[上篇](http://www.nbcio.com/archives/2195.html)[下篇](http://www.nbcio.com/archives/2202.html)

Android开发网上的一些重要知识点（转）

2011年12月09日 ⁄ [Andriod](http://www.nbcio.com/archives/category/andriod) ⁄ [暂无评论](http://www.nbcio.com/archives/2200.html#respond)⁄ 被围观 659 views+

1. android单实例运行方法

我们都知道Android平台没有任务管理器，而内部App维护者一个Activity history stack来实现窗口显示和销毁，对于常规从快捷方式运行来看都是startActivity可能会使用FLAG\_ACTIVITY\_NEW\_TASK标记来打开一个新窗口，比如Launcher，所以考虑单任务的实现方法比较简单，首先Android123纠正下大家一种错误的方法就是直接在androidmanifest.xml的application节点中加入android:launchMode="singleInstance"这句，其实这样将不会起到任何作用，Apps内部维护的历史栈作用于Activity，我们必须在activity节点中加入android:launchMode="singleInstance" 这句才能保证单实例，当然一般均加在主程序启动窗口的Activity。

2. px像素如何转为dip设备独立像素

最近有网友问如何将px像素转为dip独立设备像素，由于Android的设备分辨率众多，目前主流的为wvga，而很多老的设备为hvga甚至低端的qvga，对于兼容性来说使用dip无非是比较方便的，由于他和分辨率无关和屏幕的密度大小有关，所以推荐使用。 px= (int) (dip\*density+0.5f) //这里android开发网提示大家很多网友获取density(密度)的方法存在问题，从资源中获取的是静态定义的，一般为1.0对于HVGA是正好的，而对于wvga这样的应该从WindowsManager中获取，WVGA为1.5 这里可以再补充一下dip，sip的知识

3. Android中动态改变ImageView大小

很多网友可能发现在layout.xml文件中定义了ImageView的绝对大小后，无法动态修改以后的大小显示，其实Android平台在设计UI控件时考虑到这个问题，为了适应不同的Drawable可以通过在xml的相关ImageView中加入android:scaleType="fitXY" 这行即可，但因为使用了缩放可能会造成当前UI有所变形。使用的前提是限制ImageView所在的层，可以使用一个内嵌的方法限制显示。

4. 如何判断Android手机当前是否联网?

如果拟开发一个网络应用的程序，首先考虑是否接入网络，在Android手机中判断是否联网可以通过 ConnectivityManager 类的isAvailable()方法判断，首先获取网络通讯类的实例 ConnectivityManager cwjManager=(ConnectivityManager)getSystemService(Context.CONNECTIVITY\_SERVICE); ，使用cwjManager.getActiveNetworkInfo().isAvailable(); 来返回是否有效，如果为True则表示当前Android手机已经联网，可能是WiFi或GPRS、HSDPA等等，具体的可以通过ConnectivityManager 类的getActiveNetworkInfo() 方法判断详细的接入方式，需要注意的是有关调用需要加入<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"></uses-permission> 这个权限，android开发网提醒大家在真机上Market和Browser程序都使用了这个方法，来判断是否继续，同时在一些网络超时的时候也可以检查下网络连接是否存在，以免浪费手机上的电力资源。

5. Drawable、Bitmap、Canvas和Paint的关系

很多网友刚刚开始学习Android平台，对于Drawable、Bitmap、Canvas和Paint它们之间的概念不是很清楚，其实它们除了Drawable外早在Sun的J2ME中就已经出现了，但是在Android平台中，Bitmap、Canvas相关的都有所变化。 首先让我们理解下Android平台中的显示类是View，但是还提供了底层图形类android.graphics，今天所说的这些均为graphics底层图形接口。 Bitmap - 称作位图，一般位图的文件格式后缀为bmp，当然编码器也有很多如RGB565、RGB888。作为一种逐像素的显示对象执行效率高，但是缺点也很明显存储效率低。我们理解为一种存储对象比较好。 Drawable - 作为Android平下通用的图形对象，它可以装载常用格式的图像，比如GIF、PNG、JPG，当然也支持BMP，当然还提供一些高级的可视化对象，比如渐变、图形等。 Canvas - 名为画布，我们可以看作是一种处理过程，使用各种方法来管理Bitmap、GL或者Path路径，同时它可以配合Matrix矩阵类给图像做旋转、缩放等操作，同时Canvas类还提供了裁剪、选取等操作。 Paint - 我们可以把它看做一个画图工具，比如画笔、画刷。他管理了每个画图工具的字体、颜色、样式。 如果涉及一些Android游戏开发、显示特效可以通过这些底层图形类来高效实现自己的应用。

6. Activity切换导致的onCreate重复执行

部分网友会发现Activity在切换到后台或布局从横屏LANDSCAPE切换到PORTRAIT，会重新切换Activity会触发一次onCreate方法，我们可以在androidmanifest.xml中的activit元素加入这个属性android:configChanges="orientation|keyboardHidden" 即可，比如 <activity android:name=".android123"**android:configChanges="orientation|keyboardHidden"** android:label="@string/app\_name"> 同时在Activity的Java文件中重载onConfigurationChanged(Configuration newConfig)这个方法，这样就不会在布局切换或窗口切换时重载onCreate等方法。代码如下: @Override  
  
public void onConfigurationChanged(Configuration newConfig)  
  
{  
  
super.onConfigurationChanged(newConfig);  
  
if (this.getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE)  
  
{  
  
//land  
  
}  
  
else if (this.getResources().getConfiguration().orientation == Configuration.ORIENTATION\_PORTRAIT)  
  
{  
  
//port  
  
}  
  
}

7. Android的ImageButton问题

很多网友对Android提供的ImageButton有个疑问，当显示Drawable图片时就不会再显示文字了，其实解决的方法有两种，第一种就是图片中就写入文字，但是这样解决会增加程序体积，同时硬编码方式会影响多国语言的发布。第二种解决方法很简单，通过分析可以看到ImageButton的layout，我们可以直接直接继承，添加一个TextView，对齐方式为右侧即可实现ImageButton支持文字右侧显示。

8. Android代码优化技术

1.Java内存控制 对于字符串操作而言如果需要连加这样的操作建议使用StringBuilder，经过调试不难发现如果你的字符串每次连加，使用String需要的内存开销会远大于StringBuilder，然后Android手机常规的运行内存大约在128MB左右，对于运行多任务就需要考虑了，[Android开发网](http://www.android123.com.cn/include/FCKeditor/editor/)提示因为Java有GC不需要手动释放那么分配的时候就要格外的小心，频繁的GC操作仍然是很影响性能的，在调试时我们可以通过logcat查看内存释放情况。 2.循环使用 平时在访问一个属性的时候效率远比一个固定变量低，如果你的循环估计次数常常大于5，假设xxx.GetLength()方法的值一般大于5，推荐这样写，比如 for(int i=0;i<xxx.GetLength();i++) 这里xxx.GetLength在每次循环都要调用，必然会影响程序效率，在游戏开发中显得更为明显，改进的方法应该为 int j=xxx.GetLength() for(int i=0;i<j;i++) 3.图片的优化 在Android平台中2维图像处理库BitmapFactory做的比较智能，为了减少文件体积和效率，常常不用很多资源文件，而把很多小图片放在一个图片中，有切片的方式来完成，在J2ME中我们这样是为了将少文件头而解决MIDP这些设备的问题，而Android中虽然机型硬件配置都比较高，有关Android G1硬件配置可以参考[G1手机参数以及评测](http://www.android123.com/Androidshouji/HTC_Dream/225.html)，但是当资源多时这样的运行效率还是令人满意的，至少Dalvik优化的还不是很够。

9. Android开发进阶之NIO非阻塞包(一)

对于Android的网络通讯性能的提高，我们可以使用Java上高性能的NIO (New I/O) 技术进行处理，NIO是从JDK 1.4开始引入的，NIO的N我们可以理解为Noblocking即非阻塞的意思，相对应传统的I/O，比如Socket的accpet()、read()这些方法而言都是阻塞的。 NIO主要使用了Channel和Selector来实现，Java的Selector类似Winsock的Select模式，是一种基于事件驱动的，整个处理方法使用了轮训的状态机，如果你过去开发过Symbian应用的话这种方式有点像活动对象，好处就是单线程更节省系统开销，NIO的好处可以很好的处理并发，对于Android网游开发来说比较关键，对于多点Socket连接而言使用NIO可以大大减少线程使用，降低了线程死锁的概率，毕竟手机游戏有UI线程，音乐线程，网络线程，管理的难度可想而知，同时I/O这种低速设备将影响游戏的体验。 NIO作为一种中高负载的I/O模型，相对于传统的BIO (Blocking I/O)来说有了很大的提高，处理并发不用太多的线程，省去了创建销毁的时间，如果线程过多调度是问题，同时很多线程可能处于空闲状态，大大浪费了CPU时间，同时过多的线程可能是性能大幅下降，一般的解决方案中可能使用线程池来管理调度但这种方法治标不治本。使用NIO可以使并发的效率大大提高。当然NIO和JDK 7中的AIO还存在一些区别，AIO作为一种更新的当然这是对于Java而言，如果你开发过Winsock服务器，那么IOCP这样的I/O完成端口可以解决更高级的负载，当然了今天Android123主要给大家讲解下为什么使用NIO在Android中有哪些用处。 NIO我们分为几个类型分别描述，作为Java的特性之一，我们需要了解一些新的概念，比如ByteBuffer类，Channel，SocketChannel，ServerSocketChannel，Selector和SelectionKey。有关具体的使用，Android开发网将在明天详细讲解。网友可以在Android SDK文档中看下java.nio和java.nio.channels两个包了解。<http://www.android123.com.cn/androidkaifa/695.html>

了解下这种技术，看看在马上要做的项目中是否用得到

10. Android Theme和Styles内部定义解析

昨天我们讲到的有关在AndroidManifest.xml中定义Activity的theme方法来实现无标题的方法，在[使用xml让你的Activity无标题方法](http://www.android123.com.cn/kaifafaq/671.html) 一文中讲到的，很多网友不明白为什么这样做，其实在Android123以前的文章中多次提到了styles样式定义方法，今天Android开发网再次把一些网友回顾了解下android样式的内部定义。在一个工程的res/values/theme.xml中我们可以方便的定义自己的风格主题，比如下面的cwjTheme中我们使用了基于android内部的白色调的背景Theme.Light，设置windowsNoTitle为true代表没有标题，背景颜色我们使用了android内部定义的透明，同时设置listView控件的样式为cwjListView，xml样式代码如下: <?xml version="1.0" encoding="utf-8"?>   
  
<resources>   
  
<style name="cwjTheme" parent="android:Theme.Light">   
  
<item name="android:windowNoTitle">true</item>   
  
<item name="android:windowBackground">@android:color/transparent</item>   
  
<item name="android:listViewStyle">@style/cwjListView</item>   
  
</style> 有关ListView控件我们自定义的风格就是修改下系统listview这个控件的每行分隔符样式，这里我们在工程下res/drawable文件夹下放一个图片名为list\_selector图片，这样我们的cwjListView的代码可以这样写 <style name="cwjListView" parent="@android:style/Widget.ListView">   
  
<item name="android:listSelector">@drawable/list\_selector</item>   
  
</style>   
  
</resources> 通过定义style可以设置更多，比如让cwjListView的字体颜色就加入textAppearance属性，比如 <item name="textAppearance">@android:style/TextAppearance</item> 等等。

11.Android JSON解析示例代码

来自Google官方的有关Android平台的JSON解析示例，如果远程服务器使用了json而不是xml的数据提供，在Android平台上已经内置的org.json包可以很方便的实现手机客户端的解析处理。下面Android123一起分析下这个例子，帮助Android开发者需要有关 HTTP通讯、正则表达式、JSON解析、appWidget开发的一些知识。 public class WordWidget extends **AppWidgetProvider** { //appWidget  
  
@Override  
  
public void onUpdate(Context context, AppWidgetManager appWidgetManager,  
  
int[] appWidgetIds) {  
  
context.startService(new Intent(context, UpdateService.class)); //避免ANR，所以Widget中开了个服务  
  
} public static class UpdateService extends Service {  
  
@Override  
  
public void onStart(Intent intent, int startId) {  
  
// Build the widget update for today  
  
RemoteViews updateViews = buildUpdate(this); ComponentName thisWidget = new ComponentName(this, WordWidget.class);  
  
AppWidgetManager manager = AppWidgetManager.getInstance(this);  
  
manager.updateAppWidget(thisWidget, updateViews);  
  
} public RemoteViews buildUpdate(Context context) {  
  
// Pick out month names from resources  
  
Resources res = context.getResources();  
  
String[] monthNames = res.getStringArray(R.array.month\_names); Time today = new Time();  
  
today.setToNow(); String pageName = res.getString(R.string.template\_wotd\_title,  
  
monthNames[today.month], today.monthDay);  
  
RemoteViews updateViews = null;  
  
String pageContent = ""; try {  
  
SimpleWikiHelper.prepareUserAgent(context);  
  
pageContent = SimpleWikiHelper.getPageContent(pageName, false);  
  
} catch (ApiException e) {  
  
Log.e("WordWidget", "Couldn't contact API", e);  
  
} catch (ParseException e) {  
  
Log.e("WordWidget", "Couldn't parse API response", e);  
  
} Pattern pattern = Pattern.compile(SimpleWikiHelper.WORD\_OF\_DAY\_REGEX); //正则表达式处理，有关定义见下面的SimpleWikiHelper类  
  
Matcher matcher = pattern.matcher(pageContent);  
  
if (matcher.find()) {  
  
updateViews = new RemoteViews(context.getPackageName(), R.layout.widget\_word); String wordTitle = matcher.group(1);  
  
updateViews.setTextViewText(R.id.word\_title, wordTitle);  
  
updateViews.setTextViewText(R.id.word\_type, matcher.group(2));  
  
updateViews.setTextViewText(R.id.definition, matcher.group(3).trim()); String definePage = res.getString(R.string.template\_define\_url,  
  
Uri.encode(wordTitle));  
  
Intent defineIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse(definePage)); //这里是打开相应的网页，所以Uri是http的url，action是view即打开web浏览器  
  
PendingIntent pendingIntent = PendingIntent.getActivity(context,  
  
0 /\* no requestCode \*/, defineIntent, 0 /\* no flags \*/);  
  
updateViews.setOnClickPendingIntent(R.id.widget, pendingIntent); //单击Widget打开Activity } else {   
  
updateViews = new RemoteViews(context.getPackageName(), R.layout.widget\_message);  
  
CharSequence errorMessage = context.getText(R.string.widget\_error);  
  
updateViews.setTextViewText(R.id.message, errorMessage);  
  
}  
  
return updateViews;  
  
} @Override  
  
public IBinder onBind(Intent intent) {  
  
// We don't need to bind to this service  
  
return null;  
  
}  
  
}  
  
} 有关网络通讯的实体类，以及一些常量定义如下: public class SimpleWikiHelper {  
  
private static final String TAG = "SimpleWikiHelper"; public static final String WORD\_OF\_DAY\_REGEX =  
  
"(?s)\\{\\{wotd\\|(.+?)\\|(.+?)\\|([^#\\|]+).\*?\\}\\}"; private static final String WIKTIONARY\_PAGE =  
  
"[http://en.wiktionary.org/w/api.php?action=query&prop=revisions&titles=%s](http://en.wiktionary.org/w/api.php?action=query&prop=revisions&titles=%25s)&" +  
  
"rvprop=content&format=json%s"; private static final String WIKTIONARY\_EXPAND\_TEMPLATES =  
  
"&rvexpandtemplates=true"; private static final int HTTP\_STATUS\_OK = 200; private static byte[] sBuffer = new byte[512]; private static String sUserAgent = null; public static class ApiException extends Exception {  
  
public ApiException(String detailMessage, Throwable throwable) {  
  
super(detailMessage, throwable);  
  
} public ApiException(String detailMessage) {  
  
super(detailMessage);  
  
}  
  
} public static class ParseException extends Exception {  
  
public ParseException(String detailMessage, Throwable throwable) {  
  
super(detailMessage, throwable);  
  
}  
  
} public static void prepareUserAgent(Context context) {  
  
try {  
  
// Read package name and version number from manifest  
  
PackageManager manager = context.getPackageManager();  
  
PackageInfo info = manager.getPackageInfo(context.getPackageName(), 0);  
  
sUserAgent = String.format(context.getString(R.string.template\_user\_agent),  
  
info.packageName, info.versionName); } catch(NameNotFoundException e) {  
  
Log.e(TAG, "Couldn't find package information in PackageManager", e);  
  
}  
  
} public static String getPageContent(String title, boolean expandTemplates)  
  
throws ApiException, ParseException {  
  
String encodedTitle = Uri.encode(title);  
  
String expandClause = expandTemplates ? WIKTIONARY\_EXPAND\_TEMPLATES : ""; String content = getUrlContent(String.format(WIKTIONARY\_PAGE, encodedTitle, expandClause));  
  
try {  
  
JSONObject response = new JSONObject(content);  
  
JSONObject query = response.getJSONObject("query");  
  
JSONObject pages = query.getJSONObject("pages");  
  
JSONObject page = pages.getJSONObject((String) pages.keys().next());  
  
JSONArray revisions = page.getJSONArray("revisions");  
  
JSONObject revision = revisions.getJSONObject(0);  
  
return revision.getString("\*");  
  
} catch (JSONException e) {  
  
throw new ParseException("Problem parsing API response", e);  
  
}  
  
} protected static synchronized String getUrlContent(String url) throws ApiException {  
  
if (sUserAgent == null) {  
  
throw new ApiException("User-Agent string must be prepared");  
  
} HttpClient client = new DefaultHttpClient();  
  
HttpGet request = new HttpGet(url);  
  
request.setHeader("User-Agent", sUserAgent); //设置客户端标识 try {  
  
HttpResponse response = client.execute(request); StatusLine status = response.getStatusLine();  
  
if (status.getStatusCode() != HTTP\_STATUS\_OK) {  
  
throw new ApiException("Invalid response from server: " +  
  
status.toString());  
  
} HttpEntity entity = response.getEntity();  
  
InputStream inputStream = entity.getContent(); //获取HTTP返回的数据流 ByteArrayOutputStream content = new ByteArrayOutputStream(); int readBytes = 0;  
  
while ((readBytes = inputStream.read(sBuffer)) != -1) {  
  
content.write(sBuffer, 0, readBytes); //转化为字节数组流  
  
} return new String(content.toByteArray()); //从字节数组构建String  
  
} catch (IOException e) {  
  
throw new ApiException("Problem communicating with API", e);  
  
}  
  
}  
  
} 有关整个每日维基的widget例子比较简单，主要是帮助大家积累常用代码，了解Android平台 JSON的处理方式，毕竟很多Server还是Java的。

12.Android中使用定时器TimerTask类介绍

在Android平台中需要反复按周期执行方法可以使用Java上自带的TimerTask类，TimerTask相对于Thread来说对于资源消耗的更低，除了使用Android自带的AlarmManager使用Timer定时器是一种更好的解决方法。 我们需要引入import java.util.Timer; 和 import java.util.TimerTask; private Timer mTimer = new Timer(true);  
  
private TimerTask mTimerTask; mTimerTask = new TimerTask()  
  
{  
  
public void run()  
  
{  
  
Log.v("android123","cwj");  
  
}   
  
};  
  
mTimer.schedule(mTimerTask, 5000,1000); //在1秒后每5秒执行一次定时器中的方法，比如本文为调用log.v打印输出。 如果想取消可以调用下面方法，取消定时器的执行 while(!mTimerTask.cancel());  
  
mTimer.cancel(); 最后Android123提示大家，如果处理的东西比较耗时还是开个线程比较好，Timer还是会阻塞主线程的执行，更像是一种消息的执行方式。当然比Handler的postDelay等方法更适合处理计划任务。

13.Android应用Icon大小在不同分辨率下定义

对于Android平台来说，不同分辨率下Icon的大小设计有着不同的要求，对于目前主流的HDPI即WVGA级别来说，通常hdpi的应用icon大小为72x72，而标准的mdpi即hvga为48x48，对于目前HTC和Motorola推出的一些QVGA的使用了ldpi，图标为32x32，常见的Android图标大小设计规范如下表所示: **Launcher**  
  
**36 x 36 px**  
  
**48 x 48 px**  
  
**72 x 72 px** **Menu**  
  
**36 x 36 px**  
  
**48 x 48 px**  
  
**72 x 72 px** **Status Bar**  
  
**24 x 24 px**  
  
**32 x 32 px**  
  
**48 x 48 px** **Tab**  
  
**24 x 24 px**  
  
**32 x 32 px**  
  
**48 x 48 px** **Dialog**  
  
**24 x 24 px**  
  
**32 x 32 px**  
  
**48 x 48 px** **List View**  
  
**24 x 24 px**  
  
**32 x 32 px**  
  
**48 x 48 px** 对于android界面设计的安全色，如下表 而对于系统自带默认程序的图标，下面为png的透明格式，直接鼠标右键另存为即可

看看sdk文档上的关于界面图标的详细说明。 14.Android控件美化Shape你会用吗?

如果你对Android系统自带的UI控件感觉不够满意，可以尝试下自定义控件，我们就以Button为例，很早以前Android123就写到过[Android Button按钮控件美化方法](http://www.android123.com.cn/androidkaifa/435.html)里面提到了xml的selector构造。当然除了使用drawable这样的图片外今天Android开发网谈下自定义图形shape的方法，对于Button控件Android上支持以下几种属性shape、gradient、stroke、corners等。 我们就以目前系统的Button的selector为例说下: <shape>  
  
<gradient  
  
android:startColor="#ff8c00"  
  
android:endColor="#FFFFFF"  
  
android:angle="270" />  
  
<stroke  
  
android:width="2dp"  
  
android:color="#dcdcdc" />  
  
<corners  
  
android:radius="2dp" />  
  
<padding  
  
android:left="10dp"  
  
android:top="10dp"  
  
android:right="10dp"  
  
android:bottom="10dp" />  
  
</shape> 对于上面，这条shape的定义，分别为渐变，在gradient中startColor属性为开始的颜色，endColor为渐变结束的颜色，下面的angle是角度。接下来是stroke可以理解为边缘，corners为拐角这里radius属性为半径，最后是相对位置属性padding。 对于一个Button完整的定义可以为 <?xml version="1.0" encoding="utf-8"?>  
  
<selector  
  
xmlns:android="<http://schemas.android.com/apk/res/android>">  
  
<item android:state\_pressed="true" >  
  
<shape>  
  
<gradient  
  
android:startColor="#ff8c00"  
  
android:endColor="#FFFFFF"  
  
android:angle="270" />  
  
<stroke  
  
android:width="2dp"  
  
android:color="#dcdcdc" />  
  
<corners  
  
android:radius="2dp" />  
  
<padding  
  
android:left="10dp"  
  
android:top="10dp"  
  
android:right="10dp"  
  
android:bottom="10dp" />  
  
</shape>  
  
</item> <item android:state\_focused="true" >  
  
<shape>  
  
<gradient  
  
android:startColor="#ffc2b7"  
  
android:endColor="#ffc2b7"  
  
android:angle="270" />  
  
<stroke  
  
android:width="2dp"  
  
android:color="#dcdcdc" />  
  
<corners  
  
android:radius="2dp" />  
  
<padding  
  
android:left="10dp"  
  
android:top="10dp"  
  
android:right="10dp"  
  
android:bottom="10dp" />  
  
</shape>  
  
</item> <item>   
  
<shape>  
  
<gradient  
  
android:startColor="#ff9d77"  
  
android:endColor="#ff9d77"  
  
android:angle="270" />  
  
<stroke  
  
android:width="2dp"  
  
android:color="#fad3cf" />  
  
<corners  
  
android:radius="2dp" />  
  
<padding  
  
android:left="10dp"  
  
android:top="10dp"  
  
android:right="10dp"  
  
android:bottom="10dp" />  
  
</shape>  
  
</item>  
  
</selector> 注意Android123提示大家，以上几个item的区别主要是体现在state\_pressed按下或state\_focused获得焦点时，当当来判断显示什么类型，而没有state\_xxx属性的item可以看作是常规状态下。

15. Android开发者应该保持以下特质

Android123推荐新手应该遵循 1. 深读SDK文档 2. 深读SDK的APIDemo和Samples 3. 掌握GIT开源代码 4. 多了解Android开源项目，学习别人的手法写程序。

16. Android数组排序常见方法

Android的数组排序方式基本上使用了Sun原生的Java API实现，常用的有Comparator接口实现compare方法和Comparable接口的compareTo方法，我们对于一个数组列表比如ArrayList可以通过这两个接口进行排序和比较，这里Android123给大家一个例子 private final Comparator cwjComparator = new Comparator() { private final Collator collator = Collator.getInstance();  
  
public final int compare(Object a, Object b) {  
  
CharSequence a = ((Item) a).sName;  
  
CharSequence b = ((Item) b).sID;  
  
return collator.compare(a, b);  
  
}  
  
}; 我们的ArrayList对象名为mList，则执行排序可以调用方法 Collections.sort(mList, cwjComparator);

17.Android控件TextProgressBar进度条上显文字

Android系统的进度条控件默认的设计的不是很周全，比如没有包含文字的显示，那么如何在Android进度条控件上显示文字呢? 来自Google内部的代码来了解下，主要使用的addView这样的方法通过覆盖一层Chronometer秒表控件来实现，整个代码如下 public class TextProgressBar extends RelativeLayout implements OnChronometerTickListener {  
  
public static final String TAG = "TextProgressBar";  
  
static final int CHRONOMETER\_ID = android.R.id.text1;  
  
static final int PROGRESSBAR\_ID = android.R.id.progress;  
  
Chronometer mChronometer = null;  
  
ProgressBar mProgressBar = null;  
  
long mDurationBase = -1;  
  
int mDuration = -1; boolean mChronometerFollow = false;  
  
int mChronometerGravity = Gravity.NO\_GRAVITY;  
  
public TextProgressBar(Context context, AttributeSet attrs, int defStyle) {  
  
super(context, attrs, defStyle);  
  
} public TextProgressBar(Context context, AttributeSet attrs) {  
  
super(context, attrs);  
  
} public TextProgressBar(Context context) {  
  
super(context);  
  
} //Android开发网提示关键部分在这里 @Override  
  
public void addView(View child, int index, ViewGroup.LayoutParams params) {  
  
super.addView(child, index, params);  
  
int childId = child.getId();  
  
if (childId == CHRONOMETER\_ID && child instanceof Chronometer) {  
  
mChronometer = (Chronometer) child;  
  
mChronometer.setOnChronometerTickListener(this);  
  
// Check if Chronometer should move with with ProgressBar  
  
mChronometerFollow = (params.width == ViewGroup.LayoutParams.WRAP\_CONTENT);  
  
mChronometerGravity = (mChronometer.getGravity() & Gravity.HORIZONTAL\_GRAVITY\_MASK);  
  
} else if (childId == PROGRESSBAR\_ID && child instanceof ProgressBar) {  
  
mProgressBar = (ProgressBar) child;  
  
}  
  
} @android.view.RemotableViewMethod  
  
public void setDurationBase(long durationBase) {  
  
mDurationBase = durationBase;  
  
if (mProgressBar == null || mChronometer == null) {  
  
throw new RuntimeException("Expecting child ProgressBar with id " +  
  
"'android.R.id.progress' and Chronometer id 'android.R.id.text1'");  
  
}  
  
// Update the ProgressBar maximum relative to Chronometer base  
  
mDuration = (int) (durationBase - mChronometer.getBase());  
  
if (mDuration <= 0) {  
  
mDuration = 1;  
  
}  
  
mProgressBar.setMax(mDuration);  
  
}  
  
public void onChronometerTick(Chronometer chronometer) {  
  
if (mProgressBar == null) {  
  
throw new RuntimeException(  
  
"Expecting child ProgressBar with id 'android.R.id.progress'");  
  
}  
  
// Stop Chronometer if we're past duration  
  
long now = SystemClock.elapsedRealtime();  
  
if (now >= mDurationBase) {  
  
mChronometer.stop();  
  
} int remaining = (int) (mDurationBase - now);  
  
mProgressBar.setProgress(mDuration - remaining);  
  
if (mChronometerFollow) {  
  
RelativeLayout.LayoutParams params;  
  
params = (RelativeLayout.LayoutParams) mProgressBar.getLayoutParams();  
  
int contentWidth = mProgressBar.getWidth() - (params.leftMargin + params.rightMargin);  
  
int leadingEdge = ((contentWidth \* mProgressBar.getProgress()) /  
  
mProgressBar.getMax()) + params.leftMargin;  
  
int adjustLeft = 0;  
  
int textWidth = mChronometer.getWidth();  
  
if (mChronometerGravity == Gravity.RIGHT) {  
  
adjustLeft = -textWidth;  
  
} else if (mChronometerGravity == Gravity.CENTER\_HORIZONTAL) {  
  
adjustLeft = -(textWidth / 2);  
  
}  
  
leadingEdge += adjustLeft;  
  
int rightLimit = contentWidth - params.rightMargin - textWidth;  
  
if (leadingEdge < params.leftMargin) {  
  
leadingEdge = params.leftMargin;  
  
} else if (leadingEdge > rightLimit) {  
  
leadingEdge = rightLimit;  
  
}  
  
params = (RelativeLayout.LayoutParams) mChronometer.getLayoutParams();  
  
params.leftMargin = leadingEdge;  
  
mChronometer.requestLayout();  
  
}  
  
}  
  
}

18. Android内存管理-SoftReference的使用

很多时候我们需要考虑Android平台上的内存管理问题，Dalvik VM给每个进程都分配了一定量的可用堆内存，当我们处理一些耗费资源的操作时可能会产生OOM错误(OutOfMemoryError)这样的异常，Android123观察了下国内的类似Market客户端设计，基本上都没有采用很好的内存管理机制和缓存处理。 如果细心的网友可能发现Android Market客户端载入时，每个列表项的图标是异步刷新显示的，但当我们快速的往下滚动到一定数量比如50个，再往回滚动时可能我们看到了部分App的图标又重新开始加载，当然这一过程可能是从SQLite数据库中缓存的，但是在内存中已经通过类似SoftReference的方式管理内存。 在Java中内存管理，引用分为四大类，强引用HardReference、弱引用WeakReference、软引用SoftReference和虚引用PhantomReference。它们的区别也很明显，HardReference对象是即使虚拟机内存吃紧抛出OOM也不会导致这一引用的对象被回收，而WeakReference等更适合于一些数量不多，但体积稍微庞大的对象，在这四个引用中，它是最容易被垃圾回收的，而我们对于显示类似Android Market中每个应用的App Icon时可以考虑使用SoftReference来解决内存不至于快速回收，同时当内存短缺面临Java VM崩溃抛出OOM前时，软引用将会强制回收内存，最后的虚引用一般没有实际意义，仅仅观察GC的活动状态，对于测试比较实用同时必须和ReferenceQueue一起使用。 对于一组数据，我们可以通过HashMap的方式来添加一组SoftReference对象来临时保留一些数据，同时对于需要反复通过网络获取的不经常改变的内容，可以通过本地的文件系统或数据库来存储缓存，希望给国内做App Store这样的客户端一些改进建议。

19. 反射在Android开发中的利弊

由于Android 2.2的推出，很多新的API加入导致很多项目移植需要考虑使用Java的反射机制Reflection来动态调用，动态调用的好处就是不需要使用引用文件，直接通过JDK中声明好的方法直接调用,本身原理基于JVM的，从Java 1.5开始支持，原理上就是根据类名而不实例化对象的情况下，获得对象的方法或属性而直接调用。 Android开发时反射能帮助我们多少? 1. 有些网友可能发现Android的SDK比较封闭，很多敏感的方法常规的用户无法编译，我们如果翻看了代码直接在反射中声明动态调用即可。比如很多internal或I开头的AIDL接口均可以通过反射轻松调用。 2. 反射对于Android123来说更重要的是考虑到应用的兼容性，我们目前主要兼容从Android 1.5到2.2的项目，API Level从3到8可以方便的扩充，调用前我们预留一个标志位声明该API的最低以及最高的API Level为多少可以调用。 3. 对于调试Java的反射是功臣了，在Logcat中我们可以看到出错的地方肯定有类似java.lang.reflect.XXX的字样，这种自检机制可以帮助我们方便的调试Android应用程序。 反射的缺点有哪些? 1. 因为是动态执行的，效率自然没有预编译时引用现有的库效率高，就像平时我们Win32开发时，可以不用h文件，直接通过GetProcAddress一样去动态获取方法的地址。当然效率要根据复杂程度而决定，一般稍微复杂的处理性能损失可能超过20%，对于一些复杂的涉及Java自动类型转换判断，执行时间可能是直接引用的上千倍，所以最终我们调试时必须考虑性能问题。 2. 因为反射是动态的，所以需要处理很多异常，不然Dalvik崩溃出Force Close的概率会大很多，很简单的一个反射就需要至少3个异常捕获，本身try-catch效率就不是很高，自然进一步影响运行效率，对于Android开发我们必须考虑这些问题。 3. 反射因为导致代码臃肿，自然稍微复杂的几个方法实用反射将会导致代码可读性和维护性降低，如果很抽象的调用Android开发网强烈不推荐这种方法。 最后要说的是Reflection并不是Java的专利，微软的.Net也同样支持，同时更多的动态语言如Ruby等均支持这一特性。

20.AsyncTask对比Thread加Handler

很多网友可能发现Android平台很多应用使用的都是AsyncTask，而并非Thread和Handler去更新UI，这里Android123给大家说下他们到底有什么区别，我们平时应该使用哪种解决方案。从Android 1.5开始系统将AsyncTask引入到android.os包中，过去在很早1.1和1.0 SDK时其实官方将其命名为UserTask，其内部是JDK 1.5开始新增的concurrent库，做过J2EE的网友可能明白并发库效率和强大性，比Java原始的Thread更灵活和强大，但对于轻量级的使用更为占用系统资源。Thread是Java早期为实现多线程而设计的，比较简单不支持concurrent中很多特性在同步和线程池类中需要自己去实现很多的东西，对于分布式应用来说更需要自己写调度代码，而为了Android UI的刷新Google引入了Handler和Looper机制，它们均基于消息实现，有事可能消息队列阻塞或其他原因无法准确的使用。 Android开发网推荐大家使用AsyncTask代替Thread+Handler的方式，不仅调用上更为简单，经过实测更可靠一些，Google在Browser中大量使用了异步任务作为处理耗时的I/O操作，比如下载文件、读写数据库等等，它们在本质上都离不开消息，但是AsyncTask相比Thread加Handler更为可靠，更易于维护，但AsyncTask缺点也是有的比如一旦线程开启即dobackground方法执行后无法给线程发送消息，仅能通过预先设置好的标记来控制逻辑，当然可以通过线程的挂起等待标志位的改变来通讯，对于某些应用Thread和Handler以及Looper可能更灵活。

21. Android Drawable叠加处理方法

大家可能知道Bitmap的叠加处理在Android平台中可以通过Canvas一层一层的画就行了，而Drawable中如何处理呢? 除了使用BitmapDrawable的getBitmap方法将Drawable转换为Bitmap外，今天Android123给大家说下好用简单的LayerDrawable类，LayerDrawable顾名思义就是层图形对象。下面直接用一个简单的代码表示: Bitmap bm = BitmapFactory.decodeResource(getResources(),R.drawable.cwj);  
  
Drawable[] array = new Drawable[3]; array[0] = new PaintDrawable(Color.BLACK); //黑色  
  
array[1] = new PaintDrawable(Color.WHITE); //白色  
  
array[2] = new BitmapDrawable(bm); //位图资源  
  
LayerDrawable ld = new LayerDrawable(array); //参数为上面的Drawable数组  
  
ld.setLayerInset(1, 1, 1, 1, 1); //第一个参数1代表数组的第二个元素，为白色  
  
ld.setLayerInset(2, 2, 2, 2, 2); //第一个参数2代表数组的第三个元素，为位图资源  
  
mImageView.setImageDrawable(ld); 上面的方法中LayerDrawable是关键，Android开发网提示setLayerInset方法原型为public void setLayerInset (int index, int l, int t, int r, int b) 其中第一个参数为层的索引号，后面的四个参数分别为left、top、right和bottom。对于简单的图片合成我们可以将第一和第二层的PaintDrawable换成BitmapDrawable即可实现简单的图片合成。

22. onRetainNonConfigurationInstance和getLastNonConfigurationInstance

很多网友可能知道Android横竖屏切换时会触发onSaveInstanceState，而还原时会产生onRestoreInstanceState，但是Android的Activity类还有一个方法名为onRetainNonConfigurationInstance和getLastNonConfigurationInstance这两个方法。 我们可以通过 onRetainNonConfigurationInstance 代替 onSaveInstanceState，比如距离2 @Override  
  
public Object onRetainNonConfigurationInstance()  
  
{   
  
//这里需要保存的内容，在切换时不是bundle了，我们可以直接通过Object来代替  
  
return obj;  
  
} 在恢复窗口时，我们可以不使用 onRestoreInstanceState，而代替的是 getLastNonConfigurationInstance 方法。我们可以直接在onCreate中使用，比如 Object obj = getLastNonConfigurationInstance(); 最终obj的内容就是上次切换时的内容。 这里Android123提醒大家，每次Activity横竖屏切换时onCreate方法都会被触发。

23. Android中String资源文件的format方法

很多时候我们感性Google在设计Android时遵守了大量MVC架构方式，可以让写公共代码、美工和具体逻辑开发人员独立出来。有关Android的资源文件values/strings.xml中如何实现格式化字符串呢? 这里Android123举个简单的例子，以及最终可能会用到哪些地方。 <?xml version="1.0" encoding="utf-8"?>   
  
<resources>   
  
<string name="app\_name">cwj\_Demo</string>   
  
<string name="hello">android开发网</string>   
  
</resources> 上面是一段简单的字符串资源文件，没有用到格式化，因为比较简单直接描述了意思，当我们设计一个类似 Delete xxx File ? 的时候，我们可能需要在Java中动态获取 xxx 的名称，所以定义资源时使用格式化可以轻松解决，不需要一堆String去拼接或StringBuffer一个一个append这样的愚蠢方法，看例子 <string name="alert">Delete %1$s File</string> 这里%1$s代表这是一个字符串型的，如果是整数型可以写为%1$d，类似printf这样的格式化字符串函数，当然如果包含了多个需要格式化的内容，则第二个可以写为%2$s或%2$d了，那么最终在Java中如何调用呢? 看下面的例子: 例一: 整数型的 <string name="alert">I am %1$d years old</string> 定义的是这样的 当然，我们杜绝意外情况，比如冒出个secret这样的string类型的，注意上面是%1$d不是%1$s，所以默认标准的合并成为 int nAge=23; String sAgeFormat = getResources().getString(R.string.alert); String sFinalAge = String.format(sAgeFormat, nAge); 这样执行完后，就组成了 I am 23 years old，是不是很方便啊. 当然了，下面看下String字符串时的情况. 例二: 字符串型的 String sName="cwj" String sCity="Shanghai" 资源定义为 <string name="alert2">My name is %1$s , I am form %2$s</string> 则Java中只需要 String sInfoFormat = getResources().getString(R.string.alert2); String sFinalInfo=String.format(sInfoFormat, sName, sCity); 我们看到了整个，整个定义类似MFC的CString::Format或Mac OS中的NSLog，但是需要显示类似C#中那样显示的标出参数的数字，比如%1或%n，这里数字代表参数的第n个。本行最终sFinalInfo显示的内容为 My name is cwj , I am form Shanghai 。当然了你有什么不懂的地方可以来函至 [android123@163.com](mailto:android123@163.com)

24. Android工程内嵌资源文件的两种方法

Android软件一般处理大的资源通过sdcard比如在线下载资源到sdcard，而apk中内嵌资源或二进制文件时一般使用下面的两种方法: 方法一 res/raw目录下存放，比如cwj.dat一个二进制文件，我们可以读取可以直接 InputStream is=context.getResources().openRawResource(R.raw.cwj); 方法二 工程根目录下的assets文件夹中存放，比如assets/cwj.dat 这样我们使用下面的代码 AssetManager am = context.getAssets();   
  
InputStream is = am.open(cwj.dat); 这里Android123提示大家Google的Android系统处理Assert有个bug，在AssertManager中不能处理单个超过1MB的文件，不然会报异常具体数值大家可以测试下传个稍大的文件，我们在两年前的文章中有提到，而第一种raw没这个限制可以放个4MB的Mp3文件没问题。

25. Android自定义View以及layout属性全攻略

对于Android系统的自定义View可能大家都熟悉了，对于自定义View的属性添加，以及Android的Layout的命名空间问题，很多网友还不是很清楚，今天Android123一起再带大家温习一下 CwjView myView=new CwjView(context); 如果用于游戏或整个窗体的界面，我们可能直接在onCreate中setContentView(myView); 当然如果是控件，我们可能会需要从Layout的xml中声明，比如 <cn.com.android123.CwjView  
  
android:layout\_width="wrap\_content"  
  
android:layout\_height="wrap\_content"  
  
/> 当然，我们也可以直接从父类声明比如 <View class="cn.com.android123.CwjView"  
  
android:layout\_width="wrap\_content"  
  
android:layout\_height="wrap\_content"  
  
/> 上面我们仅用了父类View的两个属性，均来自android命名空间，而名称为layout\_width或layout\_height，我们自定义的控件可能有更多的功能，比如 <cn.com.android123.CwjView  
  
android:layout\_width="wrap\_content"  
  
android:layout\_height="wrap\_content"  
  
**cwj:age="22"  
  
cwj:university="sjtu"  
  
cwj:city="shanghai"**/> 我们可以看到上面的三个属性，是我们自定义的。作为标准xml规范，可能还包含了类似 xmlns:**android**="http://schemas.android.com/apk/res/**android**" 这样的语句，对于定义完整的View，我们的命名空间为cwj，这里可以写为 xmlns:**cwj**=http://schemas.android.com/apk/res/**cn.com.android123.cwjView** 或 xmlns:**cwj**=http://schemas.android.com/apk/res/**android** 都可以**。** 对于定义的cwj命名空间和age、university以及city的三个属性我们如何定义呢? 在工程的res/values目录中我们新建一个cwj\_attr.xml文件，编码方式为utf-8是一个好习惯，内容如下 <?xml version="1.0" encoding="utf-8" ?>  
  
<resources>  
  
<declare-styleable name="CwjView">  
  
<attr name="age" format="integer" />  
  
<attr name="city" format="string" />  
  
<attr name="university" format="string" />  
  
</declare-styleable>  
  
</resources> 这里我们可能对format不是很熟悉，目前Android系统内置的格式类型有integer比如ProgressBar的进度值，float比如RatingBar的值可能是3.5颗星，boolean比如ToggleButton的是否勾选，string比如TextView的text属性，当然除了我们常见的基础类型外，Android的属性还有特殊的比如color是用于颜色属性的，可以识别为#FF0000等类型，当然还有dimension的尺寸类型，比如23dip，15px，18sp的长度单位，还有一种特殊的为reference，一般用于引用@+id/cwj @drawable/xxx这样的类型。 当然什么时候用reference呢? 我们就以定义一个颜色为例子， <attr name="red" format="color|reference" /> 这里我们用了逻辑或的运算符，定义的红色是颜色类型的，同时可以被引用 当然，对于我们自定义的类中，我们需要使用一个名为obtainStyledAttributes的方法来获取我们的定义。在我们自定义View的构造方法(Context context, AttributeSet attrs)的重载类型中可以用 public CwjView(Context context, AttributeSet attrs) {  
  
super(context, attrs);  
  
TypedArray a = context.obtainStyledAttributes(attrs,  
  
R.styleable.cwj\_attr);  
  
mAge = a.getInteger(R.styleable.CwjView\_age, 22);  
  
mCity = a.getString(R.styleable.CwjView\_city, "shanghai");  
  
mUniversity= a.getString(R.styleable.CwjView\_university, "sjtu");  
  
a.recycle(); //Android123提示大家不要忘了回收资源 } 这样类的全局成员变量 mAge、mCity就获取了我们需要的内容，当然根据layout中的数值我们自定义的CwjView需要动态的处理一些数据的情况，可以使用AttributeSet类的getAttributeResourceValue方法获取。 public CwjView(Context context, AttributeSet attrs)  
  
{  
  
super(context, attrs);  
  
resId = attrs.getAttributeResourceValue("cn.com.android123.CwjView", "age", 100);   
  
resId = attrs.getAttributeResourceValue("cn.com.android123.CwjView", "city", "shanghai");  
  
//resID就可以任意使用了  
  
} 以上两种方法中，参数的最后一个数值为默认的，如果您有不明白的地方可以来函到[android123@163.com](mailto:android123@163.com) 我们会在第一时间回复。

26. 自定义Android主题风格theme.xml方法

在Android中可以通过自定义主题风格方式来实现个性化以及复用，首先我们创建theme.xml主题文件，保存位置为工程的res/values/theme.xml ，这里我们可以可以为主题起一个名称，比如CWJ，这里去除了xml的文件头<?xml version="1.0" encoding="utf-8"?>这行，我们在工程中只需在androidmanifest.xml文件的Activity节点中加入android:theme="@style/Theme.CWJ" 属性，则这个Activity就使用了这种主题风格，整个xml的关键代码如下: <resources>  
  
<style name="Theme.CWJ" parent="android:Theme">  
  
<item name="android:windowBackground">@drawable/android123</item>  
  
</style>  
  
</resources> 其中上面的代码中，我们定义设置全局android:windowBackground即背景值为/res/drawable中的android123图片为背景，更多的属性定义可以参考view的layout xml属性设置，比如我们设置所有字体颜色、大体大小和样式，可以在style节点中加入 <item name="android:textColor">#fff</item>  
  
<item name="android:textSize">14sp</item>  
  
<item name="android:textStyle">bold</item> 当然我们可以将上面的android123的图片改进下，使用一个xml文件替代，比如使用bitmap对象，则/res/drawable/android123.xml的完整代码变为 <?xml version="1.0" encoding="utf-8"?> <bitmap xmlns:android="<http://schemas.android.com/apk/res/android>"  
  
android:src="@drawable/cwj\_image"  
  
android:tileMode="repeat" /> 这里我们使用了一个bitmap对象来解析cwj\_image图片，当然这里可以识别各种类型的图片，其中android:tileMode是bitmap的内部属性，其中tileMode设置为repeat代表重复，这样可以节省bitmap资源，比如我们的背景是一层楼，那么全屏可以显示同样的为5层效果，而图片仅是一层大小，对于资源利用相对更高。 当然bitmap的属性tileMode的值为repeat外还有其他的值比如clamp、mirror，这些值并没有在SDK中并没有找到定义，通过上次Android开发网的 [Android自定义View以及layout属性全攻略](http://www.android123.com.cn/androidkaifa/591.html)一文，我们可以联想到bitmap属于android.graphics.Bitmap 包，由于是android框架，所以下载git的base包，找到该类，类的实例化时android123已经在 [Android自定义View以及layout属性全攻略](http://www.android123.com.cn/androidkaifa/591.html) 说的很清楚，所以我们定位到res\values中找到attr.xml有关bitmap的定义即可，有关bitmap的更多属性如 antialias、filter和dither都可以找到使用。

27. android调试工具monkey压力测试实战

很多Android开发者可能因为没有充分测试自己的软件造成很容易出现FC(Force Close)的问题，这里我们可以通过使用Android固件中自带的monkey工具来做软件的压力测试，monkey工具可以模拟各种按键，触屏，轨迹球、activity等事件，这里Android123提示大家说白了monkey就是一个小猴子随机狂玩你的android软件，看看会不会产生异常。 具体的使用我们通过Android SDK给我们的adb调试桥链接设备或模拟器，进入Linux Shell状态，当然我们可以输入adb shell获取设备的shell，也可以直接通过adb命令执行，比如说adb shell monkey来查看monkey工具中的参数说明，如图: 我们要测试的apk文件要在android设备中已经安装，当然模拟器中也可以测试的。执行adb shell monkey -p cn.com.android123.cwj -v 100 我们执行这句的中包含了p参数，这里代表已安装软件的packageName，而v代表查看monkey生成的详细随机事件名，最后的数字100为我们测试的随机事件数量为100.有关更多的测试方法，请查看上图中的参数，整个测试比较简单单很有效，不妨试试。

28. 自定义View

有关Android的自定义View的框架今天我们一起讨论下，对于常规的游戏，我们在View中需要处理以下几种问题: 1.控制事件 2.刷新View 3. 绘制View 1. 对于控制事件今天我们只处理按键事件onKeyDown，以后的文章中将会讲到屏幕触控的具体处理onTouchEvent以及Sensor重力感应等方法。 2. 刷新view的方法这里主要有invalidate(int l, int t, int r, int b) 刷新局部，四个参数分别为左、上、右、下。整个view刷新 invalidate()，刷新一个矩形区域 invalidate(Rect dirty) ，刷新一个特性Drawable， invalidateDrawable(Drawable drawable) ，执行invalidate类的方法将会设置view为无效，最终导致onDraw方法被重新调用。由于今天的view比较简单，Android123提示大家如果在线程中刷新，除了使用handler方式外，可以在Thread中直接使用postInvalidate方法来实现。 3. 绘制View主要是onDraw()中通过形参canvas来处理，相关的绘制主要有drawRect、drawLine、drawPath等等。view方法内部还重写了很多接口，其回调方法可以帮助我们判断出view的位置和大小，比如onMeasure(int, int) Called to determine the size requirements for this view and all of its children. 、onLayout(boolean, int, int, int, int) Called when this view should assign a size and position to all of its children 和onSizeChanged(int, int, int, int) Called when the size of this view has changed. 具体的作用，大家可以用Logcat获取当view变化时每个形参的变动。 下面cwjView是我们为今后游戏设计的一个简单自定义View框架，我们可以看到在Android平台自定义view还是很简单的，同时Java支持多继承可以帮助我们不断的完善复杂的问题。 public class cwjView extends View { public cwjView(Context context) {  
  
super(context);  
  
setFocusable(true); //允许获得焦点  
  
setFocusableInTouchMode(true); //获取焦点时允许触控  
  
} @Override  
  
protected Parcelable onSaveInstanceState() { //处理窗口保存事件  
  
Parcelable pSaved = super.onSaveInstanceState();  
  
Bundle bundle = new Bundle();  
  
//dosomething  
  
return bundle;  
  
}  
  
@Override  
  
protected void onRestoreInstanceState(Parcelable state) { //处理窗口还原事件  
  
Bundle bundle = (Bundle) state; //dosomething  
  
super.onRestoreInstanceState(bundle.getParcelable("cwj"));  
  
return;  
  
}  
  
@Override  
  
protected void onSizeChanged(int w, int h, int oldw, int oldh) //处理窗口大小变化事件  
  
{  
  
super.onSizeChanged(w, h, oldw, oldh);  
  
} @Override  
  
protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)   
  
{  
  
super.onMeasure(widthMeasureSpec, heightMeasureSpec); //如果不让父类处理记住调用setMeasuredDimension  
  
}  
  
@Override  
  
protected void onLayout (boolean changed, int left, int top, int right, int bottom)  
  
{  
  
super.onLayout (changed,left,top, ight,bottom) ;  
  
} @Override  
  
protected void onDraw(Canvas canvas) {  
  
Paint bg = new Paint();  
  
bg.setColor(Color.Red);  
  
canvas.drawRect(0, 0, getWidth()/2, getHeight()/2, bg); //将view的左上角四分之一填充为红色   
  
} @Override  
  
public boolean onTouchEvent(MotionEvent event) {  
  
return super.onTouchEvent(event); //让父类处理屏幕触控事件  
  
} @Override  
  
public boolean onKeyDown(int keyCode, KeyEvent event) { //处理按键事件，响应的轨迹球事件为 public boolean onTrackballEvent (MotionEvent event)  
  
switch (keyCode) {  
  
case KeyEvent.KEYCODE\_DPAD\_UP:  
  
break;  
  
case KeyEvent.KEYCODE\_DPAD\_DOWN:  
  
break;  
  
case KeyEvent.KEYCODE\_DPAD\_LEFT:  
  
break;  
  
case KeyEvent.KEYCODE\_DPAD\_RIGHT:  
  
break;  
  
case KeyEvent.KEYCODE\_DPAD\_CENTER: //处理中键按下  
  
break;  
  
default:  
  
return super.onKeyDown(keyCode, event);  
  
}  
  
return true;  
  
} } 上面我们可以看到onMeasure使用的是父类的处理方法，如果我们需要解决自定义View的大小，可以尝试下面的方法 @Override  
  
protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)   
  
{  
  
height = View.MeasureSpec.getSize(heightMeasureSpec);   
  
width = View.MeasureSpec.getSize(widthMeasureSpec);   
  
setMeasuredDimension(width,height); //这里面是原始的大小，需要重新计算可以修改本行 //dosomething }

29. Canvas和Paint实例

昨天我们在[Android游戏开发之旅三 View详解](http://www.android123.com.cn/androidkaifa/541.html)中提到了onDraw方法，有关详细的实现我们今天主要说下Android的Canvas和Paint对象的使用实例。 Canvas类主要实现了屏幕的绘制过程，其中包含了很多实用的方法，比如绘制一条路径、区域、贴图、画点、画线、渲染文本，下面是Canvas类常用的方法，当然Android开发网提示大家很多方法有不同的重载版本，参数更灵活。 void drawRect(RectF rect, Paint paint) //绘制区域，参数一为RectF一个区域 void drawPath(Path path, Paint paint) //绘制一个路径，参数一为Path路径对象 void drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint) //贴图，参数一就是我们常规的Bitmap对象，参数二是源区域(Android123提示这里是bitmap)，参数三是目标区域(应该在canvas的位置和大小)，参数四是Paint画刷对象，因为用到了缩放和拉伸的可能，当原始Rect不等于目标Rect时性能将会有大幅损失。 void drawLine(float startX, float startY, float stopX, float stopY, Paint paint) //画线，参数一起始点的x轴位置，参数二起始点的y轴位置，参数三终点的x轴水平位置，参数四y轴垂直位置，最后一个参数为Paint画刷对象。 void drawPoint(float x, float y, Paint paint) //画点，参数一水平x轴，参数二垂直y轴，第三个参数为Paint对象。  
  
void drawText(String text, float x, float y, Paint paint) //渲染文本，Canvas类除了上面的还可以描绘文字，参数一是String类型的文本，参数二x轴，参数三y轴，参数四是Paint对象。 void drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint) //在路径上绘制文本，相对于上面第二个参数是Path路径对象 从上面来看我们可以看出Canvas绘制类比较简单同时很灵活，实现一般的方法通常没有问题，同时可以叠加的处理设计出一些效果，不过细心的网友可能发现最后一个参数均为Paint对象。如果我们把Canvas当做绘画师来看，那么Paint就是我们绘画的工具，比如画笔、画刷、颜料等等。 Paint类常用方法: void setARGB(int a, int r, int g, int b) 设置Paint对象颜色，参数一为alpha透明通道  
  
void setAlpha(int a) 设置alpha不透明度，范围为0~255 void setAntiAlias(boolean aa) //是否抗锯齿 void setColor(int color) //设置颜色，这里Android内部定义的有Color类包含了一些常见颜色定义  
  
.  
  
void setFakeBoldText(boolean fakeBoldText) //设置伪粗体文本  
  
void setLinearText(boolean linearText) //设置线性文本  
  
PathEffect setPathEffect(PathEffect effect) //设置路径效果  
  
Rasterizer setRasterizer(Rasterizer rasterizer) //设置光栅化  
  
Shader setShader(Shader shader) //设置阴影 void setTextAlign(Paint.Align align) //设置文本对齐  
  
void setTextScaleX(float scaleX) //设置文本缩放倍数，1.0f为原始  
  
void setTextSize(float textSize) //设置字体大小  
  
Typeface setTypeface(Typeface typeface) //设置字体，Typeface包含了字体的类型，粗细，还有倾斜、颜色等。 void setUnderlineText(boolean underlineText) //设置下划线  
  
最终Canvas和Paint在onDraw中直接使用 @Override  
  
protected void onDraw(Canvas canvas) { Paint paintRed=new Paint(); paintRed.setColor(Color.Red); canvas.drawPoint(11,3,paintRed); //在坐标11,3上画一个红点  
  
} 下一次Android123将会具体讲到强大的Path路径，和字体Typeface相关的使用。

30. View类详解

在Android游戏开发之旅二中我们讲到了View和SurfaceView的区别，今天Android123从View类开始着重的介绍Android图形显示基类的相关方法和注意点。 自定义View的常用方法: onFinishInflate() 当View中所有的子控件均被映射成xml后触发 onMeasure(int, int) 确定所有子元素的大小 onLayout(boolean, int, int, int, int) 当View分配所有的子元素的大小和位置时触发 onSizeChanged(int, int, int, int) 当view的大小发生变化时触发 onDraw(Canvas) view渲染内容的细节 onKeyDown(int, KeyEvent) 有按键按下后触发 onKeyUp(int, KeyEvent) 有按键按下后弹起时触发 onTrackballEvent(MotionEvent) 轨迹球事件 onTouchEvent(MotionEvent) 触屏事件 onFocusChanged(boolean, int, Rect) 当View获取或失去焦点时触发 onWindowFocusChanged(boolean) 当窗口包含的view获取或失去焦点时触发 onAttachedToWindow() 当view被附着到一个窗口时触发 onDetachedFromWindow() 当view离开附着的窗口时触发，Android123提示该方法和 onAttachedToWindow() 是相反的。 onWindowVisibilityChanged(int) 当窗口中包含的可见的view发生变化时触发 以上是View实现的一些基本接口的回调方法，一般我们需要处理画布的显示时，重写onDraw(Canvas)用的的是最多的： @Override  
  
protected void onDraw(Canvas canvas) {  
  
//这里我们直接使用canvas对象处理当前的画布，比如说使用Paint来选择要填充的颜色 Paint paintBackground = new Paint();  
  
paintBackground.setColor(getResources().getColor(R.color.xxx)); //从Res中找到名为xxx的color颜色定义  
  
canvas.drawRect(0, 0, getWidth(), getHeight(), paintBackground); //设置当前画布的背景颜色为paintBackground中定义的颜色，以0,0作为为起点，以当前画布的宽度和高度为重点即整块画布来填充。 具体的请查看Android123未来讲到的Canvas和Paint，在Canvas中我们可以实现画路径，图形，区域，线。而Paint作为绘画方式的对象可以设置颜色，大小，甚至字体的类型等等。 } 当然还有就是处理窗口还原状态问题(一般用于横竖屏切换)，除了在Activity中可以调用外，开发游戏时我们尽量在View中使用类似 @Override  
  
protected Parcelable onSaveInstanceState() {  
  
Parcelable p = super.onSaveInstanceState();  
  
Bundle bundle = new Bundle();  
  
bundle.putInt("x", pX);  
  
bundle.putInt("y", pY);  
  
bundle.putParcelable("android123\_state", p);  
  
return bundle;  
  
}  
  
@Override  
  
protected void onRestoreInstanceState(Parcelable state) {   
  
Bundle bundle = (Bundle) state;  
  
dosomething(bundle.getInt("x"), bundle.getInt("y")); //获取刚才存储的x和y信息  
  
super.onRestoreInstanceState(bundle.getParcelable("android123\_state"));  
  
return;  
  
} 在View中如果需要强制调用绘制方法onDraw，可以使用invalidate()方法，它有很多重载版本，同时在线程中的postInvailidate()方法将在Android游戏开发之旅六中的 自定义View完整篇讲到。

31. View和SurfaceView

在Android游戏当中充当主要的除了控制类外就是显示类，在J2ME中我们用Display和Canvas来实现这些，而Google Android中涉及到显示的为view类，Android游戏开发中比较重要和复杂的就是显示和游戏逻辑的处理。这里我们说下android.view.View和android.view.SurfaceView。SurfaceView是从View基类中派生出来的显示类，直接子类有GLSurfaceView和VideoView，可以看出GL和视频播放以及Camera摄像头一般均使用SurfaceView，到底有哪些优势呢? SurfaceView可以控制表面的格式，比如大小，显示在屏幕中的位置，最关键是的提供了SurfaceHolder类，使用getHolder方法获取，相关的有Canvas lockCanvas()   
  
Canvas lockCanvas(Rect dirty) 、void removeCallback(SurfaceHolder.Callback callback)、void unlockCanvasAndPost(Canvas canvas) 控制图形以及绘制，而在SurfaceHolder.Callback 接口回调中可以通过下面三个抽象类可以自己定义具体的实现，比如第一个更改格式和显示画面。 abstract void surfaceChanged(SurfaceHolder holder, int format, int width, int height)  
  
abstract void surfaceCreated(SurfaceHolder holder)  
  
abstract void surfaceDestroyed(SurfaceHolder holder)  
  
对于Surface相关的，Android底层还提供了GPU加速功能，所以一般实时性很强的应用中主要使用SurfaceView而不是直接从View构建，同时Android123未来后面说到的OpenGL中的GLSurfaceView也是从该类实现。

32. Android程序内存管理必读

很多开发者都是从J2ME或J2EE上过来的，对于内存的使用和理解并不是很到位，Android开发网本次给大家一些架构上的指导，防止出现豆腐渣工程的出现。Android作为以Java语言为主的智能平台对于我们开发一些高性能和质量的软件来说了解Android程序内存管理机制是必须的。 Android的Dalvik VM在基础方面和Sun JVM没有什么大的区别仅仅是字节码的优化，我们要知道什么时候用gc什么时候用recycle以及到底用不用finalization，因为Java对内存的分配只需要new开发者不需要显示的释放内存，但是这样造成的内存泄露问题的几率反而更高。 1.对于常规开发者而言需要了解 Java的四种引用方式，比如强引用，软引用，弱引用以及虚引用。一些复杂些的程序在长期运行很可能出现类似OutOfMemoryError的异常。 2.并不要过多的指望gc，不用的对象可以显示的设置为空，比如obj=null，这里Android123提示大家，java的gc使用的是一个有向图，判断一个对象是否有效看的是其他的对象能到达这个对象的顶点，有向图的相对于链表、二叉树来说开销是可想而知。 3.Android为每个程序分配的对内存可以通过Runtime类的totalMemory() freeMemory() 两个方法获取VM的一些内存信息，对于系统heap内存获取，可以通过Dalvik.VMRuntime类的getMinimumHeapSize() 方法获取最小可用堆内存，同时显示释放软引用可以调用该类的gcSoftReferences() 方法，获取更多的运行内存。 4.对于多线程的处理，如果并发的线程很多，同时有频繁的创建和释放，可以通过concurrent类的线程池解决线程创建的效率瓶颈。 5. 不要在循环中创建过多的本地变量。 有关Android和Java的系统性能分析，Android123将在以后的文章中详细讲述如何调试Java分析内存泄露以及Android上的gdb调试器分析得出内存性能改进。

33. Android中内嵌字体实现个性化

在Android中我们的应用可以灵活的内嵌自己的字体文件，实现各个手机上可以正常的显示个性化文字，我们都知道TextView的setTypeface方法可以设置目标文字的显示特性，比如字体、颜色、粗体、斜体等。我们直接找一个TrueTypeFont的字体文件即.ttf，对于Win32系统的用户可以直接在Windows/fonts文件夹中能找到很多。比如微软雅黑就不错，可是体积太大，由于Android的Assets类有单个文件1MB体积的限制，我们先找个英文字体做测试。这里我们将字体文件android123.ttf放到工程的assets文件夹的fonts目录中。 Typeface tf = Typeface.createFromAsset(getAssets(), "fonts/android123.ttf");   
  
TextView tv = (TextView)findViewById(R.id.text); tv.setTypeface(tf); //设置TextView的风格  
  
tv.setText("CWJ Test");   
  
tv.setTextSize(12);   
  
tv.setTextColor(Color.RED);

34. 获取和设置ListView的选择项

获取当前选中项 int curPos = listView.getFirstVisiblePosition(); 当然是用getItemAtPosition(int nPos)方法也可以 ，设置当前选择位置 listView.setSelectedPosition(lastPos); 对于基于AbsListView为基类的ListView等控件均可以使用这种方法。

35. android.text.format文件大小和日期解析类

很多网友可能直接将自己的J2ME项目生硬的移植到Android平台，其实Google为我们提供好了文件大小和时间日期解析类，它位于android.text.format这个包中，它提供了强大的标准化解析方法： 1. IP地址解析类 在android.text.format.Formatter中提供了String formatIpAddress(int addr) 这个方法可以轻松方便的将socket中的int型转成类似127.0.0.1的IP格式，需要注意的是Linux平台的字节顺序，即小字节序、低字节序little-endian。 2. 文件大小解析类 细心的网友可能还看到了android.text.format.Formatter中的formatFileSize方法，该方法String formatFileSize (Context context, long number) ，第二个参数是long型，一般为File对象的最后修改时间或创建时间的方法，最终返回类似 12KB、5Bytes的值，20MB的字符串。 3. 日期时间解析类 ，该类位于android.text.format.DateFormat这个package中，该类提供了Java中的三种时间对象，Android123提示大家下面三种方法为静态可以直接调用，如下: final static CharSequence format(CharSequence inFormat, Date inDate) //传入Date对象  
  
Given a format string and a Date object, returns a CharSequence containing the requested date. final static CharSequence format(CharSequence inFormat, Calendar inDate) //Calendar对象  
  
Given a format string and a Calendar object, returns a CharSequence containing the requested date. final static CharSequence format(CharSequence inFormat, long inTimeInMillis) //long对象  
  
Given a format string and a time in milliseconds since Jan 1, 1970 GMT, returns a CharSequence containing the requested date. 我们可能看到了第一个参数均为inFormat这是一个CharSequence接口的String类型，它提供了灵活的时间格式解析字符串描述，Android开发网提示大家注意大小写要区分，如 April 6, 1970 at 3:23am 例子，那么inFormat参数的写法和最终执行的结果如下对照，下面就以Android123的CWJ生日为例子如下 "MM/dd/yy h:mmaa" -> "11/03/87 11:23am"  
  
"MMM dd, yyyy h:mmaa" -> "Nov 3, 1987 11:23am"  
  
"MMMM dd, yyyy h:mmaa" -> "November 3, 1987 11:23am"  
  
"E, MMMM dd, yyyy h:mmaa" -> "Tues, November 3, 1987 11:23am"  
  
"EEEE, MMMM dd, yyyy h:mmaa" -> "Tuesday, Nov 3, 1987 11:23am" 对于判断一个时间是否为24小时制式可以通过android.text.format.DateFormat类的static boolean is24HourFormat(Context context)方法来判断。

36. Android代码性能优化技巧

目前来说Android 2.2的JIT性能有了本质的提高，不过对于老版本的程序提高Java执行效率还有很多语言特点来说，今天Android123提到的不是语法糖，而是基础的问题，对于Java 1.5之后将会有明显的改进。下面的例子来自SDK: **static class Foo {  
  
int mSplat;  
  
}  
  
Foo[] mArray = ...** 上面的静态类Foo的执行效果和性能，我们分三个方法zero、one和two来做对比。  
  
public void zero() { //大多数人可能简单直接这样写  
  
int sum = 0;  
  
for (int i = 0; i < mArray.length; ++i) {  
  
sum += mArray*.mSplat;  
  
}  
  
}  
  
public void one() { //通过本地对象改进性能  
  
int sum = 0;  
  
Foo[] localArray = mArray;  
  
int len = localArray.length;  
  
for (int i = 0; i < len; ++i) {  
  
sum += localArray.mSplat;  
  
}  
  
}  
  
public void two() { //推荐的方法，通过Java 1.5的新语法特性可以大幅改进性能  
  
int sum = 0;  
  
for (Foo a : mArray) {  
  
sum += a.mSplat;  
  
}   
  
}* ***zero()*** *is slowest, because the JIT can't yet optimize away the cost of getting the array length once for every iteration through the loop.* ***one()*** *is faster. It pulls everything out into local variables, avoiding the lookups. Only the array length offers a performance benefit.* ***two()*** *is fastest for devices without a JIT, and indistinguishable from* ***one()*** *for devices with a JIT. It uses the enhanced for loop syntax introduced in version 1.5 of the Java programming language.*

*37. Android开发注意点 Part One*

*Android已经的很多细节问题我们通过平台开发总结不断完善这个列表，如果你有相关的内容可以联系*[*android123@163.com*](mailto:android123@163.com) *. 一、AssetManager - 已知单个文件处理不能大于1MB，所以如果资源很大，建议使用Zip格式压缩存放。 二、ScrollView中嵌入ListView - 这个作法可能会出现你的ListView仅仅显示1行半。 三、Android自带的Zip处理类对文件名编码无法识别，也没有提供显示的设置方法，在zlib中写死了。 四、使用一些资源对象记住关闭，比如对于文件流对象最后 FileOutputStream os = xxx; try {  
  
//dosomething  
  
} finally {  
  
os.close(); //显示的使用finally关闭文件对象。  
  
} 对于Cursor而言，在移动位置时首先判断Cursor是否为空，最终使用完仍然需要 close方法，如果重用，可以使用deactivate方法释放当前资源，通过requery方法再次查询。 五、SDK中标记为 deprecated 字样的，常规情况下是有更好的方法可以替代，短期内可以放心使用。这些方法一般高版本的SDK都可以向上兼容，目前尚未发现Android放弃某些API的支持。 六、Notification的Intent无法传递到目标的Activity，Service和Broardcast没有测试过，中途需要通过PendingIntent，可能这里出现了问题。*

*38. Android上HTTP协议通讯状态获取*

*通常情况下轻量级的Http传输Android平台可以直接使用Sun Java的HttpURLConnection类方法处理，比如果自己定义一次请求header可以通过setRequestProperty设置，而我们需要获取的Http Web Server状态可以通过HttpURLConnection.getResponseCode() 的方法获取。 当然Http协议返回值常见的有 200 为成功，400为请求错误，404为未找到，500为服务器内部错误，403无权查看，302为重定向等等。 对于Android平台提供更完善的Apache类有HttpClient 、HttpPost、HttpResponse、HttpGet和HttpEntity，其中对于数据报头header构造通过HttpEntity，而返回状态值可以通过HttpResponse获取。 有关Android客户端和Server通讯类相关的开发我们将会在以后文章中做大量实例介绍。*

*39. Android布局Java代码构造法*

*一般情况下对于Android程序布局我们往往使用XML文件来编写，这样可以提高开发效率，但是考虑到代码的安全性以及执行效率，可以通过Java代码执行创建，虽然Android编译过的xml是二进制的，但是加载xml解析器的效率对于资源占用还是比较大的，一般一个简单的TextView，比如 <TextView  
  
android:id="@+id/textControl "  
  
android:layout\_width="100px"  
  
android:layout\_height="wrap\_content" /> 可以等价于下面的Java代码: LinearLayout.LayoutParams textParams = new LinearLayout.LayoutParams(100, LayoutParams.WRAP\_CONTENT); //宽度为100px，高为自适应最小的高度 // setOrientation(VERTICAL); 设置布局为垂直 TextView textControl = new TextView(this);//如果从一个XXXLayout.，比如LinearLayout为View的基类时这里this应该换成为创建改类的Context  
  
textControl.setText("Android开发网欢迎您");  
  
addView( textControl, textParams ); 当然Java处理效率比XML快得多，但是对于一个复杂界面的编写，可能需要一些套嵌考虑，如果你思维灵活的话，使用Java代码来布局你的Android应用程序是一个更好的方法。*

*40. 测试Android软件性能主要方法*

*对于Android平台上软件的性能测试可以通过以下几种方法来分析效率瓶颈，目前Google在Android软件开发过程中已经引入了多种测试工具包，比如Unit测试工程，调试类，还有模拟器的Dev Tools都可以直接反应执行性能。 1. 在模拟器上的Dev Tools可以激活屏幕显示当前的FPS，CPU使用率，可以帮助我们测试一些3D图形界面的性能。 2. 一般涉及到网络应用的程序，在效率上和网速有很多关系，这里需要多次的调试才能实际了解。 3. 对于逻辑算法的效率执行，我们使用Android上最普遍的，计算执行时间来查看: long start = System.currentTimeMillis();  
  
//android开发网提示这里做实际的处理do something  
  
long duration = System.currentTimeMillis() - start; 最终duration保存着实际处理该方法需要的毫秒数。这里类似Win32上的GetTickCount，在Win 32和Symbian上都提供了高精度的性能计数器和低阶计时器，这里在Dalvik VM上的Java层这种方法对于一般的应用足以。 4. GC效率跟踪，如果你执行的应用比较简单，可以在DDMS中查看下Logcat的VM释放内存情况，大概模拟下那些地方可以缓存数据或改进算法的。 5. 线程的使用和同步，Android平台上给我们提供了丰富的多任务同步方法，但在深层上并没有过多的比如自旋锁等高级应用，不过对于Service和appWidget而言，他们实际的产品中都应该以多线程的方式处理，以释放CPU时间，对于线程和堆内存的查看这些都可以在DDMS中看到。 更多的调试和性能测试方法Android123将在以后的内容中出现。*

*41. Splash Screen开场屏在Android中的实现*

*很多网友可能发现近期Tencent推出的手机QQ Android版包含了一个开场屏Splash Screen载入效果，通常游戏或大型软件打开时可能需要一个释放解析资源的过程，需要一个前台的动画播放和后台的逻辑处理线程配合，当然对于简单的软件也可以加一个Splash Screen作为美化。在Android平台上如何实现呢? 首先创建一个Activirty，在SetContentView时直接通过ImageView创建一个全屏的图片，Android123提示大家还要考虑好分辨率和当前设备一致，onCreate添加代码如下: new Handler().postDelayed(new Runnable(){ // 为了减少代码使用匿名Handler创建一个延时的调用  
  
public void run() {   
  
Intent i = new Intent(SplashScreen.this, Main.class); //通过Intent打开最终真正的主界面Main这个Activity  
  
SplashScreen.this.startActivity(i); //启动Main界面  
  
SplashScreen.this.finish(); //关闭自己这个开场屏  
  
}   
  
}, 5000); //5秒，够用了吧*

*42. Android的Activity你知多少呢?*

*看到这个标题很多网友肯定回答，我知道Activity是Android上的窗口基类，了解Activity的生命周期比如onCreate onStop等，呵呵，按照这样说Android123还知道Activity的实现其实是从ApplicationContext，而ApplicationContext是从Context这个抽象类派生而来的，当然我们看到显示的是View或者ViewGroup，当然今天说的不是这些东西，而是很多网友来问的Android为什么不设计一个任务管理器，当然从Android 1.5开始ActivityManager类提供了restartPackage可以关闭一个程序，需要加上<uses-permission android:name="android.permission.RESTART\_PACKAGES"/>这个权限，不过我们注意到，长按Home键可以看到以前程序的运行，同时可以快速的切换回来。这就是Android独有的程序生命周期管理机制 Activity历史栈。 我们在一个普通的程序主窗口A中打开了一个窗口B，而窗口B打开了窗口C，但是按下Back键后结果出乎了预期，是的这就是Activity的history stack的原因，在数据结构中栈是FIFO的，阻止我们不愿意看的情况的发生则可以在打开新Activity时加上标记FLAG\_ACTIVITY\_NO\_HISTORY，代码如下: Intent i= new Intent(this, cwj.class);   
  
i.setFlags(Intent.FLAG\_ACTIVITY\_NO\_HISTORY); //Android开发网提示大家相关的还有Intent.FLAG\_ACTIVITY\_CLEAR\_TOP，都试试  
  
startActivity(i); 当然更多的程序Activity控制可以再androidmanifest.xml中定义。*

*43. JSONObject在Android上的应用*

*如果你过去开发过AJAX应用，相信对JSONObject不会陌生吧，作为基于JavaScript的数据交换格式，可以直接代替Xml，这里Android从1.0开始就完全支持JSONObject。在平时应用中直接引入import org.json.JSONObject;即可方便使用。当然同类的还有SOAP。 在常规使用方便JSONObject对象可以实现类似Bundle或Parcel可以封装数据，代替一个XML的ITEM，但最大的优势是可以执行一些简单的方法，比如说getString、has、put、getBoolean、getInt等数据类型的存取操作。Android123提示大家对于常规的项目开发，今天本文不考虑Server端的布局，在Android平台上处理这些比较简单，主要是一些http的请求处理。可以直接引入import org.apache.http.xxx来实现web server层的数据交换，如果你没有专业的Server开发技术，可以通过简单的Web配合JSON方式快速实现自己的交互式应用。*

*44. Android高性能文件类MemoryFile*

*很多网友抱怨Android处理底层I/O性能不是很理想，如果不想使用NDK则可以通过MemoryFile类实现高性能的文件读写操作。MemoryFile顾名思义就是内存文件的意思，如果你过去从事过Win32开发，那么它的原理就是MapViewOfFile()，当然开发过Linux的网友可能很快就联想到了mmap()，是的该类就是他们的托管代码层封装，位于android.os.MemoryFile这个位置，从Android 1.0开始就被支持。 MemoryFile适用于哪些地方呢? 对于I/O需要频繁操作的，主要是和外部存储相关的I/O操作，MemoryFile通过将 NAND或SD卡上的文件，分段映射到内存中进行修改处理，这样就用高速的RAM代替了ROM或SD卡，性能自然提高不少，对于Android手机而言同时还减少了电量消耗。Android123提示网友该类实现的功能不是很多，直接从Object上继承，通过JNI的方式直接在C底层执行。 主要的构造方法 MemoryFile(String name, int length) ，这里第二个参数为文件大小，需要说明的是Android的MemoryFile和传统的mmap还有一点点区别，毕竟是手机，它内部的内存管理方式ashmem会从内核中回收资源。毕竟目前部分低端机型的RAM也比较吃紧。 synchronized boolean allowPurging(boolean allowPurging) //允许ashmem清理内存，线程安全同步的方式。  
  
void close() //关闭，因为在Linux内部mmap占用一个句柄，不用时一定要释放了  
  
InputStream getInputStream() 返回读取的内容用Java层的InputStream保存  
  
OutputStream getOutputStream() 把一个OutputSream写入到MemoryFile中  
  
boolean isPurgingAllowed() //判断是否允许清理  
  
int length() //返回内存映射文件大小 下面就是我们熟悉的，读写细节，主要是对字符数组的操作，这里大家要计算好每个文件类型的占用，同时考虑到效率对于自己分配的大小考虑粒度对齐。  
  
int readBytes(byte[] buffer, int srcOffset, int destOffset, int count)  
  
void writeBytes(byte[] buffer, int srcOffset, int destOffset, int count) 具体的实际应用，Android开发网将在下次和大家讲到。*

*45. TextUtils类-Android字符串处理类*

*对于字符串处理Android为我们提供了一个简单实用的TextUtils类，如果处理比较简单的内容不用去思考正则表达式不妨试试这个在android.text.TextUtils的类，主要的功能如下: 是否为空字符 static boolean isEmpty(CharSequence str) 拆分字符串 public static String[] split (String text, String expression) ，Android开发网提示大家仔细看例子如下 String.split() returns [''] when the string to be split is empty. This returns []. This does not remove any empty strings from the result. For example split("a,", "," ) returns {"a", ""}. 拆分字符串使用正则 public static String[] split (String text, Pattern pattern) 确定大小写是否有效在当前位置的文本TextUtils.getCapsMode(CharSequence cs, int off, int reqModes) 使用HTML编码这个字符串 static String TextUtils.htmlEncode(String s)*

*46. InputSream输入流转String字符串，Android开发工具类*

*在Android平台上使用Java层处理I/O时主要使用流，这里Android开发网给大家一个方便的类，可以处理InputStream输入流转为String字符串，在效率上，我们使用了字符串拼接StringBuilder类减少内存碎片以及BefferedReader类实现一个缓存。 private String Stream2String(InputStream is) {  
  
BufferedReader reader = new BufferedReader(new InputStreamReader(is), 16\*1024); //强制缓存大小为16KB，一般Java类默认为8KB  
  
StringBuilder sb = new StringBuilder();  
  
String line = null;  
  
try {  
  
while ((line = reader.readLine()) != null) { //处理换行符  
  
sb.append(line + "\n");   
  
}  
  
} catch (IOException e) {  
  
e.printStackTrace();  
  
} finally {  
  
try {  
  
is.close();  
  
} catch (IOException e) {  
  
e.printStackTrace();  
  
}  
  
}  
  
return sb.toString();  
  
} }*

*47. layout资源包含，android开发必读*

*有时候我们在一个Android程序中可能会复用布局文件，这时可以在一个xml文件中复用过去的布局文件，但是和常规的使用不同的是，需要加上类似包含头文件一样的include关键字，比如下面我们需要包含layout文件夹下的view.xml布局文件，需要<include layout="@layout/view" /> 这样下，完整的如下，大家可以试一试。 <?xml version="1.0" encoding="utf-8"?>   
  
<LinearLayout xmlns:android="*[*http://schemas.android.com/apk/res/android*](http://schemas.android.com/apk/res/android)*"   
  
android:orientation="vertical"   
  
android:layout\_width="fill\_parent"   
  
android:layout\_height="fill\_parent"   
  
>   
  
<TextView   
  
android:layout\_width="wrap\_content"   
  
android:layout\_height="wrap\_content"   
  
android:text="@string/cwj"   
  
/>  
  
<include layout="@layout/view" />   
  
<include android:id="@+id/block" layout="@layout/item" /> <TextView   
  
android:layout\_width="wrap\_content"   
  
android:layout\_height="wrap\_content"   
  
android:text="@string/android123"   
  
/>   
  
</LinearLayout>*

*48.Android控件开发之ToggleButton原理*

*在Android平台上比较有特色的就是ToggleButton控件，虽然它的功能和CheckBox有些类似，但是他们的用处还是有一定的区别比如ToggleButton原本有图片装饰，通过ToggleButton可以很清楚的显示某些状态。它们均从Button为基类的CompoundButton中实现，其真假事件从Checkable来实现。 public abstract class****CompoundButton*** *extends* ***Button*** *implements Checkable {  
  
private boolean mChecked; //状态是否选中  
  
private int mButtonResource;  
  
private boolean mBroadcasting;  
  
private Drawable mButtonDrawable; //按钮的图标  
  
private OnCheckedChangeListener mOnCheckedChangeListener; //选中状态改变监听  
  
private OnCheckedChangeListener mOnCheckedChangeWidgetListener; private static final int[] CHECKED\_STATE\_SET = {  
  
R.attr.state\_checked  
  
}; public CompoundButton(Context context) {  
  
this(context, null);  
  
} public CompoundButton(Context context, AttributeSet attrs) {  
  
this(context, attrs, 0);  
  
} public CompoundButton(Context context, AttributeSet attrs, int defStyle) {  
  
super(context, attrs, defStyle); TypedArray a =  
  
context.obtainStyledAttributes(  
  
attrs, com.android.internal.R.styleable.CompoundButton, defStyle, 0); Drawable d = a.getDrawable(com.android.internal.R.styleable.CompoundButton\_button);   
  
if (d != null) {  
  
setButtonDrawable(d);  
  
} boolean checked = a  
  
.getBoolean(com.android.internal.R.styleable.CompoundButton\_checked, false);  
  
setChecked(checked); a.recycle(); //显式的GC  
  
} public void toggle() {  
  
setChecked(!mChecked);  
  
} @Override  
  
public boolean performClick() {  
  
toggle();  
  
return super.performClick();  
  
} public boolean isChecked() {  
  
return mChecked;  
  
} public void setChecked(boolean checked) {  
  
if (mChecked != checked) {  
  
mChecked = checked;  
  
refreshDrawableState(); //更新当前状态的按钮图标 if (mBroadcasting) {  
  
return;  
  
} mBroadcasting = true;  
  
if (mOnCheckedChangeListener != null) {  
  
mOnCheckedChangeListener.onCheckedChanged(this, mChecked);  
  
}  
  
if (mOnCheckedChangeWidgetListener != null) {  
  
mOnCheckedChangeWidgetListener.onCheckedChanged(this, mChecked);  
  
} mBroadcasting = false;   
  
}  
  
} public void setOnCheckedChangeListener(OnCheckedChangeListener listener) {  
  
mOnCheckedChangeListener = listener;  
  
} void setOnCheckedChangeWidgetListener(OnCheckedChangeListener listener) {  
  
mOnCheckedChangeWidgetListener = listener;  
  
} public static interface OnCheckedChangeListener {  
  
void onCheckedChanged(CompoundButton buttonView, boolean isChecked);  
  
} public void setButtonDrawable(int resid) {  
  
if (resid != 0 && resid == mButtonResource) {  
  
return;  
  
} mButtonResource = resid; Drawable d = null;  
  
if (mButtonResource != 0) {  
  
d = getResources().getDrawable(mButtonResource);  
  
}  
  
setButtonDrawable(d);  
  
} public void setButtonDrawable(Drawable d) {  
  
if (d != null) {  
  
if (mButtonDrawable != null) {  
  
mButtonDrawable.setCallback(null);  
  
unscheduleDrawable(mButtonDrawable);  
  
}  
  
d.setCallback(this);  
  
d.setState(getDrawableState());  
  
d.setVisible(getVisibility() == VISIBLE, false);  
  
mButtonDrawable = d;  
  
mButtonDrawable.setState(null);  
  
setMinHeight(mButtonDrawable.getIntrinsicHeight());  
  
} refreshDrawableState();  
  
} @Override  
  
public boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event) {  
  
boolean populated = super.dispatchPopulateAccessibilityEvent(event); if (!populated) {  
  
int resourceId = 0;  
  
if (mChecked) {  
  
resourceId = R.string.accessibility\_compound\_button\_selected;  
  
} else {  
  
resourceId = R.string.accessibility\_compound\_button\_unselected;  
  
}  
  
String state = getResources().getString(resourceId);  
  
event.getText().add(state);  
  
event.setChecked(mChecked);  
  
} return populated;  
  
} @Override  
  
protected void onDraw(Canvas canvas) {  
  
super.onDraw(canvas); final Drawable buttonDrawable = mButtonDrawable;  
  
if (buttonDrawable != null) {  
  
final int verticalGravity = getGravity() & Gravity.VERTICAL\_GRAVITY\_MASK;  
  
final int height = buttonDrawable.getIntrinsicHeight(); int y = 0; switch (verticalGravity) {  
  
case Gravity.BOTTOM:  
  
y = getHeight() - height;  
  
break;  
  
case Gravity.CENTER\_VERTICAL:  
  
y = (getHeight() - height) / 2;  
  
break;  
  
} buttonDrawable.setBounds(0, y, buttonDrawable.getIntrinsicWidth(), y + height);  
  
buttonDrawable.draw(canvas);  
  
}  
  
} @Override  
  
protected int[] onCreateDrawableState(int extraSpace) {  
  
final int[] drawableState = super.onCreateDrawableState(extraSpace + 1);  
  
if (isChecked()) {  
  
mergeDrawableStates(drawableState, CHECKED\_STATE\_SET);  
  
}  
  
return drawableState;  
  
} @Override  
  
protected void drawableStateChanged() { //android123提示状态改变时需要更换按钮的图标  
  
super.drawableStateChanged();  
  
if (mButtonDrawable != null) {  
  
int[] myDrawableState = getDrawableState();  
  
mButtonDrawable.setState(myDrawableState);  
  
invalidate();  
  
}  
  
} @Override  
  
protected boolean verifyDrawable(Drawable who) {  
  
return super.verifyDrawable(who) || who == mButtonDrawable;  
  
} static class SavedState extends BaseSavedState {  
  
boolean checked; SavedState(Parcelable superState) {  
  
super(superState);  
  
}  
  
private SavedState(Parcel in) {  
  
super(in);  
  
checked = (Boolean)in.readValue(null);  
  
} @Override  
  
public void writeToParcel(Parcel out, int flags) {  
  
super.writeToParcel(out, flags);  
  
out.writeValue(checked);  
  
} @Override  
  
public String toString() {  
  
return "CompoundButton.SavedState{"  
  
+ Integer.toHexString(System.identityHashCode(this))  
  
+ " checked=" + checked + "}";  
  
} public static final Parcelable.Creator<SavedState> CREATOR  
  
= new Parcelable.Creator<SavedState>() {  
  
public SavedState createFromParcel(Parcel in) {  
  
return new SavedState(in);  
  
} public SavedState[] newArray(int size) {  
  
return new SavedState[size];  
  
}  
  
};  
  
} @Override  
  
public Parcelable onSaveInstanceState() {  
  
// Force our ancestor class to save its state  
  
setFreezesText(true);  
  
Parcelable superState = super.onSaveInstanceState(); SavedState ss = new SavedState(superState); ss.checked = isChecked();  
  
return ss;  
  
} @Override  
  
public void onRestoreInstanceState(Parcelable state) {  
  
SavedState ss = (SavedState) state;  
  
super.onRestoreInstanceState(ss.getSuperState());  
  
setChecked(ss.checked);  
  
requestLayout();  
  
}  
  
} 从上面来看我们知道CompuundButton的实现相对繁琐了些，主要是考虑状态是否已经选中等情况的消息通知，Android开发网提醒大家而ToggleButton相对CompuundButton增加的给用户而言主要是开关的文字显示。 public class ToggleButton extends CompoundButton {  
  
private CharSequence mTextOn;  
  
private CharSequence mTextOff;  
  
private Drawable mIndicatorDrawable; private static final int NO\_ALPHA = 0xFF;  
  
private float mDisabledAlpha;  
  
public ToggleButton(Context context, AttributeSet attrs, int defStyle) {  
  
super(context, attrs, defStyle);  
  
TypedArray a =  
  
context.obtainStyledAttributes(  
  
attrs, com.android.internal.R.styleable.ToggleButton, defStyle, 0);  
  
mTextOn = a.getText(com.android.internal.R.styleable.ToggleButton\_textOn);  
  
mTextOff = a.getText(com.android.internal.R.styleable.ToggleButton\_textOff);  
  
mDisabledAlpha = a.getFloat(com.android.internal.R.styleable.ToggleButton\_disabledAlpha, 0.5f);  
  
syncTextState();  
  
a.recycle();  
  
} public ToggleButton(Context context, AttributeSet attrs) {  
  
this(context, attrs, com.android.internal.R.attr.buttonStyleToggle);  
  
} public ToggleButton(Context context) {  
  
this(context, null);  
  
} @Override  
  
public void setChecked(boolean checked) {  
  
super.setChecked(checked);  
  
syncTextState();  
  
} private void syncTextState() {  
  
boolean checked = isChecked();  
  
if (checked && mTextOn != null) {  
  
setText(mTextOn);  
  
} else if (!checked && mTextOff != null) {  
  
setText(mTextOff);  
  
}  
  
} public CharSequence getTextOn() {  
  
return mTextOn;  
  
} public void setTextOn(CharSequence textOn) {  
  
mTextOn = textOn;  
  
} public CharSequence getTextOff() {  
  
return mTextOff;  
  
} protected void onFinishInflate() {  
  
super.onFinishInflate();  
  
updateReferenceToIndicatorDrawable(getBackground());  
  
} @Override  
  
public void setBackgroundDrawable(Drawable d) {  
  
super.setBackgroundDrawable(d);  
  
updateReferenceToIndicatorDrawable(d);  
  
} private void updateReferenceToIndicatorDrawable(Drawable backgroundDrawable) {  
  
if (backgroundDrawable instanceof LayerDrawable) {  
  
LayerDrawable layerDrawable = (LayerDrawable) backgroundDrawable;  
  
mIndicatorDrawable =  
  
layerDrawable.findDrawableByLayerId(com.android.internal.R.id.toggle);  
  
}  
  
}  
  
@Override  
  
protected void drawableStateChanged() {  
  
super.drawableStateChanged();  
  
if (mIndicatorDrawable != null) {  
  
mIndicatorDrawable.setAlpha(isEnabled() ? NO\_ALPHA : (int) (NO\_ALPHA \* mDisabledAlpha));  
  
}  
  
}  
  
}*

*49. AsyncTask实例代码演示Android异步任务*

*上次我们讲到了Android提供了一个较线程更简单的处理多任务的方法AsyncTask异步任务类，相对于线程来说AsyncTask对于简单的任务处理更安全，其内部的实现方法使用了Android的Handler机制，对于常见的文件下载可以使用AsyncTask类来处理，在Browser浏览器中就是用了该类下载Web服务器URL的Favicon图标。 首先Android123以简单的下载例子演示该类的大致结构，如下 private class DownloadFilesTask extends****AsyncTask****<URL, Integer, Long> {  
  
protected Long* ***doInBackground****(URL... urls) {  
  
int count = urls.length;  
  
long totalSize = 0;  
  
for (int i = 0; i < count; i++) {  
  
totalSize += Downloader.downloadFile(urls);  
  
publishProgress((int) ((i / (float) count)100));  
  
}  
  
return totalSize;  
  
}  
  
protected void* ***onProgressUpdate****(Integer... progress) {  
  
setProgressPercent(progress[0]);  
  
}  
  
protected void* ***onPostExecute****(Long result) {  
  
showDialog("Downloaded " + result + " bytes");  
  
}  
  
} 最终我们执行 DownloadFilesTask().execute(url1, url2, url3); 即可。 在Android浏览器中下载Favicon图标的实现如下: class DownloadTouchIcon extends AsyncTask<String, Void, Bitmap> {  
  
private final ContentResolver mContentResolver;  
  
private final Cursor mCursor;  
  
private final String mOriginalUrl;  
  
private final String mUrl;  
  
private final String mUserAgent;  
  
/\* package \*/ BrowserActivity mActivity; public DownloadTouchIcon(BrowserActivity activity, ContentResolver cr,  
  
Cursor c, WebView view) { //构造方法  
  
mActivity = activity;  
  
mContentResolver = cr;  
  
mCursor = c;  
  
mOriginalUrl = view.getOriginalUrl();  
  
mUrl = view.getUrl();  
  
mUserAgent = view.getSettings().getUserAgentString();  
  
} public DownloadTouchIcon(ContentResolver cr, Cursor c, String url) { //实现本类的构造  
  
mActivity = null;  
  
mContentResolver = cr;  
  
mCursor = c;  
  
mOriginalUrl = null;  
  
mUrl = url;  
  
mUserAgent = null;  
  
} @Override  
  
public Bitmap* ***doInBackground****(String... values) { //返回Bitmap类型  
  
String url = values[0]; AndroidHttpClient client = AndroidHttpClient.newInstance(mUserAgent);  
  
HttpGet request = new HttpGet(url); HttpClientParams.setRedirecting(client.getParams(), true); //处理302等重定向问题 try {  
  
HttpResponse response = client.execute(request); if (response.getStatusLine().getStatusCode() == 200) { //如果OK  
  
HttpEntity entity = response.getEntity();  
  
if (entity != null) {  
  
InputStream content = entity.getContent(); //将图标保存到InputStream中，因为是二进制内容  
  
if (content != null) {  
  
Bitmap icon = BitmapFactory.decodeStream( //从流中取出Bitmap，这里使用了BitmapFactory类的静态方法decodeStream  
  
content, null, null);  
  
return icon;  
  
}  
  
}  
  
}  
  
} catch (IllegalArgumentException ex) {  
  
request.abort();  
  
} catch (IOException ex) {  
  
request.abort();  
  
} finally {  
  
client.close();  
  
}  
  
return null;  
  
} @Override  
  
protected void onCancelled() {  
  
if (mCursor != null) {  
  
mCursor.close();  
  
}  
  
} @Override  
  
public void onPostExecute(Bitmap icon) {  
  
if (mActivity != null) {  
  
mActivity.mTouchIconLoader = null;  
  
} if (icon == null || mCursor == null || isCancelled()) {  
  
return;  
  
} 最终图标要保存到浏览器的内部数据库中，系统程序均保存为SQLite格式，Browser也不例外，因为图片是二进制的所以使用字节数组存储数据库的BLOB类型 final ByteArrayOutputStream os = new ByteArrayOutputStream();  
  
icon.compress(Bitmap.CompressFormat.PNG, 100, os); //将Bitmap压缩成PNG编码，质量为100%存储  
  
ContentValues values = new ContentValues(); //构造SQLite的Content对象，这里也可以使用raw sql代替  
  
values.put(Browser.BookmarkColumns.TOUCH\_ICON,os.toByteArray()); //写入数据库的Browser.BookmarkColumns.TOUCH\_ICON字段 if (mCursor.moveToFirst()) {  
  
do {  
  
mContentResolver.update(ContentUris.withAppendedId(Browser.BOOKMARKS\_URI, mCursor.getInt(0)),values, null, null);  
  
} while (mCursor.moveToNext());  
  
}  
  
mCursor.close();  
  
}  
  
} 本次Android开发网通过两个AsyncTask类演示了多种类型的任务构造，这里大家注意返回类型，本节演示了Android平台上Content Provider、AsyncTask、Bitmap、HTTP以及Stream的相关操作，大家如何想很快提高开发水平其实只要理解Google如何去实现Android系统常规构架就可以轻松入门谷歌移动平台。*

*50. Android自定义View实例AnalogClock源码*

*针对Android底层View的直接构造很多网友没有实战经验，本次Android开发网结合目前平台开源代码一起通过AnalogClock类来理解View的直接继承。AnalogClock就是Home Screen上的那个带有两根指针的表盘类。它的实现我们直接从开源代码可以了解到: public class AnalogClock extends View {  
  
private Time mCalendar; private Drawable mHourHand; //时针  
  
private Drawable mMinuteHand; //分针  
  
private Drawable mDial; //表盘背景 private int mDialWidth; //表盘宽度  
  
private int mDialHeight; //表盘高度 private boolean mAttached; //附着状态 private final Handler mHandler = new Handler(); //定一个Handler类实现更新时间  
  
private float mMinutes;  
  
private float mHour;  
  
private boolean mChanged; //时间是否改变 public AnalogClock(Context context) {  
  
this(context, null);  
  
} public AnalogClock(Context context, AttributeSet attrs) {  
  
this(context, attrs, 0);  
  
} public AnalogClock(Context context, AttributeSet attrs,  
  
int defStyle) {  
  
super(context, attrs, defStyle);  
  
Resources r = mContext.getResources();  
  
TypedArray a =  
  
context.obtainStyledAttributes(  
  
attrs, com.android.internal.R.styleable.AnalogClock, defStyle, 0); mDial = a.getDrawable(com.android.internal.R.styleable.AnalogClock\_dial); //加载表盘资源  
  
if (mDial == null) {  
  
mDial = r.getDrawable(com.android.internal.R.drawable.clock\_dial);  
  
} mHourHand = a.getDrawable(com.android.internal.R.styleable.AnalogClock\_hand\_hour); //加载时针图片资源  
  
if (mHourHand == null) {  
  
mHourHand = r.getDrawable(com.android.internal.R.drawable.clock\_hand\_hour);  
  
} mMinuteHand = a.getDrawable(com.android.internal.R.styleable.AnalogClock\_hand\_minute); //加载分针图片  
  
if (mMinuteHand == null) {  
  
mMinuteHand = r.getDrawable(com.android.internal.R.drawable.clock\_hand\_minute);  
  
} mCalendar = new Time(); //获取当前系统时间 mDialWidth = mDial.getIntrinsicWidth(); //获取表盘图片的宽度  
  
mDialHeight = mDial.getIntrinsicHeight(); //高度，同上  
  
} @Override  
  
protected void* ***onAttachedToWindow****() {  
  
super.onAttachedToWindow(); if (!mAttached) {  
  
mAttached = true;  
  
IntentFilter filter = new IntentFilter(); //注册一个消息过滤器，获取时间改变、时区改变的action filter.addAction(Intent.ACTION\_TIME\_TICK);  
  
filter.addAction(Intent.ACTION\_TIME\_CHANGED);  
  
filter.addAction(Intent.ACTION\_TIMEZONE\_CHANGED); getContext().registerReceiver(mIntentReceiver, filter, null, mHandler);  
  
} mCalendar = new Time(); onTimeChanged();  
  
} @Override  
  
protected void* ***onDetachedFromWindow****() {  
  
super.onDetachedFromWindow();  
  
if (mAttached) {  
  
getContext().unregisterReceiver(mIntentReceiver); //反注册消息过滤器  
  
mAttached = false;  
  
}  
  
} @Override  
  
protected void* ***onMeasure****(int widthMeasureSpec, int heightMeasureSpec) { int widthMode = MeasureSpec.getMode(widthMeasureSpec);  
  
int widthSize = MeasureSpec.getSize(widthMeasureSpec);  
  
int heightMode = MeasureSpec.getMode(heightMeasureSpec);  
  
int heightSize = MeasureSpec.getSize(heightMeasureSpec); float hScale = 1.0f;  
  
float vScale = 1.0f; if (widthMode != MeasureSpec.UNSPECIFIED && widthSize < mDialWidth) {  
  
hScale = (float) widthSize / (float) mDialWidth;  
  
} if (heightMode != MeasureSpec.UNSPECIFIED && heightSize < mDialHeight) {  
  
vScale = (float )heightSize / (float) mDialHeight;  
  
} float scale = Math.min(hScale, vScale); setMeasuredDimension(resolveSize((int) (mDialWidth \* scale), widthMeasureSpec),  
  
resolveSize((int) (mDialHeight \* scale), heightMeasureSpec));  
  
} @Override  
  
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
super.onSizeChanged(w, h, oldw, oldh);  
  
mChanged = true;  
  
} 主要的绘图重写View的onDraw方法，我们可以看到通过canvas实例直接屏幕 @Override  
  
protected void* ***onDraw****(Canvas canvas) {  
  
super.onDraw(canvas); boolean changed = mChanged;  
  
if (changed) {  
  
mChanged = false;  
  
} int availableWidth = mRight - mLeft;  
  
int availableHeight = mBottom - mTop; int x = availableWidth / 2;  
  
int y = availableHeight / 2; final Drawable dial = mDial;  
  
int w = dial.getIntrinsicWidth();  
  
int h = dial.getIntrinsicHeight(); boolean scaled = false; if (availableWidth < w || availableHeight < h) {  
  
scaled = true;  
  
float scale = Math.min((float) availableWidth / (float) w,  
  
(float) availableHeight / (float) h);  
  
canvas.save();  
  
canvas.scale(scale, scale, x, y);  
  
} if (changed) {  
  
dial.setBounds(x - (w / 2), y - (h / 2), x + (w / 2), y + (h / 2));  
  
}  
  
dial.draw(canvas); canvas.save();  
  
canvas.rotate(mHour / 12.0f \* 360.0f, x, y); //计算时针旋转的角度，android123提示就是那个时针图片的旋转角度，直接反应的就是表盘上那个针的时间  
  
final Drawable hourHand = mHourHand;  
  
if (changed) {  
  
w = hourHand.getIntrinsicWidth();  
  
h = hourHand.getIntrinsicHeight();  
  
hourHand.setBounds(x - (w / 2), y - (h / 2), x + (w / 2), y + (h / 2));  
  
}  
  
hourHand.draw(canvas);  
  
canvas.restore(); canvas.save();  
  
canvas.rotate(mMinutes / 60.0f \* 360.0f, x, y); //同理，分针旋转的角度 final Drawable minuteHand = mMinuteHand;  
  
if (changed) {  
  
w = minuteHand.getIntrinsicWidth();  
  
h = minuteHand.getIntrinsicHeight();  
  
minuteHand.setBounds(x - (w / 2), y - (h / 2), x + (w / 2), y + (h / 2));  
  
}  
  
minuteHand.draw(canvas);  
  
canvas.restore(); if (scaled) {  
  
canvas.restore();  
  
}  
  
} private void onTimeChanged() { //获取时间改变，计算当前的时分秒  
  
mCalendar.setToNow(); int hour = mCalendar.hour;  
  
int minute = mCalendar.minute;  
  
int second = mCalendar.second; mMinutes = minute + second / 60.0f;  
  
mHour = hour + mMinutes / 60.0f;  
  
mChanged = true;  
  
} private final BroadcastReceiver mIntentReceiver = new BroadcastReceiver() { //监听获取时间改变action  
  
@Override  
  
public void onReceive(Context context, Intent intent) {  
  
if (intent.getAction().equals(Intent.ACTION\_TIMEZONE\_CHANGED)) {  
  
String tz = intent.getStringExtra("time-zone");  
  
mCalendar = new Time(TimeZone.getTimeZone(tz).getID());  
  
} onTimeChanged(); //获取新的时间  
  
invalidate(); //刷新屏幕，强制类调用onDraw方法实现分针时针的走动  
  
}  
  
}; 看了本例根据，Android开发很简单吧，感兴趣的网友可以为本程序加入一个秒针，不过Android123提醒网友的是可能对于电池，以及系统运行效率产生一定的影响，不过作为练习大家可以试一试。*

*51. ArrayList LinkedList Set HashMap介绍*

*在Android开发中我们经常需要对数据进行分类和操作，对于轻量级的数据存储我们可能不需要动用SQLite或效率以及类库不完善的XML，由于SharedPreferences不具备数据枚举方法，如果仅仅是一个String或Int数组可以通过一个标记分割设计外，我们还是主要来看看Android或者说Java提供的基础数据类型辅助类ArrayList LinkedList Set HashMap的介绍，如果你熟悉C++的STL或Boost库可以略过本文。 在Java中提供了Collection和Map接口。其中List和Set继承了Collection接口；同时用Vector、ArrayList、LinkedList三个类实现List接口，HashSet、TreeSet实现Set接口。直接有HashTable、HashMap、TreeMap实现Map接口。 Vector基于Array的List，性能也就不可能超越Array，并且Vector是“sychronized”的，这个也是Vector和ArrayList的唯一的区别。 ArrayList：同Vector一样是一个基于Array的，但是不同的是ArrayList不是同步的。所以在性能上要比Vector优越一些。Android123提示大家适用于顺序性的查找 LinkedList：不同于前面两种List，它不是基于Array的，作为链表数据结构方式，所以不受Array性能的限制。当对LinkedList做添加，删除动作的时候只要更改nextNode的相关信息就可以实现了所以它适合于进行频繁进行插入和删除操作。这就是LinkedList的优势，当然对于元素的位置获取等方面就逊色很多。 List： 1. 所有的List中只能容纳单个不同类型的对象组成的表，而不是Key－Value键值对。例如：[ tom,1,c ]； 2. 所有的List中可以有相同的元素，例如Vector中可以有 [ tom,koo,too,koo ]； 3. 所有的List中可以有null元素，例如[ tom,null,1 ]； 4. 基于Array的List（Vector，ArrayList）适合查询，而LinkedList（链表）适合添加，删除操作。 虽然Set同List都实现了Collection接口，但是他们的实现方式却大不一样。List基本上都是以Array为基础。但是Set则是在HashMap的基础上来实现的，这个就是Set和List的根本区别。 HashSet：HashSet的存储方式是把HashMap中的Key作为Set的对应存储项，HashMap的key是不能有重复的。HashSet能快速定位一个元素，但是放到HashSet中的对象需要实现hashCode()方法0。 TreeSet：将放入其中的元素按序存放，这就要求你放入其中的对象是可排序的。TreeSet不同于HashSet的根本是TreeSet是有序的。它是通过SortedMap来实现的。 Set总结： 1. Set实现的基础是Map（HashMap）； 2. Set中的元素是不能重复的，如果使用add(Object obj)方法添加已经存在的对象，则会覆盖前面的对象，不能包含两个元素e1、e2（e1.equals（e2））。 Map是一种把键对象和值对象进行关联的容器，Map有两种比较常用的实现： HashTable、HashMap和TreeMap。 HashMap也用到了哈希码的算法，以便快速查找一个键， TreeMap则是对键按序存放，因此它有一些扩展的方法，比如firstKey(),lastKey()等。 HashMap和Hashtable的区别。 HashMap允许空（null）键（key）或值（value），由于非线程安全，效率上可能高于Hashtable。 Hashtable不允许空（null）键（key）或值（value）。 有关更多实用的Android开发技巧我们将在后面的文章中着重介绍。*

*52. ConditionVariable Android线程同步*

*ConditionVariable类位于android.os.ConditionVariable，它可以帮助Android线程同步。在SDK上的介绍ConditionVariable不同于标准Java位于java.lang.Object wait() 和 notify() ，这个类可以等待自己，这就意味着 open(), close() 和 block() 可能会假死 ，如果使用ConditionVariable类的open()在调用 block() 之前， block() 将不会阻塞，相反将会返回立即。 该类一共有4个方法 boolean block(long timeout)  
  
阻止当前线程知道条件是open，或直到超时，这里参数long timeout为超时设置，Android123提示大家如果你们从事过Win32开发，这个方法类似DWORD WaitForSingleObject(HANDLE hHandle,DWORD dwMilliseconds); 函数。 void block()  
  
阻止当前线程知道条件 open ，是上面的无超时等待重载版本。 void close()  
  
重置条件为 close状态。 void open()  
  
Open条件，释放所有线程的阻塞. ConditionVariable在创建时还有一种构造方法是 public ConditionVariable (boolean state) ，如果为true，默认时为opened，如果为false则是closed. ，默认public ConditionVariable () 为close().*

*53.Android开发之Eclipse调试技巧*

*使用Google提供的ADT插件可以在Eclipse上很轻松的调试Android程序，我们切换到DDMS标签，选择“Devices”标签，我们可以看到会出现类似下面的Debug Process(调试进程)、Update Threads(更新线程)、Update Heap(更新堆)、Cause GC(引起垃圾回收)、Stop Process(停止进程)、Screen Capture(屏幕截图)、Reset adb(重启Android Debug Bridge) 这里我们就可以很好的观察Android程序运行时的各种状态，比如进程信息、线程分析、堆内存的占用，结束一个进程，当然这些操作都是在DDMS框架下进行的，日程开发的程序是无法执行调用的。如果遇到adb调试桥运行不稳定时可以选择reset adb来重新启动adb.exe进程,整个界面如图: 很多网友对于一些常的Android规程序性能测试、文件管理或屏幕截图均使用Eclipse中的DDMS插件来查看，其实通过SDK中提供的Dalvik Debug Monitor可以很好的调试Android程序，这里可以更直观的现实设备的各种信息，除了Logcat、VM Heap堆查看、Thread线程状态外，在菜单的Device中可以找到Screen capture来截图、File Explorer进行文件同步操作，使用Show process status可以显示设备当前的进程状态，以及 快速的过滤Logcat信息，可以分析无线状态radio state、程序状态app state等等。这里支持模拟器和真机的显示，该工具可以再android-sdk-windows-1.5\_r1\tools\ddms.bat找到，目前我们测试环境为Windows平台，下次讲述下CentOS中的操作，如图:* ***Android性能与调试很重要*** *用于手持的移动设备，Android软件性能上需要多加考虑。首先Java VM在资源占用上开销是很大的，很多垃圾GC处理机制直接影响到内存释放和整个平台运行的流畅度。 1.节省电量 手机软件必须考虑的问题是省电，如果需要大型处理尽量由服务器处理，直接把结果返回到手持设备上。多线程也是一种奢侈的使用，但是I/O存储必需这样才能保证流畅度，线程的阻塞将会降低用户体验，但是线程间切换调度的开销一直是重点。Android在DDMS中加入了Thread查看。 2.内存占用 在Eclipse+ADT插件的开发方式中，我们在DDMS中可以看到Heap堆内存的显示,Android开发网提示的是Java内存分配方式的问题，尽量产生少的对象，比如字符串操作如果连加比较多，可以使用StringBuilder代替String类。在游戏开发中经常用到的图片可以通过切片的方式从一个大的png图片上截取，或者放在gif文件作为逐帧保存，这样可以共用文件头减小体积。 3.调试工具 Android调试工具主要是模拟器中的Dev Tools和DDMS中的Logcat查看。当然模拟器自带的Dev Tools在功能上还是很详细的，可以显示CPU占用率，内存使用量，在单元测试时需要多加分析。****Android开发工具Dev Tools介绍****ndroid提供了很多开发调试工具除了ADB、TraceView、Logcat外，今天这个名为****Dev Tools****的Android开发调试工具隐藏在*[*Android模拟器*](http://www.android123.com.cn/moniqi/index.html)*中，为我们提供了强大的调试支持。我们在功能表中找到Dev Tools，运行后可以看到有很多条目，比如Development Settings，用来开发设置，进入后我们看到了比如Show CPU Usage这样的实用功能显示CPU占用率，帮助Android开发人员分析当前软件性能情况，今天就分析下Development Settings中的选项: Wait for debugger 等待调试器  
  
Enable ADB 启用ADB(android调试桥)  
  
Show running processs (显示运行中的进程)  
  
Show screen updates (显示屏幕更新) 下面是一些常规的调试选项，Android开发网友情提示开启这些选项后可能会影响运行效率，这些探测选项也是CPU敏感的。 Immediately destroy activites (立即销毁activities)  
  
Show CPU usage (显示CPU占用率)  
  
Show background (显示北京)  
  
Show Sleep state on LED (在休眠状态下LED开启)  
  
Keep screen on while plugged in (保持屏幕开启当插入后)  
  
Show GTalk service connection status (显示GTalk服务连接状态)*