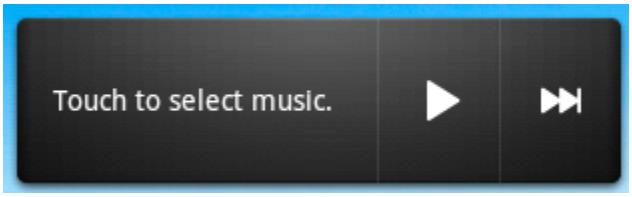
About Widget

App Widgets 是小型 views，可以嵌入到其他应用程序并且接收定期更新。这些 views 被称为 Widgets 组件，您可以通过 App Widget provider 去发布 Widgets，能够容纳其他 App Widgets 称为 App Widget host。



上图就是一个 music Widget

创建一个 Widget，你需要：

`AppWidgetProviderInfo object`

描述一个 App Widget，如 App Widget 的布局，更新频率，以及 AppWidgetProvider 类的 metadata。这在 XML 中定义。

`AppWidgetProvider` class implementation

在代码层次实现能与基于广播机制事件的 App Widget 进行交互的基本方法。通过它，当 App Widget 更新，启用，禁用和删除的时候就能收到广播。

View layout

通过 XML 定义的 App Widget 初始化布局。

另外，可以实现一个配置 App Widget 的 Activity 。 这个可选的 Activity 是当用户添加 App Widget 的时候打开，允许用户在创建时候去对 App Widget 配置。

下面介绍一下怎么去使用这些组件：

一、在 AndroidManifest.xml 中声明 App Widget

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

`<receiver>`需要 android:name 属性，它指定 App Widget 所使用的 AppWidgetProvider。

<intent-filter>需要含 android:name 属性的<action>，这个属性指定 AppWidgetProvider 接收 ACTION_APPWIDGET_UPDATE 广播，这是唯一一个需要明确声明的广播。当需要时，AppWidgetManager 自动给所有 App Widget 发送广播。

<meta-data>指定了 AppWidgetProviderInfo 资源，需要以下属性：

android:name - 指定 metadata 的名字。使用 android.appwidget.provider 将其作为 AppWidgetProviderInfo 的描述性数据。
android:resource - 指定 AppWidgetProviderInfo 资源的位置。

二、增加 AppWidgetProviderInfo Metadata

一个 App Widget 的基本属性是通过 AppWidgetProviderInfo 去定义的，例如它的最小尺寸的布局，它的初始 layout，多久更新 App Widget，还有（可选）在创建时期的一个配置 Activity。在 XML 资源中定义一个 AppWidgetProviderInfo 是通过<appwidget-provider>标签，并保存在工程的 res/xml/文件夹底下。例如

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

updatePeriodMillis 属性定义了 App Widget 应该多久向 AppWidgetProvider 请求更新。实际更新的时间未必能保证及时更新，并且建议尽量不要频繁更新---比如一小时一次去更新电量。也可以容许用户去自定义更新的时间。

configure 属性是当用户添加一个 App Widget 前启动的 Activity，这个 Activity 的作用就是配置 App Widget 的属性。(Optional)

三、为 App Widget 定义 Layout

只要你熟悉用 xml 怎么去定义 layout 的话，为 App Widget 定义一个 layout 还是很简单的。但是由于 App Widget 的布局是基于 RemoteViews，所以只能使用 RemoteViews 所支持的 layout 或者 view。

RemoteViews 支持的 layout 和 view 如下：

Layout – FrameLayout LinearLayout RelativeLayou

View -- AnalogClock Button Chronometer ImageButton ImageView ProgressBar TextView

注意：继承这些类的子类同样不支持。

四、使用 AppWidgetProvider

AppWidgetProvider 是 BroadcastReceiver 的子类，这个类处理 App Widget 广播。AppWidgetProvider 只接收于 App Widget 有关系的广播，比如 App Widget 在 updated, deleted, enabled, and disabled。当这些广播发生的时候，AppWidgetProvider 会调用一下回调方法：

`onUpdate(Context, AppWidgetManager, int[])`

间隔调用此方法去更新 App Widget，间隔时间的设置是在 AppWidgetProviderInfo 下的 updatePeriodMillis 属性，同样当用户添加 App Widget 的时候也被调用。如果你已经声明了一个 configuration Activity，用户添加 App Widget 的时候就不会调用 onUpdate，但是在随后的更新中依然会被调用。

`onDeleted(Context, int[])`

当 App Widget 从 App Widget host 中删除的时候调用。

`onEnabled(Context)`

当 App Widget 第一次创建的时候调用。比如，当用户增加两个同样的 App Widget 时候，这个方法只在第一次去调用。如果你需要打开一个新的数据库或者其他的设置，而这在所有的 App Widgets 只需要设置一次的情况下，这个是最好的地方去实现它们。

`onDisabled(Context)`

当 App Widget 的最后一个实例从 App Widget host 中被删除的时候调用。这里可以做一些在 `onEnabled(Context)` 中相反的操作，比如删除临时数据库。

`onReceive(Context, Intent)`

每一个广播的产生都会调用此方法，而且是在上面方法之前被调用。通常不需要实现此方法。

在 AppWidgetProvider 中最重要的 callback 就是 `onUpdated()`，如果你的 App Widget 接收用户交互事件，就需要在这个 callback 里面进行处理。

如果你需要一个带有 Button 的 App Widget，点击 Button 去启动一个 Activity，下面就是 AppWidgetProvider 的实现方法：

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {
public void onUpdate(Context context, AppWidgetManager
appWidgetManager, int[] appWidgetIds) {
final int N = appWidgetIds.length;
// Perform this loop procedure for each App Widget that belongs to this
provider
for (int i=0; i<N; i++) {
```

```
int appWidgetId = appWidgetIds[i];
// Create an Intent to launch ExampleActivity
Intent intent = new Intent(context, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
intent, 0);
// Get the layout for the App Widget and attach an on-click listener to
the button
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.appwidget_provider_layout);
views.setOnClickPendingIntent(R.id.button, pendingIntent);
// Tell the AppWidgetManager to perform an update on the current App
Widget
appWidgetManager.updateAppWidget(appWidgetId, views);
    }
  }
}
```

五、创建 App Widget Configuration Activity

这个 Activity 将通过 App Widget 自动启动，用户可以给 App Widget 设置有用的参数，比如 App Widget 的颜色、大小、更新时间或者其他的属性。

在 AndroidManifes.xml 中定义这个 Activity 和一般定义 Activity 基本没有区别，App Widget host 启动这个 Activity 需要一个 Action，所以：

```
<activity android:name=".ExampleAppWidgetConfigure">
 <intent-filter>
 <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE"
/>
</intent-filter>
</activity>
```

同样这个 Activity 必须在 AppWidgetProviderInfo XML 文件中定义 android:configure。

值得注意的是 App Widget host 调用 configuration Activity，configuration Activity 必须要返回一个结果（必须包含 App Widget ID）saved in the Intent extras as `EXTRA_APPWIDGET_ID`

六、通过 configuration Activity 去更新 App Widget

1. 获取 App Widget ID

```
Intent intent = getIntent();
```

```
Bundle extras = intent.getExtras();
if (extras != null) {
mAppWidgetId = extras.getInt(
AppWidgetManager.EXTRA_APPWIDGET_ID,
AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

2. 设置 App Widget 参数
3. 当设置完成，通过 `getInstance(Context)` 获取 AppWidgetManager 实例
   `AppWidgetManager appWidgetManager =AppWidgetManager.getInstance(context);`
4. 通过调用 `updateAppWidget(int, RemoteViews)` 去更新 App Widget

```
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.example_appwidget);
appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

5. 创建一个返回的 Intent，结束 Activity

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

By Vicki