【eoeAndroid 特刊】

第四期：Android widget (第二版)

发布版本: Ver 2.0.0(build 2009.06.30)

第二版前言：

首先感谢大家对 eoe 特刊的支持。我们以后也将竭诚为大家奉献出更多更好的文章。
eoe 特刊第四期发布了以后，得到了大家的广泛支持，下载量以及好评都非常多。在这里，我代表第四期特刊的翻译人员，感谢大家对我们的支持。

同时在特刊发布后，地狱怒兽和我便共同做了一款类似电子宠物的 Widget。因为我们一开始对这个 Widget 要求很高，所以在做的时候发现了很多新的问题，也有很多新的收获，一些问题让我们对 Android 上的 Widget 有了全新的认识。对比以前，我们两个人都有了更多、更深的了解。我们觉得这些经验必须奉献给大家，这样可以让避免大家在 Widget 的开发中走很多弯路。

同时我和兽兽以前的文章在也做了小幅调整和更新。

另外开篇校正一下部分朋友的发音，Widget 这个的正确读音应该是.[‘widʒit] ，而不少同学读成了[waɪdget]。
好了，这次提醒了，专业从读音开始。

本篇简介

在 Android 1.5 SDK 中，我们看到了一系列功能和 API 上的变化变化，包括软键盘、桌面 Widget 和 Live Folder API、视频录制 API，蓝牙功能升级等，Google 近来对于这些全新功能的解析使得 Android 开发变得异常活跃也异常强大。
而这其中最令人兴奋的新特性是 AppWidget framework,这个框架允许开发者开发 widgets，这些 widgets 可以被用户拖到用户的桌面并且可以交互。widgets 可以提供一个 full-featured apps 的预览 ,例如可以显示即将到来的日历事件 ,或者一首后台播放的歌曲的详细信息，其的其内容包含收录了如下文章：

1. 让我们再次从 Widget 的 Hello World 开始
--本文以一个简单、实际的例子给你一个最直观的例子。

2. 简介 widgets 以及 AppWidget framework
--本篇介绍 widget 及其相关的 AppWidget framework，让你知道 widget 到底是怎么一回事。

3. Widget 设计向导（Widget Design Guidelines）
--本篇告诉你如何设计友好、漂亮的 widget 应用，及其摆放的位置等等。

4. AppWidget 调用之系统时钟篇
--本篇以一个调用系统时钟的例子告诉你如何定时更新 widget，如何调用系统资源。

5. 让我们的 Widget 和 service 打交道
--本篇以一个实际的例子来告诉你如何使得 widget 和我们定义的 service 打交道。

6. 维基词典 每日一词，教你怎样和 web service 打交道
--本篇介绍如何和 web service 打交道，如何从互联网上取回需要的资源，并用到 widget 中。

new !7、8、9 APP Widget 编程基础实战 三部曲 (暑期巨献)

--地狱怒兽出品 让你理解所有的 Widget 开发

new !11. Widget 研究报告(暑期巨献)

--从另一个角度为你剖析 Android 的 Widget

12. Android Widgets, 潜力无穷
--本篇站在一个老外的角度告诉你哪些类型的 widget，他们是愿意埋单的。

new! 13.Widget 开发建议

地狱怒兽&游利卡 联合撰写

new! 15.eoePet 开发纪实

--记录 eoePet 开发道路上的点点滴滴

16.编后语

--本篇记录一些编后语

希望本期内容能让更多的开发者熟悉 Android Widget 及其相关内容 ,了解其周边诸如设计 ,交互 ,后台服务调用 ,和网络上的 web service 打交道等等。

本期特刊的撰写人员：

本期专题得到如下同学的大力支持和积极响应，是他们辛苦劳动，为 Android 发展和普及做出的贡献。

- apcwowo(apcwowo@hotmail.com)
- xinbohan(bhsky@163.com)
- 地狱怒兽(zyf19870302@126.com)
- thanjing
- 游利卡(http://www.cnblogs.com/pcedb0189/)

活动发起地址：

【eoeAndroid 特刊】策划 第四期：Android widget

http://www.eoeandroid.com/viewthread.php?tid=663

第四期：Android widget (第二版) 1

第二版前言： 1

本篇简介 2

1. **让我们再次从 Widget 的 Hello World 开始** 8

1.1 添加 Widget 8

1.2 hello word 实践 9

2. introducing home screen widgets and the AppWidget framework 14

2.1 有关 Widget 14

2.2 维基词典 每日一词 15

2.3 Widget 开发的一些注意事项 15

3. Widget 设计向导（Widget Design Guidelines） 17

3.1 概述 17

3.2 Widget 标准剖析（Standard widget anatomy） 18

3.3 设计一个 widget（Designing a widget） 20

3.4 标准 widget 阴影（Standard widget shadows） 25

4. AppWidget 调用之系统时钟篇 29

4.1 实现需求 30

4.2 分解步骤 30

4.3 本章总结 36

5. 让我们的 Widget 和 service 打交道 37

5.1 实例演示 37

5.2 桌面添加 Widget 后的效果 57

5.3 总结 58

6. 维基词典 每日一词，教你怎样和 web service 打交道 61

6.1 简介 61

6.2 让我们看一下后台的代码 65

6.3 我们下来说一下 SimpleWikihelper.java 66

6.4 最后来看一下我们的 widget.java 74

6.5 总结 79

##App Widget 编程基础实战(三部曲） 80

7． 先看 AppWidget 80

7.1App Wiget 生命周期——App Widget 编程基础实战(一) 80

7.2App Wiget 生命周期方法解析 81

8.如何实现 Widget 的更新——App Widget 编程基础实战(二) 83

8.1 如何实现 TextView 的更新 83

8.2 如何实现 ImageView 的更新 85

8.3 如何实现 ProgressBar 的更新 87

如何实现 AnalogClock 的更新 89

8.5 如何实现 Chronometer 的更新 90

9.如何实现 Widget 的按钮事件——App Widget 编程基础实战(三） 90

9.1 如何开启 Activity 91

9.2 如何开启 Service 92

9.3 如何发送按钮 Action 92

9.4 如何在 Service 中接收 Widget Button Action 95

9.5Service 如何发送 Broadcast 96

10.Log 调试支招——App Widget 编程基础实战(额外) 97

11.Android Widget 调查报告 99

11.1 开始 99

11.2 探究 Appwidget 还有 Widget 之间的关系 99

11.3 Appwidget 贫弱的真正罪魁祸首 102

11.4 不要对 UI 说无所谓 104

11.6 从 Widget 逃出来的信息 106

12. Android Widgets，潜力无穷 108

13.Android Widget 开发 建议 111

14.eoePet 开发纪实 114

15. 编后语 117

16. 其他 118

BUG 提交 118

资源下载： 118

参加翻译 118

关于 eoeAndroid 118

17.广告 119

正文开始》》

1. 让我们再次从 Widget 的 Hello World 开始

作者：游利卡 http://www.cnblogs.com/pcedb0189/

在做这期的特刊之前，我犯了一个很傻很天真的错误 ——居然不知道从哪里添加 Widget，调试的时候一直认为是代码没有写好，折腾了好长时间。后来发现，不仅是我，很多没有真机的朋友都犯了同样的错误。所以首先开篇扫盲，其实添加 Widget 很简单。

1.1 添加 Widget

点击 Menu——添加（add）——Widget

就这么简单,下次不要这么小白的问题了哦

1.2 hello word 实践

好了我继续废话一会，Android 相关的文档目前真的做的是够差劲的，开始这期特刊之前，google 下 Android Widget 的中文网页 ,你基本上只能得到 http://www.android123.com.cn/androidkaifa/292.html 这个链接，或者是同样一篇文章的转载。

其实这篇文章是一个项目的片段，如果只看这个，能看懂才怪呢，对 Android123 的不负责任严重抗议。在后面我将专门写了一篇对这个项目的介绍。

而很多原文文档，也有点复杂，起码我看都用了一些时间，其实 Widget 这个是个很简单的东西。就是一个小小的插件吗，它与 app 的不同之处就是，Widget 需要更加讲究 UI 的设计，即便是做一个简单的 Hello World 都需要。

所以我就自己重新写了一个 hello Widget！很简单，就是显示文字。不为别的，就是让大家看一下一个 Widget 到底有什么，那些复杂的代码都是在这样的基础上不断扩展功能，复杂一点的开发就和 app 一样了。

因为后面海阳哥还有 Ice 都还有更多的文章介绍，我就直接贴代码了，是程序员都能看懂的。

文件结构

src/hellowidget.java 代码

res/drawable/icon.png （图标）

layout/main.xml 布局文件

values/strings.xml 文字信息

xml/appwidget_info.xml 插件信息

AndroidManifest.xml Widget 的配置文件

hellowidget.java

```
package com.electron.biginwidet;

import android.appwidget.AppWidgetManager;

import android.appwidget.AppWidgetProvider;

import android.content.Context;

import android.widget.RemoteViews;

public class Hellowidget extends AppWidgetProvider {

        public Context context;

            @Override

        public void onUpdate(Context context, AppWidgetManager appWidgetManager,int[]
appWidgetIds){

                    updateAppWidget(context, appWidgetManager);}

        static void updateAppWidget(Context context, AppWidgetManager appWidgetManager) {

            CharSequence text = context.getString(R.id.widget_text);

            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.main);

            views.setTextViewText(R.id.widget_text, text);    }}
```

main.xml

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

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"    >
<TextView
    android:id="@+id/widget_text"
    android:layout_width="144dip"
    android:layout_height="72dip"
    android:layout_marginTop="12dip"
    android:padding="10dip"
    android:text="@string/widget_text"    />
</LinearLayout>
```

string.xml

```
<resources>
    <string name="app_name">BiginWidget</string>
    <string name="widget_text">hello biginwidget</string>
</resources>
```

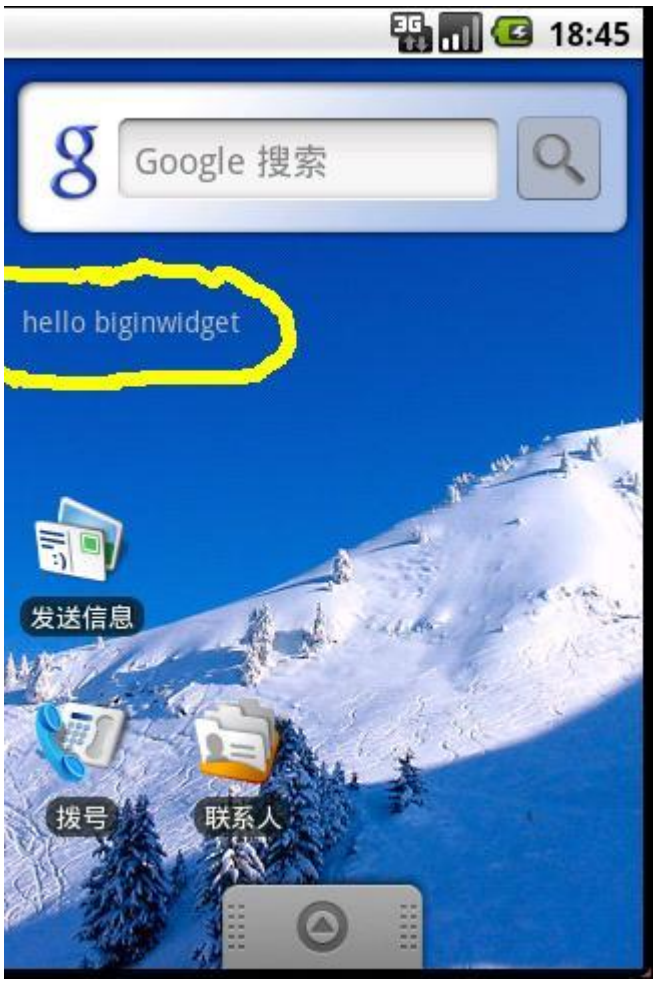
appwidget_info.xml

```
<appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="144dip"
    android:minHeight="72dp"
    android:updatePeriodMillis="86400000"
    android:initialLayout="@layout/main"    >
</appwidget-provider>
```

需要在 AndroidManifest.xml 下的 <application>添加的信息如下

```
<receiver android:name=".Hellowidget"
    android:label="@string/widget_text"
    android:icon="@drawable/icon" >
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>
         <meta-data android:name="android.appwidget.provider"    android:resource="@xml/appwidget_info" />
 </receiver>
```

很简单吧！最后的效果也很简答，就是下图，页面上一句话，其实这个蛮不错的，你可以随意添加任何语句，而且想放哪就放哪，随你托，比如……（以下省略十万字）



2. introducing home screen widgets and the AppWidget framework

原文：http://android-developers.blogspot.com/2009/04/introducing-home-screen-widgets-and.html

作者：Posted by Jeff Sharkey on 20 April 2009 at 6:43 PM

翻译：apcwowo

2.1 有关 Widget

一个比较激动的新功能将会在 Android1.5SDK 中提供，就是 AppWidgetk 框架，这个框架允许开发者在编写 widgets 时提供一个不在主应用程序桌面而与其交互的功能（这个框架允许开发者编写自己的 widgets，而且用户能够将这个 widgets 拖动到桌面并与之交互）。如显示即将来临的日历事件或者显示后台播放音乐的信息。（widgets 能够快速访问应用，比如显示将要来临的日历事件或者显示正在后台播放的歌曲信息。）

当 widgets 退出其界面，我们将给一个保留的空间去显示应用程序的自定内容，用户也可以通过它与你的应用程序交互。例如暂停或选择音乐。如果是后台服务，你可以根据自己的计划去更新你的 widget,AppWidget 也提供了一个自动更新的机制。

（当 widgets 被拖到桌面上，它们就拥有了一个保留的空间去显示应用程序的自定内容。用户也能够与你的应用程序交互，例如暂停或者切换歌曲。如果是后台服务，你可以根据自己的计划去更新你的 widget，也可以让 AppWidget 自动完成。）

At a high level, each widget is a BroadcastReceiver paired with XML metadata describing the widget details. The AppWidget framework communicates with your widget through broadcast intents, such as when it requests an update. Widget updates are built and sent using RemoteViews which package up a layout and content to be shown on the home screen.

在更高的级别，每一个 widget 都包含一对 BroadcastReceiver，并用 XML 元数据描述 widget 的详细信息。AppWidget 框架通过广播意图与你的 widget 进行通讯，比如请求一个更新。Widget 更新的建立和发送都是由 RemoteViews（打包的一个布局）完成，内容被显示在桌面上。

2.2 维基词典 每日一词

因为刚开始分配任务一点失误，让本片文章中出现了两篇以 维基词典 每日一次 为示例代码的文章，为了节约篇幅，这里保留一些原作者的精华部分，具体的代码实现请参看我们后面专门的一片文章

2.3 Widget 开发的一些注意事项

我们写一下 BroadcastReceiver 代码处理 AppWidget 请求，去帮助 widget 管理所有类型的广播消息。我们需要继承的类名为 AppWidgetProvider，一个需要注意的是我们需要一个后台程序去执行 widget 的更新，这是因为 BroadcastReceivers 取决于 ANR 时间，它可能关闭我们的程序当它运行时间比较长的时候，一个 WEB 请求可以需要几分钟，所以我们使用后台服务去避免任何的 ANR 超时。

最后是一些建议。Widgets 用于内容长时间不需要更新的应用，更新的频率超过了小时级就会消耗掉更多的电池和带宽。更新的频率要尽可能的低，或者让用户自己去设定更新频率，例如，某个用户可能会设定每 15 分钟更新一次，或者一天只更新四次。我将会在 giving at Google I/O 这一话题中讨论如何通过一些额外的措施延长电池的寿命。

最后一件很酷的事情提的是 AppWidget 框架在两个方向都是抽象的。意思是你可以任选其一，是桌面包含 widgets ,,你的 widget 也可以插入一个桌面。这些都是 AppWidget 框架所支持的

我已经写好了一些 widget，例如日历，音乐，但我们更高兴看到你写的。

3. Widget 设计向导（Widget Design Guidelines）

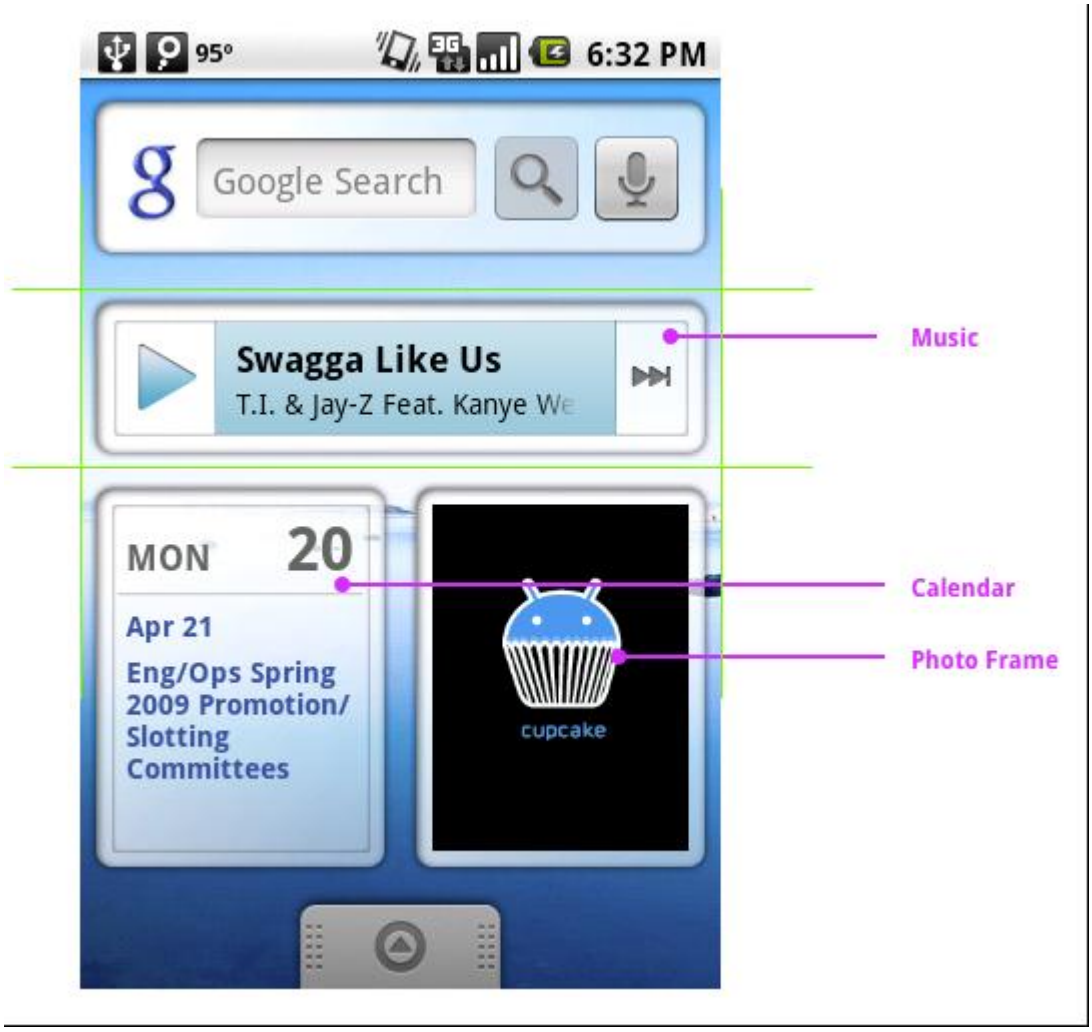
原文：http://sdk.eoeandroid.com/guide/practices/ui_guidelines/widget_design.html

翻译：xinbohan(bhksy@163.com)

3.1 概述

Widget 作为一种特色被引进到 android1.5 中。使用者手机主屏幕上，widget 扮演一种在实际应用让重要的或者及时的信息，能让使用者瞥一下就记住的角色。Android 系统标准 image 尺寸包括一些 widget 例子，包括像 Analog Clock, Music 的 widget 等，还有另外一些应用的 widget。

使用者挑选 widget 展示在主屏幕上。通过触摸来得到一个主屏幕中的空白区域，在 menu 中选择 Widget，然后选择他们想要的 widget。



这个文档描述怎么样去设计一个 widget 让它生动地和其他 widget 适应，还有一些 android 主屏幕显示的原理。它也描述一些 widget 艺术设计标准，一些 widget 制图小技巧和一些 android 的小技巧。

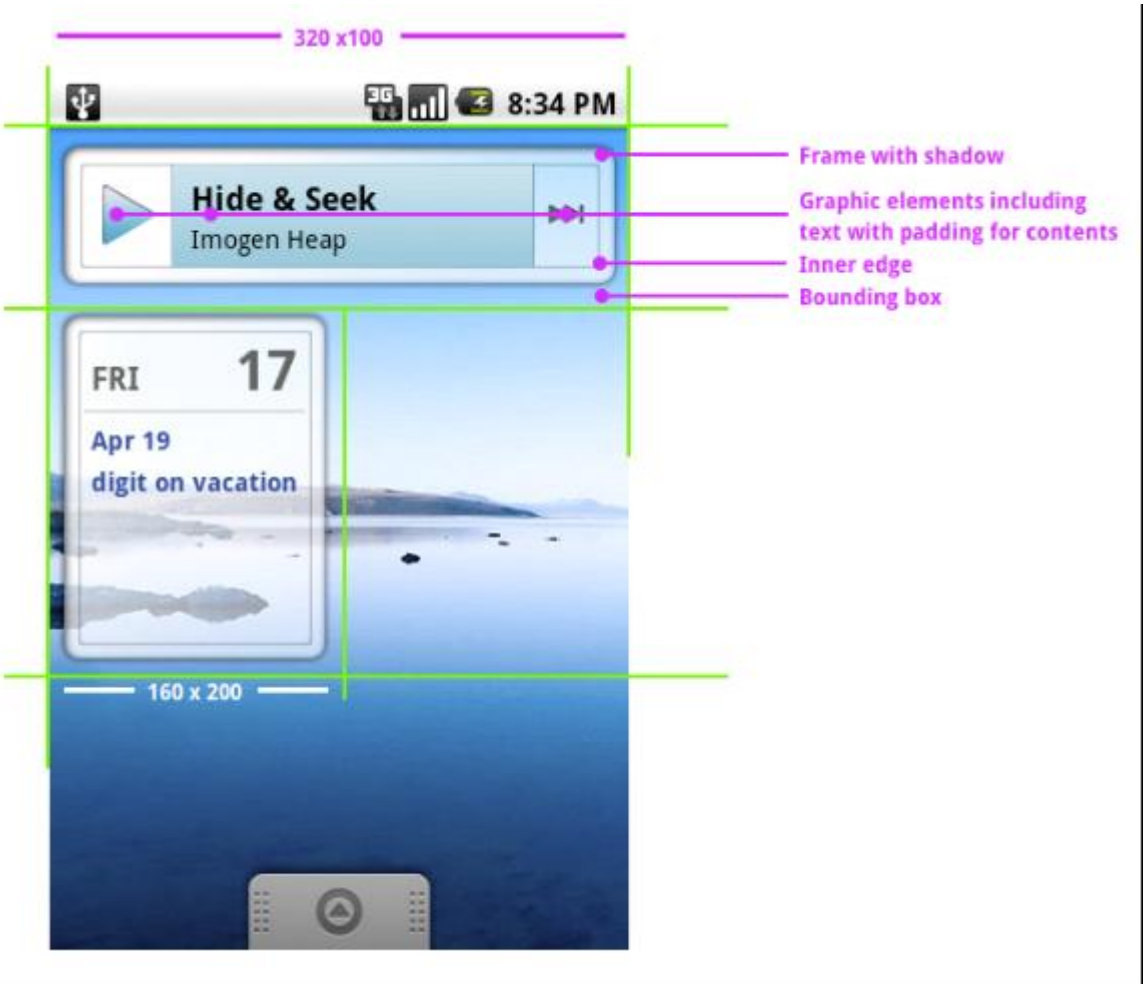
想了解更多关于开发 widget，请点击 AppWidgets section of the *Developer'sGuide* (开发者向导中关于 AppWidgets 选项)，还有一些 AppWidgets 的博文。

3.2 Widget 标准剖析（Standard widget anatomy）

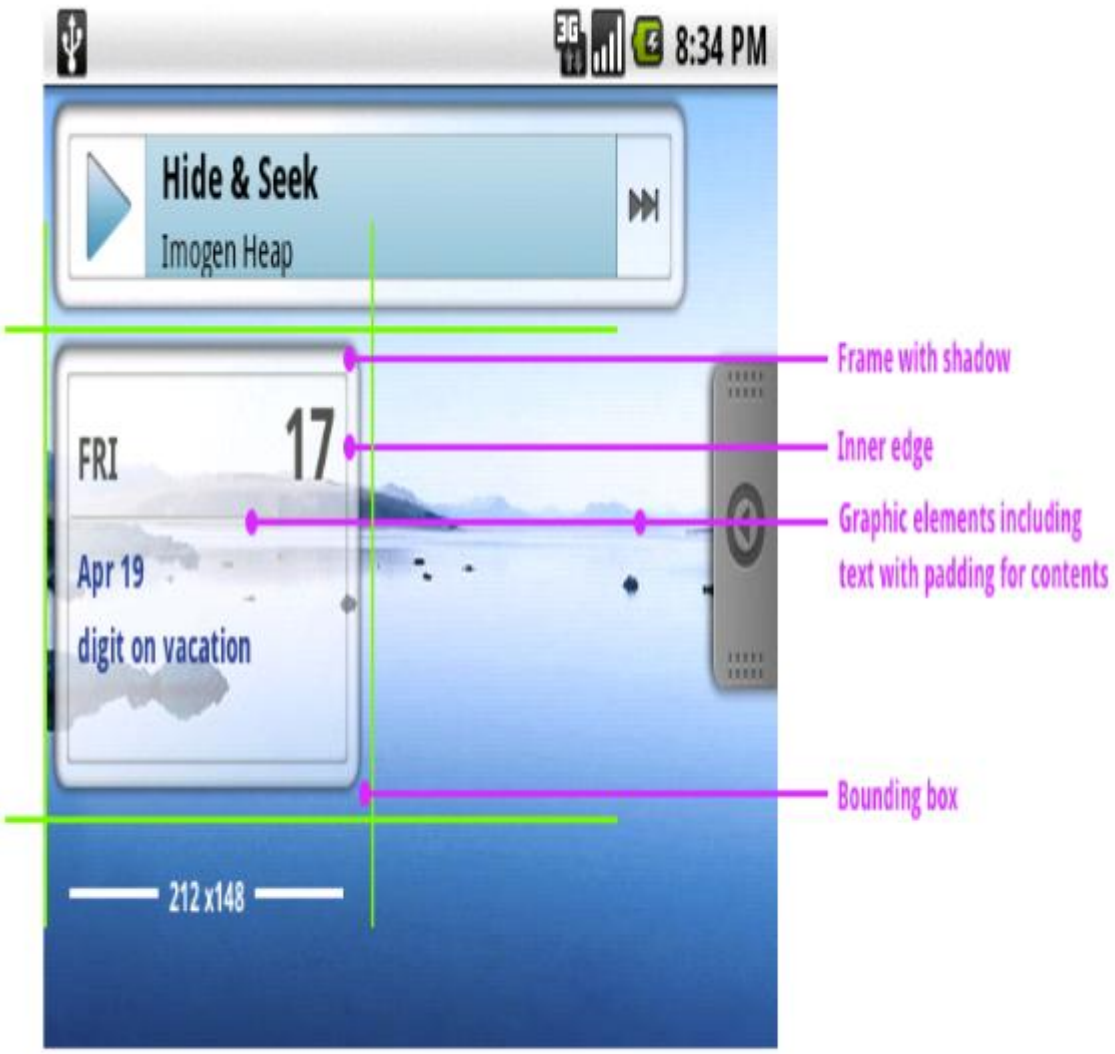
典型的 android widget 由三个部分组成 :一个 bounding box（包围盒）,一个 frame(边框),还有 widget 的部分图形控件和其它部分。精心设计的 widget ,填充在 bounding Box 和 frame 的边缘 ,还有些填充在 frame

边缘和 widget 控制的内部。widget 的设计，在主屏幕视觉上要适应其他 widget，与其它元素进行调整；也使用一些标准阴影效果。这些的详细资料在文档中都有。

纵向标准 widget 尺寸



横向标准 widget 尺寸



3.3 设计一个 widget (Designing a widget)

1，为你的 widget 选择一个 bounding box。

最有效的 widget 能在最小的 widget 尺寸基础上，显示您的应用程序最有用或者最及时的数据。

用户将权衡是否有用并用您的 widget 对部分主页屏幕进行覆盖，因此越小越好。

所有的 widget 必须适应六种支持的 widget 尺寸中的任一个的 bounding box。更重要的是，要适应人像和横向尺寸，从而在用户切换主屏幕的时候，使你的 widget 看起来觉得不错。

Standard widget sizes 说明包括六个 widget 尺寸（三个在人像方面，三个在风景方面）。

2，选择一个匹配的 frame。

Standard widget frames 包括六个 widget 尺寸中的标准 frame。点击你可以进行复制或者下载给自己用。你也可以不用这些 frame，但是如果你使用了，你的 frame 看起来会跟其他 widget 显得和谐点。

3,在您的图形上使用标准阴影效果。

再次重申，你不用强迫使用这些效果，但是 Standard widget shadows 显示用于 widget 尺寸中 photoshop 设置。

4，如果您的 widget 包括（Button）按钮，请按照三个状态去描述（默认情况下，按下，并选定）。

你可以

download a Photoshop file that contains the three states of the Play button,

摘下音乐 widget，去分析用于这三种标准按钮的 photoshop 设置。

5，完成你的绘画作品，然后规划和调整它。

Widget alignment tips and tricks

这里介绍一些技术来布置你的 widget 在标准 frame 里面。还有一些 widget 图形小技巧。

6，正确保存你的 widget 图形文件。

Windows graphics file format 描述了正确设置你的 widget 图形文件。

标准 widget 尺寸（Standard widget sizes）

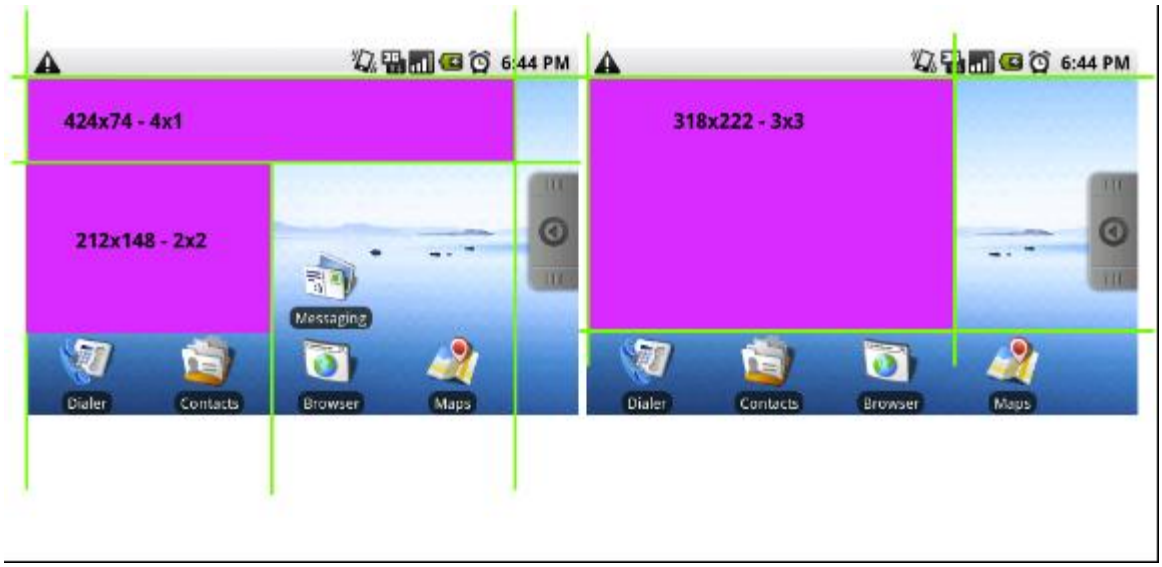
有六种标准 widget 尺寸，基于网格的主屏幕 4 × 4 （人像）或 4 × 4 （景观）单元格。这些外形尺寸是六个标准 widget 尺寸的 bounding boxes。典型的 widget 内容不画出这些尺寸的封装条，但适合一个在 bounding box 里面的 frame。就像 Designing a widget.描述的一样。

在纵向，每个单元格是 80 像素（pixels）宽 100 像素（pixels）高。（如图所示的纵向单元格），三个 widget 支持尺寸是：

Cells	Pixels
4 x 1	320 x 100
3 x 3	240 x 300
2 x 2	160 x 200

在横向，每个单元格是 106 像素（pixels）宽 74 像素（pixels）高。这三个 widget 支持横向尺寸是：

Cells	Pixels
4 x 1	424 x 74
3 x 3	318 x 222
2 x 2	212 x 148



标准工具框（Standard widget frames）

六个 widget 尺寸中每一个都有一个标准框架（standard frame），在本节中你可以点击这些框架（frame）去下载一个 photoshop 文件，你也可以使用你自己的 widget。

4x1_Widget_Frame_Portrait.psd

3x3_Widget_Frame_Portrait.psd

2x2_Widget_Frame_Portrait.psd

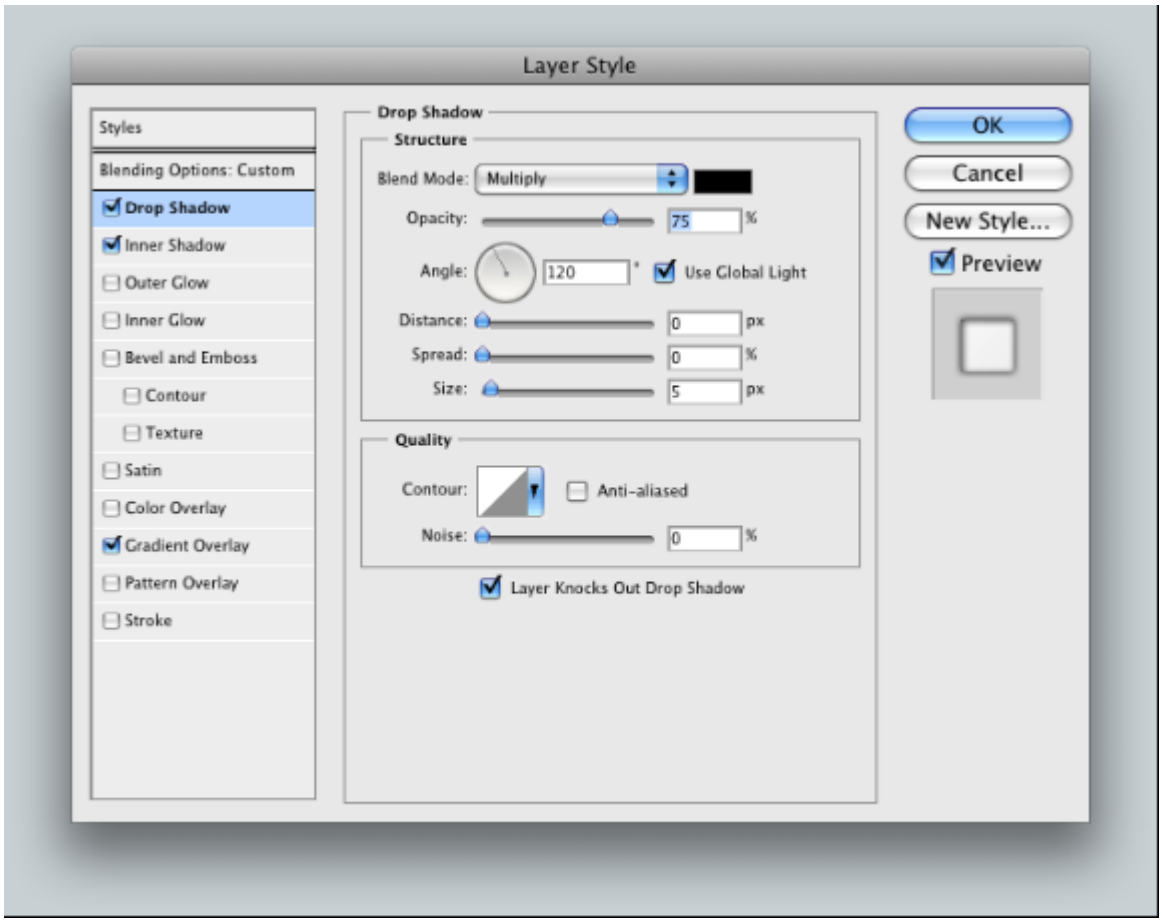
4x1_Widget_Frame_Landscape.psd

3x3_Widget_Frame_Landscape.psd

2x2_Widget_Frame_Landscape.psd

3.4 标准 widget 阴影（Standard widget shadows）

你可以套用阴影效果到你的 widget 作品中，这样它会适合其他标准的 widget，使用下列 Photoshop 的图层样式对话框设置：

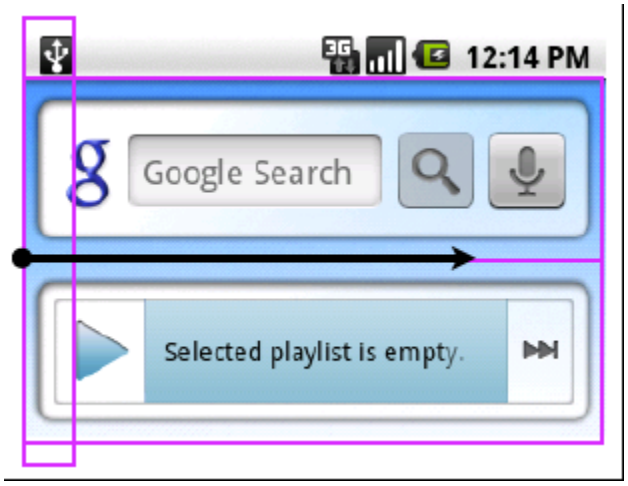


widget 的技巧与窍门（Widget graphics tips and tricks）

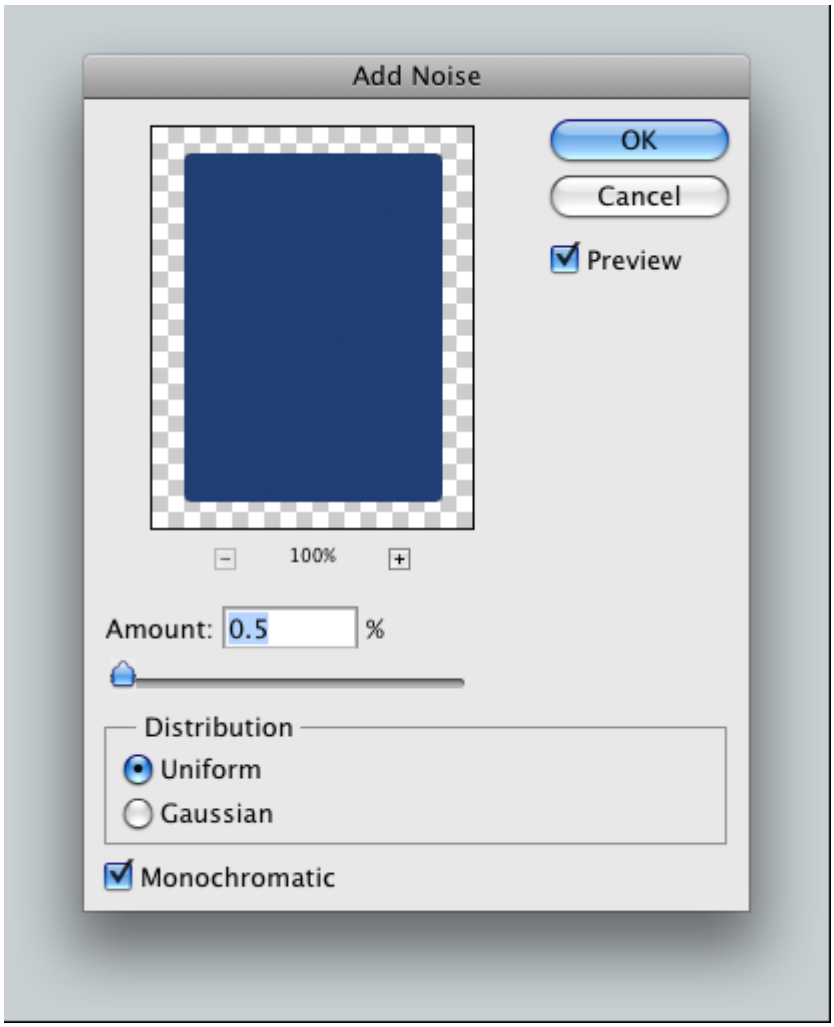
Android 小组已制定了一些调整布置 widget 作品的技巧，这样，在主屏幕上 widget 看起来和其他 widget 以及其他元素就比较协调，跟其它创造 widget 技术一样。

使用从 android SDK 模拟器截图，包括你的 widget 形状与阴影，控制与搜索工具，并与其它要素的主屏幕。

满单元格裁剪出一个 widget 作品，包括你想要的任何填充的东西。（也就是说，一个 4 × 1widget，降低到 320*100 像素）。



为了减少输出 widget 时的分类，可运用以下 photoshop 噪声设置为你的图片增加噪声。



适用于 9 -修补技术（9-patch techniques)来缩小图片，并设置填充的内容领域.

(See the detailed guide here.)

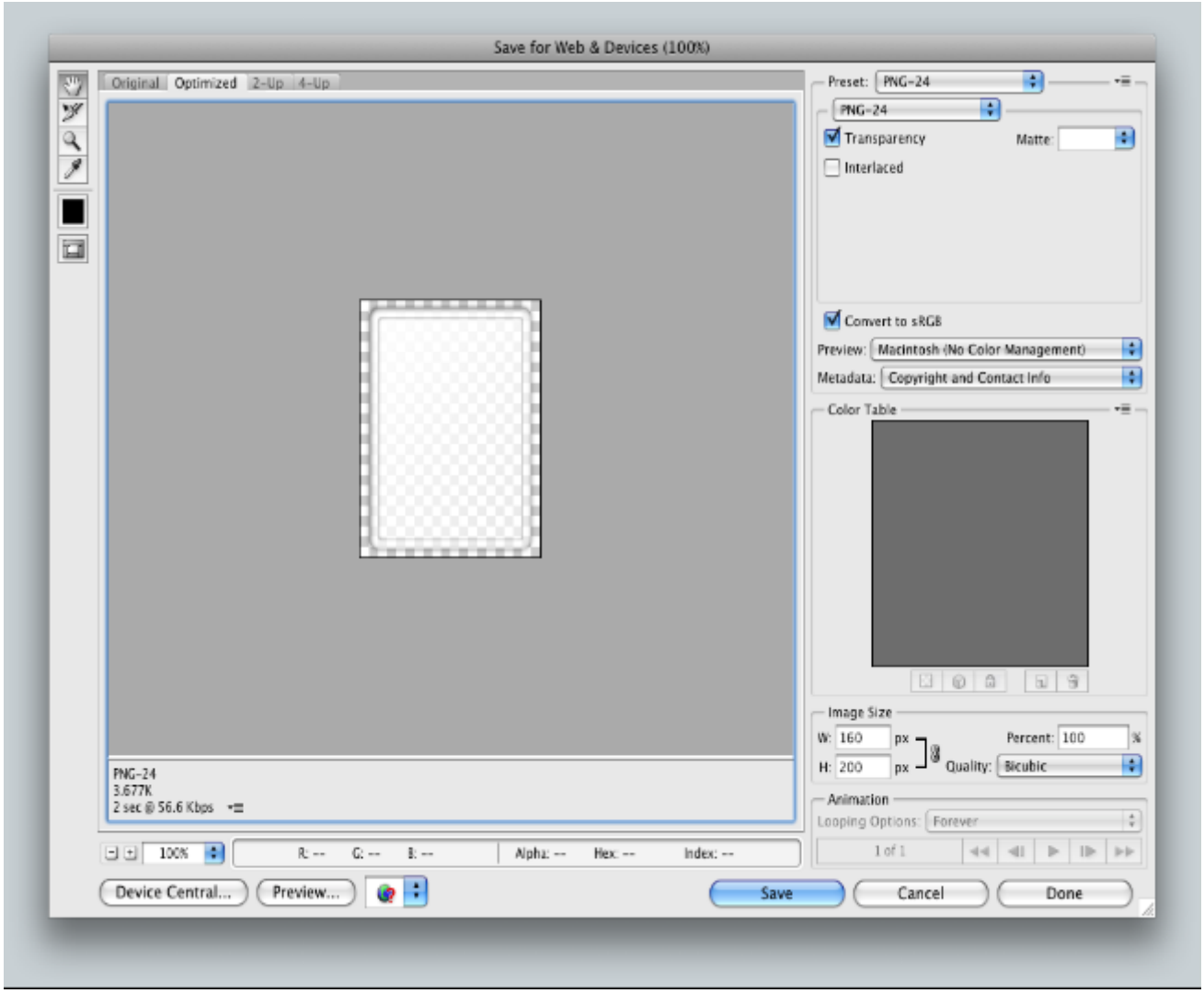
注意：目前 android widget 模板使用的是自定义梯度角，这意味着 9 修补技术（9-patch techniques)不能用于优化尺寸。

然而， 9 修补技术可以用来设置内容填充方面。

在某些情况下，器件在较低的像素深度可能造成视觉分条和抖动问题。为了解决这个问题，程序应用开发者应该通过“proxy（代理）”定义 drawable 的 XML ：这种技术参考起初的作品，"background.9.png"的情况下，指示作品做出需要的抖动。

widget 的图形文件格式（Widget graphics file format）

使用恰当的 bounding box 规格，在透明背景，8-bit 颜色，使用 PNG-24 格式进行保存。



4. AppWidget 调用之系统时钟篇

作者：bangbang.song@gmail.com

AppWidget 允许应用向别的应用提供一个可视的小部件(widget),本质上是一个 BrodaReceiver ,接受系统广播的

APPWIDGET_CONFIGURE,APPWIDGET_DELETED,APPWIDGET_UPDATE,APPWIDGET_DISABLED,APPWIDGET_ENABLED,APPWIDGET_PICK 消息，做相应的回应，以更新所提供小部件的参数，包括但不限于小部件外形，对单击事件的回应，等等。

如图：

AppWidgetHost AppwidgetProvider

| |

| APPWIDGET_CONFIGURE |

| ----------> |

| |

| setResultCode() *1

| APPWIDGET_UPDATE |

| ----------> |

| |

| AppWidgetManager.upateViews()

otherDealing()

| APPWIDGET_ENABLED |

| ----------> |

| |

| AppWidgetManager.upateViews()

otherDealing()

… … … …

*1: 动作 APPWIDGET_CONFIGURE 不是以广播的形式发送，而是以正常启动一个 Activity 的方式[startActivity(Intent)]方式传送，因而在处理这个动作时有特殊的方法。

当然，你也可以在任何时候更新你提供的部件参数，比如对某个特定系统事件(动作)的响应，此时，可以实例化一个 BroadcastReceiver 类的子类，对想要响应的事件作出反应。

通常我们要响应的动作是 APPWIDGET_UPDATE,本例子将调用系统时间，在 Home Screen 上显示一个当前日期及时间的部件，向大家展示 AppWidget 的魅力。

4.1 实现需求

1.在主页面上添加一个显示系统日期及时间的部件。

2.可以自行设置部件的标题。（没有实际意义，供示例作用）

4.2 分解步骤

1.要能能够向主页面提供一个部件有以显示，提供者必须实现对 APPWIDGET _ UPAGET 消息的处理，通过 AppWidgetManager.updateViews(),就能更新你提供的部件。为简化任务，Android 为我们提供了一个容易实

现的类 AppWidgetProvider 专门用以与 AppWidgetHost 沟通。为完成此任务，我们只要重载 AppWidgetProvider

的 onUpdate()方法就能完成，代码片断如下：

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager,
                    int[] appWidgetIds) {
    String title = context.getSharedPreferences(widgetConfigure.PREF,
0)                                                    .getString(widgetConfigure.TITLE, "");
    String systemTtme = getSystemTimeString();
    update(context, appWidgetManager, appWidgetIds, title, systemTtme);
     }

/*
* 更新部件
*
* title:      部件标题
* systemTime: 更新的时间
*
* 此处仅是一个例子，你可以做任何你想做的，
*/
private void update(Context context, AppWidgetManager appWidgetManager,
                    int[] appWidgetIds, String title, String systemTime) {
```

```
        if (LOG ) {
                Log.d(TAG,"update widget: titile = " + title + " systemtime = " + systemTime);
                }

        /* 此处仅是一个例子，你可以做任何你想做的，*/
        RemoteViews views = new
RemoteViews(context.getPackageName(),R.layout.appwidget_provider);
        views.setTextViewText(R.id.widget_title, title);
        views.setTextViewText(R.id.widget_system_time, systemTime);
            appWidgetManager.updateAppWidget(appWidgetIds, views);
         }
```

注：

因为本质上 AppWidgetProvider 是一个 BroadcastBreciver, 因而在 AndroidManifest.xml 文件里，有这样的声明：

```
<receiver android:name=".widgetSystemTimeProvider"
   android:label="小部件例子(调用系统时间)"
   android:icon="@drawable/icon"><!-- 显示的图标 -->
      <intent-filter>
        <!-- 当部件更新时，会收到此动作消息 -->
          <action android:name="android.appwidget.action.APPWIDGET_UPDATE"/>
        </intent-filter>
```

```
        <!-- shit the android.appwidget.provider you can not change this, so am I  -->
         <meta-data android:name="android.appwidget.provider"
                    android:resource="@xml/appwidget_info">
         </meta-data>
 </receiver>
```

注意其中的< meta-date>元素 ,name 必须是 android.appwidget.proivder,其 android:resource 属性指定你所提供的小部件的属性，让我们来看一看

res/xml/appwidget_info.xml:

```
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="30dp"
    android:initialLayout="@layout/appwidget_provider"
    android:updatePeriodMillis="500"
    android:configure="person.bangbang.widget.widgetConfigure">
</appwidget-provider>
```

其中的 initialLayout 指定该部件的布局（外形），可以看到，其实是一个通常的布局而已，后两个参数接下来会讲到。

让我们来看看我们的部件长什么样，

res/layout/appwidget_provider:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 指定小部件的外形 -->
<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/p_2x2">
 <TextView android:id="@+id/widget_title"
          android:paddingTop="@dimen/padding_top"
         android:paddingLeft="@dimen/padding_left"
         android:singleLine="true"
         android:layout_width="fill_parent"
          android:layout_height="wrap_content"/>
  <TextView
        android:id="@+id/widget_system_time"
      android:paddingLeft="@dimen/padding_left"
     android:layout_width="fill_parent"
      android:layout_height="fill_parent"/>
</LinearLayout>
```

可以看到，跟一般的布局文件没什么两样。

好了，到现在为止，第一个功能基本完成，不过有一个小缺陷，既然是显示系统时间，当然要时时更新，不可能我们时时去点击而更新，这时就要用到 appwidget_info.xml 文件里的 updatePeroidMillis 属性了，这个属性指定了更新我们提供的小部件的间隔时间（以毫秒记）。 因为我们希望能按时（以秒记）更新，就希望系统能在小于一秒的时间内至少更新两次，所以取 0.5 秒。

2. 通常 ,在每提供一个部件时 ,我们期望能能够对部件的某些参数进行设定 ,这时动 APPWIDGET_CONFIGURE 就出场了 ,因为这个消息是按照通常开启 Activity 的方法进行通信 ,我们就需要实现一个能接受该消息的 Activity, 这个我就不必哆嗦，看如下代码片断：

```
<activity android:name=".widgetConfigure"
android:label="配置部件标题">
<intent-filter>
  <action   android:name="android.appwidget.action.APPWIDGET_CONFIGURE"/>
</intent-filter>
<intent-filter>
  <action android:name="android.intent.action.MAIN"/>
  <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
<intent-filter>
   <action android:name="android.intent.action.MAIN"/>
   <category android:name="android.intent.category.LAUNCHER"/>
 </intent-filter>
</activity>
```

注意<intent-filter>中有声明将处理 APPWIDGET _ CONFIGURE 消息。

这里我们仅是作示范 ,设置我们部件的标题。这样每当更新时 ,两个组件(Component)就通过 SharePreference 等到设置的标题，从而就可以更新我们提供的部件啦。

4.3 本章总结

通过 AppWidget,我们可以轻易的扩展我们的应用，加强两个应用之间的通信，以一种更加吸引用户的方式。特别适合在主页面上添加这种部件，快速而轻量地向用户提供有用信息，并可实现两者的互动。

欢迎指正，讨论：bangbang.song@gmail.com

5. 让我们的 Widget 和 service 打交道

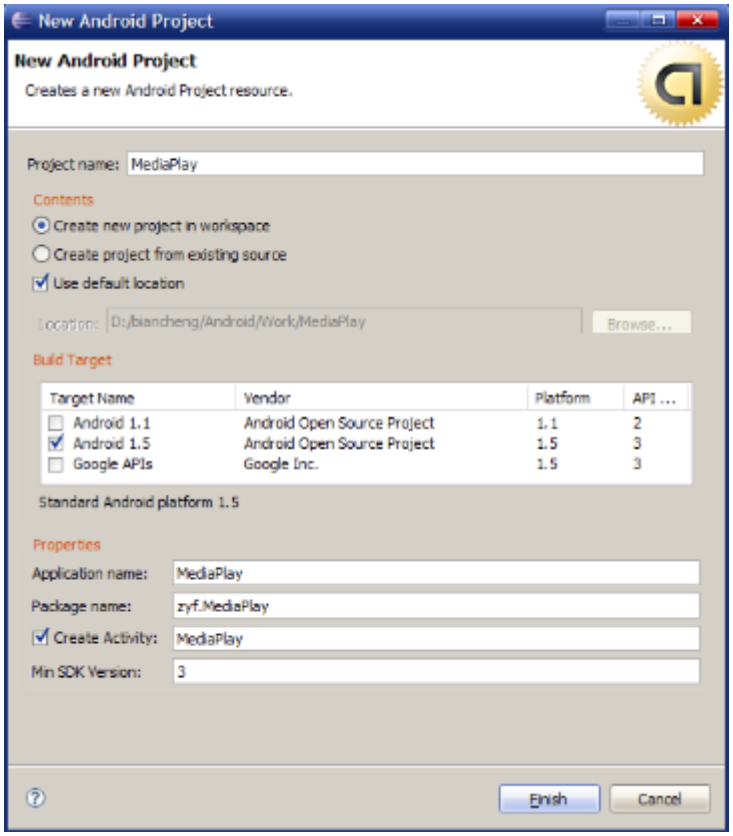
本章节主要讲解一个音乐播放的实例 ,首先创建一个 Acticity ,然后创建一个 Service 进行音乐播放 ,在这个基础之上我们再添加一个 Widget 的简单应用。然后在桌面上添加这个 Widget，我的例子中主要包含：

- 创建一个 Widget 的必要元素：(AppWidgetProviderInfo 对象、AppWidgetProvider 类、View layout(视图布局))
- 如何在 AndroidManifest.xml 中声明一个 Widget。
- AppWidgetProviderInfo 元数据的添加
- Widget 布局的基本设计
- AppWidgetProvider 类的简单使用
- 如何在桌面添加 Widget

5.1 实例演示

5.1.1 程序 Activity、Service 部分

1. 新建工程 MediaPlay，如图：



2. 向 SDCard 中导入一个 mp3 文件

3. 设计程序主页面屏幕 XML 布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget33"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
   android:orientation="vertical"
 android:background="@drawable/back">
<LinearLayout
android:id="@+id/widget36"
```

```
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="@drawable/bt_group_back"
>
<Button
android:layout_width="64px"
android:layout_height="fill_parent"
android:background="@drawable/ic_cmd_last"
android:id="@+id/button_Last_main">
</Button>
<Button
android:layout_width="64px"
android:layout_height="fill_parent"
android:background="@drawable/ic_cmd_replay"
android:id="@+id/button_Replay_main">
</Button>
<Button
android:layout_width="64px"
android:layout_height="fill_parent"
android:background="@drawable/ic_cmd_next"
android:id="@+id/button_Next_main">
</Button>
```

```
<Button
android:layout_width="64px"
android:layout_height="fill_parent"
android:background="@drawable/ic_cmd_list"
android:id="@+id/button_List_main">
</Button>
<Button
android:layout_width="65px"
android:layout_height="fill_parent"
android:background="@drawable/ic_cmd_pause"
android:id="@+id/button_Pause_main">
</Button>
</LinearLayout>

<ImageView android:id="@+id/ImageView_Copy"
android:layout_width="80px"
android:layout_height="80px"
android:background="@drawable/music_icon">
</ImageView>
</LinearLayout>
```

效果如图：

4. 创建一个 Service 子类 MusicService，添加音乐服务，从 SDCard 中读取一个 mp3 文件

```
package zyf.MusicPlay;

import java.io.IOException;

import android.app.Service;

import android.content.Intent;

import android.media.MediaPlayer;

import android.net.Uri;

import android.os.IBinder;

public class MusicService extends Service {

        // 声明一个 MediaPlayer 实例对象

        private MediaPlayer myMusicPlayer;

        @Override
```

```
        public IBinder onBind(Intent intent) {
         // TODO Auto-generated method stub
                                return null;                }
         //  服务启动方法
                                @Override
                               public void onStart(Intent intent, int startId) {
         // TODO Auto-generated method stub
                               super.onStart(intent, startId);
         //  初始化MediaPlayer对象 ,从SDCard中读取一首音乐，
 //Uri.parse("file:///sdcard/music/ronanmetal.mp3")表示读取  mp3  文件的资源  Uri  引
用

 myMusicPlayer=MediaPlayer.create(this,Uri.parse("file:///sdcard/music/ronanmetal.
mp3"));                                    try {
          // Prepares the player for playback
        //  准备播放器，进行回放
      myMusicPlayer.prepare();
    }
catch (IllegalStateException e)
{
// TODO Auto-generated catch block
        e.printStackTrace();
```

```
}
catch (IOException e)
{
// TODO Auto-generated catch block
      e.printStackTrace();
  }
//  启动 MediaPlayer
      myMusicPlayer.start();
   }
  //  服务销毁方法
@Override
public void onDestroy() {
   // TODO Auto-generated method stub
 super.onDestroy();
        //  停止音乐播放
   myMusicPlayer.stop();                    }
}
```

5. 向 AndroidManifest.xml 中添加该 Service

```
<service android:name="MusicService">
                        <intent-filter>
                                <action android:name="zyf.MusicPlay.START_AUDIO_SERVICE">
                                </action>
                                <category
                android:name="android.intent.category.DEFAULT"></category>
                        </intent-filter>
                </service>
```

6. MediaPlay.java 中添加 MENU 菜单，以及初始化各个按钮，初步设计程序框架

```
package zyf.MusicPlay;


import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
```

```
public class MusicPlay extends Activity implements OnClickListener {


        /** Called when the activity is first created. */
        private Intent musicIntent;
        //声明定义 Menu 菜单选项
        private MenuItem back_Menu, next_Menu, stop_Menu, selsect_Menu,
        exit_Menu;
        //定义各个 Menu 菜单的 ID
        private final int Back_Menu_ID = Menu.FIRST + 1;
        private final int Next_Menu_ID = Menu.FIRST + 2;
        private final int Selsect_Menu_ID = Menu.FIRST + 3;
        private final int Restart_Menu_ID = Menu.FIRST + 4;
        private final int Exit_Menu_ID = Menu.FIRST + 5;
        //声明定义主屏幕按钮
        private Button bt_last, bt_play, bt_next, bt_list, bt_stop;


        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                //加载主屏幕布局 main.xml
                setContentView(R.layout.main);
                //在自定义方法中从 XML 布局中获取按钮并初始化
```

```
        findMyButton();
        //给各个按钮添加事件监听器，监听器由本类来实现 OnClickListener 接口
        myButtonSetListener();
        //新创建一个 Intent，把启动的服务加到 Intent 中
        musicIntent=new Intent("zyf.MusicPlay.START_AUDIO_SERVICE");
    }
    private void myButtonSetListener() {
        // TODO Auto-generated method stub
        //给按钮添加事件监听器
        bt_last.setOnClickListener(this);
        bt_play.setOnClickListener(this);
        bt_next.setOnClickListener(this);
        bt_list.setOnClickListener(this);
        bt_stop.setOnClickListener(this);
    }

    private void findMyButton() {
        // TODO Auto-generated method stub
        //从 xml 中通过 ID 来获取 Button 对象
        bt_last = (Button) findViewById(R.id.button_Last_main);
        bt_play = (Button) findViewById(R.id.button_Replay_main);
        bt_next = (Button) findViewById(R.id.button_Next_main);
```

```
        bt_list = (Button) findViewById(R.id.button_List_main);
        bt_stop = (Button) findViewById(R.id.button_Pause_main);
    }

    public void onClick(View button) {
        // TODO Auto-generated method stub
        //根据 button.getId()获取 View 的 ID 来判断事件是由那个按钮来发出的，然后进行相应的事件处理
        switch (button.getId()) {
        case R.id.button_Replay_main: {
            //启动音乐播放服务
            this.startService(musicIntent);
        }
        break;
        case R.id.button_Pause_main: {
            //停止音乐播放服务
            this.stopService(musicIntent);
        }
        break;
        default:
            break;
        }
```

```
    }

    @Override

    public boolean onCreateOptionsMenu(Menu menu) {

            // TODO Auto-generated method stub

            //为程序添加 MENU 菜单项    add(组 ID，菜单项 ID，顺序,标题)

            back_Menu = menu.add(0, this.Back_Menu_ID, 1, "上一曲");

            next_Menu = menu.add(0, this.Next_Menu_ID, 2, "下一曲");

            stop_Menu = menu.add(0, this.Restart_Menu_ID, 3, "停止");

            selsect_Menu = menu.add(0, this.Selsect_Menu_ID, 4, "歌曲列表");

            exit_Menu = menu.add(0, this.Exit_Menu_ID, 5, "退出");

            //为程序添加 MENU 菜单项添加图标

            back_Menu.setIcon(R.drawable.ic_cmd_last);

            next_Menu.setIcon(R.drawable.ic_cmd_next);

            stop_Menu.setIcon(R.drawable.ic_cmd_pause);

            selsect_Menu.setIcon(R.drawable.ic_cmd_list);

            exit_Menu.setIcon(R.drawable.ic_cmd_power);

            return super.onCreateOptionsMenu(menu);

    }

    @Override

    public boolean onOptionsItemSelected(MenuItem item) {

            // TODO Auto-generated method stub

            //新建一个 Intent，用来启动下一个 Activity，显示音乐选项列表，(未完成，后期制作)
```

```
                Intent nextIntent=new Intent();
                //设置启动 Activity 类
                nextIntent.setClass(MusicPlay.this, MusicList.class);
                //启动 Activity
                startActivity(nextIntent);
                return super.onOptionsItemSelected(item);
        }
}
```

界面如图：

5.1.2 Widget 添加部分

1. 在 AndroidManifest.xml 中声明一个 App Widget

```
<receiver android:name="MusicAppWidgetProvider">
                        <intent-filter>
                                <action
            android:name="android.appwidget.action.APPWIDGET_UPDATE">
        </action>
                        </intent-filter>
                        <meta-data
        android:name="android.appwidget.provider"
                                android:resource="@xml/example_appwidget_info">
      </meta-data>
</receiver>
```

•android:name - 指定 metadata (元数据)名称。.使用 android.appwidget.provider 来识别数据，并作为 AppWidgetProviderInfo 描述符。

•android:resource - 指定 AppWidgetProviderInfo 资源位置.

2. 创建 App Widget 布局

在 res/layout 目录中创建一个新的布局，名为 mywidgetlayout.xml，进行初步的设计

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

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
        android:background="@drawable/widget_frame"
        android:layout_gravity="center_horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
          android:background="@drawable/music_icon"
          android:id="@+id/Button_music_widget"
                android:layout_marginTop="20px"
                android:layout_width="60px"
                android:layout_height="60px"
                android:layout_marginLeft="21px">
        </Button>
        <LinearLayout
          android:id="@+id/LinearLayout01"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
        </LinearLayout>
        <LinearLayout
          android:id="@+id/widget_layout_widge"
                android:layout_height="wrap_content"
```

```
android:background="@drawable/bt_group_back"
android:layout_gravity="center_vertical"
android:layout_marginLeft="21dp"
android:layout_width="200px">
<Button
  android:layout_height="fill_parent"
  android:background="@drawable/ic_cmd_last"
      android:layout_width="50px"
      android:id="@+id/button_Last_widget">
</Button>
<Button
  android:layout_height="fill_parent"
  android:background="@drawable/ic_cmd_replay"
      android:layout_width="50px"
      android:id="@+id/button_Replay_widget">
</Button>
<Button
  android:layout_height="fill_parent"
  android:background="@drawable/ic_cmd_next"
      android:layout_width="50px"
      android:id="@+id/button_Next_widget">
</Button>
```

```
                <Button
                  android:layout_height="fill_parent"
                        android:layout_width="50px"
                        android:background="@drawable/ic_cmd_pause"
                        android:id="@+id/button_Pause_widget">
                </Button>
        </LinearLayout>
</LinearLayout>
```

效果如图：

3. 添加 AppWidgetProviderInfo 元数据

在 res 目录下，新建一个 xml 目录，在其中新建一个名为 example_appwidget_info.xml 的 XML 文件,内容如下

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
  android:minWidth="294dp"
  android:minHeight="72dp"
  android:updatePeriodMillis="86400000"
  android:initialLayout="@layout/mywidgetlayout"
```

```
/>
```

4. 添加 AppWidgetProviderInfo 元数据

新建一个 MusicAppWidgetProvider 类，继承自 AppWidgetProvider

```
package zyf.MusicPlay;

import android.app.PendingIntent;

import android.appwidget.AppWidgetManager;

import android.appwidget.AppWidgetProvider;

import android.content.Context;

import android.content.Intent;

import android.widget.RemoteViews;

public class MusicAppWidgetProvider extends AppWidgetProvider {


        @Override

        public void onDeleted(Context context, int[] appWidgetIds) {

                // TODO Auto-generated method stub

                super.onDeleted(context, appWidgetIds);

        }

        @Override

        public void onDisabled(Context context) {

                // TODO Auto-generated method stub
```

```
        super.onDisabled(context);
    }
    @Override
    public void onEnabled(Context context) {
        // TODO Auto-generated method stub
        super.onEnabled(context);
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        super.onReceive(context, intent);
    }
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
            int[] appWidgetIds) {
        // TODO Auto-generated method stub
        super.onUpdate(context, appWidgetManager, appWidgetIds);
    }
}
```

5. 使用 Widget 来启动一个 Activity

```
@Override
```

```
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
                         int[] appWidgetIds) {
                // TODO Auto-generated method stub
                super.onUpdate(context, appWidgetManager, appWidgetIds);


                final int N = appWidgetIds.length;
                // 为每一个属于本 provider 的 App Widget 执行此循环程序
for (int i=0; i<N; i++) {
   int appWidgetId = appWidgetIds[i];


   // 创建一个 Intent 来启动 ExampleActivity
    Intent intent = new Intent(context, MusicPlay.class);
    PendingIntent pendingIntent =
            PendingIntent.getActivity(context, 0, intent, 0);
   //为 App Widget 取得布局并且拴一个 on-click 监听器到该 button 上
    RemoteViews views =
  new RemoteViews(context.getPackageName(), R.layout.mywidgetlayout);
    views.setOnClickPendingIntent(R.id.Button_music_widget, pendingIntent);
    // 告诉 AppWidgetManager 在当前 App Widget 上 执行一次更新
    appWidgetManager.updateAppWidget(appWidgetId, views);


}
```

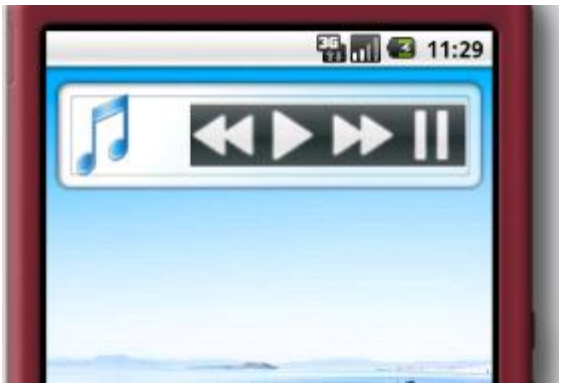
```
    }
```

6. 使用 Widget 来启动一个 Service

```
@Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
                    int[] appWidgetIds) {
                // TODO Auto-generated method stub
                super.onUpdate(context, appWidgetManager, appWidgetIds);


                final int N = appWidgetIds.length;
                // 为每一个属于本 provider 的 App Widget 执行此循环程序
  for (int i=0; i<N; i++) {
    //新创建一个 Intent，把启动的服务加到 Intent 中
                Intent musicIntent=new Intent("zyf.MusicPlay.START_AUDIO_SERVICE");
  //启动音乐播放服务
                context.startService(musicIntent);
  }
      }
```

5.2 桌面添加 Widget 后的效果



5.3 总结

5.3.1 AppWidgetProvider 类的几个方法

AppWidgetProvider 类继承 BroadcastReceiver 作为一个 便捷类来捕获 App Widget 广播(broadcasts). AppWidgetProvider 只接收与这个 App Widget 有关的事件广播(event broadcasts)，例如当该 App Widget 被更新、删除、启动和禁用。当这些广播事件发生，AppWidgetProvider 收到以下方法调用：

- onUpdate(Context, AppWidgetManager, int[])

这个方法在 AppWidgetProviderInfo 中的属性 updatePeriodMillis 定义的时间间隔被调用来更新 App Widget (参看上面 添加 AppWidgetProviderInfo 元数据). 该方法当用户添加这个 App Widget 时也被调用，因此,它应该履行基本设置,例如为 Views 定义事件捕捉器(event handlers) 如果必要并启动一个临时的 Service。然而，如果你已经声明了一个配置 Activity, 当用户添加该 App Widget 时该方法不会被调用，但是随后的更新时会被调用。当配置完成时，配置 Activitiy(configuration Activity)的责任是执行第一次更新。(参看下面 创建一个 App Widget 配置 Activity)

- onDeleted(Context, int[])

当一个 App Widget 从 App Widget host 中被删除，这个方法每次被调用.

- onEnabled(Context)

当一个实例 App Widget 首次被创建时，这个方法被调用。例如，如果用户添加你的两个 App Widget 实例，这个方法只在第一次被调用。如果你需要开启一个新的数据库或是执行其他对于所有 App Widget 实例只需要发生一次的安装，那么这个方法中是一个很好的地方来做这事情。

- onDisabled(Context)

当你的最后一个 App Widget 实例被从 App Widget host 中删除时，此方法被调用。这是你应该清理任何在 onEnabled(Context)中做的事情,例如删除一个临时的数据库。

- onReceive(Context, Intent)

每一个广播此方法都被调用，并且先于以上回调方法。你一般不需要实现此方法，因为默认的 AppWidgetProvider 实例过滤所有的 App Widget broadcasts 并且在适当时调用以上方法。一个可以捕捉其他 App Widgets 的应用程序组件被叫做 App Widget host。以下屏幕快照展示了 Music App Widget。

5.3.3 AppWidget 的布局

创建 App Widget 布局你必须在 XML 中为你的 App Widget 定义一个初始布局并且把它保存在工程的 res/layout/目录。你可以使用以下列出的 View 对象来设计你的 App Widget，但是在你开始设计你的 App Widget 之前，请阅读并理解 App Widget 设计指导。

如果你熟悉在 XML 中声明布局，那创建 App Widget 布局很简单. 然而，你必须意识到 App Widget 布局基于 RemoteViews，它不支持所有种类的布局 Layout 或是 view widget。

一个 RemoteViews 对象(并且, 必然地,一个 App Widget)能支持以下布局 Layout 类：

- FrameLayout
- LinearLayout
- RelativeLayout

和以下 widget 类:

- AnalogClock
- Button
- Chronometer
- ImageButton
- ImageView
- ProgressBar
- TextView

这些类的子类不被支持。

6. **维基词典 每日一词，教你怎样**和 web service 打交道

游利卡 http://www.cnblogs.com/pcedb0189/

代码下载：

网盘地址：http://cid-5456d4b8f95608a9.skydrive.live.com/self.aspx/Android/SimpleWiktionary/SimpleWiktionary.rar

上一篇中，我们对 widget 和 Service 之间的交互有了很深的了解，这一篇，我们再次看一个使用 Service 的例子。

6.1 简介

这个项目是由一位名叫 Jeffrey Sharkey 写的一个简单的 Widget。功能很简单，就是随机显示维基词典中的每日一词。
http://code.google.com/p/wiktionary-android/source/browse/trunk/SimpleWiktionary/
大家可以通过这个地址访问原项目的代码，或者查看更新、联系作者。我也试图联系过作者，但是人家不理我。
运行的最终效果如下，在这个只有两个单元大小的 Widget 中，显示有一些简单的背景图片和维基词典的每日一词。每次触摸这个 Widget，他都会自动变换背景，并且直接访问维基站点，这样你就能获得关于这个词条更多的信息，并且访问维基更多的内容。

由于我的网络环境不太好，正在加载 ——loading word of day...

现在显示出了我们的每日一词—— horseshoe （马蹄鞋）

下面显示这个词的含义 一个 U 形的装在马脚上的金属鞋

右上角 有我一个简单的 start_logo（貌似不是很好看）

点击以后 图片改变 我们伟大的天安门啊

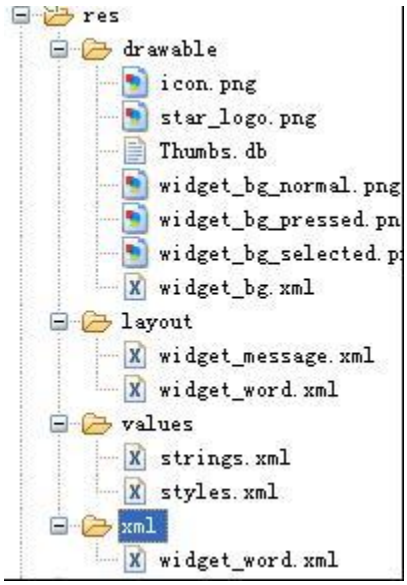
点击后访问维基字典的页面

6.2 让我们看一下后台的代码

维基字典 每日一词，其实他的结构还是非常简单的，和一个普通的 app 应用程序基本相同，只是我们这里没有 Activity。

在 src 文件夹里只有 两个类 一个是 SimpleWikiHelper 这个是用来定义访问维基字典的接口。另一个就是我们的 Widget 了。

我们首先来看一下布局代码， 在 res 文件夹里主要用来定义一些布局信息。他的结构也非常简单 Drawable 文件夹里主要是我们的背景图片，还有我们一个简单的背景信息



Layout/Widget_message.xml

Widget_word.xml

这两个文件我们主要来定义 Widget 相关控件的信息

第五期的特看重对资源文件进行了非常详细地介绍。

drawable 是存放图片文件。当大家在模拟器或者真机中选中 Widget 的时候，会发现图片会变化，点击的时候也会发生变化。这些就是 widget_bg.xml 的作用了。

layout 是布局文件。维基词典用一个巧妙的办法来改变了结构，在启动的时候使用 widget_message 的布局，成功获取词条以后，再使用 widget_word.xml 这个布局。

values 这里存放了文字的 Strings syles 存放了风格文件。

xml 下的则是定义了 widget 的整体的布局。000 毫秒一次也就是一天的时间，下面是相关的代码

android:updatePeriodMillis="86400000"

为了节省篇幅 大家可以自己下载代码来观看，基本上有过 appUI 设计经验的朋友，这些都能很容易理解。

6.3 我们下来说一下 SimpleWikihelper.java

他的功能就是帮我设定好，访问维基词典的借口，就像是我们要吃饭，饭店给我们的菜谱一样。

这里将所有的 注释做了中文翻译，基本上大家都可以完整阅读

/**

```
* SimpleWikiHelper 帮我们实现了一个轻量级的维基字典的接口。当做任何数据请求时，我们需要做一个

* 基于你的应用程序的包的名字和版本的链接，来请求所要求的维基词典中的内容。

* Helper methods to simplify talking with and parsing responses from a

* lightweight Wiktionary API. Before making any requests, you should call

* {@link #prepareUserAgent(Context)} to generate a User-Agent string based on

* your application package name and version.

*/

public class SimpleWikiHelper {

private static final String TAG = "SimpleWikiHelper";

/**

* 每日一词的变量(这里应该是做了一个正则表达式的限定)

*/

public static final String WORD_OF_DAY_REGEX =

"(?s)\\{\\{wotd\\|(.+?)\\|(.+?)\\|([^#\\|]+).*?\\}\\}";

/** * 访问和请求接口具体信息的链接，通过它来获取我们的每日一词

* Partial URL to use when requesting the detailed entry for a specific

* Wiktionary page. Use {@link String#format(String, Object...)} to insert

* the desired page title after escaping it as needed.

*/

private static final String WIKTIONARY_PAGE =

"http://en.wiktionary.org/w/api.php?action=query&prop=revisions&titles=%s&" +

"rvprop=content&format=json%s";
```

```
/** *当每日一次出现了你感兴趣的词汇的时候，你可以通过下面提供的链接，来访问维基的页面获得更多的信息 */
private static final String WIKTIONARY_EXPAND_TEMPLATES =
"&rvexpandtemplates=true";
/**当没有服务器应答时出现的 Http 状态码
* {@link StatusLine} HTTP status code when no server error has occurred.*/
private static final int HTTP_STATUS_OK = 200; /**
  * 缓存的大小
* Shared buffer used by {@link #getUrlContent(String)} when reading results
* from an API request. */
private static byte[] sBuffer = new byte[512];
  /**  * 当做请求的时候生成的字符串，他们应该在请求之前被创建。
* User-agent string to use when making requests. Should be filled using
* {@link #prepareUserAgent(Context)} before making any other calls.*/
private static String sUserAgent = null;
  /**  * 当出现接口出现问题是，抛出的异常，这可能是因为网络的错误，
 * 也有可能是服务器返回了一个错误的状态码
* Thrown when there were problems contacting the remote API server, either
* because of a network error, or the server returned a bad status code. */
public static class ApiException extends Exception {
        private static final long serialVersionUID = 1L;
```

```
public ApiException(String detailMessage, Throwable throwable) {
super(detailMessage, throwable);
}
public ApiException(String detailMessage) {
super(detailMessage);
} }

/**  * 当解析接口对请求的回应是发生错误时，抛出异常，也有可能呼应是空的或者是不规则的。
* Thrown when there were problems parsing the response to an API call,
* either because the response was empty, or it was malformed.*/
public static class ParseException extends Exception {
        private static final long serialVersionUID = 1L;

public ParseException(String detailMessage, Throwable throwable) {
super(detailMessage, throwable);
} }

/**  * 准备好一个字符串以备使用。这里要包含应用程序的名称和版本。
* Prepare the internal User-Agent string for use. This requires a
* {@link Context} to pull the package name and version number for this
* application. */
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);
} }
```

```
/**  * 这时我们就要读取并且向 widget 返回一个准确维基词典页面。这是一个轻量的接口的请求
  * 而被请求的页面有大量的内容，那么我们就会做一些筛选，来返回到 widget 上。
  * 因为这个请求指导最后一刻才会有效，所以 widget 是不会调用一个 UI 的线程。
* Read and return the content for a specific Wiktionary page. This makes a
* lightweight API call, and trims out just the page content returned.
* Because this call blocks until results are available, it should not be
* run from a UI thread.
*
* @param title The exact title of the Wiktionary page requested.
*       维基字典 每日一词的标题
* @param expandTemplates If true, expand any wiki templates found.
*       扩展的模板，如果这个可用，那么他将被找到的维基模板。
```

```
* @return Exact content of page.
*       准确地内容
* @throws ApiException If any connection or server error occurs.
*       有关连接的错误信息
* @throws ParseException If there are problems parsing the response.
*       有关内容解析的错误信息
*/
public static String getPageContent(String title, boolean expandTemplates)
throws ApiException, ParseException {
// Encode page title and expand templates if requested
String encodedTitle = Uri.encode(title);
String expandClause = expandTemplates ? WIKTIONARY_EXPAND_TEMPLATES : "";

// Query the API for content
String content = getUrlContent(String.format(WIKTIONARY_PAGE,
encodedTitle, expandClause));
try {
// Drill into the JSON response to find the content body
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);

}

}


/**

  * 我们通过 url 得到了最原始的内容。这个请求指导作业完成时，才会被同步因为还需要缓冲

* Pull the raw text content of the given URL. This call blocks until the

* operation has completed, and is synchronized because it uses a shared

* buffer {@link #sBuffer}.

* @param url The exact URL to request.

*       对地址的请求

* @return The raw content returned by the server.

*       原生的信息

* @throws ApiException If any connection or server error occurs.

*       错误信息

*/

protected static synchronized String getUrlContent(String url) throws ApiException {

if (sUserAgent == null) {
```

```
throw new ApiException("User-Agent string must be prepared");
}

// Create client and set our specific user-agent string
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(url);
request.setHeader("User-Agent", sUserAgent);

try {
HttpResponse response = client.execute(request);

// Check if server response is valid
StatusLine status = response.getStatusLine();
if (status.getStatusCode() != HTTP_STATUS_OK) {
throw new ApiException("Invalid response from server: " +
status.toString());
}

// Pull content stream from response
HttpEntity entity = response.getEntity();
InputStream inputStream = entity.getContent();
```

```
ByteArrayOutputStream content = new ByteArrayOutputStream();


// Read response into a buffered stream
int readBytes = 0;
while ((readBytes = inputStream.read(sBuffer)) != -1) {
content.write(sBuffer, 0, readBytes);
}


// Return result from buffered stream
return new String(content.toByteArray());
} catch (IOException e) {
throw new ApiException("Problem communicating with API", e);
}
}
}
```

6.4 最后来看一下我们的 widget.java

widget.java 是主要定义我们的 widget 的后台代码 ,在这里我们可以看到 widget 是如何使用 Service 来从 wiki 的 API 来获取信息的。并且如何显示到我们的屏幕上。

这里简单介绍一下维基词典的原理。widget 其实是继承了 BroadcastReceiver，所以可以使用很多 BroadcastReceiver 的方法，例如 OnStart()、OnReceive()等。

但是这里维基词典使用了，一个内部服务，即 Service。Service 调用了 SimpleWikiHelper 来获取了每天的词条。

如果拿到了词条就显示可以接受词条的布局，如果没有则显示另外一个布局。

原理就这么简单。如果你能获得金山词霸的 API，你也可以做出一个同样的。

```
/**定义一个简单的插件来展示一下维基词典的“每日一词”。建立一个更新让我们能在后台调用 API
 * Define a simple widget that shows the Wiktionary "Word of the day." To build
 * an update we spawn a background {@link Service} to perform the API queries.
 */
public class WordWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
            int[] appWidgetIds) {
                //为了防止假死状态的超时，我们会一个后台服务中进行更新
       // To prevent any ANR timeouts, we perform the update in a service
        context.startService(new Intent(context, UpdateService.class));
    }

    public static class UpdateService extends Service {
        @Override
        public void onStart(Intent intent, int startId) {
                    //建立插件每天的更新
          // Build the widget update for today
           RemoteViews updateViews = buildUpdate(this);
          //开始在我们的屏幕上进行更新
```

```
    // Push update for this widget to the home screen
    ComponentName thisWidget = new ComponentName(this, WordWidget.class);
    AppWidgetManager manager = AppWidgetManager.getInstance(this);
    manager.updateAppWidget(thisWidget, updateViews);
}

/**建立一个可以更新维基词典“每日一次”的 Widget，必须等待的 API 反馈
* Build a widget update to show the current Wiktionary
 * "Word of the day." Will block until the online API returns.
 */
public RemoteViews buildUpdate(Context context) {
        //从资源中获取月份信息
    // Pick out month names from resources
    Resources res = context.getResources();
    String[] monthNames = res.getStringArray(R.array.month_names);
    //找出当前的月日
    // Find current month and day
    Time today = new Time();
    today.setToNow();
    //构建每日一次的标题，像 “维基词典 ：每日一次 /三月 21”
    // Build today's page title, like "Wiktionary:Word of the day/March 21"
    String pageName = res.getString(R.string.template_wotd_title,
```

```
            monthNames[today.month], today.monthDay);
        RemoteViews updateViews = null;
        String pageContent = "";


        try {
            // 试试查询一下维基词典的每日一次的 应用程序接口
            // Try querying the Wiktionary API for today's word
            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);
        }
                // 用一个习惯性的表达式来分析词和他们的定义
        // Use a regular expression to parse out the word and its definition
        Pattern pattern = Pattern.compile(SimpleWikiHelper.WORD_OF_DAY_REGEX);
        Matcher matcher = pattern.matcher(pageContent);
        if (matcher.find()) {
            //建立更新的更新服务器和 Widget 的内容
            // Build an update that holds the updated widget contents
            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());
        //当用户点击 Widget 的时候，弹出维基词典的对这个词条的解释页面
       // When user clicks on widget, launch to Wiktionary definition page
        String definePage = res.getString(R.string.template_define_url,
                Uri.encode(wordTitle));
        Intent defineIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(definePage));
        PendingIntent pendingIntent = PendingIntent.getActivity(context,
                0 /* no requestCode */, defineIntent, 0 /* no flags */);
        updateViews.setOnClickPendingIntent(R.id.widget, pendingIntent);

    } else {
            //没有找到今天的句子，那就看看错误信息
       // Didn't find word of day, so show error message
        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) {
                //看来我们并不需要去关心服务
           // We don't need to bind to this service
            return null;
        }
    }
}
```

6.5 总结

这个"维基词典 每日一次"是一个非常简单的程序,就是在之前调用维基的接口还有定义错误信息的时候稍微感觉有一些繁杂。但是在最后的效果是非常好的。虽然内容不多，但是很完整地实现了一个小巧的功能。

好了，谢谢大家。

##App Widget 编程基础实战(三部曲)

来自地狱怒兽 暑期巨献

7 . 先看 AppWidget

7.1App Wiget 生命周期——App Widget 编程基础实战(一)

① 首先，搭建好一个 Widget 框架具体步骤省略

② 首次添加 Widget 到桌面 调用过程

1	public void onEnabled(Context context)
2	public void onReceive(Context context, Intent intent)
3	public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
4	public void onReceive(Context context, Intent intent)

③ 第二次再加入 Widget 到桌面 调用过程

1	public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
2	public void onReceive(Context context, Intent intent)

④ 删除多个桌面 Widget 中的一个 调用过程

1	public void onReceive(Context context, Intent intent)

⑤ 删除多个桌面 Widget 中的最后一个 过程调用

1	public void onReceive(Context context, Intent intent)
2	public void onDisabled(Context context)
3	public void onReceive(Context context, Intent intent)

7.2App Wiget 生命周期方法解析

onUpdate(Context, AppWidgetManager, int[])

每次更新时候都是调用这个方法。利用 AppWidgetManager 呼叫更新。

这个方法在 AppWidgetProviderInfo 中的属性 updatePeriodMillis 定义的时间间隔被调用来更新 App Widget (参看上面 添加 AppWidgetProviderInfo 元数据). 该方法当用户添加这个 App Widget 时也被调用 ,因此,它应该履行基本设置,例如为 Views 定义事件捕捉器(event handlers)，如果必要并启动一个临时的 Service。 然而，如果你已经声明了一个配置 Activity, 当用户添加该 App Widget 时**该方法不会被调用**，但是随后的更新时会被调用。当配置完成时，配置 Activitiy(configuration Activity)的责任是执行第一次更新。(参看下面 创建一个 App Widget 配置 Activity)

onDeleted(Context, int[])

一般在 Widget 被删除时候框架调用此方法。在此，你可以回收一些资源，释放资源。

当一个 App Widget 从 App Widget host 中被删除，这个方法每次被调用.

onEnabled(Context)

当你每次加入一个 Widget 到你的桌面时候，此方法会被调用，在这里，你可以初始化一些变量，或是开启一个服务。

当 一个实例 App Widget 首次被创建时，这个方法被调用。例如，如果用户添加你的两个 App Widget 实例，这个方法只在第一次被调用。如果你需要开启一个新的数据库或是执行其他对于所有 App Widget 实例只需要发生一次的安装，那么这个方法中是一个很好的地方来做这事情。

onDisabled(Context)

最后一个 Widget 从桌面删除时候，调用此方法，可以释放资源，删除临时的东西。

当你的最后一个 App Widget 实例被从 App Widget host 中删除时，此方法被调用。这是你应该清理任何在 onEnabled(Context)中做的事情,例如删除一个临时的数据库。

onReceive(Context, Intent)

如果代码中有广播的话，那么这个方法就是经常调用的了。而且在 onUpdate()方法前后，都会调用这个方法，来看看接收广播情况。

每一个广播此方法都被调用，并且先于以上回调方法。你一般不需要实现此方法，因为默认的 AppWidgetProvider 实例过滤所有的 App Widget broadcasts 并且在适当时调用以上方法。

8.如何实现 Widget 的更新——App Widget 编程基础实战(二)

Widget 的更新其实就是在 onUpdate()方法中利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新，当然，你也可以在 onUpdate()方法之外来更新 Widget。

8.1 如何实现 TextView 的更新

- 在 onUpdate()方法内部更新

1. 在 onUpdate()方法内部，声明一个 RemoteViews 对象

```
RemoteViews views;
```

2. 实例化 RemoteViews

```
RemoteViews views=
```

```
new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
//第一个参数 包名，第二个参数 Widget 布局 ID
```

3. 设置 TextView 要显示的字符串(颜色)

```
views.setTextViewText(R.id.TextView, "这里是更新显示的文字");
//设置 Widget 布局中 id 为 R.id.TextView 的 TextView 的更新字符串
views.setTextColor(R.id.TextView, Color.RED);
//设置 Widget 布局中 id 为 R.id.TextView 的 TextView，字体颜色为 Red
```

4. 利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新

```
appWidgetManager.updateAppWidget(appWidgetIds, views);
//更新 AppWidget，第一个参数 AppWidgetID，第二个参数 要更新的 View
```

- 在 onUpdate()方法外部更新

1. 首先在外部，一般是 Service 中定义几个变量

```
private static AppWidgetManager appManager;
//定义 APP Widget 管理器，用来更新 RemoteView
private static RemoteViews views;
//RemoteView 对象定义，是更新的单元核心
private static ComponentName thisWidget;
//组件名
```

2. 实例化这些变量

```
appManager=AppWidgetManager.getInstance(XXService.this);
views = new RemoteViews(XXService.this.getPackageName(),
                                                R.layout.widget);
thisWidget = new ComponentName(XXService.this,XXWidget.class);
```

3. 设置 TextView 要显示的字符串(颜色)

```
views.setTextViewText(R.id.TextView, "这里是更新显示的文字");
//设置 Widget 布局中 id 为 R.id.TextView 的 TextView 的更新字符串
views.setTextColor(R.id.TextView, Color.RED);
//设置 Widget 布局中 id 为 R.id.TextView 的 TextView，字体颜色为 Red
```

4. 利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新

```
appManager.updateAppWidget(thisWidget, views);//更新 AppWidget
```

8.2 如何实现 ImageView 的更新

- 在 onUpdate()方法内部更新

1.在 onUpdate()方法内部，声明一个 RemoteViews 对象

```
RemoteViews views;
```

2.实例化 RemoteViews

```
RemoteViews views=
                new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
//第一个参数 包名，第二个参数 Widget 布局 ID
```

3.设置 ImageView 要显示的图片

```
views.setImageViewResource(R.id.ImageView, R.drawable.icon);
//第一个参数 ImageView 对象的 ID，第二个参数，要显示的图片的 ID
```

4.利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新

```
appWidgetManager.updateAppWidget(appWidgetIds, views);
//更新 AppWidget，第一个参数 AppWidgetID，第二个参数 要更新的 View
```

- 在 onUpdate()方法外部更新

1.首先在外部，一般是 Service 中定义几个变量

```
private static AppWidgetManager appManager;
//定义 APP Widget 管理器，用来更新 RemoteView
private static RemoteViews views;
//RemoteView 对象定义，是更新的单元核心
private static ComponentName thisWidget;
//组件名
```

2.实例化这些变量

```
appManager=AppWidgetManager.getInstance(XXService.this);
views = new RemoteViews(XXService.this.getPackageName(),
                                        R.layout.widget);
thisWidget = new ComponentName(XXService.this,XXWidget.class);
```

3.设置 ImageView 要显示的图片

```
views.setImageViewResource(R.id.ImageView, R.drawable.icon);
//第一个参数 ImageView 对象的 ID，第二个参数，要显示的图片的 ID
```

4.利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新

```
appManager.updateAppWidget(thisWidget, views);//更新 AppWidget
```

8.3 如何实现 ProgressBar 的更新

额外说一下 ProgressBar，普通模式样式是一个圆形，会旋转的 View.如果要显示成进度条长方形，要在 XML 的 ProgressBar 声明定义里面，ProgressBar 属性 中加入：

style="?android:attr/progressBarStyleHorizontal"

- 在 onUpdate()方法内部更新

1.在 onUpdate()方法内部，声明一个 RemoteViews 对象

```
RemoteViews views;
```

2.实例化 RemoteViews

```
RemoteViews views=
                new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
//第一个参数 包名，第二个参数 Widget 布局 ID
```

3.设置 ProgressBar 要显示的进度

```
views.setProgressBar(R.id.ProgressBar, 100, 40, true);
//第一个参数 ProgressBar 对象的 ID，第二个参数进度条最大值，
//第三个参数要显示的进度，第四个参数 true 说明进度条是无定期的
```

4.利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新

```
appWidgetManager.updateAppWidget(appWidgetIds, views);
//更新 AppWidget，第一个参数 AppWidgetID，第二个参数 要更新的 View
```

- 在 onUpdate()方法外部更新

1.首先在外部，一般是 Service 中定义几个变量

```
private static AppWidgetManager appManager;
//定义 APP Widget 管理器，用来更新 RemoteView
private static RemoteViews views;
//RemoteView 对象定义，是更新的单元核心
private static ComponentName thisWidget;
```

```
//组件名
```

2.实例化这些变量

```
appManager=AppWidgetManager.getInstance(XXService.this);
views = new RemoteViews(XXService.this.getPackageName(),
                                                R.layout.widget);
thisWidget = new ComponentName(XXService.this,XXWidget.class);
```

3.设置 ProgressBar 要显示的进度

```
views.setProgressBar(R.id.ProgressBar, 100, 40, false);
//第一个参数 ProgressBar 对象的 ID，第二个参数进度条最大值，
//第三个参数要显示的进度，第四个参数 true 说明进度条是无定期的,一般用
false
```

4.利用 AppWidgetManager 调用 updateAppWidget()方法来实现 Widget 的更新

```
appManager.updateAppWidget(thisWidget, views);//更新 AppWidget
```

如何实现 AnalogClock 的更新

- AnalogClock 一般情况下，会自动更新，也就是可以作为一个装饰品。

8.5 如何实现 Chronometer 的更新

- 在 onUpdate()方法内部更新
- 在 onUpdate()方法外部更新

基本方法跟以上类似

```
views.setChronometer(R.id.Chronometer,SystemClock.elapsedRealtime(),
null, true);
//第一个参数 Chronometer ID,第二个参数 Timer 读取的时间，格式 0：00
//第三个参数 格式“0：00”，null 则简单的显示 Timer 值
//第四个参数 是否开启 Timer，true 开启 Timer
appWidgetManager.updateAppWidget(appWidgetIds, views);
//更新 AppWidget
```

9.如何实现 Widget 的按钮事件——App Widget 编程基础实战(三)

ImagButton / Button 都是 Widget 所支持的控件，在这里，我们用来实现开启 Activity、开启 Service，发送广播

9.1 如何开启 Activity

① 首先，在 OnUpdate()方法定义一个 Intent，用来开启 Activity 的 Intent

```
Intent startActivityIntent=new Intent(context ,WidgetLife.class);
```

第一个参数上下文，第二个参数 - 要开启的 Activity 类

② 用该 Intent 实例化一个 PendingIntent

```
PendingIntent Pintent=
                    PendingIntent.getActivity(context, 0,
startActivityIntent, 0);
```

③ 实例化 RemoteView,其对应相应的 Widget 布局

```
RemoteViews ActivityView=
            new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
```

④ 给 RemoteView 上的 Button 设置按钮事件

```
ActivityView.setOnClickPendingIntent(R.id.Button, Pintent);
```

⑤ 更新 Widget

```
appWidgetManager.updateAppWidget(appWidgetIds, ActivityView);
```

9.2 如何开启 Service

① 首先，在 OnUpdate()方法定义一个 Intent，用来开启 Service 的 Intent

```
Intent startServiceInten=new Intent("zyf.temp.Service.START");
```

参数是一个开启 Service 的动作

② 用该 Intent 实例化一个 PendingIntent

```
PendingIntent Pintent=
                    PendingIntent.getService(context, 0, startServiceInten,
0);
```

③ 实例化 RemoteView,其对应相应的 Widget 布局

```
RemoteViews ServiceView=
            new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
```

④ 给 RemoteView 上的 ImageButton 设置按钮事件

```
ActivityView.setOnClickPendingIntent(R.id.ImageButton, Pintent);
```

⑤ 更新 Widget

```
appWidgetManager.updateAppWidget(appWidgetIds, ServiceView);
```

9.3 如何发送按钮 Action

① 首先，在 OnUpdate()方法定义一个 Intent，用来发送 Action 的 Intent

```
Intent sendActionInten=new Intent("zyf.test.SendAction");
```

参数是一个动作

② 用该 Intent 实例化一个 PendingIntent

```
PendingIntent Pintent=
                    PendingIntent.getBroadcast(context, 0,
sendActionInten, 0);
```

③ 实例化 RemoteView,其对应相应的 Widget 布局

```
RemoteViews sendActionView=
            new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
```

④ 给 RemoteView 上的 ImageButton 设置按钮事件

```
sendActionView.setOnClickPendingIntent(R.id.ImageButton, Pintent);
```

⑤ 更新 Widget

```
appWidgetManager.updateAppWidget(appWidgetIds, sendActionView);
```

⑥ Widget 自己在 onReceive()方法中接收获取这个 Action，要在 Manifest.xml 中加入 Action

```
<receiver android:name="Widget_Main">
```

```
    <intent-filter>
      <action
            android:name="android.appwidget.action.APPWIDGET_UPD
            ATE" >
      </action>
      <action
android:name="zyf.test.Send.MESSAGE.MYTEST"></action>
    </intent-filter>
    <meta-data android:resource="@xml/widget_provider"
            android:name="android.appwidget.provider"></meta-data>
</receiver>
```

⑦ 使用 intent.getAction()来获取 Action

```
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        super.onReceive(context, intent);
        if(intent.getAction().equals("zyf.test.SendAction")){
            Toast.makeText(context,"获取到 Action 了！！",
                                    Toast.LENGTH_LONG).show();
        }
    }
```

9.4 如何在 Service 中接收 Widget Button Action

在 Service 中接收广播或是 Action，都是以 Intent 为载体的。如果只是一个动作，那么直接发送带有 Action 的 Intent，如果是要发送广播，那么，还要 Intent 上面附加发送的信息。

在 Service(外部的一个单独的 Service ,或是内部类的一个 Service ,前提都是开启 Service ）中给 Service 注册一个 BroadcastReceiver

a) 先实例化一个 IntentFilter，也就是过滤器，从 Intent 的信息中进行过滤，选出合适的 Intent

```
IntentFilter filter=new IntentFilter();
```

b) 给 IntentFilter 加入过滤信息，addAction（ ）

```
filter.addAction("zyf.test.Send.MESSAGE.MYTEST");
```

c) 给 Service 注册 BroadcastReceiver

```
Service.registerReceiver(new BroadcastReceiver(){
        @Override
        public void onReceive(Context context, Intent intent) {
            // TODO Auto-generated method stub
        }, filter);
```

d) 在 onReceive()中判断 Action

```
if(intent.getAction().equals("zyf.test.Send.MESSAGE.MYTEST")){

        Log.d("Dog", "@#@#@#@#@#----->onReceive");
    }
```

如果 Intent 有附带信息，那么应该从 Intent 中通过 getXXXExtra()系列的方法来获取

9.5Service 如何发送 Broadcast

① 首先，Service 和 Activity 都可以用 sendBroadcast(Intent intent)方法来发送附带信息的 Intent

② 定义实例化 Intent

```
Intent petState = new Intent("com.eoe.eoepet.DOG.STATE");
```

③ 给 Intent 附加广播信息

```
petState.putExtra("Name_Value", petName);
```

④ 发送

```
sendBroadcast (petState);
```

10.Log 调试支招——App Widget 编程基础实战(额外)

在复杂的程序中，有时候会碰到 1000 行级别的代码。这时候，不要晕，要学会跟踪代码，你可以使用 Debug 视图来进行断点调试，也可以使用 Toast 提示调试，呵呵，现在我主要讲一下我自己的 Log 调试。

各种 Log

Log.d(tag, msg);	显示 Debug 调试信息
Log.e(tag, msg);	显示 Error 错误信息
Log.i(tag, msg);	显示 Info 信息
Log.v(tag, msg);	显示 VERBOSE 详述信息
Log.w(tag, msg);	显示 Wrang 警告信息

也许，现在各种符号有了很好的作用了，现在 Tag 自己规定一下，例如“MyTag”

好了在你需要调试的地方加入一些相应的调试信息，注意，为了分清哪个是哪个，我一般都是采用特殊符号方式。在 DDMS 的 log 标签中，添加一个 TAG 为“MyTag” 的 Log，选择好 d、e、i、v、w，中的一种，应该要和你的方法一致。

例如：

Log.d(“MyTag”,“#########>>>>the Service is Started>>”);

Log.d(“MyTag”,“@@@@@@@@>>>> the Activity is OnCreate>>”);

Log.d(“MyTag”,“$$$$$$$$$>>>>>myService----->OnReceive()>>”);

```
Log.d(“MyTag”,“<<<Change Dog
Name---->onChange()--->Name==” +Name );
Log.d(“MyTag”,“##$$%%@@ this may be have an Error>>>#######” );
Log.d(“MyTag”,“<<<←—Debug boolean--->>>” +booleanA);
```

这样，你可以很清楚的看到，程序在哪里出问题了。因为 Log 太多了，或许你一下找不到，但是有了添加过滤器，然后再加入一定符号的 Log 信息，相信您调试没有问题了，而且相当轻松。对了，最后发布程序，这些一定要记得删除哦。

11.Android Widget 调查报告

来自游利卡 的 暑期巨献
本片将将不以代码为重 ，从另外一个侧面给大家讲述一下 Widget 的开发。

11.1 开始

对于什么是 Widget ，以及 Android 的 Widget 大家通过以前的文章基本上已经了解了。而本次调查哦报告将带大家更好得认识 Android 的 Widget。让大家更加熟练 Widget 的开发。

那么刚开始，希望大家回顾一下第四期的第一篇文章《让我们再次从 Widget 的 Helloworld 开始》。
当时我将代码全部贴出，我们可以看到这个 Widget 非常简单，他的结构就是 Mainfest 注册文件，一些布局信息，再加上最后一个集成了 appWidget 的类，类中东西也非常简单，就是一个 RemoteViews！

而对于 Widget 来说，RemoteViews 就是 Widget 表现层的最核心的要素！

也因此，Android 平台的 Widget 表现力就这样被判了重刑。Widget 为什么弱，到底弱到了什么程度，我们接下来通过对 SDK 的分析来继续。

11.2 探究 Appwidget 还有 Widget 之间的关系

当下载好了 Android 的 SDK 后，将你的路径中的这个地址/android-sdk-windows-1.5_r2/docs/index.html ，复制到你的浏览器，或者直接打开就可以进入离线版的 SDK 了。

然后我们点击 Reference，看到左边的类的列表，我们就会看到 Appwidget 这个就是我们要来分析的。

这里我想特别说明一下，虽然都是 Widget，但是在 Android 平台却存在两种不同的 Widget 在 SDK 目录中 ,我们也可以看到这里有一个 android.appwidget ,还有一个 android.widget。前者就是我们本期的主题 widget。而后者则是小部件的意思，我们可以打开 ImageView 还有 textview 的 SDK 中来看，会发现，他们全都是继承这个类。
其实我们 app 中的每一个 UI 层的表现元素，都是以 widget 的名义存在的。

但是后来 Google 发现当初这样做不太好，因为大家公认的 widget 发展越来越好，可是 widget 名称已经被占用了，所以只能做一个 Appwidget 了。

所以在你看 SDK 并且希望将相关的内容应用到你自己的程序中的时候一定要分析清楚，也许平时可以直接用 Widget 指代我们所说的 appWidget，但是在 SDK 中，两者必须区分开。

那我们打开 Appwidget 以后，我们会出现

AppWidgetHost

AppWidgetHostView

AppWidgetManager

AppWidgetProvider

AppWidgetProviderInfo

这几个类中 ,我们最常用的是 AppwidgetProvider。这各类提供了我们最基本的一个 widget。AppwidgetHost 和 AppwidgetHostView 可以帮助我们来承载 Appwidget 。进入程序管理，我们可以发现，Home 其实是以一个程序的形式出现，当然这个是不可卸载的。而我们普通点的 Widget 都是以 Home 为依托的，Home 便使用了 AppwidgetHost 这个类。那么我们的自己的 Activity 也可以添加 Widget，在你的程序内部只要使用这个 Host，然后就可以再你的 Activity 挂载你需要的 appwidget 了。

而 AppwidgetProviderinfo 则可以定义 widget 的信息，比如说 appwidget 的大小等等，当然我们通过 xml 文件也可以达到同样的效果。

另一个 AppwidgetManager 可以让我们用外部方法来管理 Appwidget。我们最常用的就是使用 AppwidgetManager 来进行 appwidget 的更新。

最后一个 AppwidgetProvider 则是我们最重要的类了，因为他决定了我们的 widget 可以达到什么样的表现效果。

因为这些内容 SDK 中都有，而且本篇的重点并不是简单罗列 SDK，如果大家有什么方法的需要，最有效的方法还是去翻 SDK 中的相关内容。

所以呢，大家大部分人所说的 Widget 都是 指的是 Appwidget，因为大家都默认了，我们也不用关心具体的名称使用了。当然本篇为了区别，当然还是会使用 Appwidget 指代当前的 Widget。

11.3 Appwidget 贫弱的真正罪魁祸首

而我们现在着重分析一下 Appwidgetprovider 这各类。首先打开 SDK，我们在最左上角的地方就可以看到这个类继承与 BroadcastReceiver。BroadcastReceiver 是 Android 平台接受广播的方法。那就是说什么呢？

也许 Google 一开始就没想过让 appWidget 有什么表现力，限制他做输出，继承 BroadcastReceiver 更加有利于 Appwidget 进行数据的接受。

也许有朋友会说了，这下好了啊，甚至都不用引入了 BroadcastReceiver 了，那你就大错特错了。因为 Appwidget 继承这个只是为了给自己图方便。

他是 Appwidget，他不是我们平常所用的 Activity。

也就是说，很多适用于 Activity 的东西，Appwidget 绝大多数都无法使用。这是真的，就拿一点简单的来说吧每次我们从 app 引入东西，我们最常用的获取控件 Id 的方式，就是 findviewbyid。

而我们可以很明确的告诉大家，Appwidget 不支持这种方法！而且这个方法很重要，没了他很多东西基本上就类似判了死刑。

这里首先来探讨一下控件与我们后台代码之间的关系

~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~

可能有些朋友会认为，起码之前我是这么认为的。就是所有的控件都是被后台代码获取，然后后台代码来进行对屏幕的输出。但实际并不是这样的，只要后台代码声明了布局中的控件，只告诉屏幕有这个东西，但是如何控制它则是另外一种方法。

这样来说吧！假如说 Android 是一个大饭店的老板，我们的 UI 控件相当于老板点的菜。 我们的后台代码则像是服务生。

有的时候，我们之需要让服务生通知到老板要这个菜就可以了，然后菜就直接被送到了。这里服务生可以不对这些菜品做任何变动，老板也满意。也就是之前所说的直接声明控件。

但有的时候，为了让老板满意，服务生必须对菜品进行控制，比如过热的菜需要冷却一下，不太好的东西要去掉。等等。但是服务生怎么知道是那些菜品呢？这里就需要一个 findviewbyid 了。
他就像是超市中的读取条形码的那个设备。我们的动画效果，图片切换，等等都是通过这个来做到的。

而 appWidget 很明确不支持这个方法。就像是这个老板很苛刻，如果菜品不好，他也不愿让服务生来加一些作料进行调整。

~~~~~~~~~~~~~~~~~~~~

在 Appwidget 上，你可以让一个 ImageView 显示出来，但是你没办法控制它。
~~~~~~~~~~~~~~~~~~~~

Android 的 Appwidget 简直就和一个聋子一样 ,改变它的唯一方法 就是更新整个 Widget，这个类似于让厨房重新做菜。

我真感觉 Android 真的是给 Widget 安了只猪耳朵（Android 谐音)！

不支持 findviewbyid，这让很多特效就这样和你的 widget,say goodbye 了

当然我和兽兽在做我们的 Widget 的时候，也不甘心与就那么放纵大好的 Animation 效果不用，但是试了很多的方法，但是最后除了惨痛的教训什么都没有获得！

Appwidget 最支持什么呢？是 RemoteViews，这个是 Appwidget 支持最好的一个控件了，所以你无论如何得保证你的 Appwidget 中至少有一个 RemoteViews。

RemoteViews 可以支持你将 Widget 自由拖动。

11.4 不要对 UI 说无所谓

也许有朋友会对此不屑一顾，但是我想特别声明一下，如果手机平台不去关注 UI，不去考虑表现，他还应该考虑些什么呢？界面层是手机最总要的一层，手机屏幕狭小，不像 PC。所以一个明了的界面会给你的 app 带来
和那的提升。
现在的手机很想当年的 PC，性能已经不算数特别大的问题了。这是事实，而手机的品种越来越多，而现在的消费者角度，普通的消费者(除了那些技术出身的发烧友)，他们才不会管你是用了什么技术、什么系统。
他们只在乎自己的使用体验。

如果一个手机 app，功能再强大，没有一个号的 UI，没有一个强大的表现层，用户还是不买账的。那些山寨系统能获得消费者的青睐也是抓住了消费者的这一心里。

11.5 Widget 的救命稻草 Service

Appwidget 的贫弱 不仅仅是在已经被我们无奈的布局上。appWidget 还被 Android 做了很多限制，甚至 Widget 不能调用线程！不知道是我和兽兽之前没做好，还是其他的，反正我们一直没能成功调用线程。

如果你要在 AppwidgetProvider 里面写东西，你所能调用的资源相当有限。

我感觉 apwidget 像是寄生在 Home 程序中的，所以也受相应的很多限制。我们的经验告诉我们，除了在 RemoteViews 中方一些固定的东西，不要指望你的 widget 能自己做什么，最好的方法是让他当一个被动接受的傀儡。

既然 Widget 成了傀儡了，那么我们就应该把更多的事务交给别人。Service 就是个很好的选择，他可以让 Widget 重焕生机。

其实 Service 是 Activity 还有 Appwidget 都可以调用的东西。而且他也有很高的灵活度。

而且更加重要的是，你甚至可以存在于你的 Widget 里面作为一个内部类。因为 Service 占用了进程，他可以完成很多 Widget 无法做到的事情。

Service 在做好了自己的工作以后，可以用广播的形式发送给 Widget，因为 appWidgetManager 可以在 Widget 之外调用，那么我们完全可以在 Service 里面控制 Widget 的更新。

Widget 的接受方法解释使用 BroadcastReceiver 原声的继承方法 ,在本篇调用系统时钟那个例子就很好的讲述了这个方法。

11.6 从 Widget 逃出来的信息

如果什么事情都让别人来做，也不好，appwidget 虽然被限制很多，但是我们还是有办法从 Widget 分出点东西来。RemoteViews 给我们提供了原生的 setOnClickPendingIntent 方法。

PendingIntent 这个想必大家都应该很清楚。因为 RemoteViews 没有按键时间，所以这个方法就可以出发 PendingIntent。

而 Widget 的点击事件很有特点，之前点击我们 Activity 的控件，会有专门的按键监听，他会根据案件的 ID ,来控制按键范围 ,比如你的 ImageViewButton 的 id 是 R.id.imagebutton。在下面你只要引入将这个 id，按钮事件发出只会在你点击这个区域以后才会发生。

下面这个例子就是点击整个 Widget 的时候会触发时间。

```
updateViews = new RemoteViews(context.getPackageName(), R.layout.widget_word);

updateViews.setOnClickPendingIntent(R.id.widget, pendingIntent);
```

PendingIntent 被出发后，发出信息，所以送出送出的消息全部在 Intent 里面。

至于 Android 上的传值，大家可以去这个地址来学习一下。

http://www.eoeandroid.com/viewthread.php?tid=967&extra=page%3D1

我觉得我讲的还算是很明白的。

PendingIntent 提供了三种方法，一个事 getActivity()另一个是 getBroadcast()最后一个事 getService()

从字面上我们就可以看出这三种方法是启动一个新的 Activity、发送一个广播、还有启动一个服务。getService 启动的服务如果已经启动了，执行这个语句将会保持服务的运行。

一些具体编程细节问题，本片的 AppWidget 编程实战 三部曲已经给打击很明确的答案了。这里不再编码赘述。

12. Android Widgets，潜力无穷

翻译：thanjing

原文：http://htcsource.com/index.php/Android/Android-Widgets-unlimited-potential.html

在未来两周内，沃达丰将会开始发售 HTC MAGIC。这将会是第二部采用 android 操作系统的手机，但这是第一次采用 android 1.5 版(别名为 cupcake)。在这个新的版本中包含了相当多的改进 ,当我评测 google 版和为 HTC MAGIC 定制的 1.5 版本的 android 时，我写过一些关于这些改进的东西。但是在 android1.5 版的介绍中最引人注目的一个新增功能无疑是对于桌面 widgets 的支持。

android widgets 不是什么新鲜事物。google 在原始的 1.0 版本中包含了三个(指针时钟,搜索框,相框)并且将它应用在了 T-Mobile G1 上。但令开发者失望的是 ,android 开发团队并没有发布任何有关如何开发附加 widgets 的文档。

xda-developers.com 上的一个开发小组曾研究出了如何美化 android 自带的 widgets ,但是进展也仅限于此了。

４月１３日，新的 1.5 版本的 android 发布，终于提供了 android appWidget 框架。开发者现在有工具可以去创建几乎任何一种他们曾梦寐以求的 widgets.它能由独立的应用程序组成(例如:天气预报，当你从商店中下载了天气频道的应用，它会创建并且安装一个 widget)或者被单独地当做应用来使用，所有的信息都直接从互联网获得。

那么，将来我们会看到什么类型的 android widget 呢？虽然 android 开发团队已经开发了日历 widget 和音乐 widget，但是这并不会阻止个任何人去开发他们自己的版本。一旦 widgets 开始出现在 market 上，我打赌绝大多数的 widgets 将会是那些 android 自带的 widgets 的简单替代品。至今为止我相信大多数人都厌倦了看着

完全一样的指针时钟，而一少部分开发者将会尝试创造更好看的音乐 widget.最终，大多数 widgets 将会是 RSS feeds,新闻标题和股票走势。

如果开发者选对了方向，一个出色的 widget 会将让他们获得成功。 我彻底考察了在 market 上我认为售价合理的 widgets，其中的大多数都被我从桌面中删除了，因为我认为它们不值得这个价格。可能至少对我而言，我更愿意花 0.99 美元去买一个设计良好的可以在桌面上给我需要的信息的 widget，而不是一个功能相似但是设计很烂的 widget。

下面是我列的我希望不久之后可以看到的 widgets 的名单：

twidroid widget: twidroid 是目前为止在 android 上最好的 Twitter 应用,我很希望能直接在桌面上写 Twitter。我可以想象有两个不同的 widgets(一个用来写 tweeting，另外一个用来浏览 tweets)。

天气预报 widget: 自从我曾被困在办公室过后，我每天都会用几次天气预报的程序。我想象有一个简单的 android widget 显示当前室外的天气情况（晴天，多云，阵风，下雨，下雪）以及实时的气温,当你轻击这个 widget 时桌面上会有一个天气的动态展示，那会是多么的神奇！想象下使用 TouchFLO 3D 效果的 H T C 天气标签——而且是直接显示在你屏幕上的，用雨或雪的特效来覆盖你的图标和其它 widgets。

CNN widget: 听起来可能很像书呆子，我是一个新闻癖,我每天都会在 CNN 或者其他新闻网站上面逛。我很乐意去拥有一个显示最近的头条新闻以及图片视频等链接的应用来代替在 twitter 上跟随 CNN。

Rss Fedd widget: 最近我发现 Twitter 已经开始侵蚀 Rss 的地盘了,例如我们在 Twitter 上添加的跟随比一年中添加的 RSS 订阅还要多。幸运的是，RSS 是一个自动更新的系统，它给与 RSS FEED 订阅者所有的信息，而不仅仅是给与订阅者他们所 Tweet 的信息。

以上只是一个我所希望在我手机上看到的 android widget 的简单罗列，我确信我还有一堆其他的关于 widget 的想法，但也许我更应该让你来分享下一些想法，让我们知道什么样的 android widgets 是你所希望看到的。

13.Android Widget 开发 建议

地狱怒兽&游利卡 联合撰写

可能程序员都已经习惯了以程序员的角度来思考问题。因为我们是干这行的，自然也很习惯这么做。

但是用户可能并不这么想。用户才不会去管到底是 Linux 还是 Windows，是 iPhone，还是 Android，在用户眼中，好用的，方便的，能让他带来惊喜的就是好东西。

说点番外话，拿电梯举例，人们喜欢电梯就是因为他能为人们节省时间和体力。他的原理就是一个能东的箱子吗！

而 Widget 也一样的。为什么 Chrome、Safari 都急着退出缩略图功能，因为这样最直观，最能为用户节省很多时间还有精力。

在 Widget 上，用户不用再重复打开一个应用程序，东西就放在桌面上，拿起来就能用。这就是 Widget 的魅力，就是让我们进一步得变懒。

所以我们在做 Widget 的时候，我个人觉得最重要的还是用户体验，以及 UI。其实这是整个移动领域都特别关注的地方。但是 Widget 需要更多关注这些，所以，如果你把一个 Widget 做的很复杂，那还不如用 App 得了。

开发人员应该将更多精力放在如何帮助用户简化自己的工作为主。

比如说天气 ,如果可以调用 GPS ,让 Widget 自己获得所在的地区的经纬度 ,然后调用 google 的 天气信息，这不是更好吗？

如果你愿意迎着 Widget 的限制勇敢开发下去，这里给你几条建议

1.等着　等着 Google 放出最新版的 SDK 增强 Widget 的功能

2.试试 Ophne 的 Widget，Ophone 的 Widget 使用了 Javascript，虽然不清楚具体会是什么效果，但是 Javascript 的灵活性有目共睹

3.试试用最新的 NDK 开发出自己的库文件(只是猜测，但是我感觉 Widget 还是要运行在 HOME 下，还是免不了限制)

4.可以使用让 Widget 依附于 Activity，比如说你开发了一个 Activity，那么让 Activity 多出一个 Widget 的小功能。

5.或用 Service，尽量让 Widget 编程信息接收的载体。

6.尽量小巧灵活一些，eoePet Beta v1.0 版本主要是考虑到 也许有些评为喜欢这样大块头而做的。如果第一眼也许大的 Widget 感觉很爽，但是如果想要在用户的桌面驻留，还是一个小巧实用的更加合适一些。

7.剑走偏锋，记住要和 App 区分开来，Widget 应该有自己别样的天空

8.widget 目前来说，属于偏静态的东西，要在其上面显示，你就得更新，而官方推荐更新率尽量低。

9.动画的原理就是一张张图片快速切换，而电影胶片也是如此。有人说，这个动画好简单，其实不然，我想你也会绞尽脑汁的。Widget 目前来说，局限很大。

10.比比 Opera，呵呵，Opera 的 Widget 很多，可以自己定制，有游戏，日历，播放器，一些贴身助理。就像一个 app，Android 的 Widget 实在是赶不上。目前来说 Android Widget 就是一个提示，方便的快捷方式多了几个交互。

11.如果想要 Hero 的那样，我觉得你还是先把基础搞懂，然后自己编写类库，就像 AdView，你可以实现其他高级功能。

12.一个简单的 Widget 用一个类就够了，Widget 自己给自己发消息，要在 Manifest 里面加入 Action

13.如果你只要显示一个时钟，那就用用个 AnalogClock,就给他背景装饰一下。

14.注意 Widget 的横屏幕显示，直屏幕下的 Layout 和横屏幕下的 Layout 分开设计，或是用一个小的。横屏幕布局最大应该在 250*250

14.eoePet 开发纪实

游利卡 http://www.cnblogs.com/pcedb0189/

当初确定的名字是 eoePet,结果后来不知道怎么改成了 EoePetPlay，最后的应用的名字也变了，但是终究是一个产品。

10.1 大约一个月之前

eoePet 算是我和兽兽，自从入行以来第一个比较拿得出手的作品了。对于我来说更是有很多意义，一个月前的平台迁移，在 Java 领域还有很多不适用，就是因为 IT168 一个活动广告，就给兽兽发了消息问他要不要一起来做这个然后参赛

说实话在做这个之前，我心里是一点底都没有，兽兽也是一样的，我们俩认识都没几天。而且一个在重庆一个在京郊，这距离这么远，协同工作也给我们俩很大的挑战。

但是上苍似乎就是这么刻意安排我们两个人走到了一起，兽兽是个很强的人，他有很多值得我学的地方，特别是他做什么事情都是稳扎稳打的，在没有搞明白之前一般都不会动手。而我就不行了，当然我也不贬低自己了，这是我们俩共同的努力。

10.2 创意诞生

其实电子宠物这个并不能说是很新鲜的创意，只是从技术层面，特别是 Widget 比较有新意一些。

一开始这个 eoePet 只是一个想法，我们俩当时都没有想好具体该做点什么，大概就是第四期特刊第一版出之前的前一两天，我们俩都还对 Widget 报有无限的幻想。所以想了很多点子。

最后我们还是确定做宠物，因为相比之下其他点子都复杂一些，其实后来也证明了，在当前的条件下，基本上不太可能用 Widget 做出安歇东西。

然后就是确定逻辑，毕竟是因为远程协同工作，为了更好地交流，我们用 Google 的协作平台。之前订立了一切的规则，然后首先是我确定了整体结构和逻辑，我真没想过最后的代码能有如此的复杂。一开始我们俩就开始分工，我负责后台的 Service 逻辑，然后兽兽主要是动画，一个懵童的梦想就这么开始了。

10.3 十几天的开发

后台的逻辑很快就完成了，中间对于我的困那大多数是在于 Java 语言的不熟悉。但是兽兽那边简直就是水深火热。看过这期特刊的朋友都知道了，因为 Widget 的限制，是没办法用 animation 类来调用动画效果的。这个我们在 17、8 号的时候才发现了这个严重的问题。

而这距离 IT168 原定的比赛日期还有 10 天的时间了。

不过这没有难倒兽兽，可动画效果比现象中的还要坎坷，因为当时对很多东西还都不了解，所以犯了很多方向性的错误。比如说，我们把大量的代码写到了 Appwidget 面了，这里麻烦就打了，只要 Update 已更新，所有的代码都得重新执行。这太没灵活性了，后来使用了内部 Service，才解决了问题。但是动画效果至今还没有一个最完美的解决方案。

其他的事情也都不简单，就离远的那个的日子还有四五天的时候，我和兽兽出现了一个巨大的分歧，这个分歧可以说是非常严重的，其实都是我之前没有讲明白，没有给兽兽说清楚当时计划中的动画效果准确是什么样的

。

如果再做的朋友将来要带团队的，一定要注意，你想的，你说的，可能你很清楚，但是听着可能会因为不熟悉你的习惯而造成误解。即便是团队的成员，在交流的时候也要注意这些。

因为这个，我和兽兽那天的聊天记录激增。不过后来达成了统一，但是最后在方法的使用上，我还是败给了兽兽。其实还是我们的距离太远了，如果在一起，一些代码就可以共同完成，这么长的距离，在加上一些不熟悉，没办法两个人同时写一个代码，为了保证代码顺利，只能让兽兽一个人独自取承受了。

我们两个人赶的日子都不怎么好。都是我们两个人快考试的时候，还有那几天重庆和北京的天气也够让人受的了，记得有一天特别热，那种热是我最受不了的干热，全身上下都特难受。重庆更是全国的四大火炉之一，那些天的天气也是酷热难耐，兽兽他们宿舍的电扇坏掉了更加剧了这个问题了。

10.4 突破阻截

最后到了大概是 26 号基本上整体的代码算是最终完成了，当然还有很多细节的地方。

就当我们紧锣密鼓准备提交的时候，发现了一个噩耗。比赛延期了，而且时间还不短。这个时候特别矛盾的心理。特别是这一期特刊的发布，如果按照原来的比赛计划，我觉得我们铁定赢了，既然参加比赛，肯定也是抱着必胜的心理的，Widget 也不是很复杂的东西，最关键的是没有人共享出开发资料，有了这期第二版的特刊能让很多朋友避免很多弯路。同时我也相信 20 天时间里会有更好的创意出现。

不过最后还是想开了，比赛那都是虚荣，如果别人正有比我们更好的创意，那人家就强，没办法，愿赌服输。而且如果能有人通过我们共享的资料制作了更好的 Widget，这也是我们的福气。

另外，比赛毕竟没有结束，我们自己也有时间，而且亲身经历了这么多，我们比别人本来就占有很大的优势，所以给自己打气，我们一定能行的，Yes we can！

这个还是个 beta 版本，很多东西都还有待加强。29 号晚上在 Market 发布，但是没过几个小时就迎来了小范围的下载。而且一个朋友还专门发来邮件问这个 eoePet 应该怎么玩。

看到那封邮件兴奋的啊！

后来调整了一下，可能 Market 里面很多都是 app，很少有人下载 Widget。

30 号下午的时候，已经有了 500 多次的访问，最关键的 Active Install 达到了 50%的高位。这个着实让我兴奋了不少。

有一个朋友还说，我们能不能做一个 Ketty 猫 或者口袋妖怪的 水精灵的 pet

这些更加激励我和兽兽加紧完成下一版的制作了。

10.5 感谢

当然除了这个还要感谢 eoe 的其他的朋友，海阳哥、ice、放开那姑娘、pcr 等等都为帮助了我们不少。

特别是帮我做图片的两个同学 许仙 和 阿里郎·fox，没有他们，现在什么都没有。

最感谢的还是支持我们的广大朋友了。

10.6 这是 EoePet 完成后的 一些总结

(1) 做开发的过程中，一定要做好文档的管理，并不是为了形式，而是为了让开发者能了解进度、当前的问题，还有为以后出错提供排查的依据

(2) 如果你是项目的负责人，一定要保证你自己对整个项目有个非常清晰的认识。起码别人问起来一定要能回答得非常好

(3) 选择一个像兽兽或者我这样的合作伙伴(哈哈自夸一下)。最首要的一点是你的合作伙伴，敢，并且愿意和你一同进退，并且同样激情满满！

你们可以在不同的地方有差别，能力的不同还可以互补。

(4) 永远不要放弃你的信念！

15. 编后语

By : IceskYsl

在本期之前，eoeAndroid 社区成功组织策划，并实施了三个专题特刊，分别是《第一期：Android 1.5 SDK, Release 1 中文版》，《第二期: 图像处理篇（1）》和《第三期：Android Market 及应用发布》，得到社区的大力支持，本期本着推广 Android，服务开发者的理念，我们再次策划、组织并成功实施，本期的主题更加前沿，Android widget 是个很新的东西，国内资料严重匮乏，仅有的几篇资料不是相互山寨，就是去 copy 国外的只言片语，问题没将清楚不说，还严重误导了很多的初学者用户，实在悲哀。
本期特刊，我们从 widget 的整体予以把握和理解，按照难易程度，由浅入深的介绍了 widget 的背景、设计、不同程度的实现，最后并对哪些 widget 是可以被人接受卖钱的也做了一定的探讨；内容组织方面，我们翻译了 2 篇 SDK 中的经典文档，翻译了 2 篇老外写的教程文档，自己撰写了几篇实际的例子，非常详细，让读者可以从整体上感知 widget，并可以轻松的实现。
本其专题得到 eoeAndroid 社区很多朋友的大力支持，不仅翻译了高质量的文档，还纂写了非常贴近大家生活的例子，最后游利卡同学还参与到最后特刊的校核和编排，在这个了非常感谢这些付出劳动的朋友们。

最后提一下 Android widget 的不足，1.5 版本的 SDK 中包含的这个 widget 功能还是比较弱的，有些东西使用起来还不是很晚上，比如暂时还不支持修改界面的尺寸等，有的时候还会出现一些莫名其妙的问题，诸如 Unable to launch app com.eoeanrdroid.widget.facebook/10083 for broadcast Intent { action=android.appwidget.action.APPWIDGET_UPDATE comp={com.eoeanrdroid.widget.facebook/com.eoeanrdroid.widget.facebook.WordWidget} (has extras) }: process is bad 等，但是这些无法遮掩其美好的前景，我们相信随着 SDK 的不断完善，在这个套框架上可以提供更多更棒的功能，这仅仅是时间问题。
最后说一句，“番外篇”是游利卡同学为了表达对社区朋友的关爱，提醒大家夏天要避暑特意搜罗来的一些凉菜，以帮助大家避暑，同时特给了我一个不砍掉这些内容的理由是：希望有人能按照其思路打造一款夏日消暑的 widget。

第二版编后语

by 游利卡

终于做完了，明天一定得睡个懒觉！

GDC 终于忍受不住了，再也不允许我们增加文档的内容了，因为这次更新以后，新的内容实在是太多了！没办法，只要放到本地的 Word 里编辑。当然这些，也是源于我们十几天对 Widget 上不断探索的结果。

我和兽兽都笑谈，我们研究了这么多，也许我们真能算是全中国对 Android AppWidget 最了解的人了。当然这是开玩笑了，天外有天、人外有人。不过期间 Widget 也让我们挺窝火的，因为太多的方法都被限制在了 Widget 的大门之外。
不过这次做完了，算是一个阶段，我和兽兽仍将继续我们的开发，如果有什么值得介绍给大家的，一定毫无保留，赠送出来。
另外也希望大家能在我们的比赛中对我们予以支持。

谢谢了！

希望这些内容也能给大家带来一些帮助。

16. 其他

BUG 提交

如果你发现文档中翻译不妥的地方，请到如下地址反馈，我们会定期更新、发布更新后的版本
http://www.eoeandroid.com/viewthread.php?tid=753

资源下载：

本期文章中包含的源码请在如下地址下载：
http://www.eoeandroid.com/viewthread.php?tid=753

参加翻译

如果你有兴趣参加我们的特刊，请参考如下链接
http://www.eoeandroid.com/forumdisplay.php?fid=39

关于 eoeAndroid

当前，3G 商业，传统互联网与移动互联网也呈现出全业务发展的融合趋势，电信与互联网行业已经踏入继单机计算时代、传统互联网时代之后的第三个纪元。
由于看好移动互联网和 Android 手机平台的商业前景，同时也拥有专业而独特的产品、技术服务能力，我们聚集了一群热爱 Android 的技术英才，组建了 eoeMobile 团队。
eoeMobile 是一支专注于 Android 平台应用开发、产品运营和相关商业与技术服务的团队，立志于建立中国最大的 Android 应用开发专业社区 eoeAndroid.com ，想为 Android 在中国的发展尽自己的微薄之力。

17.广告

eoePet beta v1.0 找不到斑点的斑点狗 已经发布了，请大家支持一下我们的作品

地址：http://www.eoeandroid.com/viewthread.php?tid=1009&extra=page%3D1

IT 168 参赛地址: http://www.mobpub.net/viewthread.php?tid=3414&extra=page%3D1

Market 搜索： Loving Dog Widget (eoePet)

下载地址:

http://www.rayfile.com/zh-cn/files/c42ee2a8-64e5-11de-bab5-0014221b798a/

http://cid-5456d4b8f95608a9.skydrive.live.com/self.aspx/Android/eoePet/EoePetPlay%20beta%20v1.0.rar