1. 环境搭建

1.1. 下载 Java JDK、AndroidSDK、Eclipse、ADT

1.1.1 AndroidSDK 安卓开发工具包

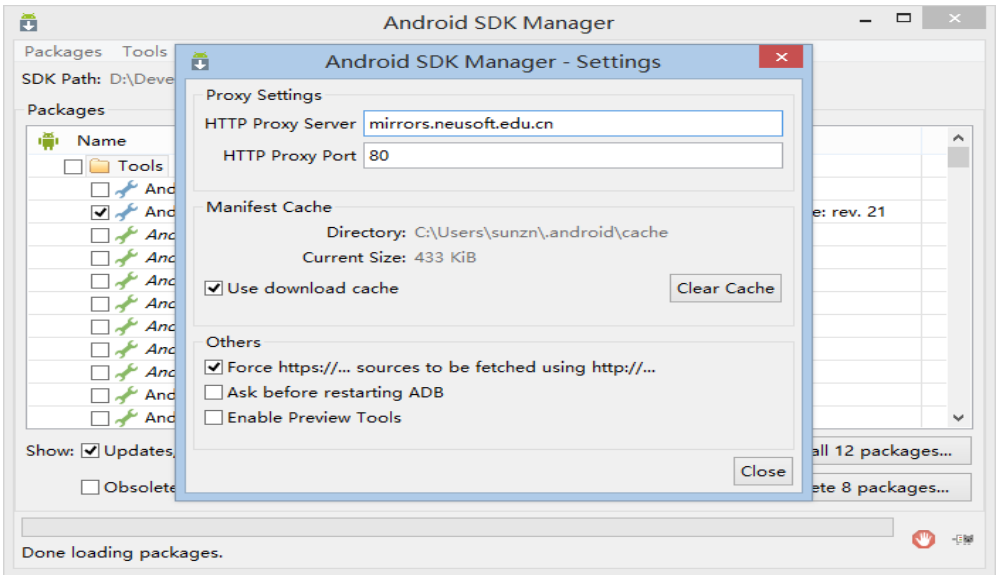
小提示：由于 Google 在中国大陆被封，需要翻墙或寻找国内镜像网站

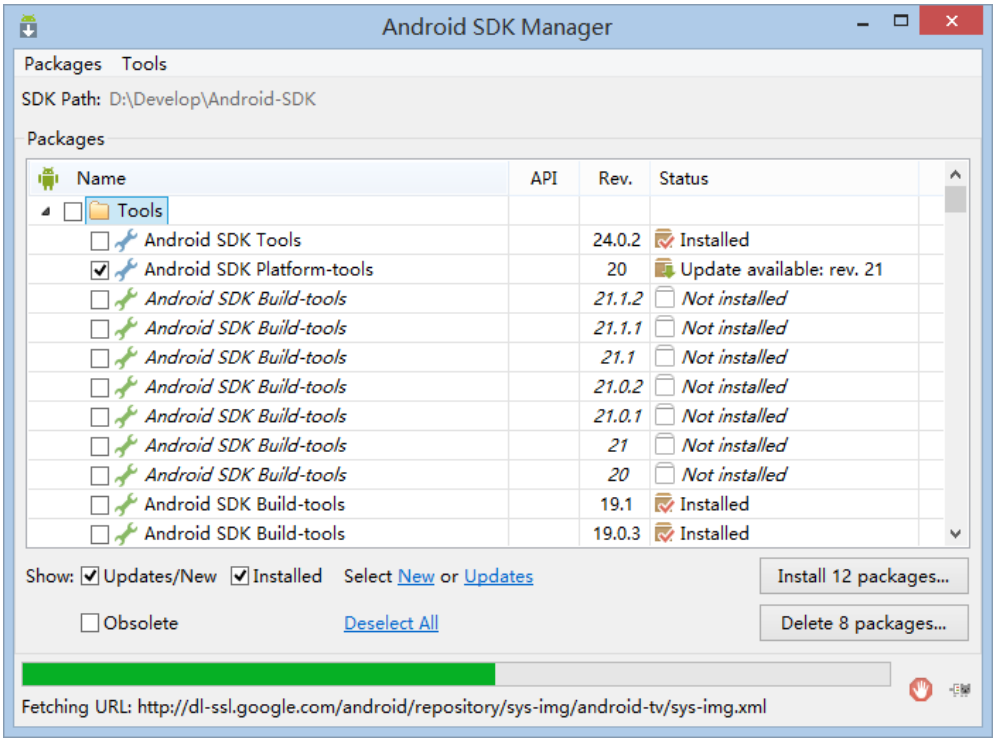
Android SDK Manager 更新代理配置

启动 Android SDK Manager ，打开主界面，依次选择「Tools」、「Options...」，弹出『Android SDK Manager - Settings』窗口；

在『Android SDK Manager - Settings』窗口中，在「HTTP Proxy Server」和「HTTP Proxy Port」输入框内填入 mirrors.neusoft.edu.cn 和 80，并且选中「Force https://... sources to be fetched using http://...」复选框。设置完成后单击「Close」按钮关闭『Android SDK Manager - Settings』窗口返回到主界面；

依次选择「Packages」、「Reload」。





Tools：

Android sdk tools 安卓开发与调试

Android sdk platform-tools 安卓开发平台工具

Android SDK Build-tools 安卓项目构建工具，将安卓项目打包成 APK

Extras:

Android support library 兼容库，jar 包，针对不同版本的 Android 兼容

Google USB Driver 安卓真机测试驱动

Android *.*(API *):

Documentation for Android SDK:帮助文档

SDK platform:此平台的 API，JAR 包

ARM EABI V7a System image:模拟器处理器镜像

Google APIs：谷歌 API，包含如谷歌地图等功能

Source forAndroid sdk：安卓应用源码

1.1.2 安装 ADT 插件

Help-install new software-add(name:),选中 Develop tools-finish-重启 Eclipse

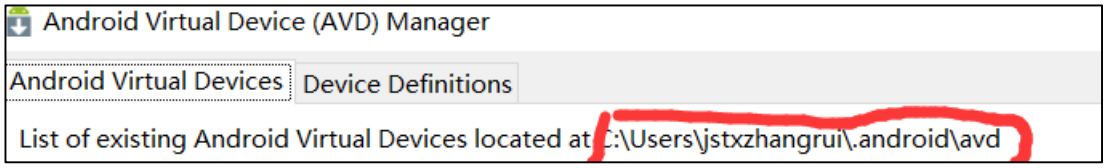
关联 Android sdk:windows-preferences-android,sdk location

1.2 创建模拟器

Android Virtual Device Manager

如果电脑性能不是很好，CPU 选择 ARM，电脑 i3 以上可以选择 Intel atom(x86)速度会快一点

KeyBoard：模拟器是否显示键盘



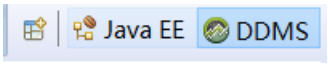
路径最好不要存在中文，否则可能启动不成功

更换路径：

配置环境变量：android_sdk_home，值选择路径

2. 基础知识

2.1 DDMS 安卓调试工具

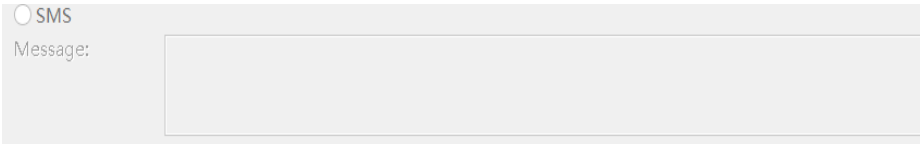


2.1.1 File Exploer 文件管理器

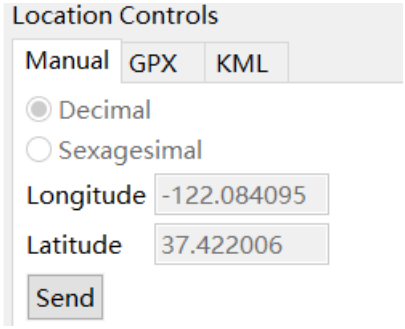
2.1.2Emulator Control 模拟器控制台

网络模式

模拟来电

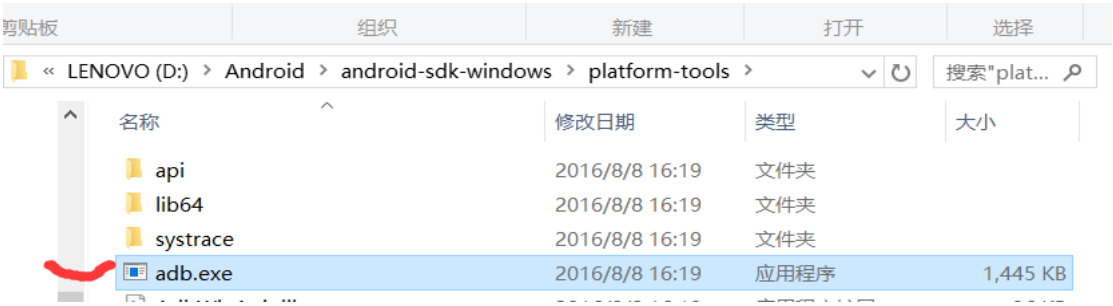


模拟短信



经纬度控制

2.2 adb 工具（android debug bridge）



adb 默认端口：5037

netstat –ano 查看绑定端口应用信息

配置 adb 环境变量，path 加 adb 所在路径

常见命令：

abd install 文件路径　　安装安卓应用

3. 创建第一个安卓项目（HelloWorld）

3.1 新建项目

Application Name:应用程序名称，可以为中文

Project Name:工程名称，可以为中文

Package Name:包名，android 的唯一标识，保证每个应用的包名不一样，不能存在中文

Minimum Required SDK:最低运行安卓版本

Target SDK:最高需求运行版本

Compile With:编译版本

Theme:窗体样式

Create custom launcher icon 创建启动图标

Create activity 创建一个活动

Mark this project as a library 创建此工程为别的工程使用

Foreground:设置图标

3.2 项目结构

src/整个项目源文件目录

gen/自动生成的目录

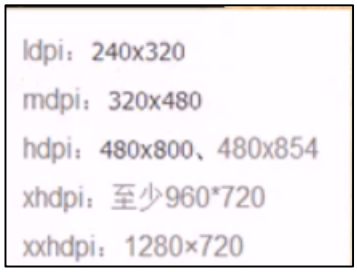
gen 目录中存放所有由 Android 开发工具自动生成的文件。目录中最重要的就是 R.java 文件。Android 开发工具会自动根据你放入 res 目

录的资源，同步更新修改 R.java 文件。应避免人工修改该文件。R.java 在应用中起到字典的作用，它包含了各种资源的 id，通过 R.java，应用可以很方便地找到对应资源。另外编译器也会检查 R.java 列表中的资源是否被使用到，没有被使用到的资源不会编译进软件中，这样可以减少应用在手机占用的空间。

res/资源（Resource）目录

在这个目录中我们可以存放应用使用到的各种资源，如 xml 界面文件，图片或数据。

res/drawable 专门存放 png、jpg 等图标文件。在代码中使用



getResources().getDrawable(resoourceID)获取该目录下的资源

res/layout,专门存放 xml 界面文件，xml 界面文件和 HTML 一样，主要用于显示用户操作界面。

res/values 专门存放应用使用到的各种数据类型。不同的存放在不同的文件中

libs/支持库目录

存放开发时需要用到的第三方 jar 包

assets 资产目录

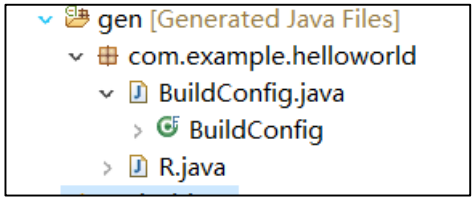
Android 除了提供/res 目录存放资源文件外，在/asesets 目录也可以存放资源文件，而且在/assets 目录下的资源文件不会在 R.java 自动生

成 ID，所以 读取/assets 目录下的文件必须指明文件路径。

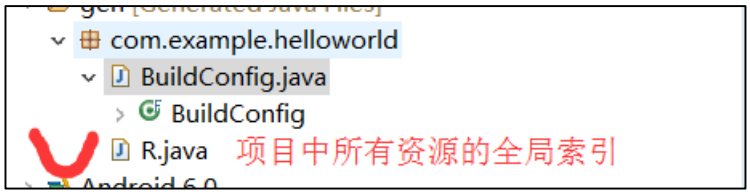
AndroidMainfest.xml 项目清单文件

这个文件列出了应用程序所提供的功能，以后开发好的各种组件需要在该文件中配置，如果应用使用到了系统内置的应用（如电话服务、互联网服务、短信服务、GPS 服务等等），还需要该文件中申明使用权限。

project.properties 项目环境信息，一般不修改此文件




```
public final class BuildConfig {
    public final static boolean DEBUG = true; 项目可调试
}
```



bin:存放编译后的文件

assets 存放了需要释放到手机上的资源

3.3 AdroidManifest.xml（清单文件）介绍

3.4 签名

导出 APK 前需要给应用签名

作用：应用更新验证签名是否一致

4. Android 核心 Activity

Android 四大组件：

Activity、Service、Content Provider、BroadcastReceiver

4.1 Activity

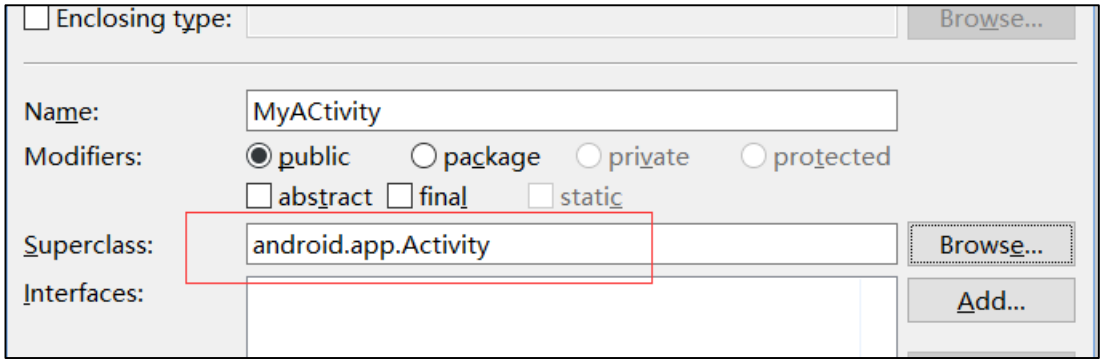
每个 Activity 都会在 AndroidManifest.xml 配置<activity>

MainActivity.java 中

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

把 activity 和布局绑定在一起。

4.1.1 创建 activity



加载布局：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.mylayout);
}
```

配置 activity

```
    </activity>
    <activity android:name="com.example.helloworld.MyACtivity">
        <intent-filter >
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

设为程序主界面

设置在手机界面上显示启动

4.2 Intent

Android 采用了栈的数据结构来管理 Activity,当新的 Activity 启动时，activity 就被压入栈， 当 Activity 结束时 Activity 被弹出栈。

1. Intent 是一种运行时绑定（runtime binding）机制，它能在程序运行时连接两个不同的组件。
2. 组件之间的通讯，主要是由 Intent 协助完成的。

   Intent 负责对应用中一次操作的动作、动作涉及数据、附件数据进行描述.,Andriod 则根据次 Intent 的描述,负责找到对应的组件，将 Intent 传递给调用的相关信息，并完成组件的调用。
3. Intent 起一个媒体中介的作用,专门提供组件相互调用的相关信息,实现调用者与被调用者之间的解耦合。
4. Android 基本的设计理念是鼓励减少组件之间的耦合，因此 Android 提供了 Intent,Intent 提供一种通用的消息系统，它允许在你的应用程序与其他应用程序间传递 Intent 来执行动作和产生事件。使用 Intent 可以激活 Android 应用的三个核心组件:Activity、Servicer、BroadcastReceiver

1. 打开另一个 Activity

```
Intent intent = new Intent();
        intent.setClass(MainActivity.this,MyActivity.class);
```

```
        startActivity(intent);
```

2.显式 Intent

显式 Intent:明确指出了目标组件名称的 Intent

隐式 Intent:没有明确指出组件名称

四种显式 Intent:

实现 OnClickListener 接口

```
public class MainActivity extends Activity implements OnClickListener{
```

添加点击事件

```
11 public class MainActivity extends Activity implements OnClickListener{
12     private Button button1;
13     private Button button2;
14     private Button button3;
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19         button1 = (Button) findViewById(R.id.button1);
20         button2 = (Button) findViewById(R.id.button2);
21         button3 = (Button) findViewById(R.id.button3);
22         button1.setOnClickListener(this);
23         button2.setOnClickListener(this);
24         button3.setOnClickListener(this);
25     }
```

添加到点击事件中

实现接口方法

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
    case R.id.button1:
        |
        break;
    case R.id.button2:

        break;
    case R.id.button3:

        break;

    default:
        break;
    }
}
```

1.

```
Intent intent = new Intent();
        intent.setClass(MainActivity.this,MyActivity.class);
        startActivity(intent);
```

2.

```
    Intent intent2 = new Intent();
            intent2.setClassName(MainActivity.this,
"com.example.helloworld.MyActivity"); //参数二，被打开的Activity全类名
            startActivity(intent2);
```

3.利用此方法还可以打开其他应用

```
            Intent intent3 = new Intent();
            intent3.setClassName("com.example.helloworld" ,
"com.example.helloworld.MyActivity");//参数一，应用程序类名，在
AndroidMainfest.xml中可以找到；参数二，被打开的Activity全类名
            startActivity(intent3);
```

```
    <activity android:name="com.example.helloworld.MyActivity"
        android:exported="true">
    </activity>
</application>
```

默认是false,若要使当前应用的Activity能被其他应用调用，应设置为true

若打开的是另一个应用的主界面，则不需要设置

4.

```
Intent intent4 = new Intent();
            intent4.setComponent(new ComponentName(MainActivity.this,
MyActivity.class));
            startActivity(intent4);
```

使用 Intent 可以激活 Android 应用的三个核心组件：Activity、Service、BroadcastReceiver

3. 隐式 Intent

```
<activity android:name="com.example.helloworld.MyActivity"
    android:exported="true">
    <intent-filter >
        <action android:name="com.zhangrui.myactivity"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

设置过滤器，匹配相应的Activity

```
Intent intent5 = new Intent();
            intent5.setAction("com.zhangrui.myactivity");
            intent5.addCategory(Intent.CATEGORY_DEFAULT);
            startActivity(intent5);
```

多个应用的不同 Activity 可以设置一个相同的 name,可以做成类似点击分享到微博、QQ、微信等的功能。

4.Intent Filter(意图过滤器)

⊙Android 使用 Intent Filter 来过滤掉与 Intent Filter 内容不符的隐式 Intent Filter 。

⊙一个没有申明 Intent Filter 的组件只能响应指明自己名字的显示 Intent 请求，而无法响应隐式 Intent 请求。如果申明了，两者皆可。在通过和 Intent Filter 比较来解析隐式 Intent 请求时，Android 将（Action、Data、Category）三个因素作为选择的参考标准。

⊙一个 Intent Filter 至少应该配置一个 Action

⊙data 元素可以指定了希望接收的 Intent 请求的数据 uri 和数据类型，uri 被分为三个部分来进行匹配：scheme、authority 和 path。其中 setData（）设定的 Intent 请求的 uri 数据类型和 scheme 必须与 Intent Filter 中所指定的一致。若 Intent Filter 中还指定了 authority 和 path，他们也必须匹配才会通过测试。

每个数据 data 元素可以指定一个 uri 和一个数据类型（MIME 媒体类型）。有一些单独的属性-模式，主机，端口和路径 Uri 的每个部分：

scheme：//host:port/path

Tips:

MIME(Multipurpose Internet Mail Extensions)多用途互联网邮件扩展类型。是设定某种扩展名的文件用一种应用程序来打开的方式类型，当该扩展名文件被访问的时候，浏览器会自动使用指定应用程序来打开。多用于指定一些客户端自定义的文件名，以及一些媒体文件打开方式。

常见 MIME 类型：

xml 文档 .xml text/xml

XHTML 文档 .xhtml application/xhtml+xml

普通文本 .txt text/plain

RTF 文本 .rtf application/rtf

PDF 文档 .pdf application/pdf

Microsoft Word 文件 .word application/msword

PNG 图像 .png image/png

GIF 图形 .gif image/gif

JPEG 图形 .jpeg,.jpg image/jpeg

au 声音文件 .au audio/basic

MIDI 音乐文件 mid,.midi audio/midi,audio/x-midi

RealAudio 音乐文件 .ra, .ram audio/x-pn-realaudio

MPEG 文件 .mpg,.mpeg video/mpeg

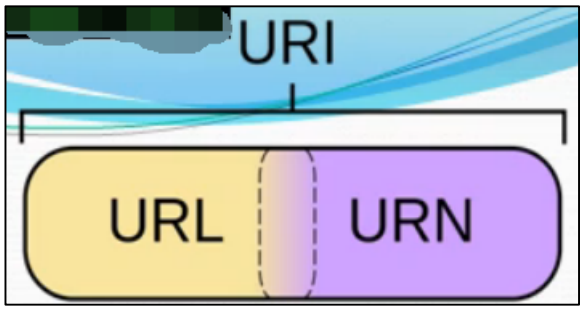
AVI 文件 .avi video/x-msvideo

GZIP 文件 .gz application/x-gzip

TAR 文件 .tar application/x-tar

任意的二进制数据 application/octet-stream

4.3 URI（通用资源标志符）

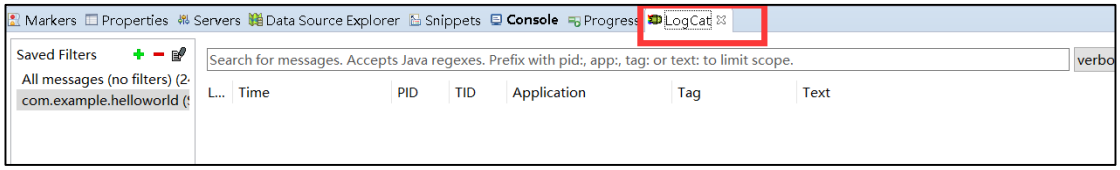


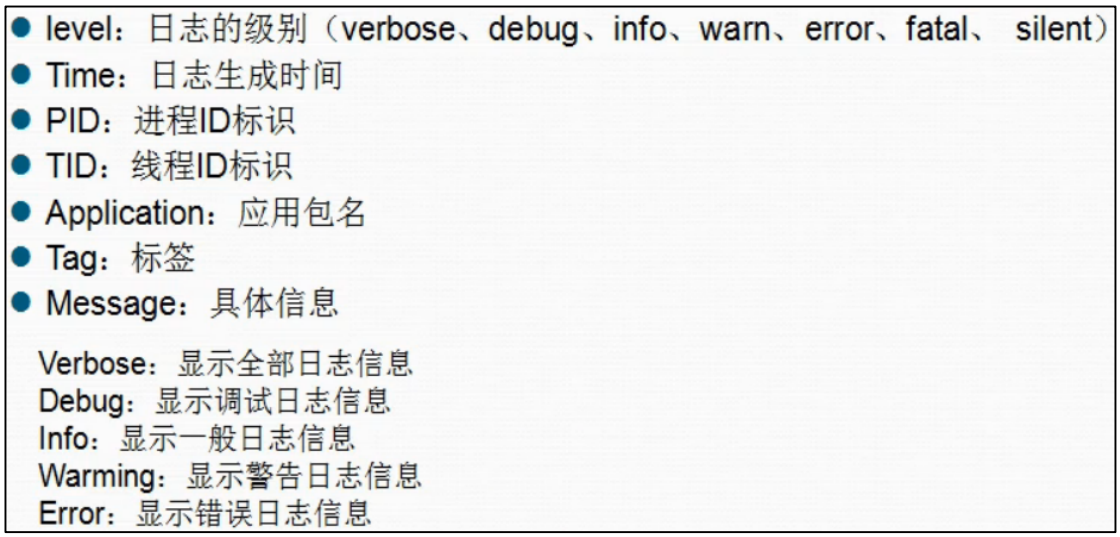
- URL：统一资源定位符（Uniform Resource Locator，缩写为URL）是对可以从互联网上得到的资源的位置和访问方法的一种简洁的表示，是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。[1]
- URN：统一资源名称 (Uniform Resource Name, URN)，它是URL的一种更新形式，不依赖于位置，并且有可能减少失效连接的个数。但是其流行还需假以时日，因为它需要更精密软件的支持。
- URI：URL(统一资源定位符)和URN(统一资源名称) 是URI的子集

URL 组成：

协议://域名或 ip 地址：端口号（默认 80）/资源具体路径

4.4 Logcat 简介





日志级别的优先级（由高到低）：

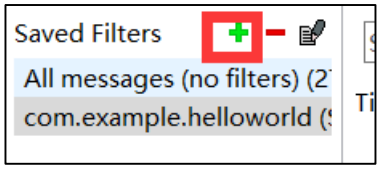
V、D、I、W、E、F、S

自己添加日志：

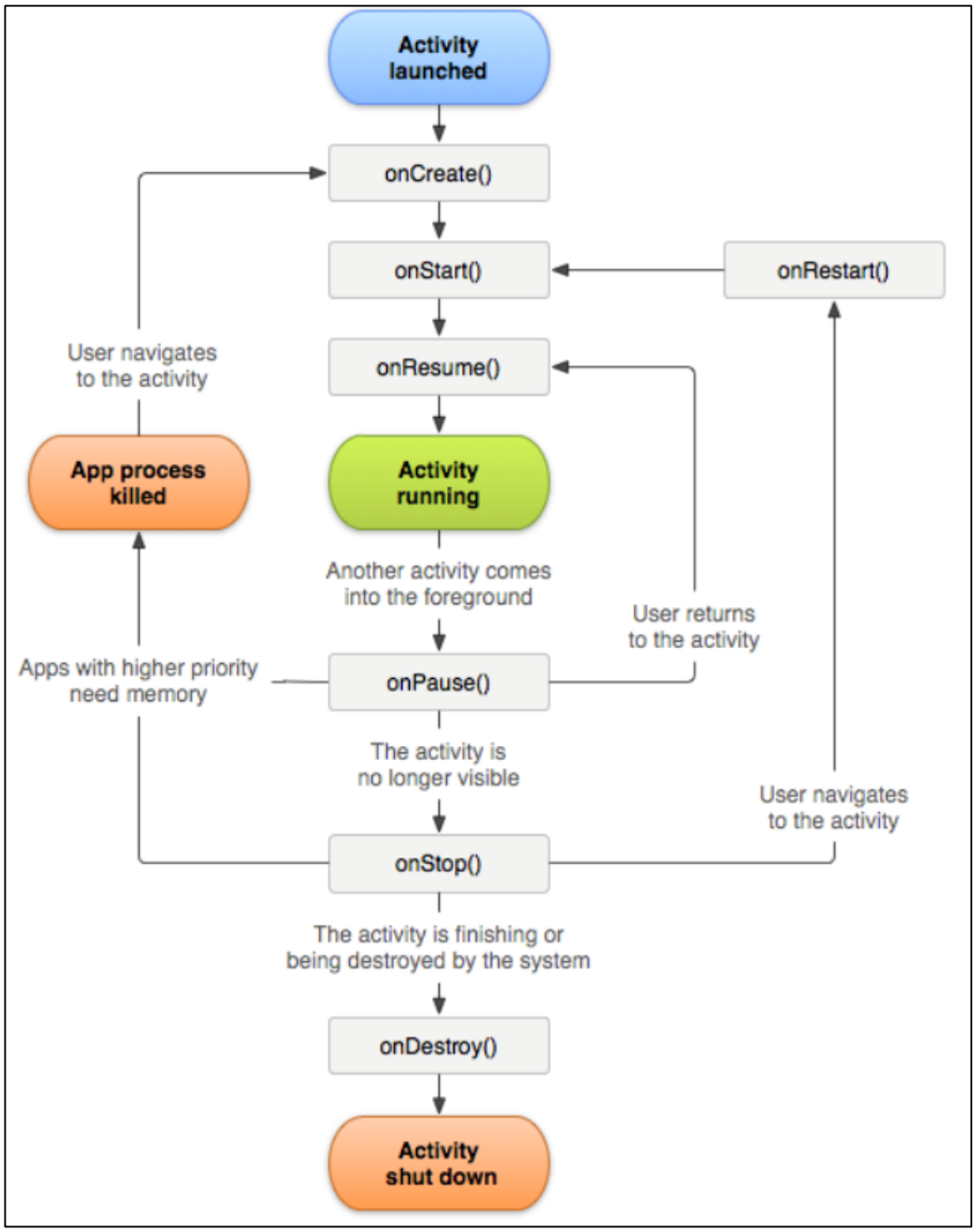
```
import android.util.Log;
```

```
Log.v(tag, msg);
```

添加日志过滤器：



4.5 Activity 生命周期



横竖屏切换生命周期

onPause()->onStop()->onDestroy()->onCreate()->onStart()->onResume

默认情况下，当“屏幕方向”或“键盘显示隐藏”变化时都会销毁当前的 Activity，创建新的 Activity，如果不希望重新创建 Activity 实

例，可按如下配置 Activity:

```
<activity
    android:name="com.example.helloworld.MainActivity"
    android:label="@string/app_name"
    android:configChanges="keyboardHidden|orientation|screenSize">
```

忽视键盘的显示隐藏和手机屏幕的旋转，执行 onConfigurationChanged()。

tips:

```
android:screenOrientation="landscape">   始终横屏
```

```
android:screenOrientation="portrait">   始终竖屏
```

4.6 案例-电话拨号器

首先在清单文件 AndroidMainfest.xml 中添加电话服务权限

```
android:versionName="1.0" >
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-sdk
```

采用隐式意图的方法调用系统自带的拨号 Activity

```
public class MainActivity extends Activity {
    private Button btn_call;
    private EditText editphonenum;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn_call = (Button) findViewById(R.id.btn_call);
        editphonenum = (EditText) findViewById(R.id.editphonenum);
        btn_call.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                //1.获取拨号号码
                String phonenum = editphonenum.getText().toString();
                //2.拨号
                Intent intent = new Intent();
                intent.setAction(Intent.ACTION_CALL);
                intent.addCategory(Intent.CATEGORY_DEFAULT);
                intent.setData(Uri.parse("tel:"+phonenum));
                startActivity(intent);
            }
        });
    }
```

其中的 Action,category,data 可以在安卓源码中查找

4.7 Activity 数据传递

2.通过静态变量传递数据

待接收的 Activity 中设置静态变量

```
7  public class OtherActivity extends Activity {
8      private TextView t;
9      public static MyObject myobject;
```

主界面赋值

```
    @Override
    public void onClick(View v) {
        Intent intent =new Intent();
        intent.setClass(MainActivity.this, OtherActivity.class);
        MyObject myObject = new MyObject();
        myObject.setName("张瑞");
    //  intent.putExtra("data", myObject);
        OtherActivity.myobject = myObject;
        startActivity(intent);
    }
```

数据传递之后，页面失效，需在 OtherActivity 中销毁对象

```
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        myobject = null;
    }
```

1.通过 Intent 传递数据

```
        @Override
        public void onClick(View v) {
            Intent intent =new Intent();
            intent.setClass(MainActivity.this, OtherActivity.class);
            intent.putExtra("data", "主页面数据");
            startActivity(intent);
        }
    });
}
```

在另一个 Activity 中接收数据

```
ublic class OtherActivity extends Activity {
    private TextView t;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.otheractivity);
        Bundle bundle = getIntent().getExtras();
        String data = bundle.getString("data");
        t = (TextView) findViewById(R.id.textView1);
        t.setText(data);
```

如果要传递一个对象的话，那么这个类需要实现 Serializable 接口（不推荐）

另一种方法（推荐），实现 Parcelable

```
public class MyObject implements Parcelable{
```

写数据

```
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
    }
```

读数据

```
    public static Parcelable.Creator<MyObject> CREATOR = new Creator<MyObject>() {

        @Override
        public MyObject[] newArray(int size) {
            // TODO Auto-generated method stub
            return null;
        }

        @Override
        public MyObject createFromParcel(Parcel source) {
            MyObject myObject = new MyObject();
            myObject.setName(source.readString());
            return myObject;
        }
    };
```

3. 通过剪切板传递数据

4. 通过全局变量传递数据

创建 Application 对象

```
public class MyApplication extends Application {
```

在 AndroidMainfest.xml 中配置 Application

```
<application
    android:allowBackup="true"
    android:name="MyApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
```

不配置默认 name 为 Application

```
        intent.setClass(MainActivity.this, OtherActivity.class);
        MyApplication application = (MyApplication) getApplication();
                application.setName("张瑞");
    //  MyObject myObject = new MyObject();
    //  myObject.setName("张瑞");
    //  intent.putExtra("data", myObject);
    //  OtherActivity.myobject = myObject;
        startActivity(intent);
```

```
    //  MyObject data = (MyObject) bundle.get("data");
        t = (TextView) findViewById(R.id.textView1);
    //  t.setText(data.getName());
        MyApplication application = (MyApplication) getApplication();
        t.setText(application.getName());
    }
```

4.8 从 Activity 中返回数据

采用 Intent 对象方式返回数据，需要使用 startActivityForResult 方法来启动 Activity

```
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, ActivityResult.class);
        startActivityForResult(intent, 0);
```

请求码

```
        @Override
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.putExtra("phonenum", tv.getText().toString());
            setResult(0,intent);  结果码
            finish();
        }                          销毁当前页面
    });
```

执行 onActivityResult 方法

```
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if(data!=null){
            et.setText(data.getExtras().getString("phonenum"));
        }
    }
    @Override
```

4.9 请求码与返回码

4.10 Activity 启动模式

在 AndroidMainfest.xml 文件中 Activity 标签的 android:launchMode 属性

1. standard

默认启动模式，每次激活 Activity 时都会创建 Activity 实例，并放入 Activity 栈中

例如：

有两个Activity名为B1,B2，B1为standard，B2为singleTop。若我打开的顺序为B1->B2->B2，则实际打开的顺序为B1->B2（后一次意图打开B2，实际只调用了前一个的onNewIntent方法若我打开的顺序为B1->B2->B1->B2，则实际打开的顺序与意图的一致，为B1->B2->B1->B2。

2. singleTop

可以有多个实例，但是不允许此 Activity 的多个实例叠加。即，如果此 Activity 有实例在栈顶的时候，启动这个 Activity，不会创建新的实例，而会调用 onNewIntent 方法。如果不在栈顶，则创建新的实例。

3. singleTask

只有一个实例，在同一个应用程序中启动他的时候，若不存在此Activity实例，则会在当前栈创建一个新的实例，若存在，则会把栈中在其之上的其它Activity实例销毁掉，并调用此实例的onNewIntent方法。如果是在别的应用程序中启动它，则会新建一个栈，并在该栈中启动这个Activity，然后我在这个activity实例中再打开新的Activity，这个新的Activity实例会在一个栈中。

例如：

应用程序中有三个Activity,C1,C2,C3，三个Activity可互相启动，其中C2为singleTask模式，那么，无论我在这个程序中如何点击启动，如：C1->C2->C3->C2->C3->C1-C2，C1,C3可能存在多个实例，但是C2只会存在一个，并且这三个Activity都在同一个task里面。并且启动singleTask模式的activity会把task中在其之上的其它Activity实例销毁掉。

操作	C1->C2	C1->C2->C3	C1->C2->C3->C2	C1->C2->C3->C2->C3->C1-C2
实际	C1->C2	C1->C2->C3	C1->C2	C1->C2

4.singleInstance

只有一个实例，并且这个实例独立运行在一个activity任务栈(task)中，这个task只有这个实例，不允许有别的Activity存在。

例如：

有三个Activity分别是D1,D2,D3，三个Activity可互相启动，其中D2为singleInstance模式。那么程序从D1开始运行，假设D1的taskId为200，那么从D1启动D2时，D2会新启动一个task，即D2与D1不在一个task中运行。假设D2的taskId为201，再从D2启动D3时，D3的taskId为200，也就是说它被压到了D1启动的task中。

若是在别的应用程序打开D2，假设Other的taskId为200，打开D2，D2会新建一个task运行，假设它的taskId为201，那么如果这时再从D2启动D1或者D3，则又会再创建一个task，因此，若操作步骤为other->D2->D1，这过程就涉及到了3个task了。

4.11 保存和恢复 Activity 的状态

保存状态

给文本框设置 Id

onSaveInstanceState 可以自动保存有 id 的控件

在方法中使用 outState.put*()方法可以保存变量。**

恢复状态

onRestoreInstanceState 方法中重新获取数据。

5. Android 视图

在 Android 系统中，可视化控件都是从 android.view.View 继承的。

有两种方式创建视图。

方式一：

使用 xml 方式来配置 view 的属性，然后装载这些 view

方式二：

完全使用 java 代码创建 view

5.1 Android 长度单位表示的三种方式

- Android表示单位长度的方式通常有三种表示方式。
- px：表示屏幕实际的象素。例如，320*480的屏幕在横向有320个象素，在纵向有480个象素。
- dp(dip): 是屏幕的物理尺寸。大小为1英寸的1/72。
- sp（与刻度无关的像素）：与dp类似，但是可以根据用户的字体大小首选项进行缩放。

5.2 Android 的 5 种布局

LinearLayout(线性布局)、RelativeLayout(相对布局)、TableLayout(表格布局)、 FrameLayout(框架布局)、AbsoluteLayout(绝对布局)等。

1.线性布局（LinearLayout）

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.layout.MainActivity"
     android:orientation="horizontal"        子节点横向排列
     >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</LinearLayout>
```

视图效果

控件显示方式：

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="right"
    />
```

```
android:orientation="horizontal"
>                          控制控件占据空间分配比例
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="7"
        android:layout_weight="1"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="8"
        android:layout_weight="1"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="9"
        android:layout_weight="1"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="/"
        android:layout_weight="1"
        />
```

案例：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.framelayout.MainActivity"
    android:orientation="vertical"
    >
    <FrameLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    >
        <ImageView
            android:layout_width="match_parent"
            l
            />

    </FrameLayout>
    <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="播放"
        android:layout_weight="1"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="暂停"
        android:layout_weight="1"
        />

    </LinearLayout>
</LinearLayout>
```

2.框架布局(FrameLayout)

- 框架布局是最简单的布局方式、所有添加到这个布局中的视图都是以层叠的方式显示。第一个添加到框架布局中的视图显示在最底层，最后一个被放在最顶层，上一层的视图会覆盖下一层的视图，因此框架布局类似堆栈布局。

图片控件

```
<ImageView
  android:src=""          路径
  android:layout_marginTop=""
  />                      距离容器边界
```

设置边界在 FrameLayout 中失效，需要结合 gravity 属性使用

```
android:gravity="right"
```

```
<ImageView
  android:src=""
  android:layout_marginTop=""
  android:visibility="invisible"   隐藏
  />
```

```
<ImageView
  android:src=""
  android:layout_marginTop=""
  android:visibility="gone"
  />
```

直接移除，在界面中不占据位置

```
<ImageView
  android:src=""
  android:layout_marginTop=""
  android:visibility="visible"
  />
```

显示

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
    case R.id.bt_pause:
        iv_play.setVisibility(View.VISIBLE);
        break;
    case R.id.bt_play:
        iv_play.setVisibility(View.INVISIBLE);
        break;
    }
}
```

设置控件可见

设置控件隐藏

3.表格布局(TableLayout)

- 在TableLayout布局中，一个列的宽度由该列中最宽的那个单元格指定，而表格的宽度是由父容器指定的。在TableLayout中，可以为列设置三种属性：
- Shrinkable：如果一个列被标识为Shrinkable，则该列的宽度可以进行收缩，以使表格能够适应其父容器的大小。
- Stretchable：如果一个列被标识为Stretchable，则该列的宽度可以进行拉伸，以使填满表格中的空闲空间。
- Collapsed：如果一个列被标识为Collapsed，则该列会被隐藏
- 注意：一个列可以同时具有Shrinkable属性和Stretchable属性，在这种情况下，该列的宽度将任意拉伸或收缩以适应父容器

<TableRow></TableRow>一行表格

4. 绝对布局(AbsoluteLayout)

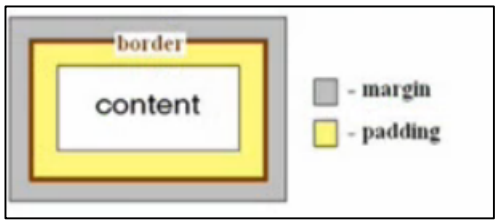
- 所谓绝对布局（AbsoluteLayout），是指屏幕中所有控件的摆放由开发人员通过设置控件的坐标来指定，控件容器不再负责管理其子控件的位置。由于子控件的位置和布局都是通过坐标来指定，因此AbsoluteLayout类中没有特殊的属性和方法。
- 可以通过android:layout_x和android:layout_y属性可以设置视图的横坐标和纵坐标的位置。

5. 相对布局（RelativeLayout）

RelativeLayout可以设置某一个视图相对于其他视图的位置，这些位置可以包括上下左右等

属性	说明
android:layout_below	在某元素的下方
android:layout_above	在某元素的的上方
android:layout_toLeftOf	在某元素的左边
android:layout_toRightOf	在某元素的右边

5.3 padding 和 margin 属性，设置监听的两张方法



Margin 为外边框，指改控件距离父控件的边距

Padding 为内边框，指该控件内部内容，如 Button 中文本距离控件的边距

设置监听可以直接控件设置监听，创建内部类，或者 Activity 实现监听接口，添加对应需要监听的控件

5.4 View 和 ViewGroup

- Android的UI界面都是由View和ViewGroup及其派生类组合而成的。
- View是所有UI组件的基类，而ViewGroup是容纳这些组件的容器，其本身也是从View类派生出来的.

View派生出的直接子类有：

AnalogClock,ImageView,KeyboardView,ProgressBar,SurfaceView,TextView,ViewGroup,ViewStub

ViewGroup派生出的直接子类有：

AbsoluteLayout,AdapterView,FragmentBreadCrumbs,FrameLayout,LinearLayout,RelativeLayout,SlidingDrawer

```
@Override
protected void onCreate(Bundle savedInstanceSta
    super.onCreate(savedInstanceState);
    View view = View.inflate(this, R.layout.acti
                创建一个View
    LayoutParams layoutParams = new LayoutParams
            LayoutParams.WRAP_CONTENT);
    TextView textView = new TextView(this);
    textView.setLayoutParams(layoutParams);
    textView.setBackgroundColor(Color.GREEN);
    textView.setText("你好啊，陌生人");

    ((RelativeLayout)view).addView(textView);
    setContentView(view);       添加到View中
}
```

5.5TextView 文本控件

- TextView主要是在android中实现文字说明等功能。
- 主要实现功能如下：
- 1、显示丰富的文本(URL、字体大小、颜色等)
- 在TextView中预定了一些类似HTML的标签，通过标签可以使TextView控件显得不同颜色、大小、字体的文字。
- <font>：设置颜色和字体
- <big>：设置大小号
- <small>：设置小号
- <i>\<b>：斜体、粗体
- <a>：链接地址
- <img>：插入图片

- 使用这些标签可以用Html.fromHtml方法将这些标签的字符串转换成Charsequenece对象，然后在TextView中进行设置。
- 如果想在显示的文本中将URL地址、邮箱地址、电话产生超链接的效果可以使用android:autoLink来设置。
- 该属性如下：

属性值	描述
None	不匹配任何链接(默认)
web	网址
email	邮箱
phone	电话号码
map	匹配映射网址
all	匹配所有链接

```
<TextView
    android:id="@+id/tv_one"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

<TextView
    android:id="@+id/tv_two"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:autoLink="all"
    />
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tv_one=(TextView) findViewById(R.id.tv_one);
    tv_two=(TextView) findViewById(R.id.tv_two);

    String text1="<font color='red'><i>你好啊，陌生人</i></for
    text1+="<a href='http://www.baidu.com'>百度</a><br/>";
    tv_one.setText(Html.fromHtml(text1));
    //产生连接效果
    tv_one.setMovementMethod(LinkMovementMethod.getInstance()

    String text2="我的网站：http://www.baidu.com";
    text2+="我的电话:1383424213";
    tv_two.setText(text2);
}
```

自动识别

1. TextView 图文混排

写 Html 代码

```
text_img = (TextView) findViewById(R.id.text_img);
String html = "<font>图片</font><img src='img'/>";
```

通过 Html.fromHtml 转化

Spanned android.text.Html.fromHtml(String source, ImageGetter imageGetter, TagHandler tagHandler)

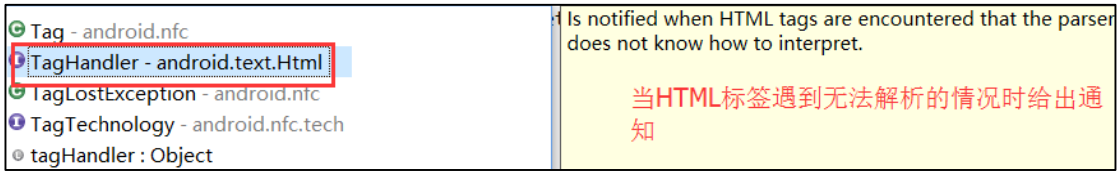
Returns displayable styled text from the provided HTML string. Any <img> tags in the HTML will use the specified ImageGetter to request a representation of the image (use null if you don't want this) and the specified TagHandler to handle unknown tags (specify null if you don't want this).

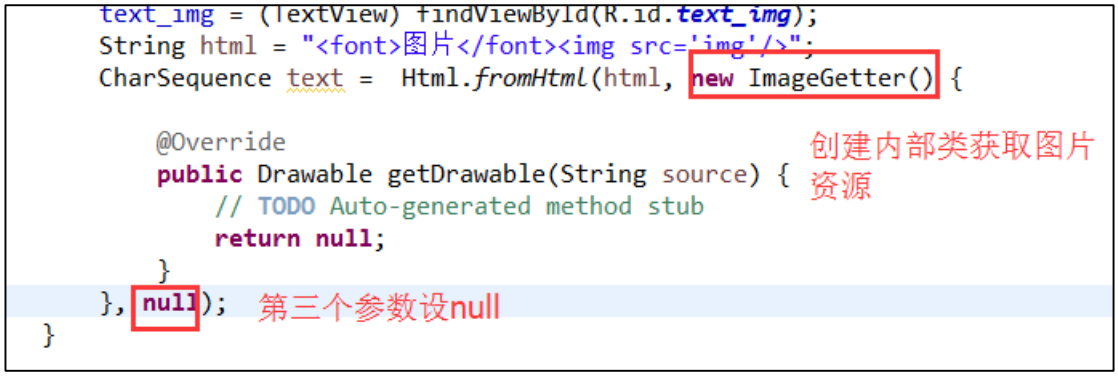
This uses TagSoup to handle real HTML, including all of the brokenness found in the wild.

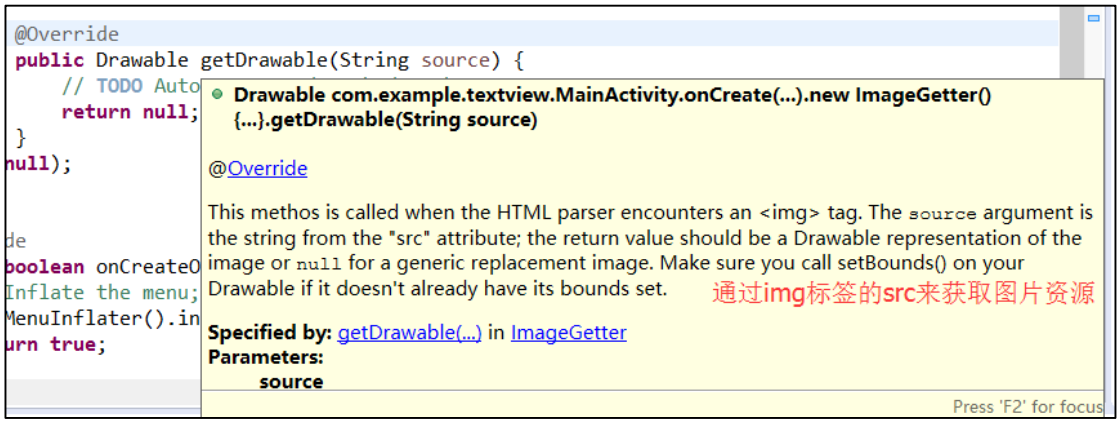
参数解释

Tip(通过 crtl+T 快捷键可以快速查看类结构)

source 传 HTML 代码







通过获取资源文件的方式获得图片，这里用的是硬编码，由于返回图片未设置大小边界，所以需对其进行边界设置

```
CharSequence text =  Html.fromHtml(html, new ImageGetter() {
    @Override
    public Drawable getDrawable(String source) {

        Drawable drawable = getResources().getDrawable(R.drawable.img);
        drawable.setBounds(0, 0, drawable.getIntrinsicWidth(), drawable.getIntrinsicHeight());
        return drawable;
    }
}, null);
text_img.setText(text);
```

我们需要通过反射的方式来从 source(src 中的值)获取图片的 ID

```
Field field = R.drawable.class.getField(source);
field.get(object)
```

反射机制获得类属性

Object java.lang.reflect.Field.get(Object object) throws IllegalAccessException, IllegalArgumentException

获得对象中属性的值，由于drawable中图片ID为静态，所以设置为null

Returns the value of the field in the specified object. This reproduces the effect of `object.fieldName`

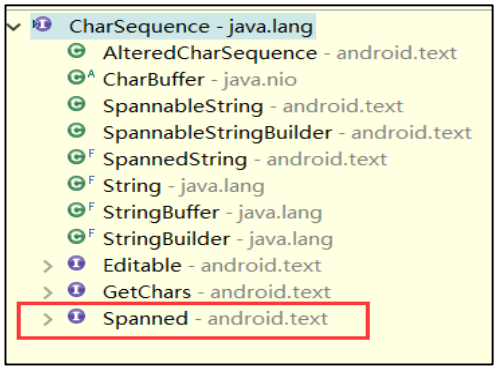
If the type of this field is a primitive type, the field value is automatically boxed.

If this field is static, the object argument is ignored. Otherwise, if the object is null, a NullPointerException is thrown. If the object is not an instance of the declaring class of the method, an IllegalArgumentException is thrown.

Press 'F2' for focus

```
CharSequence text =  Html.fromHtml(html, new ImageGetter() {
    @Override
    public Drawable getDrawable(String source) {
        int imgid  = 0;
        try {
            Field field = R.drawable.class.getField(source);
            imgid = Integer.parseInt(field.get(null).toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
        Drawable drawable = getResources().getDrawable(imgid);
        drawable.setBounds(0, 0, drawable.getIntrinsicWidth(), drawable.getIntrinsicHeight());
        return drawable;
    }
}, null);
text_img.setText(text);
}
```

返回值结构



2. SpannableString

TextView通常用来显示普通文本，但是有时候需要对其中某些文本进行样式、事件方面的设置。android系统通过**SpannableString**类来对指定文本进行相关处理，具体有以下功能：

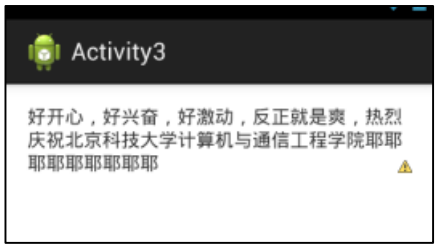
1、BackgroundColorSpan 背景色
2、ClickableSpan 文本可点击，有点击事件
3、ForegroundColorSpan文本颜色（前景色）
4、MaskFilterSpan 修饰效果，如模糊(BlurMaskFilter)、浮雕(EmbossMaskFilter)
5、RasterizerSpan 光栅效果
6、StrikethroughSpan删除线（中划线）
7、UnderlineSpan 下划线
8、AbsoluteSizeSpan 绝对大小（文本字体）
9、DynamicDrawableSpan设置图片，基于文本基线或底部对齐。
10、ImageSpan 图片
11、RelativeSizeSpan 相对大小（文本字体）
12、ScaleXSpan 基于x轴缩放
13、StyleSpan 字体样式：粗体、斜体等
14、SubscriptSpan 下标（数学公式会用到）
15、SuperscriptSpan 上标（数学公式会用到）
16、TextAppearanceSpan文本外貌（包括字体、大小、样式和颜色）
17、TypefaceSpan 文本字体
18、URLSpan 文本超链接

通过 SpannableString 对象的 setSpan 方法来使用上述效果

```
public class Spannable_Activity extends Activity {
    private TextView text;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_spannable );          设置可点击文本
        text = (TextView) findViewById(R.id.text);
        SpannableString string = new SpannableString("点击我打开activity");
        string.setSpan(new ClickableSpan() {

            @Override
            public void onClick(View widget) {
                Intent intent = new Intent(Spannable_Activity.this,MainActivity.class);
                startActivity(intent);
            }
        }, 0, string.length(), Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
                                                    不包含前半部分后后半部分
        text.setText(string);
        text.setMovementMethod(LinkMovementMethod.getInstance());
    }                                                设置为点击可响应
```

3. TextView 实现跑马灯

文本框文字较多时初始效果



设置单行显示，超出部分会以小数点显示

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:text="好开心，好兴奋，好激动，反正就是爽，热烈庆
```

设置小数点位置

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:ellipsize="start"
    android:text="好开心，好兴奋，好激动，反正就是爽，热烈
```

设置滚动

```
android:ellipsize="marquee"
```

为使文本滚动需要设置可获取焦点权限（默认没有焦点）

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:ellipsize="marquee"
    android:focusable="true"
    android:text="好开心，好兴奋，好激动，反正就是爽，热烈庆祝北京
```

设置在触摸模式下获取焦点（失去焦点后会不滚动）

```xml
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:ellipsize="marquee"
    android:focusable="true"
    android:focusableInTouchMode="true"
    android:text="好开心，好兴奋，好激动，反正就是爽，热烈庆祝北
```

为了使文本框能时刻滚动（其他控件获取焦点时也能滚动）

可以继承 TextView，重写 isFocused()方法，始终返回 true

创建控件时使用<自定义类全名></自定义类全名>标签

案例-音乐播放器

```
private MediaPlayer mp;
private Button b;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    b =(Button) findViewById(R.id.scan_music);
    mp = MediaPlayer.create(this,R.raw.start);      //背景音乐
    mp.start();
```

```
mp.start();
b.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        mp.reset();
        mp = MediaPlayer.create(MainActivity.this,R.raw.tan);      //背景音乐
        try {
            mp.prepare();
        } catch (IllegalStateException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        sb.setMax(mp.getDuration());
            mp.seekTo(music_position);
            mp.start();
            handler = new Handler();
            updateThread = new Runnable() {
                public void run() {
                    sb.setProgress(mp.getCurrentPosition());
                    handler.postDelayed(updateThread, 100);
                }
        };
        handler.post(updateThread);
            b.setVisibility(View.INVISIBLE);
            b2.setVisibility(View.VISIBLE);
    }
});
```

设置拖动条长度为音乐长度

创建线程控制拖动条同步

```
@Override
protected void onDestroy() {
    mp.release();
    if(handler!=null){
        handler.removeCallbacks(updateThread);
    }
    super.onDestroy();
}
```
释放资源

5.6EditText 控件

具有全部 TextView 属性，继承自 TextView

1. EditText 实现输入图像表情

利用 SpannableString 中的 ImageSpan

2. 特定字符限制

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="46dp"
    android:ems="10"
    android:digits="abc"
    >
```
限制只能输入abc

用这个属性可以限制只能输入指定内容

```
android:inputType="number"
```

限定输入类型

设置校验提示

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    b = (Button) findViewById(R.id.button1);
    e =(EditText) findViewById(R.id.editText1);
    b.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            String str = e.getText().toString();
            if(str!=null&&!"".equals(str)){
            Intent intent = new Intent(MainActivity.this,Other.class);
            startActivity(intent);
            }else{
                e.setError("不能为空");
            }
        }
    });
}
```

5.7AutoCompleteTextView 自动完成输入文本的控件

AutoCompleteTextView常用属性	
android:completionHint	设置出现在下拉菜单中的提示标题
android:completionThreshold	设置用户至少输入多少个字符才会显示提示
android:dropDownHorizontalOffset	下拉菜单于文本框之间的水平偏移。默认与文本框左对齐
android:dropDownHeight	下拉菜单的高度
android:dropDownWidth	下拉菜单的宽度
android:singleLine	单行显示
android:dropDownVerticalOffset	垂直偏移量

```
<AutoCompleteTextView
    android:id="@+id/actv"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

</AutoCompleteTextView>
```

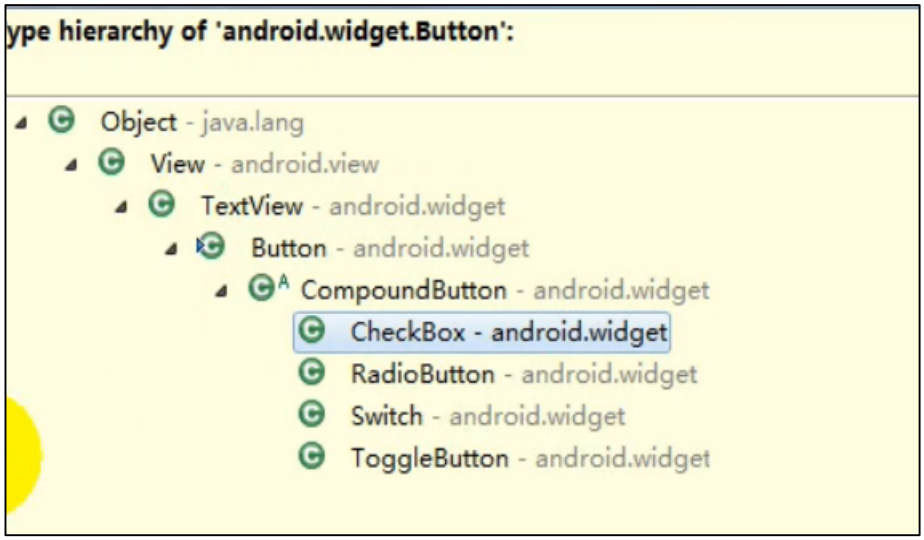
```
private AutoCompleteTextView actv;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    actv = (AutoCompleteTextView) findViewById(R.id.actv);
    String[] data = {"张瑞最帅","张瑞超级帅"};
    ArrayAdapter<String>  adapter= new ArrayAdapter<String>(MainActivity.this,android.R.layout.simple_dropdown_item_1line, data);
    actv.setAdapter(adapter);
}
```

创建适配器，将输入框与数据联系起来

注意，是android.R,这里表示下拉菜单格式

给自动文本框加适配器

5.8Button 系列按钮控件



1.Button 图文混排

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableTop="@drawable/star"
    android:text="按钮一"
    />
```

顶部有一张图片

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableTop="@drawable/star"
    android:drawablePadding="30dip"
    android:text="按钮二"
    />
```

图文间距

利用 Button 的父类 TextView 之 SpannableString

2.RadioButton（单选按钮）

需要在 RadioGroup 使用才能形成单选效果

在 java 代码中获得 RadioGroup 对象后，RadioButton 以数组形式存

放在 RadioGroup 中

3. ToggleButton（状态翻转按钮）

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textOff="横向排列"
    android:textOn="纵向排列" />
```

初始处于开启状态

设置 ToggleButton 状态转换监听

```
setOnCheckedChangeListener
```

5.9 CheckBox 复选框空间

```
private List<CheckBox> checkBoxList = new ArrayList<CheckBox
private LinearLayout ll_checkbox_list;
private Button btn_ok;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_demo1);
    ll_checkbox_list = (LinearLayout) findViewById(R.id.ll_ch
    btn_ok=(Button) findViewById(R.id.btn_ok);
    String[] strArr = {"你是学生吗？", "你是否喜欢android", "
    for(String str:strArr){
        CheckBox checkBox = (CheckBox) View.inflate(this,
                R.layout.ui_checkbox, null);
        checkBox.setText(str);
        ll_checkbox_list.addView(checkBox);
```

添加一个复选框到布局中

将 CheckBox 放到一个 List 中形成一个组合。

5.10 SeekBar 拖动条控件

```
<SeekBar
    android:id="@+id/seekBar1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="30dp"
    android:progress="10"
    />
```

进步默认值

```
<SeekBar
    android:id="@+id/seekBar1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="76dp"
    android:progress="10"
    android:secondaryProgress="20"
    />
```

第二层进度，类似视频缓冲进度

```
sb1.setOnSeekBarChangeListener(this);
sb2.setOnSeekBarChangeListener(this);
```

```
/**
 * 进度值改变回调方法
 */
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
            boolean fromUser) {
        if(seekBar.getId()==R.id.seekBar1){
            tv1.setText("进度条一进度："+progress);
        }
    }
/**
 * 点击开始拖动回调方法
 */
    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
        // TODO Auto-generated method stub

    }
/**
 * 放开停止拖动回调方法
 */
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        // TODO Auto-generated method stub
```

5.11 ImageView

设置背景色

```
android:background="@android:color/darker_gray"
```

引用图片

```
android:src="@drawable/background"
```

```
android:scaleType="fitCenter"
```

设置图片显示方式

显示网络中图片

使用 http 协议操作

```
URL url=new URL("http://img0.bdstatic.com/img/image
//拿到HTTP连接对象
HttpURLConnection connection=(HttpURLConnection)url
//设置连接超时
connection.setConnectTimeout(5000);
```

```
//能够获取服务器的响应内容
connection.setDoInput(true);
//HTTP请求方式
connection.setRequestMethod("GET");
//获取响应状态码
int code=connection.getResponseCode();
if(code==200){//表示获取成功
    //拿到输入流，用于读取响应的内容
    InputStream is=connection.getInputStream();
    byte[] data=new byte[1024];
    int len;
    while ((len=is.read(data))!=-1) {
        byteArrayOut.write(data,0,len);
    }
    return byteArrayOut.toByteArray();
```

使用 BitmapFactory 可以将图片字节数组转化为图片

需添加联网权限：

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

5.12 DataPicker 日期控件 TimePicker 时间控件

```
private DatePicker dp_date;
private TimePicker tp_time;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    dp_date=(DatePicker) findViewById(R.id.dp_date);
    tp_time = (TimePicker) findViewById(R.id.tp_time);
    dp_date.init(2014,8,24,this);
    //设置时间以24小时制显示
    tp_time.setIs24HourView(true);
}
```

初始化日期

5.13 AnalogClock 石英表控件 DigitalClock 数字表控件

设置自定义表盘

```
android:dial="@drawable/biaopan"
```

TimePickerDialog 时间对话控件

5.14 ProgressBar 进度条控件

用户不可控制，用来在传输过程中显示进度

用户浏览网页的时候，中间肯定有个传输过程，所以用进度条让用户耐心等待。再比如在下载应用中，它也会有进度条显示下载进度。

ProgressDialog 是继承自 Android.app.ProgressDialog 所设计的互动对话窗口，在应用时，必须新建 ProgressDialog 对象，在运行时会弹出“对话框”作为提醒，此时应用程序后台失去焦点，直到进程结束后，才会将控制权交给应用程序，如果在 Activity 当中不希望后台失焦，又希望提示 User 有某后台程序正处于忙碌阶段，此时，ProgressBar 就会派上用场了。

自己去自定义一些进度条。比如说定义为竖向的。或者是弧形的，等等

四种 progressbar 的风格：

android:attr/progressBarStyle、

android:attr/progressBarStyleHorizontal

android:progressBarStyleLarge、

android:progressBarStyleSmall

5.15 ScrollView 垂直滚动条

- ScrollView控件只是支持垂直滚动，而且在ScrollView中只能包含一个控件，通常是在< ScrollView >标签中定义了一个<LinearLayout>
- 标签并且在<LinearLayout>标签中android:orientation属性值设置为vertical，然后在<LinearLayout>标签中放置多个控件，如果<LinearLayout>标签中的控件所占用的总高度超出屏幕的高度，就会在屏幕的右侧出现一个滚动条。

设置滚动条不显示

```
android:scrollbars="none"
```

5.15 HorizontalScrollView 水平滚动控件

- HorizontalScrollView控件只是支持水平滚动，而且它只能包含一个控件，通常是在< HorizontalScrollView >标签中定义了一个<LinearLayout>
- 标签并且在<LinearLayout>标签中android:orientation属性值设置为horizontal，然后在<LinearLayout>标签中放置多个控件，如果<LinearLayout>标签中的控件所占用的总宽度超出屏幕的宽度，就会出现滚动效果

5.16 Spinner 下拉列表

通过数据与控件适配器 Adapter 实现

5.17 ImageSwitcher 图片切换控件

通过图片数组实现切换

Activity 实现 ViewFactory 接口

通过 ViewFactory 获取 ImageView

```
//设置被切换View对象
imageSwitcher.setFactory(this);
```

重载 makeView()方法

```
@Override
public View makeView() {
    return new ImageView(this);
}
```

```
imageSwitcher.setImageResource(images[index]);
```

设置图片切换效果

```
imageSwitcher.setInAnimation(
        R.anim.right_in));
imageSwitcher.setOutAnimation
```

5.18 include 与 viewstub

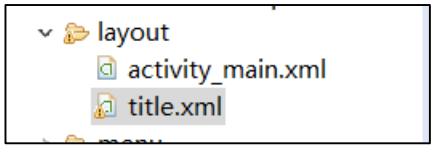
- ViewtSub的功能和<include>的功能类似，也是实现引用另外一个布局。但是唯一不同的是ViewStub并不会马上装载引用布局文件，只有在调用了ViewStub.inflate或ViewStub.setVisibility(View.VISIBILE)方法ViewStub才会装载引用的控件。

注意事项：

1. ViewStub只能Inflate一次，之后ViewStub对象会被置为空。按句话说，某个被ViewStub指定的布局被Inflate后，就不会够再通过ViewStub来控制它了。

2. 某些布局属性要加在ViewStub而不是实际的布局上面，才会起作用，比如上面用的android:layout_margin*系列属性，如果加在TextView上面，则不会起作用，需要放在它的ViewStub上面才会起作用。而ViewStub的属性在inflate()后会都传给相应的布局。

静态加载，利用<include>进行布局的复用



```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.includeandviewstub.MainActivity"
    android:orientation="vertical"
    >
    <ViewStub
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
       android:layout="@layout/title"
       android:id="@+id/vs"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

动态加载 ViewStub

需要用到时才会加载

```xml
    <ViewStub
        android:layout="@layout/title"
        />
```

MainActivity.java | activity_main.xml | *title.xml

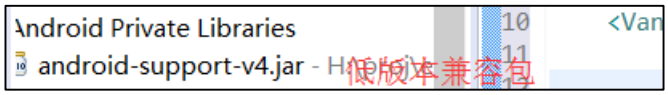
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="左边" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="101dp"
        android:layout_toRightOf="@+id/textView1"
        android:text="标题" />
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="右边" />
</RelativeLayout>
```

```java
ublic class MainActivity extends Activity {
    private ViewStub vs;
    private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        vs = (ViewStub) findViewById(R.id.vs);
        btn = (Button) findViewById(R.id.btn);
        btn.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                RelativeLayout view = (RelativeLayout) vs.inflate();    //动态加载布局
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
```

这里注意 Button 点击事件只能触发一次，ViewStub 调用 inflate 一次后就会被置空

5.19 ViewPager 多页面滑动效果

- Android的左右滑动在实际编程经常能用到，比如查看多张图片，左右切换tab页。
- 自Android 3.0之后的SDK中提供了android-support-v4包用以实现版本兼容，让老版本系统下的应用通过加入jar包实现扩展，其中有一个可以实现左右滑动的类ViewPager




```
<android.support.v4.view.ViewPager
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|center"
    >
    <ImageView android:src="@drawable/point_select"
        android:id="@+id/point1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="15dip"
        />
   <ImageView android:src="@drawable/point"
      android:id="@+id/point2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:padding="15dip"
        />
    <ImageView android:src="@drawable/point"
              android:id="@+id/point3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:padding="15dip"
        />
   <ImageView android:src="@drawable/point"
             android:id="@+id/point4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:padding="15dip"
        />
</LinearLayout>
```

实现引导页小点

```java
public class MainActivity extends Activity implements OnPageChangeListener{
    private ViewPager vp;
    private ImageView iv[] = new ImageView[4];
    private List<View> view_list = new ArrayList<View>();      //滚动图片集合
    private LinearLayout ly;
    private int[] images={R.drawable.guide1,R.drawable.guide2,R.drawable.guide1,R.drawable.guide2};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
        initData();
        PagerAdapter pa = new PagerAdapter() {

            @Override
            public boolean isViewFromObject(View arg0, Object arg1) {
                // TODO Auto-generated method stub
                return arg0==arg1;
            }
            //显示的页数
            @Override
            public int getCount() {
                return images.length;
            }
            @Override
            public void destroyItem(ViewGroup container, int position,
                    Object object) {
                container.removeView(view_list.get(position));
            }
            /**
             * containerAdapter作用的对象
             */
            @Override
            public Object instantiateItem(ViewGroup container, int position) {
                container.addView(view_list.get(position));
                return view_list.get(position);
            }
        };
        vp.setAdapter(pa);
        vp.addOnPageChangeListener(this);
    }
```

```java
    public void initView(){
        vp = (ViewPager) findViewById(R.id.vp);
        ly = (LinearLayout)findViewById(R.id.point_item);
        for (int i = 0; i < ly.getChildCount(); i++) {
            iv[i] = (ImageView) ly.getChildAt(i);
        }
    }

    public void initData(){
        for (int i = 0; i < images.length; i++) {
            ImageView iv = new ImageView(this);
            iv.setImageResource(images[i]);
            iv.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,LayoutParams.MATCH_PARENT));
            view_list.add(iv);
        }
    }
```

适配器连接数据与 ViewPager

PageAdapter 必须重写的四个函数：

•boolean isViewFromObject(View arg0, Object arg1)

•int getCount() 返回要滑动的 VIew 的个数

•void destroyItem(ViewGroup container, int position,Object object)

从当前 container 中删除指定位置（position）的 view

•Object instantiateItem(ViewGroup container, int position)

将当前视图加到 container 中，返回当前 view

```
PagerAdapter pa = new PagerAdapter() {

    @Override
    public boolean isViewFromObject(View arg0, Object arg1) {
        // TODO Auto-generated method stub
        return arg0==arg1;
    }
    //显示的页数
    @Override
    public int getCount() {
        return images.length;
    }
    @Override
    public void destroyItem(ViewGroup container, int position,
            Object object) {
        container.removeView(view_list.get(position));
    }
    /**
     * containerAdapter作用的对象
     */
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        container.addView(view_list.get(position));
        return view_list.get(position);
    }
};
vp.setAdapter(pa);
```

设置视图切换事件

```
/**
 * 新页面被调用
 */
    @Override
    public void onPageSelected(int postion) {
        for (ImageView i : iv) {
            i.setImageResource(R.drawable.point);
        }
        iv[postion].setImageResource(R.drawable.point_select);
    }
}

```

5.20 ListView


```
class MainActivity extends Activity {
vate ListView lv;
erride
tected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 lv = (ListView) findViewById(R.id.lv);
 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_expandable_list_item_1,getDat
 lv.setAdapter(adapter);
```

每一项列表结构，复杂的自定义

设置列表项被点击事件

```
lv.setOnItemClickListener(new OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view,
            int position, long id) {
        // TODO Auto-generated method stub

    }


});
```

自定义 Adapter 实现复杂的列表项（继承 BaseAdapter）

设置列表项布局

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    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:text="姓名" />

    <TextView
        android:id="@+id/nameref"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="TextView" />

</RelativeLayout>
```

```
private ListView lv;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    lv = (ListView) findViewById(R.id.lv);
    MyAdapter adapter = new MyAdapter(new String[]{"as","cd","ds"}, this);
    lv.setAdapter(adapter);
}
```

自定义适配器

设置三个属性，接受数据，上下文对象，

```
    public MyAdapter(String[] data, Context con) {
        super();
        this.data = data;
        this.con = con;
        layoutInflater = LayoutInflater.from(con);
    }
    //列表item数
```

```
//列表item数
@Override
public int getCount() {
    // TODO Auto-generated method stub
    return data.length;
}
//根据position获取对应item数据
@Override
public Object getItem(int position) {
    // TODO Auto-generated method stub
    return data[position];
    }
//根据position获取对应Item的ID
@Override
public long getItemId(int position) {
    return position;
}
//创建列表item视图
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    if(convertView==null){
        convertView = layoutInflater.inflate(R.layout.item_list, null);
    }
    TextView name = (TextView) convertView.findViewById(R.id.name);
    TextView nameref = (TextView) convertView.findViewById(R.id.nameref);
    name.setText("姓名：");
    nameref.setText(data[position]);
    return convertView;
}
```

获取列表格式

为列表控件初始化

重要：

```
    this.con = con;
    layoutInflater = LayoutInflater.from(con);
}
```

从视图对象获取一个 layoutInflater 对象

用这个对象的 Inflate 方法动态加载列表

convertView 缓存的 view

5.21 GridView

- GridView控件用于显示一个网格图像， GridView主要是用在一些相册的布局显示图片。
- GridView采用的是二维表的方式显示单元格，就需要设置二维表的行和列。设置GridView的列可以使用<GridView>标签的columnWidth属性。也可以使用GridView类的setColumnWidth方法来设置列数，
- GridView中的单元格会根据列数自动拆行显示，因此不需要设置GridView的行数，但是需要设置android:numColumns属性。否则GridView只会显示一行。

用法与 ListView 相似

5.22 特殊功能

1.分割线

```
<View
    android:layout_width="match_parent"
    android:layout_height="1dp"
    android:layout_above="@id/include1"
    android:layout_alignParentLeft="true"
    android:background="@android:color/holo_blue_bright" />
```

2. 去掉菜单栏

整个应用都去掉

```
<application
    android:allowBackup="true"
    android:theme="@android:style/Theme.NoTitleBar"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
```

如果只是某个 Activity，则在<Activity>标签中添加

此时背景会变黑色，若不想变黑

采用更改 style 文件方法

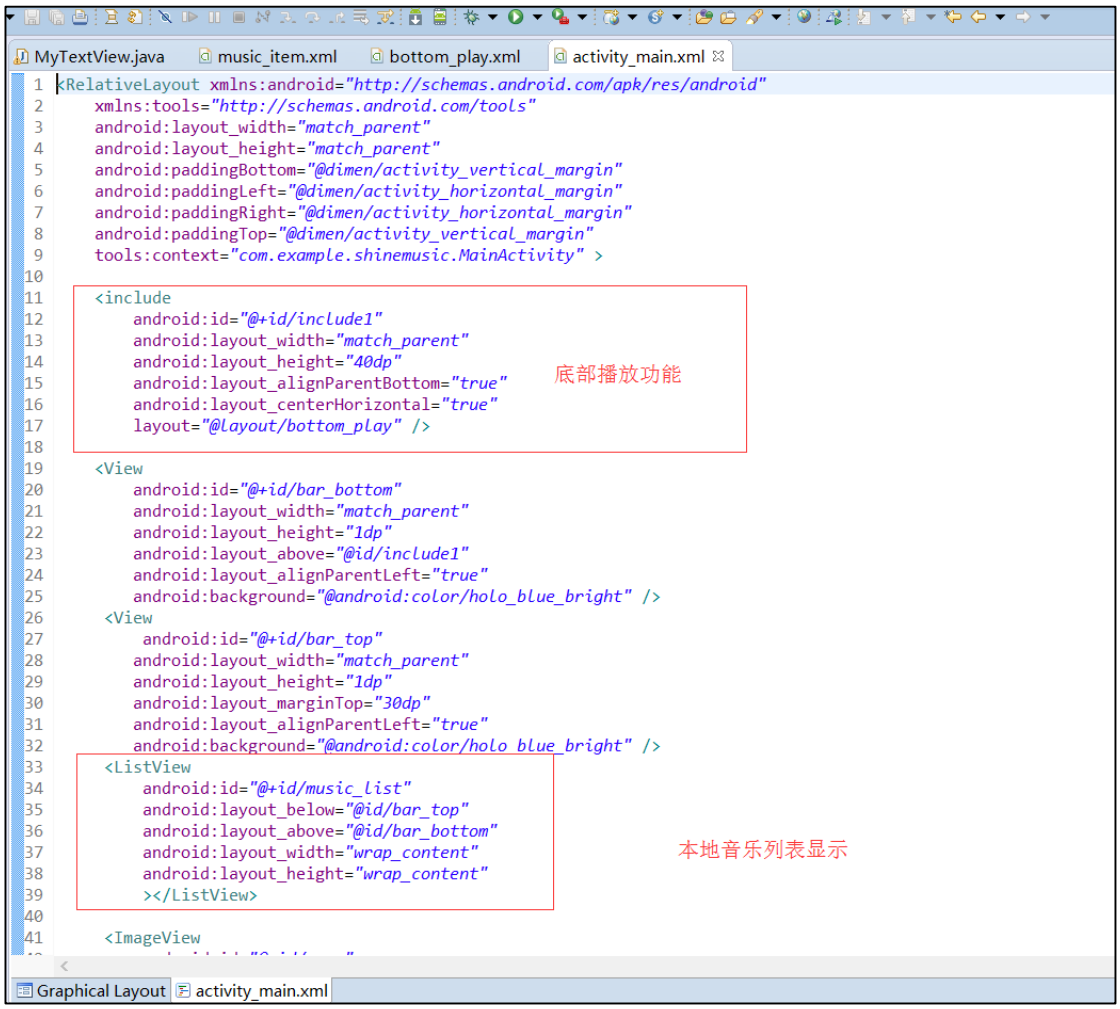
```
<style name="AppTheme" parent="AppBaseTheme">
    <!-- All customizations that are NOT specific to a particula
    <item name="android:windowNoTitle">true</item>
</style>
```

5.23 综合案例——扫描本地音乐播放

1. 自定义 TextView,重写 isFocus()方法，使控件一直处于获得焦点的状态，实现字体跑马灯效果

```
package com.example.shinemusic;

import android.content.Context;

public class MyTextView extends TextView{

    public MyTextView(Context context) {
        super(context);
    }
    public MyTextView(Context context, AttributeSet attrs) {
         super(context, attrs);
         // TODO Auto-generated constructor stub
         }
    @Override
    public boolean isFocused() {
        // TODO Auto-generated method stub
        return true;    //隐性让TextView一直获得焦点
    }
}
```

2. 创建视图，ListView 展现音乐列表，底部使用 Include 展现当前正在播放



4. 创建 Music_song 对象，存储 Music 信息

```
public class MusicSong {
    private String music_singer;
    private long music_duration;
    private String music_name;
    private String music_path;
    public String getMusic_singer() {
        return music_singer;
    }
```

5. 自定义扫描本地音乐

```
public class Music_scan {
    private static ArrayList<MusicSong> music_list = new ArrayList<MusicSong>();
    private Music_scan(){}
    public ArrayList<MusicSong> getMusic_list() {
        return music_list;
    }
    public void setMusic_list(ArrayList<MusicSong> music_list) {
        Music_scan.music_list = music_list;
    }
    public static ArrayList<MusicSong> scan_music(Context mContext){
        Cursor c=mContext.getContentResolver().query(Media.EXTERNAL_CONTENT_URI, null, null, null, MediaStore.Audio.AudioColumns.IS_MUSIC);
        if(c!=null&&c.moveToFirst()){
            do{
                MusicSong ms = new MusicSong();
                int music_name_index = c.getColumnIndex(Media.DISPLAY_NAME);
                int music_duration_index = c.getColumnIndex(Media.DURATION);
                int music_singer_index = c.getColumnIndex(Media.ARTIST);
                int music_path_index = c.getColumnIndex(Media.DATA);
                ms.setMusic_duration(c.getInt(music_duration_index));
                String music_name = c.getString(music_name_index) ;
                ms.setMusic_name(sortName(music_name));
                ms.setMusic_singer(c.getString(music_singer_index));
                ms.setMusic_path(c.getString(music_path_index));
                music_list.add(ms);
            }while(c.moveToNext());  //还存在下一个文件
        }
        return music_list;
    }
    public static String sortName(String music_name){
        int start = music_name.indexOf("-");
        if(start!=-1){
            music_name = music_name.substring(start+1);
        }
        int end = music_name.indexOf(".");
        if(end!=-1){
            music_name = music_name.substring(0,end);
        }
        int end2 = music_name.indexOf("[");
        if(end2!=-1){
            music_name = music_name.substring(0,end2);
        }
        return music_name.trim();//去掉首尾空字符串
    }
```

创建游标，搜索本地音乐

获取媒体文件信息素在索引

设置音乐对象属性

对音乐文件名进行处理

去除前缀

去掉后缀名

去掉说明信息

6. 自定义适配器，连接 ListView 控件和数据源

```
public class MyAdapter extends BaseAdapter {
    private Context con;
    private ArrayList<MusicSong> music_list = new ArrayList<MusicSong>();
    private LayoutInflater layoutInflater;

    public MyAdapter(Context con, ArrayList<MusicSong> music_list) {
        super();
        this.con = con;
        this.music_list = music_list;
        layoutInflater = LayoutInflater.from(con);
    }

    public int getCount() {

    public Object getItem(int position) {

    public long getItemId(int position) {

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if(convertView==null){
            convertView = layoutInflater.inflate(R.layout.music_item, null);
        }
        MyTextView name =(MyTextView) convertView.findViewById(R.id.music_item_name);
        MyTextView duration = (MyTextView) convertView.findViewById(R.id.music_item_duration);
        MyTextView singer = (MyTextView) convertView.findViewById(R.id.music_item_singer);
        name.setText(music_list.get(position).getMusic_name());
        duration.setText(sortTime(music_list.get(position).getMusic_duration()));
        singer.setText(music_list.get(position).getMusic_singer());
        return convertView;
    }
    public  static String sortTime(long l){
        int second = (int) (l/1000);
        int minute = second/60;
        second = second-minute*60;
        String str = new String("时长："+minute+":"+second);
        return str;
    }
}
```

6. 数据存储

6.1 读写本地文件

手机内部存储：

安卓只有一个根目录，使用正斜杠代表

apk 写到自己包名文件夹下/data/data/apk 包名（真机测试非 rootDDMS 中无法查看文件夹）

JavaSE IO 流知识

SD 卡

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

配置写入外部设备文件的权限

6.2 判断是否存在 SD 卡

获取手机外部 SD 卡路径

```
Environment.getExternalStorageDirectory();
```

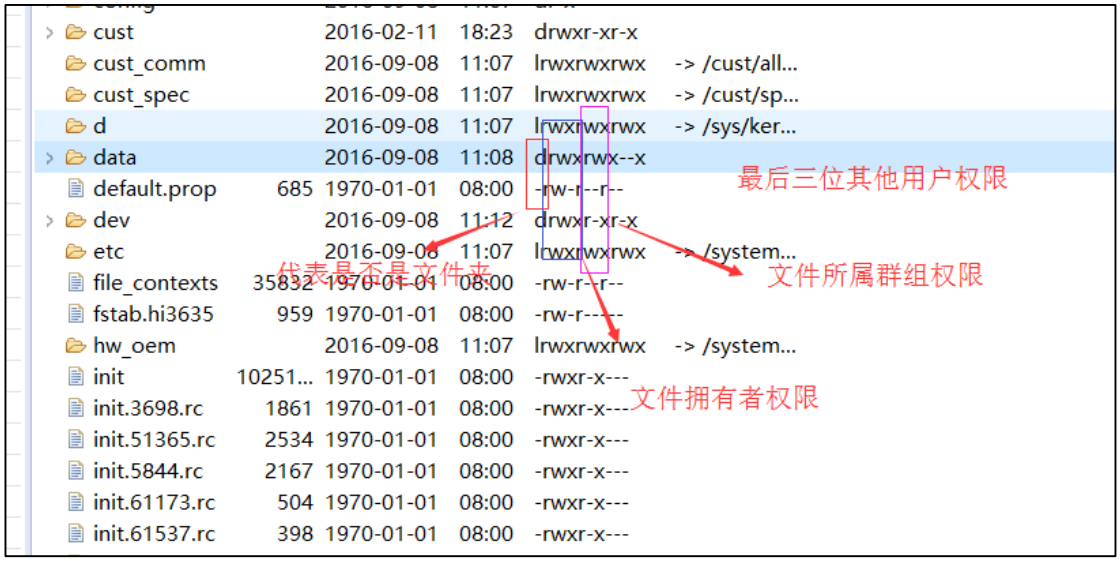
6.2 修改文件权限

Linux文件的三种身份和四种权限，三种身份分别为：
u：文件的拥有者
g：文件所属的群组
o：其他用户

对于每个身份，又有四种权限，分别为：
r：读取文件的权限（read）
w：写入文件的权限（write）
x：执行的权限（execute）
s：特殊权限

语法格式：chmod abc file

其中a,b,c各为一个数字，分别表示User、Group、及Other的权限。
r=4，w=2，x=1



cmd

adb shell 进入终端

chmod abc file

abc 分别代表 User,Group,Other 三者 rex 权限和

a) SharedPreference 存储及读取配置

```
lic class MainActivity extends Activity {
 private SharedPreferences sharedPreferences;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     //name配置文件的名字 mode配置文件的模式
     sharedPreferences = getSharedPreferences("config", MODE_PRIVATE);
 }
```

只能被当前APK读写

```
sharedPreferences = getSharedPreferences("config", MODE_WORLD_READABLE+MODE_WORLD_WRITEABLE);
```

可以被其他文件读写

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //name配置文件的名字 mode配置文件的模式
    sharedPreferences = getSharedPreferences("config", MODE_WORLD_READABLE+MODE_WORLD_WRITEABLE);
    //得到编辑器
    Editor edit = sharedPreferences.edit();
    edit.putString("lauguage", "中文");
    //提交内存中的配置信息到本地
    edit.commit();
```

案例——实现记住密码功能

7. 单元测试

Junit 框架

在 AndroidMainFest.xml 中

<application>标签中加入

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <uses-library android:name="android.test.runner" />
</application>
```

在<manifest>标签中加入

```
<instrumentation
    android:name="android.test.InstrumentationTestRunner"
    android:label="Tests for My App"
    android:targetPackage="cn.itcast.action" />
```

targetPackage 和程序包名相同

类继承 AndroidTestCase

8. Android 开发之 SQLite

在事务处理方面，SQLite 通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据，但只有一个可以写入数据。在事务处理方面，SQLite 通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据，但只有一个可以写入数据。

数据类型

SQLite 支持 NULL、INTEGER、REAL、TEXT 和 BLOB 数据类型，分别代表空值、整型值、浮点值、字符串文本、二进制对象。

smallint 16 位元的整数。

interger 32 位元的整数。

decimal(p,s) p 精确值和 s 大小的十进位整数，精确值 p 是指全部有几个数(digits)大小值，s 是指小数点後有几位数。如果没有特别指定，则系统会设为 p=5; s=0 。

float 32 位元的实数。

double　64 位元的实数。

char(n)　n 长度的字串，n 不能超过 254。

varchar(n) 长度不固定且其最大长度为 n 的字串，n 不能超过 4000。

graphic(n) 和 char(n) 一样，不过其单位是两个字元 double-bytes，n 不能超过 127。这个形态是为了支援两个字元长度的字体，例如中文字。

vargraphic(n) 可变长度且其最大长度为 n 的双字元字串，n 不能超过 2000

date　包含了 年份、月份、日期。

time　包含了 小时、分钟、秒。

timestamp 包含了 年、月、日、时、分、秒、千分之一秒。

1. 建数据库

```
D:\>sqlite3 test.db
SQLite version 3.7.7.1 2011-06-28 17:39:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------
0    main             D:\test.db

sqlite>
```

2. 创建表

```
sqlite> CREATE TABLE person (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), age SMALLINT);
sqlite> .tables
person
sqlite> .schema person
CREATE TABLE person (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), age SMALLINT);
sqlite>
```

在我们创建表之后，可以用“.tables”命令去查看已有的表，用“.schema”命令去查看表的结构，如果后面没有表名做参数，则将会输出所有表的建表语句。

3. 插入数据

从.sql文件导入数据：

```
[sql]
01. sqlite> .read test.sql
02. sqlite> SELECT * FROM person;
03. 1|john|30
04. 2|david|35
05. 3|henry|40
06. sqlite>
```

```
sqlite> INSERT INTO person VALUES (NULL, 'john', 30);
sqlite> SELECT * FROM person;
1|john|30
```

4. 分析数据库使用状态

```
D:\>sqlite3_analyzer test.db
```

5. 备份数据库

如果数据库正在使用中，则应从命令行界面使用 .dump 命令。这样可以创建一个包含必要命令和数据的文件，从而重新创建数据库。.dump 命令也可以用于备份数据库表。

```sql
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE person (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), age SMALLINT);
INSERT INTO "person" VALUES(1,'john',30);
INSERT INTO "person" VALUES(2,'david',35);
INSERT INTO "person" VALUES(3,'henry',40);
DELETE FROM sqlite_sequence;
INSERT INTO "sqlite_sequence" VALUES('person',3);
COMMIT;
sqlite> .output dump.sql
sqlite> .dump
sqlite>
```

我们可以指定输出的目标为一个文件，然后再使用命令时，输出信息就会写入指定的文件，如果想恢复为标准输出，可以这样设定：

```sql
sqlite> .output stdout
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE person (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), age SMALLINT);
INSERT INTO "person" VALUES(1,'john',30);
INSERT INTO "person" VALUES(2,'david',35);
INSERT INTO "person" VALUES(3,'henry',40);
DELETE FROM sqlite_sequence;
INSERT INTO "sqlite_sequence" VALUES('person',3);
COMMIT;
sqlite>
```

如果数据库没有处于使用状态，则可以直接将数据库文件复制到安全位置。

最后，我们可以使用“.quit”或“.exit”退出 SQLite。

6. 在 Java 中使用 SQLite

需要下载 SQLite 相关驱动 https://bitbucket.org/xerial/sqlite-jdbc/downloads

7. Android_SQLite

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //1.打开或创建数据库
    SQLiteDatabase db = openOrCreateDatabase("test.db", Context.MODE_PRIVATE, null);
    db.execSQL("DROP TABLE IF EXISTS person");
    //2.创建person表
    db.execSQL("CREATE TABLE person(p_id INTEGER PRIMARY KEY AUTOINCREMENT,name VARCHAR(20),age SMALLINT)");
    //3.插入数据
    //db.execSQL("INSERT INTO person VALUES(null,?,?)", new Object[]{"张瑞",22});
    db.close();
}
```

```
////ContentValues以键值对的形式存放数据
ContentValues values = new ContentValues();
values.put("name", "zhangrui");
values.put("age", 22);
db.insert("person", null, values);

//更新数据
values= new ContentValues();
values.put("age", 23);
db.update("person", values, "name=?", new String[]{"zhangrui"});
db.close();   //关闭数据库
```

在实际开发中，为了能够更好的管理和维护数据库，我们会封装一个继承自 SQLiteOpenHelper 类的数据库操作类，然后以这个类为基础，再封装我们的业务逻辑方法。

SQLiteOpenHelper

SQliteOpenHelper 是一个抽象类，来管理数据库的创建和版本的管理。要使用它必须实现它的

onCreate(SQLiteDatabase) //数据库第一次被创建时 onCreate 会被调用

onUpgrade(SQLiteDatabase, int, int) //如果 DATABASE_VERSION 值被更改,系统发现现有数据库版本不同,即会调用 onUpgrade

```
ublic class DBHelper extends SQLiteOpenHelper {
        private static final String name = "test.db";
        public static final int version = 1;
        public DBHelper(Context context) {
        super(context, name, null, version);
        //context上下文对象，一般在Acitvity中给出
        //name数据库名
        //factory用来创建游标，或者默认为Null
        //version版本号，用来根据版本号调用onUpdata方法更新数据库
    }

        //数据库第一次被创建时onCreate会被调用
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE IF NOT EXISTS person"+"(_id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), ag
    }

     //如果DATABASE_VERSION值被改为2,系统发现现有数据库版本不同,即会调用onUpgrade
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
          db.execSQL("ALTER TABLE person ADD COLUMN other Varchar(50)");
    }
```

使用 DBHelper 封装业务方法,使用 db 对象进行数据库操作

```
public class DBManager {
    private DBHelper helper;
    private SQLiteDatabase db;

    public DBManager(Context context) {
        helper = new DBHelper(context);
        //因为getWritableDatabase内部调用了mContext.openOrCreateDatabase(mName, 0, mFactory);
        //所以要确保context已初始化,我们可以把实例化DBManager的步骤放在Activity的onCreate里
        db = helper.getWritableDatabase();
    }
```

在 Activiy 中重写 Destory 方法，关闭数据库连接

9. ContentProvider 组件

不同应用之间数据访问的桥梁

ContentProvider 负责组织应用程序的数据；向其他应用程序提供数据；ContentResolver 则负责获取 ContentProvider 提供的数据

Android 为我们提供了 ContentProvider 来实现数据的共享，一个程序如果想让别的程序可以操作自己的数据，就定义自己的 ContentProvider，然后在 AndroidManifest.xml 中注册，其他 application 可以通过获取 ContentResolver 通过 Uri 来操作上一程序

的数据。

```
public class MyContentProvider extends ContentProvider {
```

在 AndroidMainfest.xml 中配置

```
<provider
    android:name="com.example.contentpro.MyContentProvider"
    android:authorities="com.zhangrui"
    ></provider>
```

```
//内容解析者
ContentResolver resolver = getContentResolver();
```

```
resolver.query(uri, projection, selection, selectionArgs, sortOrder)
```

对应于

```
import android.content.ContentProvider;

public class MyContentProvider extends ContentProvider {

    //创建时调用
    @Override
    public boolean onCreate() {
        Log.d("gg","onCreate");
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getType(Uri uri) {
```

调用者 ContentResolver 通过一个 uri 来找到相应的 ContentProvider 来进行实际操作。

1. uri 形如：scheme://authorities/path/id

根据 scheme 不同调用不程序来处理，常用的：content, android_resource, file, http 等

authorities 是 provider 定义的在 AndroidManifest.xml 中定义

path 操作的表的名字

id 如果 URI 中包含表示需要获取的记录的 ID；则就返回该 id 对应的数据，如果没有 ID，就表示返回全部

"content://com.android.calendar/calendars/#"

#表示数据 id（#代表任意数字）

"content://com.android.calendar/calendars/*"

*来匹配任意文本

如电话记录：

```
1.  public static final Uri CONTENT_URI = Uri.parse("content://call_log/calls");
```

UriMatcher：用于匹配 Uri

把需要匹配 Uri 路径全部给注册上

常量 UriMatcher.NO_MATCH 表示不匹配任何路径的返回码(-1)。

添加需要匹配 uri，如果匹配就会返回匹配码 //如果 match()方法匹配 content://com.android.calendar/calendars/23 路径，返回匹配码为 2

uriMatcher.addURI(“content://com.android.calendar”, “calendars/#”, 2);

注册完需要匹配的 Uri 后，就可以使用 uriMatcher.match(uri)方法对输入的 Uri 进行匹配，如果匹 配就返回匹配码

ContentObserver

```
observer=new MyObserver(new Handler());
//注册一个内容观察者（观察指定数据）
Uri uri=Uri.parse("content://sms");
getContentResolver().registerContentObserver(uri,true
```

```
public class MyObserver extends ContentObserver {

    public MyObserver(Handler handler) {
        super(handler);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onChange(boolean selfChange) {
        // TODO Auto-generated method stub
        super.onChange(selfChange);
    }

}
```

数据发生变化时回调此函数

10.进程与线程

进程：当一个程序第一次启动时，Android 会启动一个 LINUX 进程和一个主线程。默认情况下，所有该程序的组件都在该进程中运行。（一个应用可能存在不止一个进程）

可以通过 android:process 控制组件在哪个进程中运行

android 会根据进程中运行的组件的类别及组件的状态来判断进程的重要性，当系统内存不足时，Android 会根据进程优先级杀死相对不重要的进程

重要性由低到高：

活动进程->可见进程->启动服务进程->后台进程->空进程

1. 活动进程
a. 进程中包含处于前台的正与用户交互的activity;
b. 进程中包含与前台activity绑定的service;
c. 进程中包含调用了startForeground()方法的service;
d. 进程中包含正在执行onCreate(), onStart(), 或onDestroy()方法的service;
e. 进程中包含正在执行onReceive()方法的BroadcastReceiver.
系统中前台进程的数量很少, 前台进程几乎不会被杀死. 只有当内存低到无法保证所有的前台进程同时运行时才会选择杀死某个前台进程.
2. 可视进程
a. 进程中包含未处于前台但仍然可见的activity(调用了activity的onPause()方法, 但没有调用onStop()方法). 典型的情况是运行activity时弹出对话框, 此时的activity虽然不是前台activity, 但其仍然可见.
b. 进程中包含与可见activity绑定的service.
可视进程不会被系统杀死, 除非为了保证前台进程的运行而不得已为之.
3. 服务进程. 进程中包含已启动的service.
4. 后台进程. 进程中包含不可见的activity(onStop()方法调用后的activity). 后台进程不会直接影响用户体验, 为了保证前台进程/可视进程/服务进程的运行, 系统随时都有可能杀死一个后台进程. 一个正确的实现了生命周期方法的activity处于后台时被系统杀死, 可以在用户重新启动它时恢复之前的运行状态.
5. 空进程. 不包含任何处于活动状态进程是一个空进的程. 系统经常杀死空进程, 这不会造成任何影响. 空进程存在的唯一理由是为了缓存一些启动数据, 以便下次可以更快的启动.

1. Handler 消息机制

子线程不能更新 view，也就是 view 不能由自己创建的线程来更新。

```
//自定义子线程
class MyThread extends Thread{
    @Override
    public void run() {
        //伪代码来体现
        try {
            Thread.sleep(6000);
            Log.d("bh","访问到网络了");
            String str="我是网络数据";
            tv_txt.setText(str);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

解决方案：

```
private TextView tv_txt;
private Handler handler=new Handler(){
    //处理消息（被主线程执行）
    public void handleMessage(Message msg) {
        String str=(String) msg.obj;
        tv_txt.setText(str);
    };
};
```

```
//自定义子线程
class MyThread extends Thread{
    @Override
    public void run() {
        //伪代码来体现
        try {
            Thread.sleep(6000);
            Log.d("bh","访问到网络了");
            String str="我是网络数据";
            //创建message对象
            Message msg=new Message();
            msg.obj=str;
            //发送一个消息
            handler.sendMessage(msg);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

安卓提供了 Handler 和 Looper 来满足线程间通信

```
//判断当前函数是否被主线程调用的方式
boolean result=Looper.getMainLooper()==Looper.myLoope
Log.d("bh",result+"");
```

主线程有 Looper 对象，子线程默认没有

2. 幽灵线程

主线程已经关闭，但是子线程的 run 方法没有执行完

在子线程中判断主线程是否结束

主线程中设置标志

```
@Override
protected void onDestroy() {
    super.onDestroy();
    flag=false;
}
```

handler.post（new Runnable(){}）调用线程的 run 方法，在主线程中执行，可以用在子线程对 UI 进行更新

handler.removeCallbacks()结束线程

handler.postDeploy()方法可用在 run 方法实现周期性调用 run 方法。

3. AsyncTask(封装了线程池)

```
import android.os.AsyncTask;

public class MyAsyncTask extends AsyncTask<Params, process, Result> {
    /**
     * 被子线程执行，处理耗时操作
     */
    @Override
    protected Result doInBackground(Params... params) {
```

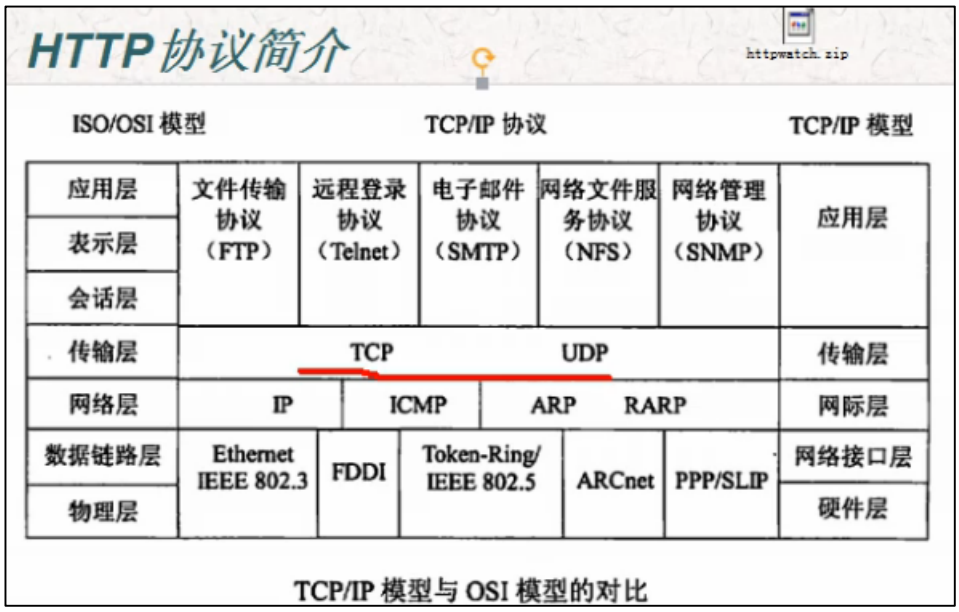
```
MyAsyncTask task=new  MyAsyncTask();
//执行
task.execute(params);
```

```
    @Override
    protected Result doInBackground(Params... params) {
        // TODO Auto-generated method stub
        publishProgress(values);
        return Result;
    }
```

推送进度

```
protected transient void onProgressUpdate(process[] values) {
    //在publishProgress被调用时执行，更新doInBackground进度
};
```

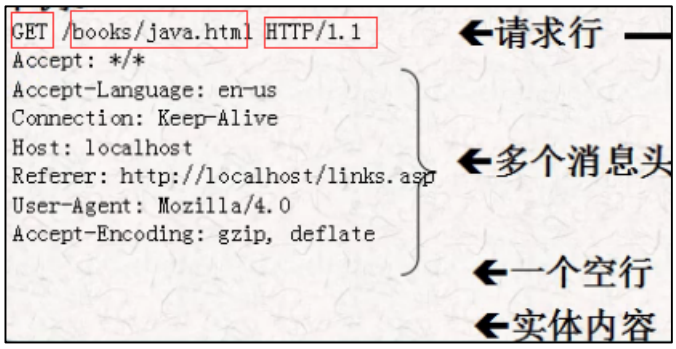
11. Http 协议



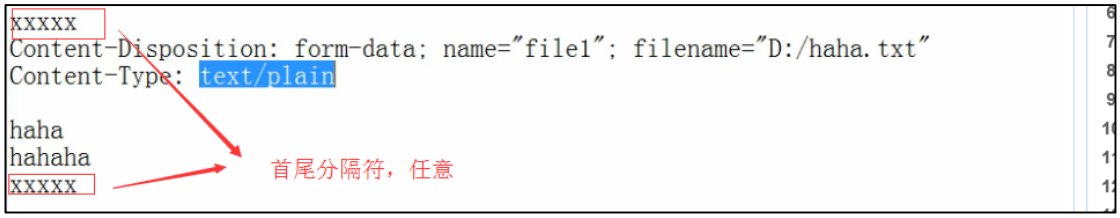
TCP/IP 模型与 OSI 模型的对比

http 请求包含一个请求行、若干消息头、以及实体内容，消息头和实

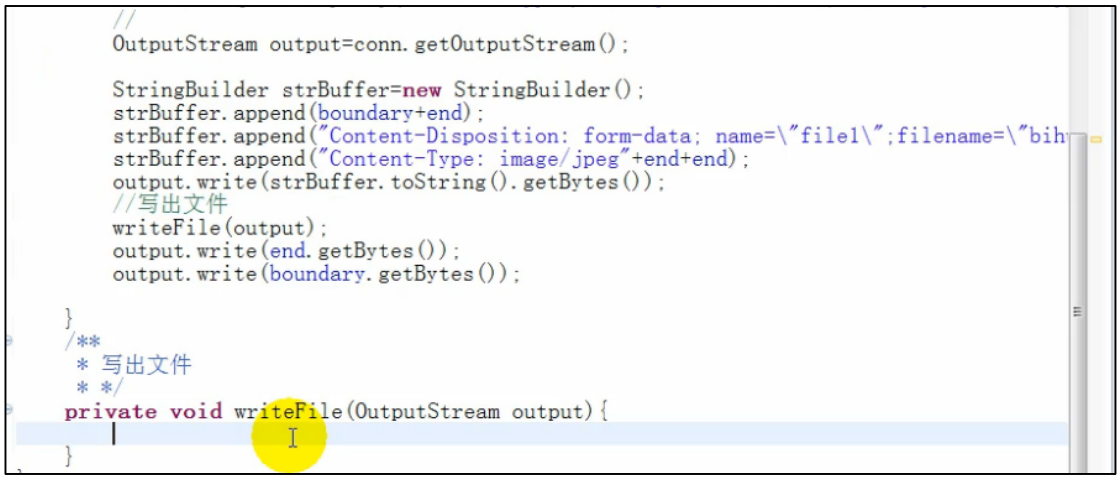
体内容是可选的，消息头和实体内容之间要用空行隔开



上传文件



boundary=文件中分隔符（/r/n 换行，分隔符最前面要加--）



```
        //
        OutputStream output=conn.getOutputStream();

        StringBuilder strBuffer=new StringBuilder();
        strBuffer.append(boundary+end);
        strBuffer.append("Content-Disposition: form-data; name=\"file1\";filename=\"bih
        strBuffer.append("Content-Type: image/jpeg"+end+end);
        output.write(strBuffer.toString().getBytes());
        //写出文件
        writeFile(output);
        output.write(end.getBytes());
        output.write(boundary.getBytes());

    }
    /**
     * 写出文件
     * */
    private void writeFile(OutputStream output){

    }
```

```java
private void writeFile(OutputStream output) throws IOException{
    FileInputStream input=new FileInputStream("/sdcard/bihu.jpg");
    //写出数据
    int len=0;
    byte[] buffer=new byte[1024];
    while((len=input.read(buffer))!=-1){
        output.write(buffer,0,len);
        output.flush();
    }
}
```

12. Json

语法规则：

数据由逗隔开，数据由键值对构成，键必须是 String

例如{"name":"许仙","age":18,"gender":true}

json中键值对（"key":"value"）中值的类型可以是下面数据类型中的任意一种：

1. null
2. 数字（整数或浮点数）
3. 逻辑值（true 或 false）
4. 字符串（在双引号中）
5. 数组（在方括号中）
7. 对象（在花括号中）

13. BroadcastReceiver 和 Service

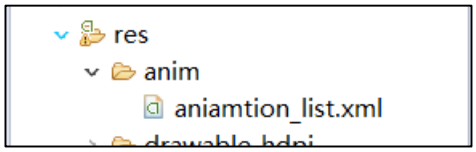
14. 安卓动画

1. 帧动画（通过 AnimationDrawable 加载 Drawable 动画实现）

1.1 使用 xml 实现动画（推荐）

在 XML 文件中<animation-list>元素为根节点，<item>节点定义了每一帧，表示一个 drawable 资源的帧和帧间隔。

在 res 目录下 anim 文件夹存放帧动画 xml;



activity_main.xml | MainActivity.java | aniamtion_list.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot = "false"
    >
 <!--
 oneshot代表是否只播放一次
 duration表示一幅图片展示时间
  -->
  <item android:drawable="@drawable/a1" android:duration="150"></item>
  <item android:drawable="@drawable/a2" android:duration="150"></item>
  <item android:drawable="@drawable/a3" android:duration="150"></item>
  <item android:drawable="@drawable/a4" android:duration="150"></item>

</animation-list>
```

activity_main.xml | MainActivity.java

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.myanimation.MainActivity" >

    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="21dp"
        android:text="动画开始" />

    <ImageView
        android:id="@+id/iv_animation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btn"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="33dp"
        android:src="@drawable/a1" />
```

播放动画

```
public class MainActivity extends Activity {
    private Button btn;
    private ImageView iv;
    private AnimationDrawable animationDrawable;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = (Button) findViewById(R.id.btn);
        iv = (ImageView) findViewById(R.id.iv_animation);
        btn.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                iv.setImageResource(R.anim.aniamtion_list);
                animationDrawable = (AnimationDrawable) iv.getDrawable();
                animationDrawable.start();
            }
        });
    }
```

AnimationDrawable本身就是一个Drawable

2. 使用程序控制，动态加载动画

xml 不填充 item

```
    iv.setImageResource(R.anim.aniamtion_list);
    animationDrawable = (AnimationDrawable) iv.getDrawable();
    Drawable frame= getResources().getDrawable(R.drawable.a1);
    animationDrawable.addFrame(frame, 1000);
```

动态加入一帧图片

2. 补间动画（View Animations）

View Animation 分为 4 大类：

AlphaAnimation,RotateAnimation,ScaleAnimation,TranslateAni

mation，分别对应透明度，旋转，大小，位移四种变化。

View Animation 的效果由四个因素决定：1）初始状态；2）结束状态；3）持续时间；4）Interpolator（内插器）

定义一个 View Animation，只要定义以上四个因素，中间的过程怎么变化则有系统自动计算出来。

Interpolator 是决定动画进行过程的速度变化。比如：你将一个按钮从屏幕左侧运动到屏幕右侧。可以让它一直加速前进，或者先减速然后减速。

2.1 代码控制实现

首先是页面布局

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.viewanimationdemo.MainActivity" >
    <TextView
        android:id="@+id/translation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="50dp"
        android:text="这个会移动" />
    <TextView
        android:id="@+id/rotate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
          android:layout_marginBottom="50dp"
          android:layout_below="@+id/translation"
        android:text="这个会旋转" />
    <TextView
```

```
        android:id="@+id/scale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:layout_marginBottom="50dp"
         android:layout_below="@+id/rotate"
        android:text="这个会变大" />
    <TextView
        android:id="@+id/alpha"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/scale"
         android:layout_marginBottom="150dp"
        android:text="这个会变暗"
        />
    <Button
        android:id="@+id/fire"
        android:layout_width="match_parent"
         android:layout_height="50dp"
         android:layout_below="@id/alpha"
        android:text="开始变化" />
    <RelativeLayout/>
```

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

        @Override
        public void onClick(View v) {
            //1.确定起始状态和结束状态
            TranslateAnimation traAni = new TranslateAnimation(0, 400, 0, 40);//横向移400，纵向移40
            RotateAnimation rotAni = new RotateAnimation(0, 45);   //顺时针旋转45度
            ScaleAnimation scaAni = new ScaleAnimation(0, 3, 0,2);//横向三倍，纵向两倍
            AlphaAnimation alpAni = new AlphaAnimation(1, 0); //从不透明变为全透明
            //2.确定持续时间
            traAni.setDuration(2000);
            rotAni.setDuration(2000);
            scaAni.setDuration(2000);
            alpAni.setDuration(2000);
            //2. 确定插值变化
            traAni.setInterpolator(new AccelerateDecelerateInterpolator());
            //3.启动动画
            translation.startAnimation(traAni);
            rotate.startAnimation(rotAni);
            scale.startAnimation(scaAni);
            alpha.startAnimation(alpAni);
        }
```

2.2 xml 实现

View Animation 的 xml 只能有一个根节点

可以是<set>,<alpha>,<scale>,<translate>,<rotate>,而其中，<set>结点可以包含子节点，即可以包含

<set>,<alpha>,<scale>,<translate>,<rotate>

标签属性介绍

android:interpolator 和 android:shareInterpolator，前者代表你所使用的 interpolator，可以是系统自带，也可以是自定义。而后者代表，是否将该 Interpolator 共享给子节点。

android:pivotX 和 android:pivotY,我们知道，pivot 的意思是轴心的意思，所以这两个属性定义的是此次动画变化的轴心位置，默认是左上角，当我们把它们两者都赋值为 50%，则变化轴心在中心。

andoird:fillAfter 通过设置 fillAfter 为 true，则动画将保持结束的状态。（View Animation 的动画效果是绘制出来的，并非该组件真正移动了）

startOffset：该属性定义动画推迟多久开始，通过这个属性的设置，我们可以设计一些前后按序发生的动画，当然，除了最后一个发生的动画，其他动画得设置 fillAfter 为 true.

```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"
   android:shareInterpolator="true">
    <alpha
       android:fromAlpha="1"
        android:toAlpha="0"
        android:duration="2000"/>
    <scale
        android:fromXScale="0"
        android:toXScale="3"
        android:fromYScale="0"
        android:toYScale="0.5"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="2000"/>
    <rotate
        android:fromDegrees="0"
        android:toDegrees="90"
        android:pivotX="0.5"
        android:pivotY="0.5"
        android:duration="2000"/>

</set>
```

```
Animation ani =AnimationUtils.loadAnimation(MainActivity.this, R.anim.view_animation);
xml.startAnimation(ani);
```

3. 属性动画（Property Animation）

Property Animation 翻译为属性动画，从 Android3.0 开始引入，相比与 View Animation，官方更推荐开发者使用 Property Animation。以下先讲解 View Animation 和 Property Animation 的区别：

1，View Animation 只能对界面上可见的组件进行修改，而 Property Animation 除此之外还能修改一些不是界面上可见的组件，比如修改一个 float 类型的值。

2，View Animation 通过动画效果改变一个组件时，其实只是在屏幕上另一个地方绘制，而组件的响应位置还是没改变，比如一个在屏幕左侧的按钮，通过 View Animation 移动到右侧，虽然在屏幕上看到按钮到了右侧了，但是你还是需要点击左侧原来的位置，按钮才会响

应。

3，Proerty Animation 能修改很多 View Animation 不能修改的 View 控件的属性，比如背景颜色。

Property Animation 的用法，包含了 ValueAnimator，ObjectAnimator 和 AnimatorSet

3.1 ValueAnimator

```
public class MainActivity extends Activity {
    private TextView tv;
    private ValueAnimator animator;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.tv);
        animator = ValueAnimator.ofFloat(0,1,2);
        animator.setDuration(10000);
        animator.start();
        animator.addUpdateListener(new AnimatorUpdateListener() {

            @Override
            public void onAnimationUpdate(ValueAnimator animation) {
                tv.setText(animator.getAnimatedValue().toString());
            }
        });
    }

    @Override
```

从0到1到2变化

数值变化监听器

默认更新时间是 10ms

3.2 ObjectAnimator

它是 ValueAnimator 的子类

ValueAnimator 需要我们监听 onAnimationUpdate 来将变化的值应用于某个 UI 属性。而 ObjectAnimation 的不同之处在于，它能自动将变化的值应用于某个 UI 属性

ObjectAnimator.ofFloat 的四个参数，分别是作用的对象，以及作用的属性，起始值，目标值。其中作用的属性，必须在作用的对象中有

一个 setter 方法。用 ObjectAnimator，会自动将更新值应用于构造方法中声明的属性。

使用 ObjectAnimator 时需要注意以下几点：

1，当你想使用 ObjectAnimator 更新某个对象的某个属性，需要有一个该属性的 set 方法，比如上面的“alpha”，则在 foo 对象所属的类中，需要有 setAlpha 方法，系统使用该方法将最新的变化值应用于 UI 属性。当使用 ObjectAnimator 改变某个属性，而且没有该属性的 set 方法，则你可以这样做：

1)，如果你有权限修改代码，则直接为该属性添加一个 set 方法

2)，使用一个包装类，在该类中为该属性增加一个 set 方法，然后再方法中将接收到的值应用于该属性。

3)，使用 ValueAnimator。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tv = (TextView) findViewById(R.id.tv);
    ObjectAnimator animator = ObjectAnimator.ofInt(tv, "textColor", Color.WHITE,Color.RED);
    animator = ValueAnimator.ofFloat(0,1,2);
    animator.setDuration(10000);
    animator.setEvaluator(new ArgbEvaluator());
    animator.start();
```

这里需要设置 TypeEvaluator，否则颜色变化将是一闪一闪的。从这里可以发现，TypeEvaluator 的作用就是计算动画每次渐变的值，控制渐变效果，而 ArgbEvaluator 就是系统提供来处理颜色值变化的。

3.3 Animator

使用 AnimatorSet，能让多个动画同时进行，或者以前后顺序进行，也可以指定某个动画延迟某段时间后进行。

关于动画的前后顺序，主要 with，before，after 三个方法，with，表示同时进行，before 和 after 表示前后关系。

AnimatorSet 中可以包含多个 ObjectAnimator，而且可以包含 AnimatorSet。

```
animator3 = new AnimatorSet();
animator3.play(animator).before(animator2);
```

3.4 Animation Listeners

动画的 Listener 主要是在动画的一些关键时刻做一些操作，比如开始，结束，每一帧的刷新，重复等等时刻。动画主要有连个 Listener，Animator.AnimatorListener 以及 ValueAnimator.AnimatorUpdateListener.

Animator.AnimatorListener

主要提供了以下回调方法

•onAnimationStart()

•onAnimationEnd()

•onAnimationRepeat()

•onAnimationCancel() 动画取消时会回调该方法，也会回调 onAnimationEnd()

该 Listener 提供这么多未实现的方法，如果你只需要实现其中一个方法，但是你只要继承 AnimatorListener 就得实现所有方法。为了避免这个问题，你可以选择继承 AnimatorListenerAdapter，该 adapter 为上面的各个方法提供了一个空实现，继承该类，你只需要实现你想用

的方法。

ValueAnimator.AnimatorUpdateListener

主要提供了 onAnimationUpdate()回调方法，该方法在动画每一帧执行一次，主要应用于 ValueAnimator

3.5 TypeEvaluator 原理

动画是每一帧更新，而动画一共分为多少帧，每一帧应该做出多少变化，这两个问题分别交给 TimeInterpolator 和 TypeEvaluator，TimeInterpolator 计算出每个帧在什么时刻，然后使用 TypeEvaluator 计算在各个帧需要对动画施加的属性做出多少改变。

android 提供了以下几个 evalutor：

- IntEvaluator：属性的值类型为 int；
- FloatEvaluator：属性的值类型为 float；
- ArgbEvaluator：属性的值类型为十六进制颜色值；
- TypeEvaluator：一个接口，可以通过实现该接口自定义 Evaluator。

3.6 TimeInterplator

Time interplator 定义了属性值变化的方式，如线性均匀改变，开始慢然后逐渐快等。在 Property Animation 中是 TimeInterplator，在 View Animation 中是 Interplator，这两个是一样的，在 3.0 之前只有 Interplator，3.0 之后实现代码转移至了 TimeInterplator。Interplator 继承自 TimeInterplator，内部没有任何其他代码。

•**AccelerateInterpolator**

加速，开始时慢中间加速

•**DecelerateInterpolator**

减速，开始时快然后减速

•**AccelerateDecelerateInterolator**

先加速后减速，开始结束时慢，中间加速

•**AnticipateInterpolator**

反向 ，先向相反方向改变一段再加速播放

•**AnticipateOvershootInterpolator**

反向加回弹，先向相反方向改变，再加速播放，会超出目的值然后缓慢移动至目的值

•**BounceInterpolator**

跳跃，快到目的值时值会跳跃，如目的值 100，后面的值可能依次为 75，77，70，80，90，100

•**CycleIinterpolator**

循环，动画循环一定次数，值的改变为一正弦函数：Math.sin(2 * mCycles * Math.PI * input)

•**LinearInterpolator**

线性，线性均匀改变

•**OvershottInterpolator**

回弹，最后超出目的值然后缓慢改变到目的值

•**TimeInterpolator**

一个接口，允许你自定义 interpolator，以上几个都是实现了这个接口

3.7 当 Layout 改变时应用动画

ViewGroup 中的子元素可以通过 setVisibility 使其 Visible、Invisible 或 Gone，当有子元素可见性改变时(VISIBLE、GONE)，可以向其应用动画，通过 LayoutTransition 类应用此类动画：

```
transition.setAnimator(LayoutTransition.DISAPPEARING,
customDisappearingAnim);
```

通过 setAnimator 应用动画，第一个参数表示应用的情境，可以以下 4 种类型：

- APPEARING 当一个元素在其父元素中变为 Visible 时对这个元素应用动画
- CHANGE_APPEARING 当一个元素在其父元素中变为 Visible 时，因系统要重新布局有一些元素需要移动，对这些要移动的元素应用动画
- DISAPPEARING 当一个元素在其父元素中变为 GONE 时对其应用动画
- CHANGE_DISAPPEARING 当一个元素在其父元素中变为 GONE 时，因系统要重新布局有一些元素需要移动，这些要移动的元

素应用动画.

第二个参数为一 Animator。

mTransitioner.setStagger(LayoutTransition.CHANGE_APPEARING, 30);

此函数设置动画延迟时间，参数分别为类型与时间。

3.8 keyframes

keyFrame 是一个 时间/值 对，通过它可以定义一个在特定时间的特定状态，即关键帧，而且在两个 keyFrame 之间可以定义不同的 Interpolator，就好像多个动画的拼接，第一个动画的结束点是第二个动画的开始点。KeyFrame 是抽象类，要通过 ofInt(),ofFloat(),ofObject() 获得适当的 KeyFrame，然后通过 PropertyValuesHolder.ofKeyframe 获得 PropertyValuesHolder 对象，如以下例子：

```
Keyframe kf0 = Keyframe.ofInt(0, 400);
Keyframe kf1 = Keyframe.ofInt(0.25f, 200);
Keyframe kf2 = Keyframe.ofInt(0.5f, 400);
Keyframe kf4 = Keyframe.ofInt(0.75f, 100);
Keyframe kf3 = Keyframe.ofInt(1f, 500);
PropertyValuesHolder pvhRotation = PropertyValuesHolder.ofKeyframe("width", kf0, kf1, kf2, kf4, kf3);
```

ObjectAnimator rotationAnim = ObjectAnimator.ofPropertyValuesHolder(btn2, pvhRotation);

rotationAnim.setDuration(2000);

上述代码的意思为：设置 btn 对象的 width 属性值使其：

- 开始时 Width=400
- 动画开始 1/4 时 Width=200
- 动画开始 1/2 时 Width=400
- 动画开始 3/4 时 Width=100
- 动画结束时 Width=500

第一个参数为时间百分比，第二个参数是在第一个参数的时间时的属性值。

定义了一些 Keyframe 后，通过 PropertyValuesHolder 类的方法 ofKeyframe 一个 PropertyValuesHolder 对象，然后通过 ObjectAnimator.ofPropertyValuesHolder 获得一个 Animator 对象。

用下面的代码可以实现同样的效果（上述代码时间值是线性，变化均匀）：

```
ObjectAnimator oa=ObjectAnimator.ofInt(btn2, "width", 400,200,400,100,500);
oa.setDuration(2000);
oa.start();
```

3.9 Animating Views

在 View Animation 中，对 View 应用 Animation 并没有改变 View 的属性，动画的实现是通过其 Parent View 实现的，在 View 被 drawn 时 Parents View 改变它的绘制参数，draw 后再改变参数 invalidate，这样虽然 View 的大小或旋转角度等改变了，但 View 的实际属性没变，所以有效区域还是应用动画之前的区域，比如你把一按钮放大两倍，但还是放大这前的区域可以触发点击事件。为了改变这一点，在 Android 3.0 中给 View 增加了一些参数并对这些参数增加了相应的 getter/setter 函数（ObjectAnimator 要用这些函数改变这些属性）：

•translationX,translationY: View 相对于原始位置的偏移量

•rotation,rotationX,rotationY: 旋转，rotation 用于 2D 旋转角度，3D 中用到后两个

•scaleX,scaleY: 缩放比

•x,y: View 的最终坐标，是 View 的 left，top 位置加上 translationX，translationY

•alpha: 透明度

跟位置有关的参数有 3 个，以 X 坐标为例，可以通过 getLeft(),getX(),getTranslateX()获得，若有一 Button btn2，布局时其坐标为（40,0）：

```
//应用动画之前
btn2.getLeft();      //40
btn2.getX();      //40
```

```
btn2.getTranslationX();      //0
//应用 translationX 动画
ObjectAnimator     oa=ObjectAnimator.ofFloat(btn2,"translationX",
200);
oa.setDuration(2000);
oa.start();
/*应用 translationX 动画后
btn2.getLeft();      //40
btn2.getX();      //240
btn2.getTranslationX();      //200
*/
//应用 X 动画，假设没有应用之前的 translationX 动画
ObjectAnimator oa=ObjectAnimator.ofFloat(btn2, "x", 200);
oa.setDuration(2000);
oa.start();
/*应用 X 动画后
btn2.getLeft();      //40
btn2.getX();      //200
btn2.getTranslationX();      //160
*/
```

无论怎样应用动画，原来的布局肘的位置通过 getLeft()获得，保持不变；

X 是 View 最终的位置；

translationX 为最终位置与布局时初始位置这差。

所以若就用 translationX 即为在原来基础上移动多少，X 为最终多少

getX()的值为 getLeft()与 getTranslationX()的和

对于 X 动画，源代码是这样的：

```
case X:
        info.mTranslationX = value - mView.mLeft;
        break;
```

Property Animation 也可以在 XML 中定义

•<set> - AnimatorSet

•<animator> - ValueAnimator

•<objectAnimator> - ObjectAnimator

XML 文件应放大/res/animator/中，通过以下方式应用动画：

```
AnimatorSet set = (AnimatorSet)
AnimatorInflater.loadAnimator(myContext,
R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

3.10 ViewPropertyAnimator

如果需要对一个 View 的多个属性进行动画可以用

ViewPropertyAnimator 类，该类对多属性动画进行了优化，会合并一些 invalidate()来减少刷新视图，该类在 3.1 中引入。

以下两段代码实现同样的效果：

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvyY).start();
myView.animate().x(50f).y(100f);
```

4. 案例-人物运动（surfaceView）

在 Android 系统中，有一种特殊的视图，称为 SurfaceView，它拥有独立的绘图表面，即它不与其宿主窗口共享同一个绘图表面。由于拥有独立的绘图表面，因此 SurfaceView 的 UI 就可以在一个独立的线程中进行绘制。又由于不会占用主线程资源，SurfaceView 一方面可以实现复杂而高效的 UI，另一方面又不会导致用户输入得不到及时响应

在前面 Android 控件 TextView 的实现原理分析一文中提到，普通的 Android 控件，例如 TextView、Button 和 CheckBox 等，它们都是将自己的 UI 绘制在宿主窗口的绘图表面之上，这意味着它们的 UI 是在应用程序的主线程中进行绘制的。由于应用程序的主线程除了要绘

制 UI 之外，还需要及时地响应用户输入，否则的话，系统就会认为应用程序没有响应了，因此就会弹出一个 ANR 对话框出来。对于一些游戏画面，或者摄像头预览、视频播放来说，它们的 UI 都比较复杂，而且要求能够进行高效的绘制，因此，它们的 UI 就不适合在应用程序的主线程中进行绘制。这时候就必须要给那些需要复杂而高效 UI 的视图生成一个独立的绘图表面，以及使用一个独立的线程来绘制这些视图的 UI。

SurfaceView是视图(View)的继承类，这个视图里内嵌了一个专门用于绘制的Surface。你可以控制这个Surface的格式和尺寸。Surfaceview控制这个Surface的绘制位置。

surface是纵深排序(Z-ordered)的，这表明它总在自己所在窗口的后面。surfaceview提供了一个可见区域，只有在这个可见区域内 的surface部分内容才可见，可见区域外的部分不可见。surface的排版显示受到视图层级关系的影响，它的兄弟视图结点会在顶端显示。这意味者 surface的内容会被它的兄弟视图遮挡，这一特性可以用来放置遮盖物(overlays)(例如，文本和按钮等控件)。注意，如果surface上面 有透明控件，那么它的每次变化都会引起框架重新计算它和顶层控件的透明效果，这会影响性能。

你可以通过SurfaceHolder接口访问这个surface，getHolder()方法可以得到这个接口。

surfaceview变得可见时，surface被创建；surfaceview隐藏前，surface被销毁。这样能节省资源。如果你要查看 surface被创建和销毁的时机，可以重载surfaceCreated(SurfaceHolder)和surfaceDestroyed(SurfaceHolder)。

surfaceview的核心在于提供了两个线程：UI线程和渲染线程。这里应注意：

1> 所有SurfaceView和SurfaceHolder.Callback的方法都应该在UI线程里调用，一般来说就是应用程序主线程。渲染线程所要访问的各种变量应该作同步处理。

2> 由于surface可能被销毁，它只在SurfaceHolder.Callback.surfaceCreated()和 SurfaceHolder.Callback.surfaceDestroyed()之间有效，所以要确保渲染线程访问的是合法有效的surface。

1、定义

可以直接从内存或者DMA等硬件接口取得图像数据,是个非常重要的绘图容器。

它的特性是：可以在主线程之外的线程中向屏幕绘图上。这样可以避免画图任务繁重的时候造成主线程阻塞，从而提高了程序的反应速度。在游戏开发中多用到SurfaceView，游戏中的背景、人物、动画等等尽量在画布canvas中画出。

2、实现

首先继承SurfaceView并实现SurfaceHolder.Callback接口

使用接口的原因：因为使用SurfaceView 有一个原则，所有的绘图工作必须得在Surface 被创建之后才能开始(Surface一表面，这个概念在 图形编程中常常被提到。基本上我们可以把它当作显存的一个映射，写入到Surface 的内容

可以被直接复制到显存从而显示出来，这使得显示速度会非常快)，而在Surface 被销毁之前必须结束。所以Callback 中的surfaceCreated 和surfaceDestroyed 就成了绘图处理代码的边界。

需要重写的方法

(1) public void surfaceChanged(SurfaceHolder holder,int format,int width,int height){}

//在surface的大小发生改变时激发

(2) public void surfaceCreated(SurfaceHolder holder){}

//在创建时激发，一般在这里调用画图的线程。

(3) public void surfaceDestroyed(SurfaceHolder holder) {}

//销毁时激发，一般在这里将画图的线程停止、释放。

整个过程：继承SurfaceView并实现SurfaceHolder.Callback接口 ----> SurfaceView.getHolder()获得SurfaceHolder对象 ---->SurfaceHolder.addCallback(callback)添加回调函数 ---->SurfaceHolder.lockCanvas()获得Canvas对象并锁定画布----> Canvas绘画 ---->SurfaceHolder.unlockCanvasAndPost(Canvas canvas)结束锁定画图，并提交改变，将图形显示。

3、SurfaceHolder

这里用到了一个类SurfaceHolder,可以把它当成surface的控制器，用来操纵surface。处理它的Canvas上画的效果和动画，控制表面，大小，像素等。

几个需要注意的方法：

(1)、abstract void addCallback(SurfaceHolder.Callback callback);

// 给SurfaceView当前的持有者一个回调对象。

(2)、abstract Canvas lockCanvas();

// 锁定画布，一般在锁定后就可以通过其返回的画布对象Canvas，在其上面画图等操作了。

(3)、abstract Canvas lockCanvas(Rect dirty);

// 锁定画布的某个区域进行画图等..因为画完图后，会调用下面的unlockCanvasAndPost来改变显示内容。

// 相对部分内存要求比较高的游戏来说，可以不用重画dirty外的其它区域的像素，可以提高速度。

(4)、abstract void unlockCanvasAndPost(Canvas canvas);

// 结束锁定画图，并提交改变。

4、实例

图片资源

使用时会对图片进行剪切

mCanvas.clipRect(mClipRect);

绘制人物角色的思想是使用 mCanvas.clipRect(mClipRect);函数来设置画布显示的位置及大小,假设为(presentX，presentY，presentX +

width/10, presentY + height)(presentX 和 presentY 为现在图片的位置，width 和 height 为图片的宽度和高度)，然后使用 mCanvas.drawBitmap 来绘制图片，第一次绘制图片的位置是(presentX，presentY)，然后将绘制图片的位置修改为(presentX - width / 10, presentY),第二次设置的画布显示的位置仍然是 presentX，presentY，presentX + width/10, presentY + height，绘制图片的位置是(presentX - width / 10, presentY)，而第二张图片的位置刚好是 presentX，presentY，所以显示的是第二张图片，依次类推的实现其它图片的显示。

```
blic class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this, null));
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

```
package com.example.propertyanimation_demo;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.graphics.Rect;

import android.util.AttributeSet;
```

```
import android.view.Surface;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
public class MyView extends SurfaceView implements
SurfaceHolder.Callback,
        Runnable {
    private SurfaceHolder holder;
    private Canvas mCanvas;
    private Bitmap mBg1;
    private Bitmap mPlay1;
    private int mWidth;
    private int mHeight;
    private Paint mPaint;
    private String tag = "xiao";
    private BitmapFactory.Options ops;
    private Rect mRect;
    private Rect mClipRect;
    private int mPosition = 20;
    private int mPicPosition = 0;
    private int mStep = 5;
    private int mBamHeight = 600;
```

```
    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        holder = this.getHolder();
        holder.addCallback(this);
        mPaint = new Paint();
        mPaint.setColor(Color.YELLOW);
        ops = new BitmapFactory.Options();
        mBg1 = BitmapFactory.decodeResource(this.getResources(),
        R.drawable.sky, ops);
        mPlay1 = BitmapFactory.decodeResource(getResources(),
R.drawable.run_people, ops);
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
            while(true){
                try {
                    mClipRect = new Rect(mPosition * mStep +
mPlay1.getWidth() / 10, mBamHeight,mPosition * mStep + 2 *
mPlay1.getWidth() / 10, mBamHeight - mPlay1.getHeight());
                    mCanvas = holder.lockCanvas();
```

```
                    if (mCanvas != null) {
                        mCanvas.drawBitmap(mBg1,  null,mRect,
mPaint);

                       mCanvas.save();
                        mCanvas.clipRect(mClipRect);
                        mCanvas.drawBitmap(mPlay1,
mPlay1.getWidth() / 10 + mPosition * mStep - mPicPosition *
mPlay1.getWidth() / 10, mBamHeight - mPlay1.getHeight(), mPaint);
                        mCanvas.restore();
                        mPicPosition++;
                        if(mPosition * mStep > mWidth){
                            mPosition = 0;
                        }
                        if(mPicPosition > 9){
                            mPicPosition = 0;
                        }
                    }

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

```
                    if (mCanvas != null) {
                        holder.unlockCanvasAndPost(mCanvas);
                    }
                }
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

}



@Override
public void surfaceCreated(SurfaceHolder holder) {
    // TODO Auto-generated method stub


}


@Override
```

```
    public void surfaceChanged(SurfaceHolder holder, int format, int
width,
            int height) {
        mWidth = width;
        mHeight = height;
        mRect = new Rect(0, 0, mWidth, mHeight);
        new Thread(this).start();
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {

    }

}
```

15. BroadcastReceiver(广播接收者)

监听系统或应用发出的广播消息

15.1 ip 拨号

```
6
7  public class MyBroadcastReceiver extends BroadcastReceiver {
8      @Override
9      public void onReceive(Context context, Intent intent) {
10         //获得所拨打号码
11         String phoneNum = getResultData();
12         //根据呼出加区号
13         setResultData("区号："+phoneNum);
14     }
15
16 }
17
```

收到广播执行，广播结束销毁

```
    </activity>
    <receiver android:name="com.example.mybroadcastreceiver.MyBroadcastReceiver">
        <intent-filter >
            <action android:name="android.intent.action.NEW_OUTGOING_CALL"/>
        </intent-filter>
    </receiver>
```

过滤呼出电话

设置权限

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

15.2 动态注册与静态注册

静态注册，如上，在 AndroidMainfect.xml 中配置

动态注册，

```
    //动态注册广播接收者
    IntentFilter filter = new IntentFilter();
    filter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);
    mbr = new MyBroadcastReceiver();
    registerReceiver(mbr , filter);
```

销毁

```
    @Override
    protected void onDestroy() {
        unregisterReceiver(mbr);
        super.onDestroy();
    }
```

两者及其接收广播的区别：

1.动态注册广播不是常驻型广播，也就是说广播跟随 activity 的生命周期。注意：在 activity 结束前，移除广播接收器。

静态注册是常驻型，也就是说当应用程序关闭后，如果有信息广播来，程序也会被系统调用自动运行。

2.当广播为有序广播时：

1 优先级高的先接收

2 同优先级的广播接收器，动态优先于静态

3 同优先级的同类广播接收器，静态：先扫描的优先于后扫描的，动态：先注册的优先于后注册的。

3 当广播为普通广播时：

1 无视优先级，动态广播接收器优先于静态广播接收器

2 同优先级的同类广播接收器，静态：先扫描的优先于后扫描的，动态：先注册的优先于后注册的。

15.3 自定义广播

广播若要带数据可用,Intent.puExtras()方法

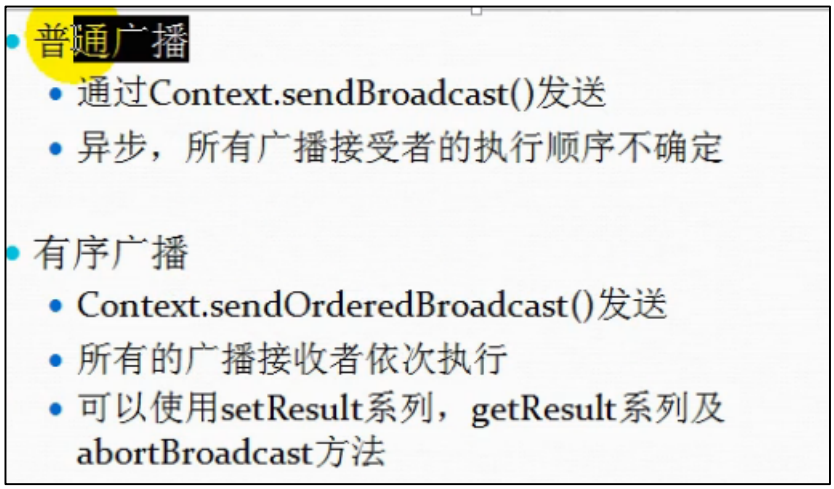
```
public void click(View v){
    //发送广播
    Intent intent = new Intent();
    intent.setAction("com.zhangrui");
    sendBroadcast(intent);
```

发送带有权限的广播

```java
    public void click(View v){
        //发送广播
        Intent intent = new Intent();
        intent.setAction("com.zhangrui");
        sendBroadcast(intent, "com.zhangrui.permission");
    }
```

```xml
<!-- 自定义权限 -->
<permission android:name="com.zhangrui.permission"></permission>
<!-- 赋予权限 -->
<uses-permission android:name="com.zhangrui.permission"/>
```

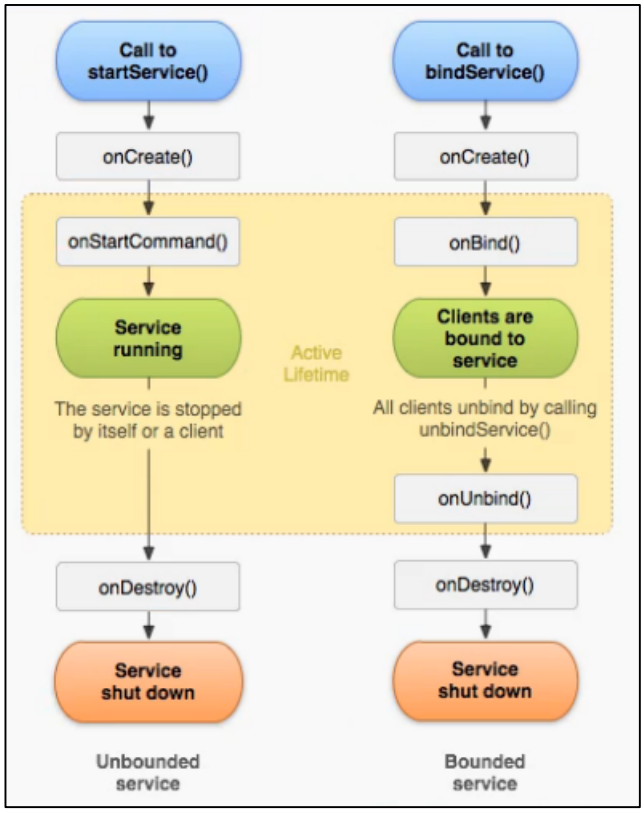
15.4 普通和有序广播



普通广播无法使用 getResult()和 setResult**()系列方法**

有序广播可以使用

16. Service 概述



特点：

1. 在程序后台运行（对用户不可见，无法与用户交互）

2. 生命周期长，被启动后，只要没有停止或所在进程没有结束，就会一直运行

16.1 Service 的创建

在 AndroidMainfest.xml 中申明

```
<service android:name="com.example.service.MyService"></service>
```

```java
public class MyService extends Service {
    /**
     * service被绑定时调用
     */
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    /**
     * service被创建后调用
     */
    @Override
    public void onCreate() {
        super.onCreate();
        Log.e("ss", "onCreate");
    }
    /**
     * service被停止后调用
     */
    @Override
    public void onDestroy() {
        Log.e("ss", "onDestroy");
        super.onDestroy();
    }
    /**
     * service被start后调用
     */
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.e("ss", "onStartCommand");
        return super.onStartCommand(intent, flags, startId);
    }

}
```

```java
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    intent = new Intent();
    intent.setClassName(this, "com.example.service.MyService");

}
public void onClick(View v){

    switch (v.getId()) {
    case R.id.start:
        startService(intent);
        break;
    case R.id.stop:
        stopService(intent);
        break;
    default:
        break;
    }
}
```

16.2 绑定 Service

通过 iBinder 调用服务

Activity、contentProvider、service 本身都可以绑定一个服务，BroadcastReceiver 不可以

```
        intent = new Intent();
        intent.setClassName(this, "com.example.service.MyService");
        sc = new ServiceConnection() {
            /**
             * 当Service崩溃或者被系统强制杀死后调用
             */
            @Override
            public void onServiceDisconnected(ComponentName name) {
                // TODO Auto-generated method stub

            }
            /**
             * 当服务访问者与服务绑定成功后调用
             */
            @Override
            public void onServiceConnected(ComponentName name, IBinder service) {
                // IBinder service是Service中onBind()返回的对象
                MyServiceBinder binder = (MyServiceBinder) service;
                binder.myMethod();//通过在这里实现Activity对服务进行操控

            }
        };
        //绑定服务(参数：服务；服务连接；绑定操作选择策略)，保证两者在同一进程，此时服务与服务的绑定着生命周期相同
        bindService(intent, sc, Context.BIND_AUTO_CREATE);
        unbindService(sc);     //解绑服务
    }
```

```
/**
 * service被绑定时调用
 */
@Override
public IBinder onBind(Intent intent) {
    return new MyServiceBinder();
}
/**
```

```
        return super.onStartCommand(intent, flags, startId);
    }
    public void myMethod(){
        //自定义方法，通过IBinder对象来进行操作
    }
    public class MyServiceBinder extends Binder{
        public void myMethod(){
            MyService.this.myMethod();//内部类访问
        }
    }

}
```

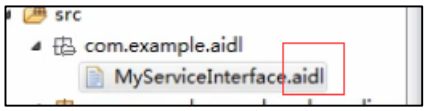
服务与服务绑定者间的桥梁

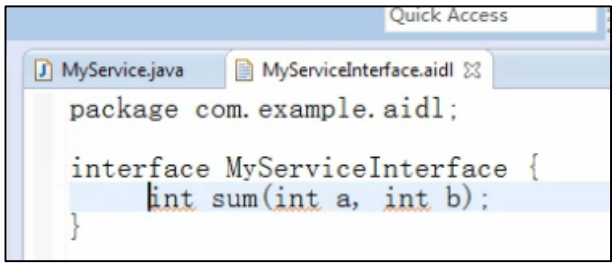
16.3 aidl 实现远程服务绑定（服务与服务绑定不在同一进程）

在Android中，每个应用程序都有自己的进程，当需要在不同的进程之间传递对象时，该如何实现呢？显然，Java中是不支持跨进程内存共享的。因此要传递对象，需要把对象解析成操作系统能够理解的数据格式，以达到跨界对象访问的目的。在JavaEE中，采用RMI通过序列化传递对象。在Android中，则采用AIDL(Android Interface Definition Language：接口定义语言)方式实现。

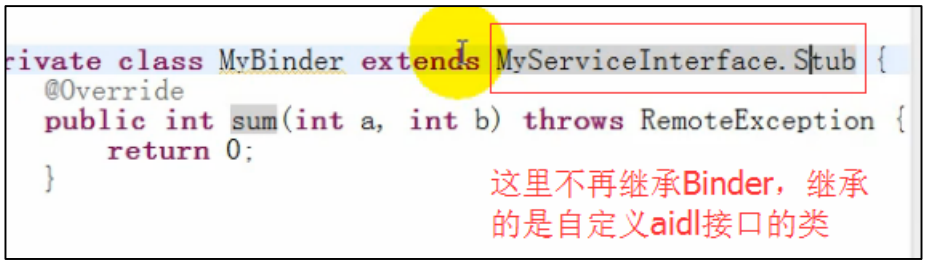
AIDL是一种接口定义语言，用于约束两个进程间的通讯规则，供编译器生成代码，实现Android设备上的两个进程间通信(IPC)。AIDL的IPC机制和EJB所采用的CORBA很类似，进程之间的通信信息，首先会被转换成AIDL协议消息，然后发送给对方，对方收到AIDL协议消息后再转换成相应的对象。由于进程之间的通信信息需要双向转换，所以android采用代理类在背后实现了信息的双向转换，代理类由android编译器生成，对开发人员来说是透明的。

编写Aidl文件时，需要注意下面几点：
1.接口名和aidl文件名相同。
2.接口和方法前不用加访问权限修饰符public,private,protected等,也不能用final,static。
3.Aidl默认支持的类型包括java基本类型（int、long、boolean等）和（String、List、Map、CharSequence），使用这些类型时不需要import声明。对于List和Map中的元素类型必须是Aidl支持的类型。如果使用自定义类型作为参数或返回值，自定义类型必须实现Parcelable接口。
4.自定义类型和AIDL生成的其它接口类型在aidl描述文件中，应该显式import，即便在该类和定义的包在同一个包中。
5.在aidl文件中所有非Java基本类型参数必须加上in、out、inout标记，以指明参数是输入参数、输出参数还是输入输出参数。
6.Java原始类型默认的标记为in,不能为其它标记。





service 中实现 aidl 方法，aidl 会在佛那个生成一个 Stub 类，可在 gen 中观察到



16.4 绑定服务与启动服务混合使用（例如后台播放音乐）

只绑定服务的话，解绑服务后，服务会被销毁，

若想解绑后并不会销毁服务可以在绑定服务后，启动服务，那么服务

就不会被销毁或者先启动服务，再绑定服务

OnUnbind()方法返回 true，当服务被解绑，然后又绑定后会调用 OnRebind（）方法

启动服务再绑定服务后无法停止服务，解绑服务会自动销毁

16.5 IntentService(继承自 Service)

service 中执行耗时操作时使用

```
public class MyIntentService extends IntentService {

    public MyIntentService() {
        super(MyIntentService.class.getName());
    }
    @Override
    public void onCreate() {
        super.onCreate();
        Log.d("bihu","onCreate");
    }
    /**
     * 异步函数
     * */
    @Override
    protected void onHandleIntent(Intent intent) {
        boolean result=Looper.myLooper()==Looper.getMainLooper();
        Log.d("bihu", result+"");
    }
    @Override
    public void onDestroy() {
```

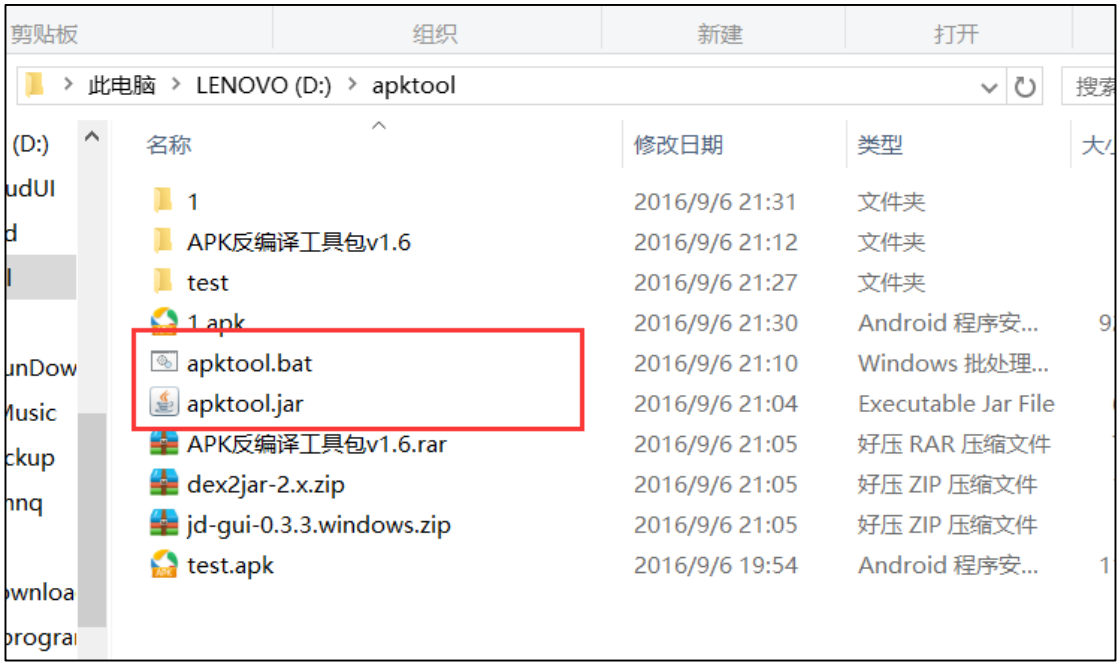
此函数执行结束，服务自动销毁

17. 安卓反编译

工具 apktool.jar、apktoo.bat

dos 界面进入文件所在位置

```
D:\>cd apktool
```



apktool d *.apk 反编译**

apktool b ***将反编译文件夹重新打包成 APK

反编译文件夹中 dist 中便是重新打包的 APK

此时 APK 未签名，需要对其进行签名

签名我们需要用到的工具是 Auto-sign。它主要是利用批处理命令，使用 signapk.jar 对 APK 文件进行签名的。

把 new.apk 文件复制到签名软件的目录下，再用记事本打开 Sign.bat，将其修改为如下代码：

java -jar signapk.jar testkey.x509.pem testkey.pk8 new.apk new_signed.apk

最后双击一下 Sign.bat 即可签名完成，

签名后的 APK 文件就可以在模拟器或者 Android 机器上安装了。

18. Android 常见问题

1. ANR（Application Not Responding）

在主线程执行耗时操作，界面无法响应输入事件，就会发生 ANR 异常。当用户触发输入事件，如果应用 6 秒内没有响应用户输入时间，那么，Android 会认为应用无响应，弹出 ANR 对话框。

2. AndroidManifest.xml file missing!

安卓工程创建不能含有中文