文献翻译

Application Fundamentals

Android applications are written in the Java programming language. The compiled Java code — along with any data and resource files required by the application — is bundled by the aapt tool into an Android package, an archive file marked by an .apk suffix. This file is the vehicle for distributing the application and installing it on mobile devices; it's the file users download to their devices. All the code in a single .apk file is considered to be one application.

In many ways, each Android application lives in its own world:

1. By default, every application runs in its own Linux process. Android starts the process when any of the application's code needs to be executed, and shuts down the process when it's no longer needed and system resources are required by other applications.

2. Each process has its own virtual machine (VM), so application code runs in isolation from the code of all other applications.

3. By default, each application is assigned a unique Linux user ID. Permissions are set so that the application's files are visible only to that user and only to the application itself — although there are ways to export them to other applications as well.

It's possible to arrange for two applications to share the same user ID, in which case they will be able to see each other's files. To conserve system resources, applications with the same ID can also arrange to run in the same Linux process, sharing the same VM.

1.1 Application Components

A central feature of Android is that one application can make use of elements of other applications (provided those applications permit it). For example, if your application needs to display a scrolling list of images and another application has developed a suitable scroller and made it available to others, you can call upon that scroller to do the work, rather than develop your own. Your application doesn't incorporate the code of the other application or link to it. Rather, it simply starts up that piece of the other application when the need arises.

For this to work, the system must be able to start an application process when any part of it is needed, and instantiate the Java objects for that part. Therefore, unlike applications on most other systems, Android applications don't have a single entry point for everything in the application (no main() function, for example). Rather, they have essential components that the system can instantiate and run as needed. There are four types of components:

(1)Activities

An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one.

Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen.

The visual content of the window is provided by a hierarchy of views — objects derived from the base View class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions directed at that space. Thus, views are where the activity's interaction with the user takes place.

For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more.

A view hierarchy is placed within an activity's window by the Activity.setContentView() method. The content view is the View object at the root of the hierarchy. (See the separate User Interface document for more information on views and the hierarchy.)

(2)Services

A service doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the Service base class.

A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them. However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen. It's possible to connect to (bind to) an ongoing service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes. For the music service, this interface might allow users to pause, rewind, stop, and restart the playback.

Like activities and the other components, services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks (like music playback). See Processes and Threads, later.

(3)Broadcast receivers

A broadcast receiver is a component that does nothing but receive and react to broadcast announcements. Many broadcasts originate in system code — for example, announcements that the timezone has changed, that the battery is low, that a picture has been taken, or that the user changed a language preference. Applications can also initiate broadcasts — for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

An application can have any number of broadcast receivers to respond to any announcements it considers important. All receivers extend the BroadcastReceiver base class.

Broadcast receivers do not display a user interface. However, they may start an activity in response to the information they receive, or they may use the

NotificationManager to alert the user. Notifications can get the user's attention in various ways — flashing the backlight, vibrating the device, playing a sound, and so on. They typically place a persistent icon in the status bar, which users can open to get the message.

(4)Content providers

A content provider makes a specific set of the application's data available to other applications. The data can be stored in the file system, in an SQLite database, or in any other manner that makes sense. The content provider extends the ContentProvider base class to implement a standard set of methods that enable other applications to retrieve and store data of the type it controls. However, applications do not call these methods directly. Rather they use a ContentResolver object and call its methods instead. A ContentResolver can talk to any content provider; it cooperates with the provider to manage any interprocess communication that's involved.

See the separate Content Providers document for more information on using content providers.

Whenever there's a request that should be handled by a particular component, Android makes sure that the application process of the component is running, starting it if necessary, and that an appropriate instance of the component is available, creating the instance if necessary.

1.2 Activating components: intents

Content providers are activated when they're targeted by a request from a

ContentResolver. The other three components — activities, services, and broadcast

receivers — are activated by asynchronous messages called intents. An intent is an Intent object that holds the content of the message. For activities and services, it names the action being requested and specifies the URI of the data to act on, among other things. For example, it might convey a request for an activity to present an image to the user or let the user edit some text. For broadcast receivers, the

Intent object names the action being announced. For example, it might announce to interested parties that the camera button has been pressed.

There are separate methods for activating each type of component:

1. An activity is launched (or given something new to do) by passing an Intent object to Context.startActivity() or Activity.startActivityForResult(). The responding activity can look at the initial intent that caused it to be launched by calling its getIntent() method. Android calls the activity's onNewIntent() method to pass it any subsequent intents. One activity often starts the next one. If it expects a result back from the activity it's starting, it calls startActivityForResult() instead of startActivity(). For example, if it starts an activity that lets the user pick a photo, it might expect to be returned the chosen photo. The result is returned in an Intent object that's passed to the calling activity's onActivityResult() method.

2. A service is started (or new instructions are given to an ongoing service) by passing an Intent object to Context.startService(). Android calls the service's onStart() method and passes it the Intent object. Similarly, an intent can be passed to Context.bindService() to establish an ongoing connection between the calling component and a target service. The service receives the Intent object in an onBind() call. (If the service is not already running, bindService() can optionally start it.) For example, an activity might establish a connection with the music playback service mentioned earlier so that it can provide the user with the means (a user interface) for controlling the playback. The activity would call bindService() to set up that connection, and then call methods defined by the service to affect the playback.

A later section, Remote procedure calls, has more details about binding to a service.

3. An application can initiate a broadcast by passing an Intent object to methods like Context.sendBroadcast(), Context.sendOrderedBroadcast(), and Context.sendStickyBroadcast() in any of their variations.

Android delivers the intent to all interested broadcast receivers by calling their

onReceive() methods. For more on intent messages, see the separate article, Intents and Intent Filters.

1.3 Shutting down components

A content provider is active only while it's responding to a request from a ContentResolver. And a broadcast receiver is active only while it's responding to a broadcast message. So there's no need to explicitly shut down these components. Activities, on the other hand, provide the user interface. They're in a long-running conversation with the user and may remain active, even when idle, as long as the conversation continues. Similarly, services may also remain running for a long time. So Android has methods to shut down activities and services in an orderly way:

1. An activity can be shut down by calling its finish() method. One activity can shut down another activity (one it started with startActivityForResult()) by calling finishActivity().

2. A service can be stopped by calling its stopSelf() method, or by calling Context.stopService().

Components might also be shut down by the system when they are no longer being used or when Android must reclaim memory for more active components. A later section, Component Lifecycles, discusses this possibility and its ramifications in more detail.

1.4 The manifest file

Before Android can start an application component, it must learn that the component exists. Therefore, applications declare their components in a manifest file that's bundled into the Android package, the .apk file that also holds the application's code, files, and resources.

The manifest is a structured XML file and is always named AndroidManifest.xml for all applications. It does a number of things in addition to declaring the application's components, such as naming any libraries the application needs to be linked against (besides the default Android library) and identifying any permissions the application expects to be granted.

But the principal task of the manifest is to inform Android about the application's components. For example, an activity might be declared as follows:

The name attribute of the <activity> element names the Activity subclass that implements the activity. The icon and label attributes point to resource files containing an icon and label that can be displayed to users to represent the activity.

The other components are declared in a similar way — <service> elements for services, <receiver> elements for broadcast receivers, and <provider> elements for content providers. Activities, services, and content providers that are not declared in the manifest are not visible to the system and are consequently never run. However, broadcast receivers can either be declared in the manifest, or they can be created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling Context.registerReceiver().

For more on how to structure a manifest file for your application, see The Android

Manifest.xml File.

1.5 Intent filters

An Intent object can explicitly name a target component. If it does, Android finds that component (based on the declarations in the manifest file) and activates it. But if a target is not explicitly named, Android must locate the best component to respond to the intent. It does so by comparing the Intent object to the intent filters of potential targets. A component's intent filters inform Android of the kinds of intents the component is able to handle. Like other essential information about the component, they're declared in the manifest file. Here's an extension of the previous example that adds two intent filters to the activity:

The first filter in the example — the combination of the action

"android.intent.action.MAIN" and the category

"android.intent.category.LAUNCHER" — is a common one. It marks the activity as one that should be represented in the application launcher, the screen listing

applications users can launch on the device. In other words, the activity is the entry point for the application, the initial one users would see when they choose the application in the launcher.

The second filter declares an action that the activity can perform on a particular type of data.

A component can have any number of intent filters, each one declaring a different set of capabilities. If it doesn't have any filters, it can be activated only by intents that explicitly name the component as the target.

For a broadcast receiver that's created and registered in code, the intent filter is instantiated directly as an IntentFilter object. All other filters are set up in the manifest. For more on intent filters, see a separate document, Intents and Intent Filters.

1.6 XML-Based Layout

While it is technically possible to create and attach widgets to our activity purely through Java code, the way we did in Chapter 4, the more common approach is to use an XML-based layout file. Dynamic instantiation of widgets is reserved for more complicated scenarios, where the widgets are not known at compile-time (e g., populating a column of radio buttons based on data retrieved off the Internet).

With that in mind, it’s time to break out the XML and learn how to lay out Android activities that way.

As the name suggests, an XML-based layout is a specification of widgets’ relationships to each other—and to their containers (more on this in Chapter

7)—encoded in XML format. Specifically, Android considers XML-based layouts to be resources, and as such layout files are stored in the res/layout directory inside your Android project.

Each XML file contains a tree of elements specifying a layout of widgets and their containers that make up one view hierarchy. The attributes of the XML elements are properties, describing how a widget should look or how a container should behave. For example, if a Button element has an attribute value of android:textStyle = "bold", that means that the text appearing on the face of the button should be rendered in a boldface font style.

Android’s SDK ships with a tool (aapt) which uses the layouts. This tool should be automatically invoked by your Android tool chain (e.g., Eclipse, Ant’s build.xml). Of particular importance to you as a developer is that aapt generates the R.java source file within your project, allowing you to access layouts and widgets within those layouts directly from your Java code.

XML as a GUI definition format is becoming more commonplace. Microsoft’s XAML2, Adobe’s Flex3, and Mozilla’s XUL4 all take a similar approach to that of Android.

应用程序基础

Android应用程序使用Java编程语言开发。aapt工具把编译后的Java代码连同应用程序所需的其他数据和资源文件一起打包到一个Android包文件中，这个文件使用.apk作为扩展名。此文件是分发并安装应用程序到移动设备的载体；是用户下载到他们的设备的文件。单一.apk文件中的所有代码被认为是一个应用程序。

从多个角度来看，每个Android应用程序都存在于它自己的世界之中：

1 默认情况下，每个应用程序均运行于它自己的Linux进程中。当应用程序中的任何代码需要被执行时，Android启动此进程，而当不再需要此进程并且其它应用程序又请求系统资源时，则就关闭了这个进程。

2 每个进程都有其独有的虚拟机（VM），所以应用程序代码与所有其它应用程序代码是隔离运行的。

3 默认情况下，每个应用程序均被赋予一个唯一的Linux用户ID，并加以权限设置，使得应用程序的文件仅对此用户及此应用程序可见——尽管也有其它的方法使得这些文件同样能为其他应用程序所访问。

1.1 应用程序组件

Android的一个核心的特性就是一个应用程序可以使用其它应用程序的元素（如果那个应用程序允许的话）。例如，如果你的应用程序需要显示一个图片卷动列表，而另一个应用程序已经开发了一个合用的而又允许别的应用程序使用的话，你可以直接调用那个卷动列表来完成工作，而不用自己再开发一个。你的应用程序并没有吸纳或链接其它应用程序的代码。它只是在有需求的时候启动了其它应用程序的那个功能部分。

为达到这个目的，系统必须能够在一个应用程序的任何一部分被需要时启动一个此应用程序的进程，并将那个部分的Java对象实例化。因此，不像其它大多数系统上的应用程序，Android应用程序并没有为应用程序提供一个单独的入口点（比如说，没有main()函数），而是为系统提供了可以实例化和运行所需的必备组件。一共有四种组件类型：

(1)Activities

一个应用程序可以只有一个activity，或者，如刚才提到的短信应用程序那样，包含很多个。每个activity的作用，以及有多少个activity，当然是取决于应用程序及其设计的。一般情况下，总有一个应用程序被标记为用户在应用程序启动的时候第一个看到的。从一个activity转向另一个靠的是用当前的activity启动下一个。

每个activity都被给予一个默认的窗口以进行绘制。一般情况下，这个窗口是满屏的，但它也可以是一个小的位于其它窗口之上的浮动的窗口。一个activity也可以使用附加的窗口——例如，一个在activity运行过程中弹出的供用户响应的对话框，这是一个当用户选择了屏幕上特定项目后显示的必要信息的窗口。

窗口显示的可视内容是由一系列层次化view构成的，这些view均继承自 View 基类。每个view均控制着窗口中一块特定的矩形区域。父级view包含并组织其子view的布局。叶节点view（位于层次结构最底端）在它们控制的矩形区域中进行绘制，并对用户直达其区域的操作做出响应。因此，view是activity与用户进行交互的界面。例如，view可以显示一个小图片，并在用户指点它的时候产生动作。Android有一些预置的view供开发者使用——包括按钮、文本域、滚动条、菜单项、复选框等等。

view的层次结构是由Activity.setContentView() 方法放入activity的窗口之中的。content view是位于层次结构根位置的View对象。（参见独立的用户界面文档以获取关于view及层次结构的更多信息。）

(2) Service

service没有可视化的用户界面，而是在一段时间内在后台运行。例如，一个service可以在用户做其它事情的时候在后台播放背景音乐、从网络上获取数据或者计算一些东西并提供给需要这个运算结果的activity使用。每个service都继承自Service基类。

一个媒体播放器播放播放列表中的曲目是一个不错的例子。播放器应用程序可能有一个或多个activity来给用户选择歌曲并进行播放。然而，音乐播放这个任务本身丌应该由任何activity来处理，因为用户的期望即使在他们离开播放器的应用程序而已经在开始做别的事情时，音乐仍然在继续播放。为达到这个目的，媒体播放器activity可以启动一个运行于后台的service服务。系统将在这个activity不再显示于屏幕后，仍维持音乐播放的service的运行。

连接至（绑定到）一个正在运行的service（如果service没有运行，则启动之）是可能的。连接之后，你可以通过那个service暴露出来的接口不service进行通讯。对于音乐service来说，这个接口可以允许用户暂停、回退、停止以及重新开始播放。

如同activity和其它的组件一样，service服务运行于应用程序进程的主线程内。

所以它不会对其它组件或者用户界面有任何的妨碍作用，它们一般会派生一个新线程来执行一些时间消耗型任务（比如音乐回放和网络下载）。参见稍后的进程和线程介绍。

(3) BroadcastReceiver

broadcast receiver是一个与注于接收广播通知信息，并做出相应处理的组件。许多广播是由系统代码产生的——例如，通知时区改变、电池电量低、拍摄了一张照片或者用户改变了语言选项。应用程序也可以发起广播——例如，通知其它应用程序一些数据已经下载到设备上并处于可用状态。

一个应用程序可以拥有任意数量的broadcast receiver，以对所有它认为重要的通知信息予以各种响应。所有的receiver均继承自BroadcastReceiver基类。

broadcast receiver没有用户界面。然而，它们可以启动一个activity或者service来响应它们收到的信息，当然也可以使用NotificationManager来通知用户。通知可以用多种方式来吸引用户的注意力──闪动背光灯、震动设备、播放声音等等。通知一般是在状态栏上放一个持丽的图标，用户可以点击打开它并获取所要消息。

(4)Contentprovider

content provider将一些特定的应用程序数据供给其它应用程序使用处理。数据可以存储于文件系统、SQLite数据库或其它有意丿的方式。content provider继承于ContentProvider 基类，实现了一套使得其他应用程序能够检索和存储它所管理的类型数据的标准方法。然而，应用程序并不直接调用返些方法，而是使用一个

ContentResolver 对象，调用它的方法作为替代。ContentResolver可以与任何content provider进行会话；与其合作对任何相关的进程间通讯进行管理。

参阅独立的Content Providers文档以获得更多关于使用content provider的信息。

每当出现一个需要被特定组件处理的请求时，Android会确保那个组件的应用程序进程处于运行状态，必要时会启动它，并确保那个组件的一个合适的实例可用，必要时会创建那个实例。

1.2激活组件：intent

当接收到ContentResolver发出的请求后，content provider被激活。而其它三种组件——activity、service和broadcast receiver，被一种叫做intent的异步消息所激活。intent是一个保存着消息内容的Intent对象。对于activity和service来说，它指明了所请求的操作名称，并指定了用来操作的数据的URI和其它一些信息。例如，它可以承载一个对一个activity的请求，让它为用户显示一张图片，或者让用户编辑一些文本。而对于broadcast receiver来说，Intent对象指明了所通报的操作。例如，它可以对所有感兴趣的对象通报照相按钮被按下。

对于每种组件来说，激活的方法也是不同的：

1 通过传递一IntentContext.startActivity()Activity.startActivityForResult(以启动（或指定新工作给）另外一个activity。相应的activity可以通过调用自身的 getIntent() 方法来查看并且激活它的intent。Android通过调用activity的

onNewIntent()方法来传递给它随后的任何intent。

一个activity经常启动另一个activity。如果它期望它所启动的那个activity返回一个结果，它会调用startActivityForResult()而不是startActivity()。例如，如果它启动了另外一个activity以使用户挑选一张自己的照片，它也许想知道哪张照片被选中了。其结果将会被封装在一个Intent对象中，并传递给发出调用的activity的onActivityResult() 方法。

2 通过传递一个Intent对象至Context.startService()以启动一个service（或向正在运行的service给出一个新的指令）。Android调用此service的 onStart()方法并将Intent对象传递给它。

与此类似，一个intent可以被传递给 Context.bindService()以建立一个处于调用组件和目标service乀间的活动连接。此service会通过onBind() 方法的调用来获取此Intent对象（如果此service尚未运行，bindService()会先启动它）。例如，一个activity可以建立一个不前述的音乐回放service的连接，这样它就可以提供给用户一些途径（用户界面）来控制回放。这个activity可以调用 bindService()来建立此连接，然后调用service中定之的方法来控制回放。

稍后的远程方法调用一节有关于如何绑定至一个service的更多细节。

3 应用程序可以通过传递一个Intent对象至 Context.sendBroadcast() ，Context. sendOrderedBroadcast()， 以及Context.sendStickyBroadcast()和其它类似方法来发起一个广播。Android会调用所有对此广播有兴趣的broadcast receiver的 onReceive()方法，将此intent传递给它们。

1.3 关闭组件

content provider仅在响应来自ContentResolver的请求时处于不同活动状态。而broadcast receiver仅在响应一条广播信息的时候处于各种活动状态。所以没有必要去显式地关闭这组件。

而activity则不同，它提供了用户界面。只要会话依然持续，无论会话过程有无空闲，activity同用户进行长时间会话且可能一直处于活动状态。与此相似，service也会在很长一段时间内在后台保持运行。所以Android为关闭activity和service提供了一系列有序的方法。

activity可以通过调用自身的finish()方法来关闭。一个activity可以通过调用finishActivity()方法来关闭另外一个activity（它用startActivityForResult() 启动的）。

service可以通过调用自身的stopSelf()方法，或调用 Context.stopService()来停止。

系统也会在组件不再被使用的时候戒者当Android必须为更多的活动组件回收内存时关闭它。稍后的组件的生命的周期一节，将对这种可能性及结果进行更详细的介绍讨论。

1.4 manifest文件

当Android启动一个应用程序组件之前，它必须知道那个组件是存在的。因此，应用程序会在一个被打包到Android包中的manifest文件中声明它的组件，.apk文件还将涵括应用程序的代码、文件以及其它资源。

manifest文件是一个结构化的XML文件，而且对于所有应用程序，文件名总是AndroidManifest.xml。除了声明此应用程序各个组件，它会做很多其他工作，比如指明应用程序所需链接到的库的名称（除了默认的Android库外）以及标出应用程序期望获得的各种权限。

但manifest文件最重要的任务是向Android报告此应用程序的各个组件。丼例说明，一个activity可能声明如下：

<activity>元素的name属性指定了实现此activity的 Activity子类。icon和label属性指向包含展示给用户的此activity的图标和标签的资源文件。

其它组件也以类似的方法声明——<service> 元素用于声明service，

<receiver> 元素用于声明broadcast receiver，而 <provider> 元素用于声明content provider。未在manifest文件中进行声明的activity、service以及content provider将不为系统所见，从而也就永不会被运行。然而，broadcast receiver既可以在manifest文件中声明，也可以在代码中动态创建（为BroadcastReceiver对象），并以调用Context.registerReceiver()的方式注册至系统。

1.5 Intent过滤器

一个Intent对象可以显式地指定一个目标组件。如果进行了返种指定，Android会找到这个组件（基于manifest文件中的声明）并激活它。但如果intent没有显式地指定一个目标，Android就必须找到最合适的组件来响应此intent。这个过程是通过比较Intent对象和所有潜在目标的intent过滤器完成的。组件的intent过滤器会通知Android它所能处理的intent类型。如同组件的其它必要信息一样，这些intent过滤器是在manifest文件中进行声明的。返里有一个对先前例子的扩展，其中加入了针对activity的两个intent过滤器：

示例中的第一个过滤器——action“android.intent.action.MAIN”和

category“android.intent.category.LAUNCHER”的组合——是常见的一个。它标明了此activity应该在应用程序启动器中显示，就是用户在屏幕上看到的此设备上可供启动的应用程序的列表。换句话说，这个activity是应用程序的入口点，是用户在启动器中选择运行这个应用程序后所见到的第一个activity。

第二个过滤器声明了此activity在一种特定类型的数据上可以执行的操作。 一个组件可以拥有任意数量的intent过滤器，每个都声明了一套不同的功能。如果组件没有包含任何过滤器，它只能被显式地指明作为目标组件的intent激活。

对于在代码中创建并注册的broadcast receiver来说，intent过滤器将被直接实例化IntentFilter为对象。其它所有的过滤器都在manifest文件中设置。

1.6基于XML的布局

虽然纯粹通过Java代码在activity上创建和添加部件，在技术上是可行的，我们在第4章中做的一样，更常见的方法是使用一种基于XML的布局文件。动态的小部件实例保留更多，情况复杂，小工具在编译时不为人所知（例如，在数据检索了互联网基础上将单选按钮填充柱。

考虑到这一点，现在是时候打破XML来学习如何用此种方式来布置Android activities。

正如其名称所示，一个基于XML的布局是一个关系到每个规格的小部件，和他们的容器（更多关于此内容的在第7章）编码的XML格式。具体来说，Android认为基于XML的布局是资源，因此布局的文件存储在res /在你的Android项目布局目录中。

每个XML文件包含一个指定的部件和容器布局元素树，一种意见认为构成层次。对XML元素的属性，描述一个部件应如何看或者一个容器应如何运转。例如，如果一个按钮元素。

有一个Android的属性值：文字样式=“bold”，这意味着该文本出现在按钮的表面应该是呈现一个粗体字体样式.

Android的SDK中附带一个使用的布局的工具（aapt）。这个工具应自动调用你的Android工具链（例如，Eclipse中，Ant’s build.xml）。作为一个开发人员，尤其重要的是，在您的项目中aapt生成R.java源文件，让您能在那些布局中直接从Java代码中获取布局和部件。

XML作为一个GUI定义格式是越来越流行普遍。微软的XAML，Adobe的Flex，和Mozilla的XUL都采取Android类似的方法。