

浙江工业大学

移动应用开发

课程设计报告



电话簿管理软件

成 员	王程飞 201806061219 胡皓睿 201806061108
班 级	软工 1805
任课教师	邱杰凡
提交日期	2021 年 6 月 29 日

1. 目录

2.	功能说明.....	4
2.1.	实验题目和要求.....	4
2.2.	功能说明.....	4
3.	用户界面设计.....	5
4.	数据库设计.....	7
4.1.	客户端.....	7
4.2.	服务端.....	9
4.2.1.	User 表	9
4.2.2.	Person 表	10
4.2.3.	UploadFile 表	11
5.	程序模块设计.....	12
5.1.	客户端.....	12
5.2.	服务端.....	13
5.2.1.	Controller.....	13
5.2.2.	Service.....	13
5.2.3.	Mapper.....	13
5.2.4.	Util.....	14
6.	文件结构及用途.....	15
6.1.	客户端.....	15
6.1.1.	app 文件夹	15
6.1.2.	gradle 文件夹	17
6.1.3.	.gitignore 文件	17
6.1.4.	settings.gradle 文件	17
6.2.	服务端.....	18
6.2.1.	Controllers 文件夹	18
6.2.2.	Mappers 文件夹	18
6.2.3.	Migrations 文件夹	18
6.2.4.	Model 文件夹	18

6.2.5. Services 文件夹	18
6.2.6. Utils 文件夹	19
7. 创新点及所用技术.....	19
7.1. 基于 NFC 标签与二维码的信息分享.....	19
7.2. 基于四状态模型的通讯录信息云同步.....	20
7.3. 基于自定义绘制的 View 实现侧边栏.....	20
7.4. 基于 GitHub Action 的持续集成.....	20
8. 总结与思考.....	21

2. 功能说明

2.1. 实验题目和要求

- 开发一个电话簿管理软件利用原语级 SQLite 设计一个通讯录，要求包含：姓名，电话，工作单位以及家庭住址信息。（10 分）
- 利用原语级 SQLite，实现对现有通信录的新增、查询（按名字搜索）、删除条目。（10 分）
- 在通讯录中通过选中电话号码，直接拨打电话。（10 分）
- 构思一个与后台服务（线程）相关的功能模块（10 分）
- 功能创新或手段创新（20 分）
- 现场知识问答。（10 分）
- 期末大作业报告。（30 分）

2.2. 功能说明

本电话簿应用使用 SQLite 进行存储和读取用户保存的通讯录信息，可以对通讯录进行按首字母进行分组显示，并通过侧边栏导航。用户可以添加、修改、删除、模糊查找通讯录联系人，并通过二维码和 NFC TAG 进行分享。

3. 用户界面设计

该系统有四个主要的界面，各界面的展示如下：





姓名 蔡德泽

电话 19896406869

电子邮件 caideze@nfcv.xyz

家庭住址 浙江工业大学屏峰校区

工作单位 浙江工业大学屏峰校区

使用云同步账户登录

用户名

0/20

密码



登录

注册账户

忘记密码

同步

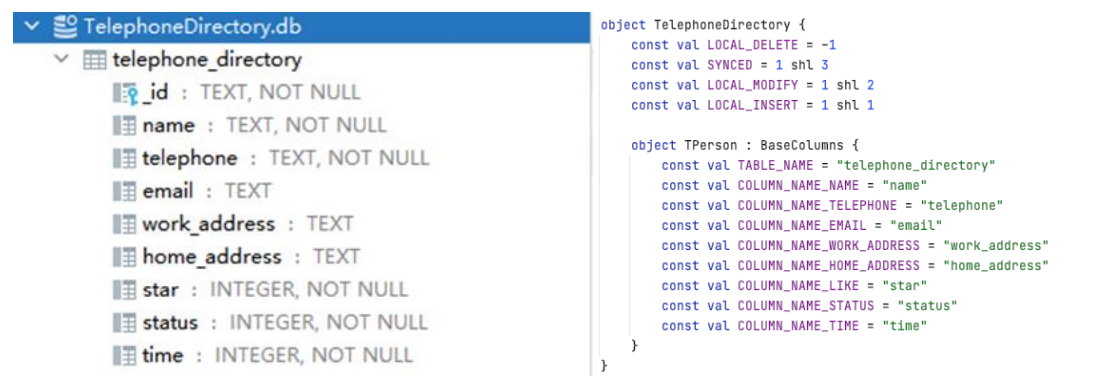
登录

4. 数据库设计

本项目包括客户端与服务端，我们会分别对其进行介绍。

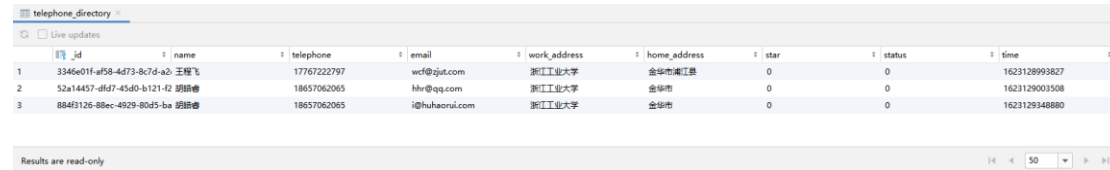
4.1. 客户端

在客户端中，我们使用了 SQLite 作为数据库引擎。客户端使用了一张数据表，存放了联系人有关信息。数据表设计如下：



```
object TelephoneDirectory {
    const val LOCAL_DELETE = -1
    const val SYNCED = 1 shl 3
    const val LOCAL_MODIFY = 1 shl 2
    const val LOCAL_INSERT = 1 shl 1

    object TPerson : BaseColumns {
        const val TABLE_NAME = "telephone_directory"
        const val COLUMN_NAME_NAME = "name"
        const val COLUMN_NAME_TELEPHONE = "telephone"
        const val COLUMN_NAME_EMAIL = "email"
        const val COLUMN_NAME_WORK_ADDRESS = "work_address"
        const val COLUMN_NAME_HOME_ADDRESS = "home_address"
        const val COLUMN_NAME_LIKE = "star"
        const val COLUMN_NAME_STATUS = "status"
        const val COLUMN_NAME_TIME = "time"
    }
}
```

	_id	name	telephone	email	work_address	home_address	star	status	time
1	3346e01f-af58-4d73-8c7d-a2	王辉飞	17767222797	wcf@jst.com	浙江工业大学	金华市东江里	0	0	1623128993827
2	52a14457-df47-45d0-b121-f2	胡皓睿	18657062065	hlu@qq.com	浙江工业大学	金华市	0	0	1623129003508
3	884f3126-88ec-4929-80d5-ba	胡皓睿	18657062065	ihuhacn@163.com	浙江工业大学	金华市	0	0	1623129348880

其中建表语句如下：

```
create table telephone_directory(
    _id varchar(36) not null primary key,
    name varchar(64) not null,
    telephone varchar(20) not null,
    email varchar(20),
    work_address varchar(64),
    home_address varchar(64),
    star integer not null,
    status integer not null,
    time integer not null
)
```

本地 SQLiteOpenHelper 采用单例模式，保证不同代码在操作数据库时数据的同步：

```
companion object {
    private const val DATABASE_NAME = "TelephoneDirectory.db"

    private const val DATABASE_VERSION = 1
}
```

```

private var helper: TelephoneDirectoryDbHelper? = null

@Synchronized
fun getHelper(context: Context): TelephoneDirectoryDbHelper {
    val helper: TelephoneDirectoryDbHelper =
        this.helper ?: TelephoneDirectoryDbHelper(context)
    this.helper = helper
    return helper
}
}

```

对于数据库操作，我们采用了数据库的增删改查语句，使用 ContentValues 进行包装数据，并执行相应的操作：

```

companion object {
    fun insert(context: Context, vararg persons: Person) {...}

    fun insertCloud(context: Context, vararg persons: Person) {...}

    fun delete(context: Context, person: Person): Int {...}

    fun deleteCloud(context: Context, person: Person): Int {...}

    fun update(context: Context, person: Person): Int {...}

    fun updateCloud(context: Context, person: Person): Int {...}

    fun synced(context: Context, person: Person): Int {...}

    fun all(context: Context): List<Person> {...}

    fun allWithStatus(context: Context): List<Person> {...}

    fun clear(context: Context): Int {...}

    fun select(context: Context, value: String): Person? {...}

    fun likeName(context: Context, value: String): ArrayList<Person> {...}

    fun likeAddress(context: Context, value: String): ArrayList<Person> {...}
}

```

以新增联系人为例：

```

fun insert(context: Context, vararg persons: Person) {
    TelephoneDirectoryDbHelper.getHelper(context).let { helper: TelephoneDirectoryDbHelper ->
        helper.writableDatabase.use { db: SQLiteDatabase ->
            for (person in persons) {
                person.status = TelephoneDirectory.LOCAL_INSERT
                person.time = System.currentTimeMillis()
                val contentValues = ContentValues()
                contentValues.put(BaseColumns._ID, UUID.randomUUID().toString())
                contentValues.put(TPerson.COLUMN_NAME_NAME, person.name)
                contentValues.put(TPerson.COLUMN_NAME_TELEPHONE, person.telephone)
            }
        }
    }
}

```



```

        contentValues.put(TPerson.COLUMN_NAME_EMAIL, person.email)
        contentValues.put(TPerson.COLUMN_NAME_WORK_ADDRESS, person.workAddress)
        contentValues.put(TPerson.COLUMN_NAME_HOME_ADDRESS, person.homeAddress)
        contentValues.put(TPerson.COLUMN_NAME_LIKE, person.like)
        contentValues.put(TPerson.COLUMN_NAME_STATUS, person.status)
        contentValues.put(TPerson.COLUMN_NAME_TIME, person.time)
        db.insert(TPerson.TABLE_NAME, null, contentValues)
    }
}
}
}

```

4.2. 服务端

服务端使用 Microsoft SQL Server 作为数据库，使用了 EF Core 技术，实现从数据实体自动生成数据库。服务端主要包括三张数据表。

4.2.1. User 表

User 表存放了注册用户有关的信息。其 Model 定义如下：

[Table(name: "User")]

34 usages HHR +1 1 ext method 9 exposing APIs

public class User

{

8 usages

[Key] [JsonProperty("userId")] public int UserId { get; set; }

11 usages

[JsonProperty("password")] public string Password { get; set; } = "";

9 usages

[JsonProperty("username")] public string Username { get; set; } = "";

[JsonProperty("email")] public string? Email { get; set; }

[JsonProperty("sex")] public Sex? Sex { get; set; }

[JsonProperty("description")] public string? Description { get; set; }

[JsonProperty("credit")] public int Credit { get; set; }

1 usage

[JsonProperty("avatarId")] public int AvatarId { get; set; } = -1;











[JsonProperty("datetime")] public DateTime? Birthday { get; set; }

8 usages

[JsonProperty("token")] public string? Token { get; set; }

}

使用 EF Core 提供的迁移工具，我们可以得到以下数据表：

User	
 UserId	int
 Password	nvarchar(max)
 UserName	nvarchar(max)
 Email	nvarchar(max)
 Sex	int
 Description	nvarchar(max)
 Credit	int
 AvatarId	int
 Birthday	datetime2
 Token	nvarchar(max)

4.2.2. Person 表

Person 表存放了用户备份的通讯录信息。其 Model 定义如下：

```
[Table(name: "Person")]
```

 17 usages  1 inheritor  HHR +1  2 exposing APIs

```
public class Person
```

```
{
```

```
    [JsonIgnore][Key] public int Id { get; set; }
```

 10 usages

```
    [JsonProperty("id")] public string PersonId { get; set; } = null!;
```

 2 usages

```
    [JsonProperty("name")] public string Name { get; set; } = null!;
```

 2 usages

```
    [JsonProperty("telephone")] public string Telephone { get; set; } = null!;
```

 2 usages

```
    [JsonProperty("email")] public string? Email { get; set; }
```

 2 usages

```
    [JsonProperty("workAddress")] public string? WorkAddress { get; set; }
```

 2 usages

```
    [JsonProperty("homeAddress")] public string? HomeAddress { get; set; }
```

 2 usages

```
    [JsonProperty("like")] public int Like { get; set; }
```

 6 usages











```
    [JsonProperty("time")] public long Time { get; set; }
```

 2 usages

```
    [JsonIgnore] public int Owner { get; set; }
```

```
}
```

完成迁移之后，数据库如下：

Person	
 Id	int
 Name	nvarchar(max)
 Telephone	nvarchar(max)
 Email	nvarchar(max)
 WorkAddress	nvarchar(max)
 HomeAddress	nvarchar(max)
 Like	int
 Owner	int
 PersonId	nvarchar(max)
 Time	bigint




4.2.3. UploadFile 表

该表包含了用户主动上传的文件的有关信息。

```
[Table(name: "UploadFile")]
```

6 usages HHR

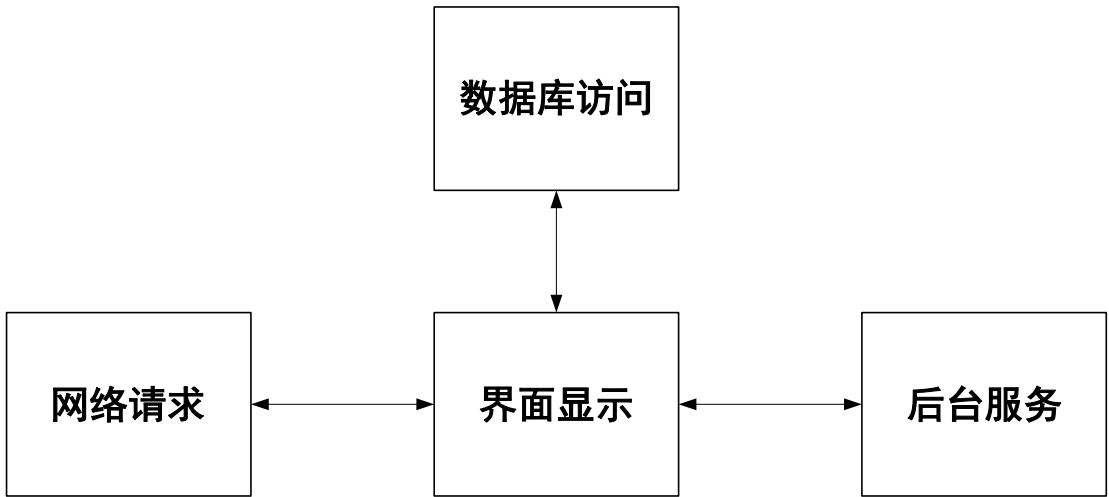
```
public class UploadFile
{
     2 usages
    ... [Key] [JsonProperty("fileId")] public int FileId { get; set; }
     1 usage
    ... [JsonProperty("fileType")] public FileType FileType { get; set; }
     4 usages
    ... [JsonProperty("fileUuid")] public string FileGuid { get; set; } = null!;
}
```

UploadFile	
 FileId	int
 FileType	int
 FileGuid	nvarchar(max)

5. 程序模块设计

5.1. 客户端

客户端程序主要包含 数据库访问，界面显示，网络请求，后台服务几个模块，各模块之间互相关联，构成了一个有机的整体。

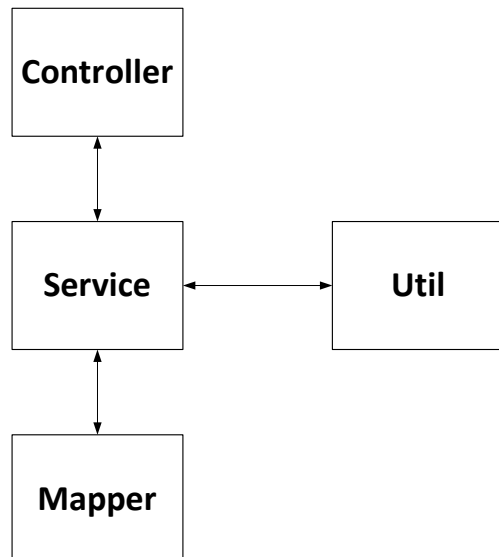


用户通过点击界面上的按钮，可以触发特定的功能，例如在点击删除按钮时，会触发数据库访问模块的删除操作，在 SQLite 数据库中删除掉对应的记录。在进行点击网络同步的按钮时，retrofit2 组件会通过 okhttp3，对后端 API 进行请求。在收到后端服务的回调后，通过 Handler 与 Looper 进行消息传递，将同步的结果渲染至界面上进行显示。

启动 APP 时主界面的列表会通过适配器加载数据库的数据，点击数据项时进入联系人详情界面模块进行查看详情，可以在该页进行修改、删除、分享等操作，点击主页的“+”按钮可以选择不同的功能。点击云同步进入用户模块，可以进行登录和退出以及云同步数据。云同步模块通过后台服务和数据库记忆网络后端服务器进行交互。

5.2. 服务端

服务端程序主要包含 Controller, Service, Mapper 三层，另外提供了 Util 工具包保存了常用的方法，用于 Service 层调用。



5.2.1. Controller

Controller 的代码用于接收网络请求，将 HTTP 请求的报文转化为 C# 中的变量，进行一个初步的校验，然后将请求转发至 Service 层。

5.2.2. Service

Service 用于处理复杂请求。Service 层会调用 Mapper 层执行 CURD 的操作，或是调用 Util 层进行一些复制而又通用的逻辑。

5.2.3. Mapper

Mapper 层用于数据库映射。在这一层中，我们使用了微软公司的 EF Core 进行开发。EF Core 使用 LINQ 等技术，将对数据库的访问简化，使开发者可以像操作本地数据集合一样操作数据库。

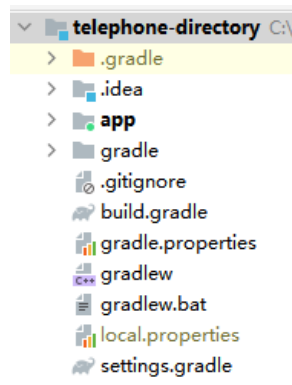
5.2.4. Util

Util 包含了一些通用的工具类，用于实现一个可复制而又固定的功能。

6. 文件结构及用途

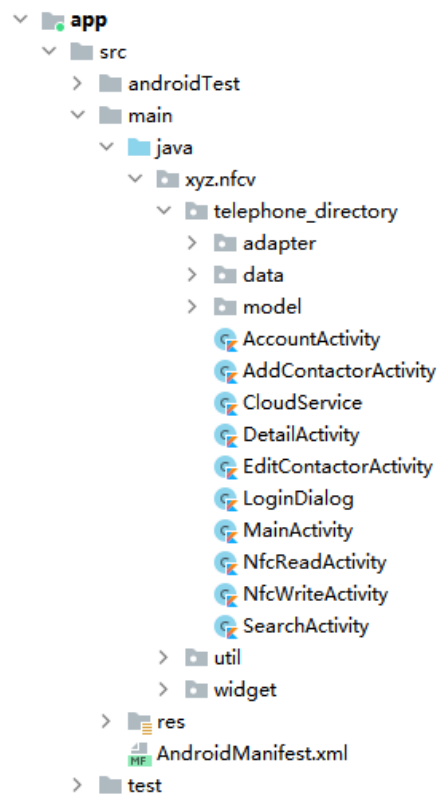
6.1. 客户端

项目主要包含以下文件。



6.1.1. app 文件夹

app 文件夹存放了软件主要的代码。



文件说明如下：

包名称	文件名	说明
.util	Base.kt	对于字符串和文件的处理类
.widget	Header.kt	侧边栏控件首字母枚举
	RoundImageView.kt	自定义圆角 ImageView
	SideBar.kt	自定义侧边栏
.telephone_directory	AccountActivity.kt	用户登录和云同步活动
	AddContactorActivity.kt	添加和编辑联系人活动
	EditContactorActivity.kt	
	CloudService.kt	云同步后台服务
	MainActivity.kt	主界面，展示联系人列表
	LoginDialog.kt	登录对话框
	QRCodeDialog.kt	二维码展示对话框
	NfcWriteActivity.kt	NFC 标签写入和读取活动
	NfcReadActivity.kt	
	SearchActivity.kt	模糊搜索活动
.adapter	UriBrowserActivity.kt	二维码扫描跳转活动
	ContactorListAdapter.kt	主界面的联系人列表适配器
	SearchContactorAdapter.kt	搜索界面的联系人列表适配器
.model	PeopleGroupData.kt	联系人按首字母分组的 model
	User.kt	云同步用户 model
	Person.kt	联系人 model 及数据库操作方法
.data	Account.kt	云同步用户操作类
	CloudApi.kt	云同步 API 及处理类
	Cloud.kt	
	TelephoneDirectoryDbHelper.kt	SQLiteOpenHelper 的实现

6.1.2. gradle 文件夹

gradle 文件夹保存了 gradle 所使用的二进制文件

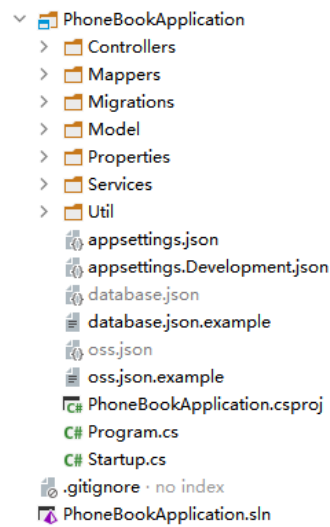
6.1.3. .gitignore 文件

.gitignore 文件存放了 git 的配置信息，用于存储 git 的忽略列表，不与远程仓库进行同步。

6.1.4. settings.gradle 文件

settings.gradle 文件存储了 gradle 的全局配置信息。

6.2. 服务端



6.2.1. Controllers 文件夹

存放了与 Controller 有关的代码

6.2.2. Mappers 文件夹

存放了 Mapper 有关的代码，提供了访问数据库的能力

6.2.3. Migrations 文件夹

存放了 EF Core 迁移有关的代码

6.2.4. Model 文件夹

类似于 Java 的 POJO，存储了数据模型

6.2.5. Services 文件夹

存放了 Service 层有关的代码

6.2.6. Utils 文件夹

存放了工具类的代码

7. 创新点及所用技术

7.1. 基于 NFC 标签与二维码的信息分享

一般来说，安卓开发者会使用系统标准接口进行信息的分享。在保留了标准的接口的同时，我们也增加了两种更方便的信息共享方式。我们可以将联系人信息序列化后，存入一张符合 ISO/IEC 14443, Type A 标准的卡片中，即完成了信息的录入。在另外一台支持 NFC 功能的设备上读取，即可将卡片中存储的通讯录信息进行反序列化，其结果可以直接存入该用户的通讯录列表中。

同时，对于没有 NFC 功能的较低端设备，我们也提供了使用二维码分享及录入的功能。我们可以将通讯录相关的数据存放至二维码中，完成信息的分享，而在接收端通过扫描二维码的方式，将信息存入通讯录中。

7.2. 基于四状态模型的通讯录信息云同步

为了解决状态标记的问题，我们引入了一个包含四种状态的状态机模型，其包括 LOCAL_DELETE、SYNCED、LOCAL_MODIFY、LOCAL_INSERT 四种状态。通过标记状态，我们即可知道哪些记录需要被同步，哪些记录已经被同步，即可轻松完成整个通讯录信息云同步的功能。

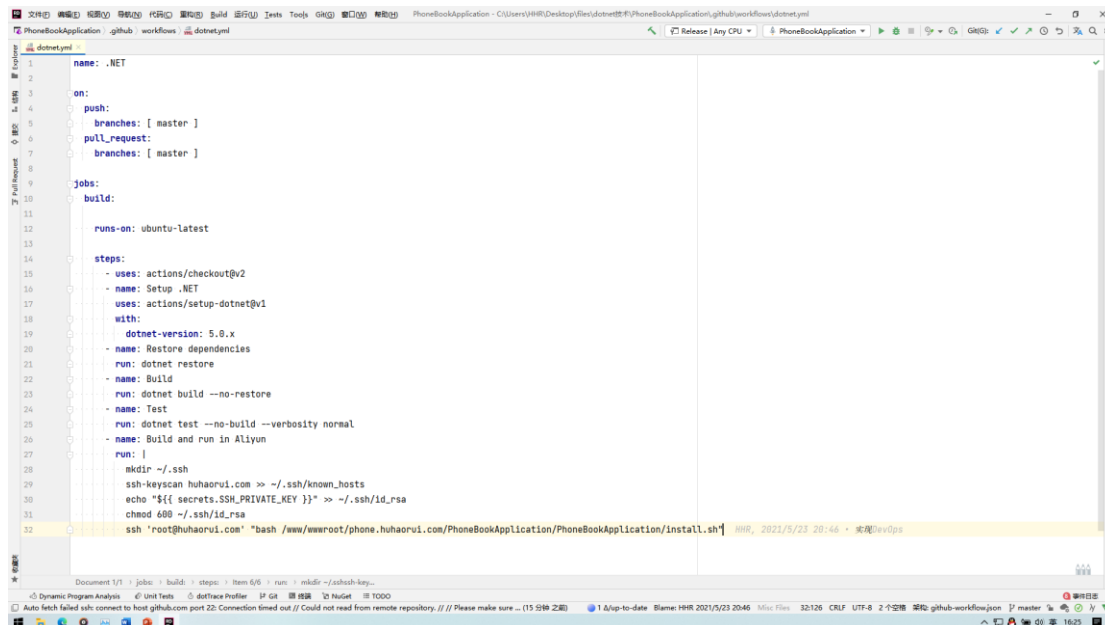
7.3. 基于自定义绘制的 View 实现侧边栏

联系人界面的侧边导航栏是我们通过自定义 View 绘制的，通过实现 View 的 init、onMeasure、onDraw 等方法绘制垂直的侧边栏，并在 onTouchEvent 方法中获取相对坐标计算触摸点，通过与 ExpandableListView 的双向绑定来更新状态和快捷导航。

7.4. 基于 GitHub Action 的持续集成

在通常的开发流程中，在建立一个新版本之后，我们需要对其进行打包，上传的工作，这其中可能会引发一些不必要的麻烦与错误。一个有效的解决方法是使用持续集成，自动的进行代码发布与版本更新。

在这里，我们使用了 GitHub 提供的服务，它可以自动的在每个版本被提交时，执行一段特定的代码以完成自动的构建与部署。



8. 总结与思考

通过这次课程设计，我们学会了基本的安卓应用开发知识，能够熟练使用 Android Studio 进行安卓应用程序的开发。掌握了运用利用原语级 SQLite 进行数据库的增删改查操作的技能，加深了自己对 SQLite 数据库的理解。掌握了调用安卓设备的硬件设备如 NFC 等来进行外界交互。学会了通过网络与服务器进行交互进行云同步，学会了二维码的相关使用方法。学会了安卓基本的开发规范，为以后更进一步的开发奠定了坚实的基础。