Chapter 1　类型介绍

1.1　Activity 的成员函数

1.1.1 生命周期函数

protected void onCreate(Bundle savedInstanceState);//Bundle 一般为空，不为空的情况见 P67

protedted void onStart();

protected void onResume();

protected void onPause();

protected void onStop();

protected void onDestroy();

1.1.2 活动启动

public void startActivity(Intent intent);

public void startActivityForResult(Intent intent,int requestCode); P52

1.1.3 界面

public final boolean requestWindowFeature(int featureId);

public void setContentView(int layoutResID);

1.1.4 获取意图对象（通过显式启动的活动，由上向下传递 Intent 对象）

public Intent getIntent();P51

1.1.5 Back 键

public void onBackPressed();P54

1.1.6 销毁活动

public boolean isFinishing();

pubilc void finish(){finish(false);}

1.1.7 具有返回值的活动，设置返回值

public final void setResult(int resultCode); P53

1.1.8 获取返回值

protected void onActivityResult(int resquestCode,int resultCode,Intent data);P53

1.1.9 获取当前返回栈的 ID

public int getTaskId();P76

1.1.10　根据 id 查找 View

public View findViewById(int id)//会在当前的布局（或者说成控件）中根据 id 查找，因此在 ListView 中，多个子布局是相同的，子布局中包含的控件 id 也相同，由于只会查找当前 View 中的对应 id 的控件，因此不会产生问题

1.2　Intent

1.2.1 构造函数

public Intent(Context packageContext,class<?> cls);

其中 Context 是一个抽象类：Activity、Service、Application 都是其的子类

1.2.2 传递数据，

写入数据时的函数是重载函数，在读取数据时需要根据数据类型来选择读取函数

public Intent putExtra(String name,T t); **//name 是键，可自定义**
T getTExtra(String name);**//获取传输的数据**

1.3 View

1.3.1 控件可视化控制函数

public void setVisibility(int v);
v 有三种可选值：View.VISIBLE,View.INVISIBLE,View.GONE
public int getVisibility();

1.3.2 View.progressBar

public synchronized void setProgress(int pregress);
public synchronized int getProgress();

Chapter 2　探究活动

2.1　活动启动模式（在 AndroidManifest.xml 中设置）

1、在 activity 头标签内部设置：android:launchMode="singleTop"

a、standard（默认）

b、singTop 当活动在返回栈栈顶时，不会再创建该活动的新实例

c、singTask 当活动在返回栈中，不会再创建该活动的新实例

d、singleInstance 在该模式下会有一个单独的返回栈来管理这个活动，无论哪个应用程序来访问这个活动，都共用一个返回栈，解决了共享的问题（每个应用程序都会有自己的返回栈，同一个活动在不同的返回栈中入栈时，必然是创建了新的实例）

2.2　Button

1、获取：Button button1=(Button) findViewById(R.id_1)

2、使用：button1.setOnClickListener(new View.OnClickListener(){

```
        @Override
        public onClick(View v){

        }
    });
```

3、也可以不用匿名内部类，参考 P96

2.3　活动启动的最佳写法

```
public static void actionStart(Context context,String data1,String data2){
    Intent intent=new Intent(context,AcitvityName.class)
    intent.putExtra("param1",data1);//表示传入的参数
    intent.putExtra("param2",data2);
    context.startActivity(intent);
}
```

使用：　在另一个活动的.java 中

```
ActivityName.actionStart(AnotherActivityName.this,data1,data2);
```

2.4　返回数据给上一个活动

活动 B 返回数据给活动 A

活动 B 中：

Activity.setResult(int resultCode,Intent data)

活动 A 中：

Activity.startActivityForResult(Intent intent,int requestCode)

然后覆 onActivityResult 方法

```
protected void onActivityResult(int requestCode,int resultCode,Intent data)
    switch(requestCode){
        case 1:
            if(resultCode==RESULT_OK){...}
        ...
    }
}
```

Chapter 3 UI (User Ininterface) 开发

3.1 尽量用 XML 代码来设计界面

3.2 常见控件的使用方法

3.2.1 TextView

1、属性介绍

android:id *给当前控件定义一个唯一标识符

android:layout_width *指定控件的宽度

android:laytou_height *指定控件的高度

所有控件都有以上两种属性，可选值有

match_parent：表示让当前控件的大小和父布局的大小一样，也就是由父布局来决定当前控件的大小

fill_parent

wrap_content：表示当前控件的大小能够刚好包含住里面的内容，也就是由控件内容决定当前控件的大小

match_parent 和 fill_parent 相同，但推荐使用 match_parent

也可以设定固定大小：会在不同手机屏幕的适配方面出现问题

android:gravity *指定文字对齐方式

可选值有：top,bottom,left,right,center,center_vertical,center_horizontal 等

可以用|来同时指定多个值

android:textSize="24sp" *指定文字大小

android:textColor="#00ff00" *指定文字颜色

3.2.2 Button

1、其属性和 TextView 类似

b、在.java 文件中可以为 Button 的点击事件注册一个监听器

```
button.setOnClickListener(new OnClickListener());
```

3.2.3 EditText

1、EditText 是程序用于和用户进行交互的一个重要控件，它允许用户在控件里输入和编辑内容，并可以在程序中对这些内容进行处理。

2、属性介绍

android:hint="Type something here" *显示一些提示性的文字

android:maxLines="2" *限制最多的行数,超过时，EditText 不会继续拉伸，但文本会向上滚动（由于随着输入的内容增多，EditText 会被不断拉长（高度指定 wrap_content 时））

android:inputType="textPassword" *输入类型

3.2.4 ImageView

1、将图片直接复制到 res/drawable 目录下，并起名字 name，该名字可以通过 R.drawable.name 进行引用。

2、属性：

android:src="drawable/imagename"　　*给 ImageView 指定一张图片

3、public void setImageResoure(@DrawableRes int resId);

3.2.5 ProgressBar

1、用于在界面上显示一个进度条，表示程序正在加载一些数据

2、属性

android:visibility　　*可见属性

可选值有：

visible：表示可见的

invisible：表示不可见，但仍占据原来的大小和位置

gone：表示不仅不可见，而且不再占用任何屏幕空间

style="?android:attr/progressBarStyleHorizontal"　*将进度条设成水平的

android:max="100"　　*进度条最大值为 100

3.2.6 AlertDialog 详见 P210

1、可以在当前页面弹出一个对话框，置于所有界面元素之上，能够屏蔽调其他控件的交互能力，一般都是用于提示一些非常重要的内容或警告信息。

2、一般不写入 XML,而在.java 文件中用以下语句来创建实例

AlertDialog.Builder dialog=new AlertDialog.Builder(MainActivity.this);

(Builder 是 AlertDialog 的一个静态方法，工厂模式)

3、例子代码 P101

3.2.7 ProgressDialog

1、与 AlertDialog 类似，可以在界面上弹出一个对话框，能够屏蔽掉其他控件的交互能力。不同的是 ProgressDialog 会在对话框中显示一个进度条（一般用于表示当前操作比较耗时，让用户耐心等待）

2、一般也不写入 XML 作为布局，而在.java 文件中用以下语句来创建实例

ProgressDialog progressDialog=new ProgressDialog(MainActivity.this);

3.2.8 CheckBox

1、复选框

3.2.9 ScrollView

1、**由于手机屏幕比较小，有时候过多内容一屏显示不了，该控件允许我们以滚动的形式查看屏幕外的那部分内容**

3.3　基本布局

布局是一种可以防止很多控件的容器，可以按照一定的规律调整内部控件的位置，从而编写出精美的界面

3.3.1 LinearLayout

1、LinearLayout 又称为线性布局，这个布局会将它所包含的控件在线性方向上

依次排列
2、属性
android:orientation *指定了排列的方向
可选值有：
vertical：垂直
horizontal：水平

android:layout_gravity *用于指定控件在布局中的对齐方式
对比 android:gravity(指定文字在控件中的对齐方式）

android:layout_weight *允许用比例来指定控件大小
系统会吧 LinearLayout 下所有控件指定的 layout_weight 值相加，得到一个总值，然后每个控件所占大小的比例就是用该控件的 layout_weight 值除以刚才算出的总值。（如果有部分控件
指定 weight，那么按权值分配的空间就是除了这些控件所占的空间外的剩余的空间）

3.3.2 RelativeLayout

1、RelativeLayout 又称为相对布局，可以通过相对定位的方式让空间出现在布局的任何位置
2、属性

相对于父布局的定位

android:layout_alignParentLeft="true" *与父布局的左侧对齐
android:layout_alignParentRight="true" *与父布局的右侧对齐
android:layout_alignParentTop="true" *与父布局的顶侧对齐
android:layout_alignParentBottom="true" *与父布局的底侧对齐
android:layout_centerInParent="true" *与父布局的中心对齐

相对于控件的定位

android:layout_above="@id/control_id" *位于指定控件上方
android:layout_below="@id/ control_id " *位于指定控件下方
android:layout_toLeftOf="@id/ control_id " *位于指定控件左侧
android:layout_toRightOf="@id/ control_id " *位于指定控件右侧
被指定的控件的定义必须出现在指定控件的前面

相对于控件的边缘对齐

android:layout_alignLeft="@id/control_id" *与指定控件左边缘对齐
android:layout_alignRight="@id/ control_id " *与指定控件右边缘对齐
android:layout_alignTop="@id/ control_id " *与指定控件上边缘对齐
android:layout_alignBottom="@id/ control_id " *与指定控件下边缘对齐

3.3.3 FreameLayout

1、这种布局没有任何定位方式，所有的控件都会摆放在布局的左上角
2、在介绍碎片的时候会用到它

3.3.4 TableLayout

1、TableLayout 允许我们使用表格的方式来排列控件。
2、在 TableLayout 中每加入一个 TableRow 就表示在表格中添加了一行，然后在 TableRow 中每加入一个控件，就表示在该行中添加了一列
3、**TableLayout 中的控件是不能指定宽度的**
4、属性

android:layout_span="2"　　*让该控件占据两列的空间
android:stretchColumns="i"　　*自动适应屏幕宽度

i：表示如果表格不能完全占满屏幕宽度，就将第 i+1 列进行拉伸

3.4　自定义控件

类型继承体系 P121

3.4.1 引入布局

引入布局可以解决重复编写布局代码的问题。但如果该布局中有相应的控件所触发的事件，那么该事件还是需要进行重写，因此此时应该转为自定义控件，既能够引入布局，又能够避免控件的重写

```
<include layout="@layout/layout_name">
```

3.4.2 自定义控件

例如每一个活动都会需要一个返回按钮的点击事件
首先新建一个 java 文件，创建一个自定义的布局，继承自（LinearLayout 等等）重载带有两个参数的构造函数（在布局中引入该自定义布局控件就会调用这个构造函数）

```
public TitleLayout(Context context, AttributeSet attrs){
    super(context,attrs);
    LayoutInflater.from(context).inflate(R.layout.title,this);
    //...控件出发事件
}
```

LayoutInflater.from(context)可以构造出一个 LayoutInflater 对象，然后调用 inflate()方法可以动态加载一个布局文件

public View inflate(int resource,ViewGroup root)

然后，在 main.xml 中写入以下语句添加自定义控件，必须写出完整的包名

```
<complete_package_name.Layout_name>
</complete_package_name.Layout_name>
```

3.5　ListView 控件 *P129-P134*

ListView 用于展示大量的数据，数据可以来源于互联网，数据库，或者自定义等等。**这些数据是不能直接传递给 ListView 的，需要用适配器，推荐用 ArrayAdapter<T>，可以指定泛型参数，并有多个重载版本的构造器。**

然后

ListView listView=(ListView) findViewById(...)

listView.setAdapter(adapter);

listView.setSelection(int position);定位到指定位置

android:divider="#0000"　　*用于将分割线设置成透明

3.5.1 ArrayAdapter<T>

public ArrayAdapter(Context context, int resource,List<T> objects)

其中 resource 指的是 res 下的布局 id，用于 ListView 的子项布局

public View getView (int position,View convertView,ViewGroup parent)

这个方法在子项被滚动到屏幕内的时候会被调用

convertView 用于缓存之前被加载好的布局，便于提高效率

public void notifyDataSetChanged();//当数据链表变化时会调用这个函数

3.6　单位和尺寸

px：像素

pt：磅数

dp：像素密度（每英寸所包含的像素数）

sp：可伸缩像素（与 dp 的设计理念相同）

160dpi 的屏幕下：1dp=1px

为了适应不同的分辨率，一律用 dp 和 sp 来指定宽度

聊天界面的编写：

.xml:

main: 1、ListView　2、{EditText+Button}LinearLayout_horizon

list_item: 1、left_TextView　2、right_TextView。

一个 item 会同时包含这两项，但是根据消息的类型选择显示其中一个

.java

```
Message
MessageAdapter extends ArrayAdapter{
    int resourceId;//记录 item 的 id 号
    public MessageAdapter(Context context,int textViewResourceId,
                          List<Message> objects){...}
    @Override
    public View getView(int position,View convertView,ViewGroup parent){...}
}
```

Chapter 4　碎片

4.1　定义

可以将碎片理解成一个迷你型的活动。

4.2　碎片的简单用法

1、属性

android:name　　*显示指明要添加碎片的类名，要加上包名

碎片布局与碎片建立关联

2、**动态添加碎片步骤：**

1) **创建待添加的碎片实例**
2) **获取到 FragmentManager，在活动中可以直接调用 getFragmentManager()**
3) **开启一个事物，通过调用 beginTransaction()**
4) **向容器内加入碎片，一般用 replace()方法，需要传入容器 id 和待添加的碎片实例**
5) **提交事物，调用 commit()方法**

3、碎片和活动之间进行通信

findFragmentById 方法可以在活动中得到相应碎片的实例

getActivity 方法可以得到和当前碎片相关联的活动的实例

4、碎片和碎片之间的通信，首先建立碎片 1-活动的通信，再建立活动-碎片 2 的通信

4.3　碎片的生命周期

1、**运行状态：**当一个碎片课件，并且它所关联的活动处于运行状态，那么该碎片也处于运行状态

2、**暂停状态：**当一个活动进入暂停状态时（由于另一个未占满屏幕的活动被添加到了栈顶），与它相关联的可见碎片就会进入到暂停状态

3、**停止状态：**当一个活动进入停止状态时，与它相关联的碎片就会进入到停止状态。或者通过调用 FragmentTranscation 的 remove()，replace()方法也会进入到停止状态。

4、**销毁状态：**由于碎片总是依附于活动的，因此当活动被销毁时，与它相关联的碎片就会进入到销毁状态。或者调用 FragmentTransaction 的 remove()，replace()方法将碎片从活动中移除，但在事物提交之前并没有调用 addToBackStack()方法()

5、

onAttach()：当碎片和活动建立关联时调用

onCreateView()：为碎片创建视图（加载布局）时调用

onActivityCreated()：确保与碎片相关联的活动一定已经创建完毕时调用

onDestroyView()：当与碎片关联的视图被移除时调用

onDetach()：当碎片和活动接触关联的时候调用

6、碎片生命周期图：P167

4.4 动态加载布局

使用限定符

若想要大屏的设备加载另一种布局，在 res 目录下新建 layout-large 文件夹，那么该设备就会自动加载该文件夹下的布局文件

屏幕特征	限定符	描述
大小	small	提供给小屏幕设备的资源
	normal	提供给中等屏幕设备的资源
	large	提供给大屏幕设备的资源
	xlarge	提供给超大屏幕设备的资源
分辨率	ldpi	提供给低分辨率设备的资源(120dpi 以下)
	mdpi	提供给中等分辨率设备的资源(120dpi-160dpi)
	hdpi	提供给高分辨率设备的资源(160dpi-240dpi)
	xhdpi	提供给超高分辨率设备的资源(240dpi-320dpi)
方向	land	提供给横屏设备的资源
	port	提供给竖屏设备的资源

4.5 新闻例子

1、

对于大屏设备：主活动布局上放置两个碎片标签，并与对应的碎片类建立关联

对于手机设备：主活动布局上防止一个标题碎片标签，并与对应的碎片类建立关联，另外新建一个活动，用于放置内容碎片标签，并与对应的碎片类建立关联

2、

新建标题碎片布局以及内容碎片布局，这些布局会在对应的碎片类中的 onCreateView 中被动态加载。

3、

由于标题是一个 ListView，新建列表子布局，该布局会在适配器中被动态加载

4、

新建列表适配器类，新建适配器的实例需要传入列表子布局的 id，以及列表

4.6 碎片总结

1、<fragment>标签，用 name 属性指定.java 文件就代表这个碎片的实例是这个类型的实例，**每一个 xml 文件中的标签都对应着一个控件实例实例（当然，布局不算）**

2、<fragment>标签中的 id 属性，可以利用

getFragmentManager().findFragmentById()来获取该碎片的实例（只要在 xml 文件中的控件，都是一个实例），（View.findViewById()是通过 id 获取一个对应放置在 xml 文件中的 View 的实例）

3、碎片类中

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState){
    view=inflater.inflate(R.layout.?,container,false);
    return view;
 }
```

注意，加载的是一个布局

4、ListView

4.1 适配器

```
private int resourceId;//代表的是一个项目的布局文件！！！，不是 id
public NewsAdapter(Context context,int textViewResourceId,List<T> objects){
    super(context,textViewResourceId,objects);
    resourceId=textViewResourceId;
}
@Override
public View getView(int position,View convertView,ViewGroup parent){
    T t=getItem(position);
    View view;
    if(convertView==null){
        view= LayoutInflater.from(getContext()).inflate(resourceId,null);
    }
    else{
        view=convertView;
    }
    //dosomething，若这个 resourceId 含 TextView，设置其值，等等
    return view;
}
```

4.2 流程

获取列表的实例，传入适配器

4.3 监听

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
                                                int position, long id) {
        T t=tList.get(position);
        //do something
    }
```

```
});
```

Chapter 5　广播

5.1　广播机制简介

标准广播：完全异步执行的广播，所有接收器都会同时接收到这条广播信息，没有先后顺序，无法被截断

有序广播：同步执行的广播，同一时刻只有一个接收器能接收到这条广播信息，广播接收器有先后顺序，可以被截断

5.2　接收系统广播

1、可以在代码中动态注册，也可以在 AndroidManifest.xml 中静态注册

2、创建广播接收器只需要新建一个类，继承自 BroadcastReceiver，重写 onReceive 方法即可。

3、动态注册方法：

```
IntentFilter intentFilter=new IntentFilter();//创建 IntentFilter 实例 intentFilter
intentFilter.addAction("...");//添加的 action 指明想要监听的广播类型
TReceiver tReceiver=new TReceiver();//创建一个广播接收器实例
registerReceiver(tReceiver,intentFilter);//调用 registerReceiver()
```

在 onDestroy()中进行取消注册

4、动态注册可以自由控制注册与注销，但是必须在程序启动后才能接收到广播，因为注册的逻辑是写在 onCreate 方法中的

5、在<application>标签内新建标签<receiver>进行注册

6、注意点：**不要在 onReceive(Context context,Intent intent)方法中添加过多的逻辑或者耗时的操作，广播接收器中是不允许开启线程的，广播接收器更多扮演的是一种打开程序和其他组件的角色**

5.3　发送自定义广播

步骤：

1、新建自定义接收器

2、在 AndroidManifest.xml 中对其进行静态注册，并指明自定义类型的 action

```
<receiver android:name=".AnotherBroadcastReceiver">
    <intent-filter android:priority="100">//设置优先级
        <action android:name="com.example.broadcasttest.MY_BROADCAST" />
    </intent-filter>
</receiver>
```

3、发送广播

```
Intent intent =new Intent("com.example.broadcasttest.MY_BROADCAST");
sendBroadcast(intent);//异步广播,不可截断
sendOrderBroadcast(intent,null);//同步广播，可以用 abortBroadcast();进行截断
```

5.4 使用本地广播

1、 系统全局广播会造成一个问题：携带关键性数据的广播可能被其他应用截获，或者其他程序不停地向我们的广播接收器里发送各种垃圾广播

2、本地广播使用了 LocalBroadcastManager 来对广播进行管理，并提供了发送广播和注册广播接收器的方法 P203

```
private IntentFilter intentFilter;
private LocalBroadcastManager localBroadcastManager;
```

onCreate:

```
localBroadcastManager=LocalBroadcastManager.getInstance(this);
intentFilter =new IntentFilter();//创建 IntentFilter 实例
 intentFilter.addAction("com.example.broadcasttest.LOCAK_BROADCAST");//指明监听广播的类型
localReceiver=new LocalReceiver();//LocalReceiver 是自己定义的接收器类
localBroadcastManager.registerReceiver(localReceiver,intentFilter);//动态注册
```

onDestroy:

```
localBroadcastManager.unregisterReceiver(localReceiver);//动态注销
```

3、本地广播是不能通过静态注册的方式来接受的，因为本地广播只能被自己接收到，而发送本地广播时，程序已经启动了，因此不必要使用静态注册功能

4、本地广播的优势

1) 不需要担心机密数据泄漏问题（广播不会离开程序）
2) 不需要担心有安全漏洞的隐患（其他程序无法将广播发送到程序内部）
3) 本地广播比系统全局广播更加高效

Chapter 6　持久化技术

6.1　简介

1、安卓系统中提供了三种方式用于简单地实现数据持久化功能，文件存储、SharedPreference 存储以及数据库存储

6.2　文件存储

1、文件存储是 Android 中最基本的一种数据存储方式，它不对存储的内容进行任何的格式化处理，所有数据都是原封不动地保存到文件当中，因而它比较适合用于存储一些简单的文本数据或二进制数据。

2、写数据：套两个 try，close 放在 finally 中，finally 里还有一个 try

```
try{
    FileOutputStream out=openFileOutput(<filename>, Context.MODE_PRIVATE);
// MODE_PRIVATE：若存在，写入内容会覆盖原内容，不存在就创建新文件
//MODE_APPEND：若该文件已存在，往里面追加内容，若不存在就创建新文件
//存放位置为/data/data/<packagename>/files/<filename>，因此不必指定路径
    BufferedWriter writer=new BufferedWriter(new OutputStreamWriter(out));
//转为面向字符的输出流，再包装成缓冲类型
    writer.write(inputText);
}catch(IOException e) {
    e.printStackTrace();
}finally{//注意写法与读数据的区别
    try{
        if(writer!=null) writer.close();
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

3、读数据：套两个 try，close 放在 finally 中，finally 里还有一个 try

```
try{
    FileInputStream in=openFileInput("data");
    BufferedReader reader=new BufferedReader(new InputStreamReader(in));
    String line="";
    while((line=reader.readLine())!=null){
        content.append(line);
    }
}catch(IOException e){
    e.printStackTrace();
}finally{//注意写法与写数据的区别
    if(reader!=null)
        try {
            reader.close();
        }catch(IOException e){
            e.printStackTrace();
        }
}
```

TextUtils.isEmpty(s) 等价于 s==null||s.length()==0

6.3 SharedPreferences 存储

1、不同于文件的存储方式，SharedPreference 采用键值对的方式来存储数据

2、写入数据

Android 提供了三种方法用于得到 SharedPreferences 对象

1) **Context.getSharedPreference(String filename,int MODE);**

第一个参数指定 SharedPreference 文件的名称，如果指定文件不存在会创建一个，存放在/data/data/<packagename>/shared_prefs

第二个参数用于指定操作模式，主要有两种模式可以选择： MODE_PRIVATE(值为 0，表示只有当前的应用程序才可以对这个 SharedPreverence 文件进行读写)和 MODE_MULTI_PROCESS(用于多个进程中对同一个 SharedPreference 进行读写。

2) **Activity.getPreferences(int MODE)**

与 Context.getSharedPreference(String filename,int MODE)方法类似，但只接受一个操作模式的参数，会自动将当前活动的类名作为 SharedPreferences 的文件名

3) ***PreferenceManager*.getDefaultSharedPreferences(Context context) 静态方法**

只接受一个 Context 参数，使用当前应用程序的包名作为前缀来命名 SharedPreferences 文件。

操作步骤：

1) 调用 SharedPreferences.edit()方法获取一个 SharedPreferences.Editor 对象
2) 向 SharedPreferences.Editor 对象中添加数据，putBoolean,putInt,putString...
3) 调用 Editor.commit()方法将添加的数据提交，从而完成数据存储操作

3、读取数据

利用 SharedPreferences 对象中对应每个类型的方法来提取数据

SharedPreferences.getBoolean("<Key>",<default>);//第二个参数提供默认值，当不存在对应键的时候，返回默认值

6.4 SQLite

1、Android 内置了数据库

2、提供了 SQLiteOpenHelper 帮助类（抽象类），包含两个抽象方法，分别是：onCreate()和 onUpgrade()

3、**SQLiteOpenHelper 中提供两个非常重要的实例方法，getReadableDatabase()和 getWritableDatabase()。这两个方法都可以打开一个现有数据库（存在打开，不存在新建然后打开），并返回一个可对数据库进行读写操作的对象。当数据库不可写入时（磁盘空间已满）getReadableDatabase()方法返回的对象以只读的方式打开数据库，而 getWritableDatabase()方法则将抛出异常。**

4、SQLiteOpenHelper 有两个构造函数，一般用四个参数的构造函数

1) Context：必须有它才能对数据库进行操作
2) 数据库名：创建数据库使用的就是这里指定的名字
3) Cursor：允许我们在查询数据的时候返回一个自定义的 Cursor(一般 null)

4) 数据库版本号

5、数据库文件存放位置:/data/data/<package name>/database

6、数据库基础知识

基本类型

1) integer：整形
2) real：浮点型
3) text：文本类型
4) blob：二进制类型

关键字：

1) primary key：将该变量设为主键
2) autoincrement：自增长

7 、 **用 SQLiteOpenHelper 对 象 调 用 getReadableDatabase() 或 getWritableDatabase()方法打开数据库**（**该方法会检测数据库是否含有该（对应 SQLiteOpenHelper 构造器的第二个参数）数据库，如果没有，则调用 onCreate 创建数据库，数据库创建成功后，onCreate 将不再被调用，这里指的是不再在用 getReadableDatabase()或 getWritableDatabase()打开数据库时自动调用，但可以手动调用**）

8、添加数据 CRUD：

- C：Create，SQL 语句为 insert
- R：Retrieve，SQL 语句为 select
- U：Updata，SQL 语句为 update
- D：Delete，SQL 语句为 delete

8.1 添加

- SQLiteDatabase.insert(String table,
  String nullColumnHack,ContentValues values)

第一个参数：表

第二个参数：用于未指定添加数据的情况下给某些可为空的列自动赋值 NULL，一般用不到这个功能，直接传入 null

第三个参数：ContentValues 对象，提供了 put 方法重载，用于向 ContentValues 中添加数据，只需要将表中每个列
以及相应待添加数据传入即可

ContentValues.put(String <ColumnName>,T value);

8.2 更新

- SQLiteDatabase.update(String table, ContentValues values,
  String whereClause, String[] whereArgs)

第一个参数：表名

第二个参数：ContentValues 对象，把更新数据在这里组装进去

第三、四个参数：约束更新某几行，不指定就是默认所有行

db.update("Book",values,"name=?",new String[]{"The Da Vinci Code"});

其中'?'为占位符，用第四个参数代替其位置作为含义，这个语句就是删除名字为" The Da Vinci Code"的书

问题：P252 的例子中，ContentValues 对象只添加了价格属性，那么用这个对象进行更新，名字为什么没有变。

8.3 删除

- SQLiteDatabase.delete(String table, String whereClause, String[] whereArgs)

第一个参数：表名
第二、三个参数：约束删除哪几行，不指定的话默认删除所有行

8.4 查询

- public Cursor query(String table, String[] columns, String selection,
  String[] selectionArgs, String groupBy, String having,
  String orderBy)

6.5 最佳实践

1、**SQLite 数据库是支持事务的，事务的特性可以保证一系列操作要么全部完成，要么一个都不会完成。**（比如转账，扣钱，转钱，必须要么全部完成，要么一个不完成，否则钱就会凭空消失了）

2、基本步骤

```
SQLiteDatabase db=dbHelper.getWritableDatabase();
db.beginTransaction();//开启事务
try{
    //do something first
    //do something second
    //...
     db.setTransactionSuccessful();//表示事务正确执行,若在此前有异常抛出，那么该语句不会执行，事务就被认为没有完成，会将各个时间还原到初始状态（在抛出异常之前‘成功执行’的语句也会失效）
}catch(Exception e){
    e.printStackTrace();
}finally{
    db.endTransaction();//无论事务是否完成执行，都要关闭事务
}
```

9、Cursor

```
Cursor curosr;
String name=cursor.getString(cursor.getColumnIndex("name"));
int pages=cursor.getInt(cursor.getColumnIndex("pages"));
...
```

Chapter 7　跨程序共享数据，探究内容提供器

7.1　简介

1、不同于文件存储和 SharedPreferences 存储中的两种全局可读可写操作模式。**内容提供器可以选择只对哪以部分数据进行共享，从而保证我们程序中的隐私部分不会有泄漏的风险**

2、内容提供器的用法一般有两种：**一种是使用现有的内容提供器来读取和操作相应程序中的数据**，**另一种是创建自己的内容提供器给我们程序的数据提供外部访问接口。**

7.2　访问其他程序中的数据

1、当一个应用程序通过内容提供器对其数据提供了外部访问接口，任何其他的应用程序就都可以对这部分数据进行访问。

7.2.1 ContentResolver 的基本用法

1、可以通过 Context.getContentResolver()方法获得 ContentResolver 的实例

2、Context 提供了一系列方法用于对数据进行 CRUD 操作，**其中，insert()方法用于添加数据，update()方法用于更新数据，delete()方法用于删除数据**

query()方法用于查询数据，与 SQLiteDatabase 的方法在参数上有一些区别

3、不同于 SQLiteDatabase，ContentResolver 不接受表名参数，而是使用一个 Uri 参数代替，这个参数被称为内容 URI。

4、内容 URI 给内容提供器中的数据建立了唯一标识符，它主要由两部分组成，权限，和路径。

权限：用于对不同的应用程序做区分，一般为了避免冲突，都会采用程序包名的方式来命名。

路径：用于对同一应用程序中不同的表作区分

例如：content://com.example.app.provider/table1

content://com.example.app.provider/table/1//1 额外的 1 表示 id

5、得到了内容 URI 字符串后，还需将它解析成 Uri 对象才能作为参数输入

```
Uri uri=Uri.parse("content://com.example.app.provider/table1")
```

6、查询语句：

```
Cursor cursor=getContentResolver().query(
    uri,//指定查询某个应用程序下的某一张表
    projection,//指定查询的列名
    selection,//指定 where 的约束条件
    selectionArgs,//为 where 重的占位符提供具体的值
    sortOrder);//指定查询结果的排列方式
```

7.2.2 创建内容提供器

1、通过继承 ContentProvider 的方式来创建一个自己的内容提供器。

2、ContentProvider 类中有六个抽象方法，必须全部重写

3、这六个抽象方法分别是：

- public boolean onCreate()
- public Cursor query(Uri uri,String[] projection,String selection,
  String[] selectionArgs,String sortOrder)

- public Uri insert(Uri uri,ContentValues values)
- public int update(Uri uri,ContentValues values,String selection,
  String[] selectionArgs)
- public int delete(Uri uri,String selection,String[] selectionArgs)
- public String getType(Uri uri)

4、方法解释：

- onCreate:**初始化内容提供器的时候调用，通常会在这里完成对数据库的创建和升级等操作，返回 true 表示内容提供器初始化成功，false 表示失败。仅当 ContentResolver 尝试访问程序中的数据时，内容提供器才会被初始化**
- query:从内容提供器中查询数据，用 uri 来确定查询哪张表，projection 参数用于确定哪些列，selection 和 selectionArgs 用于约束查询哪些行，sortOrder 用于对结果进行排序，查询结果存放在 Cursor 对象中返回
- insert:向内容提供器中添加数据，uri 表示添加的表，待添加数据放在 values 中，添加完成后，返回一个 Cursor 对象表示这条新纪录的 URI
- update:更新内容提供器中已有的数据，uri、selection、selectionArg 同上，受影响的行数将作为返回值返回
- delete:从内容提供器中删除数据，uri、selection、selectionArg 同上，被删除的行数作为返回值返回
- getType:根据传入的内容 URI 来返回相应的 MIME 类型

5、UriMatcher 可以实现匹配内容 URI 的功能。

5.1 **提供了一个 addURI 的方法，该方法接受三个参数：权限，路径，和自定义代码，返回值是某个能够匹配这个 Uri 对象所对应的自定义代码。**利用这个代码就能判断出调用方期望访问的是哪张表

6、getType 返回的 MIME 类型

- 必须以 vnd 开头
- **如果内容 URI 以路径结尾，后接"android.cursor.dir/";如果内容以 URI 以 id 结尾，后接"android.cursor.item/"**
- 最后接上 vnd.<authority>.<path>

对应于 URI="content://com.example.app.provider/table1"
MIME="vnd.android.cursor.dir/vnd.com.example.app.provider.table1"
对应于 URI=" content://com.example.app.provider/table1/1"
MIME="vnd.android.cursor.item/vnd.com.example.app.provider.table1"

7、**实现跨程序共享需要将自定义的提供器在 AndroidManifest.xml 文件中注册才可以：在<application>标签内部添加<provider>...</provider>**

```
<application>
    ...
    <provider
        android:name="com.example.databasetest.DatabaseProvider"
        //name:指明所属的类型;authorities:指明权限
        android:authorities="com.example.databasetest.provider"
        android:exported="true">//指明可被其他程序访问
```

```
            </provider>
</application>
```

Chapter 8　运用手机多媒体

8.1　通知

1、首先需要创建 NotificationManager 来对通知进行管理

2、可以通过 Context.getSystemService(Context.NOTIFICATION_SERVICE)来创建

```
NotificationManager manager=(NotificationManager)
                getSystemService(Context.NOTIFICATION_SERVICE);
manager.notify(int id,Notification notification);//显示通知，第一个参数表示通知的 id 号
```

3、具体的步骤，书上的是错误的，Notification.setLatestEventInfo()现在已经被移除了

```
NotificationManager manager=(NotificationManager)
                                        getSystemService(NOTIFICATION_SERVICE);
Notification.Builder builder=new Notification.Builder(MainActivity.this);
builder.setSmallIcon(R.drawable.th_ieldrunners);//设置通知图标
builder.setTicker("Something Important!!!");//设置提示语
builder.setContentTitle("This is content title");//设置内容标题
builder.setContentText("This is content text");//设置内容文本
builder.setWhen(System.currentTimeMillis());//设置显示的通知时间
builder.setDefaults(Notification.DEFAULT_ALL);//?
builder.setContentIntent(pi);//设置点击内容要启动的 PendingIntent
Notification notification=builder.build();//创建 Notification 实例
manager.notify(1,notification);
```

4、PendingIntent

```
public static PendingIntent getActivity(
        Context context,
        int requestCode, //一般用不到，传入 0 即可
        Intent intent,
        @Flags int flags)//确定 PendingIntent 的行为
```

getActivity 可以换成 getBroadcast 或者 getService，参数都一样

- getActivity：通过这个函数名获得的 PendingIntent 用于启动活动
- getBroadcast：通过这个函数名获得的 PendingIntent 用于启动广播
- getService：通过这个函数名获得的 PendingIntent 用于启动服务

5、高级技巧

- 指定通知声音声音

```
Uri soundUri= Uri.fromFile(new
                        File("/system/media/audio/ringtones/Basic_tone.ogg"));
notification.sound=soundUri;
```

- 设定震动模式

```
long[] vibrates={0,1000,1000,1000};//通知来了后，先静止 0ms，震动 1000ms，再静止 0ms，再震动 1000ms
```

notification.vibrate=vibrates;

- LED 灯

6、接受短信（方法比较特别，P310）

接受短信主要利用了广播机制，当接收到一条短信的时候，系统会发出一条值为 android.provider.Telephony.SMS_RECEIVED 的广播，这条广播里携带与短信相关的所有数据，每个应用程序都可以在广播接收器里对它进行监听，收到广播时再从中解析出短信即可

7、发送短信

利用 SmsManager 实例进行短信的发送

```
SmsManager smsManager= SmsManager.getDefault();//获取 SmsManager 实例
PendingIntent pi= PendingIntent.getBroadcast(MainActivity.this,0,sentIntent,0);
//pi 的作用是对短信的发送状态进行监控
smsManager.sendTextMessage(String destinationAddress, String scAddress,
                String text, PendingIntent sentIntent, PendingIntent deliveryIntent)
```

8.2 播放音频(MediaPlayer)

!!!访问手机 SD 卡需要权限!!!

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

方法名	功能
setDataSource()	设置要播放的音频文件的位置
prepare()	开始播放之前调用这个方法完成准备工作
start()	开始或继续播放音频
pause()	暂停播放音频
reset()	将 MediaPlayer 对象重置到刚刚创建的状态
seekTo()	从指定的位置开始播放音频
stop()	停止播放音频，调用这个后，MediaPlayer 对象无法再播放
release()	释放掉与 MediaPlayer 对象相关的资源
isPlaying()	判断当前 MediaPlayer 是否正在播放音频
getDuration()	获取载入的音频文件的时长

初始化：

```
File file=new File(Environment.getExternalStorageDirectory(),<musicPath>);
mediaPlayer.setDataSource(file.getPath());
mediaPlayer.prepare();
```

8.3 播放视频(VideoView)

方法名	功能
setVideoPath()	设置播放视频文件的位置
start()	开始活继续播放视频
pause()	暂停播放视频
resume()	将视频重头开始播放
seekTo()	从指定的位置开始播放视频

isPlaying()	判断当前是否正在播放视频
getDuration()	获取载入的视频文件的时长

更容易初始化，只需要调用 setVideoPath()即可

Chapter 9　服务

9.1　服务

1、**服务是 Android 中实现程序后台运行的解决方案，它非常适用于执行那诶邪恶不需要和用户交互而且还要求长期运行的任务。**

2、**服务并不是运行在一个独立的进程当中，而是依赖于创建服务时所在的应用程序进程。当某个程序进程被杀掉时，所有依赖于该进程的服务都会停止运行。**

9.2　Android 多线程

1、使用与 Java 多线程相同的语法：定义一个线程只需要新建一个类继承自 Thread，然后重写父类的 run()方法，并在里面编写耗时逻辑即可

2、使用继承的耦合性有点高，更多时候选择实现 Runnable 接口

使用语法：

```
MyThread myThread=new MyThread();
new Thread(myThread).start();
```

//Thread 有一个接受 Runnable 参数的构造函数

3、当然还可以用匿名内部类来定义一个实现了 Runnalbe 接口的类

9.2.1 在子线程中更新 UI

Android 的 UI 是线程不安全的：如果想要更改应用程序里的 UI 元素，必须在主线程中进行，否则就会抛出异常。

9.2.2 异步消息处理机制

1、Android 的异步消息处理主要由四个部分组成，Message、Handler、MessageQueue 和 Looper。

- Message：在线程之间传递的消息，它可以在内部携带少量的信息，用于在不同线程之间小环数据。字段有：what;arg1,arg2(携带整型数据);obj(携带 Object 对象)
- Handler：用于发送和处理消息，发送消息使用 Handler.sendMessage()方法；**而发出的消息经过一系列处理后，会传递到 Handler.handleMessage()（覆盖该方法，实现自己需要的处理逻辑）方法中（该方法会被自动调用）**
- MessageQueue：消息队列，主要用于存放所有通过 Handler 发送的消息，这部分消息会一直存在于消息队列中，等待被处理，每个线程中只会有一个 MessageQueue 对象
- Looper：每个线程中 MessageQueue 的管家，调用 Looper.loop()方法后，就会进入到无限循环中，每当发现 MessageQueue 存在一条消息，就会将它取出，并传递到 Handler 的 handleMessage()方法中。每个线程只有一个 Looper 对象

2、Handler 在主线程中创建，因此 handleMessage()方法中的代码也会在主线程中运行，于是可以进行 UI 操作了。

3、处理流程键 P349，一条 Message 经过这样一个流程的辗转调用后，从子线程进入到了主线程，从不能更新 UI 到可以更新 UI

9.2.3 AsyncTask

1、AsyncTask 是基于异步消息处理机制的，android 帮我们做了很好的封装。

2、AsyncTask 是一个抽象类，如果想要使用它，必须创建一个子类去继承它，在继承时，我们可以为 AsyncTask 类指定三个泛型参数。

- Params：在执行 AsyncTask 时需要传入的参数，可用于在后台任务中使用
- Progress：后台任务执行时，如果需要在界面上显示当前的进度，则使用这里指定的泛型参数作为进度单位
- Result：当任务执行完毕后，如果需要对结果进行返回，则使用这里的泛型参数作为返回值的类型

例如：

```
class DownloadTask extends AsyncTask<Void,Integer,Boolean>
```

3、经常需要重写的方法：

onPreExecute()：这个方法会在后台任务开始执行之前调用，用于进行一些界面上的初始化操作，比如显示一个进度条对话框等

- doInBackground(Params...)：这个方法中的所有代码都会在子线程中运行，我们应该在这里去处理所有的耗时任务。任务一旦完成就通过 return 语句返回任务执行结果。如果 AsyncTask 第三个参数指定为 Void，可以
- 返回任务执行结果。这个方法是不能进行 UI 操作的，如果要更新 UI 元素，比如反馈当前任务的执行进度，可以调用 publicProgress(Progress...)方法来完成
- onProgressUpdate(Progress...) ： 当在后台任务中调用了 publicProgress(Progress...)方法后，这个方法就会很快被调用，方法中携带的参数就是在后台任务中传递过来的。这个方法可以对 UI 进行操作，利用参数中的数值就可以对界面元素进行相应的更新
- onPostExecute(Result)：当后台任务执行完毕并通过 return 语句进行返回时，这个方法就很快会被调用。返回的数据会作为参数传递到此方法中。可以利用返回数据来进行一些 UI 操作，比如：提醒任务执行的结果，以及关闭掉进度条对话框等。

publicProgress(Progress...)：从子线程切换到主线程，不需要重写！

其中 doInBackground 是在子线程中运行的，onPreExecute、onProgressUpdate、onPostExecute 是在主线程中运行的

9.3 服务的基本用法

9.3.1 定义服务

1、继承自 Service，覆盖抽象方法 public IBinder onBind(Intent intent)

2、方法：

- onCreate()：在服务创建的时候调用
- onStartCommand()：会在每次服务启动的时候调用
- onDestroy()：会在服务销毁的时候调用，回收那些不再使用的资源

3、每一个服务必须要在 AndroidManifest.xml 文件中进行注册才能生效，这是四大组件（活动，服务，广播接收器，内容提供器）的共性。

9.3.2 启动和停止服务

1、启动：

- 首先构建 Intent 对象
  Intent startIntent=new Intent(Context context,Class<?>cls)
- 调用 Context.startService(startIntent)

2、停止：

- 首先构建 Intent 对象
  Intent stopIntent=new Intent(Context context,Class<?>cls)
- 调用 Context.stopService(stopIntent)

9.3.3 活动和服务的通信-onBind

1、绑定
public boolean bindService(Intent service, ServiceConnection conn, int flags)
2、解绑
public void unbindService(ServiceConnection conn)

3、ServiceConnection 中包含两个方法

- onServiceConnected(ComponentName name)：活动与服务成功绑定的时候调用
- onServiceDisconnected(ComponentName name,IBinder service)：活动与服务接触绑定的时候调用

4、逻辑：

- 服务中：
  - 创建一个 IBinder 对象 iBinder（用匿名内部类定义自己所需要的方法，**继承 Binder，因为想继承其他 Binder 含有的功能**）
  - 在 onBind()方法中返回这个 IBinder 对象 iBinder（与活动的通信中，会调用 onBind()方法）
- 活动中：
  - 定义实现了 ServiceConnection 接口的对象，并覆盖 onServiceConnected(...)与 onServiceDisconnected(...)（一般用匿名内部类）
  - 在 onServiceConnected(...)中，第二个参数类型是 IBinder，传入的就是 onBind()方法返回的参数 iBinder，即获取了包含服务中任务方法的对象。于是在活动中就能自由调用服务中所定义的方法了。

5、IBinder 与 Binder 的区别：

- IBinder：接口
- Binder：实现了 IBinder 接口的类

9.4 服务的生命周期

1、**在项目任何位置调用了 Context.startService()方法，相应的服务就会启动起来，并回调 onStartCommand(...)方法。如果这个活动之前没有创建过，onCreate(...)会先于 onStartCommand(...)方法执行**
2、**每个服务只会存在一个实例，无论调用多少次 startService(...)，只需调用一次 stopService(...)或 stopSelf(...)方法就会停止下来**

3、调用 Context.bindService(...)来获取一个服务的持久连接，这时会回调 onBind(...)方法，类似如果这个服务没有被创建过，会先调用 onCreate()。调用方法可以获取到 onBind(...)方法里返回的 IBinder 对象的实例，就能和活动自由通信了

4、onDestroy

- 当用 Context.startService(...)启动服务，调用 Context.stopService(...)，onDestroy 会执行。
- 当用 Context.bindService(...)启动服务，调用 Context.unbindService(...)，onDestroy 会执行
- 当 Context.startService(...)、Context.bindService(...)均被调用，那么根据 Android 系统的机制，一个服务只要被启动或被绑定之后就会一直处于运行状态，必须两者同时不满足，服务才能被销毁

9.5 服务的更多技巧

9.5.1 前台服务

1、服务几乎都在后台运行，但是服务系统优先级比较低，当内存不足时，可能回收掉正在后台运行的服务。

2、前台服务和普通服务最大的区别就在于，它会一直有一个正在运行的图标在系统的状态栏显示，下拉状态栏后可以看到详细的信息，类似于通知的效果

3、构建出 Notification 对象，调用 startForeground()方法（通知是利用 NotificationManager 对象来显示的），会让服务变成一个前台服务，并在系统状态栏显示出来

9.5.2 使用 IntentService

1、如果服务中的代码在主线程中运行，并且在服务里处理一些耗时的逻辑，就很容易出现 ANR 的情况。

2、为了避免出现 ANR，我们应该在服务的每个具体方法里开启一个子线程，然后在这里去处理耗时逻辑。这种服务一旦启动后就会一直处于运行状态，必须调用 stopService()或者 stopSelf()才能让服务停止下来。

3、为了可以简单创建一个异步、会自动停止的服务，Android 提供了一个 IntentService 类。

4、启动这类服务与启动普通服务完全一致，唯一的区别就是创建的 Intent 对象，第二个参数：普通服务就是 Class<? extends Service>，IntentService 就是 Class<? extends IntentService>

9.6 最佳实践--在后台执行定时任务

1、定时任务一般有两种实现方式，一种是使用 Java API 的 Timer 类，另一种是使用 Android 中的 Alarm 机制。

2、Timer 不适用于需要长期在后台运行的定时任务，因为，为了能让电池更耐用，每种手机都会有休眠策略，Android 手机就会在长时间不操作的情况下让 CPU 进入睡眠状态，这可能导致 Timer 无法正常运行。而 Alarm 具有唤醒功能，即保证每次需要执行定时任务时，CPU 都能正常工作。

3、Alarm 主要借助 AlarmManager 类来实现，与 NotificationManager 类似，通过调用 Context.getSystemService(...)方法来获取实例。这里传入的参数是

Context.ALARM_SERVICE

4、语法

```
AlarmManager manager=(AlarmManager) getSystemService(ALARM_SERVICE);
long triggerAtTime= SystemClock.elapsedRealtime()+anHour;
manager.set(int type, long triggerAtMillis, PendingIntent operation);
```

type：

- ELAPSED_REALTIME：表示任务触发时间从系统开机算起，不会唤醒 CPU
- ELAPSED_REALTIME_WAKEUP：表示让定时任务触发时间从系统开机算起，但会唤醒 CPU
- RTC：表示任务的触发时间从 1970 年 1 月 1 日 0 时算起，不会唤醒 CPU
- RTC_WAKEUP：表示任务的触发时间从 1970 年 1 月 1 日 0 时算起，但会唤醒 CPU

5、**从 Android4.4 开始 Alarm 触发时间会变得不准确，这是因为系统在耗电性方面进行了优化，系统会自动检测目前有多少个 Alarm 任务存在，然后将触发时间相近的几个任务放在一起执行，可以大幅度减少 CPU 被唤醒的次数，从而延长电池使用时间。如果想要准确定时，可以利用 AlarmManager.setExact(...)方法代替 AlarmManager.set(...)。**

Chapter 10 网络技术

10.1 WebView

```
WebView.getSettings().setJavaScriptEnabled(true);
//使 WebView 支持 JavaScript 脚本
WebView.getWebViewClient(new WebViewClient());
//当从一个网页跳转到另一个网页时，我们希望目标网页仍在当前的 WebView 中显示，而不是打开系统浏览器
WebView.loadUrl("http://www.baidu.com")
//载入网页
```

10.2 HTTP 协议访问网络

10.2.1 使用 HttpURLConnection

1、Android 上发送 HTTP 请求的方式有两种：HttpURLConnection 和 HttpClient

2、步骤：

- 首先需要获取 HttpURLConnection 实例，一般用 new 创建 URL 对象（构造器接受包含目标网址的 String），然后对该 URL 对象调用 openConnection()方法即可创建 HttpURLConnection 实例。
  - URL url=new URL("http://www.baidu.com");
  - HttpURLConnection connection=
    (HttpURLConnection) url.openConnection();
- 得到 HttpURLConnection 实例之后，可以设置一下 HTTP 请求所使用的方法。常用的有 GET 和 POST：
  - GET：表示希望从服务器那里获取数据
  - POST：表示希望提交数据给服务器
  - connection.setRequestMethod("GET")
- 接下来进行自由的定制，比如设置链接超时，读取超时的毫秒数，以及服务器希望得到的一些消息头等
  - connection.setConnectTimeout(8000);
  - connection.setReadTimeout(8000);
- 之后调用 getInputStream()方法
  - InputStream in=connection.getInputStream();
- 最后调用 disconnect()方法将 HTTP 连接关闭掉
  - connection.disconnect();

3、如果要提交数据给服务器

- connection.setRequestMethod("POST");
- DataOutputStream out=
  new DataOutputStream(connection.getOutputStream());
- out.writeBytes("username=admin&password=123456");//每条数据要以键值对的形式存，并且以"&"符号隔开

10.2.2 使用 HttpClient（在 android 6.0 已被移除）

1、HttpClient 是 Apache 提供的 HTTP 网络访问接口，从一开始的时候就被引入到了 Android 的 API 中。它可以完成和 HttpURLConnection 几乎一模一样的效果，

但两者之间的用法却有较大的差别。

2、HttpClient 是一个接口，通常情况会创建 DefaultHttpClient 的实例

3、步骤

- **创建 HttpClient 对象**
  - HttpClient httpClient=new DefaultHttpClient();
- **接下来，如果想要发起一条 GET 请求，需要创建 HttpGet 对象**
  - HttpGet httpGet=new HttpGet("http://www.baidu.com");
  - HttpRespnse httpResponse=httpClient.execute(httpGet);
- 如果发起 POST 请求，需要创建 HttpPost
  - HttpPost httpPost=new HttpPost("http://www.baidu.com");
  - **以下步骤等效于 HttpURLConnection 的以&链接的键值对的形式**
  - List<NameValuePair> params=new ArrayList<NameValuePair>();
  - params.add(new BasicNameValuePair("username","admin");
  - params.add(new BasicNameValuePair("password",123456");
  - UrlEncodedFromEntity entity=
    new UrlEncodedFormEntity(params,"utf-8");
  - HttpRespnse httpResponse=httpPost.execute(httpPost);
- 执行 execute()方法后，会返回一个 HttpResponse 对象，服务器所返回的所有信息就会包含在这里面，先取出服务器返回的状态码，若等于 200 说明请求和响应都成功了
- 接下来在这个 if 判断的内部取出服返回的具体内容，调用 getEntiry()方法获取到 HttpEntity 实例，然后调用 *EntityUtils*.toString()将 HttpEntity 转换成字符串即可
  - HttpEntiry entity=httpResponse.getEntity();
  - String response=EntityUtils.toString(entity);
  - 如果服务器返回数据携带中文的话，直接调用 *EntityUtils*.toString()会出现乱码，只需要指定字符集为"utf-8"即可
  - String response=EntityUtils.toString(entity,"utf-8");

10.3 解析 XML 格式数据

10.3.1 Pull 解析方式

1、例子：

```
private void parseXMLWithPull(String xmlData){
    try{
        XmlPullParserFactory factory=XmlPullParserFactory.newInstance();//获取工厂实例
        XmlPullParser xmlPullParser=factory.newPullParser();//获取 XmlPullParser 对象
        xmlPullParser.setInput(new StringReader(xmlData));//设置要解析的数据
        int eventType=xmlPullParser.getEventType();//获取当前事件类型
        String id="";
        String name="";
        String version="";
        while(eventType!=XmlPullParser.END_DOCUMENT){//当前事件不等于时间尾,继续解析
            String nodeName=xmlPullParser.getName();//获取当前节点的名字(标签名字)
            switch(eventType){
                case XmlPullParser.START_TAG:
                    if("id".equals(nodeName)){
                        id=xmlPullParser.nextText();//nextText()获取节点具体的内容(标签内夹的内容)
                    }else if("name".equals(nodeName)){
                        name=xmlPullParser.nextText();
```

```
                }else if("version".equals(nodeName)){
                    version=xmlPullParser.nextText();
                }
                break;
            case XmlPullParser.END_TAG:
                if("app".equals(nodeName)){//只在 app 标签结束时打印信息
                    Log.d("MainActivity","id is "+id);
                    Log.d("MainActivity","name is "+name);
                    Log.d("MainActivity","version is "+version);
                }
                break;
            default:
                break;
            }
            eventType=xmlPullParser.next();//更新当前事件的类型
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

2、标志说明：

- XmlPullParser.START_TAG：标签开始标志
- XmlPullParser.END_TAG：标签结束标志
- XmlPullParser.END_DOCUMENT：整个文件的结束标志

10.3.2 SAX 解析方式

1、一般会新建类继承自 DefaultHandler,并重写父类的五个方法:

- public void **startDocument**() throws SAXException{}
- public void **startElement**(String uri,String localName,String qName,Attributes attributes) throws SAXException{}
- public void **characters**(char[] ch,int start,int length) throws SAXexception{}
- public void **endElement**(String uri,String localName,String qName)throws SAXException{}
- public void **endDocument**() throws SAXException

10.4 解析 JSON 格式数据

1、JSON 的优势在于体积更小，更节省流量，缺点在于语义较差，不如 XML 直观

2、解析 JSON 数据的方法

- 官方提供的 JSONObject
- 谷歌开源库 GSON
- 第三方开源库 Jackson、FastJSON

10.4.1 使用 JSONObject

1、例子：

```
private void parseJSONWithJSONObject(String jsonData){
    try{
        JSONArray jsonArray=new JSONArray(jsonData);//得到 JSON 数组
        for(int i=0;i<jsonArray.length();i++){
            JSONObject jsonObject=jsonArray.getJSONObject(i);//从 JSON 数组中取出的每一个元素都是一个
JSONObject 对象,每个 JSONObject 对象又包含了 id name version 等数据,用 getString()方法取出对应于键的
```

```
值即可
            String id=jsonObject.getString("id");
            String name=jsonObject.getString("name");
            String version=jsonObject.getString("version");
            Log.d("MainActivity","id is "+id);
            Log.d("MainActivity","name is "+name);
            Log.d("MainActivity","version is "+version);
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

10.4.2 使用 GSON

```
private void parseJSONWithGSON(String jsonData){
    Gson gson=new Gson();
    List<App> appList=gson.fromJson(jsonData,new TypeToken<List<App>>(){}.getType());
    for(App app:appList){
        Log.d("MainActivity","id is "+app.getId());
        Log.d("MainActivity","name is "+app.getName());
        Log.d("MainActivity","version is "+app.getVersion());
    }
}
```

10.5 网络编程的最佳实践

1、一个应用程序很可能会在许多地方都用到网络功能，而发送 HTTP 请求的代码基本都是相同的，如果每次都去编写发送 HTTP 请求的代码，这显然是非常差劲的做法。

2、通常我们将这些通过网络操作提取到一个公共类里，并提供一个静态方法，想要发起网络请求的时候只需要简单地调用一下这个方法即可。

3、网络请求通常属于耗时操作，在静态方法里开启一个线程即可。

4、子线程可能会在服务器还没来得及响应的时候就执行结束了，利用回调机制可以解决这个问题

Chapter 11　Android 特色开发，基于位置的服务

11.1　基于位置的服务简介

1、基于位置的服务简称 LBS。

2、主要工作原理就是利用无线通讯网络或 GPS 定位方式来确定移动设备的所在位置。

3、有了 Android 系统作为载体，我们可以利用定位出的位置进行许多丰富的操作。

11.2　找到自己的位置

11.2.1　LocationManager 的基本用法

步骤：

- 获取实例：Context.getSystemService(LOCATION_SERVICE);
- 获取位置提供器的 String 对象

```
List<String> providerList=locationManager.getProviders(true);//传入 true 表示只有"状态为启用"的位置提供器才会返回
if(providerList.contains(LocationManager.GPS_PROVIDER)){
   provider=LocationManager.GPS_PROVIDER;
}else if(providerList.contains(LocationManager.NETWORK_PROVIDER)){
   provider=LocationManager.NETWORK_PROVIDER;
}else{
   Toast.makeText(this,"No location provider to use",Toast.LENGTH_SHORT).show();
   return;
}
```

- 选择好的位置提供器，传入到 LocationManager.getLastKnownLocation(...)方法中，就可以得到一个 Location 对象

```
Location location=locationManager.getLastKnownLocation(provider);
```

- 调用 LocationManager.requestLocationUpdates()方法，更新位置信息

```
locationManager.requestLocationUpdates(provider,5000,1,locationListener);
```

  - 第一个参数：位置提供器
  - 第二个参数：监听位置变化时间间隔（毫秒）
  - 第三个参数：监听位置变化距离讲个（米）
  - 第四个参数 LocationListener 监听器

11.3　反向地理编码，看得懂的位置信息

11.3.1　Geocoding API 用法

1、工作原理：手机端向谷歌服务器发起一条 HTTP 请求，并将经纬度的值作为参数一同传递过去，然后服务器会帮我们将这个经纬度转换成看得懂的位置信息，再将这些信息返回给手机端，最后手机端去解析服务器返回的信息，并进行处理就可以。

2、http://maps.googleapis.com/maps/api/geocode/json?latlng=40.714224,-73.961452&sensor=true_or_false

- 红色表示固定
- json：表示返回 json 格式的数据
- 也可以指定成 xml
- sensor=true_or_false：表示这条请求是否来自于某个设备的位置传感器

11.3.2 对经纬度进行解析

1、首先发送 HTTP 请求给谷歌服务器，然后再对返回的 JSON 数据进行解析。

11.4 使用百度地图

11.4.1 申请 API Key

步骤：

- http://developer.baidu.com/user/reg ：进行注册
- http://lbsyun.baidu.com/apiconsole/key：
  - 选中标题栏的"开发"
  - 选中"android 开发"
  - 选中"android 地图 SDK"
  - 在 Android 地图 SDK vx.x.x 框内点击链接字体"申请秘钥"
  - 选中"创建应用"
    - 填写名称，应用类型
    - 安全码如何获取：
      - Mac 平台下：在终端输入：keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
  - 完成！取得 API Key

11.4.2 让地图显示出来

1、准备百度地图 android 版 sdk

- 将 baidumapapi_v3_1_0.jar 复制到 libs 文件夹下，右击 baidumapapi_v3_1_0.jar 选中"Add As Libraryv
- 创建 armeabi 与 armeabi-v7a 文件夹，将 libBaiduMapSDK_v3_1_0.so 复制到这两个文件夹下，**并且修改 build.gradle 文件，添加如下语句**：

```
android {
    sourceSets {
        main() {
            jniLibs.srcDirs = ['libs']
        }
    }
    //...原来的代码
}
```

2、权限以及<meta-data>标签

11.4.3 定位到我的位置

1、**百度地图提供了 BaiduMap 类，它是地图的总控制器，调用 MapView.getMap()方法就能获取到 BaiduMap 的实例。**

```
MapView mapView=(MapView) findViewById(R.id.map_view);

BaiduMap baiduMap=mapView.getMap();
```

2、百度地图的缩放级别取值范围在 3-19 之间，小数点也是可取的，取值越大，显式得信息就越精细，例如

```
MapStatusUpdate update=MapStatusUpdateFactory.zoomTo(16f);
baiduMap.animateMapStatus(update);
```

3、LatLng 类：主要用于存放经纬度值，其构造器接受两个参数，第一个参数是纬度值，第二个参数是经度值。我们调用 *MapStatusUpdateFactory*.newLatLng()方法将 LatLng 对象传入，该方法返回一个 MapStatusUpdate 对象，最后将其传入 BaiduMap.animateMapStatus()方法即可。

```
LatLng ll=new LatLng(location.getLatitude(),location.getLongitude());
update= MapStatusUpdateFactory.newLatLng(ll);
baiduMap.animateMapStatus(update);
```

11.4.4 让我显示在地图上

1、百度地图 API 当中提供了一个 MyLocationData.Builder 类，这个类是用来封装设备当前所在位置的，我们只需要将经纬度信息传入到这个类的相应方法中即可。

```
MyLocationData.Builder locationBuilder=new MyLocationData.Builder();
locationBuilder.latitude(location.getLatitude());
locationBuilder.longitude(location.getLongitude());
MyLocationData locationData=locationBuilder.build();
baiduMap.setMyLocationData(locationData);
```

需要在 onCreate：

```
baiduMap.setMyLocationEnabled(true);
```

需要在 onDestroy：

```
baiduMap.setMyLocationEnabled(false);
```

11.5 Git 时间，版本控制工具的高级用法

Chapter 12　Android 特色开发，使用传感器

12.1 传感器简介

1、Android 通常支持多种传感器，如光照传感器，加速度传感器，地磁传感器，压力传感器，温度传感器。

12.2 光照传感器

1、步骤

- 首先获取传 SensorManager 的实例
  - SensorManager sensorManager=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
  - **SensorManager 是系统所有传感器的管理器，有它的实例后就可以调用 getDefaultSensor()方法来得到任意传感器类型了**
- 接下来要对传感器输出的信号进行监听，这需要借助 SensorEventListener 来实现。SensorEventListener 是一个接口，包含以下方法
  - public void onAccuracyChanged(Sensor sensor,int acuracy)
  - public void onSensorChanged(SensorEvent event)
- 当传感器的经度发生变化时会调用 onAccuracyChanged()方法，当传感器检测到数值发生变化时就会调用 onSensorChanged()方法。
- **然后用 SensorManager.registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs)**
  - 第三个参数表示传感器的更新速率，可选值有（更新速率依次递增）
    - SENSOR_DELAY_UI
    - SENSOR_DELAY_NORMAL
    - SENSOR_DELAY_GAME
    - SENSOR_DELAY_FASTEST
- **最后别忘了调用 SensorManager.unregisterListener(SensorEventListener listener)将使用的资源释放掉。**

12.3 加速度传感器

1、步骤

- 与光照传感器相同，只是获取 Sensor 实例的时候需要指定加速度传感器的常量
  - Sensor sensor=sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

12.4 方向传感器

1、Android 已经废弃了 Sensor.TYPE+ORIENTATION 这种传感器类型，虽然代码还是有效的，但是不推荐这么写。

2、目前，Android 获取手机旋转的方向和角度是通过加速度传感器和地磁传感器共同计算得出的。

3、步骤

- 与光照传感器相同，首先获取 SensorManager 的对象，然后，需要获取加速度传感器和地磁传感器的实例，并给他们注册监听器，监听器的更新速率提高到 SENSOR_DELAY_GAME

4、**其中监听器在一个时刻只会有一种类型的 SensorEvent 对象传入，判断它是加速度传感器还是地磁传感器（光照传感器的时候，为什么传入的一定是光照传感器变化的事件，什么地方对其进行了设定？注册的时候吗？），**并对相应的数组进行更新。

Chapter 13 高级技巧

13.1 获取全局 Context

1、Android 提供了一个 Application 类，每当应用程序启动的时候，系统就会自动将这个类进行初始化。而我们可以定制一个自己的 Application 类，以便于管理程序内一些全局的状态信息，比如说 Context。

2、例子：

```
public class MyApplication extends Application {
    private static Context context;
    @Override
    public void onCreate(){
        super.onCreate();
        context=getApplicationContext();
    }
    public static Context getContext(){
        return context;
    }
}
```

- 重写了父类的 onCreate()方法，并通过调用 getApplicationContext()方法得到一个应用程序级别的 Context，然后提供了静态方法 getContext()，将获取到的 Context 返回
- 需要在 AndroidManifest.xml 进行注册

13.2 使用 Intent 传递对象

1、Intent.putExtra()方法中所支持的数据类型是有限的，但想要传递自定义对象的时候就会无从下手。

13.2.1 Serializable 方式

1、Serializable 是序列化的意思，表示将一个对象转换成可存储或可传输的状态（将该对象转换成一个字节序列，并且会覆盖整个对象网络）

2、步骤：（以 Person 为例）

- 只需要让需要传输的类继承自 Serializable 接口即可（没有方法需要重写）。
- Intent.putExtra("person_data",person);
- Person person=(Person) getIntent().getSerializableExtra("person_data");

13.2.2 Parcelable 方式

1、不同于序列化，**Parcelable 方式实现的原理是将一个完整的对象进行分解，而分解后的每一部分都是 Intent 所支持的数据类型，这样也就实现传递对象的功能**

2、**感觉有点类似于 Externalizable，需要自己重写 writeExternal() 和 readExternal()方法。**

3、步骤（以 Person 为例）

- 首先，让 Person 实现 Parcelable，需要重写 **describeContents()** 和 **writeToParcel()** 这两个方法，其中 describeContents() 返回 0 即可。writeToParcel() 方法需要将需要保留的字段以此写入 Parcel.writeInt/Parcel.writeString/...
- 必须在 Person 类中提供名为 CREATOR 的常量引用，引用一个以匿名内部类形式创建的 Parcelable.Creator 接口的实例。并且重写该接口的方法
  - public T createFromParcel(Parcel source);
    - **一定要完全按照 writeToParcel 中写入的顺序依次读取**

- public T[] newArray(int size)

- Intent.putExtra("person_data",person);
- Person person=(Person) getIntent().getParcelableExtra("person_data");

4、**Serializable 的方式比较简单，但是会把整个对象进行序列化，因此效率方面比较低，因此，通常情况下推荐使用 Parcelable**

13.3 定制自己的日志工具

1、Android 自带的日志工具：在项目正式上线后仍然会照常打印，这样会降低程序运行效率，还有可能将一些机密数据泄露出去。

2、理想情况是：开发阶段就让日志打印出来，当程序上线了之后就把日志屏蔽掉。

3、步骤：（以 LogUtil 为例）

- 新建自己的日志类 LogUtil，设置常量静态数据 VERBOSE DEBUG INFO ERROR NOTHING 等等。添加静态常量属性 LEVEL
- 编写静态方法，LogUtil.v/d/e...，在里面调用 Log.v/d/e，加一个判断条件
  - if (LEVEL<=VERBOSE) if (LEVEL<=debug)...

4、**通过调整 LEVEL 的值，来选择屏蔽等级，小于 LEVEL 都无法显示**

13.4 调试 Android 程序

13.5 编写测试用例

1、对于大型项目，给每一项功能都编写了测试用例，每当修改或新增任何功能后，就讲所有的测试用例都跑一遍，只要有任何测试用例没有通过，就说明修改或新增的这个功能影响到了现有功能。

错误：

1、创建 layout 文件时，更改 leyout 的类型，居然变成了 RelativeLinearLayout,编译器也不会检查！！！

2、ExitText 的定义放在 onCreate，然后调用 ExitTest.getText().toString()会有编译错误，要把 ExitText 定义成字段而不是函数内部的临时变量

问题

1、在利用 ContentValues 进行插入或者更新时，若某些列没有填值，那么就视为不变？

2、打印函数调用栈

Log.e("MainActivity",Log.getStackTraceString(e));

权限

- 查询系统网络状态的权限

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

- 监听系统开机广播的权限

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

- 读取系统联系人的权限

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

- 控制手机震动的权限

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

- 发送短信的权限

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

- 访问外部存储的权限

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

- 访问网络的权限

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

- 获取设备位置信息

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

- 使用百度地图的权限

```
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_SETTINGS"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.BROADCAST_STICKY"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```