手机应用平台软件开发

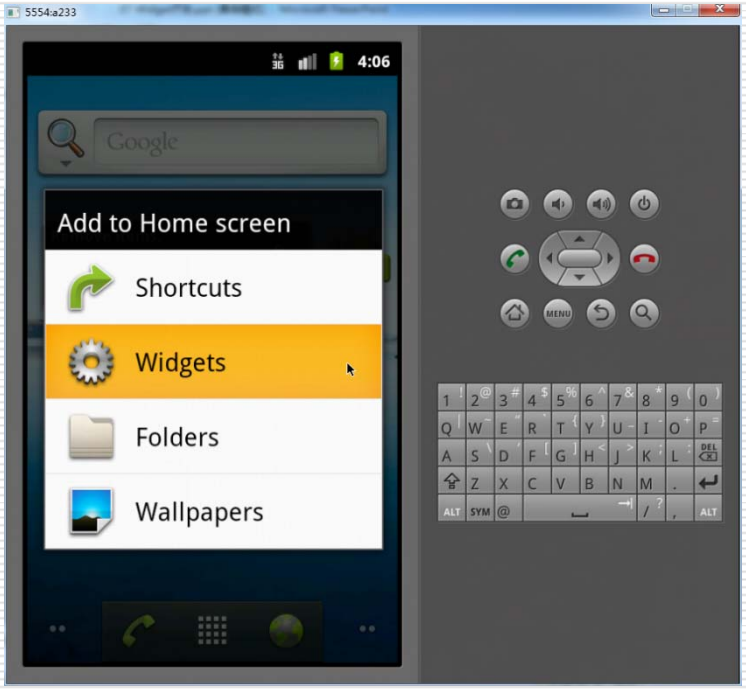
7、Widget开发

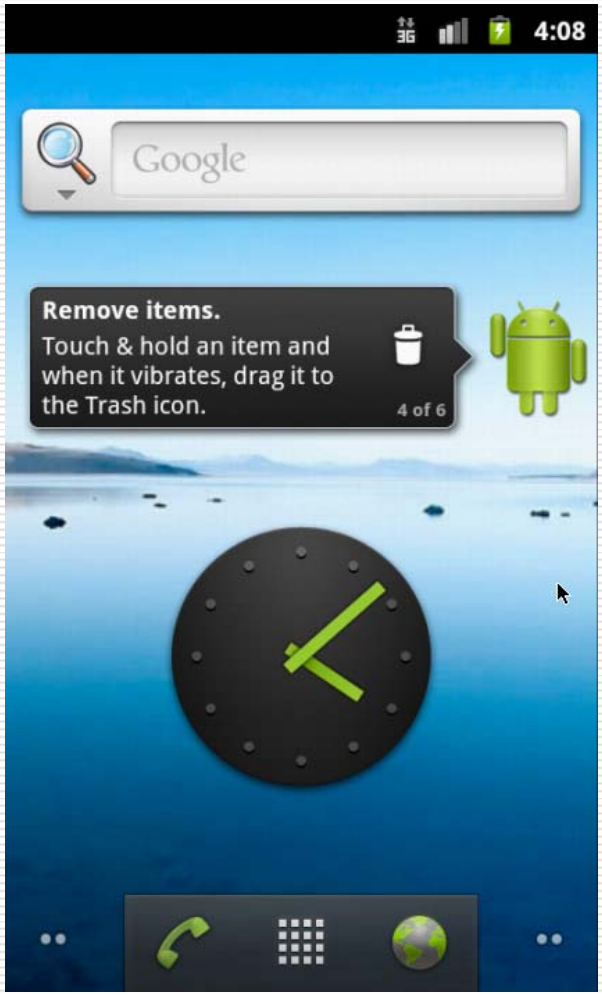
刘宁
Email：lnsongz@mail.sysu.edu.cn

Widget的开发

SUN YAT-SEN UNIVERSITY

App Widgets 是小型应用程序视图嵌入在另外的应用程序中，并且周期接收更新。这些视图被称作Widget





Widget的开发

- Widget是Android 1.5的一个新特性，允许程序显示一些常用而又重要的信息在用户的Home screen(桌面主屏)上

- 标准的Android系统映像包含了一些示例widgets包括指针时钟、音乐播放器和其他工具如Google搜索栏

- Widget和标准的Apps相比没有太大的区别，更多的是在UI上的处理，逻辑执行设计成服务，具备更稳定和更高的可靠性

Widget的开发

- 直接显示到桌面上的小控件，定期更新
- 每个Widget就是一个广播接收器
- 显示的内容封装成RemoteViews对象

Widget的开发

- Widget不是运行在自己进程里，而是宿主进程，所以交互需要处理App Widget广播。AppWidgetProvider只接收和这个App Widget相关的事件广播，比如这个App Widget被更新，删除，启用，以及禁用。

Widget的开发

- 每个Widget就是一个BroadcastReceiver，它们用XML metadata来描述Widget细节。AppWidget framework通过Broadcast intents 和Widget通信，Widget更新使用RemotesViews来发送。RemotesViews被包装成一个layout和特定的内容来显示到桌面上。

Widget开发

1.定义Widget的广播

2.定义Widget配置文件

3.定义Widget布局文件

4.定时发送广播给widget在widget的接收器做处理

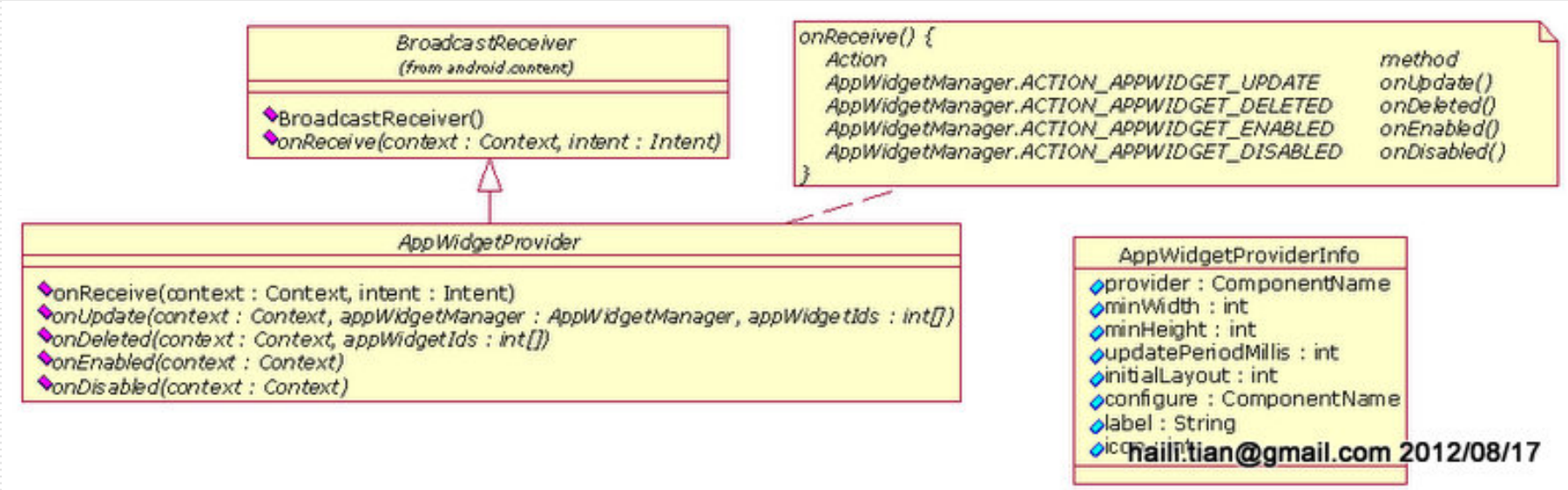
Widget的开发

包含三个XML文件，一个class…

1. **Widget**的布局文件；
2. **AndroidManifest**文件
3. **AppWidgtProviderInfo**：用于描述App Widget基本信息，它是AppWidget在XML中定义的一个元数据；
4. **AppWidgetProvider**：定义了一些基于广播事件的基本方法。用于完成业务逻辑操作；

AppWidgetProvider

继承自 BroadcastRecevier，这些广播事件发生时，AppWidgetProvider 将通过自己的方法来处理，这些方法包括：update、enable、disable 和 delete 时接收通知。其中，onUpdate、onReceive 是最常用到的方法。



- **onUpdate**

  间隔性更新App Widget，间隔时间在AppWidgetProviderInfo 里的updatePeriodMillis属性定义。该方法也会在添加App Widget时被调用，进行widget配置。

- **onDeleted**

  当App Widget从宿主中删除时被调用。

- **onEnabled**

  当Widget实例第一次创建时被调用。若用户添加两个同一个App Widget实例，只在第一次被调用。适用于需要打开一个新的数据库或者执行其他对于所有的App Widget实例只需要发生一次的处理。

- **onDisabled**

  当App Widget的最后一个实例被从宿主中删除时被调用。例如在onDisabled中做一些清理工作，比如关掉后台服务。

- **onReceive**

  接收到每个广播时都会被调用，而且在上面的回调函数之前。

AppWidgetManger：

负责管理 AppWidget，向 AppwidgetProvider 发送通知。

RemoteViews

可以在其他应用进程中运行的类，向AppWidgetProvider 发送通知。描述一个view,而该view是在另外一个进程显示的。AppWidgetProvider是一个BrocaseReceiver,只是接受到Enable,Update,disale,delete这些message,而真正显示界面的是AppWidgetHostView(这是在Launcher里面实现的) 中间就是通过RemoteView来沟通。通过RemoteView告诉Launcher想要的AppWidget是长什么样。

Android Manifest中声明App Widget

```
<!-- Broadcast Receiver that will process AppWidget updates -->
      <receiver android:name=".Widget“
      <!-- “//类名为Widget -->
      android:label="@string/widget_name">
        <intent-filter>
          <action android:name=
            "android.appwidget.action.APPWIDGET_UPDATE" />
      <!– 能够接收APPWIDGET_UPDATE广播-->
        </intent-filter>
        <meta-data android:name="android.appwidget.provider"
          android:resource="@xml/appwidget_info.xml" />
      <!--描述Widget的资源文件是在xml目录下的appwidget_info.xml文件-->
</receiver>
```

metadata:元数据，描述数据的数据，为xml目录下的widget.xml文件

AppWidgetProvderInfo

描述 AppWidget 的大小、更新频率和初始界面等信息，一般以XML 文件形式存在于应用的 res/xml/目录下。

```
<!--放在 res/xml 文件夹中命名为appwidget_info.xml-->
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
   android:minWidth="294dp"
   android:minHeight="72dp"
   android:updatePeriodMillis="86400000"
   android:initialLayout="@layout/appwidget"
   android:configure="AppWidgetConfigure" >
</appwidget-provider>
```

minWidth: 定义Wdiget组件的宽度　　minHeight: 定义Wdiget组件的高度

updatePeriodMillis: 更新的时间周期　　initialLayout: Widget的布局文件

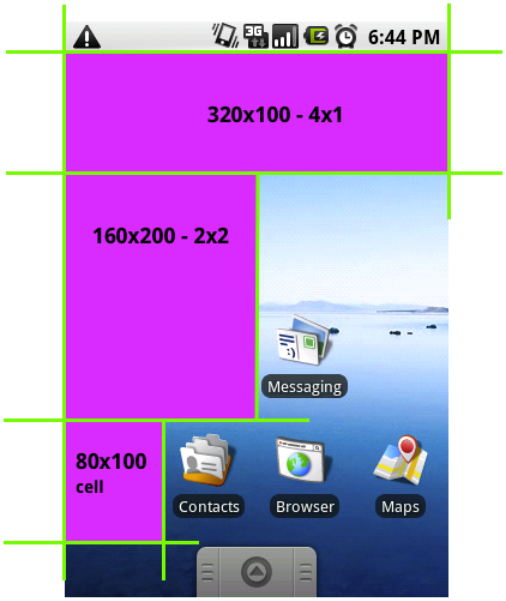
configure: 如果需要在启动前先启动一个Activity进行设置，在这里给出Activity的完整类名

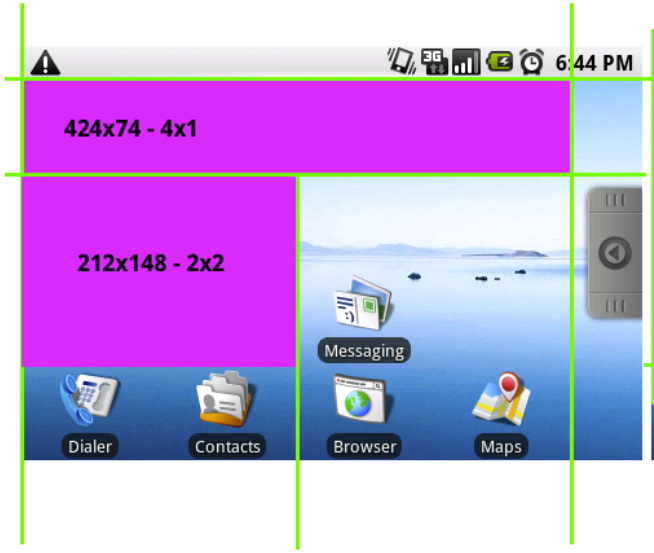
AppWidgetProviderInfo

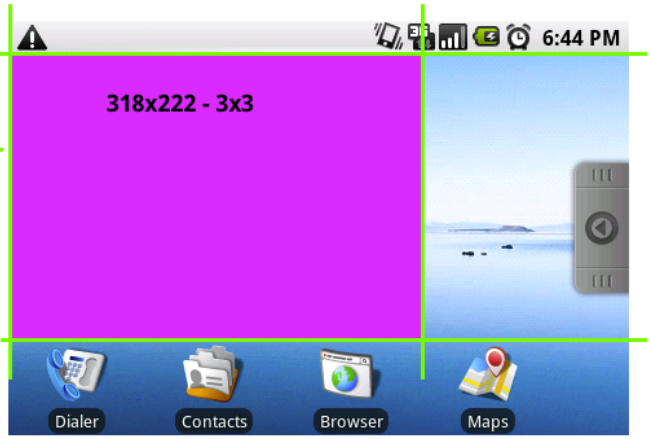
- 桌面的每格宽度和高度为74dip，Widget最小高宽都是72DP，需要注意的是这是最小宽高，一般要求其大小等于74乘以占用格数减去2，这里是只占用一格，所以是72。
- Android框架每隔一段时间，会回调AppWidgetProvider类的onUpdate()事件，更新间隔是86400000，updatePeriodMillis系统是有限制的，最短周期不能低于30分钟，就是1800000毫秒。以前android的版本设定为1毫秒为单位，现在版本为了省电，更新时间为30～60分钟，所以现在设定30分钟以内的更新意义不大，系统默认为30～60分钟更新
- android:initialLayout属性指定此App Widget的UI layout定义，”@layout/main”表示layout目录下的main.xml文件

设置合适的大小

SUN YAT-SEN UNIVERSITY







实现类 AppWidgetProvider

Defines the basic methods that allow you to programmatically interface with the App Widget, based on broadcast events. Through it, you will receive broadcasts when the App Widget is updated, enabled, disabled and deleted.

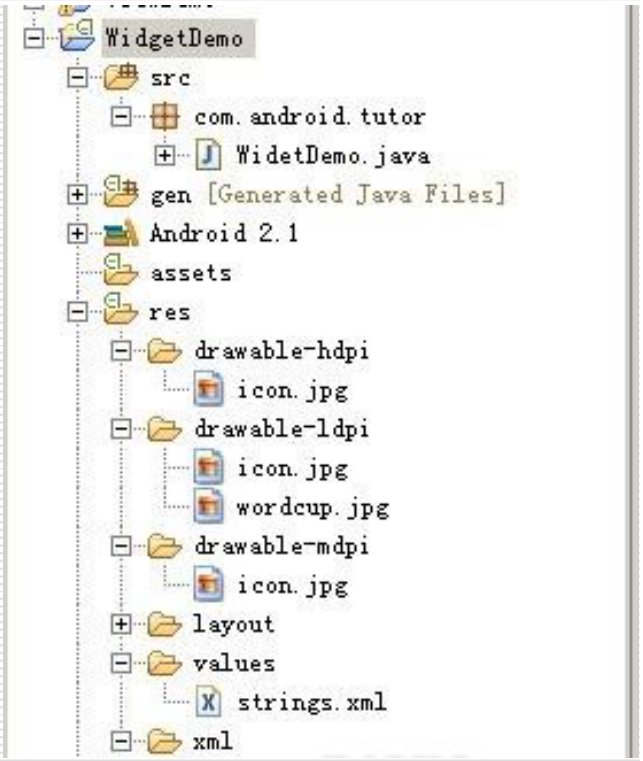
实现AppWidgetProvider的子类，并至少override onUpdate()方法[非必须，但是如果不这样做，该AppWidgetProvider就没有提供任何内容，也就不是AppWidgetProvider了]；

Widget的开发

典型的Android Widget有三个主要组件，一个边框、一个框架和图形控件以及其他元素。

1. 新建一个Android工程命名为:WidgetDemo.
2. 准备素材，一个是Widget的图标，一个是Widget的背景。存放目录如图:
3. 修改string.xml文件

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="hello">Hello World, WidetDemo!</string>
        <string name="app_name">DaysToWorldCup</string>

</resources>
```



4、建立Widget内容提供者文件，在res下建立xml文件夹，并且新建一个widget_provider.xml代码入下:

```
<?xml  version="1.0"  encoding="utf-8"?>
<appwidget-provider  xmlns:android="http://schemas.android.com/apk/res/android"
        android:minWidth="72dip"
        android:minHeight="72dip"
        android:updatePeriodMillis="86400000"
        android:initialLayout="@layout/main"  />
```

其中android:updatePeriodMillis="0" 是指自动更新的时间间隔。

五、修改main.xml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:background="@drawable/wordcup"  >
<TextView
        android:id="@+id/wordcup"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:textSize="12px"
        android:textColor="#ff0000"  />
</LinearLayout>
```

Widget的开发

6、修改WidgetDemo.java代码

```
public class WidetDemo extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,int[]
appWidgetIds) { }
```

7、修改配置文件AndroidManifest.xml

8、添加到桌面（widget的启动与普通程序不同，它不会在程序列表中显示，而是要长按，在桌面在弹出的列表中选择Widgets项目）

问题：如何处理Widget事件？

Working with remote views

- 主屏幕实际上是一个系统程序，因为安全原因，程序员不容易直接修改其运行代码
- Android提供给用户程序访问主屏幕和修改特定区域内容的方法：RemoteView架构
- RemoteView架构允许用户程序更新主屏幕的View
  - 点击Widget激活点击事件
  - Android会将其转发给用户程序，由AppWidgetProviders类处理
  - 用户程序可更新主屏幕Widget.

Widget事件处理

配置文件添加：

```
<receiver android:label="@string/app_name"  //Widget 标签名称
        android:name=".LunchWidget">        //
        <intent-filter>
        <action
        android:name="android.appwidget.action.APPWIDGET_UPDATE">
        </action>
        <action
        android:name="com.lunchtime.LunchWidget.CLICK">
         </action>
         </intent-filter>
         <meta-data android:name="android.appwidget.provider"
           android:resource="@xml/widget_provider">
         </meta-data>
   </receiver>
```

<receiver>标签，指定android:name属性为主要的provider类，即“ LunchWidget”，注意「.」表示后面的字符串为一个「类别名」，不要忽略了这个重要的小数点

Widget事件处理

SUN YAT-SEN UNIVERSITY

在onUpdate添加代码定义并发送事件

```
public class LunchWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[]
    appWidgetIds)
    {
        super.onUpdate(context, appWidgetManager, appWidgetIds);
        RemoteViews updateViews = new RemoteViews(context.getPackageName(),
        R.layout.widget);
        Intent i = new Intent("com.lunchtime.LunchWidget.CLICK");
        PendingIntent pi = PendingIntent.getBroadcast(context, 0, i, 0);
        updateViews.setOnClickPendingIntent(R.id.widgetbutton, pi);
        ComponentName me=new ComponentName(context, LunchWidget.class);
        appWidgetManager.updateAppWidget(me, updateViews);}
```

Widget事件处理

- 添加**onReceive()**处理事件

```
@Override
public void onReceive(Context context, Intent intent) {

    super.onReceive(context, intent);

    if (intent.getAction().equals("com.lunchtime.LunchWidget.CLICK")) {

    Toast.makeText(context, "Opps!!You hit me!!", Toast.LENGTH_SHORT)
    .show();

    }
}
```

Gallery组件综合例子

SUN YAT-SEN UNIVERSITY

Gallery组件可以横向显示一个图像列表，当单击当前图像的后一个图像时，这个图像列表会向左移动一格，当单击当前图像的前一个图像时，这个图像列表会向右移动一样。也可以通过拖动的方式来向左和向右移动图像列表。

1、横向滑动（Gallery）

2、显示相应图片（ImageSwitcher）

Gallery组件综合例子

Gallery组件的使用

1. 拷贝相关文件，并将这些文件都放在**res\drawable**目录中（大图、缩略图）；
2. 引入资源：将这些图像资源ID都保存在int[]组中：

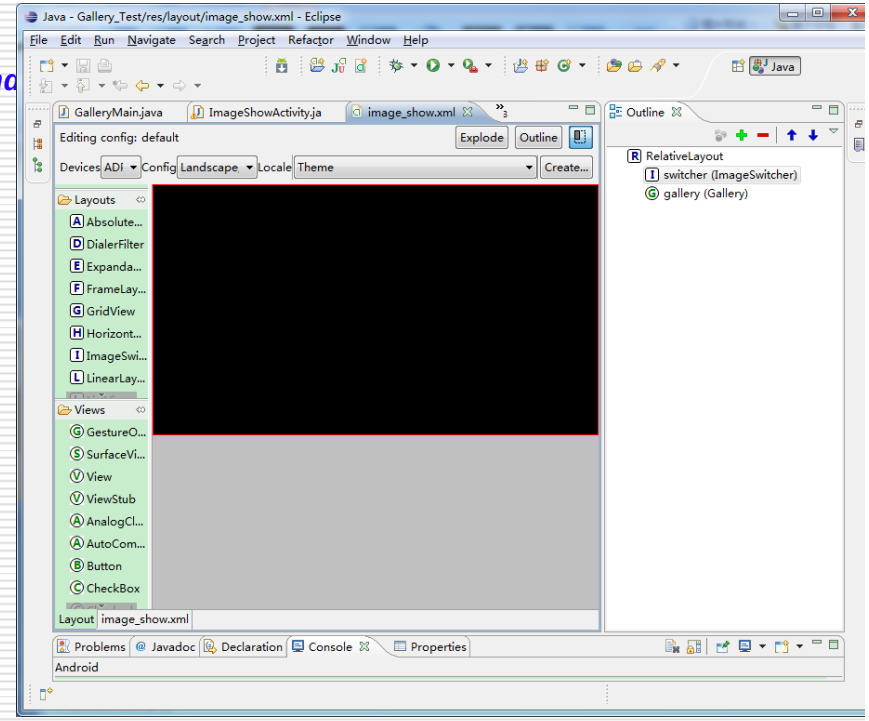
```
private Integer[] mThumbIds = {
R.drawable.sample_thumb_0, R.drawable.sample_thumb_1,
R.drawable.sample_thumb_2, R.drawable.sample_thumb_3,
R.drawable.sample_thumb_4, R.drawable.sample_thumb_5,
R.drawable.sample_thumb_6, .drawable.sample_thumb_7};

private Integer[] mImageIds = {
      R.drawable.sample_0, R.drawable.sample_1,
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7};
```

Gallery组件综合例子

引入组件

在image_show.xml文件中配置Gallery组件和ImageSwitcher组件

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/and
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
      <ImageSwitcher
        android:id="@+id/switcher"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_alignParentTop="true"
                android:layout_alignParentLeft="true" />
       <Gallery android:id="@+id/gallery"
        android:background="#55000000"
                android:layout_width="fill_parent"
                android:layout_height="60dp"
                android:layout_alignParentBottom="true"
                android:layout_alignParentLeft="true"
                android:gravity="center_vertical"
                android:spacing="16dp" />
</RelativeLayout>
```



Gallery组件综合例子

在ImageShowActivity的onCreate方法中加载组件

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.image_show);
    setTitle("ImageShowActivity");
  //   装载ImageSwitcher组件
    mSwitcher=(ImageSwitcher)findViewById(R.id.switcher);
    mSwitcher.setFactory(this);
      mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,android.R.anim.fade_in));
      mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,android.R.anim.fade_out));
   //   装载Gallery组件
    Gallery g=(Gallery)findViewById(R.id.gallery);
   //  创建用于描述图像数据的ImageAdapter对象
    g.setAdapter(new ImageAdapter(this));
    g.setOnItemSelectedListener(this);}
```

Gallery组件综合例子

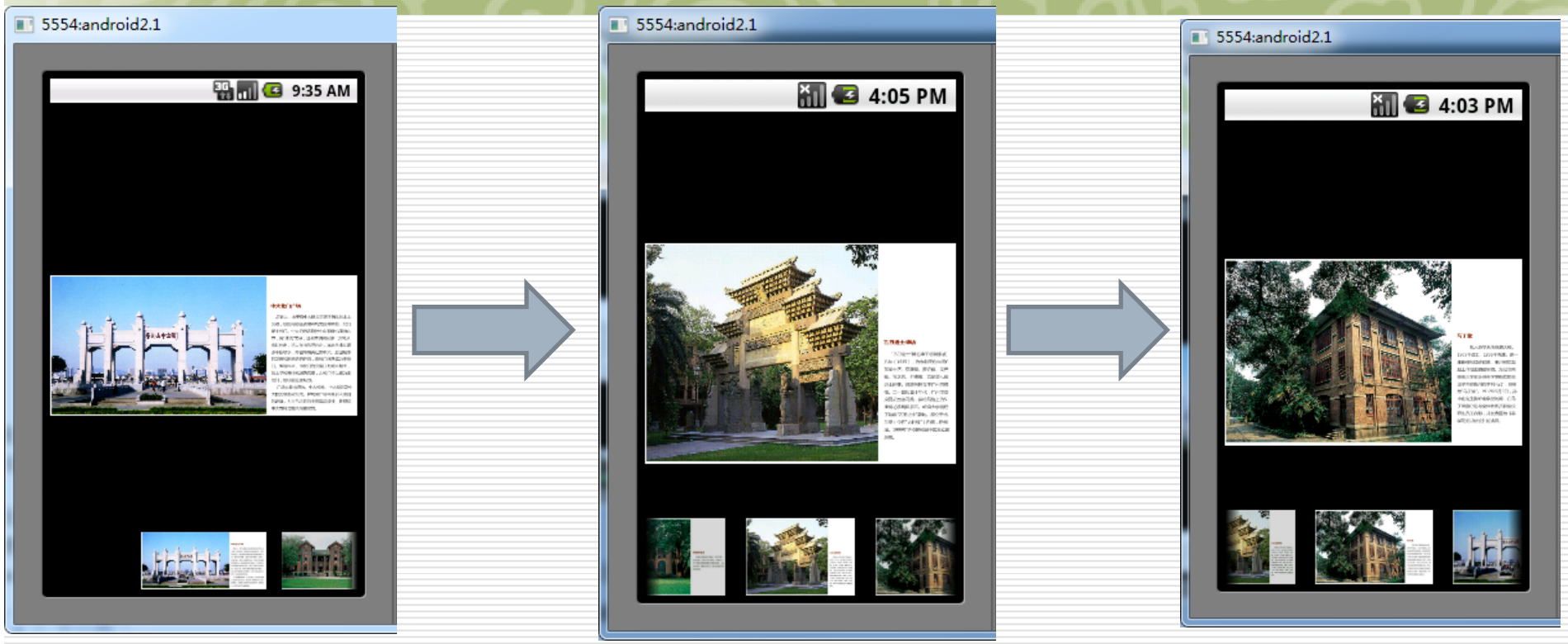
SUN YAT-SEN UNIVERSITY

重要的类：ImageAdapter是android.widget.BaseAdapter的子类，用于描述图像信息。

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;
    public ImageAdapter(Context c) {mContext = c;}
    public int getCount() {return mThumbIds.length}
    public Object getItem(int position) {return position;}
    public long getItemId(int position) {return position;}
    public View getView(int position, View convertView, ViewGroup parent) {
      ImageView i = new ImageView(mContext);
      i.setImageResource(mThumbIds[position]);
      i.setImageResource(mThumbIds[position]);
      i.setAdjustViewBounds(true);
      i.setLayoutParams(new Gallery.LayoutParams(
      LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
      i.setBackgroundResource(R.drawable.picture_frame);
      return i;} }
```

Gallery组件综合例子

SUN YAT-SEN UNIVERSITY



Gallery只能有限地显示指定的图像。当组件显示到最后一张时，就不会再继续显示。当希望图像显示到最后一张时再重第一张开始显示，也就是循环显示。就需要对Gallery的Adapter对象进行一番改进。

Gallery组件综合例子

SUN YAT-SEN UNIVERSITY

在**ImageAdapter**类中有两个非常重要的方法：

1. getCount：此方法在基类中的描述

   abstract int getCount() How many items are in the data set represented by this Adapter.

   用于返回图像总数。

2. getView。当Gallery组件要显示某一个图像时，就会调用getView方法，并将当前的图像索引（position参数）传入该方法。一般getView方法用于返回每一个显示图像的组件（ImageView对象）。在getView方法中除了创建了ImageView对象，还用从mThumbIDs数组中获得了相应的图像资源ID来设置在ImageView中显示的图像。最后还设置了Gallery组件的背景显示风格。

```
public int getCount() {
 return mThumbIds.length;}

public View getView(int position, View
convertView, ViewGroup parent) {
ImageView i = new ImageView(mContext);
i.setImageResource(mThumbIds[position]);
i.setImageResource(mThumbIds[position]);
i.setAdjustViewBounds(true);
i.setLayoutParams(new
Gallery.LayoutParams(LayoutParams.WRAP_C
ONTENT, LayoutParams.WRAP_CONTENT));
i.setBackgroundResource(R.drawable.pictu
re_frame);
return i; }
```

Gallery组件综合例子

循环显示图像

1. 循环显示模拟循环链表，最后一个结点的下一个结点又是第1个结点。
2. getView方法中的position参数和getCount方法。position参数的值是不可能超过getCount方法返回的值，也就是说，position参数值的范围是0至getCount() - 1。
3. 当Gallery组件正好显示到最后一个图像，position参数值正好为getCount() - 1。那如何再让Gallery显示下一个图像呢？也就是说让position参数值再增1？

Gallery组件综合例子

SUN YAT-SEN UNIVERSITY

让getCount方法返回一个很大的数，position参数值就可以随着Gallery组件的图像不断向前移动而增大。现在mThumbIds数组只有7个元素，如果position的值超过数组边界，要想继续循环取得数组中的元素,最简单的方法就是取余

```
int ipos=position%mThumbIds.length;
 position=ipos;
ImageView i = new ImageView(mContext);
i.setImageResource(mThumbIds[ipos]);
i.setImageResource(mThumbIds[ipos]);

 public int getCount() {
return Integer.MAX_VALUE;}

 public void
onItemSelected(AdapterView parent,
View v, int position, long id) {

mSwitcher.setImageResource(mImageId
s[position%mThumbIds.length]);
    }
```

工作原理

1. ListView 针对List中每个item，要求 adapter “给我一个视图”(getView)。
2. 一个新的视图被返回并显示

如果我们有上亿个项目要显示怎么办？为每个项目创建一个新视图？

实际上Android为你缓存了视图。

ListView 中View的复用

SUN YAT-SEN UNIVERSITY

ListView有这么一个东西可以称作——view池

view池有一个最大数量限制，为了方便，取个名字叫maxNum，就是设备所能显示的最大item数目

例如进入ListView页面时，刚好显示5个item，则maxNum的值就是6。当向下滑动ListView要显示下边的item时，就有可能第一个item已经有一半滑出了屏幕，而最下边一个item还有一半没有进入屏幕，但这已经是显示了6个item了，

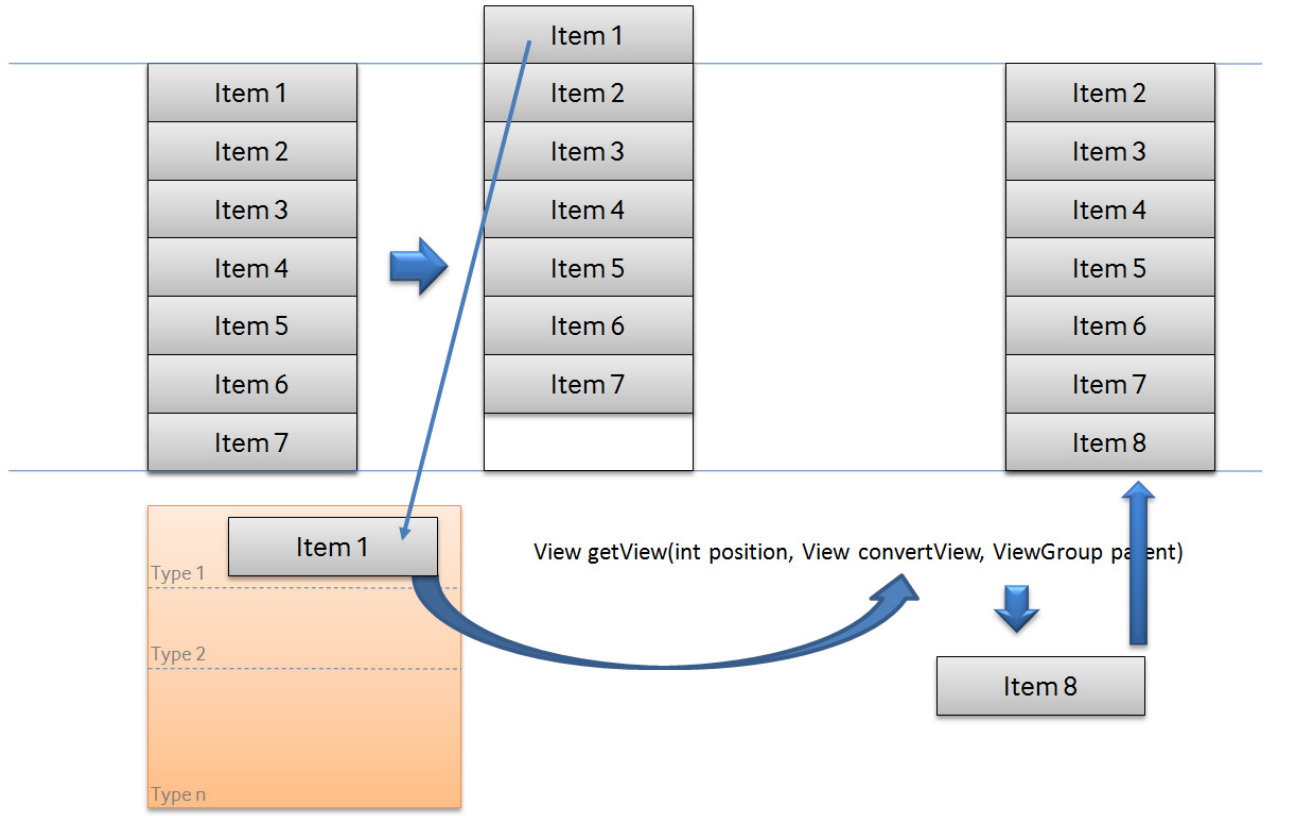
继续往上滑，第一个item已经完全从屏幕消失了，Android将其放到view池里，此时第6个item也已经完全显示了，继续往上滑，上边还是重复前边的步骤。

首先要去view池里查询现在已经创建的item数目是否达到了maxNum

若没有，继续创建新item view，若已经是最大数目了，则就去view池里去拿闲置的item view（这时view池里边肯定有闲置view）这个被拿过来的view就是传到Adapter中getView()方法中的参数 convertView，完成view的复用

ListView 和 Adapter 的基础

Android中有个叫做Recycler的构件，下图是其工作原理：



http://www.cnblogs.com/xiaowenji/archive/2010/12/08/1900579.html

ListView 和 Adapter 的基础

1. 如果有10亿个项目(item)，其中只有可见的项目存在内存中，其他的在Recycler中。
2. ListView先请求一个type1视图(getView)然后请求其他可见的项目。convertView在getView中是空(null)的。
3. 当item1移出屏幕，并且一个新的项目从屏幕低端上来时，ListView再请求一个type1视图。convertView此时不是空值了，它的值是item1。你只需设定新的数据然后返回convertView，不必重新创建一个视图。

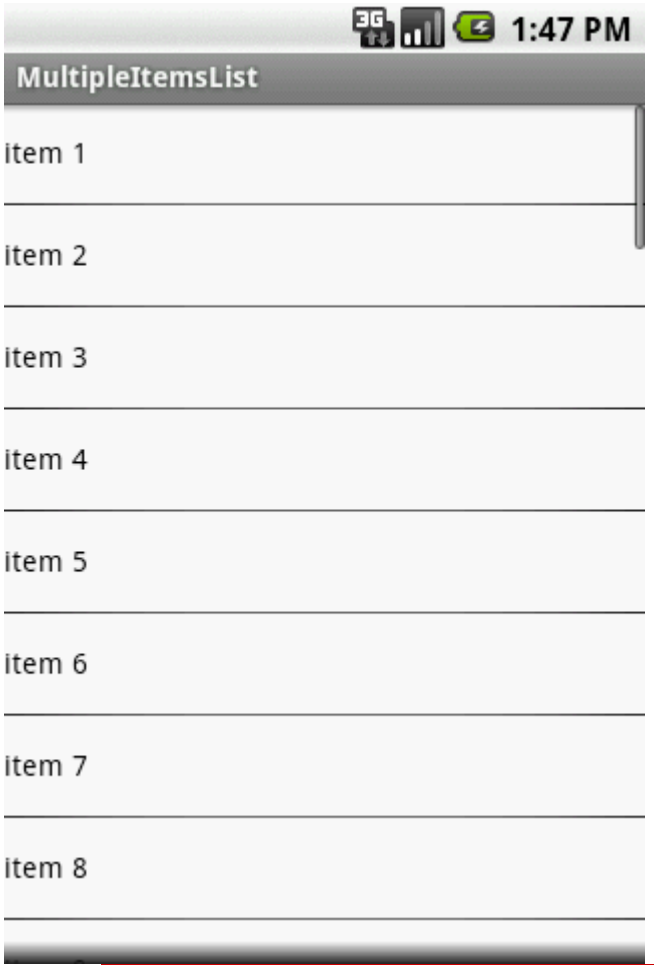
下面的示例代码，这里在getView中使用了System.out进行输出

ListView 和 Adapter 的基础

SUN YAT-SEN UNIVERSITY

```
public class MultipleItemsList extends ListActivity {
   private MyCustomAdapter mAdapter;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      mAdapter = new MyCustomAdapter();
      for (int i = 0; i < 50; i++) {
         mAdapter.addItem("item " + i);}
      setListAdapter(mAdapter);
   }
   private class MyCustomAdapter extends BaseAdapter {
      private ArrayList mData = new ArrayList();
      private LayoutInflater mInflater;
      public MyCustomAdapter() {
         mInflater =
(LayoutInflater)getSystemService(Context.LAYOUT_INFLATER_S
ERVICE);}
      public void addItem(final String item) {
         mData.add(item);
         notifyDataSetChanged();}

      @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) {
         System.out.println("getView " + position + " " +
convertView);
         ViewHolder holder = null;
         if (convertView == null) {
            convertView = mInflater.inflate(R.layout.item1, null)
            holder = new ViewHolder();
            holder.textView =
(TextView)convertView.findViewById(R.id.text);
            convertView.setTag(holder);
         } else {
            holder = (ViewHolder)convertView.getTag();
         }
         holder.textView.setText(mData.get(position));
         return convertView;}

   }

   public static class ViewHolder {
      public TextView textView;}
}
```

执行程序，然后在**Logcat**中查看日志

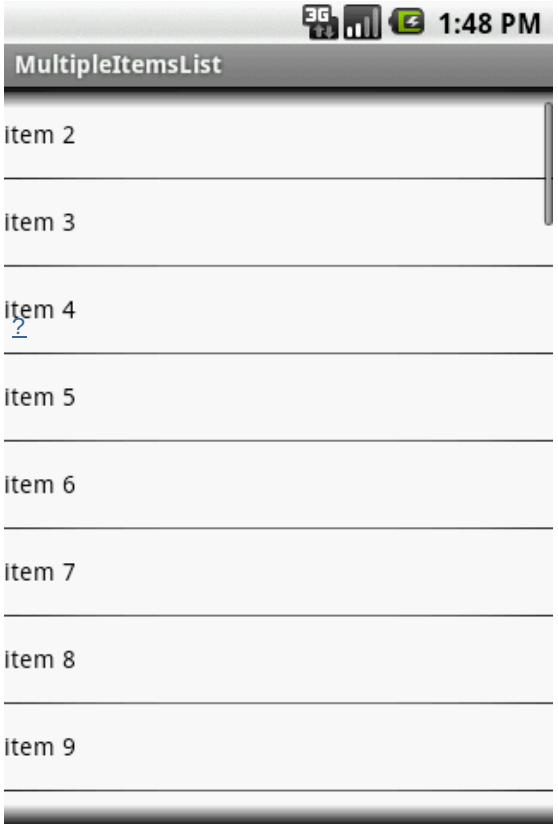


getView 被调用 **9** 次，**convertView** 对于所有的可见项目是空值（如下）

```
02-05 13:47:32.559: INFO/System.out(947): getView 0 null
02-05 13:47:32.570: INFO/System.out(947): getView 1 null
02-05 13:47:32.589: INFO/System.out(947): getView 2 null
02-05 13:47:32.599: INFO/System.out(947): getView 3 null
02-05 13:47:32.619: INFO/System.out(947): getView 4 null
02-05 13:47:32.629: INFO/System.out(947): getView 5 null
02-05 13:47:32.708: INFO/System.out(947): getView 6 null
02-05 13:47:32.719: INFO/System.out(947): getView 7 null
02-05 13:47:32.729: INFO/System.out(947): getView 8 null
```

然后稍微向下滚动**List**，直到**item10**出现：

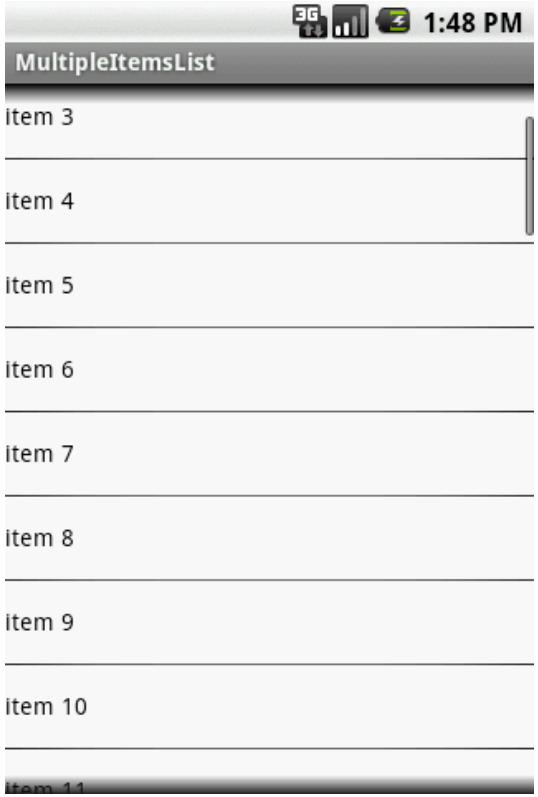
convertView仍然是空值，因为**recycler**中没有视图（**item1**的边缘仍然可见，在顶端）




```
02-05 13:48:25.169:
INFO/System.out(947): getView 9 null
```

再滚动**List**

convertView不是空值了！**item1**离开屏幕到**Recycler**中去了，然后**item11**被创建




```
02-05 13:48:42.879: INFO/System.out(947):
getView 10 android.widget.LinearLayout@437430f8
```

Android的Logcat命令详解

SUN YAT-SEN UNIVERSITY

Android LOG系统提供了收集和查看系统调试输出的功能。各种应用程序和系统其他部分输出的LOG都存储在一些循环缓冲区里，这些缓冲区可以通过 logcat 命令来查看和过滤使用。

Android的Logcat命令详解

SUN YAT-SEN UNIVERSITY

使用**logcat**命令

可以用 logcat 命令来查看和控制系统LOG Buffer里内容，通常用法：

[adb] logcat [<option>] ... [<filter-spec>] ...

可以在电脑，也可以通过运行在模拟器/设备上的远程adb shell端来使用logcat命令。在你的电脑上查看LOG输出：

$ adb logcat

通过远端 adb shell，可以这样：

logcat

Android的Logcat命令详解

SUN YAT-SEN UNIVERSITY

过滤Log输出

Android LOG信息都有一个标签（tag）和它的优先级（priority）。

- LOG的标签是一个简短的描述来指示发生在哪一个系统部件内，在写LOG时指定。（比如：“View”就是查看VIEW系统的标签）.
- 优先级有下列几种，按照优先级从低到高顺序排列为：
  - V — Verbose (最低优先级)
  - D — Debug
  - I — Info
  - W — Warning
  - E — Error
  - F — Fatal
  - S — Silent (最高优先级，没有任何输出)

Android的Logcat命令详解

android.util.Log常用的方法有以下**5**个：

- Log.v()
- Log.d()
- Log.i()
- Log.w()
- Log.e()。

根据首字母对应VERBOSE，DEBUG,INFO, WARN，ERROR。

Android的Logcat命令详解

SUN YAT-SEN UNIVERSITY

1. Log.v 的调试颜色为黑色的，任何消息都会输出，这里的v代表verbose啰嗦的意思，平时使用就是Log.v("","");
2. Log.d的输出颜色是蓝色的，仅输出debug调试的意思，但他会输出上层的信息，过滤起来可以通过DDMS的Logcat标签来选择.
3. Log.i的输出为绿色，一般提示性的消息information，它不会输出Log.v和Log.d的信息，但会显示i、w和e的信息
4. Log.w的意思为橙色，可以看作为warning警告，一般需要我们注意优化Android代码，同时选择它后还会输出Log.e的信息。
5. Log.e为红色，可以想到error错误，这里仅显示红色的错误信息，这些错误就需要我们认真的分析，查看栈的信息了。

Android的Logcat命令详解

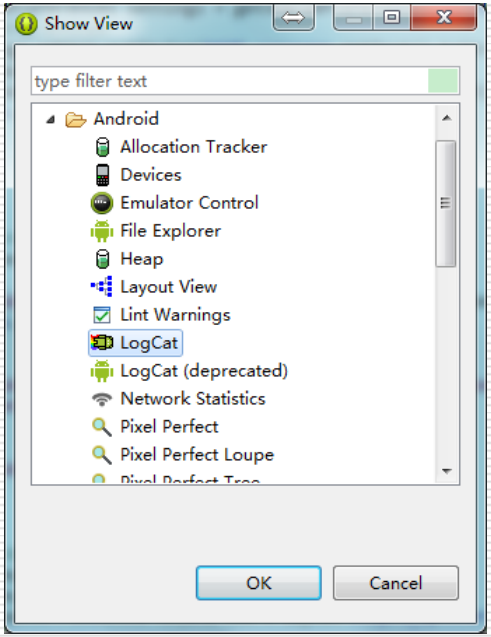
SUN YAT-SEN UNIVERSITY

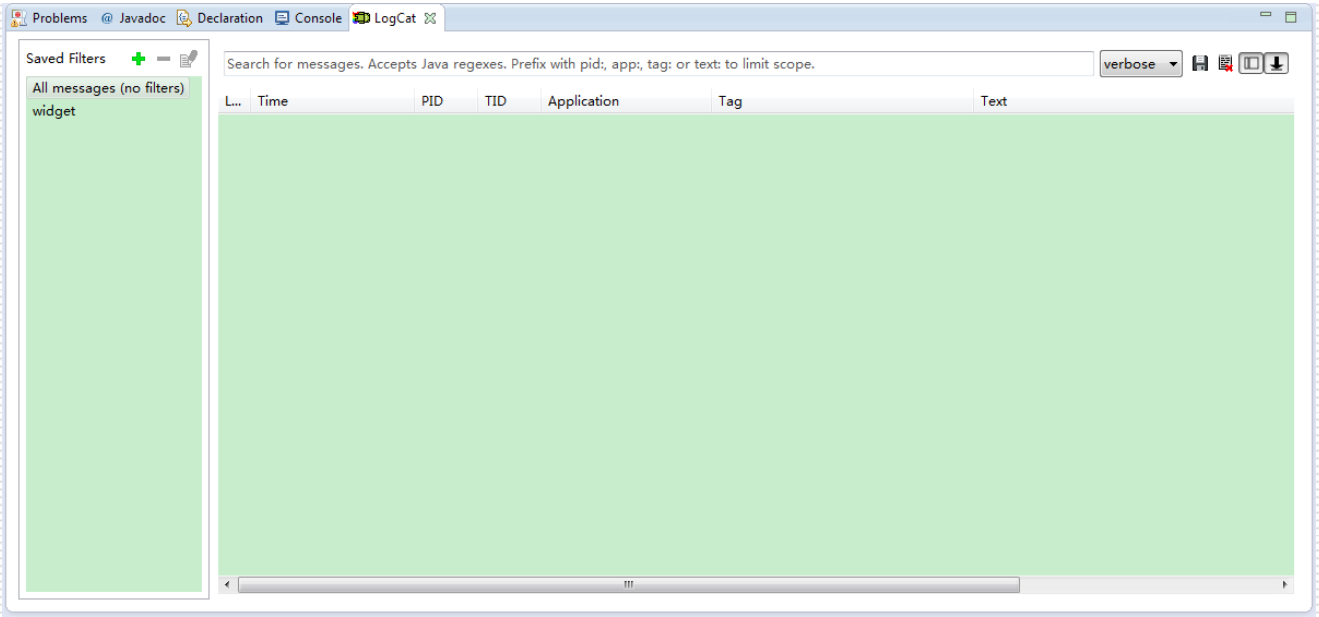
1. Log.v 的调试颜色为黑色的，任何消息都会输出，这里的v代表verbose啰嗦的意思，平时使用就是Log.v("","");
2. Log.d的输出颜色是蓝色的，仅输出debug调试的意思，但他会输出上层的信息，过滤起来可以通过DDMS的Logcat标签来选择.
3. Log.i的输出为绿色，一般提示性的消息information，它不会输出Log.v和Log.d的信息，但会显示i、w和e的信息
4. Log.w的意思为橙色，可以看作为warning警告，一般需要我们注意优化Android代码，同时选择它后还会输出Log.e的信息。
5. Log.e为红色，可以想到error错误，这里仅显示红色的错误信息，这些错误就需要我们认真的分析，查看栈的信息了。

Android的Logcat命令详解

打开LogCat视窗

启动Eclipse,在Window->Show View会出来一个对话框，当点击Ok按钮时，会在控制台窗口出现LogCat视窗.如下图：





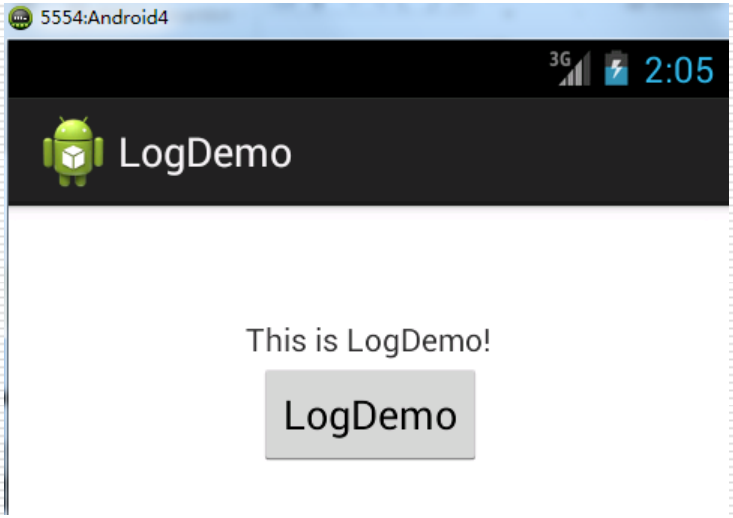
Android的Logcat命令详解

SUN YAT-SEN UNIVERSITY

代码片段:

```
public class MainActivity extends Activity {
    private Button bt;
    private static final String DemoTAG="日志";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        bt=(Button)findViewById(R.id.Button1);
        bt.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
Log.v(DemoTAG, "This is Verbose.");
                Log.d(DemoTAG, "This is Debug.");
                Log.i(DemoTAG, "This is Information");
                Log.w(DemoTAG, "This is Warnning.");
                Log.e(DemoTAG, "This is Error.");
            }
        });
    }
```

Android的Logcat命令详解



点击后，在LogCat观察到的日志

L...	Time	PID	TID	Application	Tag	Text
V	10-29 13:53:50.958	853	853	com.example.logdemo	日志	This is Verbose.
D	10-29 13:53:50.958	853	853	com.example.logdemo	日志	This is Debug.
I	10-29 13:53:50.968	853	853	com.example.logdemo	日志	This is Information
W	10-29 13:53:50.968	853	853	com.example.logdemo	日志	This is Warnning.
E	10-29 13:53:50.968	853	853	com.example.logdemo	日志	This is Error.

Questions?