中山大学移动信息工程学院本科生实验报告

年级	1501	专业（方向）	移动（互联网）
学号	15352015	姓名	曹广杰
电话	13727022190	Email	1553118845@qq.com

Content

中山大学移动信息工程学院本科生实验报告
Content
一、实验题目
二、实现内容
三、课堂实验结果
实验截图
实验步骤以及关键代码
布局与页面设计
ViewHolder
CommonAdapter
调用适配器
使用SQL管理数据库信息
数据库创建
数据库管理
ContentProvider
申请权限
获得指向通信录的指针
遍历通讯录列表
核对信息并返回数据
LayoutInflater实现自定义对话框
实验遇到困难以及解决思路
四、课后实验结果
五、实验思考及感想

一、实验题目

数据库使用

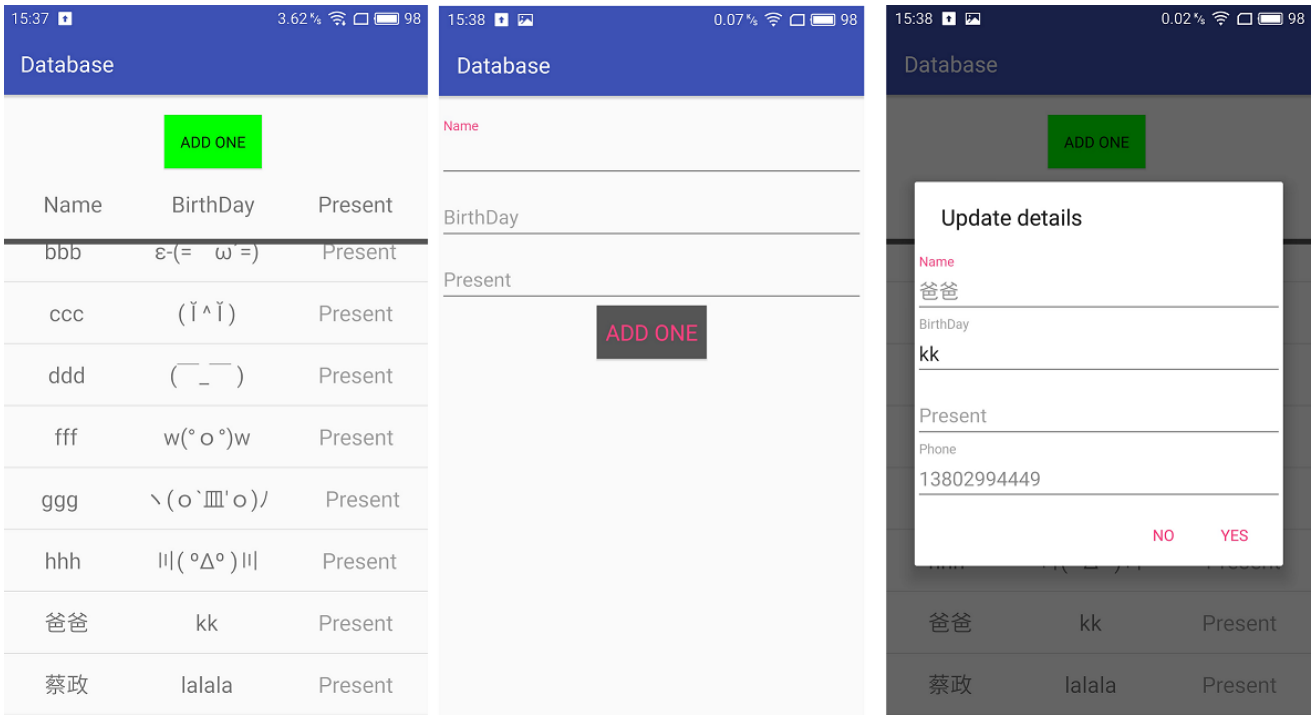
二、实现内容

本次实验模拟实现一个播放器。

- 使用SQL语句实现数据库的使用
- 使用ContentProvider获取通信录电话号码
- 使用AlertDialog实现自定义对话框与点击按钮的使用

三、课堂实验结果

实验截图



实验步骤以及关键代码

布局与页面设计

添加`RecyclerView`温故而知道自己真的没记住

1. 将每一个item信息拓展为view
2. 为item设置适配器
3. 为`RecyclerView`添加点击事件

ViewHolder

`ViewHolder`是`RecyclerView`的一个部分，用于对列表中的每一项进行管理。因为布局中的组件都是原始的格式，尚未拓展为view，故而不能使用UI应的接口信息。因此前提是使用inflate对该组件拓展为view。

在我们平时设置布局文件的时候，通常是使用函数`findById`将布局文件中的组件设置为view信息，类似于以下的实现方式：

```
1  Button bt = (Button)findViewById(R.id.wahaha);
```

此处函数之前的强制转换正是转换的实现过程，而在`ViewHolder`中，需要对这样来自于布局文件中的组件进行管理，在此之前一定要对这种布局信息进行拓展：

```
public static ViewHolder get(Context context,
                             ViewGroup parent, int layoutid){
    View itemview
        = LayoutInflater.from(context).inflate(layoutid, parent, false);
    ViewHolder holder = new ViewHolder(context, itemview, parent);
    return holder;
}
```

所以使用函数`inflate`，根据上下文将原生的组件信息拓展为`ViewHolder`的格式，这种格式也是View的一种，至此已经可以使用view的相关接口，并完成了对于每一个item的设置。而对于一个列表来说，还需要考虑各个item之间的综合布局，而对于整个`RecyclerView`的数据管理则需要使用适配器。

CommonAdapter

适配器是针对某种布局控件的资源管理系统，自定义的适配器可以使得资源管理更加灵活可靠。这里`RecyclerView`不具备完善的资源管理系统，适配器的设置就是为了使之可以灵活管理资源。而本次实验中的适配器的管理，又是针对每一个元组，每一行，因此在适配器实现的时候需要实现几个接口：

1. `onCreateViewHolder`

   用于对每一个正在操作的`RecyclerView`的元组创建适配器管理资源，其内部使用的参数也正是序号、上下文以及父布局之类的参数，这些参数可以非常明显地表明当前操作的对象是列表中的某一个元组，而非整个列表。

2. `onBindViewHolder`

   分析该函数的意义十分简单，观察其传入的参数即可：

   ```
   public void onBindViewHolder(final ViewHolder holder, int position);
   ```

   所谓绑定，就是将原有的大布局中的第`position`个元组与第一个参数连接起来。此时，`ViewHolder`（也是一种view）的click等同于某一个Item的click。

调用适配器

在完成了适配器的实现之后，调用的时候只需要提供3个参数即可：

1. 当前使用适配器的Activity
2. 传入的数据序列
3. 对每一行数据的布局

使用SQL管理数据库信息

由于本次实验中添加的一系列信息都将储存在数据库中，这里就需要数据库的管理技术。

数据库创建

数据库创建需要继承于类`SQLiteOpenHelper`，并在内部设定一些数据库需要的信息。

```
public class Membase extends SQLiteOpenHelper {
    private static final String DB_name = "mem.db";
    private static final int DB_version = 1;
    private static final String TABLE_name = "member";
    private static final String SQL_CREATE_TABLE =
            "create table if not exists "
            + TABLE_name
            + "(id integer, name text primary key,"
            + " birth text, info text);";
}
```

内部数据库的信息可以包括：数据库名字、版本信息、表格名称以及常用的数据库管理命令。而内部实现的接口需要包括：`OnCreate`、结构函数、增删改查的一系列接口。

数据库管理

1. 创建表格

   笔者在创建表格的时候是使用`exec`执行SQL命令，SQL创建表格的命令如下：

   ```
   String SQL_CREATE_TABLE =
               "create table if not exists "
               + TABLE_name
               + "(id integer, name text primary key,"
               + " birth text, info text);";
   ```

   如果该表格名称在数据库中不存在，则创建表格，名称为`TABLE_name`。之后的括号内的语句则表明其内部有多少元素，有什么属性。这些语句与常规的SQL语句相似，在穿点SQL的时候直接使用函数`execSQL`即可：

   ```
       @Override
       public void onCreate(SQLiteDatabase db) {
           db.execSQL(SQL_CREATE_TABLE);// command
       }
   ```

2. 表格插入

   上文中`onCreate`函数的使用在该class中指定创建了一个table，故而此后的函数操作对象都将是这个名为`TABLE_name`的`table`。

   插入函数需要先获取当前需要操作的对象：

   ```
   SQLiteDatabase db = getWritableDatabase();
   ```

   获取对象之后使用结构`ContentValues`包含需要传入的信息，并将容器作为参数传递到Database中：

   ```
               // initialize container
               ContentValues values = new ContentValues();
               // fill the container
                        values.put("id", entity.getID());
               // post container
                long rid = db.insert(TABLE_name, null, values);

               db。close();
   ```

这里需要调用函数接口`insert`实现对数据库的真实操作。

3. 表格修改

表格修改的操作与插入操作原理相似，也是使用`SQLiteDatabase`中的函数接口，将封装好的容器作为参数传入，进一步读取数据：

```
1    public int update(Member entity){
2        SQLiteDatabase db = getWritableDatabase();
3        String whereClause = "name=?";
4        String[] whereArgs = { entity.getName() };
5        ContentValues values = new ContentValues();
6        values.put("id", entity.getID());
7        int rows
8          = db.update(TABLE_name, values, whereClause, whereArgs);
9        db.close();
10       return rows;
11   }
```

这里涉及到的函数update由于是需要针对指定的Tuple进行操作，因此包含查询语句是必须的，对于函数`update`而言：

- 第一个参数是表格的名称

  这里就是需要修改的表格，名为`TABLE_name`。

- 第二个参数是传入的数据

  是第5、6行实现的`ContentValues`的实例，以及对于values的填充操作。

- 第三、四个参数是属性的选择

  限定具体修改哪一个数据时，需要根据属性来限定，在上述代码中，第三个参数的值为：`“name=?”`，则意味着是根据姓名信息对数据库中的内容进行查询，找到姓名为指定信息的数据。而姓名的具体限定，就是第四个参数索要表达的内容。

4. 表格删除

对于表格的删除操作，与之前的插入与更新代码相似，对应的参数代表的意义也非常相似。

```
1    public int deleteByName(String name){
2        SQLiteDatabase db = getWritableDatabase();
3        String whereClause = "name=?";
4        String[] whereArgs = { name };
5        int row = db.delete(TABLE_name, whereClause, whereArgs);
6        db.close();
7        return row;
8    }
```

ContentProvider

使用`ContentProvider`作为通讯录信息的访问代理，保证了用户的信息安全并获得了我们需要的权限，使用流程分为步：

1. 申请权限
2. 获得指向通讯录列表的指针

3. 遍历通讯录列表
4. 核对姓名信息
5. 获取电话号码

申请权限

在访问手机的通信列表的时候，是需要获得用户的许可权限的。这里笔者使用静态的权限申请：

```
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
```

获得指向通信录的指针

之后才可以获得指向通讯录的列表信息：

```
Cursor cursor
    = getContentResolver()
    .query(ContactsContract.Contacts.CONTENT_URI,
                null, null, null, null);
```

query的第一个参数限定了我们访问的内容，这里是指联系人的信息，其他的参数不必要也就无需设置。至此我们已经获得了指向通讯录的指针，也就意味着我们可以访问通讯录了。

遍历通讯录列表

由于通讯录中的数据毕竟不是以数据库的格式储存的，所以在访问和读取的时候都不能直接读取，而是需要遍历每一条记录进行核对的。

遍历时需要使用接口`moveToNext()`：

```
                    while (cursor.moveToNext()){
                        /* Operation */
                    }
```

核对信息并返回数据

核对信息需要同时获得笔者正在查询的姓名信息以及当前的姓名信息。

1. 来自对话框中的姓名

   对话框只有在用户点击`RecyclerView`的时候才会弹出，所以理想的情况下应该是在adapter的监听器中添加查询代码，以确定用户点击的位置以及希望获取的姓名信息：

   ```
   adapter.setOnItemClickListener(new CommonAdapter.OnItemClickListener() {
       @Override
       public void onClick(final int position) {
           // Prepare datas for Dialog
           final Map<String, Object> tmp = listitem.get(position);
           final String namedata = tmp.get("name").toString();
   }
   ```

   确定位置信息`position`之后，为了得知当前点击位置的姓名信息，可以从`listItem`中根据位置查找到对应的映射键值对`tmp`，并通过键值对根据我们需要查询信息的特征进行查询返回。

2. 来自通讯录的姓名

   通过正在遍历通讯录的指针返回当前查询的联系人的姓名信息：

```
    String display_name
        = cursor.getString(cursor.getColumnIndex("display_name"));
```

将两者进行比较，相同则摘取电话号码：

```
if (display_name.equals(Name)){
    // pointer to phone number
    Cursor phone
     = getContentResolver()
      .query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,null,
             ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "=" +
 contactid,null, null);
    // collect digits into number
    while(phone.moveToNext()) {
      Number += phone.getString(
              phone.getColumnIndex(
                  ContactsContract.CommonDataKinds.Phone.NUMBER))+ " ";
    }
    // clear
    phone.close();
}
```

LayoutInflater实现自定义对话框

用户点击`RecyclerView`的时候，会弹出对话框，如果希望使用自定义布局，则需要先实现布局文件`dialog_update.xml`，并在后续的使用过程中引用它：

```
        LayoutInflater factor
            = LayoutInflater.from(MainActivity.this);
        View view_in = factor.inflate(R.layout.dialog_update, null);
```

因为布局文件中的组件都是非常原始的，尚未成为view，所以需要将其拓展为view，与`ViewHolder`的使用方式一样，使用函数`inflate`。此时返回的View是整个布局的view，如果想要获取其中的组件，可以在当前view的基础上进行查询：

```
        final TextInputLayout Namebox
               = (TextInputLayout)view_in.findViewById(R.id.namelog);
```

接下来就可以针对不同的组件进行不同的设置了。

实验遇到困难以及解决思路

1. 无法clean，也无法加载`RecyclerView`的添加项

- 已知信息：

  之前的代码可以正常运作，说明SDK中信息完备。

- 解决方案：

故而查找两个文件之间的不同，得知在文件`build.gradle(Project Database)`代码多了一个Url:

```
allprojects {
  repositories {
    jcenter()
      maven {
      url"http://maven.google.com"
      }
  }
}
```

添补上即可。

2. 打开文件闪退

- 现象：

  运行结果显示是在布局文件运行结束后闪退。

- 异常原因：

  布局文件中出现闪退问题多为组件选择不当

3. `OnItemClickListener`不能被识别

- 现象：

  系统提示添加头部：`AdapterView`，但是`AdapterView`的对应组件不能使用`Item`的`OnClick`

- 解决方案：

  添加`public interface OnItemClickListener`并重载`onClick`，因为查询注册文件与关联文件无果，常常是在本地文件上的代码使用不规范。

4. 布局文件中使用的添加信息无法使用

- 现象：

  不能使用后续添加的颜色以及字符串

- 解决方案：

  剪切掉后添加的资源并clean文件，报错后添加回去。这种情况大多是系统内的文件没有同步，使之报错是为了让其他的部分都完成同步，再添加语句可以保证关联信息的加载完成。

5. 进入`insert`函数内部`getWritableDatabase`闪退

- 分析：

  闪退是由于内存出错，这种情况下大概是没有database可以读取。因此向前查询，找到exec的运行SQL代码，发现错误并修正。

6. insert函数中插入信息的顺序不符合期望

   参数储存是按照整体顺序储存的，序列号有所不同

7. `RecyclerView`中没显示传入的参数
   - 现象：

     没有调用convert函数

   - 原因：

没有绑定`Adapter`和`RecyclerView`（由下向上逐级查找）

8. 运行完`onCreate`闪退

- 现象：

  `RecyclerView`设置adapter导致闪退；

  不是野指针或者空指针的问题；

- 分析：

  那就是组件的信息出了差错。调用的组件信息将`present`误写成了`parent`。

9. `RecyclerView`中每一个`item`的高度过高

- 分析：

  界面的问题有一大部分是因为布局文件设置失误，比如在`LinearLayout`上设置的高度为`match_parent`

10. 删除信息条目的时候总是出错

- 现象：

  删除序号与期望值不同

  更新时对话框中的值，不能与数据保持同步。

- 原因：

  删除顺序问题，先删除了索引列表导致索引错位。

11. 不会修改从布局文件`inflate`的`view`中各组件的数据。

- 解决方法：

  直接截取关键的代码`“inflate(R.layout”`搜索，搜索结果必然有相关用法。

12. 点击事件监听失效

- 现象：

  点击过一次之后，`RecyclerView`的每一个`Item`点击事件监听失效；

- 原因：

  点击事件在`adapter`的监听器中，同时需要更新adapter；

  在adapter中更新adapter会导致新的adapter没有设置监听事件，因此失效。

13. `notifyDatasChanged`不能更新界面

- 现象：

  adapter已经改变，但是activity不能随之改变；

- 解决过程：

  1. 多线程试用效果不好；
  2. `notifyItemChanged`无效；
  3. 直接使用组件调用接口更新：已经更新了adapter，但是没有更新`RecyclerView`；
  4. 搜索问题：“Android RecyclerView自定义适配器更新但是界面没有更新”，得到的第一篇blog从内存的角度分析；

- 得出结论：

  是由于再度创建了adapter，导致原有的adapter没有变化，而新的adapter没有设置点击监听器。

- 具体操作：

  保证原有的`listItem`与`adapter`的地址，避免创建新的组件混淆视听；

  清空与修改适配器内容，视图会随之改变。

- 总结：

  控件的初始化一定要尽可能放在初始化函数中，不可复用。

14. 获取电话号码的时候，无法根据名字条件获取

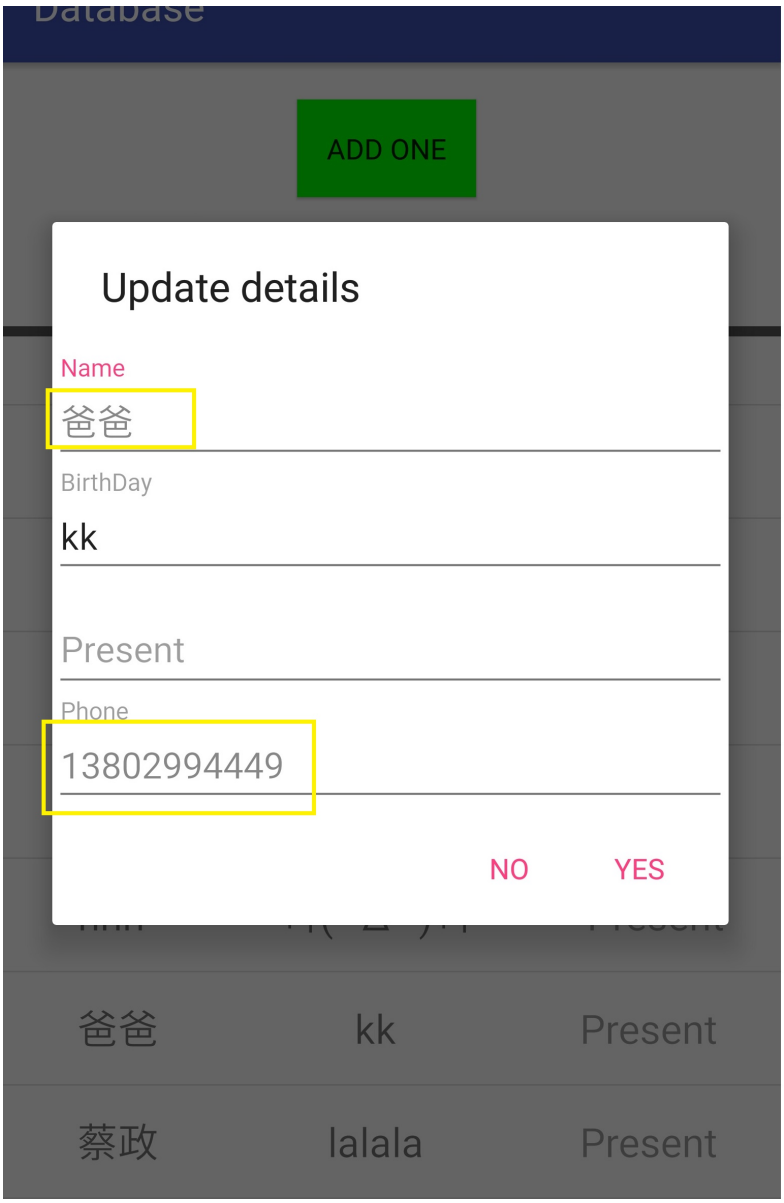
    查看师兄的blog，知晓实现的架构；

四、课后实验结果

使用对EditText的设置，控制用户的修改权限，保证了系统的稳定性。

```
1            Namebox.getEditText().setInputType(InputType.TYPE_NULL);
2            Namebox.getEditText().setTextColor(Color.GRAY);
```

实现效果如下：



标记处不可修改。

五、实验思考及感想

1. 控件的初始化一定要尽可能放在初始化函数中，不可复用。

   有时候笔者初始化的代码进行了封装并实现后续复用，此时就已经埋下了巨大的隐患。因为在后续复用的过程中，可能由于在设置信息的时候，使用的是新创建的组件，新创建的组件代替了原来的组件，就会引起一系列错误。

   组件的声明与初始化最好只使用一次，OOP的架构希望在一个文件中对应一个对象，使用一套数据管理系统。

2. 布局文件的不同。

   从`LinearLayout`到`DrawerLayout`，不同的布局文件可能会有不同的要求，比如在实现对话框的时候，就不可以使用`DrawerLayout`。

   而在一个文件调用另一个布局文件的时候，需要考虑的事情更多，因为：

   1. 不同的布局文件可能会有同样的id命名；
   2. 在使用函数`findViewById`的时候，即便是找不到对应的组件也不会提示；
   3. 一个.java文件通常只在`onCreate`函数中装载一个布局文件；

   如果非要使用当前装载布局文件之外的布局文件，那么请使用`inflate`对布局文件进行拓展，将整个布局文件拓展为View，.java文件在应对View的时候使用起来比较方便。

3. 不是野指针或者空指针的问题，那就是组件的信息出了差错。

   初始化的信息由于是在文件编写的前期实现的，整体架构比较简单，因此也就很少会出错，至少在数据的导入时正确性很高。而组件信息则可能会出错，因为其调用的位置比较广，表现为闪退。此时使用debug工具将其确定在某几行之内，排雷查找就非常人容易了。