实验三

Intent、Bundle 的使用和 ListView 的应用

【实验目的】

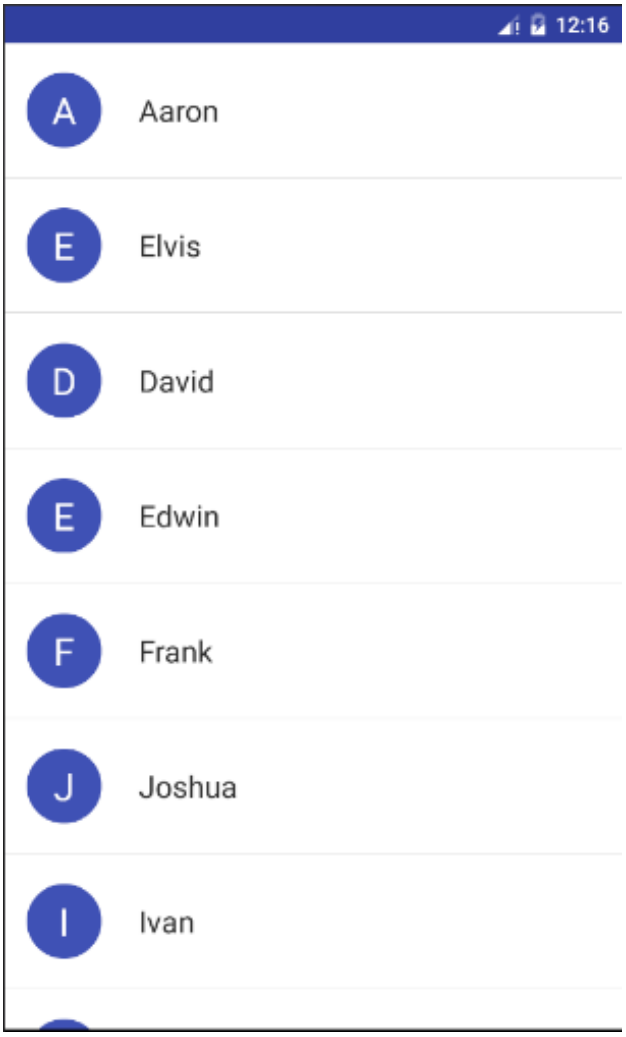
1. 复习事件处理

2. 学习 Intent、Bundle 在 Activity 跳转中的应用

3. 学习 ListView 以及各类适配器的用法

【实验内容】

本次实验模拟实现一个通讯录，有两个界面，第一个界面用于呈现通讯录，如下所示：



下面还有数据，就不截图了，数据在素材中有给出。

点击任意一项后，可以看到详细的信息：

实验要求：

布局方面的要求：

1、通讯录界面

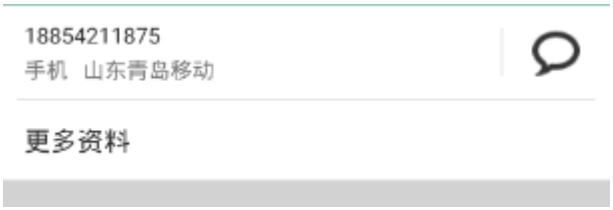
每一项为一个圆圈和一个名字，圆圈与名字均竖直居中。圆圈中为名字的首字母，首字母要处于圆圈的中心，首字母为白色，名字为黑色，圆圈的颜色自定义即可，建议用深色的颜色，否则白色的首字母可能看不清。关于圆圈效果如何实现，还是参照实验一中的文

章：http://blog.csdn.net/sysukehan/article/details/52022307，个人觉得文章中的方法灵活性不够高，如果能用更好方法实现的同学，请在实验报告中写明，有加分。

2、联系人详情界面顶部

顶部的背景色在通讯录数据中已经给出，每个人对应有一种特定的颜色，这块背景色占整个界面的 1/3，返回图标处于这块 View 的左上角，联系人名字处于左下角，星标处于右下角，它们与边距都有一定距离，自己调出合适的距离即可。需要注意的是，返回图标与名字左对齐，名字与星标底边对齐。这一块建议大家去看一下 RelativeLayout 的使用。

3、联系人详情界面中部



使用的黑色 argb 编码值为#D5000000，稍微偏灰色一点的“手机”、“山东青岛移动”的 argb 编码值为#8A000000。注意，电话号码那一栏的下边有一条分割线，argb 编码值为 #1E000000，右边聊天符号的左边也有一条分割线，argb 编码值也是#1E000000，这条分割线要求高度与聊天符号的高度一致，并且竖直居中。字体的大小看着调就可以了。“更多资料”底部的分割线高度自己定，argb 编码值与前面的分割线一致。

4、联系人详情页面底部

这个没什么说的，内容和样式都很清楚。

5、特别提醒，这次的两个界面顶部都没有标题栏，要用某些方法把它们去掉。

逻辑方面的要求：

1、点击通讯录中的某一个联系人会跳转到联系人详情界面，呈现该联系人的详细信息；长按通讯录中的联系人会弹出对话框询问是否删除该联系人，点击确定则删除该联系人，点击取消则对话框消失。

注意对话框中的人名为被长按的联系人。

2、联系人详情界面中点击返回图标会返回上一层，点击星标会切换状态，如果原先是空心星星，则会变成实心星星；如果原先是实心星星，则会变成空心星星。

【基础知识】

1、ListView 的使用

布局上比较简单，在布局文件中写上

```
<ListView
    android:id="@+id/contacts_list"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

即可，这样就创建了一个空的列表，然后在.java 文件中再填充数据，所以 id 是一定要设的。

在.java 文件中获得这个 ListView 之后，使用 Adapter 为这个 ListView 填充数据，常用的 Adapter 有 ArrayAdapter、SimpleAdapter，这两种在下面会详细讲述如何使用。随着 ListView 中内容的丰富，以上两种 Adapter 已经很难满足需要，因此现在一般使用自定义的 Adapter 来填充数据，如何使用自定义的 Adapter 会在拓展知识中讲。

ArrayAdapter

最简单的 Adapter，创建 ArrayAdapter 时需指定如下三个参数：

Context：这个参数无须多说，它代表了访问整个 Android 应用的接口。几乎创建所有组件都需要传入 Context 对象。

textViewResourceId：一个资源 ID，该资源 ID 代表一个 TextView，该 TextView 组件将作为 ArrayAdapter 的列表项组件。

数组或 List：该数组或 List 将负责为多个列表项提供数据。

示例：

```
operations = new String[] {"text1", "text2", "text3"};
ArrayAdapter<String> arrayAdapter
        = new ArrayAdapter<>(this, R.layout.operation_list_item, operations);
operationList.setAdapter(arrayAdapter);
```

在创建完 ArrayAdapter 之后，调用 ListView 的 setAdapter 方法即可将数据填充到 ListView 中。

这里有一点要特别注意的是 textViewResourceId 是一个 layout，在这个 layout 中只能有一个 TextView，其它任何组件都不能有，包括 LinearLayout 等布局组件，否则会报错。

SimpleAdapter

由于 ArrayAdapter 只能显示文字，功能实在有限，如果需要多填充一些内容的话指望不上，这时候可以使用 SimpleAdapter。

SimpleAdapter 相比 ArrayAdapter 强大很多，创建 SimpleAdapter 需要 5 个参数，第一个参数依然是 Context，就不多说了，下面介绍余下的 4 个参数：

第 2 个参数：该参数应该是一个 List<? Extends Map<String, ?>>类型的集合对象，该集合中每个 Map<String, ?>对象生成一个列表项。

第 3 个参数：该参数指定一个界面布局的 ID。该界面布局指定每一个列表项的样式。

第 4 个参数：该参数应该是一个 String[]类型的参数，该参数决定提取 Map<String, ?>对象中哪些 key 对应的 value 来生成列表项。

第 5 个参数：该参数应该是一个 int[]类型的参数，该参数决定填充哪些组件。

示例：

首先构建好数据，这里模拟了一个图书清单，一个 map 中有两个 key，存放书名和价格，然后添加到 list 中。

```
List<Map<String, Object>> data = new ArrayList<>();
String[] booksName = new String[] {"疯狂Android讲义(第2版)", "疯狂Java讲义(第3版)", "设计模式"};
String[] booksPrice = new String[] {"99.00元", "109.00元", "35.00元"};
for (int i = 0; i < 3; i++) {
    Map<String, Object> temp = new LinkedHashMap<>();
    temp.put("name", booksName[i]);
    temp.put("price", booksPrice[i]);
    data.add(temp);
}
```

然后创建 SimpleAdapter：

```
ListView listview = (ListView) findViewById(R.id.listview);
SimpleAdapter simpleAdapter = new SimpleAdapter(this, data, R.layout.item,
        new String[] {"name", "price"}, new int[] {R.id.name, R.id.price});
listview.setAdapter(simpleAdapter);
```

之后还是用 ListView 的 setAdapter 方法添加 Adapter。

看一下 R.layout.item 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/r
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="15dp"
        android:textSize="20sp"
        android:textColor="#000000"/>

    <TextView
        android:id="@+id/price"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:gravity="end"
        android:padding="15dp"
        android:textSize="20sp"
        android:textColor="#000000"/>

</LinearLayout>
```

可以看到，一个 LinearLayout 包含两个 TextView，一个用于显示书名，一个用于显示价格，这个 layout 用于规定 ListView 中每一个列表项的样式。SimpleAdapter 中的第四个参数 String 数组与 map 的两个 key 对应，第五个参数 int 数组与这个 layout 中两个 TextView 的 id 相对应，注意 String[]数组与 int[]数组中的值要一一对应，在这个示例中，key 为 name 的 value 填充到 id 为 name 的 TextView 中。效果如下图所示：

2、ListView 列表项的单击和长按

方法原型如下：

```
contactsList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        //  单击事件处理在这里进行
    }
});

contactsList.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
        //  长按事件处理在这里进行
        return false;
    }
});
```

长按有返回值，在理论课的课件中写的很清楚了，这里就不解释了。注意在两个方法的参数中都有 int i, long l 这两个参数，i 指的是这一项在列表中的位置，l 指的是这一项的 id，在 ArrayAdapter 和 SimpleAdapter 中，i 和 l 是相等的，在另一种 Adapter——CursorAdapter 中，l 指的是从数据库中取出的数据在数据库中的 id 值。

3、ListView 数据更新

直观地想，要实现数据更新，只要更新 List，重新创建一个 SimpleAdapter 就可以了，这样会比较麻烦，SimpleAdapter 有一个 notifyDataSetChanged()方法，当之前创建该 SimpleAdapter 的 List 发生改变时，调用该方法就可以刷新列表了。要特别注意的一点是，List 不能指向新的内存地址，即不能 list = new ArrayList<>();这样是不起作用的，只能调用它的 remove()，add()等方法来改变数据集。

示例：

```
data.remove(0);
simpleAdapter.notifyDataSetChanged();
```

错误写法：

```
data = new ArrayList<>();
simpleAdapter.notifyDataSetChanged();
```

4、去掉标题栏

要去掉标题栏有多种做法，这里举一种方法。

Android Studio 创建项目时默认的 theme 是：

```
android:theme="@style/AppTheme">
```

它的定义是：

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

修改 parent 即可：

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
```

5、星星图标的切换

星星的切换难点在于如何得知星星此时是空心的还是实心的，这个也有多种做法，这里也只介绍一种。

每个 View 都可以设置 tag，通过 tag 可以用来判断该 View 现在的状态。在初始化的时候，将 tag 设置为 0，标记此时为空心星星，如果星星被点击了，并且 tag 为 0，那么就把图片换为实心的星星，然后设置 tag 为 1；如果 tag 为 1，那么就把图片换为空心的星星，然后设置 tag 为 0。建议在 java 文件中给需要的 view 设置 tag。

【拓展知识】

自定义 Adapter

前面介绍的 ArrayAdapter 和 SimpleAdapter 都有一定的局限性，SimpleAdapter 较 ArrayAdapter 要好一些，但还是不够灵活，假如我的某些列表项需要有一些特性，或者我的列表项中的某些控件需要设置监听器，就不够用了。因此，强烈建议大家一开始就习惯

自定义 Adapter 来适配自己的列表，只在某些简单的情况下才使用前面介绍的两种 Adapter。

自定义的 Adapter 需要继承 BaseAdapter：

```
public class MyAdapter extends BaseAdapter {
    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        return null;
    }
}
```

上面列出的四个方法是必须重写的四个方法，下面一一介绍这四个方法：

int getCount();获得数据项列表的长度，也就是一共有多少个数据项。

Object getItem(int i);获得某一个数据项。

long getItemId(int i);获得数据项的位置。

View getView(int i, View view, ViewGroup viewGroup);获得数据项的布局样式，最重要的一个方法。

自定义 Adapter 需要提供给一个数据列表才能填充数据，一般是一个 List 类型，以我们刚刚图书列表的例子为例，我们可以先给列表项创建一个类 Book，然后将 List<Book>传入 Adapter 中作为数据提供的列表：

```
public class Book {

    private String bookName;
    private String bookPrice;

    public Book(String bookName, String bookPrice) {
        this.bookName = bookName;
        this.bookPrice = bookPrice;
    }

    public String getBookName() {
        return bookName;
    }

    public String getBookPrice() {
        return bookPrice;
    }
}
public class MyAdapter extends BaseAdapter {

    private List<Book> list;

    public MyAdapter(List<Book> list) {
        this.list = list;
    }

    @Override
    public int getCount() {
        if (list == null) {
            return 0;
        }
        return list.size();
    }

    @Override
    public Object getItem(int i) {
        if (list == null) {
            return null;
        }
        return list.get(i);
    }

    @Override
    public long getItemId(int i) {
        return i;
    }
```

依照刚刚的想法重写完之后，接下来是最重要的重写 getView()方法，首先解释一下三个参数的含义：

i 指的是当前是在加载第几项的列表项

viewGroup 是列表项 View 的父视图，调整列表项的宽高用的

view 指的是一个列表项的视图，我们需要给 view 一个布局，然后在布局中放置我们需要的内容

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
    view = LayoutInflater.from(context).inflate(R.layout.item, null);
    //  获得布局中显示书名和价格的两个TextView
    TextView bookName = (TextView) view.findViewById(R.id.name);
    TextView bookPrice = (TextView) view.findViewById(R.id.price);
    //  从数据列表中取出对应的对象，然后赋值给它们
    bookName.setText(list.get(i).getBookName());
    bookPrice.setText(list.get(i).getBookPrice());
    //  将这个处理好的view返回
    return view;
}
```

getView()方法的最基本写法：

这种方法没有充分利用 view 的特点，只是每次从屏幕外滚进来一个新的项就再加载一次布局。其实 ListView 每次从屏幕外滚进来一项就会有一项滚出屏幕外，这个时候 view 是有内容的，不过是旧的内容，因此我们只需要改变一下 view 的内容然后返回它就可以了，不需要再去加载一次布局。

getView()方法的改进版写法：

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    //  当view为空时才加载布局，否则，直接修改内容即可
    if (view == null) {
        //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
        view = LayoutInflater.from(context).inflate(R.layout.item, null);
    }
    //  获得布局中显示书名和价格的两个TextView
    TextView bookName = (TextView) view.findViewById(R.id.name);
    TextView bookPrice = (TextView) view.findViewById(R.id.price);
    //  从数据列表中取出对应的对象，然后赋值给它们
    bookName.setText(list.get(i).getBookName());
    bookPrice.setText(list.get(i).getBookPrice());
    //  将这个处理好的view返回
    return view;
}
```

这样写可以减少一些重复的加载布局的操作，提高效率。

但是每次 findViewById()也是一件很麻烦的事情，如果控件一多，也会降低 ListView 的效率。因此，使用 setTag 的方法和新建一个 ViewHolder 类来提高这部分的效率。

getView()方法改进版 2.0：

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {

    //  新声明一个View变量和ViewHolder变量
    View convertView;
    ViewHolder viewHolder;

    //  当view为空时才加载布局，并且创建一个ViewHolder，获得布局中的两个控件
    if (view == null) {
        //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
        convertView = LayoutInflater.from(context).inflate(R.layout.item, null);
        viewHolder = new ViewHolder();
        viewHolder.bookName = (TextView) convertView.findViewById(R.id.name);
        viewHolder.bookPrice = (TextView) convertView.findViewById(R.id.price);
        convertView.setTag(viewHolder);  //  用setTag方法将处理好的viewHolder放入view中
    } else {  //  否则，让convertView等于view，然后从中取出ViewHolder即可
        convertView = view;
        viewHolder = (ViewHolder) convertView.getTag();
    }
    //  从viewHolder中取出对应的对象，然后赋值给它们
    viewHolder.bookName.setText(list.get(i).getBookName());
    viewHolder.bookPrice.setText(list.get(i).getBookPrice());
    //  将这个处理好的view返回
    return convertView;
}

private class ViewHolder {
    public TextView bookName;
    public TextView bookPrice;
}
```

这样写的话 ListView 的效率就比较高了。

贴一下最终版的自定义 Adapter：

```java
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

import java.util.List;

public class MyAdapter extends BaseAdapter {

    private Context context;
    private List<Book> list;

    public MyAdapter(Context context, List<Book> list) {
        this.context = context;
        this.list = list;
    }

    @Override
    public int getCount() {
        if (list == null) {
            return 0;
        }
        return list.size();
    }

    @Override
    public Object getItem(int i) {
        if (list == null) {
            return null;
        }
        return list.get(i);
    }

    @Override
    public long getItemId(int i) {
        return i;
    }

```

```
43      @Override
44      public View getView(int i, View view, ViewGroup viewGroup) {
45
46          //  新声明一个View变量和ViewHolder变量
47          View convertView;
48          ViewHolder viewHolder;
49
50          //  当view为空时才加载布局，并且创建一个ViewHolder，获得布局中的两个控件
51          if (view == null) {
52              //  通过inflate的方法加载布局，context这个参数需要使用这个adapter的Activity传入
53              convertView = LayoutInflater.from(context).inflate(R.layout.item, null);
54              viewHolder = new ViewHolder();
55              viewHolder.bookName = (TextView) convertView.findViewById(R.id.name);
56              viewHolder.bookPrice = (TextView) convertView.findViewById(R.id.price);
57              convertView.setTag(viewHolder);  //  用setTag方法将处理好的viewHolder放入view中
58          } else {  //  否则，让convertView等于view，然后从中取出ViewHolder即可
59              convertView = view;
60              viewHolder = (ViewHolder) convertView.getTag();
61          }
62          //  从viewHolder中取出对应的对象，然后赋值给它们
63          viewHolder.bookName.setText(list.get(i).getBookName());
64          viewHolder.bookPrice.setText(list.get(i).getBookPrice());
65          //  将这个处理好的view返回
66          return convertView;
67      }
68
69      private class ViewHolder {
70          public TextView bookName;
71          public TextView bookPrice;
72      }
73  }
```

【检查内容】

1、有多个联系人的界面中，圆圈是圆的，字母居中，圆圈与名字竖直居中

2、联系人详情界面中，顶部占三分之一的实现方法，返回图标，姓名，星标放置在指定位置，对齐，没有使用硬编码的形式实现（对齐可以使用硬编码的形式）

3、联系人详情界面中，界面中部聊天符号旁边的分割线上下和符号等高并竖直居中

4、多个联系人界面使用 ListView 实现，单个联系人详情界面中底部的四个操作的列表使用 ListView 实现

5、单击后跳转各项资料显示正确，星星点击后行为正常，返回图标功能正常

6、长按后弹出框的显示内容正常，点击确定能正确删除联系人，删除后列表工作正常（点击其它联系人显示信息正确，长按其它联系人删除操作正确）

【提交说明】

1、deadline：下一次实验课前一天晚上 12 点

2、提交 ftp：ftp://222.200.185.18:1890/，作业提交文件夹中对应的文件夹下

3、命名与目录结构要求：

附件命名及格式要求：学号_姓名_labX.zip（姓名中文拼音均可）

重复提交命名格式要求：学号_姓名_labX_Vn.zip

目录结构：

```
14331111_huashen_lab1 --
                      |
                      --  lab1实验报告.pdf
                      |
                      --  lab1_code（包含项目代码文件）
```

其中项目代码文件为项目文件夹，提交之前先 clean