卡通人物

中度可信度描述已自动生成

**移动应用开发**

**课程设计报告**

徽标

描述已自动生成

**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 |  |

目录

[2. 功能说明 4](#_Toc75018938)

[2.1. 实验题目和要求 4](#_Toc75018939)

[2.2. 功能说明 4](#_Toc75018940)

[3. 用户界面设计 5](#_Toc75018941)

[4. 数据库设计 7](#_Toc75018942)

[4.1. 客户端 7](#_Toc75018943)

[4.2. 服务端 9](#_Toc75018944)

[4.2.1. User表 9](#_Toc75018945)

[4.2.2. Person表 10](#_Toc75018946)

[4.2.3. UploadFile表 11](#_Toc75018947)

[5. 程序模块设计 12](#_Toc75018948)

[5.1. 客户端 12](#_Toc75018949)

[5.2. 服务端 13](#_Toc75018950)

[5.2.1. Controller 13](#_Toc75018951)

[5.2.2. Service 13](#_Toc75018952)

[5.2.3. Mapper 13](#_Toc75018953)

[5.2.4. Util 14](#_Toc75018954)

[6. 文件结构及用途 15](#_Toc75018955)

[6.1. 客户端 15](#_Toc75018956)

[6.1.1. app文件夹 15](#_Toc75018957)

[6.1.2. gradle文件夹 17](#_Toc75018958)

[6.1.3. .gitignore文件 17](#_Toc75018959)

[6.1.4. settings.gradle文件 17](#_Toc75018960)

[6.2. 服务端 18](#_Toc75018961)

[6.2.1. Controllers文件夹 18](#_Toc75018962)

[6.2.2. Mappers文件夹 18](#_Toc75018963)

[6.2.3. Migrations文件夹 18](#_Toc75018964)

[6.2.4. Model文件夹 18](#_Toc75018965)

[6.2.5. Services文件夹 18](#_Toc75018966)

[6.2.6. Utils文件夹 19](#_Toc75018967)

[7. 创新点及所用技术 19](#_Toc75018968)

[7.1. 基于NFC标签与二维码的信息分享 19](#_Toc75018969)

[7.2. 基于四状态模型的通讯录信息云同步 20](#_Toc75018970)

[7.3. 基于自定义绘制的View实现侧边栏 20](#_Toc75018971)

[7.4. 基于GitHub Action的持续集成 20](#_Toc75018972)

[8. 总结与思考 21](#_Toc75018973)

# 功能说明

## 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）

## 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。

# 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：

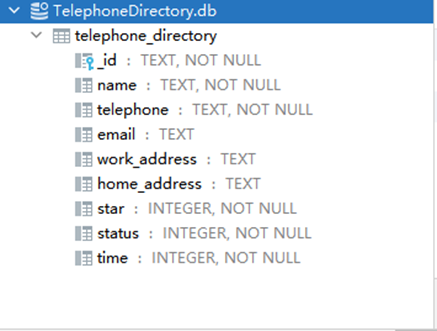
  

# 数据库设计

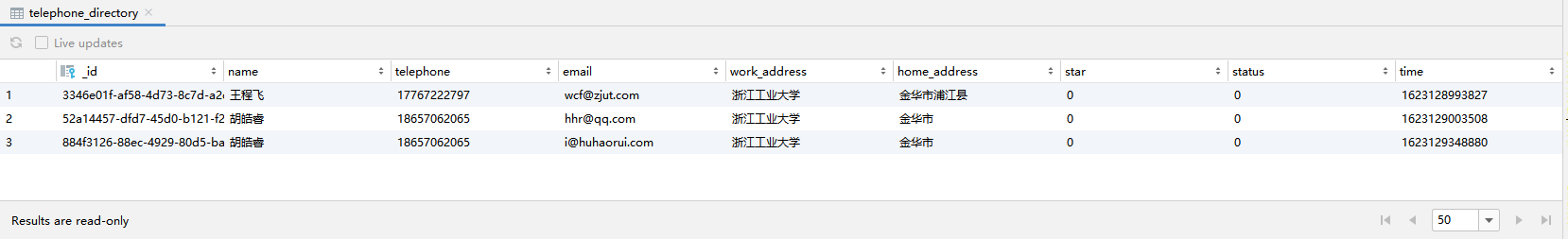
本项目包括客户端与服务器端，我们会分别对其进行介绍。

## 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：

文本

描述已自动生成



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：

图形用户界面, 文本

描述已自动生成

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)  
 }  
 **}  
 }**}

## 服务端

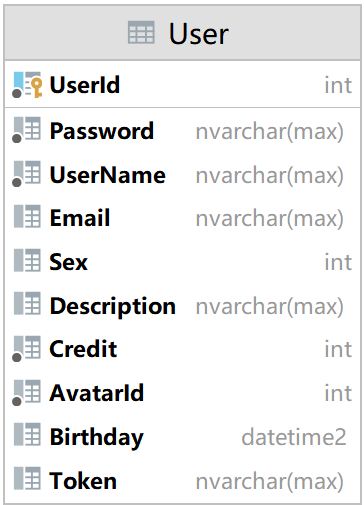
服务端使用 Microsoft SQL Server作为数据库，使用了EF Core技术，实现从数据实体自动生成数据库。服务端主要包括三张数据表。

### User表

User表存放了注册用户有关的信息。其Model定义如下：



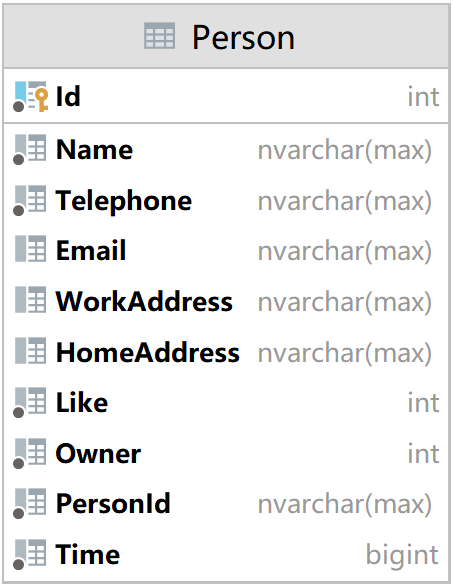
使用 EF Core 提供的迁移工具，我们可以得到以下数据表：



### Person表

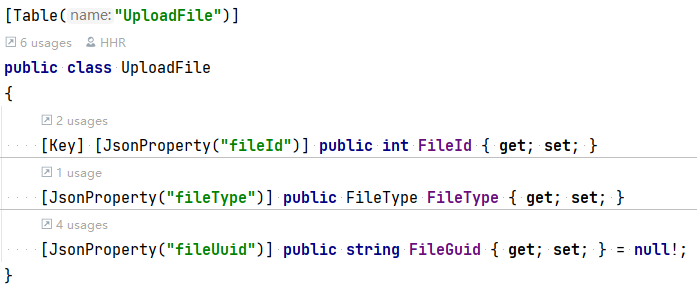
Person表存放了用户备份的通讯录信息。其Model定义如下：  

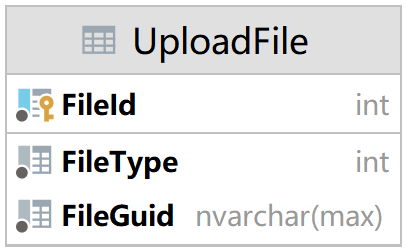

完成迁移之后，数据库如下：



### UploadFile表

该表包含了用户主动上传的文件的有关信息。





# 程序模块设计

## 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。

## 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。



### 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。

### 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。

### 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。

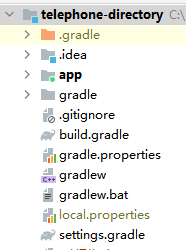
### Util

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。

# 文件结构及用途

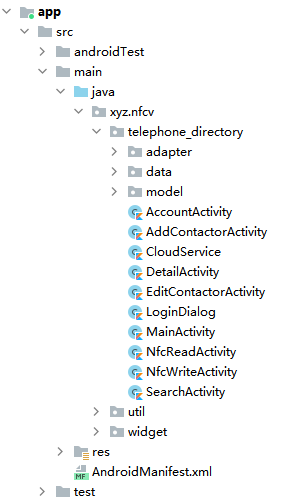
## 客户端

项目主要包含以下文件。



### 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  EditContactorActivity.kt | 添加和编辑联系人活动 |
| 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  NfcReadActivity.kt | NFC标签写入和读取活动 |
| SearchActivity.kt | 模糊搜索活动 |
| UriBrowserActivity.kt | 二维码扫描跳转活动 |
| .adapter | ContactorListAdapter.kt  SearchContactorAdapter.kt | 主界面的联系人列表适配器  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  Cloud.kt | 云同步API及处理类 |
| TelephoneDirectoryDbHelper.kt | SQLiteOpenHelper的实现 |

### gradle文件夹

gradle文件夹保存了gradle所使用的二进制文件

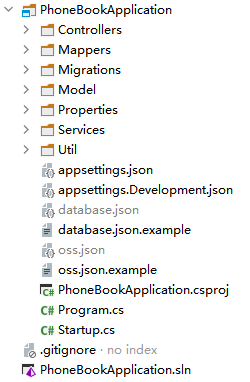
### .gitignore文件

.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。

### settings.gradle文件

settings.gradle文件存储了gradle的全局配置信息。

## 服务端



### Controllers文件夹

存放了与Controller有关的代码

### Mappers文件夹

存放了Mapper有关的代码，提供了访问数据库的能力

### Migrations文件夹

存放了 EF Core 迁移有关的代码

### Model文件夹

类似于Java的POJO，存储了数据模型

### Services文件夹

存放了Service层有关的代码

### Utils文件夹

存放了工具类的代码

# 创新点及所用技术

## 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功能的设备上进行读取，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。

## 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。

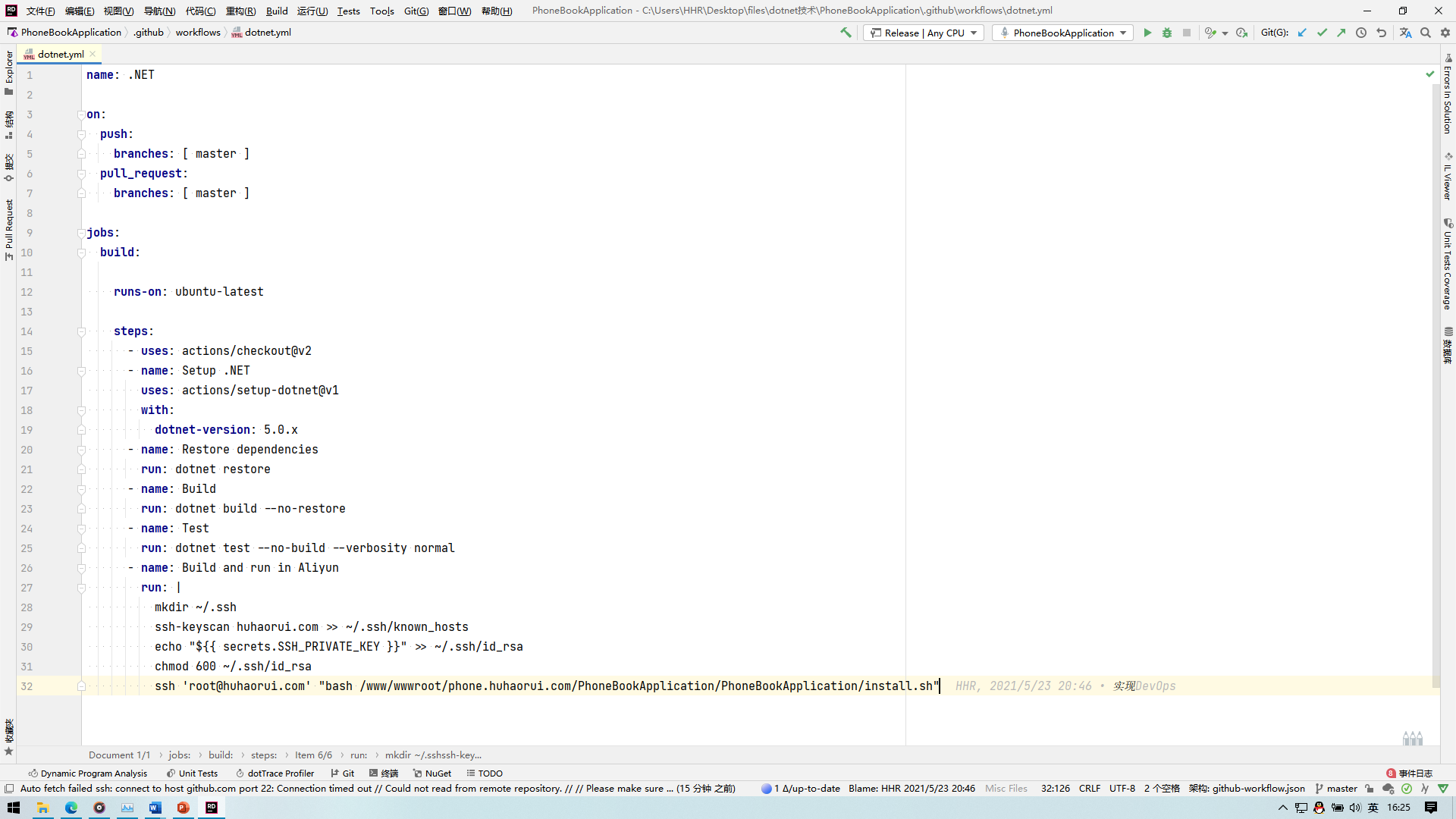
## 基于自定义绘制的View实现侧边栏

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、onMeature、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。

## 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。



# 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。