第10章 Widget组件开发

杨刚
中国人民大学

10.1 Widget简介

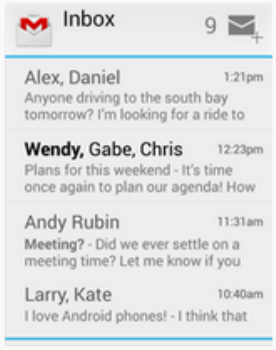
- Widget
  - Widget是一个具有特定功能的视图，一般被嵌入到主屏幕（Home screen）中，并接收周期性更新。用户可以在主屏幕上直接浏览Widget所显示的信息；
  - Widget可以有效的利用手机的屏幕，快捷、方便的浏览信息，为用户带来良好的交互体验
  - 如：时钟、音乐播放器、相框和Google搜索、天气预报、股市信息等

10.1 Widget简介

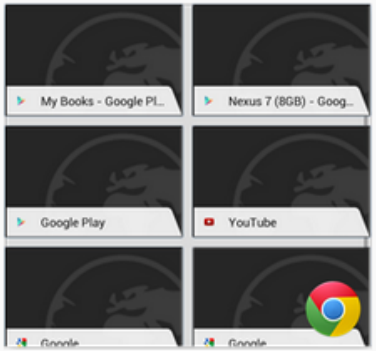
- Widget
  - Widget是Android 1.5引入的新特性，发展到Android 4.0已经有很大的进步和改变，例如在Android 3.1引入的更改Widget尺寸功能，以及Android 4.0增加的自动设置边界功能
  - Widget在主屏幕上可以出现多个相同的副本，也可以根据用户的设置，产生尺寸、布局、刷新速率和更新逻辑完全不同的副本
  - 将Widget程序设计成多个界面风格的版本，有助于适应不同用户的喜好

10.1 Widget简介

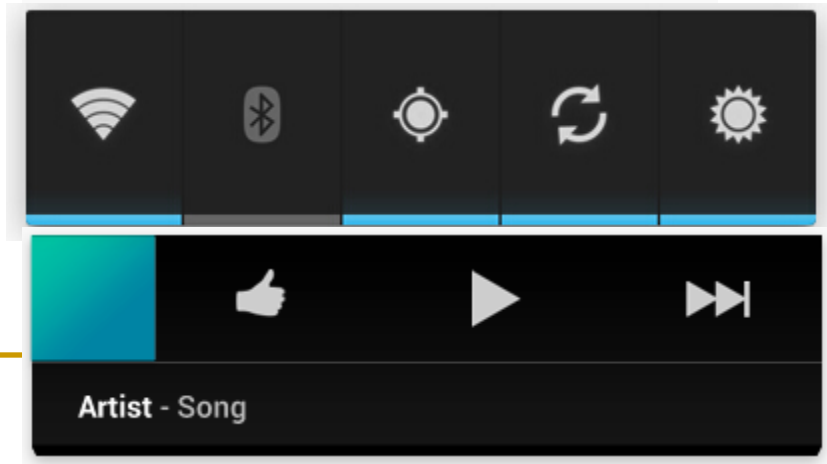
- Widget
  - Widget类型
    - Information widgets
    - Collection widgets
    - Control widgets
    - Hybrid widgets



ListView widget



GridView widget

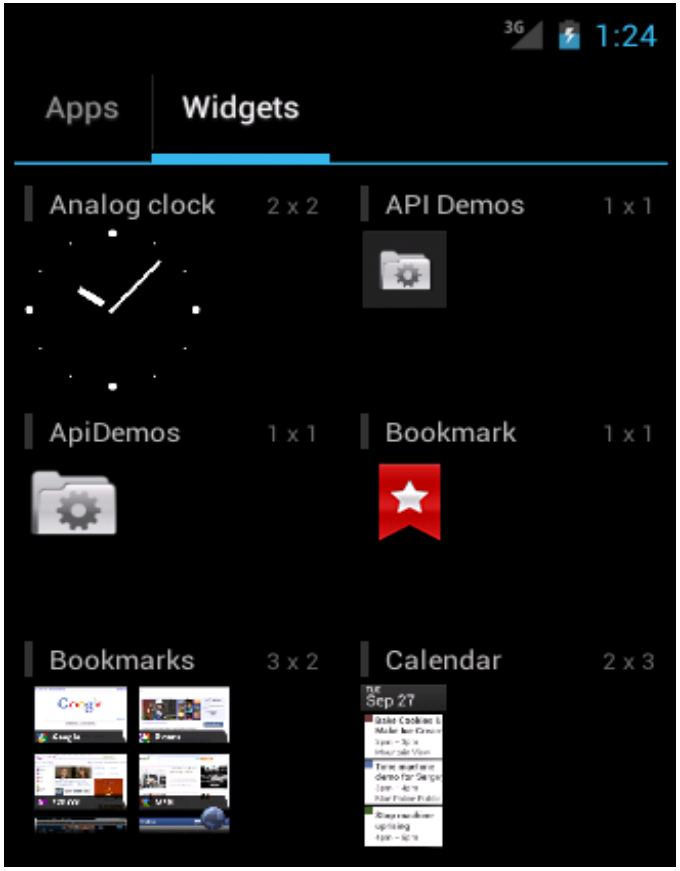


10.1 Widget简介

- 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

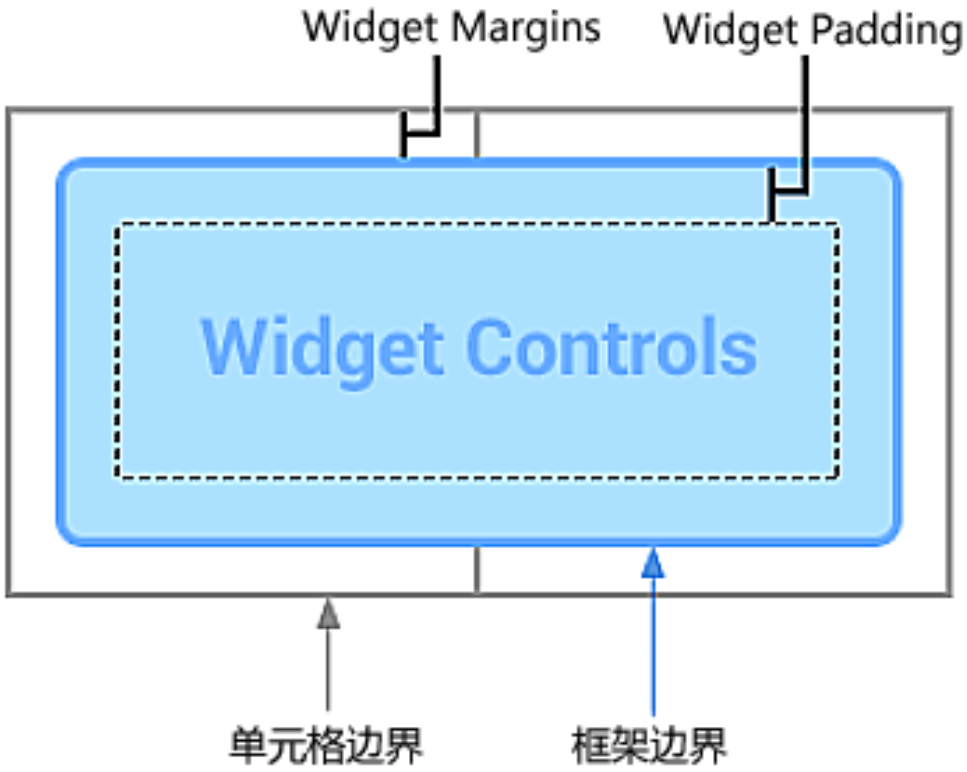


10.2 Widget基础

- **设计方法**
  - 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

- 设计方法
  - Widget构成



10.2 Widget基础

- **设计方法**
  - 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

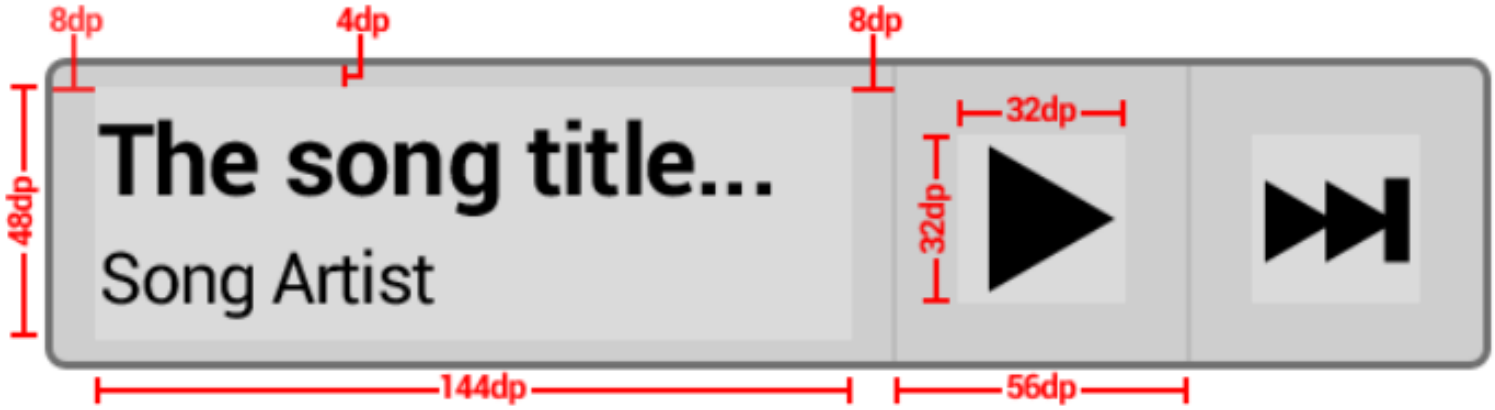
- **设计方法**
  - Android系统将主屏幕划分为单元格，单元格的大小和数量会随设备的变化而完全不同，一般智能手机会被划分为4×4的单元格，而平板电脑一般会被划分为8×7的单元格
  - 当用户将Widget加入到主屏幕时，Widget会占据一定数量的单元格，占据单元格的数量由minWidth和minHeight决定，这两个属性是缺省情况下Widget的显示尺寸

10.2 Widget基础

- **设计方法**
  - 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

- 设计方法
  - 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

- **设计方法**
  - 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

- **设计方法**
  - 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

- 设计方法
  - 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

- 设计方法
  - **AppWidget 框架类**
    - 1、AppWidgetProvider ： 继承自 BroadcastRecevier ，在AppWidget 应用 update、enable、disable 和 delete 时接收通知。其中，onUpdate、onReceive 是最常用到的方法，它们接收更新通知。
    - 2、 AppWidgetProvderInfo：描述 AppWidget 的大小、更新频率和初始界面等信息，以XML 文件形式存在于应用的 res/xml/目录下。
    - 3、AppWidgetManger ： 负责管理 AppWidget ， 向 AppwidgetProvider 发送通知。
    - 4、RemoteViews ： 一个可以在其他应用进程中运行的类，向 AppWidgetProvider 发送通知。

10.2 Widget基础

- 设计方法
  - **AppWidgetManger** 类

    bindAppWidgetId(int appWidgetId, ComponentName provider)

    通过给定的ComponentName 绑定appWidgetId

    getAppWidgetIds(ComponentName provider)

    通过给定的ComponentName 获取AppWidgetId

    getAppWidgetInfo(int appWidgetId)

    通过AppWidgetId 获取 AppWidget 信息

    getInstalledProviders()

    返回一个List<AppWidgetProviderInfo>的信息

10.2 Widget基础

- 设计方法
  - **AppWidgetManger** 类

    getInstance(Context context)

    获取 AppWidgetManger 实例使用的上下文对象

    updateAppWidget(int[] appWidgetIds, RemoteViews views)

    通过appWidgetId 对传进来的 RemoteView 进行修改，并重新刷新AppWidget 组件

    updateAppWidget(ComponentName provider, RemoteViews views)

    通过 ComponentName 对传进来的 RemoeteView 进行修改，并重新刷新AppWidget 组件

    updateAppWidget(int appWidgetId, RemoteViews views)

    通过appWidgetId 对传进来的 RemoteView 进行修改，并重新刷新AppWidget 组件

10.2 Widget基础

- 设计方法
  - 继承自 AppWidgetProvider 可实现的方法
    - onDeleted(Context context, int[] appWidgetIds)
    - onDisabled(Context context)
    - onEnabled(Context context)
    - onReceive(Context context, Intent intent)

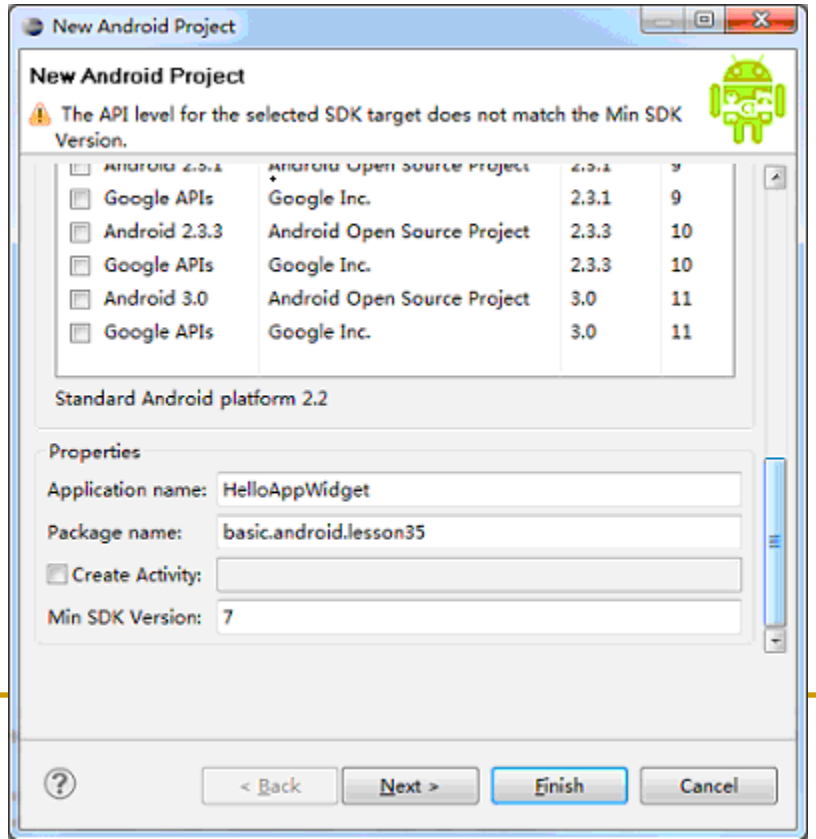
      因为 AppWidgetProvider 是继承自BroadcastReceiver 所以可以重写onRecevie 方法，
    - onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)

10.2 Widget基础

- **开发步骤**
  - Widget的一般开发步骤如下
    - 设计Widget的布局
    - 设计Widget内容提供者文件，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

10.2 Widget基础

- 开发步骤
  - 创建Widget工程
    - 注意创建时可以不选Create Activity。



10.2 Widget基础

- 开发步骤
  - 设计Widget的布局
    - 设计Widget的显示布局文件 layout/widget.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:layout_width="fill_parent“
    android:layout_height="fill_parent“
    android:orientation="vertical"
    android:gravity="center">
<textview
   android:layout_height="wrap_content"
   android:layout_width="wrap_content"
   android:id="@+id/textView1"
   android:text="欢迎进入App Widget的世界！"
   android:textcolor="#ff0000ff">
</textview></linearlayout>
```

10.2 Widget基础

- **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

- 开发步骤
  - 设计Widget内容提供者文件
    - 在res下建立xml/widget_provider.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <!-- appwidget-provider Widget的配置文件  -->
3. <!-- android:minWidth 最小宽度 -->
4. <!-- android:minHeight 最小高度 -->
5. <!-- android:updatePeriodMillis 组件更新频率（毫秒） -->
6. <!-- android:initialLayout 组件布局XML的位置 -->
7. <!-- android:configure Widget设置用Activity -->
8. <appwidget -provider="" xmlns:android="http://schemas.android.com/apk/res/android" android:initiallayout="@layout/widget" android:updateperiodmillis="86400000" android:minheight="72dp" android:minwidth="294dp">
9. </appwidget>
```

10.2 Widget基础

- 开发步骤
  - 从上面的代码可以看出，在设计中有以下几个重要参数（元数据）须加以注意：
    - 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，如果声明android:resizeMode="horizontal|vertical"，则说明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。可选的参数有none、horizontal、vertical、horizontal|vertical等四种
    - 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

10.2 Widget基础

- 开发步骤
  - 实现Widget的添加、删除、更新
    - 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onUpdate、onDelete、onEnabled和onDisabled 。
    - 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

- 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

- 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

- **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

- **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。

10.2 Widget基础

- 开发步骤
  - 在AnroidManifest.xml文件中声明Widget
    - 增加receiver标签，receiver的 android:name指向的是widget的请求处理器或者说请求接收者。Manifest文件中的部分代码

```
<application
   android:icon="@drawable/ic_launcher”
   android:label="@string/app_name" >
   <receiver android:name=".WidgetProvider">
   <meta-data android:name="android.appwidget.provider“
          <intent-filter>
              <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
          </intent-filter>
        </receiver>
     </application>
```

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();  }
```

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

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