**搜索jar http://search.maven.org/**

# Rxjava

1. 最核心的两个东西是Observables（被观察者，事件源）和Subscribers（观察者）。
2. Observable和Subscriber可以做任何事情  
   Observable可以是一个数据库查询，Subscriber用来显示查询结果；Observable可以是屏幕上的点击事件，Subscriber用来响应点击事件；Observable可以是一个网络请求，Subscriber用来显示请求结果。  
     
   2.Observable和Subscriber是独立于中间的变换过程的。  
   在Observable和Subscriber中间可以增减任何数量的map。整个系统是高度可组合的，操作数据是一个很简单的过程

通常要实现调用一个API需要如下这几个步骤（每个步骤都有一堆公式化代码）：

1. 创建所需的模型类（必要时，添加上注解）。
2. 实现请求—回应管理的网络层代码，并带错误处理。
3. 用后台线程实现请求调用（一般是用异步任务的形式实现），用一个回调函数（Callback Function）来实现在UI线程上呈现回应信息。

# 图片的优化

**将图片转化为缩略图再加载：**

**[java]** [view plain](http://blog.csdn.net/h48582291/article/details/8230130" \o "view plain) [copy](http://blog.csdn.net/h48582291/article/details/8230130" \t "_blank" \o "copy)

1. BitmapFactory.Options options = **new** BitmapFactory.Options（）;
3. options.inSampleSize = 2;
5. Bitmap img = BitmapFactory.decodeFile（"/sdcard/1.png"， options）;

该段代码便是读取1.png的缩略图，长度、宽度都只有原图片的1/2。图片大小削减，占用的内存天然也变小了。这么做的弊病是图片质量变差，inSampleSize的值越大，图片的质量就越差。因为各手机厂商缩放图片的算法不合，在不合手机上的缩放图片质量可能会不合。笔者就遭受过moto手机上图片缩放后质量可以接管，三星手机上同样的缩放比例，质量却差很多的景象。

**2、用ARBG\_4444色彩模式加载图片：**

Android中有四种，分别是：

ALPHA\_8：每个像素占用1byte内存

ARGB\_4444:每个像素占用2byte内存

ARGB\_8888:每个像素占用4byte内存

RGB\_565:每个像素占用2byte内存

Android默认的色彩模式为ARGB\_8888，这个色彩模式色彩最细腻，显示质量最高。但同样的，占用的内存也最大。

**[java]** [view plain](http://blog.csdn.net/h48582291/article/details/8230130" \o "view plain) [copy](http://blog.csdn.net/h48582291/article/details/8230130" \t "_blank" \o "copy)

1. BitmapFactory.Options options = **new** BitmapFactory.Options（）;
3. options.inPreferredConfig = Bitmap.Config.ARGB\_4444;
5. Bitmap img = BitmapFactory.decodeFile（"/sdcard/1.png"， options）

**3、调用图片的recycle（）办法：**

这个其实不是真正降落图片内存的办法。首要目标是标识表记标帜图片对象，便利收受接管图片对象的本地数据。图片对象的本地数据占用的内存最大，并且与法度Java项目组的内存是分隔策画的。所以经常呈现Java heap足够应用，而图片产生OutOfMemoryError的景象。在图片不应用时调用该办法，可以有效降落图片本地数据的峰值，从而削减OutOfMemoryError的概率。不过调用了recycle（）的图片对象处于“放弃”状况，调用时会造成法度错误。所以在无法包管该图片对象绝对不会被再次调用的景象下，不建议应用该办法。希罕要重视已经用setImageBitmap（Bitmap img）办法分派给控件的图片对象，可能会被体系类库调用，造成法度错误。

**4、应用Matrix对象放大的图片如何更改色彩模式：**

固然应用Matrix对象放大图片，必然会花费更多的内存，但有时辰也不得不如许做。放大后的图片应用的ARGB\_8888色彩模式，就算原图片是ARGB\_4444色彩模式也一样，并且没有办法在放大时直接指定色彩模式。可以采取以下办法更改图片色彩模式。

**[java]** [view plain](http://blog.csdn.net/h48582291/article/details/8230130" \o "view plain) [copy](http://blog.csdn.net/h48582291/article/details/8230130" \t "_blank" \o "copy)

1. Matrix matrix = **new** Matrix（）;
3. **float** newWidth = 200;//图片放大后的宽度
5. **float** newHeight = 300;//图片放大后的长度
7. matrix.postScale（newWidth / img.getWidth（）， newHeight/ img.getHeight（））;
9. Bitmap img1 = Bitmap.createBitmap（img， 0， 0， img.getWidth（）， img.getHeight（）， matrix， **true**）;//获得放大的图片
11. img2 = img1.copy（Bitmap.Config.ARGB\_4444， **false**）;//获得ARGB\_4444色彩模式的图片
13. img = **null**;
15. img1 = **null**;

这里比起本来的图片额外生成了一个图片对象img1。然则体系会主动收受接管img1，所以实际内存还是削减了。

归结起来还是以缩略图模式读取图片和削减图片中每个像素占用的内存。这两种办法固然有效，然则也有各自的弊病。实际开辟中还是应当按照景象酌情应用。最王道的办法，还是避免垃圾对象的产生。例如在ListView的应用中，复用convertView等。若是应用AsyncTask加载图片，要及时将引用的ImageView对象置为null。因为AsyncTask是用线程池实现的，所以此中引用的对象可能会拥有很长的生命周期，造成GC无法开释。我还是信赖Android的内存收受接管机制的，recycle什么的固然必然程度上有效，但总感觉不合适Java内存收受接管的原则。（最后这句美满是着魔了）

补充...

尽量不要使用setImageBitmap或setImageResource或BitmapFactory.decodeResource来设置一张大图，因为这些函数在完成decode后，最终都是通过java层的createBitmap来完成的，需要消耗更多内存.

因此，改用先通过BitmapFactory.decodeStream方法，创建出一个bitmap，再将其设为ImageView的 source，decodeStream最大的秘密在于其直接调用 JNI >> nativeDecodeAsset（） 来完成decode，无需再使用java层的createBitmap，从而节省了java层的空间.

如果在读取时加上图片的Config参数，可以更有效减少加载的内存，从而有效阻止抛出out of Memory异常另外，decodeStream直接拿的图片来读取字节码了， 不会根据机器的各种分辨率来自动适应，使用了decodeStream之后，需要在hdpi和mdpi，ldpi中配置相应的图片资源，否则在不同分辨率机器上都是同样大小（像素点数量），显示出来的大小就不对了.

BitmapFactory.Options.inPreferredConfig  
ALPHA\_8：数字为8，图形参数应该由一个字节来表示，应该是一种8位的位图  
ARGB\_4444：4+4+4+4=16，图形的参数应该由两个字节来表示，应该是一种16位的位图.  
ARGB\_8888：8+8+8+8=32，图形的参数应该由四个字节来表示，应该是一种32位的位图.  
RGB\_565：5+6+5=16，图形的参数应该由两个字节来表示，应该是一种16位的位图.  
ALPHA\_8，ARGB\_4444，ARGB\_8888都是透明的位图，也就是所字母A代表透明.  
ARGB\_4444：意味着有四个参数，即A，R，G，B，每一个参数由4bit表示.  
ARGB\_8888：意味着有四个参数，即A，R，G，B，每一个参数由8bit来表示.  
RGB\_565：意味着有三个参数，R，G，B，三个参数分别占5bit，6bit，5bit.

BitmapFactory.Options.inPurgeable；  
如果 inPurgeable 设为True的话表示使用BitmapFactory创建的Bitmap用于存储Pixel的内存空间在系统内存不足时可以被回收，在应用需要再次访问Bitmap的Pixel时（如绘制Bitmap或是调用getPixel），系统会再次调用BitmapFactory decoder重新生成Bitmap的Pixel数组.为了能够重新解码图像，bitmap要能够访问存储Bitmap的原始数据.在inPurgeable为false时表示创建的Bitmap的Pixel内存空间不能被回收，这样BitmapFactory在不停decodeByteArray创建新的Bitmap对象，不同设备的内存不同，因此能够同时创建的Bitmap个数可能有所不同，200个bitmap足以使大部分的设备重新OutOfMemory错误.当isPurgable设为true时，系统中内存不足时，可以回收部分Bitmap占据的内存空间，这时一般不会出现OutOfMemory 错误.

# ViewType

**Getviewtypecount()设置viewtype的总数**

**GetitemViewtype(int position) 得到当前资源的viewtype，可以在这里返回不同的数值**

**Int type=getitemviewtype(posstion);**

**Switch(type){**

**可以在这里加载不同的布局**

**}**

**package** com.example.listdemo;

**import** java.util.ArrayList;

**import** java.util.TreeSet;

**import** android.app.ListActivity;

**import** android.content.Context;

**import** android.os.Bundle;

**import** android.util.Log;

**import** android.view.LayoutInflater;

**import** android.view.View;

**import** android.view.ViewGroup;

**import** android.widget.BaseAdapter;

**import** android.widget.TextView;

**public** **class** MainActivity **extends** ListActivity {

**private** MyCustomAdapter mAdapter;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

mAdapter = **new** MyCustomAdapter();

**for** (**int** i = 1; i < 50; i++) {

mAdapter.addItem("item " + i);

**if** (i % 4 == 0) {

mAdapter.addSeparatorItem("separator " + i);

}

}

setListAdapter(mAdapter);

}

**private** **class** MyCustomAdapter **extends** BaseAdapter {

**private** **static** **final** **int** *TYPE\_ITEM* = 0;

**private** **static** **final** **int** *TYPE\_SEPARATOR* = 1;

**private** **static** **final** **int** *TYPE\_MAX\_COUNT* = *TYPE\_SEPARATOR* + 1;

**private** ArrayList<String> mData = **new** ArrayList();

**private** LayoutInflater mInflater;

**private** TreeSet mSeparatorsSet = **new** TreeSet();

**public** MyCustomAdapter() {

mInflater = (LayoutInflater)getSystemService(Context.*LAYOUT\_INFLATER\_SERVICE*);

}

**public** **void** addItem(**final** String item) {

mData.add(item);

notifyDataSetChanged();

}

**public** **void** addSeparatorItem(**final** String item) {

mData.add(item);

// save separator position

mSeparatorsSet.add(mData.size() - 1);

notifyDataSetChanged();

}

@Override

**public** **int** getItemViewType(**int** position) {

**return** mSeparatorsSet.contains(position) ? *TYPE\_SEPARATOR* : *TYPE\_ITEM*;

}

@Override

**public** **int** getViewTypeCount() {

**return** *TYPE\_MAX\_COUNT*;

}

@Override

**public** **int** getCount() {

**return** mData.size();

}

@Override

**public** String getItem(**int** position) {

**return** mData.get(position);

}

@Override

**public** **long** getItemId(**int** position) {

**return** position;

}

@Override

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

ViewHolder holder = **null**;

**int** type = getItemViewType(position);

Log.*e*("getview","getView " + position + " " + convertView + " type = " + type);

**if** (convertView == **null**) {

holder = **new** ViewHolder();

**switch** (type) {

**case** *TYPE\_ITEM*:

convertView = mInflater.inflate(R.layout.*item1*, **null**);

holder.textView = (TextView)convertView.findViewById(R.id.*text*);

**break**;

**case** *TYPE\_SEPARATOR*:

convertView = mInflater.inflate(R.layout.*item2*, **null**);

holder.textView = (TextView)convertView.findViewById(R.id.*textSeparator*);

**break**;

}

convertView.setTag(holder);

} **else** {

holder = (ViewHolder)convertView.getTag();

}

holder.textView.setText(mData.get(position));

**return** convertView;

}

}

**public** **static** **class** ViewHolder {

**public** TextView textView;

}

}

# 代码混淆

为了防止自己的劳动成果被别人窃取，混淆代码能有效防止被反编译，下面来总结以下混淆代码的步骤：

1. 大家也许都注意到新建一个工程会看到项目下边有这样proguard-project.txt一个文件，这个对混淆代码很重要，如果你不小心删掉了，没关系，从其他地方拷贝一个过来

2. 最重要的就是在proguard-project.txt添加混淆的申明了：

 a. 把所有你的jar包都申明进来，例如：  
 -libraryjars libs/apns\_1.0.6.jar  
 -libraryjars libs/armeabi/libBaiduMapSDK\_v2\_3\_1.so  
 -libraryjars libs/armeabi/liblocSDK4.so  
-libraryjars libs/baidumapapi\_v2\_3\_1.jar  
-libraryjars libs/core.jar  
-libraryjars libs/gesture-imageview.jar  
-libraryjars libs/gson-2.0.jar  
-libraryjars libs/infogracesound.jar  
-libraryjars libs/locSDK\_4.0.jar  
-libraryjars libs/ormlite-**[Android](http://lib.csdn.net/base/15" \t "_blank" \o "undefined)**-4.48.jar  
-libraryjars libs/ormlite-core-4.48.jar  
-libraryjars libs/universal-image-loader-1.9.0.jar

b. 将你不需要混淆的部分申明进来，因为有些类经过混淆会导致程序编译不通过，如下：

-keep public class \* extends android.app.Fragment    
-keep public class \* extends android.app.Activity  
-keep public class \* extends android.app.Application  
-keep public class \* extends android.app.Service  
-keep public class \* extends android.content.BroadcastReceiver  
-keep public class \* extends android.content.ContentProvider  
-keep public class \* extends android.app.backup.BackupAgentHelper  
-keep public class \* extends android.preference.Preference  
-keep public class \* extends android.support.v4.\*\*  
-keep public class com.android.vending.licensing.ILicensingService

--以上都是API里边的类，最好都要避免混淆

有些很特殊的，例如百度地图，你需要添加以下申明：

-keep class com.baidu.\*\* { \*; }   
-keep class vi.com.gdi.bgl.android.\*\*{\*;}

根据我的经验，一般model最好避免混淆（model无关紧要，不混淆也没多大关系）如：

-keep class com.bank.pingan.model.\*\* { \*; }  
下面在贴上关于Umeng分享统计的避免混淆的申明

-dontwarn android.support.v4.\*\*    
-dontwarn org.apache.commons.net.\*\*   
-dontwarn com.tencent.\*\*

-keepclasseswithmembernames class \* {  
    native <methods>;  
}

-keepclasseswithmembernames class \* {  
    public <init>(android.content.Context, android.util.AttributeSet);  
}

-keepclasseswithmembernames class \* {  
    public <init>(android.content.Context, android.util.AttributeSet, int);  
}

-keepclassmembers enum \* {  
    public static \*\*[] values();  
    public static \*\* valueOf(java.lang.String);  
}

-keep class \* implements android.os.Parcelable {  
  public static final android.os.Parcelable$Creator \*;  
}

-keepclasseswithmembers class \* {  
    public <init>(android.content.Context);  
}

-dontshrink  
-dontoptimize  
-dontwarn com.google.android.maps.\*\*  
-dontwarn android.webkit.WebView  
-dontwarn com.umeng.\*\*  
-dontwarn com.tencent.weibo.sdk.\*\*  
-dontwarn com.facebook.\*\*

-keep enum com.facebook.\*\*  
-keepattributes Exceptions,InnerClasses,Signature  
-keepattributes \*Annotation\*  
-keepattributes SourceFile,LineNumberTable

-keep public interface com.facebook.\*\*  
-keep public interface com.tencent.\*\*  
-keep public interface com.umeng.socialize.\*\*  
-keep public interface com.umeng.socialize.sensor.\*\*  
-keep public interface com.umeng.scrshot.\*\*

-keep public class com.umeng.socialize.\* {\*;}  
-keep public class javax.\*\*  
-keep public class android.webkit.\*\*

-keep class com.facebook.\*\*  
-keep class com.umeng.scrshot.\*\*  
-keep public class com.tencent.\*\* {\*;}  
-keep class com.umeng.socialize.sensor.\*\*

-keep class com.tencent.mm.sdk.openapi.WXMediaMessage {\*;}

-keep class com.tencent.mm.sdk.openapi.\*\* implements com.tencent.mm.sdk.openapi.WXMediaMessage$IMediaObject {\*;}

-keep class im.yixin.sdk.api.YXMessage {\*;}  
-keep class im.yixin.sdk.api.\*\* implements im.yixin.sdk.api.YXMessage$YXMessageData{\*;}

-keep public class [your\_pkg].R$\*{  
    public static final int \*;  
}

3.以上工作完成，混淆工作就完成了一大半了，最后需要做的就是在project.properties文件中加上你的混淆文件申明了，如下：

   proguard.config=${sdk.dir}/tools/proguard/proguard-android.txt:proguard-project.txt

4. OK， 最后一步，打签名包测试，如果有问题，仔细看下Log也许有得类不能混淆，那么你得加入到proguard-project.txt文件中

-------以上就是混淆代码的全过程了

最后贴上proguard-project.txt的全部代码：

**[java]** [view plain](http://blog.csdn.net/lovexjyong/article/details/24652085" \o "view plain) [copy](http://blog.csdn.net/lovexjyong/article/details/24652085" \o "copy)

[在CODE上查看代码片](https://code.csdn.net/snippets/316969)

1. # To enable ProGuard in your project, edit project.properties
2. # to define the proguard.config property as described in that file.
3. #
4. # Add project specific ProGuard rules here.
5. # By **default**, the flags in **this** file are appended to flags specified
6. # in ${sdk.dir}/tools/proguard/proguard-android.txt
7. # You can edit the include path and order by changing the ProGuard
8. # include property in project.properties.
9. #
10. # For more details, see
11. #   http://developer.android.com/guide/developing/tools/proguard.html
13. # Add any project specific keep options here:
15. # If your project uses WebView with JS, uncomment the following
16. # and specify the fully qualified **class** name to the JavaScript **interface**
17. # **class**:
18. #-keepclassmembers **class** fqcn.of.javascript.**interface**.**for**.webview {
19. #   **public** \*;
20. #}
21. -optimizationpasses 5
22. -dontusemixedcaseclassnames
23. -dontskipnonpubliclibraryclasses
24. -dontpreverify
25. -verbose
26. -optimizations !code/simplification/arithmetic,!field/\*,!**class**/merging/\*
28. -keepattributes \*Annotation\*
29. -keepattributes Signature
31. -libraryjars libs/apns\_1.0.6.jar
32. -libraryjars libs/armeabi/libBaiduMapSDK\_v2\_3\_1.so
33. -libraryjars libs/armeabi/liblocSDK4.so
34. -libraryjars libs/baidumapapi\_v2\_3\_1.jar
35. -libraryjars libs/core.jar
36. -libraryjars libs/gesture-imageview.jar
37. -libraryjars libs/gson-2.0.jar
38. -libraryjars libs/infogracesound.jar
39. -libraryjars libs/locSDK\_4.0.jar
40. -libraryjars libs/ormlite-android-4.48.jar
41. -libraryjars libs/ormlite-core-4.48.jar
42. -libraryjars libs/universal-image-loader-1.9.0.jar
44. -keep **class** com.baidu.\*\* { \*; }
45. -keep **class** vi.com.gdi.bgl.android.\*\*{\*;}
47. -keep **public** **class** \* **extends** android.app.Fragment
48. -keep **public** **class** \* **extends** android.app.Activity
49. -keep **public** **class** \* **extends** android.app.Application
50. -keep **public** **class** \* **extends** android.app.Service
51. -keep **public** **class** \* **extends** android.content.BroadcastReceiver
52. -keep **public** **class** \* **extends** android.content.ContentProvider
53. -keep **public** **class** \* **extends** android.app.backup.BackupAgentHelper
54. -keep **public** **class** \* **extends** android.preference.Preference
55. -keep **public** **class** \* **extends** android.support.v4.\*\*
56. -keep **public** **class** com.android.vending.licensing.ILicensingService
58. -keep **class** com.google.gson.stream.\*\* { \*; }
59. -keep **class** com.google.gson.examples.android.model.\*\* { \*; }
60. -keep **class** com.uuhelper.Application.\*\* { \*; }
61. -keep **class** net.sourceforge.zbar.\*\* { \*; }
62. -keep **class** com.google.android.gms.\*\* { \*; }
64. -keep **class** com.bank.pingan.model.\*\* { \*; }
66. -keep **public** **class** \* **extends** com.j256.ormlite.android.apptools.OrmLiteSqliteOpenHelper
67. -keep **public** **class** \* **extends** com.j256.ormlite.android.apptools.OpenHelperManager
69. -keep **class** com.android.vending.licensing.ILicensingService
70. -keep **class** android.support.v4.\*\* { \*; }
71. -keep **class** org.apache.commons.net.\*\* { \*; }
72. -keep **class** com.tencent.\*\* { \*; }
74. -keep **class** com.umeng.\*\* { \*; }
75. -keep **class** com.umeng.analytics.\*\* { \*; }
76. -keep **class** com.umeng.common.\*\* { \*; }
77. -keep **class** com.umeng.newxp.\*\* { \*; }
79. -keep **class** com.j256.ormlite.\*\* { \*; }
80. -keep **class** com.j256.ormlite.android.\*\* { \*; }
81. -keep **class** com.j256.ormlite.field.\*\* { \*; }
82. -keep **class** com.j256.ormlite.stmt.\*\* { \*; }
84. -dontwarn android.support.v4.\*\*
85. -dontwarn org.apache.commons.net.\*\*
86. -dontwarn com.tencent.\*\*
88. -keepclasseswithmembernames **class** \* {
89. **native** <methods>;
90. }
92. -keepclasseswithmembernames **class** \* {
93. **public** <init>(android.content.Context, android.util.AttributeSet);
94. }
96. -keepclasseswithmembernames **class** \* {
97. **public** <init>(android.content.Context, android.util.AttributeSet, **int**);
98. }
100. -keepclassmembers **enum** \* {
101. **public** **static** \*\*[] values();
102. **public** **static** \*\* valueOf(java.lang.String);
103. }
105. -keep **class** \* **implements** android.os.Parcelable {
106. **public** **static** **final** android.os.Parcelable$Creator \*;
107. }
109. -keepclasseswithmembers **class** \* {
110. **public** <init>(android.content.Context);
111. }
113. -dontshrink
114. -dontoptimize
115. -dontwarn com.google.android.maps.\*\*
116. -dontwarn android.webkit.WebView
117. -dontwarn com.umeng.\*\*
118. -dontwarn com.tencent.weibo.sdk.\*\*
119. -dontwarn com.facebook.\*\*
121. -keep **enum** com.facebook.\*\*
122. -keepattributes Exceptions,InnerClasses,Signature
123. -keepattributes \*Annotation\*
124. -keepattributes SourceFile,LineNumberTable
126. -keep **public** **interface** com.facebook.\*\*
127. -keep **public** **interface** com.tencent.\*\*
128. -keep **public** **interface** com.umeng.socialize.\*\*
129. -keep **public** **interface** com.umeng.socialize.sensor.\*\*
130. -keep **public** **interface** com.umeng.scrshot.\*\*
132. -keep **public** **class** com.umeng.socialize.\* {\*;}
133. -keep **public** **class** javax.\*\*
134. -keep **public** **class** android.webkit.\*\*
136. -keep **class** com.facebook.\*\*
137. -keep **class** com.umeng.scrshot.\*\*
138. -keep **public** **class** com.tencent.\*\* {\*;}
139. -keep **class** com.umeng.socialize.sensor.\*\*
141. -keep **class** com.tencent.mm.sdk.openapi.WXMediaMessage {\*;}
143. -keep **class** com.tencent.mm.sdk.openapi.\*\* **implements** com.tencent.mm.sdk.openapi.WXMediaMessage$IMediaObject {\*;}
145. -keep **class** im.yixin.sdk.api.YXMessage {\*;}
146. -keep **class** im.yixin.sdk.api.\*\* **implements** im.yixin.sdk.api.YXMessage$YXMessageData{\*;}
148. -keep **public** **class** [your\_pkg].R$\*{
149. **public** **static** **final** **int** \*;
150. }

# Textview所有属性

**[Android](http://lib.csdn.net/base/15" \t "_blank" \o "undefined)**:autoLink设置是否当文本为URL链接/email/电话号码/map时，文本显示为可点击的链接。可选值(none/web/email/phone/map/all)

android:autoText如果设置，将自动执行输入值的拼写纠正。此处无效果，在显示输入法并输入的时候起作用。

android:bufferType指定getText()方式取得的文本类别。选项editable 类似于StringBuilder可追加字符，也就是说getText后可调用append方法设置文本内容。spannable 则可在给定的字符区域使用样式，参见这里1、这里2。

android:capitalize设置英文字母大写类型。此处无效果，需要弹出输入法才能看得到，参见EditView此属性说明。

android:cursorVisible设定光标为显示/隐藏，默认显示。  
android:digits设置允许输入哪些字符。如“1234567890.+-\*/% ()”  
android:drawableBottom在text的下方输出一个drawable，如图片。如果指定一个颜色的话会把text的背景设为该颜色，并且同时和background使用时覆盖后者。  
android:drawableLeft在text的左边输出一个drawable，如图片。  
android:drawablePadding设置text与drawable(图片)的间隔，与drawableLeft、 drawableRight、drawableTop、drawableBottom一起使用，可设置为负数，单独使用没有效果。  
android:drawableRight在text的右边输出一个drawable。  
android:drawableTop在text的正上方输出一个drawable。  
android:editable设置是否可编辑。  
android:editorExtras设置文本的额外的输入数据。  
android:ellipsize设置当文字过长时,该控件该如何显示。有如下值设置：”start”—-省略号显示在开头;”end” ——省略号显示在结尾;”middle”—-省略号显示在中间;”marquee” ——以跑马灯的方式显示(动画横向移动)  
android:freezesText设置保存文本的内容以及光标的位置。  
android:gravity设置文本位置，如设置成“center”，文本将居中显示。  
android:hintText为空时显示的文字提示信息，可通过textColorHint设置提示信息的颜色。此属性在 EditView中使用，但是这里也可以用。  
android:imeOptions附加功能，设置右下角IME动作与编辑框相关的动作，如actionDone右下角将显示一个“完成”，而不设置默认是一个回车符号。这个在EditView中再详细说明，此处无用。  
android:imeActionId设置IME动作ID。  
android:imeActionLabel设置IME动作标签。  
android:includeFontPadding设置文本是否包含顶部和底部额外空白，默认为true。  
android:inputMethod为文本指定输入法，需要完全限定名(完整的包名)。例如：com.google.android.inputmethod.pinyin，但是这里报错找不到。  
android:inputType设置文本的类型，用于帮助输入法显示合适的键盘类型。在EditView中再详细说明，这里无效果。  
android:linksClickable设置链接是否点击连接，即使设置了autoLink。  
android:marqueeRepeatLimit在ellipsize指定marquee的情况下，设置重复滚动的次数，当设置为 marquee\_forever时表示无限次。  
android:ems设置TextView的宽度为N个字符的宽度。这里测试为一个汉字字符宽度  
android:maxEms设置TextView的宽度为最长为N个字符的宽度。与ems同时使用时覆盖ems选项。  
android:minEms设置TextView的宽度为最短为N个字符的宽度。与ems同时使用时覆盖ems选项。  
android:maxLength限制显示的文本长度，超出部分不显示。  
android:lines设置文本的行数，设置两行就显示两行，即使第二行没有数据。  
android:maxLines设置文本的最大显示行数，与width或者layout\_width结合使用，超出部分自动换行，超出行数将不显示。  
android:minLines设置文本的最小行数，与lines类似。  
android:lineSpacingExtra设置行间距。  
android:lineSpacingMultiplier设置行间距的倍数。如”1.2”  
android:numeric如果被设置，该TextView有一个数字输入法。此处无用，设置后唯一效果是TextView有点击效果，此属性在EdtiView将详细说明。  
android:password以小点”.”显示文本  
android:phoneNumber设置为电话号码的输入方式。  
android:privateImeOptions设置输入法选项，此处无用，在EditText将进一步讨论。  
android:scrollHorizontally设置文本超出TextView的宽度的情况下，是否出现横拉条。  
android:selectAllOnFocus如果文本是可选择的，让他获取焦点而不是将光标移动为文本的开始位置或者末尾位置。 TextView中设置后无效果。  
android:shadowColor指定文本阴影的颜色，需要与shadowRadius一起使用。  
android:shadowDx设置阴影横向坐标开始位置。  
android:shadowDy设置阴影纵向坐标开始位置。  
android:shadowRadius设置阴影的半径。设置为0.1就变成字体的颜色了，一般设置为3.0的效果比较好。  
android:singleLine设置单行显示。如果和layout\_width一起使用，当文本不能全部显示时，后面用“…”来表示。如android:text="test\_ singleLine "  
android:singleLine="true" android:layout\_width="20dp"将只显示“t…”。如果不设置singleLine或者设置为false，文本将自动换行  
android:text设置显示文本.  
android:textAppearance设置文字外观。如 “?android:attr/textAppearanceLargeInverse”这里引用的是系统自带的一个外观，?表示系统是否有这种外观，否则使用默认的外观。可设置的值如下：textAppearanceButton/textAppearanceInverse/textAppearanceLarge/textAppearanceLargeInverse/textAppearanceMedium/textAppearanceMediumInverse/textAppearanceSmall/textAppearanceSmallInverse  
android:textColor设置文本颜色  
android:textColorHighlight被选中文字的底色，默认为蓝色  
android:textColorHint设置提示信息文字的颜色，默认为灰色。与hint一起使用。  
android:textColorLink文字链接的颜色.  
android:textScaleX设置文字之间间隔，默认为1.0f。  
android:textSize设置文字大小，推荐度量单位”sp”，如”15sp”  
android:textStyle设置字形[bold(粗体) 0, italic(斜体) 1, bolditalic(又粗又斜) 2] 可以设置一个或多个，用“|”隔开  
android:typeface设置文本字体，必须是以下常量值之一：normal 0, sans 1, serif 2, monospace(等宽字体) 3]  
android:height设置文本区域的高度，支持度量单位：px(像素)/dp/sp/in/mm(毫米)  
android:maxHeight设置文本区域的最大高度  
android:minHeight设置文本区域的最小高度  
android:width设置文本区域的宽度，支持度量单位：px(像素)/dp/sp/in/mm(毫米)，与layout\_width 的区别看这里。  
android:maxWidth设置文本区域的最大宽度  
android:minWidth设置文本区域的最小宽度

# Edittext所有属性

**[Android](http://lib.csdn.net/base/15" \t "_blank" \o "undefined)**:layout\_gravity="center\_vertical"//设置控件显示的位置：默认top，这里居中显示，还有bottom   
  android:hint="请输入数字！"//设置显示在空间上的提示信息   
  android:numeric="integer"//设置只能输入整数，如果是小数则是：decimal   
  android:singleLine="true"//设置单行输入，一旦设置为true，则文字不会自动换行。

   <!--  
     android:gray="top" //多行中指针在第一行第一位置  
   et.setSelection(et.length());//调整光标到最后一行  
    Android:autoText //自动拼写帮助  
    Android:capitalize //首字母大写  
    Android:digits //设置只接受某些数字  
    Android：singleLine //是否单行或者多行，回车是离开文本框还是文本框增加新行  
    Android：numeric //只接受数字  
    Android：password //密码  
    Android：phoneNumber // 输入电话号码  
    Android：editable //是否可编辑  
    Android:autoLink=”all” //设置文本超链接样式当点击网址时，跳向该网址  
  android:password="true"//设置只能输入密码   
  android:textColor = "#ff8c00"//字体颜色   
    android:textStyle="bold"//字体，bold, italic, bolditalic   
    android:textSize="20dip"//大小   
    android:capitalize = "characters"//以大写字母写   
    android:textAlign="center"//EditText没有这个属性，但TextView有   
    android:textColorHighlight="#cccccc"//被选中文字的底色，默认为蓝色   
    android:textColorHint="#ffff00"//设置提示信息文字的颜色，默认为灰色   
    android:textScaleX="1.5"//控制字与字之间的间距   
    android:typeface="monospace"//字型，normal, sans, serif, monospace   
    android:background="@null"//空间背景，这里没有，指透明   
    android:layout\_weight="1"//权重 在控制控   件显示的大小时蛮有用的。   
  android:textAppearance="?android:attr/textAppearanceLargeInverse"//文字外观，这里引用的是系统自带的一个外观，？表示系统是否有这种外观，否则使用默认的外观。不知道这样理解对不对？   
         
　属性名称描述  
　　android:autoLink设置是否当文本为URL链接/email/电话号码/map时，文本显示为可点击的链接。可选值(none/web/email/phone/map/all)  
　　android:autoText如果设置，将自动执行输入值的拼写纠正。此处无效果，在显示输入法并输入的时候起作用。  
　　android:bufferType指定getText()方式取得的文本类别。选项editable 类似于StringBuilder可追加字符，  
　　也就是说getText后可调用append方法设置文本内容。spannable 则可在给定的字符区域使用样式，参见这里1、这里2。  
　　android:capitalize设置英文字母大写类型。此处无效果，需要弹出输入法才能看得到，参见EditView此属性说明。  
　　android:cursorVisible设定光标为显示/隐藏，默认显示。  
　　android:digits设置允许输入哪些字符。如“1234567890.+-\*/% ()”  
　　android:drawableBottom在text的下方输出一个drawable，如图片。如果指定一个颜色的话会把text的背景设为该颜色，并且同时和background使用时覆盖后者。  
　　android:drawableLeft在text的左边输出一个drawable，如图片。  
　　android:drawablePadding设置text与drawable(图片)的间隔，与drawableLeft、drawableRight、drawableTop、drawableBottom一起使用，可设置为负数，单独使用没有效果。

android:drawableRight在text的右边输出一个drawable，如图片。  
　　android:drawableTop在text的正上方输出一个drawable，如图片。  
　　android:editable设置是否可编辑。这里无效果，参见EditView。  
　　android:editorExtras设置文本的额外的输入数据。在EditView再讨论。  
　　android:ellipsize设置当文字过长时,该控件该如何显示。有如下值设置：”start”—?省略号显示在开头;”end”——省略号显示在结尾;”middle”—-省略号显示在中间;”marquee” ——以跑马灯的方式显示(动画横向移动)  
　　android:freezesText设置保存文本的内容以及光标的位置。参见：这里。  
　　android:gravity设置文本位置，如设置成“center”，文本将居中显示。  
　　android:hintText为空时显示的文字提示信息，可通过textColorHint设置提示信息的颜色。此属性在EditView中使用，但是这里也可以用。  
　　android:imeOptions附加功能，设置右下角IME动作与编辑框相关的动作，如actionDone右下角将显示一个“完成”，而不设置默认是一个回车符号。这个在EditView中再详细说明，此处无用。  
　　android:imeActionId设置IME动作ID。在EditView再做说明，可以先看这篇帖子：这里。  
　　android:imeActionLabel设置IME动作标签。在EditView再做说明。  
　　android:includeFontPadding设置文本是否包含顶部和底部额外空白，默认为true。  
　　android:inputMethod为文本指定输入法，需要完全限定名(完整的包名)。例如：com.google.android.inputmethod.pinyin，但是这里报错找不到。  
　　android:inputType设置文本的类型，用于帮助输入法显示合适的键盘类型。在EditView中再详细说明，这里无效果。  
　　android:linksClickable设置链接是否点击连接，即使设置了autoLink。  
　　android:marqueeRepeatLimit在ellipsize指定marquee的情况下，设置重复滚动的次数，当设置为marquee\_forever时表示无限次。  
　　android:ems设置TextView的宽度为N个字符的宽度。这里测试为一个汉字字符宽度，如图：  
android:maxEms设置TextView的宽度为最长为N个字符的宽度。与ems同时使用时覆盖ems选项。  
android:minEms设置TextView的宽度为最短为N个字符的宽度。与ems同时使用时覆盖ems选项。  
　　android:maxLength限制显示的文本长度，超出部分不显示。  
　　android:lines设置文本的行数，设置两行就显示两行，即使第二行没有数据。  
　　android:maxLines设置文本的最大显示行数，与width或者layout\_width结合使用，超出部分自动换行，超出行数将不显示。  
　　android:minLines设置文本的最小行数，与lines类似。  
　　android:lineSpacingExtra设置行间距。  
　　android:lineSpacingMultiplier设置行间距的倍数。如”1.2”  
　　android:numeric如果被设置，该TextView有一个数字输入法。此处无用，设置后唯一效果是TextView有点击效果，此属性在EdtiView将详细说明。  
　　android:password以小点”.”显示文本  
　　android:phoneNumber设置为电话号码的输入方式。  
　　android:privateImeOptions设置输入法选项，此处无用，在EditText将进一步讨论。  
　　android:scrollHorizontally设置文本超出TextView的宽度的情况下，是否出现横拉条。  
　　android:selectAllOnFocus如果文本是可选择的，让他获取焦点而不是将光标移动为文本的开始位置或者末尾位置。TextView中设置后无效果。  
　　android:shadowColor指定文本阴影的颜色，需要与shadowRadius一起使用。效果：  
android:shadowDx设置阴影横向坐标开始位置。  
　　android:shadowDy设置阴影纵向坐标开始位置。  
　　android:shadowRadius设置阴影的半径。设置为0.1就变成字体的颜色了，一般设置为3.0的效果比较好。  
　　android:singleLine设置单行显示。如果和layout\_width一起使用，当文本不能全部显示时，后面用“…”来表示。如android:text="test\_ singleLine " android:singleLine="true" android:layout\_width="20dp"将只显示“t…”。如果不设置singleLine或者设置为false，文本将自动换行  
　　android:text设置显示文本.  
android:shadowDx设置阴影横向坐标开始位置。  
　　android:shadowDy设置阴影纵向坐标开始位置。  
　　android:shadowRadius设置阴影的半径。设置为0.1就变成字体的颜色了，一般设置为3.0的效果比较好。  
　　android:singleLine设置单行显示。如果和layout\_width一起使用，当文本不能全部显示时，后面用“…”来表示。如android:text="test\_ singleLine " android:singleLine="true" android:layout\_width="20dp"将只显示“t…”。如果不设置singleLine或者设置为false，文本将自动换行  
　　android:text设置显示文本.  
　android:textSize设置文字大小，推荐度量单位”sp”，如”15sp”  
　　android:textStyle设置字形[bold(粗体) 0, italic(斜体) 1, bolditalic(又粗又斜) 2] 可以设置一个或多个，用“|”隔开  
　　android:typeface设置文本字体，必须是以下常量值之一：normal 0, sans 1, serif 2, monospace(等宽字体) 3]  
　　android:height设置文本区域的高度，支持度量单位：px(像素)/dp/sp/in/mm(毫米)  
　　android:maxHeight设置文本区域的最大高度  
　　android:minHeight设置文本区域的最小高度  
　　android:width设置文本区域的宽度，支持度量单位：px(像素)/dp/sp/in/mm(毫米)，与layout\_width的区别看这里。  
　　android:maxWidth设置文本区域的最大宽度  
　　android:minWidth设置文本区域的最小宽度  
　　android:textAppearance设置文字外观。如“?android:attr/textAppearanceLargeInverse  
　　”这里引用的是系统自带的一个外观，?表示系统是否有这种外观，否则使用默认的外观。可设置的值如下：textAppearanceButton/textAppearanceInverse/textAppearanceLarge/textAppearanceLargeInverse/textAppearanceMedium/textAppearanceMediumInverse/textAppearanceSmall/textAppearanceSmallInverse  
　　android:textAppearance设置文字外观。如“?android:attr/textAppearanceLargeInverse  
　　”这里引用的是系统自带的一个外观，?表示系统是否有这种外观，否则使用默认的外观。可设置的值如下：textAppearanceButton/textAppearanceInverse/textAppearanceLarge/textAppearanceLargeInverse/textAppearanceM

# [SpannableStringBuilder的使用](http://blog.csdn.net/lovexjyong/article/details/12421345)

通常用于显示文字，但有时候也需要在文字中夹杂一些图片，比如QQ中就可以使用表情图片，又比如需要的文字高亮显示等等，如何在android中也做到这样呢？   
记得android中有个android.text包，这里提供了对文本的强大的处理功能。   
添加图片主要用SpannableString和ImageSpan类：  
   
     Drawable drawable = getResources().getDrawable(id);    
        drawable.setBounds(0, 0, drawable.getIntrinsicWidth(), drawable.getIntrinsicHeight());    
        //需要处理的文本，[smile]是需要被替代的文本    
        SpannableString spannable = new SpannableString(getText().toString()+"[smile]");    
        //要让图片替代指定的文字就要用ImageSpan    
        ImageSpan span = new ImageSpan(drawable, ImageSpan.ALIGN\_BASELINE);    
        //开始替换，注意第2和第3个参数表示从哪里开始替换到哪里替换结束（start和end）    
       //最后一个参数类似数学中的集合,[5,12)表示从5到12，包括5但不包括12    
        spannable.setSpan(span, getText().length(),getText().length()+"[smile]".length(), Spannable.SPAN\_INCLUSIVE\_EXCLUSIVE);      
        setText(spannable);    
   
将需要的文字高亮显示：   
   
   
   
public void highlight(int start,int end){    
        SpannableStringBuilder spannable=new SpannableStringBuilder(getText().toString());//用于可变字符串    
        ForegroundColorSpan span=new ForegroundColorSpan(Color.RED);    
        spannable.setSpan(span, start, end, Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
        setText(spannable);    
    }    
     
加下划线：   
   
   
   
public void underline(int start,int end){    
        SpannableStringBuilder spannable=new SpannableStringBuilder(getText().toString());    
        CharacterStyle span=new UnderlineSpan();    
        spannable.setSpan(span, start, end, Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
        setText(spannable);    
    }    
     
组合运用：  
   
   
   
SpannableStringBuilder spannable=new SpannableStringBuilder(getText().toString());    
        CharacterStyle span\_1=new StyleSpan(android.graphics.Typeface.ITALIC);    
        CharacterStyle span\_2=new ForegroundColorSpan(Color.RED);    
        spannable.setSpan(span\_1, start, end, Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
        spannable.setSpan(span\_2, start, end, Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
        setText(spannable);    
     
案例：带有\n换行符的字符串都可以用此方法显示2种颜色  
   
   
   
    /\*\*   
     \* 带有\n换行符的字符串都可以用此方法显示2种颜色   
     \* @param text   
     \* @param color1   
     \* @param color2   
     \* @return   
     \*/    
    public SpannableStringBuilder highlight(String text,int color1,int color2,int fontSize){    
        SpannableStringBuilder spannable=new SpannableStringBuilder(text);//用于可变字符串    
        CharacterStyle span\_0=null,span\_1=null,span\_2;    
        int end=text.indexOf("\n");    
        if(end==-1){//如果没有换行符就使用第一种颜色显示    
            span\_0=new ForegroundColorSpan(color1);    
            spannable.setSpan(span\_0, 0, text.length(), Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
        }else{    
            span\_0=new ForegroundColorSpan(color1);    
            span\_1=new ForegroundColorSpan(color2);    
            spannable.setSpan(span\_0, 0, end, Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
            spannable.setSpan(span\_1, end+1, text.length(), Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
                
            span\_2=new AbsoluteSizeSpan(fontSize);//字体大小    
            spannable.setSpan(span\_2, end+1, text.length(), Spannable.SPAN\_EXCLUSIVE\_EXCLUSIVE);    
        }    
        return spannable;    
    }

# 判断是否安卓此应用

public static boolean AHAappInstalledOrNot(Context context, String packageName) {  
        PackageManager pm = context.getPackageManager();  
        boolean app\_installed = false;  
        try {  
            pm.getPackageInfo(packageName, PackageManager.GET\_ACTIVITIES);  
            app\_installed = true;  
        } catch (PackageManager.NameNotFoundException e) {  
            app\_installed = false;  
        }  
        return app\_installed;  
    }

# 检查网络状态

/\*\*  
     \* @Description: 检查网络状态  
     \* @return void  
     \*/  
    public void checkNetworkState(Context context) {  
        String msg = "";  
        ConnectivityManager manager = (ConnectivityManager) context  
                .getSystemService(Context.CONNECTIVITY\_SERVICE);  
        State mobile = manager.getNetworkInfo(ConnectivityManager.TYPE\_MOBILE)  
                .getState();  
        State wifi = manager.getNetworkInfo(ConnectivityManager.TYPE\_WIFI)  
                .getState();  
        // 如果3G、wifi、2G等网络状态是连接的，则退出，否则显示提示信息进入网络设置界面  
        if (mobile == State.CONNECTED || mobile == State.CONNECTING) {  
            msg = "3G or 2G is connected!";  
        } else {  
            msg = "3G or 2G is disconnected!";  
        }  
        Log.e("Network", msg);  
        if (wifi == State.CONNECTED || wifi == State.CONNECTING) {  
            msg = "wifi is connected!";  
        } else {  
            msg = "wifi is disconnected!";  
        }  
        Log.e("Network", msg);  
    }

# 横竖屏相关

a.配置文件Manifest中给activity加  android:configChanges="orientation|keyboardHidden"可以防止重新加载activity，  
        加android:screenOrientation="portrait"可强制竖屏  
    b . android:configChanges="orientation|keyboardHidden|locale|screenSize"  4.0以上需要 加locale属性才能横竖屏切换  
    c . android:screenOrientation="user" 加此属性后 可在代码中实现 横竖屏的代码设置    
        <uses-permission android:name="android.permission.CHANGE\_CONFIGURATION" /> 此权限允许用户代码修改横竖屏setting权限  
     代码设定横竖屏权限：  
     Settings.System.putInt(getContentResolver(),Settings.System.ACCELEROMETER\_ROTATION,0); 不允许横竖屏切换  
     Settings.System.putInt(getContentResolver(),Settings.System.ACCELEROMETER\_ROTATION,1); 允许横竖屏切换  
    d . 重写public void onConfigurationChanged(Configuration newConfig) {  
      super.onConfigurationChanged(newConfig);   
       if (this.getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE) {  
       }else if (this.getResources().getConfiguration().orientation == Configuration.ORIENTATION\_PORTRAIT) {  
       }  
      } 可实现横竖屏后操作

# EditText添加键盘search功能

 a: 添加 android:imeOptions="actionSearch" 属性  
    b:  
  
        searchText.setOnEditorActionListener(new OnEditorActionListener() {  
  
                @Override  
                public boolean onEditorAction(TextView v, int actionId,  
                        KeyEvent event) {  
  
                        if ((actionId == 0 || actionId == 3) && event != null) {  
                        ｝  
  
        ｝

# 邮箱正则验证

 private boolean isValidEmail(String mail) {  
  Pattern pattern = Pattern  
    .compile("^[A-Za-z0-9][\\w\\.\_]\*[a-zA-Z0-9]+@[A-Za-z0-9-\_]+\\.([A-Za-z]{2,4})");  
  Matcher mc = pattern.matcher(mail);  
  return mc.matches();  
 }

# [Android多点触摸放大缩小图片](http://blog.csdn.net/lovexjyong/article/details/6852553)

1.Activity

package com.fit.touchimage;

import **[Android](http://lib.csdn.net/base/15" \t "_blank" \o "undefined)**.app.Activity;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Matrix;  
import android.graphics.PointF;  
import android.os.Bundle;  
import android.util.FloatMath;  
import android.view.MotionEvent;  
import android.view.View;  
import android.view.View.OnTouchListener;  
import android.view.ViewGroup.MarginLayoutParams;  
import android.widget.ImageView;

public class MainActivity extends Activity implements OnTouchListener {  
    /\*\* Called when the activity is first created. \*/  
   
 //放大缩小  
 Matrix matrix=new Matrix();  
 Matrix savedMatrix=new Matrix();  
   
 PointF start=new PointF();  
 PointF mid=new PointF();  
 float oldDist;  
   
 private ImageView myImageView;  
   
 //模式  
 static final int NONE=0;  
 static final int DRAG=1;  
 static final int ZOOM=2;  
 int mode=NONE;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
          
        myImageView=(ImageView) findViewById(R.id.myImage);  
        myImageView.setOnTouchListener(this);  
          
    }

 @Override  
 public boolean onTouch(View v, MotionEvent event) {  
  ImageView myImageView=(ImageView) v;  
  switch(event.getAction()&MotionEvent.ACTION\_MASK){  
   //设置拖拉模式   
  case MotionEvent.ACTION\_DOWN:  
   matrix.set(myImageView.getImageMatrix());  
   savedMatrix.set(matrix);  
   start.set(event.getX(),event.getY());  
   mode=DRAG;  
  break;  
  case MotionEvent.ACTION\_UP:  
  case MotionEvent.ACTION\_POINTER\_UP:  
   mode=NONE;  
   break;  
    
  //设置多点触摸模式  
  case MotionEvent.ACTION\_POINTER\_DOWN:  
   oldDist=spacing(event);  
   if(oldDist>10f){  
    savedMatrix.set(matrix);  
    midPoint(mid, event);  
    mode=ZOOM;  
   }  
   break;  
   //若为DRAG模式，则点击移动图片  
  case MotionEvent.ACTION\_MOVE:  
   if(mode==DRAG){  
    matrix.set(savedMatrix);  
    matrix.postTranslate(event.getX()-start.x,event.getY()-start.y);  
   }  
   //若为ZOOM模式，则点击触摸缩放  
   else if(mode==ZOOM){  
    float newDist=spacing(event);  
    if(newDist>10f){  
     matrix.set(savedMatrix);  
     float scale=newDist/oldDist;  
     //设置硕放比例和图片的中点位置  
     matrix.postScale(scale,scale, mid.x,mid.y);  
    }  
   }  
   break;  
  }   
  myImageView.setImageMatrix(matrix);  
  return true;  
 }  
 //计算移动距离  
 private float spacing(MotionEvent event){  
  float x=event.getX(0)-event.getX(1);  
  float y=event.getY(0)-event.getY(1);  
  return FloatMath.sqrt(x\*x+y\*y);  
 }  
 //计算中点位置  
 private void midPoint(PointF point,MotionEvent event){  
  float x=event.getX(0)+event.getX(1);  
  float y=event.getY(0)+event.getY(1);  
  point.set(x/2,y/2);  
 }  
}

2.布局

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
 android:orientation="vertical" android:layout\_width="fill\_parent"  
 android:layout\_height="fill\_parent" android:gravity="center">  
 <ImageView android:layout\_width="fill\_parent"  
  android:layout\_height="fill\_parent" android:scaleType="matrix"  
  android:id="@+id/myImage" android:src="@drawable/xiaoxiong"/>  
</LinearLayout>

# 自定义view

<http://www.cnblogs.com/mengdd/p/3332882.html>

## Android中View的绘制过程

　　当Activity获得焦点时，它将被要求绘制自己的布局，Android framework将会处理绘制过程，Activity只需提供它的布局的根节点。

　　绘制过程从布局的根节点开始，从根节点开始测量和绘制整个layout tree。

　　每一个**[ViewGroup](http://developer.android.com/reference/android/view/ViewGroup.html)**负责要求它的每一个孩子被绘制，每一个**[View](http://developer.android.com/reference/android/view/View.html)**负责绘制自己。

　　因为整个树是按顺序遍历的，所以父节点会先被绘制，而兄弟节点会按照它们在树中出现的顺序被绘制。

**绘制是一个两遍（two pass）的过程：一个measure pass和一个layout pass。**

**测量过程（measuring pass）**是在**[measure(int, int)](http://developer.android.com/reference/android/view/View.html" \l "measure(int, int))**中实现的，是从树的顶端由上到下进行的。

　　在这个递归过程中，每一个View会把自己的dimension specifications传递下去。

　　在measure pass的最后，每一个View都存储好了自己的measurements，即测量结果。

　　第二个是**布局过程（layout pass），**它发生在**[layout(int, int, int, int)](http://developer.android.com/reference/android/view/View.html" \l "layout(int, int, int, int))**中，仍然是从上到下进行（top-down）。

　　在这一遍中，每一个parent都会负责用测量过程中得到的尺寸，把自己的所有孩子放在正确的地方。

## 尺寸的父子关系处理

　　当一个View对象的**[measure()](http://developer.android.com/reference/android/view/View.html" \l "measure(int, int))** 方法返回时，它的 **[getMeasuredWidth()](http://developer.android.com/reference/android/view/View.html" \l "getMeasuredWidth())** 和**[getMeasuredHeight()](http://developer.android.com/reference/android/view/View.html" \l "getMeasuredHeight())**值应该被设置好了，并且它的所有子孙的值也应该一起被设置好了。

**一个View对象的measured width 和measured height的值必须考虑到它的父容器给它的限制。**

　　这样就保证了在measure pass的最后，所有的parent都接受了它的所有孩子的measurements结果。

**注意：**一个parent可能会不止一次地对它的孩子调用**[measure()](http://developer.android.com/reference/android/view/View.html" \l "measure(int, int))**方法。

　　比如，第一遍的时候，一个parent可能测量它的每一个孩子，并没有指定尺寸，parent只是为了发现它们想要多大；

　　如果第一遍之后得知，所有孩子的无限制的尺寸总和太大或者太小，parent会再次对它的孩子调用**[measure()](http://developer.android.com/reference/android/view/View.html" \l "measure(int, int))**方法，这时候parent会设定规则，介入这个过程，使用实际的值。

　　（**即，让孩子自由发展不成，于是家长介入**）。

## 布局属性说明

**LayoutParams**是View用来告诉它的父容器它想要怎样被放置的参数。

　　最基本的LayoutParams基类仅仅描述了View想要多大，即指明了尺寸属性。

　　即View在XML布局时通常需要指明的宽度和高度属性。

**每一个维度都可以指定成下列三种值之一：**

　　1.**FILL\_PARENT**(API Level 8之后重命名为**MATCH\_PARENT**)，表示View想要尽量和它的parent一样大（减去边距）。

　　2.**WRAP\_CONTENT**，表示View想要刚好大到可以包含它的内容（包括边距）。

　　3.**具体的数值**。

　　ViewGroup的不同子类（不同的布局类）有相应的LayoutParams子类，其中会包含更多的布局相关属性。

## onMeasure方法

**onMeasure**方法是测量view和它的内容，决定measured width和measured height的，这个方法由**[measure(int, int)](http://developer.android.com/reference/android/view/View.html" \l "measure(int, int))**方法唤起，子类可以覆写onMeasure来提供更加准确和有效的测量。

　　有一个约定：在覆写onMeasure方法的时候，必须调用**[setMeasuredDimension(int,int)](http://developer.android.com/reference/android/view/View.html" \l "setMeasuredDimension(int, int))**来存储这个View经过测量得到的measured width and height。

　　如果没有这么做，将会由**[measure(int, int)](http://developer.android.com/reference/android/view/View.html" \l "measure(int, int))**方法抛出一个**IllegalStateException**。

　　onMeasure方法的声明如下：

protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)

　　其中两个输入参数：

　　widthMeasureSpec

　　heightMeasureSpec

　　分别是**parent提出的水平和垂直的空间要求**。

　　这两个要求是按照**[View.MeasureSpec](http://developer.android.com/reference/android/view/View.MeasureSpec.html)**类来进行编码的。

　　参见**[View.MeasureSpec](http://developer.android.com/reference/android/view/View.MeasureSpec.html)**这个类的说明：这个类包装了从parent传递下来的布局要求，传递给这个child。

　　每一个MeasureSpec代表了对宽度或者高度的一个要求。

**每一个MeasureSpec有一个尺寸（size）和一个模式（mode）构成。**

　　MeasureSpecs这个类提供了把一个<size, mode>的元组包装进一个int型的方法，从而减少对象分配。当然也提供了逆向的解析方法，从int值中解出size和mode。

**有三种模式：**

**UNSPECIFIED**

　　这说明parent没有对child强加任何限制，child可以是它想要的任何尺寸。

**EXACTLY**

　　Parent为child决定了一个绝对尺寸，child将会被赋予这些边界限制，不管child自己想要多大。

**AT\_MOST**

　　Child可以是自己任意的大小，但是有个绝对尺寸的上限。

　　覆写onMeasure方法的时候，子类有责任确保measured height and width至少为这个View的最小height和width。

　　（**[getSuggestedMinimumHeight()](http://developer.android.com/reference/android/view/View.html" \l "getSuggestedMinimumHeight())** and**[getSuggestedMinimumWidth()](http://developer.android.com/reference/android/view/View.html" \l "getSuggestedMinimumWidth())**）。

## onLayout

　　这个方法是在layout pass中被调用的，用于确定View的摆放位置和大小。方法声明：

protected void onLayout (boolean changed, int left, int top, int right, int bottom)

　　其中的上下左右参数都是相对于parent的。

　　如果View含有child，那么onLayout中需要对每一个child进行布局。

## 自定义View Demo

　　API Demos中的LabelView类是一个继承自View的自定义类的例子：

[复制代码](javascript:void(0);)

/\*

\* Copyright (C) 2007 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.example.android.apis.view;

// Need the following import to get access to the app resources, since this

// class is in a sub-package.

import android.content.Context;

import android.content.res.TypedArray;

import android.graphics.Canvas;

import android.graphics.Paint;

import android.util.AttributeSet;

import android.view.View;

import com.example.android.apis.R;

/\*\*

\* Example of how to write a custom subclass of View. LabelView

\* is used to draw simple text views. Note that it does not handle

\* styled text or right-to-left writing systems.

\*

\*/

public class LabelView extends View {

private Paint mTextPaint;

private String mText;

private int mAscent;

/\*\*

\* Constructor. This version is only needed if you will be instantiating

\* the object manually (not from a layout XML file).

\* @param context

\*/

public LabelView(Context context) {

super(context);

initLabelView();

}

/\*\*

\* Construct object, initializing with any attributes we understand from a

\* layout file. These attributes are defined in

\* SDK/assets/res/any/classes.xml.

\*

\* @see android.view.View#View(android.content.Context, android.util.AttributeSet)

\*/

public LabelView(Context context, AttributeSet attrs) {

super(context, attrs);

initLabelView();

TypedArray a = context.obtainStyledAttributes(attrs,

R.styleable.LabelView);

CharSequence s = a.getString(R.styleable.LabelView\_text);

if (s != null) {

setText(s.toString());

}

// Retrieve the color(s) to be used for this view and apply them.

// Note, if you only care about supporting a single color, that you

// can instead call a.getColor() and pass that to setTextColor().

setTextColor(a.getColor(R.styleable.LabelView\_textColor, 0xFF000000));

int textSize = a.getDimensionPixelOffset(R.styleable.LabelView\_textSize, 0);

if (textSize > 0) {

setTextSize(textSize);

}

a.recycle();

}

private final void initLabelView() {

mTextPaint = new Paint();

mTextPaint.setAntiAlias(true);

// Must manually scale the desired text size to match screen density

mTextPaint.setTextSize(16 \* getResources().getDisplayMetrics().density);

mTextPaint.setColor(0xFF000000);

setPadding(3, 3, 3, 3);

}

/\*\*

\* Sets the text to display in this label

\* @param text The text to display. This will be drawn as one line.

\*/

public void setText(String text) {

mText = text;

requestLayout();

invalidate();

}

/\*\*

\* Sets the text size for this label

\* @param size Font size

\*/

public void setTextSize(int size) {

// This text size has been pre-scaled by the getDimensionPixelOffset method

mTextPaint.setTextSize(size);

requestLayout();

invalidate();

}

/\*\*

\* Sets the text color for this label.

\* @param color ARGB value for the text

\*/

public void setTextColor(int color) {

mTextPaint.setColor(color);

invalidate();

}

/\*\*

\* @see android.view.View#measure(int, int)

\*/

@Override

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

setMeasuredDimension(measureWidth(widthMeasureSpec),

measureHeight(heightMeasureSpec));

}

/\*\*

\* Determines the width of this view

\* @param measureSpec A measureSpec packed into an int

\* @return The width of the view, honoring constraints from measureSpec

\*/

private int measureWidth(int measureSpec) {

int result = 0;

int specMode = MeasureSpec.getMode(measureSpec);

int specSize = MeasureSpec.getSize(measureSpec);

if (specMode == MeasureSpec.EXACTLY) {

// We were told how big to be

result = specSize;

} else {

// Measure the text

result = (int) mTextPaint.measureText(mText) + getPaddingLeft()

+ getPaddingRight();

if (specMode == MeasureSpec.AT\_MOST) {

// Respect AT\_MOST value if that was what is called for by measureSpec

result = Math.min(result, specSize);

}

}

return result;

}

/\*\*

\* Determines the height of this view

\* @param measureSpec A measureSpec packed into an int

\* @return The height of the view, honoring constraints from measureSpec

\*/

private int measureHeight(int measureSpec) {

int result = 0;

int specMode = MeasureSpec.getMode(measureSpec);

int specSize = MeasureSpec.getSize(measureSpec);

mAscent = (int) mTextPaint.ascent();

if (specMode == MeasureSpec.EXACTLY) {

// We were told how big to be

result = specSize;

} else {

// Measure the text (beware: ascent is a negative number)

result = (int) (-mAscent + mTextPaint.descent()) + getPaddingTop()

+ getPaddingBottom();

if (specMode == MeasureSpec.AT\_MOST) {

// Respect AT\_MOST value if that was what is called for by measureSpec

result = Math.min(result, specSize);

}

}

return result;

}

/\*\*

\* Render the text

\*

\* @see android.view.View#onDraw(android.graphics.Canvas)

\*/

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

canvas.drawText(mText, getPaddingLeft(), getPaddingTop() - mAscent, mTextPaint);

}

}

# 属性动画

参数：alpha，scaleX，scaleY

# 图片变暗（待测试）

www.MyException.Cn  网友分享于：2013-09-03  浏览：21次

ColorMatrixColorFilter色彩过滤（离线用户的灰色头像处理）

ColorMatrixColorFilter颜色过滤（离线用户的灰色头像处理）  
Android的图片资源默认是静态的，单实例；如果两个IM好友的头像一样，最简单的都是用的软件自带头像，有一个在线，一个离线，直接改变头像的灰度，则两个用户的头像都会变灰或者在线，答案是：Drawable.mutate()。

Drawable mDrawable = context.getResources().getDrawable(R.drawable.face\_icon);

//Make this drawable mutable.

//A mutable drawable is guaranteed to not share its state with any other drawable.

mDrawable.mutate();

ColorMatrix cm = new ColorMatrix();

cm.setSaturation(0);

ColorMatrixColorFilter cf = new ColorMatrixColorFilter(cm);

mDrawable.setColorFilter(cf);

# RelativeLayout.LayoutParams.addRule()

1. 原理说明：

我们知道，在 RelativeLayout 布局中有很多特殊的属性，通常在载入布局之前，在相关的xml文件中进行静态设置即可。

但是，在有些情况下，我们需要动态设置布局的属性，在不同的条件下设置不同的布局排列方式，这时候就需要用到 RelativeLayout.LayoutParams.addRule() 方法，该方法有两种重载方式：

* addRule(int verb) ：用此方法时，所设置节点的属性不能与其它兄弟节点相关联或者属性值为布尔值（布尔值的属性，设置时表示该属性为 true，不设置就默认为 false），比如：addRule(RelativeLayout.CENTER\_VERTICAL)  就表示在 RelativeLayout 中的相应节点是垂直居中的。
* addRule(int verb, int anchor) ：该方法所设置节点的属性必须关联其它的兄弟节点或者属性为布尔值（ 属性为布尔值时，anchor 为 RelativeLayout.TRUE 表示 true，anchor 为0表示 false），比如：addRule(RelativeLayout.ALIGN\_LEFT, R.id.date) 就表示 RelativeLayout 中的相应节点放置在一个 id 值为 date 的兄弟节点的左边。

# RecyclerView

RecyclerView本身提供了三个LayoutManager的实现

* LinearLayoutManager
* GridLayoutManager
* StaggeredGridLayoutManager

## 瀑布流

RecyclerView本身提供了三个LayoutManager的实现

* LinearLayoutManager
* GridLayoutManager
* StaggeredGridLayoutManager

第一个和第二个大家比较常用，今天我们就来使用第三个比较陌生的StaggeredGridLayoutManager，让你分分钟实现瀑布流布局。   
首先来看下最后的效果



好吧，让我们来实现它吧   
首先是Item的布局masonry\_item.xml很简单，就是一张图片加文字,item背景设置为白色

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@color/white">

<ImageView

android:id="@+id/masonry\_item\_img"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:adjustViewBounds="true"

android:scaleType="centerCrop"/>

<TextView

android:id="@+id/masonry\_item\_title"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:gravity="center"/>

</LinearLayout>

为了简单起见，我们假设每个item的数据是一个产品信息

public class Product {

private int img;

private String title;

public Product(int img, String title) {

this.img = img;

this.title = title;

}

public int getImg() {

return img;

}

public void setImg(int img) {

this.img = img;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

}

recyclerView的adapter也很简单，构造方法接受产品列表数据源

public class MasonryAdapter extends RecyclerView.Adapter<MasonryAdapter.MasonryView>{

private List<Product> products;

public MasonryAdapter(List<Product> list) {

products=list;

}

@Override

public MasonryView onCreateViewHolder(ViewGroup viewGroup, int i) {

View view= LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.masonry\_item, viewGroup, false);

return new MasonryView(view);

}

@Override

public void onBindViewHolder(MasonryView masonryView, int position) {

masonryView.imageView.setImageResource(products.get(position).getImg());

masonryView.textView.setText(products.get(position).getTitle());

}

@Override

public int getItemCount() {

return products.size();

}

public static class MasonryView extends RecyclerView.ViewHolder{

ImageView imageView;

TextView textView;

public MasonryView(View itemView){

super(itemView);

imageView= (ImageView) itemView.findViewById(R.id.masonry\_item\_img );

textView= (TextView) itemView.findViewById(R.id.masonry\_item\_title);

}

}

}

主界面Activity代码就可以把数据源和view连起来了

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

recyclerView= (RecyclerView) findViewById(R.id.recycler);

//设置layoutManager

recyclerView.setLayoutManager(new StaggeredGridLayoutManager(2,StaggeredGridLayoutManager.VERTICAL));

//设置adapter

initData();

MasonryAdapter adapter=new MasonryAdapter(productList);

recyclerView.setAdapter(adapter);

//设置item之间的间隔

SpacesItemDecoration decoration=new SpacesItemDecoration(16);

recyclerView.addItemDecoration(decoration);

}

**第一**   
大家看到我们把recyclerview的layoutManager设置成了

new StaggeredGridLayoutManager(2,StaggeredGridLayoutManager.VERTICAL)

参数含义显而易见，2就是2列，第二个参数是垂直方向

**第二**

SpacesItemDecoration decoration=new SpacesItemDecoration(16);

recyclerView.addItemDecoration(decoration);

设置间隔，我们自定义了一个SpacesItemDecoration,代码非常简单

public class SpacesItemDecoration extends RecyclerView.ItemDecoration {

private int space;

public SpacesItemDecoration(int space) {

this.space=space;

}

@Override

public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {

outRect.left=space;

outRect.right=space;

outRect.bottom=space;

if(parent.getChildAdapterPosition(view)==0){

outRect.top=space;

}

}

}

重点就这2个地方 ，几行代码就实现了一个漂亮的瀑布流布局，有兴趣自己去玩下哦。

# Svn和git的区别 安装配置

<http://pan.baidu.com/s/1qYCFQoc> phsr

百度云“资源”文件夹下的： 20160304版本控制原理SVN和GIT区别.ppt

# 二维码

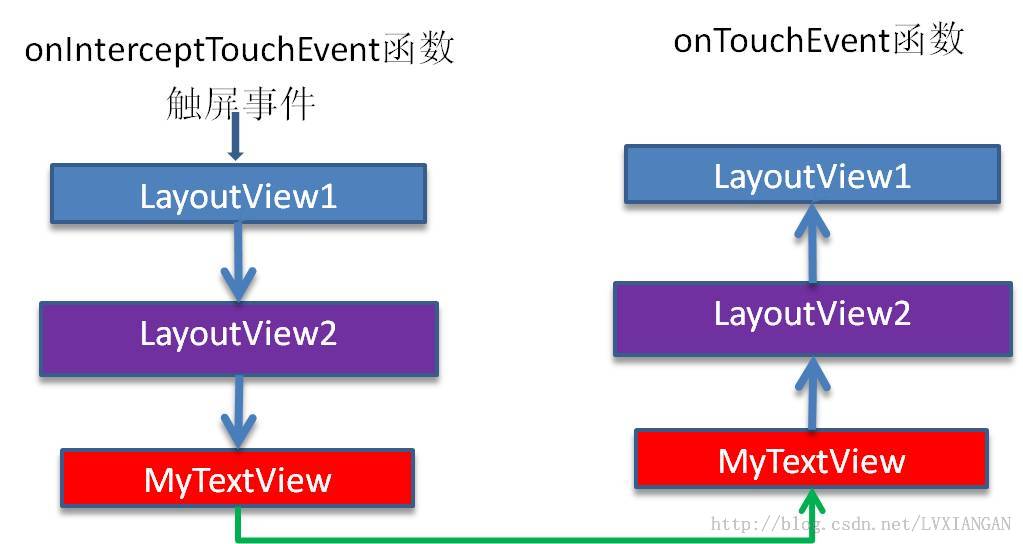
## 生成二维码

1. /\*\*
2. \* 根据字符串进行二维编码
3. \* @param str   需要编码的字符串
4. \* @param widthAndHeight  需要生成的bitmap的高宽
5. \* @return
6. \*/
7. **public** Bitmap enCode(String str,**int** widthAndHeight){

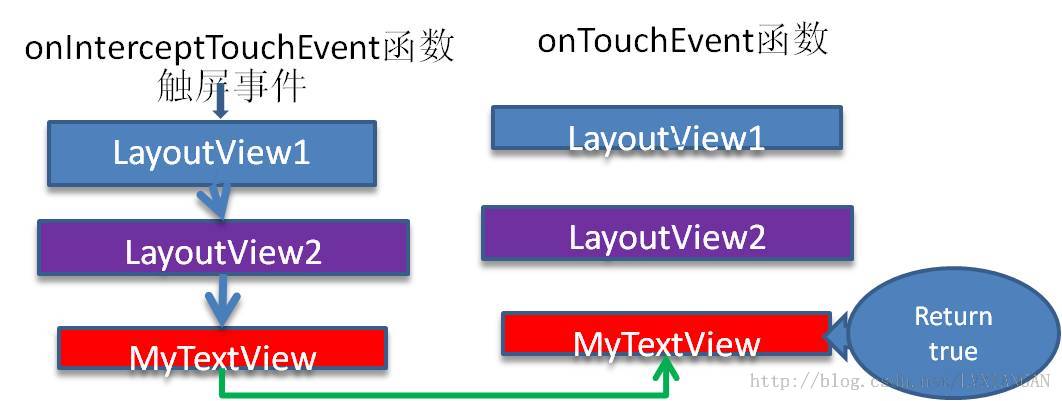
10. **if** (!str.equals("")&&str!=**null**)
11. **try** {
12. **return** EncodingHandler.createQRCode(str, widthAndHeight);
13. } **catch** (WriterException e) {
14. // TODO Auto-generated catch block
15. e.printStackTrace();
16. }
17. **else**
18. **try** {
19. **return** EncodingHandler.createQRCode("null", widthAndHeight);
20. } **catch** (WriterException e) {
21. // TODO Auto-generated catch block
22. e.printStackTrace();
23. }
24. **return** **null**;
25. }

# OnInterceptTouchEvent和OnTouchEvent

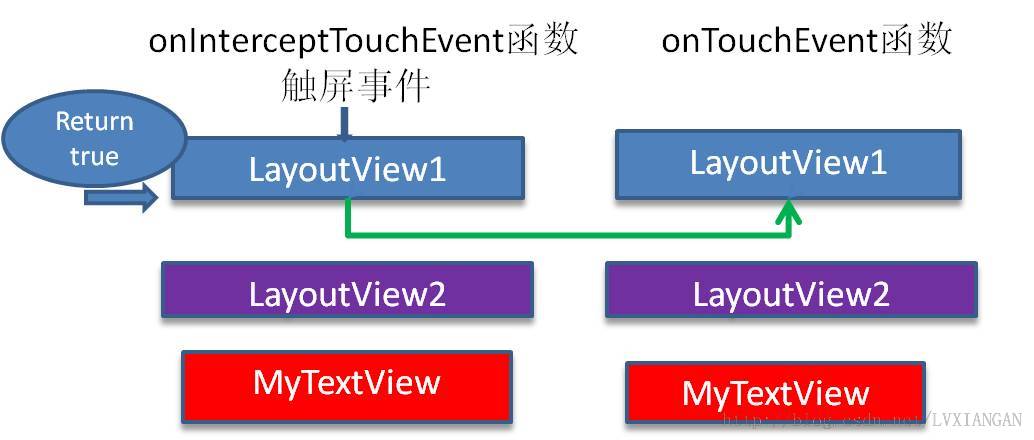
通常外围的layoutview1,layoutview2,只是布局的容器不需要响应触屏的点击事件，仅仅Mytextview需要相应点击。但这只是一般情况，一些特殊的布局可能外围容器也要响应，甚至不让里面的mytextview去响应。更有特殊的情况是，动态更换响应对象。  
那么首先看一下默认的触屏事件的在两个函数之间的传递流程。如下图：



如果仅仅想让MyTextView来响应触屏事件，让MyTextView的OnTouchEvent返回true,那么事件流就变成如下图，可以看到layoutview1,layoutview2已经不能进入OnTouchEvent：



另外一种情况，就是外围容器想独自处理触屏事件，那么就应该在相应的onInterceptTouchEvent函数中返回true,表示要截获触屏事件，比如layoutview1作截获处理，处理流变成如下图：



# 适配

一、关于布局适配

1、不要使用绝对布局

2、尽量使用match\_parent 而不是fill\_parent 。

3、能够使用权重的地方尽量使用权重（android:layout\_weight）

4、如果是纯色背景，尽量使用android的shape 自定义。

5、如果需要在特定分辨率下适配，可以在res目录上新建layout-HxW.xml的文件夹。比如要适配1080\*1800的屏幕（魅族MX3采用此分辨率）则新建layout-1800x1080.xml的文件夹，然后在下面定义布局。Android系统会优先查找分辨率相同的布局，如果不存在则换使用默认的layout下的布局。

二、关于图片制作

1、关于设计：

设计图先定下一个要设计的尺寸，而且尽量采用在目前最流行的屏幕尺寸（比如目前占屏幕比重比较多的是480系列，也即是480\*800或者400\*854，下面的图标制作也在次基础上进行比例的换算）上设计。

先了解一下屏幕的级别：

说明：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 屏幕级别 | 屏幕密度 | 比率（相对） | | 物理大小（英寸） | 像素大小 | 通常的分辨率 |
| ldpi | 120 | 3 | 0.75 | 1 | 120 |  |
| mdpi | 160 | 4 | 1 | 1 | 160 | 320\*480 |
| hdpi | 240 | 6 | 1.5 | 1 | 240 | 480\*800 |
| xhdpi | 320 | 8 | 2 | 1 | 320 | 720\*1280 |
| xxhdpi | 480 | 12 | 3 | 1 | 480 | 1080\*1800 |

屏幕级别：

注意屏幕级别是按照密度分级，和像素没有关系。如果非要让密度和像素扯上关系，则需要一个参照系，android使用mdpi级别作为标准参照屏幕，也就是说在320\*480分辨率的手机上一个密度可以容纳一个像素。然后其他密度级别则在此基础上进行对比。如果理想情况下，480\*800的屏幕一个密度可以容纳1.5个像素。

物理大小：

单位是英寸而不是像素，也就说一个英寸在任何分辨率下显示的大小都是一样的，但是像素在密度不同的手机里面显示的实际的大小是不一样的（这就是为什么android手机需要适配的原因）。

然后就是重点。

假设1像素在160密度下显示1英寸，则1像素在240密度基础上显示大约0.67英寸，在320密度下显示0.5英寸。于是就出现一种情况，在电脑上的一个像素，在不同的手机上看实际的大小不一样。那么怎么让“设计效果”在不同的手机上看起来显示的区域一样呢？

还是假设一个像素在160密度下的显示在一个密度内，也假设就是一英寸。那么需要几个像素才能在240密度级别下显示在一英寸范围内呢？答案是1.5个像素（根据上图的比率换算）。

了解了这个关系，接下来就是图标的制作。

2、关于切图。

关于切图有几个建议：

第一，长宽最好是3的倍数（根据android的推荐logo图标的大小是48（mdpi），72（hdpi），96（xhdpi）得出的最小公约数）。

第二，长宽最好是偶数。因为奇数在进行等比压缩的时候可能有问题。

第三，根据上面两条，如果长宽是6的倍数最理想。

第四，如果可以拉伸而不改变设计意图的情况下，比如纯色背景，则使用android的9path工具制作成.9的图片。

3、关于图标的适配。

然后接下来的一切就和设计稿没什么关系。在切好图的基础上，根据屏幕密度、像素和实际大小的比例关系。假如设计司在480\*800的分辨率下做好了设计图，并且切好图，如果你需要适配720\*1280屏幕，该怎么做？根据比例，他们的关系是2:3，于是你需要按照1.5倍比例制作图标，比如你在480\*800的设计稿上切下来一个20\*20像素的图，那么你就需要制作一个等比放大成30\*30像素的图标，这样同一个图标在480\*800的屏幕和720\*1280的屏幕上显示的实际大小才一样。同理，如果你需要适配xxhdpi则需要在20\*20的基础上制作一个等比放大成40\*40像素的图标。

4、关于图标的目录，480\*800切下来的图我们放在drawable-hdpi目录下，按照2:3放大的图标放在drawable-xhdpi目录下，按照2倍放大的图标放在drawable-xxhdpi目录下。

android会根据手机的密度优先查找对应的目录的资源，

比如408\*800分辨率下的手机如果密度是160，则自动加载drawable-hdpi这个目录下的图标，

如果720\*1280密度是240的手机自动加载drawable-xhdpi这个目录下的图标。如果没有这个文件夹，则查找和240最接近的对应密度文件夹。

三、其它

接下来要说的估计会让你失望，根据上面的步骤也不能完全解决适配的问题，只能是大概适配，而就算根据上面的步骤大概适配了，实际在手机上的效果也有出入。

比如魅族MX3的分辨率是1080\*1800，标准情况下密度是480，但是他的密度大约是524，和480接近，也就是会查找drawable-xxhdpi这个资源下的文件。也就是说你在480\*800分辨率下切图然后按两倍放大的图标在这台手机上显示的效果还是比实际的小。

而另一个要说的问题是540\*960或者640\*960，他们的密度很可能是或者接近240也可能是320。于是在480\*800的设计稿上切下来的图并且进行的适配制作，在这些手机上显示的实际大小也可能或大或小。

# Retrofit

<http://www.open-open.com/lib/view/open1453552147323.html>

# 沉浸式状态栏

## 1

**if**( Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*KITKAT* ){ // 4.4以上

// 透明状态栏

getWindow().addFlags(WindowManager.LayoutParams.*FLAG\_TRANSLUCENT\_STATUS*);

// 透明导航栏

getWindow().addFlags(WindowManager.LayoutParams.*FLAG\_TRANSLUCENT\_NAVIGATION*);

isVersionLevel = **true**;

} **else** {

isVersionLevel = **false**;

}

## 2

**if** (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*KITKAT*) {

Window window = getWindow();

/\*

\* window.setFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS

\* , WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS);

\*/

window.addFlags(WindowManager.LayoutParams.*FLAG\_TRANSLUCENT\_STATUS*);

**int** statusBarHeight = *getStatusBarHeight*(BaseActivity.**this**);

view.setPadding(0, statusBarHeight, 0, 0);

}

/\*\*

\* 用于获取状态栏的高度。 使用Resource对象获取（推荐这种方式）

\*

\* **@return** 返回状态栏高度的像素值。

\*/

**public** **static** **int** getStatusBarHeight(Context context) {

**int** result = 0;

**int** resourceId = context.getResources().getIdentifier(

"status\_bar\_height", "dimen", "android");

**if** (resourceId > 0) {

result = context.getResources().getDimensionPixelSize(resourceId);

}

**return** result;

}

# Picasso

 picasso是Square公司开源的一个Android图形缓存库，地址<http://square.github.io/picasso/>，可以实现图片下载和缓存功能。仅仅只需要一行代码就能完全实现图片的异步加载：

|  |  |
| --- | --- |
| 1 | Picasso.with(context).load("<http://i.imgur.com/DvpvklR.png>").into(imageView); |

  Picasso不仅实现了图片异步加载的功能，还解决了android中加载图片时需要解决的一些常见问题：

   1.在adapter中需要取消已经不在视野范围的ImageView图片资源的加载，否则会导致图片错位，Picasso已经解决了这个问题。

   2.使用复杂的图片压缩转换来尽可能的减少内存消耗

   3.自带内存和硬盘二级缓存功能

**特性以及示例代码：**

**ADAPTER 中的下载**：Adapter的重用会被自动检测到，Picasso会取消上次的加载

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Override public void getView(int position, View convertView, ViewGroup parent) {    SquaredImageView view = (SquaredImageView) convertView;    if (view == null) {      view = new SquaredImageView(context);    }    String url = getItem(position);    Picasso.with(context).load(url).into(view);  } |

**图片转换**：转换图片以适应布局大小并减少内存占用

|  |  |
| --- | --- |
| 1  2  3  4  5 | Picasso.with(context)    .load(url)    .resize(50, 50)    .centerCrop()    .into(imageView); |

   你还可以自定义转换：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class CropSquareTransformation implements Transformation {    @Override public Bitmap transform(Bitmap source) {      int size = Math.min(source.getWidth(), source.getHeight());      int x = (source.getWidth() - size) / 2;      int y = (source.getHeight() - size) / 2;      Bitmap result = Bitmap.createBitmap(source, x, y, size, size);      if (result != source) {        source.recycle();      }      return result;    }    @Override public String key() { return "square()"; }  } |

   将CropSquareTransformation 的对象传递给**transform**方法即可。

**Place holders-**空白或者错误占位图片：picasso提供了两种占位图片，未加载完成或者加载发生错误的时需要一张图片作为提示。

|  |  |
| --- | --- |
| 1  2  3  4  5 | Picasso.with(context)      .load(url)      .placeholder(R.drawable.user\_placeholder)      .error(R.drawable.user\_placeholder\_error)  .into(imageView); |

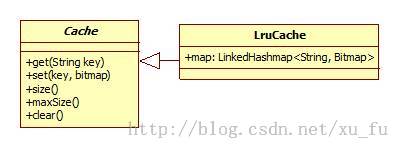
   如果加载发生错误会重复三次请求，三次都失败才会显示erro **Place holder**

**资源文件的加载**：除了加载网络图片picasso还支持加载Resources, assets, files, content providers中的资源文件。

|  |  |
| --- | --- |
| 1  2 | Picasso.with(context).load(R.drawable.landing\_screen).into(imageView1);  Picasso.with(context).load(new File(...)).into(imageView2); |

下面是picasso源码的解析（不看不影响使用）

****Cache，缓存类****



Lrucacha，主要是get和set方法，存储的结构采用了LinkedHashMap，这种map内部实现了lru算法（Least Recently Used 近期最少使用算法）。

|  |  |
| --- | --- |
| 1 | this.map = new LinkedHashMap<String, Bitmap>(0, 0.75f, true); |

最后一个参数的解释：

true if the ordering should be done based on the last access (from least-recently accessed to most-recently accessed), and false if the ordering should be the order in which the entries were inserted.

因为可能会涉及多线程，所以在存取的时候都会加锁。而且每次set操作后都会判断当前缓存区是否已满，如果满了就清掉最少使用的图形。代码如下

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | private void trimToSize(int maxSize) {          while (true) {              String key;              Bitmap value;              synchronized (this) {                  if (size < 0 || (map.isEmpty() && size != 0)) {                      throw new IllegalStateException(getClass().getName()                              + ".sizeOf() is reporting inconsistent results!");                  }                    if (size <= maxSize || map.isEmpty()) {                      break;                  }                    Map.Entry<String, Bitmap> toEvict = map.entrySet().iterator()                          .next();                  key = toEvict.getKey();                  value = toEvict.getValue();                  map.remove(key);                  size -= Utils.getBitmapBytes(value);                  evictionCount++;              }          }  } |

****Request，操作封装类****



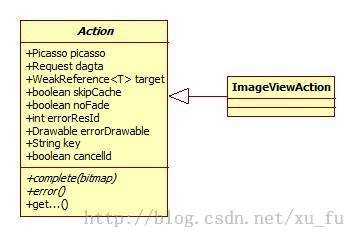
所有对图形的操作都会记录在这里，供之后图形的创建使用，如重新计算大小，旋转角度，也可以自定义变换，只需要实现Transformation，一个bitmap转换的接口。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public interface Transformation {    /\*\*     \* Transform the source bitmap into a new bitmap. If you create a new bitmap instance, you must     \* call {@link android.graphics.Bitmap#recycle()} on {@code source}. You may return the original     \* if no transformation is required.     \*/    Bitmap transform(Bitmap source);      /\*\*     \* Returns a unique key for the transformation, used for caching purposes. If the transformation     \* has parameters (e.g. size, scale factor, etc) then these should be part of the key.     \*/    String key();  } |

当操作封装好以后，会将Request传到另一个结构中Action。

****Action****

Action代表了一个具体的加载任务，主要用于图片加载后的结果回调，有两个抽象方法，complete和error，也就是当图片解析为bitmap后用户希望做什么。最简单的就是将bitmap设置给imageview，失败了就将错误通过回调通知到上层。

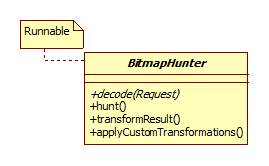


ImageViewAction实现了Action，在complete中将bitmap和imageview组成了一个PicassoDrawable，里面会实现淡出的动画效果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | @Override      public void complete(Bitmap result, Picasso.LoadedFrom from) {          if (result == null) {              throw new AssertionError(String.format(                      "Attempted to complete action with no result!\n%s", this));          }            ImageView target = this.target.get();          if (target == null) {              return;          }            Context context = picasso.context;          boolean debugging = picasso.debugging;          PicassoDrawable.setBitmap(target, context, result, from, noFade,                  debugging);            if (callback != null) {              callback.onSuccess();          }      } |

有了加载任务，具体的图片下载与解析是在哪里呢？这些都是耗时的操作，应该放在异步线程中进行，就是下面的BitmapHunter。

****BitmapHunter****



BitmapHunter是一个Runnable，其中有一个decode的抽象方法，用于子类实现不同类型资源的解析。

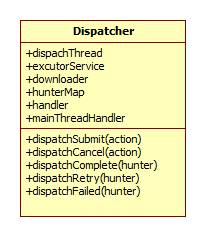
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | @Override      public void run() {          try {              Thread.currentThread()                      .setName(Utils.THREAD\_PREFIX + data.getName());                result = hunt();                if (result == null) {                  dispatcher.dispatchFailed(this);              } else {                  dispatcher.dispatchComplete(this);              }          } catch (IOException e) {              exception = e;              dispatcher.dispatchRetry(this);          } catch (Exception e) {              exception = e;              dispatcher.dispatchFailed(this);          } finally {              Thread.currentThread().setName(Utils.THREAD\_IDLE\_NAME);          }      }        abstract Bitmap decode(Request data) throws IOException;        Bitmap hunt() throws IOException {          Bitmap bitmap;            if (!skipMemoryCache) {              bitmap = cache.get(key);              if (bitmap != null) {                  stats.dispatchCacheHit();                  loadedFrom = MEMORY;                  return bitmap;              }          }            bitmap = decode(data);            if (bitmap != null) {              stats.dispatchBitmapDecoded(bitmap);              if (data.needsTransformation() || exifRotation != 0) {                  synchronized (DECODE\_LOCK) {                      if (data.needsMatrixTransform() || exifRotation != 0) {                          bitmap = transformResult(data, bitmap, exifRotation);                      }                      if (data.hasCustomTransformations()) {                          bitmap = applyCustomTransformations(                                  data.transformations, bitmap);                      }                  }                  stats.dispatchBitmapTransformed(bitmap);              }          }            return bitmap;      } |

可以看到，在decode生成原始bitmap，之后会做需要的转换transformResult和applyCustomTransformations。最后在将最终的结果传递到上层dispatcher.dispatchComplete(this)。

基本的组成元素有了，那这一切是怎么连接起来运行呢，答案是Dispatcher。

Dispatcher任务调度器

在bitmaphunter成功得到bitmap后，就是通过dispatcher将结果传递出去的，当然让bitmaphunter执行也要通过Dispatcher。



Dispatcher内有一个HandlerThread，所有的请求都会通过这个thread转换，也就是请求也是异步的，这样应该是为了Ui线程更加流畅，同时保证请求的顺序，因为handler的消息队列。  
外部调用的是dispatchXXX方法，然后通过handler将请求转换到对应的performXXX方法。  
例如生成Action以后就会调用dispather的dispatchSubmit()来请求执行，

|  |  |
| --- | --- |
| 1  2  3 | void dispatchSubmit(Action action) {          handler.sendMessage(handler.obtainMessage(REQUEST\_SUBMIT, action));      } |

handler接到消息后转换到performSubmit方法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | void performSubmit(Action action) {          BitmapHunter hunter = hunterMap.get(action.getKey());          if (hunter != null) {              hunter.attach(action);              return;          }            if (service.isShutdown()) {              return;          }            hunter = forRequest(context, action.getPicasso(), this, cache, stats,                  action, downloader);          hunter.future = service.submit(hunter);          hunterMap.put(action.getKey(), hunter);      } |

这里将通过action得到具体的BitmapHunder，然后交给ExecutorService执行。

下面是Picasso.with(context).load("http://i.imgur.com/DvpvklR.png").into(imageView)的过程，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | public static Picasso with(Context context) {          if (singleton == null) {              singleton = new Builder(context).build();          }          return singleton;      }        public Picasso build() {              Context context = this.context;                if (downloader == null) {                  downloader = Utils.createDefaultDownloader(context);              }              if (cache == null) {                  cache = new LruCache(context);              }              if (service == null) {                  service = new PicassoExecutorService();              }              if (transformer == null) {                  transformer = RequestTransformer.IDENTITY;              }                Stats stats = new Stats(cache);                Dispatcher dispatcher = new Dispatcher(context, service, HANDLER,                      downloader, cache, stats);                return new Picasso(context, dispatcher, cache, listener,                      transformer, stats, debugging);          } |

在Picasso.with()的时候会将执行所需的所有必备元素创建出来，如缓存cache、执行executorService、调度dispatch等，在load()时创建Request，在into()中创建action、bitmapHunter，并最终交给dispatcher执行。

# Js与java互相调用

自定义view 文件 JsJava

# Palette 调色板

## Palette是什么？

http://www.07net01.com/2015/04/813754.html

它能让你从图像中提取突出的颜色。这个类能提取以下突出的颜色：

Vibrant(充满活力的)

Vibrant dark(充满活力的黑)

Vibrant light(充满活力的亮)

Muted(柔和的)

Muted dark(柔和的黑)

Muted lighr(柔和的亮)

## 如何使用？

要提取这些颜色，在你加载图片的后台线程中传递一个位图对象给[Palette.generate()](http://developer.android.com/reference/android/support/v7/graphics/Palette.html" \l "generate(android.graphics.Bitmap)" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)静态方法。如果你不适用线程，则调用[Palette.generateAsync()](http://blog.csdn.net/bbld_/article/details/android.support.v7.graphics.Palette.PaletteAsyncListener)" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)方法并且提供一个监听器去替代。

你可以在[Palette](http://developer.android.com/reference/android/support/v7/graphics/Palette.html" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)类中使用getter方法来从检索突出的颜色，比如Palette.getVibrantColor。

如果是Android Studio 要在你的项目中使用Palette类，增加下面的[Gradle依赖](http://developer.android.com/sdk/installing/studio-build.html" \l "dependencies" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)到你的程序的模块(module)中：

**[java]** [view plain](http://blog.csdn.net/cym492224103/article/details/41626889" \o "view plain" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)[copy](http://blog.csdn.net/cym492224103/article/details/41626889" \o "copy" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)

1. dependencies {
2. ...
3. compile 'com.android.support:palette-v7:21.0.+'
4. }

如果是Eclipse首先我们找到sdk/extras/android/support/v7/palette/libs/android-support-v7-palette.jar导入我们的工程。

然后使用generateAsync方法传入当前图片的bitmap，在传入一个监听，在监听里面我们拿到图片上颜色充满活力的颜色，最后设置标题背景和字体的颜色，代码如下：

**[java]** [view plain](http://blog.csdn.net/cym492224103/article/details/41626889" \o "view plain" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)[copy](http://blog.csdn.net/cym492224103/article/details/41626889" \o "copy" \t "http://blog.csdn.net/zhangweiwtmdbf/article/details/_blank)

1. Palette.generateAsync(bitmap,
2. **new** Palette.PaletteAsyncListener() {
3. @Override
4. **public** **void** onGenerated(Palette palette) {
5. Palette.Swatch vibrant =
6. palette.getVibrantSwatch();
7. **if** (swatch != **null**) {
8. // If we have a vibrant color
9. // update the title TextView
10. titleView.setBackgroundColor(
11. vibrant.getRgb());
12. titleView.setTextColor(
13. vibrant.getTitleTextColor());
14. }
15. }
16. });

# connection的三种链接方法

**三种连接方法**

// 方法一 URL url = new URL("http://www.sina.com.cn");

URLConnection urlcon = url.openConnection();

InputStream is = urlcon.getInputStream();

// 方法二 URL url = new URL("http://www.yhfund.com.cn");

HttpURLConnection urlcon = (HttpURLConnection)url.openConnection();

InputStream is = urlcon.getInputStream();

//方法三 URL url = new URL("http://www.yhfund.com.cn");

InputStream is = url.openStream();

**具体例子**

long begintime = System.currentTimeMillis();

URL url = new URL("http://www.yhfund.com.cn");

HttpURLConnection urlcon = (HttpURLConnection)url.openConnection();

urlcon.connect(); //获取连接 InputStream is = urlcon.getInputStream();

BufferedReader buffer = new BufferedReader(new InputStreamReader(is));

StringBuffer bs = new StringBuffer();

String l = null;

while((l=buffer.readLine())!=null){

bs.append(l).append("/n");

}

System.out.println(bs.toString());

//System.out.println(" content-encode："+urlcon.getContentEncoding());

//System.out.println(" content-length："+urlcon.getContentLength());

//System.out.println(" content-type："+urlcon.getContentType());

//System.out.println(" date："+urlcon.getDate());

System.out.println("总共执行时间为："+(System.currentTimeMillis()-begintime)+"毫秒");

# [初步使用EventBus](http://blog.csdn.net/harvic880925/article/details/40660137)

1、下载EventBus的类库  
源码：[https://github.com/greenrobot/EventBus](https://github.com/greenrobot/EventBus" \t "http://blog.csdn.net/harvic880925/article/details/_blank)

2、基本使用

（1）自定义一个类，可以是空类，比如：

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_256](https://code.csdn.net/snippets/504371)[IMG_257](https://code.csdn.net/snippets/504371/fork)

1. **public** **class** AnyEventType {
2. **public** AnyEventType(){}
3. }

（2）在要接收消息的页面注册：

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_258](https://code.csdn.net/snippets/504371)[IMG_259](https://code.csdn.net/snippets/504371/fork)

1. eventBus.register(**this**);

（3）发送消息

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_260](https://code.csdn.net/snippets/504371)[IMG_261](https://code.csdn.net/snippets/504371/fork)

1. eventBus.post(**new** AnyEventType event);

（4）接受消息的页面实现(共有四个函数，各功能不同，这是其中之一，可以选择性的实现，这里先实现一个)：

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_262](https://code.csdn.net/snippets/504371)[IMG_263](https://code.csdn.net/snippets/504371/fork)

1. **public** **void** onEvent(AnyEventType event) {}

（5）解除注册

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_264](https://code.csdn.net/snippets/504371)[IMG_265](https://code.csdn.net/snippets/504371/fork)

1. eventBus.unregister(**this**);

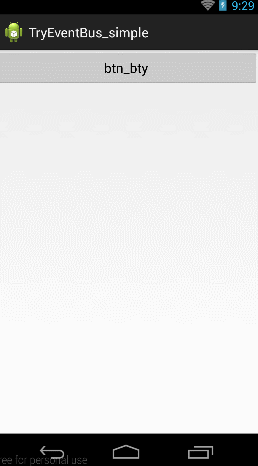
顺序就是这么个顺序，可真正让自己写，估计还是云里雾里的，下面举个例子来说明下。

首先，在EventBus中，获取实例的方法一般是采用EventBus.getInstance()来获取默认的EventBus实例，当然你也可以new一个又一个，个人感觉还是用默认的比较好，以防出错。

## ****二、实战****

先给大家看个例子：

当击btn\_try按钮的时候，跳到第二个Activity，当点击第二个activity上面的First Event按钮的时候向第一个Activity发送消息，当第一个Activity收到消息后，一方面将消息Toast显示，一方面放入textView中显示。



按照下面的步骤，下面来建这个工程：

### 1、基本框架搭建

想必大家从一个Activity跳转到第二个Activity的程序应该都会写，这里先稍稍把两个Activity跳转的代码建起来。后面再添加EventBus相关的玩意。

**MainActivity布局（activity\_main.xml）**

**[html]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_267](https://code.csdn.net/snippets/504371)[IMG_268](https://code.csdn.net/snippets/504371/fork)

1. **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android"
2. xmlns:tools="http://schemas.android.com/tools"
3. android:layout\_width="match\_parent"
4. android:layout\_height="match\_parent"
5. android:orientation="vertical"**>**
7. **<Button**
8. android:id="@+id/btn\_try"
9. android:layout\_width="match\_parent"
10. android:layout\_height="wrap\_content"
11. android:text="btn\_bty"**/>**
12. **<TextView**
13. android:id="@+id/tv"
14. android:layout\_width="wrap\_content"
15. android:layout\_height="match\_parent"**/>**
17. **</LinearLayout>**

**新建一个Activity，SecondActivity布局（activity\_second.xml）**

**[html]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_269](https://code.csdn.net/snippets/504371)[IMG_270](https://code.csdn.net/snippets/504371/fork)

1. **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android"
2. xmlns:tools="http://schemas.android.com/tools"
3. android:layout\_width="match\_parent"
4. android:layout\_height="match\_parent"
5. android:orientation="vertical"
6. tools:context="com.harvic.try\_eventbus\_1.SecondActivity" **>**
8. **<Button**
9. android:id="@+id/btn\_first\_event"
10. android:layout\_width="match\_parent"
11. android:layout\_height="wrap\_content"
12. android:text="First Event"**/>**
14. **</LinearLayout>**

**MainActivity.java （点击btn跳转到第二个Activity）**

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_271](https://code.csdn.net/snippets/504371)[IMG_272](https://code.csdn.net/snippets/504371/fork)

1. **public** **class** MainActivity **extends** Activity {
3. Button btn;
5. @Override
6. **protected** **void** onCreate(Bundle savedInstanceState) {
7. **super**.onCreate(savedInstanceState);
8. setContentView(R.layout.activity\_main);
10. btn = (Button) findViewById(R.id.btn\_try);
12. btn.setOnClickListener(**new** View.OnClickListener() {
14. @Override
15. **public** **void** onClick(View v) {
16. // TODO Auto-generated method stub
17. Intent intent = **new** Intent(getApplicationContext(),
18. SecondActivity.**class**);
19. startActivity(intent);
20. }
21. });
22. }
24. }

到这，基本框架就搭完了，下面开始按步骤使用EventBus了。

### 2、新建一个类FirstEvent

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_273](https://code.csdn.net/snippets/504371)[IMG_274](https://code.csdn.net/snippets/504371/fork)

1. **package** com.harvic.other;
3. **public** **class** FirstEvent {
5. **private** String mMsg;
6. **public** FirstEvent(String msg) {
7. // TODO Auto-generated constructor stub
8. mMsg = msg;
9. }
10. **public** String getMsg(){
11. **return** mMsg;
12. }
13. }

这个类很简单，构造时传进去一个字符串，然后可以通过getMsg()获取出来。

### 3、在要接收消息的页面注册EventBus：

在上面的GIF图片的演示中，大家也可以看到，我们是要在MainActivity中接收发过来的消息的，所以我们在MainActivity中注册消息。

通过我们会在OnCreate()函数中注册EventBus，在OnDestroy（）函数中反注册。所以整体的注册与反注册的代码如下：

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_275](https://code.csdn.net/snippets/504371)[IMG_276](https://code.csdn.net/snippets/504371/fork)

1. **package** com.example.tryeventbus\_simple;
3. **import** com.harvic.other.FirstEvent;
5. **import** de.greenrobot.event.EventBus;
6. **import** android.app.Activity;
7. **import** android.content.Intent;
8. **import** android.os.Bundle;
9. **import** android.util.Log;
10. **import** android.view.View;
11. **import** android.widget.Button;
12. **import** android.widget.TextView;
13. **import** android.widget.Toast;
15. **public** **class** MainActivity **extends** Activity {
17. Button btn;
18. TextView tv;
20. @Override
21. **protected** **void** onCreate(Bundle savedInstanceState) {
22. **super**.onCreate(savedInstanceState);
23. setContentView(R.layout.activity\_main);
24. //注册EventBus
25. EventBus.getDefault().register(**this**);
27. btn = (Button) findViewById(R.id.btn\_try);
28. tv = (TextView)findViewById(R.id.tv);
30. btn.setOnClickListener(**new** View.OnClickListener() {
32. @Override
33. **public** **void** onClick(View v) {
34. // TODO Auto-generated method stub
35. Intent intent = **new** Intent(getApplicationContext(),
36. SecondActivity.**class**);
37. startActivity(intent);
38. }
39. });
40. }
41. @Override
42. **protected** **void** onDestroy(){
43. **super**.onDestroy();
44. EventBus.getDefault().unregister(**this**);//反注册EventBus
45. }
46. }

### 4、发送消息

发送消息是使用EventBus中的Post方法来实现发送的，发送过去的是我们新建的类的实例！

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_277](https://code.csdn.net/snippets/504371)[IMG_278](https://code.csdn.net/snippets/504371/fork)

1. EventBus.getDefault().post(**new** FirstEvent("FirstEvent btn clicked"));

完整的SecondActivity.**[Java](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "http://blog.csdn.net/harvic880925/article/details/_blank)**的代码如下：

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_279](https://code.csdn.net/snippets/504371)[IMG_280](https://code.csdn.net/snippets/504371/fork)

1. **package** com.example.tryeventbus\_simple;
3. **import** com.harvic.other.FirstEvent;
5. **import** de.greenrobot.event.EventBus;
6. **import** android.app.Activity;
7. **import** android.os.Bundle;
8. **import** android.view.View;
9. **import** android.widget.Button;
11. **public** **class** SecondActivity **extends** Activity {
12. **private** Button btn\_FirstEvent;
14. @Override
15. **protected** **void** onCreate(Bundle savedInstanceState) {
16. **super**.onCreate(savedInstanceState);
17. setContentView(R.layout.activity\_second);
18. btn\_FirstEvent = (Button) findViewById(R.id.btn\_first\_event);
20. btn\_FirstEvent.setOnClickListener(**new** View.OnClickListener() {
22. @Override
23. **public** **void** onClick(View v) {
24. // TODO Auto-generated method stub
25. EventBus.getDefault().post(
26. **new** FirstEvent("FirstEvent btn clicked"));
27. }
28. });
29. }
30. }

### 5、接收消息

接收消息时，我们使用EventBus中最常用的onEventMainThread（）函数来接收消息，具体为什么用这个，我们下篇再讲，这里先给大家一个初步认识，要先能把EventBus用起来先。

在MainActivity中重写onEventMainThread（FirstEvent event），参数就是我们自己定义的类：

在收到Event实例后，我们将其中携带的消息取出，一方面Toast出去，一方面传到TextView中；

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_281](https://code.csdn.net/snippets/504371)[IMG_282](https://code.csdn.net/snippets/504371/fork)

1. **public** **void** onEventMainThread(FirstEvent event) {
3. String msg = "onEventMainThread收到了消息：" + event.getMsg();
4. Log.d("harvic", msg);
5. tv.setText(msg);
6. Toast.makeText(**this**, msg, Toast.LENGTH\_LONG).show();
7. }

完整的MainActiviy代码如下：

**[java]** [view plain](http://blog.csdn.net/harvic880925/article/details/40660137" \o "view plain) [copy](http://blog.csdn.net/harvic880925/article/details/40660137" \o "copy) [IMG_283](https://code.csdn.net/snippets/504371)[IMG_284](https://code.csdn.net/snippets/504371/fork)

1. **package** com.example.tryeventbus\_simple;
3. **import** com.harvic.other.FirstEvent;
5. **import** de.greenrobot.event.EventBus;
6. **import** android.app.Activity;
7. **import** android.content.Intent;
8. **import** android.os.Bundle;
9. **import** android.util.Log;
10. **import** android.view.View;
11. **import** android.widget.Button;
12. **import** android.widget.TextView;
13. **import** android.widget.Toast;
15. **public** **class** MainActivity **extends** Activity {
17. Button btn;
18. TextView tv;
20. @Override
21. **protected** **void** onCreate(Bundle savedInstanceState) {
22. **super**.onCreate(savedInstanceState);
23. setContentView(R.layout.activity\_main);
25. EventBus.getDefault().register(**this**);
27. btn = (Button) findViewById(R.id.btn\_try);
28. tv = (TextView)findViewById(R.id.tv);
30. btn.setOnClickListener(**new** View.OnClickListener() {
32. @Override
33. **public** **void** onClick(View v) {
34. // TODO Auto-generated method stub
35. Intent intent = **new** Intent(getApplicationContext(),
36. SecondActivity.**class**);
37. startActivity(intent);
38. }
39. });
40. }
42. **public** **void** onEventMainThread(FirstEvent event) {
44. String msg = "onEventMainThread收到了消息：" + event.getMsg();
45. Log.d("harvic", msg);
46. tv.setText(msg);
47. Toast.makeText(**this**, msg, Toast.LENGTH\_LONG).show();
48. }
50. @Override
51. **protected** **void** onDestroy(){
52. **super**.onDestroy();
53. EventBus.getDefault().unregister(**this**);
54. }
55. }

好了，到这，基本上算初步把EventBus用起来了，下篇再讲讲EventBus的几个函数，及各个函数间是如何识别当前如何调用哪个函数的。

**如果我的文章有帮到你，请关注哦。**

**源码地址：**[http://download.csdn.net/detail/harvic880925/8111357](http://download.csdn.net/detail/harvic880925/8111357" \t "http://blog.csdn.net/harvic880925/article/details/_blank)

**请大家尊重原创者版权，转载请标明出处：**[http://blog.csdn.net/harvic880925/article/details/40660137](http://blog.csdn.net/harvic880925/article/details/40660137" \t "http://blog.csdn.net/harvic880925/article/details/_blank) **谢谢！**

# Retrolambda

<http://www.open-open.com/lib/view/open1433898197176.html>

# SwipeRefreshLayout

Google自己推出的下拉加载的软件

# 相见恨晚的类

作者：StephenLee  
链接：http://www.zhihu.com/question/33636939/answer/57171337  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

1、Throwable接口中的**getStackTrace()**方法（或者Thread类的**getStackTrace()**方法），根据这个方法可以得到函数的逐层调用地址，其返回值为StackTraceElement[]；  
  
2、**StackTraceElement**类，其中四个方法getClassName()，getFileName()，getLineNumber()，getMethodName()在调试程序打印Log时非常有用；  
  
3、**UncaughtExceptionHandler**接口，再好的代码异常难免，利用此接口可以对未捕获的异常善后；  
使用参见：[Android使用UncaughtExceptionHandler捕获全局异常](//link.zhihu.com/?target=http://blog.csdn.net/whu_zhangmin/article/details/42294141" \t "_blank)  
  
4、Resources类中的**getIdentifier(name, defType, defPackage)**方法，根据资源名称获取其ID，做UI时经常用到；  
  
5、View中的**isShown()**方法，以前都是用view.getVisibility() == View.VISIBLE来判断的(╯□╰)；（谢评论提醒，这里面其实有一个坑：[【android】view.isShown ()的用法](//link.zhihu.com/?target=http://blog.csdn.net/u011494050/article/details/41700207" \t "_blank)）  
  
6、**Arrays**类中的一系列关于数组操作的工具方法：binarySearch()，asList()，equals()，sort()，toString()，copyOfRange()等；  
**Collections**类中的一系列关于集合操作的工具方法：sort()，reverse()等；  
  
7、android.text.format.Formatter类中**formatFileSize(Context, long)**方法，用来格式化文件Size（B → KB → MB → GB）；  
  
8、**android.media.ThumbnailUtils**类，用来获取媒体（图片、视频）缩略图；  
  
9、String类中的**format(String, Object...)**方法，用来格式化strings.xml中的字符串（多谢 [@droider An](//www.zhihu.com/people/3f8b60ccfd562eb0b481f4c1adffd95c) 提示：Context类中getString(int, Object... )方法用起来更加方便）；  
  
10、View类中的三个方法：**callOnClick()**，**performClick()**，**performLongClick()**，用于触发View的点击事件；  
  
11、TextUtils类中的**isEmpty(CharSequence)**方法，判断字符串是否为null或""；  
  
12、TextView类中的**append(CharSequence)**方法，添加文本。一些特殊文本直接用+连接会变成String；  
  
13、View类中的**getDrawingCache()**等一系列方法，目前只知道可以用来截图；  
  
14、**DecimalFormat**类，用于字串格式化包括指定位数、百分数、科学计数法等；  
  
15、System类中的**arraycopy(src, srcPos, dest, destPos, length)**方法，用来copy数组；  
  
16、Fragment类中的**onHiddenChanged(boolean)**方法，使用FragmentTransaction中的hide()，show()时貌似Fragment的其它生命周期方法都不会被调用，太坑爹！  
  
17、Activity类中的**onWindowFocusChanged(boolean)**，**onNewIntent(intent)**等回调方法；  
  
18、View类中的**getLocationInWindow(int[])**方法和**getLocationOnScreen(int[])**方法，获取View在窗口/屏幕中的位置；  
  
19、TextView类中的**setTransformationMethod(TransformationMethod)**方法，可用来实现“显示密码”功能；  
  
20、**TextWatcher**接口，用来监听文本输入框内容的改变，可用来实现一系列具有特殊功能的文本输入框；  
  
21、View类中的**setSelected(boolean)**方法结合android:state\_selected=""用来实现图片选中效果；  
  
22、Surface设置透明：**SurfaceView.setZOrderOnTop(true);  
SurfaceView.getHolder().setFormat(PixelFormat.TRANSLUCENT);**但是会挡住其它控件；  
  
23、ListView或GridView类中的**setFastScrollEnabled(boolean)**方法，用来设置快速滚动滑块是否可见，当然前提是item够多；  
  
24、**PageTransformer**接口，用来自定义ViewPager页面切换动画，用**setPageTransformer(boolean, PageTransformer)**方法来进行设置；  
  
25、apache提供的一系列jar包：**commons-lang.jar**，**commons-collections.jar**，**commons-beanutils.jar**等，里面很多方法可能是你曾经用几十几百行代码实现过的，但是执行效率或许要差很多，比如：ArrayUtils，StringUtils……；  
  
26、**AndroidTestCase**类，Android单元测试，在AndroidStudio中使用非常方便；  
  
27、TextView类的**setKeyListener(KeyListener)**方法；  
其中DigitsKeyListener类，使用getInstance(String accepted)方法即可指定EditText可输入字符集；  
  
28、**ActivityLifecycleCallbacks**接口，用于在Application类中监听各Activity的状态变化；  
  
29、Context类中的**createPackageContext(packageName, flags)**方法，可用来获取指定包名应用程序的Context对象。