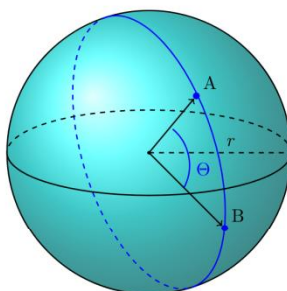
#### 1.5.6. 地理位置计算

使用 `math` 库中的函数来计算两个经纬度之间的距离。`calculate_distance` 函数使用 Haversine 公式来计算地球表面两点之间的距离。

Haversine 公式详情请看下面:

## 4.2. Definition of the Haversine Formula

We can now define the formula of haversine for calculating the distance between two points in the spherical coordinate system. The formula itself is simple, and it works for any pair of points that are defined according to their radial coordinates for a given radius:



Let's define the two points  $A$  and  $B$  according to their respective latitudes  $\phi$  and longitude  $\lambda$ , as  $A = (\phi_A, \lambda_A)$  and  $B = (\phi_B, \lambda_B)$ , respectively, expressed in radians. Let's also define the great circle distance between  $A$  and  $B$  as  $d_{AB}$ .

Since  $d_{AB}$  is an arc of a great circle with radius  $r$  and central angle  $\Theta$ , we know that  $d_{AB} = r \times \Theta$ . In this system, the law of haversine  $\text{hav}(\Theta)$  corresponding to the central angle  $\Theta$  is:

$$\text{hav}(\Theta) = \text{hav}(\phi_B - \phi_A) + \cos(\phi_B) \cos(\phi_A) \text{hav}(\lambda_B - \lambda_A)$$

Once we compute  $\text{hav}(\Theta)$  for the pair of points  $(A, B)$ , we can then calculate  $d_{AB}$  by using the inverse haversine function  $\Theta = \text{archav}(\text{hav}(\Theta))$ . If we do that, we can finally calculate  $d_{AB}$  as:

$$d_{AB} = r \times \text{archav}(\text{hav}(\Theta))$$

In its computation, we then simply replace  $\text{hav}(\Theta)$  with its full expression defined above, in terms of the latitudes  $\phi_A, \phi_B$  and longitudes  $\lambda_A, \lambda_B$  of the points  $A$  and  $B$ .

计算两个经纬度之间的距离

```
def calculate_distance(lat1, lon1, lat2, lon2):
```

```
    地球半径，单位：米
```

```
    R = 6371000.0
```

```
    将角度转换为弧度
```

```
    lat1 = radians(lat1)
```

```
    lon1 = radians(lon1)
```

```
    lat2 = radians(lat2)
```

```
    lon2 = radians(lon2)
```

```
    计算经纬度的差值
```

```
    dlon = lon2 - lon1
```

```
    dlat = lat2 - lat1
```

```
    使用 Haversine 公式计算距离
```

```
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
```

```
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
```

```
    distance = R * c
```

```
    return distance
```

### 1.5.7. 数据处理逻辑

包括几个核心功能：

教室位置设置：通过 `POST /set_position` 路由设置教室的位置，包括纬度、经度和高度。

学生签到：通过 `POST /sign` 路由处理学生签到请求，记录学生的签到位置和时间。



数据获取: 通过 GET /get\_sign\_info 和 GET /get\_place\_info 路由获取签到和教室信息。

数据清除: 通过 GET /delete\_sign\_info 和 GET /delete\_place\_info 路由清除签到和教室信息。

## 1.5 网络请求与响应处理

### 1.3.1 网络请求

本项目中我使用 `HttpURLConnection` 进行网络请求的发送和响应的处理。`HttpURLConnection` 是 Java 标准库中的一个类, 用于发送和接收 HTTP 请求和响应。它提供了一种简单的方法来与网络服务器进行交互。

### 1.3.2 异步任务

为了避免在主线程上进行网络操作, 该类使用了 `AsyncTask` 来处理网络请求。`AsyncTask` 是一个轻量级的异步任务处理类, 适合在后台线程中执行耗时操作, 并在任务完成后将结果传回主线程。

### 1.3.3 响应处理

此 APP 后端共有 7 个网络请求, 其中 3 个 GET, 4 个 POST。对于通过 GET 获得的数据在 APP 端需要处理后格式化输出:

- GET /get\_sign\_info

```
StringBuilder signInfoStringBuilder = new StringBuilder();
signInfoStringBuilder.append("成功获取签到信息: \n");

try {
    for (int i = 0; i < signInfoArray.length(); i++) {
        JSONObject signInfoObject = signInfoArray.getJSONObject(i);
        String latitude = signInfoObject.getString("latitude");
        String longitude = signInfoObject.getString("longitude");
        String altitude = signInfoObject.getString("altitude");
        String studentId = signInfoObject.getString("student_id");
        String studentName = signInfoObject.getString("student_name");
        String position = signInfoObject.getString("position");
        String time = signInfoObject.getString("time");
        // 格式化输出每个签到信息
        String signInfo = "学生 ID: " + studentId + "\n"
            + "学生姓名: " + studentName + "\n"
            + "签到位置: " + position + "\n"
            + "签到时间: " + time + "\n"
            + "纬度: " + latitude + "\n"
            + "经度: " + longitude + "\n"
            + "海拔: " + altitude + "\n\n";

        signInfoStringBuilder.append(signInfo);
    }
}
```

```

    }
} catch (JSONException e) {
    e.printStackTrace();
}

String formattedSignInInfo = signInInfoStringBuilder.toString();
Log.d("GetSignInInfoTask", formattedSignInInfo);

// 这里将格式化后的签到信息显示在界面的合适位置
// 将其设置为 TextView 的文本内容
list.setText(formattedSignInInfo);

```

#### ● GET /get\_place\_info

```

StringBuilder signInInfoStringBuilder = new StringBuilder();
signInInfoStringBuilder.append("成功获取位置信息: \n");
try {
    // 获取 JSON 对象的所有键（即教室名称）
    Iterator<String> keys = result.keys();
    while (keys.hasNext()) {
        String roomName = keys.next();
        // 根据教室名称获取对应的 JSON 对象
        JSONObject roomInfo = result.getJSONObject(roomName);

        // 提取教室的经纬度和海拔
        double latitude = roomInfo.getDouble("latitude");
        double longitude = roomInfo.getDouble("longitude");
        double altitude = roomInfo.getDouble("altitude");
        signInInfoStringBuilder.append("\n 地点: ").append(roomName).append("\n 纬度: ").append(latitude).append("\n 经度: ").append(longitude).append("\n 高度: ").append(altitude).append("\n");
        // 在这里处理数据，例如更新 UI 或进行其他操作
    }
} catch (JSONException e) {
    // 处理 JSON 解析异常
    e.printStackTrace();
}

String formattedSignInInfo = signInInfoStringBuilder.toString();
Log.d("GetSignInInfoTask", formattedSignInInfo);

// 将其设置为 TextView 的文本内容
list.setText(formattedSignInInfo);

```

服务器端的相应处理体现在对 POST 请求参数的处理上:

#### ● POST /sign （服务器）

```

# 学生签到
@app.route('/sign', methods=['POST'])
def sign():
    data = request.json
    latitude = data.get('latitude')
    longitude = data.get('longitude')
    altitude = data.get('altitude')
    student_id = data.get('student_id')
    student_name = data.get('student_name')
    position="未知"
    #遍历 place_info, 计算 class 与当前学生签到位置的距离, 如果距离小于 100 米则 position
    设置为教室名, 否则不改
    # 设置学生签到位置
    position = set_student_position(float(latitude), float(longitude))
    curr_time=datetime.now()
    # 将时间格式化为人类可读的格式
    formatted_time = curr_time.strftime("%Y%m%d %H:%M:%S")
    # 记录学生签到信息
    this_sign_info = {'time':formatted_time,'position':position,'latitude': latitude, 'longitude':
longitude, 'altitude': altitude, 'student_id': student_id, 'student_name': student_name}
    #print(this_sign_info)
    # 追加到签到信息
    sign_info.append(this_sign_info)

    return jsonify({'position':position}) # 返回签到成功

```

#### ● POST /set\_position

```

# 设置教室位置
@app.route('/set_position', methods=['POST'])
def set_position():
    data = request.json
    who=data.get('name')
    latitude = data.get('latitude')
    longitude = data.get('longitude')
    altitude = data.get('altitude')
    classroom_name = data.get('classroom_name')

    # 更新教室位置信息
    classroom_info[classroom_name] = {'latitude': latitude, 'longitude': longitude, 'altitude':
altitude}

    return "1" # 返回设置成功

```

## 1.6 动态配置服务器

在本项目中，动态配置服务器是通过获取网页内容的方式来实现的。具体而言，应用程序会通过网络请求获取一个包含服务器地址的 HTML 页面，然后使用 Jsoup 库解析页面内容并提取出服务器地址。这种方法允许服务器地址随时在服务器端进行更改，而无需更新应用程序的代码。如果获取服务器地址的过程失败（比如网络连接问题），则应用程序会使用预定义的默认服务器地址。通过动态配置服务器，本项目实现了一种灵活且可维护的方式来管理服务器地址，从而提高了应用程序的可扩展性和可维护性。

用 githubpages 里的静态网页动态配置服务器地址。

如果服务器有发生更改，只需要将 <http://abdusalam1.github.io/server.html> 此静态页面的服务器 IP 地址改为新的，而不需要更新 APP。保证了 App 的稳定性。

```
public class GlobalVariable {

    public static final String BASE_URL;

    static {
        String baseUrl = "";
        try {
            // 获取网页内容
            Document doc = Jsoup.connect("http://abdusalam1.github.io/server.html").get();
            // 查找 id 为 server 的元素
            Element serverElement = doc.getElementById("server");
            // 获取元素的文本内容，即你想要的字符串
            baseUrl = serverElement.text();
        } catch (IOException e) {
            e.printStackTrace();
            // 如果获取失败，则使用默认值
            baseUrl = "http://8.130.105.61:8082";
        }
        // 将获取到的字符串赋值给 BASE_URL
        BASE_URL = baseUrl;
    }
}
```

## 2. 系统功能需求

### 2.1 用户需求分析

- 用户管理：系统支持用户的注册、登录以及权限管理功能。
- 定位签到：用户可以通过 GPS 或网络定位进行签到，并记录签到的地理位置信息。
- 签到和信息管理：管理员可以指定和管理签到地点，可以查看和管理用户的签到记录，

包括查看签到列表和删除签到信息。

## 2.2 系统功能

- 位置获取：系统通过 `LocationManager` 获取定位服务，实现定位功能。
- 服务器通信：系统通过 `HttpURLConnection` 与服务器进行 `http` 通信。
- 界面友好：系统有简洁明了的用户界面，确保用户能够轻松上手。
- 数据安全：系统确保用户数据的安全性和隐私性。
- 准确性：系统能够准确地进行定位、数据传输。
- 可靠性：系统能够稳定地进行消息处理和网络请求，并能够处理错误情况，如用户名为空，地点名为空等。

## 3. 系统设计与实现

### 3.1 总体设计

#### 3.1.1 架构设计

- 客户端架构：

客户端系统采用 MVC（ModelViewController）架构设计，将业务逻辑、数据模型和用户界面分离，以提高代码的可维护性和可扩展性。MVC 模式将应用分为三个主要组件，通过定义清晰的职责划分，实现了模块之间的解耦和重用。

**Model（模型）：**负责应用的数据和业务逻辑。它处理数据的读取、存储和处理，以及与数据相关的操作。

**View（视图）：**负责应用的用户界面。它展示数据给用户，并接收用户的输入。

**Controller（控制器）：**负责应用的逻辑控制。它接收用户的输入，对模型进行操作并更视图。

下面是客户端主要程序文件：

辅助类：

`UserList.java`

`ToastUtil.java`

`GlobalVariable.java`

`AdminActivity.java`

`CheckPlaceList.java`

`CheckSignList.java`

布局（Layout）：

`activity_main.xml`

`activity_location_sign.xml`

`activity_function.xml`

`activity_check_sign_list.xml`

`activity_check_place_list.xml`

`activity_admin.xml`

活动（Activity）：

`MainActivity.java`

`FunctionActivity.java`

`LocationSign.java`

- 服务器端架构：

Flask 服务器的架构主要包括以下几个组成部分：

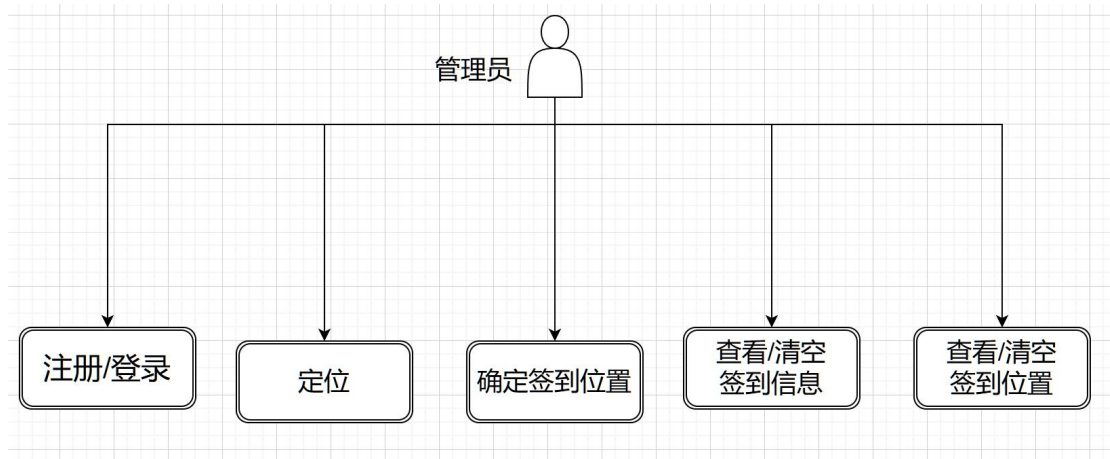
1. 路由和视图函数：使用`@app.route`装饰器定义了不同的路由，每个路由对应一个视图函数，处理特定的 HTTP 请求。
2. 全局变量和数据处理函数：定义了一些全局变量（如`classroom\_info`和`sign\_info`），以及处理数据的函数（如`load\_data\_from\_file()`和`save\_data\_to\_file()`）。这些函数负责从文件读取数据并初始化全局变量，以及将全局变量写回文件。
3. 请求处理钩子：使用`@app.before\_request`和`@app.teardown\_appcontext`装饰器定义了请求处理钩子函数。`@app.before\_request`装饰的函数会在每次请求处理之前执行，用于加载数据。`@app.teardown\_appcontext`装饰的函数会在应用上下文结束时执行，用于保存数据。
4. HTTP 请求处理函数：定义了处理不同 HTTP 请求的函数，如`set\_position()`处理设置教室位置的请求，`sign()`处理学生签到请求，`get\_sign\_info()`处理获取签到信息的请求等。
5. 辅助函数：定义了一些辅助函数，如`calculate\_distance()`用于计算两个经纬度之间的距离，`set\_student\_position()`用于设置学生的签到位置。

### 3.1.2 技术选型

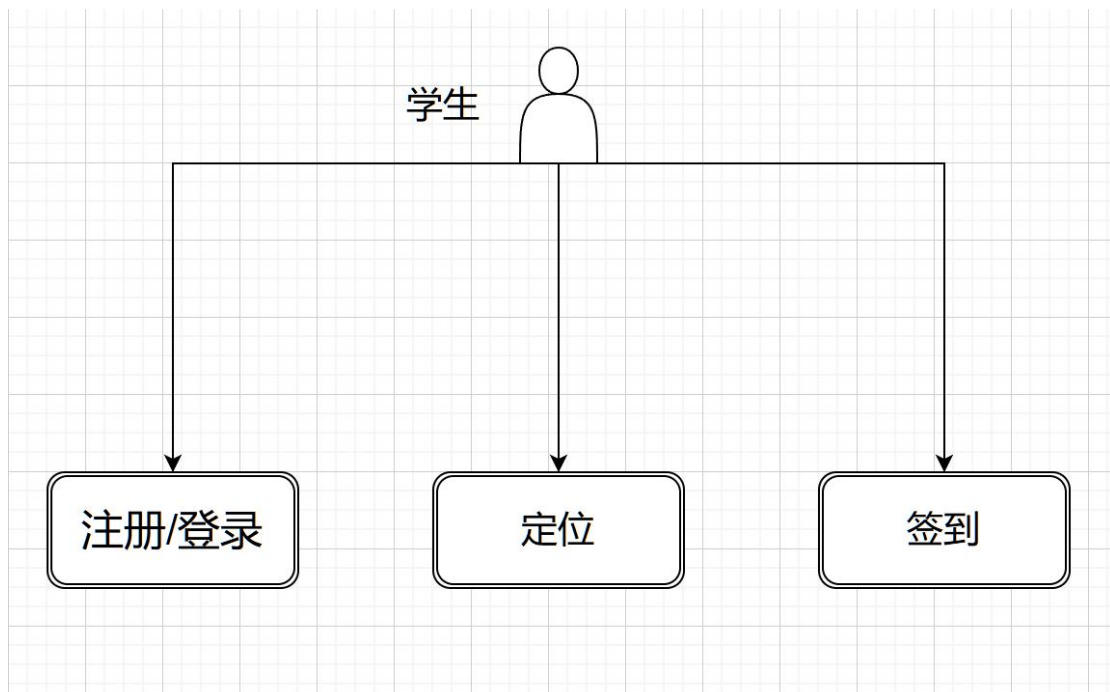
- 开发语言：Java
- 开发工具：Android Studio
- 网络通信：HTTP 协议，使用 HttpURLConnection 进行网络请求。
- 数据解析：使用 JSON 格式进行数据传输和解析。
- 后端服务器：python Flask

### 3.1.3 功能划分

- **注册登录**：每次 App 启动以后需要进行注册和登录。管理员通过唯一的管理人员账号登录，不能注册新的管理员。学生可以通过账号/密码注册登录，账号建议选择学号，方便区分。
- **定位**：登录完成之后就可以进行定位获取当前地理位置。
- **管理员规定签到地址**：管理员到达签到地点，先定位，再输入地名提交就建立了新的签到地点。
- **管理员查看和删除签到地址**：管理员通过点击对应按钮查看和删除（更新）签到地址
- **管理员查看和删除学生签到信息**：管理员通过点击对应按钮查看和删除学生签到信息
- **学生到达签到地点定位签到**：学生到达签到地点后先定位，然后输入姓名提交即可完成签到。



管理员账号密码默认为: admin



### 3.1.4 界面设计

- 注册/登录页 (activity\_main.xml)



整个页面在一个 `LinearLayout` 里，其中两个按钮在一个横向 `LinearLayout` 里

- 学生功能页 (activity\_function.xml)



此页面是为了给 `App` 功能的扩展预留足够的空间而设计的。开发人员根据需求对 `App` 功能进行扩展。



● 学生详情页 (activity\_location\_sign.xml)

10:03 0.13 KB/s HD 48

开始定位

☒ GPS  
☐ Network

经度: 116.35025321  
纬度: 39.95874773  
高度: 60.8690185546875  
加速度: 10.317723  
方向: 0.0  
Address[addressLines=[0:"北京市海淀区西土城路10号院-14号楼",1:"北京邮电大学",2:"北京邮电大学计算机科学与技术学院",3:"北京邮电大学电信工程学院教研室",4:"北京邮电大学继续教育学院实验室",5:"自动化学院实验室",6:"北京邮电大学-BUPT-QMUL

1.输入姓名

2.签到

\*签到前务必进行定位, 否则提交上次的定位结果\*

此页面用 `LinearLayout` 构建  
定位结果信息在一个 `ScrollView`, 可以上下滑动  
学生可以先定位, 然后输入姓名点击签到

● 管理员详情页 (activity\_admin.xml)

10:07 3.49 KB/s HD 48

开始定位

☒ GPS  
☐ Network

经度: 116.35025321  
纬度: 39.95874773  
高度: 60.8690185546875  
加速度: 10.317723  
方向: 0.0  
Address[addressLines=[0:"北京市海淀区西土城路10号院-14号楼",1:"北京邮电大学",2:"北京邮电大学计算机科学与技术学院",3:"北京邮电大学电信工程学院教研室",4:"北京邮电大学继续教育学院实验室",5:"自动化学院实验室",6:"北京邮电大学-BUPT-QMUL

添加新的签到地址

查看签到名单 查看签到地点

此页面用 `LinearLayout` 构建  
定位结果信息在一个 `ScrollView`, 可以上下滑动  
管理员可以先定位, 然后添加为新签到地址, 也可以管理服务器上的数据。  
当点击“添加新的签到地址”时该按钮会消失并显示为如下两个视图

1.地名/教室名/其他备注

2.规定为签到地址

● 管理员管理签到信息页  
(activity\_check\_sign\_list.xml)

10:08 0.50 KB/s HD 48

成功获取签到信息:  
学生ID: administration  
学生姓名: did  
签到位置: 未知  
签到时间: 2024-06-06 14:01:32  
纬度: 39.96241791  
经度: 116.35127568  
海拔: 44.172119140625

学生ID: 拜拜  
学生姓名: 几几声  
签到位置: 未知  
签到时间: 2024-06-06 14:03:52  
纬度: 39.962087  
经度: 116.349876  
海拔: 0.0

消除记录

● 管理员管理点到地点信息页  
(activity\_place\_sign\_list.xml)

10:08 2.72 KB/s HD 48

成功获取位置信息:  
地点: 教三437  
纬度: 39.95877134  
经度: 116.35025042  
高度: 60.2310791015625

消除记录

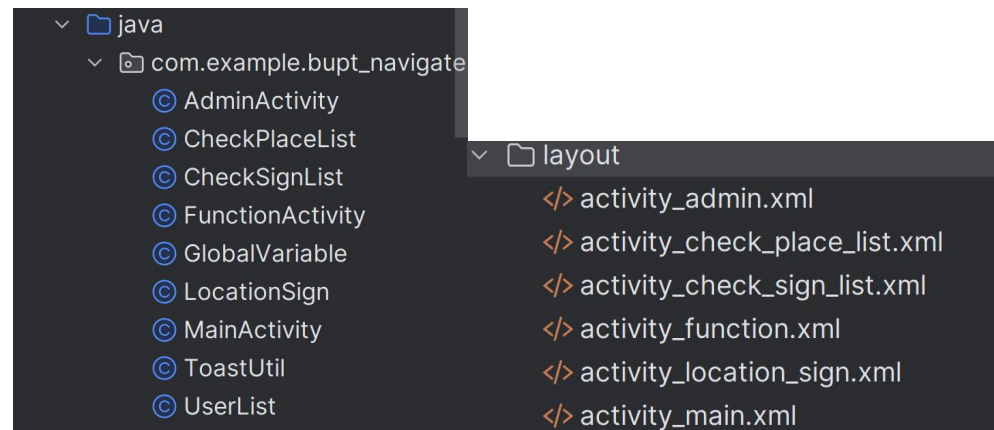
这俩页面都是用 `LinearLayout` 构建, 当收到服务器发来的 `json` 签到信息后会格式化显示在此屏幕上。点击消除记录会清空服务器和本地的相关记录。

## 3.2 系统组成

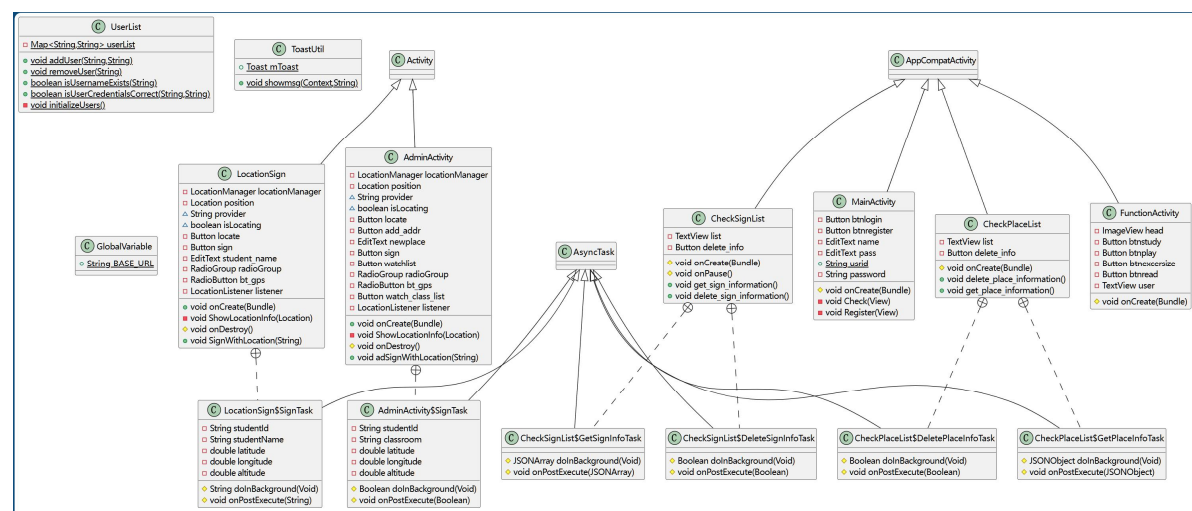
App 系统由客户端和服务端组成。客户端包括 app 开发过程中的程序，素材（图片，文字等），打包好的可执行程序（apk）。服务器端包括用到的云服务器，服务器上运行的 flask 后端程序，以及负责维护程序的其他软件。

### ● 客户端

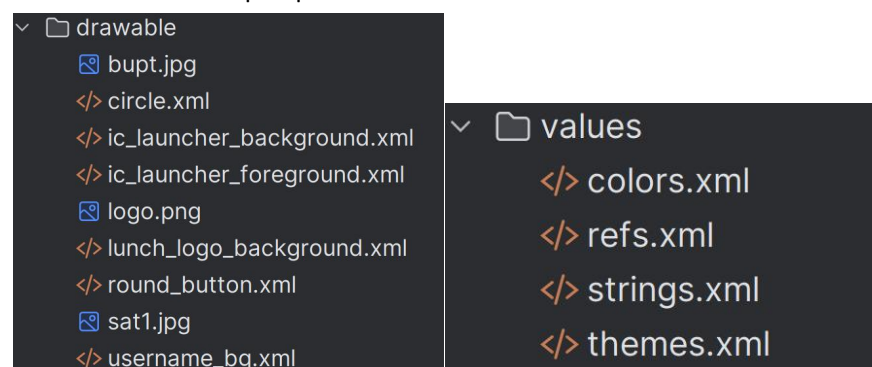
客户端总共有 6 个 activity 和对应布局文件，另外三个辅助类



下面是客户端 java 程序的类图：



用到的素材除了 mipmap 里面的图标以外包括下面的：



- 服务器端

服务器端包括阿里云弹性服务器，负责运行后端程序的 `gunicorn` 软件和后端程序：

```
from flask import Flask, request, jsonify, g
from flask_cors import CORS # 导入 CORS
from math import radians, sin, cos, sqrt, atan2
from datetime import datetime
import json
app = Flask(__name__)
CORS(app) # 添加 CORS 支持

# 教室签到信息
classroom_info = {}
sign_info = []

# 从文件读取数据并初始化全局变量
def load_data_from_file(classroom_file='place_info.json', sign_file='sign_info.json'):
    global classroom_info, sign_info
    try:
        with open(classroom_file, 'r') as f:
            classroom_info = json.load(f) # 直接赋值而不是 update
        with open(sign_file, 'r') as f:
            sign_info = json.load(f) # 直接赋值而不是 extend
    except FileNotFoundError:
        # 如果文件不存在，创建一个空文件
        with open(classroom_file, 'w') as f:
            pass # 创建空文件
        with open(sign_file, 'w') as f:
            pass # 创建空文件

# 将全局变量写回文件
def save_data_to_file(classroom_file='place_info.json', sign_file='sign_info.json'):
    with open(classroom_file, 'w') as f:
        json.dump(classroom_info, f)
    with open(sign_file, 'w') as f:
        json.dump(sign_info, f)

# 在收到请求时加载数据
@app.before_request
def load_data():
    load_data_from_file()
```

```

# 在应用上下文结束时保存数据
@app.teardown_appcontext
def save_data(exception=None):
    save_data_to_file()

#test
@app.route('/',methods=['GET'])
def index():
    return "<h1>man ,what can I say</h1>"

# 设置教室位置
@app.route('/set_position', methods=['POST'])
def set_position():
    data = request.json
    who=data.get('name')
    latitude = data.get('latitude')
    longitude = data.get('longitude')
    altitude = data.get('altitude')
    classroom_name = data.get('classroom_name')

    # 更新教室位置信息
    classroom_info[classroom_name] = {'latitude': latitude, 'longitude': longitude, 'altitude': altitude}

    return "1" # 返回设置成功

# 学生签到
@app.route('/sign', methods=['POST'])
def sign():
    data = request.json
    latitude = data.get('latitude')
    longitude = data.get('longitude')
    altitude = data.get('altitude')
    student_id = data.get('student_id')
    student_name = data.get('student_name')
    position="未知"
    #遍历 place_info, 计算 class 与当前学生签到位置的距离, 如果距离小于 100 米则 position 设置为教室
    # 设置学生签到位置
    position = set_student_position(float(latitude), float(longitude))
    curr_time=datetime.now()
    # 将时间格式化为人类可读的格式

```

```

        formatted_time = curr_time.strftime("%Y-%m-%d %H:%M:%S")
        # 记录学生签到信息
        this_sign_info = {'time':formatted_time,'position':position,'latitude': latitude, 'longitude': longitude,
        'altitude': altitude, 'student_id': student_id, 'student_name': student_name}
        #print(this_sign_info)
        # 追加到签到信息
        sign_info.append(this_sign_info)

        return jsonify({'position':position}) # 返回签到成功

# 获取学生签到信息
@app.route('/get_sign_info', methods=['GET'])
def get_sign_info():
    # print(sign_info)
    return jsonify(sign_info)

# 获取签到位置信息
@app.route('/get_place_info', methods=['GET'])
def get_place_info():
    #print(classroom_info)
    return jsonify(classroom_info)

# 消除签到信息
@app.route('/delete_sign_info', methods=['POST'])
def delete_sign_info():
    data = request.json
    name = data.get('name')
    if name == "007": # 这里简单验证请求是否合法
        global sign_info
        sign_info = []
        return "1"
    else:
        return "Unauthorized", 403

# 消除位置信息
@app.route('/delete_place_info', methods=['POST'])
def delete_place_info():
    data = request.json
    name = data.get('name')
    if name == "007": # 这里简单验证请求是否合法
        global classroom_info
        classroom_info = {}
        return "1"
    else:

```

```

        return "Unauthorized", 403

# 计算两个经纬度之间的距离
def calculate_distance(lat1, lon1, lat2, lon2):
    # 地球半径，单位：米
    R = 6371000.0

    # 将角度转换为弧度
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    # 计算经纬度的差值
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # 使用 Haversine 公式计算距离
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c

    return distance

# 设置学生的签到位置
def set_student_position(latitude, longitude):
    closest_classroom = "未知"
    min_distance = float('inf')

    # 遍历教室位置信息，计算每个教室与学生位置的距离
    for classroom, info in classroom_info.items():
        classroom_lat = info['latitude']
        classroom_lon = info['longitude']

        distance = calculate_distance(latitude, longitude, float(classroom_lat), float(classroom_lon))
        # print("distance between",classroom,"is: ",distance)
        # 如果距离小于最小距离，更新最小距离和最近教室
        if distance < min_distance:
            min_distance = distance
            closest_classroom = classroom

    # 如果最小距离小于 100 米，将学生位置设置为最近教室，否则保持未知
    if min_distance < 50:

```

```
        return closest_classroom
    else:
        return "未知"

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=8081)
```

用下面命令可以在自己希望的端口运行后端程序：

```
nohup python -m gunicorn -w 5 -b 0.0.0.0:8082 -t 120 server:app > server.log 2>&1 &
```

## 3.3 各层模块设计

### 3.3.1 理论设计

本系统可按层次划分为以下几个关键部分组成，每个部分承担着特定的功能和职责，共同协作以实现系统的整体目标。

#### 用户界面层（UI Layer）

用户界面层是系统与用户交互的前端部分，提供了一个直观、易用的操作界面。它包括：

- 登录/注册界面：允许用户创建账户和登录系统。
- 主界面：展示系统的主要功能和操作选项。
- 功能页面：根据用户角色（如管理员、学生）展示不同的功能选项。
- 签到界面：用户可以在此进行签到操作，并查看签到状态。

#### 业务逻辑层（Business Logic Layer）

业务逻辑层处理系统的中间件，实现具体的业务规则和逻辑。它包括：

- 用户管理模块：处理用户的注册、登录和权限验证。
- 位置服务模块：负责获取和处理用户的位置信息。
- 签到管理模块：处理签到请求，验证签到位置和时间等。
- 数据交互模块：负责前端与后端之间的数据请求和响应。



数据访问层（Data Access Layer）

数据访问层负责与数据库的交互，包括数据的增删改查操作。它包括：

- 本地数据库：存储用户信息和应用设置等。
- 远程数据库接口：与后端服务器通信，进行数据的同步和更新。

后端服务层（Backend Service Layer）

后端服务层提供了系统所需的服务和计算能力。它包括：

- 服务器端应用程序：处理业务逻辑，提供 API 接口。
- 位置计算服务：计算位置距离，判断签到有效性。
- 数据持久化：将签到信息和位置信息持久化存储。

网络通信层（Network Communication Layer）

网络通信层负责系统的网络请求和数据传输。它包括：

- HTTP 请求处理：发送和接收 HTTP 请求，处理网络通信。
- 数据序列化：将数据转换为 JSON 格式，便于网络传输。

安全性与隐私保护（Security and Privacy）

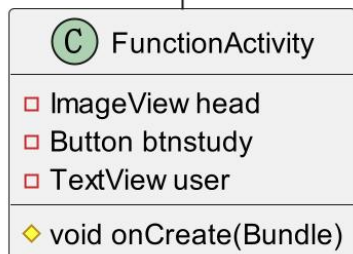
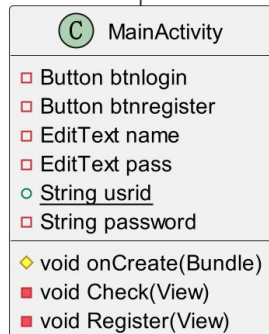
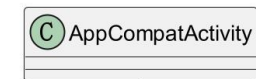
- 用户数据加密：确保用户数据的安全性。
- 权限控制：确保用户只能访问对应的资源和数据。

系统扩展性（Scalability）

- 模块化设计：便于未来功能的添加和修改。
- 服务端扩展：支持水平扩展，应对用户数量增长。

3.3.2 模块程序设计

|          |        |
|----------|--------|
| activity | xml 布局 |
|----------|--------|





10:07

开始定位

GPS

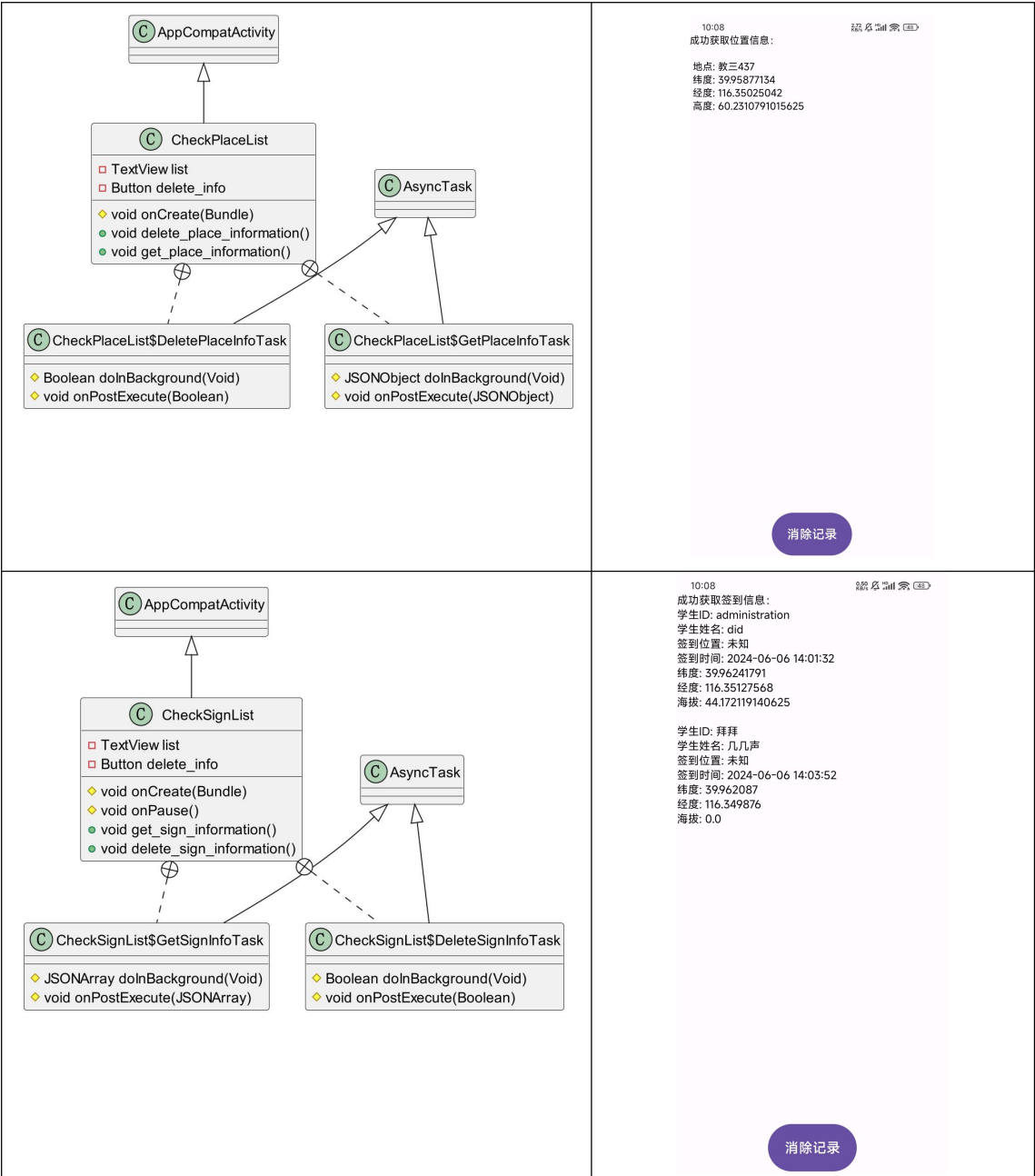
Network

经度: 116.35025321  
纬度: 39.95874773  
高度: 60.8690185546875  
加速度: 10.317723  
方向: 0.0  
Address[addressLines=[0:"北京市海淀区西土城路10号院-14号楼",1:"北京邮电大学",2:"北京邮电大学计算机科学与技术学院",3:"北京邮电大学电信工程学院教研室",4:"北京邮电大学继续教育学院实验室",5:"自动化学院实验室",6:"北京邮电大学-BUPT-QMUL"]]

添加新的签到地址

查看签到名单

查看签到地点



UserList 辅助类。用于用户注册/登录信息的记录和检查合法性。

C

UserList

Map<String,String> userList

void addUser(String,String).

void removeUser(String).

boolean isUsernameExists(String).


boolean isUserCredentialsCorrect(String,String).

void initializeUsers().

GlobalVariable 辅助类。获取服务器地址用。

|  |
|--|
|  GlobalVariable |
| o <u>String BASE_URL</u>   |

ToastUtil 辅助类。把原来 ToastUtil 封装成用法简单的辅助类。

|   |
|---|
|  ToastUtil |
| o <u>Toast mToast</u>   |
| ● <u>void showmsg(Context,String).</u>  |

### 3.4 关键代码解释

- MainActivity 判断用户名是否注册/登录，是否重名，是否为空：

```

1.     public class MainActivity extends AppCompatActivity {
2.         private Button btnlogin;    // 声明控件，要对Button 添加动作，声明为Button
3.         private Button btnregister;
4.         private EditText name;      // 分别用于获取用户名，密码
5.         private EditText pass;
6.         public static String usrid ;
7.         private String password ;
8.         @Override
9.         protected void onCreate(Bundle savedInstanceState) {
10.            super.onCreate(savedInstanceState);
11.            EdgeToEdge.enable(this);
12.            setContentView(R.layout.activity_main);
13.            ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
14.                Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
15.                v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
16.                return insets;
17.            });
18.
19.            // 找到控件
20.            btnlogin = findViewById(R.id.button_login);
21.            btnregister = findViewById(R.id.button_register);
22.            name = findViewById(R.id.edit1);
23.            pass = findViewById(R.id.edit2);
24.
25.

```

```
26.
27.         //匹配用户名，密码登录
28.         btnlogin.setOnClickListener(this::Check);
29.         btnregister.setOnClickListener(this::Register);
30.
31.     }
32.
33.     private void Check(View v) {
34.         //获取用户输入
35.
36.         usrid = name.getText().toString();
37.         password = pass.getText().toString();
38.         Intent intent = null;
39.         //弹窗弹出内容
40.         String yes = "登录成功! ";
41.         String no = "账号或密码有误，重新登录! ";
42.         if (usrid.equals("admin") && password.equals("admin")){
43.             ToastUtil.showmsg(getApplicationContext(), "管理员登录");
44.
45.             intent = new Intent(MainActivity.this, AdminActivity.class);
46.             startActivity(intent);
47.         }
48.         //账号密码配对
49.         else if (UserList.isUserCredentialsCorrect(usrid,password)) {
50.             // 用封装好的类 ToastUtil
51.             ToastUtil.showmsg(getApplicationContext(), yes);
52.             intent = new Intent(MainActivity.this, FunctionActivity.class);
53.             startActivity(intent);
54.         } else {
55.             //error,弹出登陆失败，用 toast
56.             //用封装的类
57.             ToastUtil.showmsg(getApplicationContext(), no);
58.         }
59.     }
60.
61.
62.     private void Register(View v) {
63.         String username = name.getText().toString();
64.         String password = pass.getText().toString();
65.         String yes="注册成功! ";
66.         String no="注册失败，用户名已存在! ";
67.         String empty="用户名不能为空! ";
68.         if(username.equals("")){
69.             ToastUtil.showmsg(getApplicationContext(), empty);
```

```

70.     }
71.     else {
72.         if (!UserList.isUsernameExists(usrname)) {
73.             UserList.addUser(usrname, password);
74.             ToastUtil.showmsg(getApplicationContext(), yes);
75.         } else {
76.             ToastUtil.showmsg(getApplicationContext(), no);
77.         }
78.     }
79.
80. }
81.
82. }

```

### ● FunctionActivity 的跳转:

```

1.     public class FunctionActivity extends AppCompatActivity {
2.
3.         // 声明控件
4.         private ImageView head;
5.
6.         private Button btnstudy;
7.
8.         private TextView user;
9.         @Override
10.        protected void onCreate(Bundle savedInstanceState) {
11.            super.onCreate(savedInstanceState);
12.            EdgeToEdge.enable(this);
13.            setContentView(R.layout.activity_function);
14.            ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
15.                Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
16.                v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
17.                return insets;
18.            });
19.
20.            // 找到控件
21.
22.            head = findViewById(R.id.touxiang);
23.
24.
25.            btnstudy = findViewById(R.id.btn_main1);
26.            user=findViewById(R.id.username);
27.            user.setText(MainActivity.usrid);
28.
29.            // 按钮跳转

```

```

30.         btnstudy.setOnClickListener(new View.OnClickListener() {
31.             @Override
32.             public void onClick(View v) {
33.                 Intent intent=null;
34.                 intent=new Intent(FunctionActivity.this,LocationSign.class);
35.                 startActivity(intent);
36.             }
37.         });
38.     }
39.
40. }

```

- LocationSign 的定位和签到:

Oncreate 里面是初始化布局-->找到并绑定控件--> 监听控件-->修改位置信息/进行 http 通信

```

1.     public class LocationSign extends Activity {
2.
3.         //系统服务对象
4.         private LocationManager locationManager;
5.
6.         //定位的位置
7.         private Location position;
8.
9.         //定位方式
10.        String provider = LocationManager.GPS_PROVIDER;
11.        // String provider = LocationManager.NETWORK_PROVIDER;
12.
13.        //是否正在定位--GPS 是否正在工作?
14.        boolean isLocating;
15.
16.        //按钮对象-- 开始, 停止
17.        private Button locate;
18.        private Button sign;
19.        private EditText student_name;
20.        private RadioGroup radioGroup;
21.        private RadioButton bt_gps, bt_network;
22.
23.        @Override
24.        public void onCreate(Bundle savedInstanceState) {
25.            super.onCreate(savedInstanceState);
26.            setContentView(R.layout.activity_location_sign);
27.
28.            isLocating = false;
29.            locate = (Button) findViewById(R.id.locate);
30.            locate.setText("开始定位");

```



```
31.
32.
33.         radioGroup = (RadioGroup) findViewById(R.id.postion_group);
34.         bt_gps = (RadioButton) findViewById(R.id.gps);
35.         bt_network = (RadioButton) findViewById(R.id.network);
36.         sign = findViewById(R.id.sign);
37.         student_name=findViewById(R.id.student_name);
38.         radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
39.             @Override
40.             public void onCheckedChanged(RadioGroup group, int checkedId) {
41.                 // TODO Auto-generated method stub
42.
43.                 if (isLocating) { //如果正在定位
44.                     locationManager.removeUpdates(listener); //停止定位
45.                     locate.setText("开始定位");
46.                     isLocating = false;
47.                 }
48.
49.                 if (bt_gps.getId() == checkedId) {
50.                     provider = locationManager.GPS_PROVIDER;
51.
52.                 } else if (bt_network.getId() == checkedId) {
53.                     provider = locationManager.NETWORK_PROVIDER;
54.                 }
55.             }
56.         });
57.
58.         //按钮跳转
59.         sign.setOnClickListener(new View.OnClickListener() {
60.             @Override
61.             public void onClick(View v) {
62.                 String ss=student_name.getText().toString();
63.                 SignWithLocation(ss);
64.             }
65.         });
66.
67.
68.         //获取位置服务管理对象
69.         locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
70.         if (locationManager == null) {
71.             Toast.makeText(this, "获取系统服务失败!", Toast.LENGTH_SHORT).show();
72.             return;
73.         }
74.
```

```

75.         //如果没有打开 位置信息开关,则转至 位置信息开关 设置页面
76.         if (!locationManager.isProviderEnabled(provider)) {
77.             (new AlertDialog.Builder(this))
78.                 .setTitle("提示")
79.                 .setMessage("需要启用相应的定位设备")
80.                 .setPositiveButton("确定", new OnClickListener() { // 点击确定后打开设
置对话框
81.                     @Override
82.                     public void onClick(DialogInterface dialog, int which) {
83.                         // TODO Auto-generated method stub
84.                         //打开设置页面
85.                         Intent intent = new Intent();
86.                         intent.setAction(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
87.                         intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
88.                         try {
89.                             startActivity(intent);
90.                         } catch (ActivityNotFoundException ex) {
91.                             // General settings activity
92.                             intent.setAction(Settings.ACTION_SETTINGS);
93.                             try {
94.                                 startActivity(intent);
95.                             } catch (Exception e) {
96.                             }
97.                         }
98.                     }
99.
100.                })
101.                .create().show();
102.        }

```

## ● 位置获取和显示模块

```

103.         // 获取定位信息
104.         if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
105.             {
106.                 ((TextView) findViewById(R.id.info)).setText("权限获取失败!");
107.             }
108.             String[] locationPermission = {Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION};
109.             ActivityCompat.requestPermissions(this, locationPermission, 300);
110.             return;
111.         }
112.         position = locationManager.getLastKnownLocation(provider);

```

```
113.
114.         //显示位置信息
115.         ShowLocationInfo(position);
116.
117.         locate.setOnClickListener(new View.OnClickListener() {
118.             @Override
119.             public void onClick(View v) {
120.                 // TODO Auto-generated method stub
121.                 if (isLocating) { //如果正在定位
122.                     locationManager.removeUpdates(listener); //停止定位
123.                     locate.setText("开始定位");
124.                     isLocating = false;
125.                 } else {
126.                     //设置位置改变的监听器
127.                     //时间大于2秒,距离大于10米时通知改变
128.                     if (
129.                         ContextCompat.checkSelfPermission(getApplicationContext(), android.Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
130.                         ContextCompat.checkSelfPermission(getApplicationContext(), android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
131.
132.                         return;
133.                     } else {
134.                         ((TextView) findViewById(R.id.info)).setText("没有可用的位置信息!");
135.                     }
136.                     locationManager.requestLocationUpdates(provider, 2000, 10, listener);
137.                     locate.setText("停止定位");
138.                     isLocating = true;
139.                 }
140.
141.             }
142.         });
143.
144.     }
145.
146.     //显示位置信息
147.     private void ShowLocationInfo(Location position) {
148.         String s = "";
149.
150.         if (position == null) {
151.             ((TextView) findViewById(R.id.info)).setText("没有可用的位置信息!");
152.             return;
153.         }
154.
```

```

155.         // 获取位置信息
156.         s += "经度: ";
157.         s += position.getLongitude() + "\n";
158.         s += "纬度: ";
159.         s += position.getLatitude() + "\n";
160.         s += "高度: ";
161.         s += position.getAltitude() + "\n";
162.         s += "加速度: ";
163.         s += position.getAccuracy() + "\n";
164.         s += "方向: ";
165.         s += position.getBearing();
166.
167.         // 解析经纬度对应的地址信息
168.         List<Address> address = null;
169.         Geocoder gc = new Geocoder(LocationSign.this, Locale.CHINA);
170.         try {
171.             address = gc.getFromLocation(position.getLatitude(), position.getLongitude(), 2)
172.             ;
173.             // TODO Auto-generated catch block
174.             e.printStackTrace();
175.         }
176.         String t = "";
177.         if (address != null && address.size() > 0) {
178.             for (int i = 0; i < address.size(); i++) {
179.                 t += address.get(i).toString() + "\n\n";
180.             }
181.         } else
182.             t = "没有解析出地址...\n";
183.
184.         // 显示位置信息
185.         ((TextView) findViewById(R.id.info)).setText(s + "\n" + t);
186.     }
187.
188.     // 位置改变的监听器--追踪移动
189.     private LocationListener listener = new LocationListener() {
190.         @Override
191.         public void onStatusChanged(String provider, int status, Bundle extras) {
192.             // TODO Auto-generated method stub
193.         }
194.
195.         @Override
196.         public void onProviderEnabled(String provider) {
197.             // TODO Auto-generated method stub

```

```

198.         ShowLocationInfo(null);
199.     }
200.
201.     @Override
202.     public void onProviderDisabled(String provider) {
203.         // TODO Auto-generated method stub
204.         ShowLocationInfo(null);
205.     }
206.
207.     @Override
208.     public void onLocationChanged(Location location) {
209.         // TODO Auto-generated method stub
210.         ShowLocationInfo(location);
211.     }
212. };
213.
214. @Override
215. protected void onDestroy() { //关闭Activity 时停止接收GPS 定位信息
216.     // TODO Auto-generated method stub
217.     super.onDestroy();
218.     if (isLocating) { //如果正在定位
219.         locationManager.removeUpdates(listener); //停止定位
220.         isLocating = false;
221.     }
222. }
223.

```

- 网络请求模块之 POST /sign : 通过 POST 请求给服务器发送用户数据完成签到

```

224.     // 异步任务类
225.
226.     private class SignTask extends AsyncTask<Void, Void, String> {
227.
228.         private String studentId;
229.         private String studentName;
230.         private double latitude;
231.         private double longitude;
232.         private double altitude;
233.
234.         public SignTask(String studentId, String studentName, double latitude, double longitude, double altitude) {
235.             this.studentId = studentId;
236.             this.studentName = studentName;
237.             this.latitude = latitude;
238.             this.longitude = longitude;

```

```

239.         this.altitude = altitude;
240.     }
241.
242.     @Override
243.     protected String doInBackground(Void... voids) {
244.         try {
245.             String urlString = GlobalVariable.BASE_URL + "/sign";
246.             URL url = new URL(urlString);
247.             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
248.
249.             connection.setRequestMethod("POST");
250.             connection.setRequestProperty("Content-Type", "application/json");
251.             connection.setRequestProperty("Accept", "application/json");
252.             connection.setDoOutput(true);
253.
254.             String requestBody = "{ \"student_id\": \"\" + studentId + \"\", \"student_name\" : \"\" + studentName + \"\", \"latitude\": \"\" + latitude + \"\", \"longitude\": \"\" + longitude + \"\", \"altitude\": \"\" + altitude + \"\" }";
255.
256.             OutputStream outputStream = connection.getOutputStream();
257.             byte[] input = requestBody.getBytes(StandardCharsets.UTF_8);
258.             outputStream.write(input, 0, input.length);
259.             outputStream.close();
260.
261.             int responseCode = connection.getResponseCode();
262.
263.             if (responseCode == HttpURLConnection.HTTP_OK) {
264.                 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
265.                 String inputLine;
266.                 StringBuilder response = new StringBuilder();
267.
268.                 while ((inputLine = in.readLine()) != null) {
269.                     response.append(inputLine);
270.                 }
271.                 in.close();
272.
273.                 // 解析响应
274.                 JSONObject jsonResponse = new JSONObject(response.toString());
275.                 return jsonResponse.getString("position");
276.             } else {
277.                 return null;
278.             }
279.         } catch (Exception e) {

```

```

280.         e.printStackTrace();
281.         return null;
282.     }
283. }
284.
285. @Override
286. protected void onPostExecute(String position) {
287.     super.onPostExecute(position);
288.     if (position != null) {
289.         ToastUtil.showmsg(getApplicationContext(), "提交成功, 位置: " + position);
290.     } else {
291.         ToastUtil.showmsg(getApplicationContext(), "请求失败");
292.     }
293. }
294. }
295.
296. public void SignWithLocation(String student_name) {
297.     if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
298.         return;
299.     }
300.     position = locationManager.getLastKnownLocation(provider);
301.     if (position == null) {
302.         ToastUtil.showmsg(getApplicationContext(), "请先定位");
303.         return;
304.
305.     }
306.     String studentId = MainActivity.usrid;
307.     double latitude = position.getLatitude();
308.     double longitude = position.getLongitude();
309.     double altitude = position.getAltitude();
310.
311.     // 启动异步任务, 并传入参数
312.     new SignTask(studentId, student_name, latitude, longitude, altitude).execute();
313. }

```

AdminActivity 的通信机制也跟 LocationSign 一样, 用异步化的 HttpURLConnection 完成了 http 通信。另外 AdminActivity 里面比较关键的部分就是按钮更新:

GPS

Network

经度: 116.35025321

纬度: 39.95874773

高度: 60.8690185546875

加速度: 10.317723

方向: 0.0

Address[addressLines=[0:"北京市海淀区西土城路10号院-14号楼",1:"北京邮电大学",2:"北京邮电大学计算机科学与技术学院",3:"北京邮电大学电信工程学院教研室",4:"北京邮电大学继续教育学院实验室",5:"自动化学院实验室",6:"北京邮电大学-BUPT-QMUL"]]

添加新的签到地址

当管理员点击“添加新的签到地址”这个按钮，并显示一个输入框劲儿一个提交按钮时，我们先隐藏

1.地名/教室名/其他备注

2.规定为签到地址

代码实现：

```
1.         add_addr.setOnClickListener(new View.OnClickListener() {
2.             @Override
3.             public void onClick(View v) {
4.                 newplace.setVisibility(View.VISIBLE);
5.                 sign.setVisibility(View.VISIBLE);
6.                 add_addr.setVisibility(View.GONE);
7.             }
8.         });
```

CheckPlaceList，CheckSignList 的逻辑完全一样，但是处理的数据不一样，一个处理地点信息，另一个签到信息。我们就将 CheckSignList 举例吧：

此页面就两个功能 1.从服务器获取签到信息格式化输出到屏幕 2.如果清楚按钮被点击那么删除服务器数据库的同时清空屏幕

Oncreate 方法里面唯一干的事情就是调用 get\_sign\_information();获取签到信息

7:04  
成功获取签到信息：  
学生ID: 12345  
学生姓名: John Doe  
签到位置: 未知  
签到时间: 2024-06-07 16:14:56  
经度: 34.052235  
纬度: -118.243683  
海拔: 89  
  
学生ID: 2021211563  
学生姓名: 阿卜杜萨拉本  
签到位置: 927  
签到时间: 2024-06-07 19:04:03  
经度: 39.96205  
纬度: 116.34993  
海拔: 0.0

protected void onCreate(Bundle savedInstanceState) {  
  
super.onCreate(savedInstanceState);  
  
EdgeToEdge.enable(this);  
  
setContentView(R.layout.activity\_check\_sign\_list);  
  
ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {  
  
Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());  
  
v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);  
  
return insets;  
  
});



```

10.
11.
12.         // 初始化布局中的 TextView
13.         list = findViewById(R.id.sign_list);
14.
15.         delete_info=findViewById(R.id.deleteinfo);
16.
17.         delete_info.setOnClickListener(new View.OnClickListener() {
18.             @Override
19.             public void onClick(View v) {
20.                 delete_sign_information();
21.                 list.setText("");
22.             }
23.         });
24.         get_sign_information();
25.     }

```

● 获取签到信息模块：

```

1.
2.         public void get_sign_information() {
3.             // 启动异步任务获取签到信息
4.             new GetSignInfoTask().execute();
5.         }
6.
7.         // 内部类，用于从服务器获取签到信息的异步任务
8.         private class GetSignInfoTask extends AsyncTask<Void, Void, JSONArray> {
9.
10.             @Override
11.             protected JSONArray doInBackground(Void... voids) {
12.                 // 服务器接口地址
13.                 String urlString = GlobalVariable.BASE_URL + "/get_sign_info";
14.
15.                 try {
16.                     // 创建 URL 对象
17.                     URL url = new URL(urlString);
18.
19.                     // 创建 HTTP 连接对象
20.                     HttpURLConnection connection = (HttpURLConnection) url.openConnection();
21.
22.                     // 设置请求方法为 GET
23.                     connection.setRequestMethod("GET");
24.
25.                     // 设置请求头部信息
26.                     connection.setRequestProperty("Content-Type", "application/json");

```

```
25.         connection.setRequestProperty("Accept", "application/json");
26.
27.         // 获取响应状态码
28.         int responseCode = connection.getResponseCode();
29.
30.         // 如果响应状态码为 200, 表示请求成功
31.         if (responseCode == HttpURLConnection.HTTP_OK) {
32.             // 读取响应数据流
33.             InputStream inputStream = connection.getInputStream();
34.             BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream, StandardCharsets.UTF_8));
35.             StringBuilder response = new StringBuilder();
36.             String line;
37.             while ((line = bufferedReader.readLine()) != null) {
38.                 response.append(line);
39.             }
40.             bufferedReader.close();
41.
42.             // 解析响应数据为 JSON 格式
43.             return new JSONArray(response.toString());
44.         } else {
45.             Log.e("GetSignInfoTask", "请求失败, 状态码: " + responseCode);
46.             // 如果请求失败, 返回 null
47.             return null;
48.         }
49.     } catch (IOException | JSONException e) {
50.         Log.e("GetSignInfoTask", "请求异常", e);
51.         // 如果请求异常, 返回 null
52.         return null;
53.     }
54. }
55.
56. @Override
57. protected void onPostExecute(JSONArray signInfoArray) {
58.     super.onPostExecute(signInfoArray);
59.     if (signInfoArray != null) {
60.         // 成功获取签到信息, 可以在这里处理数据, 例如显示在界面上
61.
62.         StringBuilder signInfoStringBuilder = new StringBuilder();
63.         signInfoStringBuilder.append("成功获取签到信息: \n");
64.
65.         try {
66.             for (int i = 0; i < signInfoArray.length(); i++) {
67.                 JSONObject signInfoObject = signInfoArray.getJSONObject(i);
```

```

68.         String latitude = signInfoObject.getString("latitude");
69.         String longitude = signInfoObject.getString("longitude");
70.         String altitude = signInfoObject.getString("altitude");
71.         String studentId = signInfoObject.getString("student_id");
72.         String studentName = signInfoObject.getString("student_name");
73.         String position=signInfoObject.getString("position");
74.         String time=signInfoObject.getString("time");
75.         // 格式化输出每个签到信息
76.         String signInfo = "学生 ID: " + studentId + "\n"
77.             + "学生姓名: " + studentName + "\n"
78.             + "签到位置: " + position+ "\n"
79.             + "签到时间: " + time+ "\n"
80.             + "纬度: " + latitude + "\n"
81.             + "经度: " + longitude + "\n"
82.             + "海拔: " + altitude + "\n\n";
83.
84.         signInfoStringBuilder.append(signInfo);
85.     }
86.     } catch (JSONException e) {
87.         e.printStackTrace();
88.     }
89.
90.     String formattedSignInfo = signInfoStringBuilder.toString();
91.     Log.d("GetSignInfoTask", formattedSignInfo);
92.
93.     // 这里您可以将格式化后的签到信息显示在界面的合适位置
94.     // 将其设置为 TextView 的文本内容
95.     list.setText(formattedSignInfo);
96.     Log.d("GetSignInfoTask", "成功获取签到信息: " + signInfoArray.toString());
97. } else {
98.     // 获取签到信息失败, 可以在这里处理错误情况
99.     Log.e("GetSignInfoTask", "获取签到信息失败");
100. }
101. }
102. }

```

## ● 删除签到信息模块:

```

1.
2.     public void delete_sign_information() {
3.         // 启动异步任务获取签到信息
4.         new DeleteSignInfoTask().execute();
5.     }

```

```
6.      // 内部类，用于从服务器获取签到信息的异步任务
7.      private class DeleteSignInfoTask extends AsyncTask<Void, Void, Boolean> {
8.
9.          @Override
10.         protected Boolean doInBackground(Void... voids) {
11.             // 服务器接口地址
12.
13.             String urlString = GlobalVariable.BASE_URL + "/delete_sign_info";
14.
15.             try {
16.                 // 创建 URL 对象
17.                 URL url = new URL(urlString);
18.
19.                 // 创建 HTTP 连接对象
20.                 HttpURLConnection connection = (HttpURLConnection) url.openConnection();
21.
22.                 // 设置请求方法为 POST
23.                 connection.setRequestMethod("POST");
24.
25.                 // 设置请求头部信息
26.                 connection.setRequestProperty("Content-Type", "application/json");
27.                 connection.setRequestProperty("Accept", "application/json");
28.                 // 启用输出流
29.                 connection.setDoOutput(true);
30.
31.
32.                 // 构建请求体数据
33.                 String requestBody = "{\"name\": \"007\"}";
34.
35.                 // 将请求体数据写入输出流
36.                 OutputStream outputStream = null;
37.                 try {
38.                     outputStream = connection.getOutputStream();
39.                     byte[] input = requestBody.getBytes(StandardCharsets.UTF_8);
40.                     outputStream.write(input, 0, input.length);
41.                 } catch (IOException e) {
42.                     e.printStackTrace();
43.                 } finally {
44.                     if (outputStream != null) {
45.                         try {
46.                             outputStream.close();
47.                         } catch (IOException e) {
48.                             e.printStackTrace();
49.                         }
49.                     }

```

```

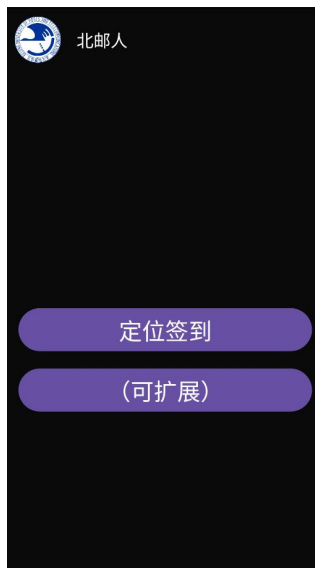
50.         }
51.     }
52.     // 获取响应状态码
53.     int responseCode = connection.getResponseCode();
54.
55.     // 如果响应状态码为 200，表示请求成功
56.     if (responseCode == HttpURLConnection.HTTP_OK) {
57.
58.         return true;
59.     } else {
60.         Log.e("GetSignInfoTask", "请求失败，状态码: " + responseCode);
61.         // 如果请求失败，返回 null
62.         return false;
63.     }
64. } catch (Exception e) {
65.     // 异常处理
66.     e.printStackTrace();
67.     return false;
68. }
69. }
70.
71. @Override
72. protected void onPostExecute(Boolean success) {
73.     super.onPostExecute(success);
74.     if (success) {
75.         // 提示用户提交成功
76.         ToastUtil.showmsg(getApplicationContext(), "消除成功");
77.     } else {
78.         // 提示用户请求失败
79.         ToastUtil.showmsg(getApplicationContext(), "请求失败");
80.     }
81. }
82. }

```

## 4. 系统可能的扩展

### 4.1 扩展签到方式

当前系统提供了基于位置的签到方式，未来可以扩展更多类型的签到方式以适应不同的使用场景。Function\_activity 界面预留了足够的空间来扩展/添加其他类型的签到功能，比如 NFC 签到，二维码签到等。开发者只需要扩展对应功能并在服务器建立新的接口接受并记录 http 请求即可。



## 4.2 增加用户角色和权限管理

系统可以增加更多的用户角色，并为每个角色分配不同的权限，以满足不同用户的需求：

- 教师角色：可以查看学生的签到情况，生成考勤报告。
- 访客角色：可以进行特定的签到，适用于对外开放的活动或会议。

## 4.3 集成第三方服务

与第三方服务集成，以提高系统的功能性和用户体验：

- 天气预报服务：根据签到地点的天气情况，提醒用户注意天气变化。
- 交通信息服务：提供签到地点的交通信息，帮助用户规划出行路线。

## 4.4 增强数据分析和报告功能

系统可以提供更深入的数据分析和报告功能，帮助管理员和教师更好地了解签到情况：

- 签到统计报告：生成签到统计图表，展示签到频率、迟到率等关键指标。
- 考勤异常分析：对异常签到数据进行分析，识别潜在的问题并提出改进建议。

## 4.5 引入机器学习算法

利用机器学习算法对签到数据进行分析，以预测和识别签到模式和趋势：

- 签到预测：根据历史数据预测未来的签到情况，帮助管理员进行资源分配和规划。
- 异常检测：自动识别异常签到行为，提高系统的安全性。

## 4.7 支持多语言和国际化

为系统增加多语言支持，满足不同国家和地区用户的需求：

- 语言切换：允许用户根据自己的语言偏好切换系统界面语言。

## 4.8 增强安全性和隐私保护

加强系统的安全性和隐私保护措施，确保用户数据的安全：

- 数据加密：对存储和传输的数据进行加密处理。
- 隐私设置：提供隐私设置选项，让用户能够控制自己的个人信息。

通过这些扩展，"Zigzag 签到系统"将能够更好地满足不同用户的需求，提供更加全面和高效的签到解决方案。

# 5. 总结体会

个人开发“Zigzag 签到系统”是一个全面而深入的学习经历，以下是从这个项目获得的一些关键体会：

## 5.1 自我学习的深化

在整个开发过程中，我通过自学掌握了多种技术栈，包括 Android 开发、后端服务以及数据库管理等。这个过程加深了我对移动互联网技术的理解，并提高了自主解决问题的能力。

## 5.2 项目规划的重要性

我意识到良好的项目规划对于个人项目的成功至关重要。从需求分析到架构设计，合理的规划帮助我保持开发进度，并确保最终产品符合预期目标。

## 5.3 全栈开发的挑战与收获

作为全栈开发者，我体验了从前端界面到后端逻辑的全过程。这不仅提高了我的技术广度，也加深了对系统整体运作的理解。

## 5.4 用户中心设计的认识

我认识到以用户为中心的设计对于开发成功产品的重要性。在设计界面和功能时，始终考虑用户体验，努力打造直观、易用的系统。

## 5.5 数据处理与安全的重要性

在处理用户数据时，我学会了实施适当的安全措施来保护数据的安全性和用户隐私，这是对用户负责的体现。比如我把最初的 GET /delete\_sign\_info 改成了 POST /delete\_sign\_info 并且添加参数进行校验，不然任何人都可以进行删除服务器数据。

## 5.6 网络编程的实践经验

通过实现系统的网络通信部分，我获得了处理 HTTP 请求、响应以及数据传输的实践经验，这对于构建现代网络应用至关重要。

## 5.7 持续迭代与改进

在整个开发周期中，我学会了通过持续迭代来改进产品。每个版本都基于 debug 和测试结果

进行优化。

### **5.8 独立解决问题的能力**

面对开发中的难题，我锻炼了独立思考和解决问题的能力。这不仅提升了我的技术技能，也增强了解决复杂问题的信心。

个人开发“Zigzag 签到系统”不仅是一个技术实现的过程，更是一次个人能力全面提升的旅程。这些体会将指导我未来在技术及其他领域的成长和发展。