2021

第10章 Widget组件开发

复旦大学 陈辰

学习目标

AIMS

01 了解Widget的概念及特征

02 掌握Widget的设计原则和开发步骤

03 了解Widget的调试方法

04 掌握使用Activity配置Widget的方法

05 掌握使用Service更新Widget 的方法

10.1 Widget简介

- **Widget**
  - Widget是一个具有特定功能的视图，一般被嵌入到主屏幕（Home screen）中，用户在不启动任何程序的前提下，就可以在主屏幕上直接浏览Widget所显示的信息
  - Widget在主屏幕上显示自定义的界面布局，在后台周期性的更新数据信息，并根据这些更新的数据修改主屏幕的显示内容
  - Widget可以有效的利用手机的屏幕，快捷、方便的浏览信息，为用户带来良好的交互体验

10.1 Widget简介

- **Widget**
  - Widget是Android 1.5引入的新特性，发展到Android 4.0已经有很大的进步和改变，例如在Android 3.1引入的更改Widget尺寸功能，以及Android 4.0增加的自动设置边界功能
  - Widget在主屏幕上可以出现多个相同的副本，也可以根据用户的设置，产生尺寸、布局、刷新速率和更新逻辑完全不同的副本
  - 将Widget程序设计成多个界面风格的版本，有助于适应不同用户的喜好

10.1 Widget简介

- **Widget**
  - 各种Widget

10.1 Widget简介

- **Widget**
  - 目前，在Android智能手机和平板电脑上具有非常广泛的应用，包括用Widget实现的微博客、RSS订阅器、股市信息、天气预报、日历、时钟、信息提醒、电量显示、邮件、便签、音乐播放、相册和新闻等
  - 在Android 4.0系统中，自带了多个Widget程序，包括时钟、书签、音乐播放器、相框和搜索栏等，如下图所示
  - 在Widget列表中可以查看所有的Widget组件，通过长时间点击Widget组件，可以将Widget组件添加到主屏幕上

10.1 Widget简介

- **Widget**
  - Android 4.0中的Widget

10.2 Widget基础

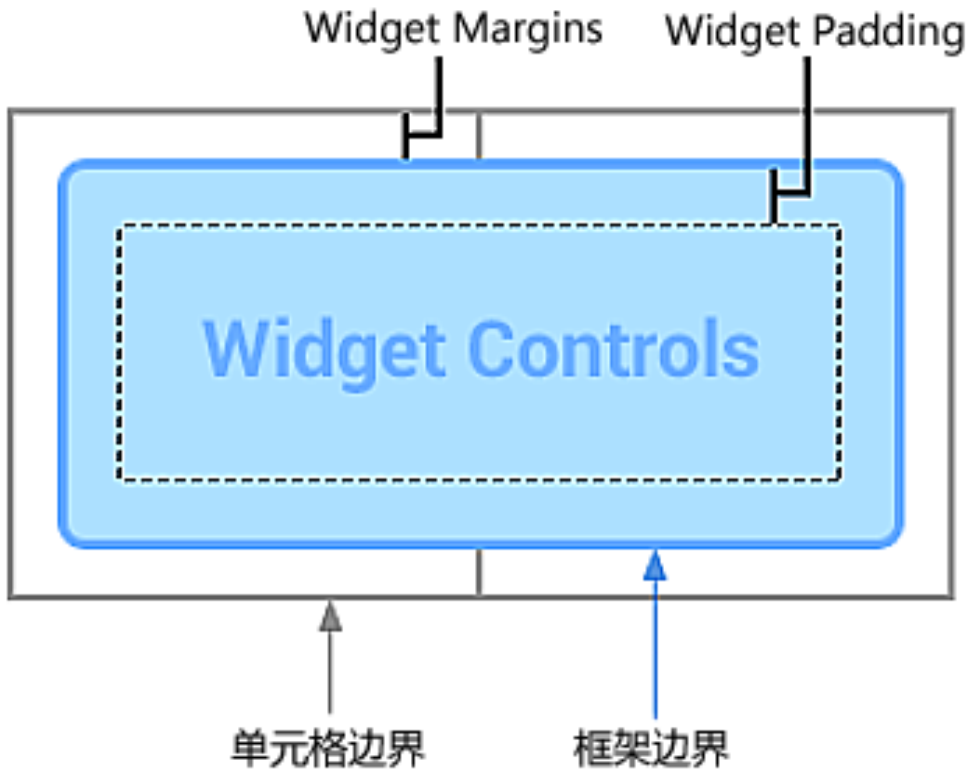
- **Widget基础**
  - Widget的设计原则：介绍Widget界面布局的设计要求
  - 开发步骤：以SimpleWidget为例
  - 调试过程 ：介绍Widget的安装、加载和删除方法

10.2 Widget基础

- **10.2.1 设计原则**
  - Widget是主屏幕上的显示元素，不仅自身具有一定的设计规则，还要与主屏幕上其它的元素保持美观一致
  - Widget显示在主屏幕上的结构如下图所示
  - 最外层是单元格边界，这个边界是不同Widget的分隔界限，在界面上这个界限对用户是不可见的
  - 框架边界是Widget背景图像的界限，背景图形会填充满整个框架（Frame）。最里面是Widget Controls，这是显示Widget界面元素的空间

10.2 Widget基础

- **10.2.1 设计原则**
  - Widget构成



10.2 Widget基础

- **10.2.1 设计原则**
  - Widget Padding是框架边界与Widget Controls之间的距离，可将Widget的界面元素显示在背景图片的中间区域
  - 为了保证多个Widget显示时不会靠的太近，一般都会设定Widget Margins，这个值是单元格边界与框架边界的距离
  - 如果Widget Margins的值为0，则两个Widget就会连在一起
  - 在Android 4.0中，系统会自动在添加Margins，保持两个Widget可以保持一定的间隔距离

10.2 Widget基础

- **10.2.1 设计原则**
  - 笔者建议使用这个新功能，方法是只要将AnroidManifest.xml文件中的targetSdkVersion设置为14
  - 下面介绍如何设计出同时适应Android 4.0以及较早Android系统的Widget界面布局
  - 使之在较早的Android系统上具有自定义的Widget Margins值，而在Android 4.0上保持相同的显示方式，而不会因为Android 4.0自动添加边界间隔而出现显示不一致的情况

10.2 Widget基础

- **10.2.1 设计原则**
  - 具体方法如下
    - 首先，将AnroidManifest.xml文件中的targetSdkVersion设置为14
    - 第二步建立布局文件，引用dimension资源，布局文件如下：

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="@dimen/widget_margin">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
          android:background="@drawable/widget_background">
     </LinearLayout>
</FrameLayout>
```

10.2 Widget基础

- **10.2.1 设计原则**
  - 第三步
    - 建立两个dimension资源，第1个在/res/values目录下，为较早的Android系统提供自定义的Margins；第2个在/res/values-v14目录下，为Android 4.0系统设定Margins

res/values/dimens.xml:

```
<dimen name="widget_margin">15dp</dimen>
```

res/values-v14/dimens.xml:

```
<dimen name="widget_margin">0dp</dimen>
```

10.2 Widget基础

- **10.2.1 设计原则**
  - Android系统将主屏幕划分为单元格，单元格的大小和数量会随设备的变化而完全不同，一般智能手机会被划分为4×4的单元格，而平板电脑一般会被划分为8×7的单元格
  - 当用户将Widget加入到主屏幕时，Widget会占据一定数量的单元格，占据单元格的数量由minWidth和minHeight决定，这两个属性是缺省情况下Widget的显示尺寸，具体的计算方法可以查询下表
  - 其中，dp表示与设备无关的像素，计算公式中之所以要减去30，是为了防止像素计算时的整数舍入导致错误

10.2 Widget基础

• 10.2.1 设计原则

- Widget尺寸与单元格数量的对应关系

Widget尺寸（minWidth和minHeight）	**单元格数量**
40dp	1
110dp	2
180dp	3
250dp	4
……	……
70*n-30	n

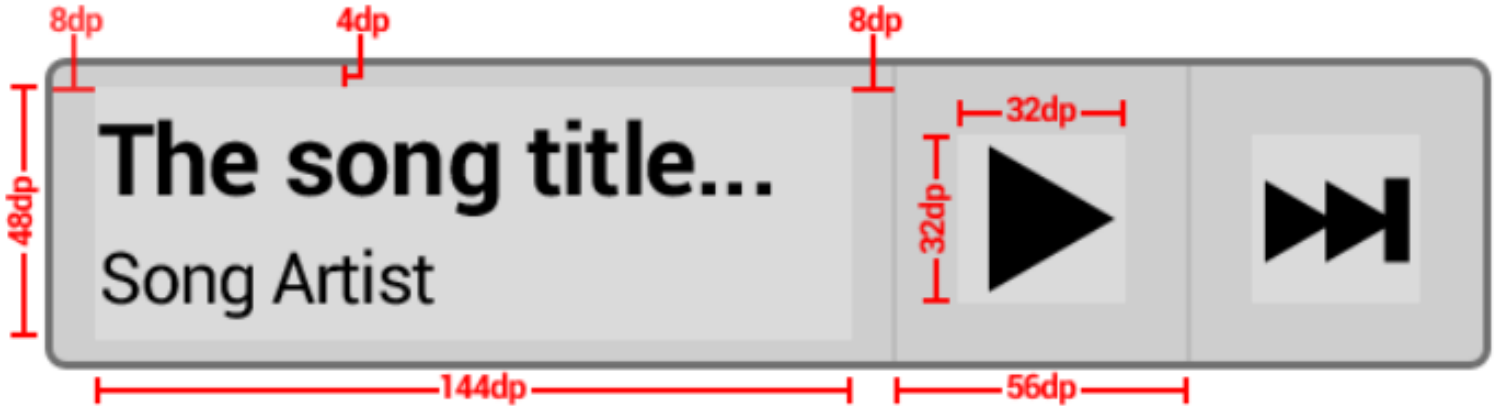
10.2 Widget基础

- **10.2.1 设计原则**
  - 在设定minWidth和minHeight时，最基本的原则是使Widget处于最佳的显示状态
  - 下面以“音乐播放器”为例说明如何计算Widget的minWidth和minHeight值
  - 音乐播放器的界面如下图所示：

10.2 Widget基础

• 10.2.1 设计原则

- 音乐播放器由一个显示歌曲信息的TextView和两个控制音乐播放的按钮组成
- 音乐播放器的界面元素尺寸如下图所示：



10.2 Widget基础

- **10.2.1 设计原则**
  - minWidth应等于三个控件的宽度和，加上控件之间的空隙，minHeight应等于TextView控件的高度加上边界空隙
  - 具体的计算方法可以参考下面的公式：

minWidth = 144dp + (2 × 8dp) + (2 × 56dp) = 272dp

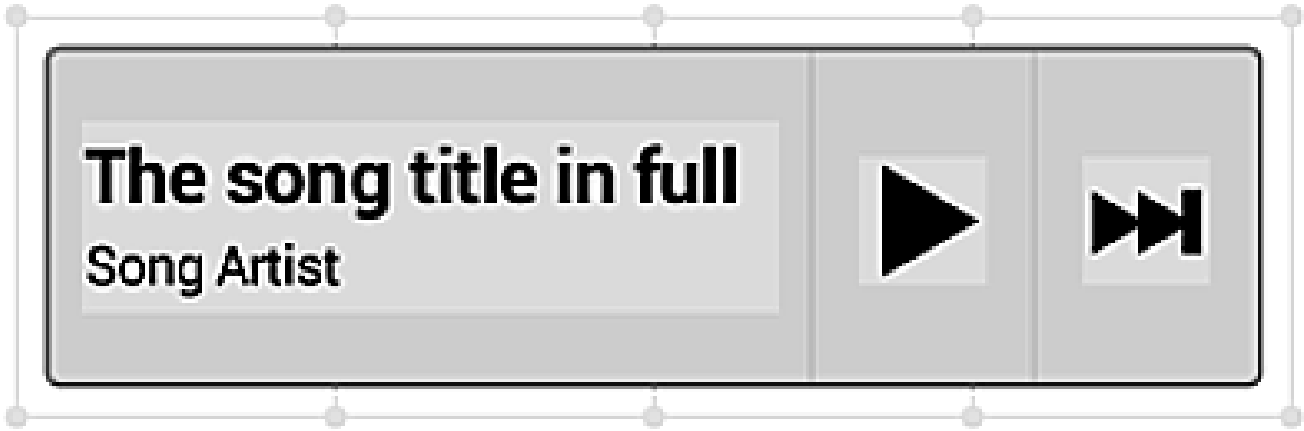
minHeight = 48dp + (2 × 4dp) = 56dp

10.2 Widget基础

- **10.2.1 设计原则**
  - 为了增加Widget对不同屏幕尺寸和单元格尺寸的适应性，建议尽量使用具有自适应能力的布局，例如线性布局、相对布局或框架布局
  - 在设计界面元素时，将不可改变尺寸的界面元素的高度和宽度设置成固定值，而让尺寸可改变的界面元素填充全部剩余空间
  - 应保证所有界面元素在纵向上居中显示

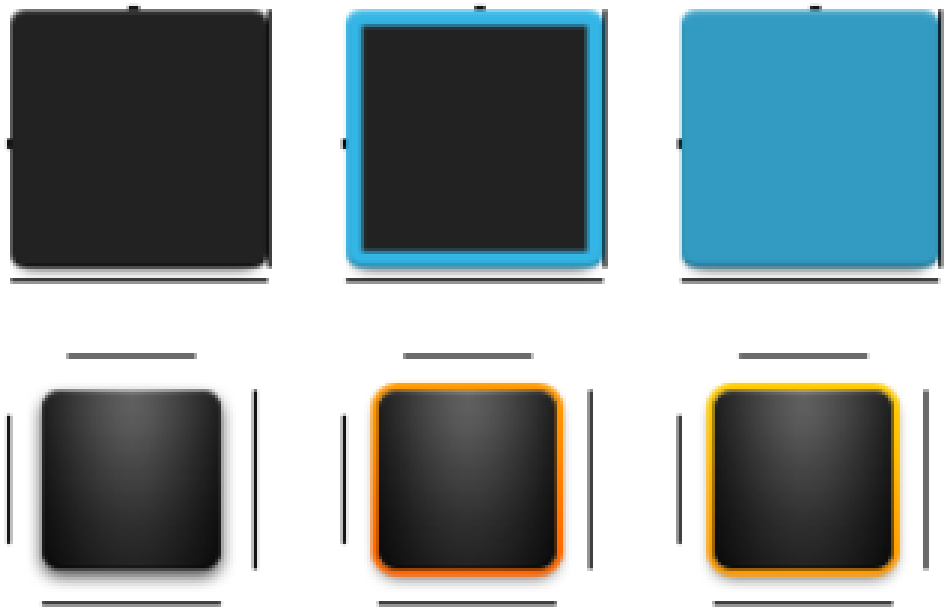
10.2 Widget基础

- **10.2.1 设计原则**
  - 当Widget的尺寸不够填充满所应占的单元格时，Widget会在横向和纵向拉伸，以填充所有应该占据的单元格
  - 下图是音乐播放器在单元格尺寸为80dp×100dp，Margins为16的显示效果



10.2 Widget基础

- **10.2.1 设计原则**
  - 建议读者使用9-patche文件作为背景图像，文件扩展名为.9.png。这种图像文件可以自动填充整个背景空间，同时不会影响界面的美观



10.2 Widget基础

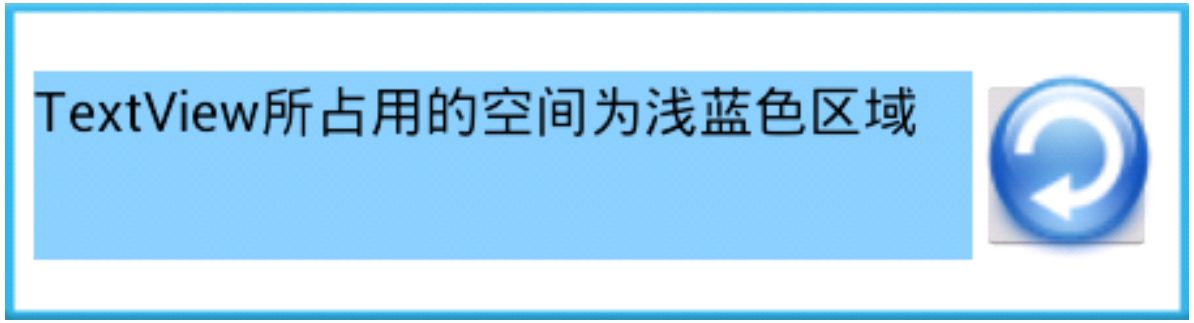
- **10.2.1 设计原则**
  - Widget模板包的下载
    - 模板包里面包括NinePatch图像文件、XML文件和Photoshop源文件等内容，适用于不同屏幕分辨率和Android 版本系统
    - 下载地址为：**http://developer.android.com/shareables/app_widget_templates-v4.0.zip**

10.2 Widget基础

- **10.2.2 开发步骤**
  - Widget的一般开发步骤如下：
    - 设计Widget的布局
    - 定义Widget的元数据
    - 实现Widget的添加、删除、更新
    - 在AnroidManifest.xml文件中声明Widget
  - 以SimpleWidget为例，介绍Widget的开发步骤，以及Widget框架类中个函数的调用顺序

10.2 Widget基础

- **10.2.2 开发步骤**
  - 设计Widget的布局
    - 创建用户Widget的第一步是设计并实现Widget的组件布局，就是Widget和用户交互的界面
    - SimpleWidget示例设计目标如下图所示，背景使用NinePatch的PNG图片，内部为白色背景，具有浅蓝色的边框
    - Widget内部包含TextView和ImageButton控件，使用线性水平布局



10.2 Widget基础

- **10.2.2 开发步骤**
  - 设计Widget的布局
    - Widget与Activity的布局设计和实现方法上十分相似，都是在/res/layout目录中建立基于XML的布局资源文件
    - SimpleWidget示例建立的Widget布局文件的文件名为widget_layout.xml，将Widget背景图片放置在/res/drawable 目录中，文件名为widget_background.9.png
    - widget_layout.xml的完整代码 如下：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:orientation="horizontal"
```

10.2 Widget基础

- **10.2.2 开发步骤**

```
6       android:background="@drawable/widget_background"
7       android:padding="8dp">
8
9       <TextView  android:id = "@+id/label"
10          android:layout_width="wrap_content"
11          android:layout_height="48dp"
12          android:text="TextView所占用的空间为浅蓝色区域"
13          android:textColor="@color/black"
14          android:background="@color/lightskyblue"
15          android:layout_weight="1"
16          android:layout_gravity="center_vertical"/>
17
18        <ImageButton
19          android:id="@+id/image_button"
20          android:layout_width="48dp"
```

10.2 Widget基础

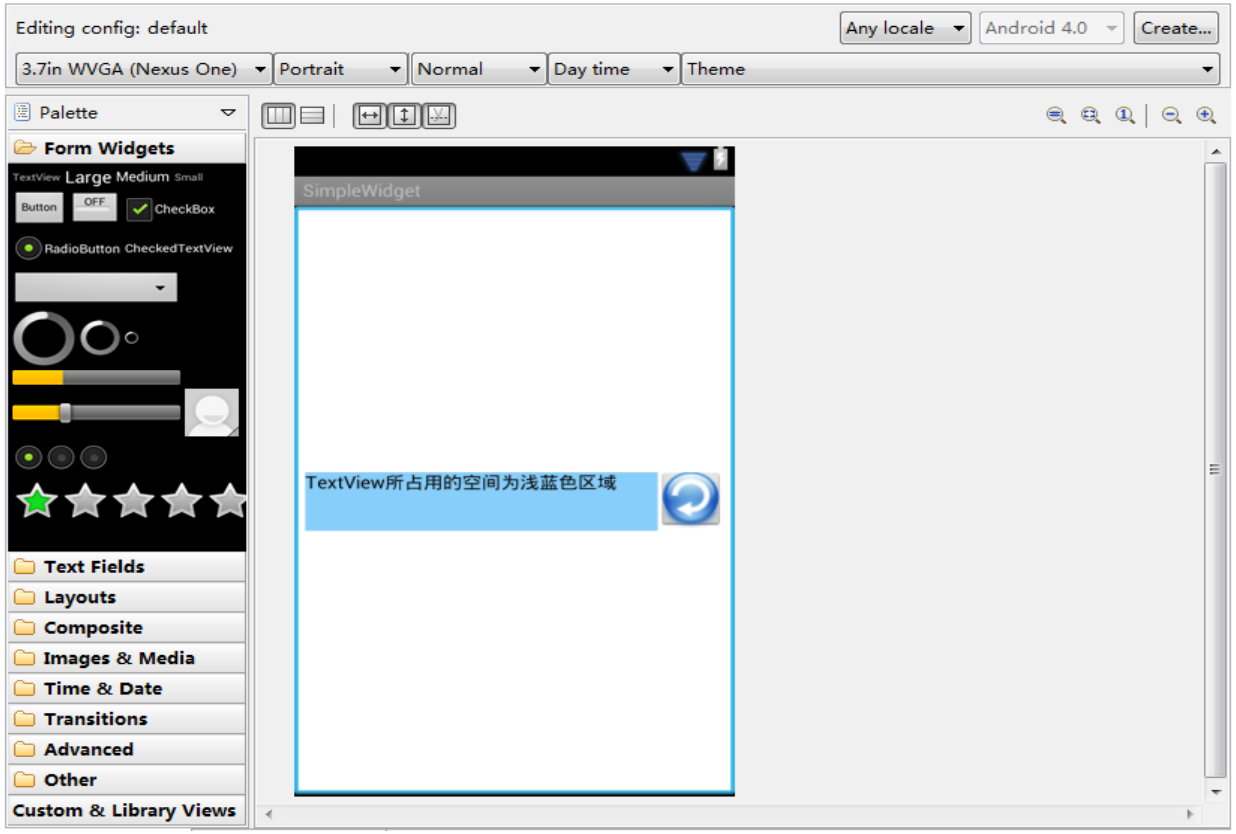
• 10.2.2 开发步骤

```
21          android:layout_height="48dp"
22          android:src="@drawable/button_image"
23          android:layout_gravity="center_vertical"/>
24
25    </LinearLayout>
```

- 第13行将TextView的字体颜色设置为黑色，第14行将TextView的背景颜色设置为浅蓝色，主要用来确定TextView所占据的区域范围
- 第15行将layout_weight设为1，而没有在ImageButton中设置这个参数，表明TextView控件会占据父节点所拥有的剩余空间
- 在Android Studio的界面控制器中，Widget的显示效果与设计目标略有区别，主要原因是线性布局的layout_width和layout_height属性都被设置成fill_parent

10.2 Widget基础

- **10.2.2 开发步骤**
  - 界面设计器中显示效果



10.2 Widget基础

- **10.2.2 开发步骤**
  - 出于Widget的安全和性能考虑，Widget支持的布局和控件存在一些限制
  - 目前Widget支持的布局有框架布局、线性布局和相关布局
  - 支持的界面控件有AnalogClock 、Button、Chronometer、ImageButton、ImageView、ProgressBar、TextView、ViewFlipper、ListView、GridView、StatckView和AdapterViewFlipper

10.2 Widget基础

- **10.2.2 开发步骤**
  - 定义Widget的元数据
    - Widget元数据定义了Widget最基本的信息，包括Widget的尺寸、更新周期、布局文件位置、预览图片、拉伸方向和配置界面等
    - SimpleWidget示例Widget元数据的文件保存在/res/xml/widget_template.xml，该文件的完整代码如下：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <appwidget-provider
3    xmlns:android="http://schemas.android.com/apk/res/android"
4    android:minWidth="150dp"
5    android:minHeight="60dp"
6    android:resizeMode="horizontal|vertical"
7    android:minResizeHeight="80dp"
8    android:minResizeWidth="48dp"
```

10.2 Widget基础

• 10.2.2 开发步骤

- 定义Widget的元数据

```
9     android:updatePeriodMillis="36000"
10     android:initialLayout="@layout/widget_layout"
11     android:previewImage="@drawable/preview"
12   />
```

  - 第2行使用appwidget-provider标签声明了Widget的元数据
  - 第4行和第5行定义了Widget的两个关键属性
  - minWidth和minHeight分别表示缺省情况下Widget的显示宽度和高度，也就是Widget在拖拽到主屏幕时的尺寸
  - Android 3.1后的系统支持改变Widget的显示尺寸，代码第6行声明Widget的尺寸可变，horizontal|vertical表示在水平和垂直方向上的大小都是可以变化的

10.2 Widget基础

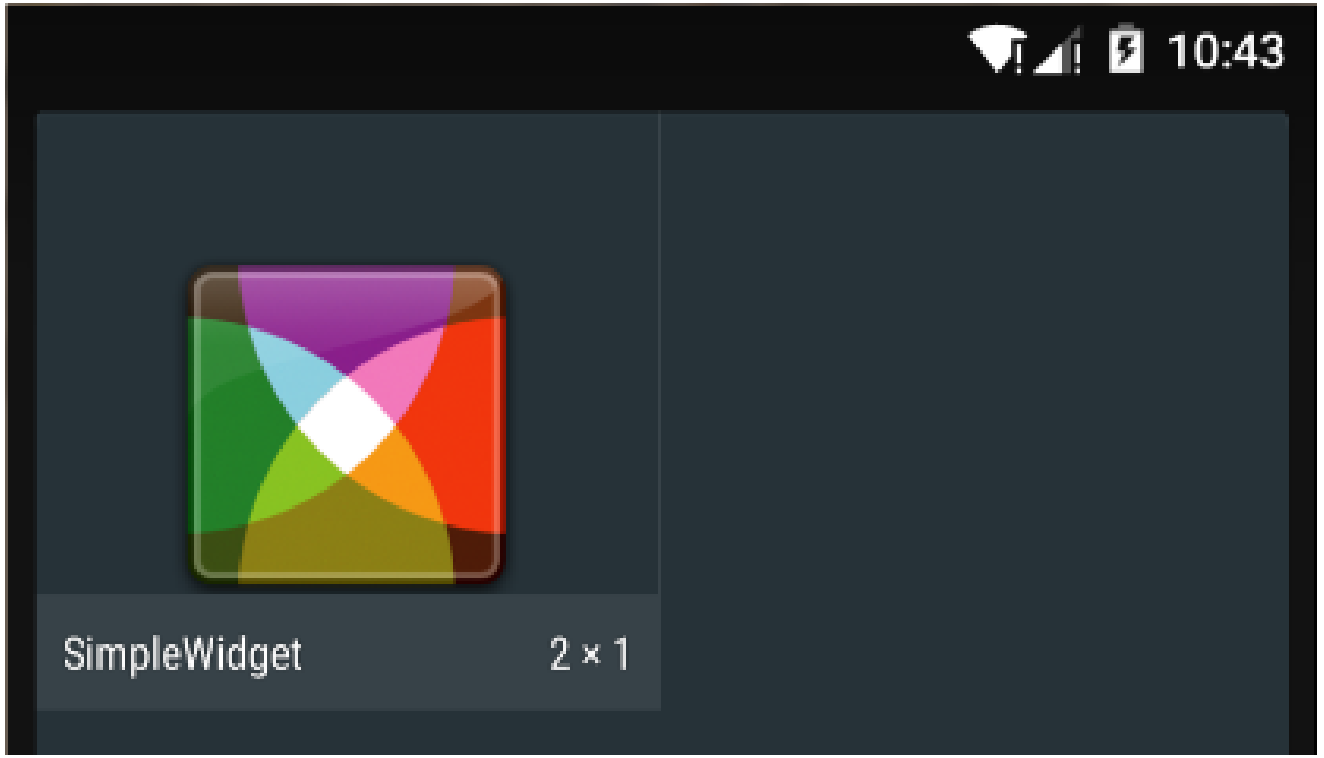
- **10.2.2 开发步骤**
  - 定义Widget的元数据
    - 其中，不可调整、水平方向调整、垂直方向调整、水平与垂直方向调整，这四种方式的参数分别为none、horizontal、vertical、horizontal|vertical
    - 第7行和第8行中，Widget的最小尺寸由minResizeWidth和minResizeHeight决定
    - minResizeHeight是Widget能够重新设置的最小高度，此值在大于minHeight时，或resizeMode中不支持垂直（vertical）拖拽时，此属性不起作用
    - minResizeWidth是Widget能够重新设置的最小宽度，此值在超过minWidth时，或者resizeMode不支持水平（horizontal）拖拽时，此属性不起作用

10.2 Widget基础

- **10.2.2 开发步骤**
  - 定义Widget的元数据
    - 第9行的updatePeriodMillis表示以毫秒为单位的更新周期，Android会以这个速率唤醒设备以便更新Widget，开发人员应尽可能的降低设备被唤醒的次数，以降低设备的能量消耗
    - 当更新周期小于30分钟时，Android系统并不按照此参数更新Widget，如果需要频繁更新Widget，可以在Service服务中实现
    - 第10行的initialLayout用来指定Widget的布局
    - 第11行的previewImage定义了在Android系统Widget列表中预览图像，如果不设置该值，则以程序的图标作为预览图像
    - SimpleWidget示例的预览图像如下图所示：

10.2 Widget基础

- **10.2.2 开发步骤**
  - 定义Widget的元数据
    - SimpleWidget示例的预览图像



10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新
    - 实现Widget的添加、删除、更新等过程，主要是通过AppWidgetProvider类来实现
    - 这个类本身继承BroadcastReceiver，用来接收与Widget相关的更新、删除、生效和失效等消息，当AppWidgetProvider接收这些消息后，会分别调用响应的事件处理函数，如下表所示：

10.2 Widget基础

• 10.2.2 开发步骤

- 实现Widget的添加、删除、更新

事件	调用函数	说明
ACTION_APPWIDGET_UPDATE	onUpdate()	Widget更新
ACTION_APPWIDGET_DELETED	onDelete()	Widget删除
ACTION_APPWIDGET_ENABLED	onEnabled()	Widget生效
ACTION_APPWIDGET_DISABLED	onDisabled()	Widget失效

10.2 Widget基础

• 10.2.2 开发步骤

- 实现Widget的添加、删除、更新
  - 在SimpleWidget示例中，WidgetProvider继承AppWidgetProvider类，在Widget更新、删除等操作过程中调用其内部的函数。WidgetProvider.java文件的完整代码如下：

```
1   package edu.hrbeu.SimpleWidget;
2
3   import android.appwidget.AppWidgetManager;
4   import android.appwidget.AppWidgetProvider;
5   import android.content.Context;
6   import android.util.Log;
7
8   public class WidgetProvider extends AppWidgetProvider {
9       private static final String TAG = "WIDGET";
```

10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新

```
11      @Override
12      public void onUpdate(Context context, AppWidgetManager
        appWidgetManager, int[] appWidgetIds) {
13          Log.d(TAG, "onUpdate");
14      }
15
16      @Override
17      public void onDeleted(Context context, int[] appWidgetIds) {
18          Log.d(TAG, "onDeleted");
19      }
20
21      @Override
22      public void onEnabled(Context context) {
```

10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新

```
23          Log.d(TAG, "onEnabled");
24      }
25
26      @Override
27      public void onDisabled(Context context) {
28          Log.d(TAG, "onDisabled");
29      }
30  }
```

10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新
    - 代码中虽然重载了的onUpdate()、onDelete()、onEnabled()和onDisabled()四个函数，但仅在函数中设置了调试信息，后期可以利用调试信息观察这些函数何时会被调用
    - onUpdate(Context, AppWidgetManager, int[])函数在updatePeriodMillis定义时间间隔到期时被调用，主要用来更新Widget组件的界面显示
    - 除此以外，在用户每次将Widget拖拽到主屏幕时，该函数也会被调用，可在此函数中为界面元素定义按钮点击事件处理函数，或者启动一个临时的Service进行数据获取等

10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新
    - onDeleted(Context context, int[] appWidgetIds)函数是当一个AWidget从主屏幕上被删除时调用的函数，用来回收资源
    - onEnabled(Context context)函数在首个Widget实例被创建并添加到主屏幕时被调用
    - Widget可以在主屏幕上创建多个实例，但只有在第一个Widget实例被创建时才调用该函数
    - onEnabled()一般用来进行一些初始化工作，比如打开一个新的数据库，或者执行对所有Widget实例来说只需进行一次的设置

10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新
    - onDisabled(Context context)函数在最后一个Widget实例被删除时调用，用来释放在onEnabled()中使用的资源，如删除在onEnabled()函数中创建临时数据库
    - 将Widget添加到主屏幕上，或者从主屏幕删除Widget都会引发AppWidgetProvider中的事件处理函数
    - 以SimpleWidget示例，通过观察Eclipse中LogCat的输出信息，分析用户对Widget进行不同操作所引发的事件处理函数，以及其调用顺序关系

10.2 Widget基础

- **10.2.2 开发步骤**
  - 实现Widget的添加、删除、更新
    - 当Widget第一次添加到主屏幕时，系统会按顺序调用onEnable()和onUpdate()
    - 当再次向主屏幕添加Widget时，系统则仅调用onUpdate()
    - 当从主屏幕删除Widget时，如果主屏幕还有这个Widget的实例，则系统仅调用onDelete()
    - 如果被删除的是这个Widget的最后一个实例，则系统在调用onDelete()后会调用onDisable()

10.2 Widget基础

• 10.2.2 开发步骤

- 在AnroidManifest.xml文件中声明Widget
  - 要让Widget生效还需在AnroidManifest.xml文件中进行声明，主要在该文件中声明AppWidgetProvider类。AnroidManifest.xml的完整代码如下：

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="edu.hrbeu.SimpleWidget"
4       android:versionCode="1"
5       android:versionName="1.0" >
6
7       <uses-sdk android:minSdkVersion="14" />
8
9       <application
```

10.2 Widget基础

- **10.2.2 开发步骤**
  - 在AnroidManifest.xml文件中声明Widget

```
10          android:icon="@drawable/ic_launcher"
11          android:label="@string/app_name" >
12           <receiver android:name=".WidgetProvider">
13              <meta-data android:name="android.appwidget.provider"
14                    android:resource="@xml/widget_template" />
15              <intent-filter>
16                 <action
      android:name="android.appwidget.action.APPWIDGET_UPDATE" />
17              </intent-filter>
18          </receiver>
19       </application>
20   </manifest>
```

10.2 Widget基础

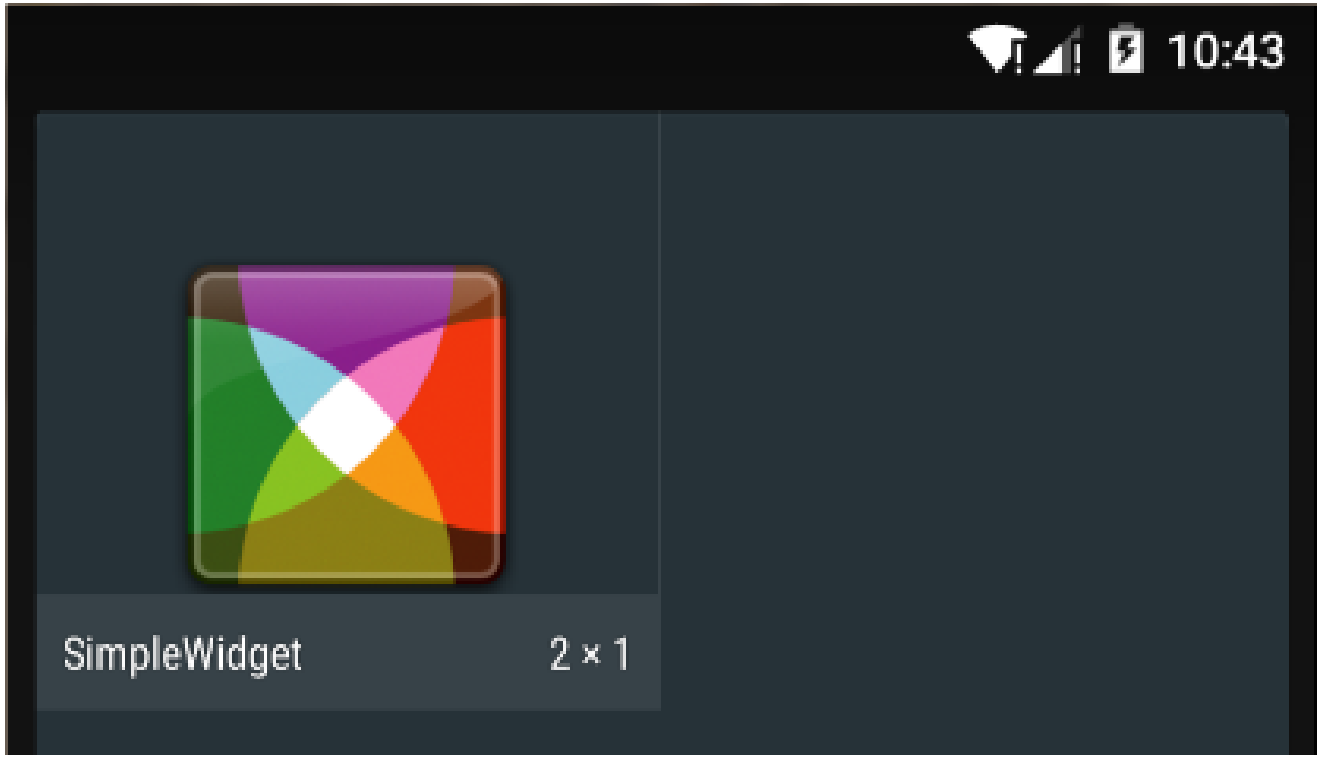
- **10.2.2 开发步骤**
  - 在AnroidManifest.xml文件中声明Widget
    - 第12行声明了receiver标签，android:name属性定义了AppWidgetProvider的子类
    - 第13行meta-data标签中的android:name属性，使用android.appwidget.provider表示这里的数据是Widget的元数据
    - 第14行的android:resource属性声明了元数据的资源路径
    - 第15行定义了intent-filter标签，代码第16行声明接收ACTION_APPWIDGET_UPDATE 消息

10.2 Widget基础

- **10.2.3 调试过程**
  - 在完成SimpleWidget示例的所有代码后，进入Widget的调试过程
  - 在进行Widget调试前，首先介绍如何安装、加载和删除Widget组件
  - 安装Widget与安装其它程序相似，是通过Android Studio上的运行（Run）按钮启动程序的编译、链接、打包和安装过程，唯一区别是在Widget安装到模拟器后，不会直接出现在主屏幕上，而需要用户在Android系统的Widget列表中手动将Widget添加到主屏幕上。Android系统的Widget列表如下图所示：

10.2 Widget基础

- **10.2.3 调试过程**
  - Android系统的Widget列表



10.2 Widget基础

- **10.2.3 调试过程**
  - 用户通过长时间（超过2秒）点击SimpleWidget的预览图标，将SimpleWidget实例加载到主屏幕上，缺省情况下占据2×1个单元格，如下图（a）所示
  - 在主屏幕上，通过长时间点击SimpleWidget实例，可以进入调整Widget尺寸状态，如下图（b）所示，Widget边缘出现四个实心菱形，通过拖拽这些实心菱形，可以调整Widget的尺寸。SimpleWidget实例在下图（b）中占据了4×2个单元格

10.2 Widget基础

- **10.2.3 调试过程**
  - SimpleWidget示例效果
    - 左：图（a）初始尺寸　　右：图（b）拉伸效果

10.2 Widget基础

- **10.2.3 调试过程**
  - 如果希望添加第二个SimpleWidget实例，过程与添加第一个SimpleWidget实例的过程完全一样
  - 在希望删除Widget时，同样是通过长时间点击主屏幕上的Widget实例，主屏幕上方会会出现垃圾桶，直接将Widget实例拖到垃圾桶即可
  - 需要注意的是主屏幕上的垃圾桶是隐藏的，需要通过长时间点击Widget示例才会出现
  - 当Widget实例在垃圾桶上方呈现出红色时，松开手指便可完成了删除操作

10.3 Widget配置

- 在Widget的使用过程中，有时用户需要根据个人喜好设置Widget的不同特征，如Widget的外观风格、字体颜色、字体大小、更新时间或背景图案等
- 比较普遍的做法是在Widget添加到主屏幕时，启动一个用于配置Widget的Activity，用户在这个Activity中设定Widget的特征
- 配置Widget特征的Activity，需要在Widget元数据XML文件中进行声明，声明的属性为android:configure，其值为Activity所在的类，示例代码如下

10.3 Widget配置

Widget元数据XML文件中的声明代码

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <appwidget-provider
3    xmlns:android="http://schemas.android.com/apk/res/android"
4    …… ……
5    android:configure="edu.hrbeu.ConfigWidget.ConfigActivity"
6  />
```

- 代码第5行中，Activity使用了带命名空间（edu.hrbeu.ConfigWidget）的声明方式，这是因为调用Activity的Widget宿主与Activity并不在相同的命名空间中
- 元数据中声明的Activity，在每个Widget实例被添加到主屏幕前会被启动
- 当用户完成配置选择关闭Activity，Widget才会出现在主屏幕上

10.3 Widget配置

- **用户配置Widget的Activity也需要在AndroidManifest.xml文件中声明**
- **不同于声明普通的Activity，这种Activity是被Widget的宿主通过发送android.appwidget.action.APPWIDGET_CONFIGURE动作（Action）启动的，所以此Activity需要接收Intent消息，示例代码如下：**

```
1  <activity android:name=".ConfigActivity">
2      <intent-filter>
3          <action
     android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
4      </intent-filter>
5  </activity>
```

10.3 Widget配置

- 当用户使用Activity完成Widget的配置后，Activity有责任调用相应代码对Widget进行更新
- Activity可以直接调用AppWidgetManager类更新Widget，也可也调用开发人员在AppWidgetProvider中编写的静态更新函数，实现Widget的更新
- AppWidgetManger是负责管理Widget的类，向AppWidgetProvider发送通知

10.3 Widget配置

- **要实现使用Activity配置Widget特征，并在适当的时候更新Widget，可以参考如下步骤**
  - 获取Widget的ID
    - Widget的宿主在启动Activity时，将Widget的ID保存在Intent中，通过调用extras.getInt()函数，获取Widget的ID
    - extras.getInt(String key, int defaultValue)函数中，参数1是获取数据的关键字，应使用关键字appWidgetId，或AppWidgetManager.EXTRA_APPWIDGET_ID
    - 参数2是无法获取数据时函数返回的代替数据，示例代码如下：

```
1   Intent intent = getIntent();
2   Bundle extras = intent.getExtras();
3   if (extras != null) {
4         mAppWidgetId = extras.getInt( AppWidgetManager.EXTRA_APPWIDGET_ID,
      AppWidgetManager.INVALID_APPWIDGET_ID);
5   }
6   if (mAppWidgetId == AppWidgetManager.INVALID_APPWIDGET_ID) {
7         finish();
8   }
```

10.3 Widget配置

- 第4行的AppWidgetManager.INVALID_APPWIDGET_ID的值为0，表示没有获取到Widget的ID
- 第6行和第7行说明，在没有正确获取到Widget的ID时，可以立即关闭Activity，因为没有正确的ID，即使完成配置工作，也无法将配置信息正确传递回Widget

- 配置Widget
  - 这个过程用户会在界面上选择相应的配置方案和配置信息，并最终通过事件引发更新Widget过程，并关闭Activity
- 更新Widget
  - 在更新Widget时，首先通过调用getInstance(context)函数获取AppWidgetManager实例，然后建立一个RemoteViews，在这个RemoteViews上更改Widget的界面元素，最后调用updateAppWidget(int, views)函数完成Widget更新

10.3 Widget配置

- RemoteViews是可在其它进程中显示的视图类，提供对部分界面控件的最基本的操作。示例代码如下：

```
1 AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
2 RemoteViews views = new RemoteViews(context.getPackageName(),R.layout.widget_layout);
3 views.setTextColor(R.id.label,textColor);
4 appWidgetManager.updateAppWidget(appWidgetId, views);
```

  - 第2行的R.layout.widget_layout是Widget的布局
  - 第3行setTextColor()函数可以设置TextView控件的字体颜色，TextView控件的ID为R.id.label，textColor是代表颜色的Int型整数
  - 第4行的updateAppWidget()函数中，参数1是Widget的ID，参数2是刚建立的RemoteViews
- 设置返回信息，并关闭Activity
  - 通过调用setResult(int resultCode, Intent data)函数， 设置Activity的返回代码和返回数据
  - 返回代码应为RESULT_OK或RESULT_CANCELED。RESULT_OK表示Widget设置成功，Widget宿主会将Widget实例加载到主屏幕上

10.3 Widget配置

- 如果返回的是RESULT_CANCELED，Widget宿主则取消Widget实例的加载过程，Widget也不会出现在主屏幕上
- 返回数据应包含Widget的ID，并使用AppWidgetManager.EXTRA_APPWIDGET_ID作为关键字，示例代码如下

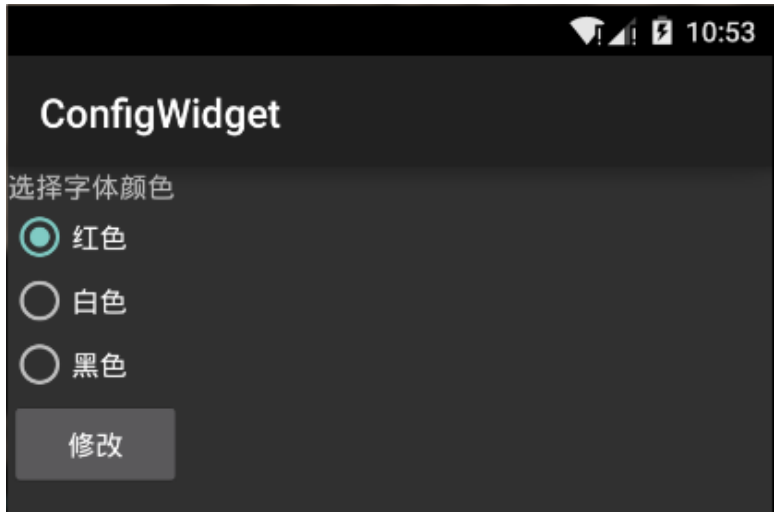
```
1  Intent resultValue = new Intent();
2  resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
3  setResult(RESULT_OK, resultValue);
4  finish();
```

- 需要注意的是，需要处理用户在未完成Widget配置前，通过回退键离开Activity的情况，方法非常简单，只有在Activity的onCreate()函数开始处，添加如下代码即可

```
1  public void onCreate(Bundle icicle) {
2          setResult(RESULT_CANCELED);
3          ......
4  }
```

10.3 Widget配置

- 在未正确完成Widget配置前，如果用户离开Activity配置界面，Activity的返回代码则是RESULT_CANCELED
- ConfigWidget示例中提供了完整的代码，说明如何在Activity中选择Widget中TextView的字体颜色
- ConfigWidget示例是在SimpleWidget示例代码的基础上进行的修改和添加，部分代码的理解可以参考SimpleWidget示例代码的说明
- ConfigWidget示例的Widget配置界面如下图所示:

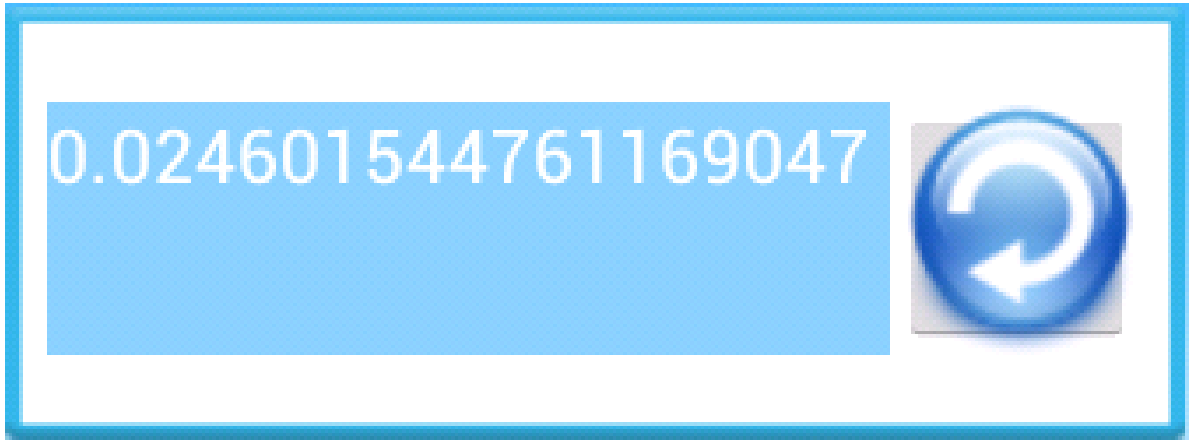


10.4 Widget与Service

- 在Widget中如果需要进行频繁更新，一般采用Service周期性更新Widget的方法
- Widget元数据中的updatePeriodMillis属性是无法进行频繁更新的，对于低于30分钟的设定值，该属性并不生效
- 当进行Widget更新时，如果在onUpdate()函数中代码运行时间超过5秒钟，例如进行网络操作、复杂运算等，则会产生应用程序无响应(ANR，Application Not Responding) 错误
- 使用Service更新Widget可以避免这种问题的出现，将比较耗时的代码在Service中实现，然后直接在Service中更新Widget的界面

10.4 Widget与Service

- 下面以ServiceWidget为例，说明如何使用Service更新Widget。ServiceWidget示例的用户界面如下图所示
- ServiceWidget示例在AppWidgetProvider中启动Service，当最后一个Widget实例在主屏幕上被删除时停止这个Service
- Service在启动后会开启一个工作线程，线程每2秒钟产生一个随机小数，并将这个随机小数显示在Widget的界面上



10.4 Widget与Service

- **Service的核心代码**

```
1   @Override
2       public void run() {
3           while(!Thread.interrupted()){
4               double randomDouble = Math.random();
5               String msg = String.valueOf(randomDouble);
6               WidgetProvider.updateAppWidget(this, msg);
7
8               try {
9                   Thread.sleep(2000);
10              } catch (InterruptedException e) {
11                  e.printStackTrace();
12              }
13          }
14      }
```

- 第6行调用了WidgetProvider中的静态函数updateAppWidget()，进行Widget界面更新

10.4 Widget与Service

- **WidgetProvider类继承AppWidgetProvider，其中的公有静态函数updateAppWidget()的代码如下：**

```
1   private static Queue<Integer> widgetIds=new LinkedList<Integer>();
2
3   public static void updateAppWidget(Context context, String displayMsg ) {
4      AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
5      RemoteViews views = newRemoteViews(context.getPackageName(),R.layout.widget_layout);
6      views.setTextViewText(R.id.label, displayMsg);
7
8      final int N = widgetIds.size();
9      for (int i=0; i<N; i++) {
10           int appWidgetId = widgetIds.poll();
11             appWidgetManager.updateAppWidget(appWidgetId, views);
12             widgetIds.add(appWidgetId);
13       }

14  }
```

10.4 Widget与Service

- updateAppWidget()函数每2秒被执行一次，负责所有Widget实例的更新
- 代码第1行定义了一个队列widgetIds，用于保存所有Widget实例的ID值
- 代码第8行获取Widget实例的数量，并在代码第11行实现Widget的更新操作
- 代码第10行和第12行分别实现队列数据的取出和加入，主要目的是为了遍历队列中所有Widget的ID值

- **更新所有Widget实例需要Widget的ID值，因此在WidgetProvider类onUpdate()函数中需将新建Widget的ID值添加到widgetIds队列中，并在onDeleted()函数中删除被移除Widget的ID值**

10.4 Widget与Service

- **WidgetProvider类onUpdate()函数的代码如下：**

```
1   @Override
2       public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
3           Log.d(TAG, "onUpdate");
4
5           for (int i = 0 ;i<appWidgetIds.length; i++ ){
6               widgetIds.add(appWidgetIds[i]);
7               Log.d(TAG," widgetId:" + appWidgetIds[i]+ ", Size:" + widgetIds.size());
8           }
9
10           Log.d(TAG, "appWidgetIds.length:" + appWidgetIds.length);
11           context.startService(new Intent(context, TRandomService.class));
12       }
```

10.4 Widget与Service

- 代码第11行调用startService()函数，启动TRandomService服务
- 虽然比较优雅的方法是在onEnable()函数中调用startService()函数启动服务，但在Widget实际运行过程中，偶然会出现服务没有启动，却不是首次添加Widget的情况
- 如果将启动服务的代码放在onEnable()函数中，此时将无法启动服务，Widget也无法进行更新

10.4 Widget与Service

- 在onUpdate()函数中启动服务，会导致服务被多次启动，如果不进行控制，服务会开启多个线程，频繁更新Widget
- 因此，TRandomService类声明一个布尔值threadRunning，表示是否已经有工作线程在运行，并在onStart()函数中进行判断
- **TRandomService类onStart()函数的代码如下：**

```
1   @Override
2       public void onStart(Intent intent, int startId) {
3           super.onStart(intent, startId);
4           Toast.makeText(this, "(2) 调用onStart() :" ,
        Toast.LENGTH_SHORT).show();
5           if (!threadRunning) {
6               threadRunning=true;
7               new Thread(this).start();
8           }
9       }
```

10.4 Widget与Service

- WidgetProvider类onDeleted ()函数负责将Widget的ID从widgetIds队列中删除，首先判断ID值是否在队列中，如果在则删除
- **WidgetProvider类onDeleted ()函数的代码如下**：

```
1   @Override
2       public void onDeleted(Context context, int[] appWidgetIds) {
3           Log.d(TAG, "onDeleted");
4           for (int i = 0; i< appWidgetIds.length ;i++){
5               if (widgetIds.contains(appWidgetIds[i])){
6                   widgetIds.remove((Object)appWidgetIds[i]);
7               }
8               Log.d(TAG," widgetIds:" + appWidgetIds[i]+ ", Size:" + widgetIds.size());
9           }
10           Log.d(TAG, "appWidgetIds.length:" + appWidgetIds.length);
11       }
```

10.4 Widget与Service

- 在最后一个Widget从主屏幕上被删除后，此时则没有必要让服务继续运行，因此在onDisabled ()函数中调用stopService()函数停止服务
- **WidgetProvider类onDisabled ()函数的代码如下：**

```
1    @Override
2     public void onDisabled(Context context) {
3         Log.d(TAG, "onDisabled");
4         context.stopService(new Intent(context, TRandomService.class));
5     }
```

习题：

1. 分析Widget的优势和不足。

2. 简述Widget的设计原则和注意事项。

3. 尝试开发显示电量信息或短信内容的Widget。

THANKS

谢　　谢　　观　　看