

Android API Guides

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

编辑流程

这里主要是和Android技术相关的开发指南，很多都是来源于官方的API Guides（<http://developer.android.com/intl/zh-CN/guide/components/index.html>）

请找到自己要翻译的部分，按下面的'链接规范'添加到下面列表中，然后开始翻译～

- [应用的组成部分 - Application Components](#) (已经完成已经排版)
 - [应用的基本原理 - Application Fundamentals](#) (已完成已排版)
 - [活动 - Activities](#) (已完成已排版)
 - [片段 - Fragments](#) (未完成)
 - [加载器 - Loaders](#) (已完成)
 - [任务和返回栈 - Tasks and Back Stack](#) (已完成已排版)
 - [服务 - Services](#) (未完成)
 - [绑定服务 - Bound Services](#) (已完成已排版)
 - [Android接口描述语言-Android Interface Definition Language \(AIDL\)](#) (已完成已排版)
 - [内容提供器 - Content Providers](#) (已完成已排版)
 - [内容提供器基础 - Content Provider Basics](#) (未完成)
 - [创建一个内容提供器 - Creating a Content Provider](#) (已完成已排版)
 - [日历提供器 - Calendar Provider](#) (已完成已排版)
 - [通信录提供器 - Contacts Provider](#) (未完成)
 - [意图和意图过滤器 - Intents and Intent Filters](#) (已完成已排版)
 - [进程和线程 - Processes and Threads](#) (部分内容未翻译，已排

- 版)
- [权限系统 - Permissions](#) (已完成已排版)
 - [窗口小部件 - App Widgets](#)(已完成已排版)
 - [Android清单 - Android Manifest](#) (已完成已排版)
- [用户界面 - User Interface](#) (已完成已排版)
 - [概述 - Overview](#) (已完成已排版)
 - [布局 - Layouts](#) (已完成已排版已审核)
 - [线形布局 - Linear Layout](#) (已完成已排版)
 - [相对布局 - Relative Layout](#) (已完成已排版)
 - [列表视图 - List View](#) (已完成已排版)
 - [网格视图 - Grid View](#) (已完成已排版)
 - [输入控件 - Input Controls](#) (已完成已排版)
 - [按钮 - Buttons](#) (已完成已排版)
 - [文本区域 - Text Fields](#) (已完成已排版)
 - [复选框 - Checkboxes](#) (已完成已排版)
 - [单选按钮 - Radio Buttons](#) (已完成已排版)
 - [开关按钮 - Toggle Buttons](#) (已完成已排版)
 - [下拉列表 - Spinners](#) (已完成已排版)
 - [选择器 - Pickers](#) (已完成已排版)
 - [输入事件 - Input Events](#) (已完成已排版)
 - [菜单 - Menus](#) (已完成已排版)
 - [对话框 - Dialogs](#)(已完成已排版)
 - [动作条 - Action Bar](#) (已完成已排版)
 - [通知栏 - Notifications](#) (已完成已排版)
 - [Toast通知 - Toast Notifications](#) (已完成已排版)
 - [状态通知 - Status Notifications](#) (已完成已排版)
 - [搜索 - Search](#) (未完成)
 - [创建一个搜索界面 - Creating a Search Interface](#) (已完成已排版)
 - [增加当前搜索提醒 - Adding Recent Query Suggestions](#) (已完成已排版)
 - [增加个性化提醒 - Adding Custom Suggestions](#) (未完成)
 - [搜索配置 - Searchable Configuration](#) (已完成已排版)
 - [拖放操作 - Drag and Drop](#) (已完成已排版)

[可访问性 - Accessibility](#) (已完成已排版)

- [应用程序的可访问性 - Making Applications Accessible](#)(已完成已排版)
- [构建可访问性服务 - Building Accessibility Services](#) (已完成已排版)

- [风格和主题 - Styles and Themes](#)(已完成已排版)
- [自定义控件 - Custom Components](#) (未完成)

- [应用程序资源 - App Resources](#) (已完成已排版)

- [概述 - Overview](#) (已完成已排版)
- [提供的资源 - Providing Resources](#) (已完成已排版)
- [对资源的访问 -Accessing Resources](#)(已完成已排版)
- [运行时变化的处理 - Handling Runtime Changes](#)(部分内容未翻译, 已排版)
- [本地化 - Localization](#) (未完成)
- [资源类型 - Resource Types](#) (未完成)
 - [动画 - Animation](#) (未完成)
 - [状态颜色列表 - Color State List](#) (已完成已排版)
 - [图形处理类资源 - Drawable](#) (已完成已排版)
 - [布局 - Layout](#) (未完成)
 - [菜单 - Menu Resource](#)(已完成已排版)
 - [字符串 - String Resources](#)(已完成已排版)
 - [风格 - Style Resource](#)(已完成已排版)
 - [其他类型 - More Resource Types](#)(已完成已排版)

- [动画和图形 - Animation and Graphics](#) (已完成已排版)

- [概述 - Overview](#) (已完成已排版)
- [属性动画 - Property Animation](#) (部分内容未翻译, 已排版)
- [补间动画 - View Animation](#) (已完成已排版)
- [帧动画 - Drawable Animation](#) (已完成已排版)
- [画布和绘制 - Canvas and Drawables](#) (已完成已排版)
- [OpenGL - OpenGL](#)
- [硬件加速 - Hardware Acceleration](#) (已完成已排版)

- [高性能计算 - Computation](#) (已完成已排版)

- [RenderScript编程 - Renderscript](#) (已完成已排版)
- [RenderScript编程进阶 - Advanced Renderscript](#) (已完成已排版)

[运行时API说明 - Runtime API Reference](#) (未完成)

- [多媒体和照相机 - Media and Camera](#) (已完成已排版)
 - [媒体播放 - Media Playback](#) (已完成已排版)
 - [支持的媒体格式 - Supported Media Formats](#) (已完成已排版)
 - [音频捕获 - Audio Capture](#) (已完成已排版)
 - [JET引擎 - JetPlayer](#) (已完成已排版)
 - [照相机 - Camera](#) (未完成5)
- [定位和传感器 - Location and Sensors](#) (已完成已排版)
 - [定位和地图 - Location and Maps](#) (已完成已排版)
 - [定位策略 - Location Strategies](#) (已完成已排版)
 - [传感器概述 - Sensors Overview](#) (已完成已排版)
 - [手势传感器 - Motion Sensors](#) (未完成5)
 - [位置传感器 - Position Sensors](#) (未完成5)
 - [环境传感器 - Environment Sensors](#) (未完成5)
- [通信 - Connectivity](#) (已完成已排版)
 - [蓝牙 - Bluetooth](#)
 - [NFC通信 - NFC](#) (已完成已排版)
 - [NFC基础 - NFC Basics](#) (已完成已排版)
 - [NFC进阶 - Advanced NFC](#) (已完成已排版)
 - [Wi-Fi直连 - Wi-Fi Direct](#) (已完成已排版)
 - [USB通信 - USB](#) (已完成已排版)
 - [附件模式 - Accessory](#) (已完成已排版)
 - [主机模式 - Host](#) (已完成已排版)
 - [SIP协议 - SIP](#) (已完成已排版)
- [文本输入 - Text and Input](#) (已完成已排版)
 - [复制和粘贴 - Copy and Paste](#) (已完成已排版)
 - [创建一个输入法 - Creating an IME](#) (已完成已排版)
 - [拼写检查器 - Spelling Checker Framework](#) (已完成已排版)
- [数据存储 - Data Storage](#) (已完成已排版)
 - [存储选项 - Storage Options](#) (已完成已排版)
 - [数据备份 - Data Backup](#) (已完成未排版)
 - [应用程序安装位置 - App Install Location](#) (已完成已排版)
- [系统管理员 - Administration](#) (已完成未排版)
 - [硬件管理 - Device Policies](#) (已完成未排版)

- [web应用 - Web Apps](#) (已完成已排版)
 - [概述 - Overview](#) (已完成已排版)
 - [web应用的屏幕适配 - Targeting Screens from Web Apps](#) (已完成已排版)
 - [利用webview构建web应用 - Building Web Apps in WebView](#) (已完成已排版)
 - [调试web应用 - Debugging Web Apps](#) (已完成已排版)
 - [web应用的优化 - Best Practices for Web Apps](#) (已完成已排版)
- [更好的策略 - Best Practices](#) (已完成已排版)
 - [支持多屏幕 - Supporting Multiple Screens](#) (已完成已排版)
 - [适配指定屏幕 - Distributing to Specific Screens](#) (已完成已排版)
 - [屏幕兼任模式 - Screen Compatibility Mode](#) (已完成已排版)
 - [支持平板和手机 - Supporting Tablets and Handsets](#) (已完成已排版)
 - [性能考虑 - Designing for Performance](#) (已完成已排版)
 - [JNI使用技巧 - JNI Tips](#) (已完成已排版)
 - [响应灵敏性设计 - Designing for Responsiveness](#) (已完成已排版)
 - [最佳流畅性设计 - Designing for Seamlessness](#) (已完成已排版)
 - [更高安全性设计 - Designing for Security](#) (未完成)
- [Google提供的服务 - Google Services](#) (已完成已排版)
 - [应用程序内部付费机制 - In-app Billing](#) (已完成已排版)
 - [应用程序内部付费机制概述 - In-app Billing Overview](#) (已完成已排版)
 - [如何使用应用程序付费服务 - Implementing In-app Billing](#) (已完成已排版)
 - [订阅机制 - Subscriptions](#) (已完成已排版)
 - [安全与设计 - Security and Design](#) (已完成已排版)
 - [测试应用程序付费服务 - Testing In-app Billing](#) (已完成已排版)
 - [应用程序付费机制的管理 - Administering In-app Billing](#) (已完成已排版)
 - [应用程序付费的相关API - In-app Billing Reference](#) (已完成已排版)
 - [应用程序许可机制 - Application Licensing](#) (已完成已排版)

- [许可机制概述 - Licensing Overview](#) (已完成已排版)
- [设置许可机制 - Setting Up for Licensing](#) (已完成已排版)
- [在应用中增加许可 - Adding Licensing to Your App](#) (已完成已排版)
- [许可机制API - Licensing Reference](#) (已完成已排版)
- [Google软件商店服务 - Google Play Services](#) (已完成已排版)
- [在Google Play中加过滤器 - Filters on Google Play](#) (已完成已排版)
- [多APK支持 - Multiple APK Support](#) (已完成已排版)
- [对APK附加文件的服务 - APK Expansion Files](#) (未完成)
- [Google云消息服务 - Google Cloud Messaging](#) (已完成已排版)
 - [如何使用google云服务 - Getting Started](#) (已完成已排版)
 - [架构概述 - GCM Architectural Overview](#) (未完成)
 - [演示教程 - GCM Demo Application](#) (已完成已排版)
 - [Google云服务进阶 - GCM Advanced Topics](#) (已完成已排版)
 - [信息迁移 - Migration](#) (已完成已排版)

来自“[index.php?title=Android_API_Guides&oldid=13903](#)”

Application Components

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

- [1 应用程序组件](#)
 - [1.1 训练](#)
 - [1.1.1 管理Activity的生命周期](#)
 - [1.1.2 Building a Dynamic UI with Fragments-使用Fragment创建一个动态的UI](#)
 - [1.1.3 Content Provider](#)
 - [1.2 博客文章](#)
 - [1.2.1 Using DialogFragments-使用DialogFragments](#)
 - [1.2.2 Fragments For All-通用的Fragments](#)
 - [1.2.3 Multithreading for Performance-多线程展示](#)

应用程序组件

Android's application framework lets you create extremely rich and innovative apps using a set of reusable components. This section explains how Android apps work and how you use components to build them.



通过一组可重用的组件,Android的应用程序框架使您可以创建功能丰富,充满创新的应用程序,本节将介绍Android应用是如何工作的,以及教您如何使用组件来构建Android应用程序。

训练

管理**Activity**的生命周期

This class explains important lifecycle callback methods that each Activity instance receives and how you can use them so your activity does what the user expects and does not consume system resources when your activity doesn't need them.

这节课将介绍每个**Activity**实例含有的重要的生命周期回调方法，使用它们，你可以使你的**activity**做用户希望**activity**做的事情。同时，掌握了这些回调方法，当你的**activity**不需要他们的时候，你可以使他们尽量少的或者不耗费系统资源。

Building a Dynamic UI with Fragments-使用**Fragment**创建一个动态的**UI**

This class shows you how to create a dynamic user experience with fragments and optimize your app's user experience for devices with different screen sizes, all while continuing to support devices running versions as old as Android 1.6.

这节课向您介绍怎样使用**Fragment**来创建一个动态的用户交互界面，怎样优化您的应用，使您的应用支持不同大小的屏幕，这些知识支持**android1.6**版本以上的系统

Content Provider

This class covers some common ways you can send and receive content between applications using Intent APIs and the ActionProvider object.

这节课介绍一些在 Intent API 和ActionProvider 的应用程序之间可以发送和接受Content Provider的通用方法。这个类包括一些常见的方法，您可以通过试用Intent API来实现不同应用程序之间的Content Provider数据共享

博客文章

Using DialogFragments-使用DialogFragments

In this post, I'll show how to use DialogFragments with the v4 support library (for backward compatibility on pre-Honeycomb devices) to show a simple edit dialog and return a result to the calling Activity using an interface. 本文将向您展示怎么通过V4支持库使用DialogFragments(兼容Android 3.0之前的版本)来显示一个简单的编辑对话框并通过一个接口返回结果给之前调用他的Activity.

Fragments For All-通用的Fragments

Today we've released a static library that exposes the same Fragments API (as well as the new LoaderManager and a few other classes) so that applications compatible with Android 1.6 or later can use fragments to create tablet-compatible user interfaces. 今天我们发布了一个静态库,这个库提供了和3.0版本相同的Fragments API(也包括新的LoaderManager和一些其他的类),通过他们,Android 1.6或更高版本可以运行使用fragments 创建的含有 平板级界面的应用.

Multithreading for Performance-多线程展示

A good practice in creating responsive applications is to make sure your main UI thread does the minimum amount of work. Any potentially long task that may hang your application should be handled in a different thread. 一个好的交换应用需要确保您的主UI线程做做少的工作.然后潜在的可能是应用挂起的耗时任务应该在其他线程中处理

来自“[index.php?title=Application_Components&oldid=14028](#)”

Application Fundamentals

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/fundamentals.html>

- 负责人： kris

目录

[[隐藏](#)]

[1 应用基础-Application Fundamentals](#)

[2 应用组件-Application Components](#)

- [2.1 Activities](#)活动
- [2.2 Services](#)服务
- [2.3 Content providers](#) 内容提供者
- [2.4 Broadcast receivers](#)广播接收者
- [2.5 激活组件-Activating Components](#)

[3 清单文件-The Manifest File](#)

- [3.1 声明组件-Declaring components](#)
- [3.2 声明组件功能-Declaring component capabilities](#)
- [3.3 声明应用需求-Declaring application requirements](#)

[4 应用资源-Application Resources](#)

应用基础-Application Fundamentals

Android应用程序以java作为开发语言。用Android SDK 提供的工具，可以将应用程序所需要的数据和资源文件打包到一个android包文件中，这个文件用.apk作为扩展名。所有代码都在单个.apk文件中，这个文件就是通常

安装在**Android**设备中的应用。一旦安装到了一个设备，每个应用生存在它自己的安全沙箱中。

- 一个**Android**系统是一个多用户的**Linux**系统，其中的每个应用都是一个不同的用户。
- 默认情况下，系统给每个应用分配一个独立的**Linux**用户**ID**(这个**ID**只由系统使用并且对应用来说是不可知的)，系统给在某个应用中的所有文件设置了权限，所以只有分配了那个用户**ID**的应用才能访问它们
- 每个进程拥有它自己的虚拟机，所以一个应用代码的运行，与其他应用代码的运行是隔离的。
- 默认情况下，每个应用程序均运行于它自己的**Linux**进程中。当应用程序中的任意代码开始执行时，**Android**启动一个进程，而当不再需要此进程而其它应用程序又需要系统资源时，则关闭这个进程。

通过这种方法，**Android**系统实现了最小特权原则。默认，每个应用仅仅访问需要工作的组件，并不多做其他的事。这样创建了一个非常安全的环境，应用不能访问系统没有授权的其他部分。

然而，应用可以有多种方法来与其他应用，共享数据及访问系统服务：

- 有可能安排两个应用共用一个**linux**系统**ID**，在那种情况下，它们能互相访问相互的数据。为了节约系统资源，拥用相同用户**ID**的应用，可能也被安排运行在同一个**Linux**进程中并共享相同的**VM**(应用必须被签名成同样的认证)。
- 所用应用能请求允许访问硬件数据，比如像用户通信录，**SMS**消息及可挂载的存储设备(**SD card**)，摄像头，蓝牙等，所有应用的权限必须在用户安装时被许可。

上述了一个应用怎样存在于一个系统中的相关基本概念，这个文档的其他部分将向你介绍如下 内容：

- 定义在你的应用中核心框架组件
- 在**manifest**中，给你的应用，声明组件及设备特点请求
- 独立于应用代码的资源，可以让你的应用极大的优化它在各种配置设备的表现

应用组件-Application Components

应用组件是构建Android应用程序的关键和基石。 每个组件是一个不同的入点， 系统可以从这些点进入到你的应用。对于用户来说，并不是每个组件都是实际的入点，但它们之间有一些依赖.但是每一个存在的组件 都有它自己的一个入点，并扮演一个特定的角色--每一个都是独一无二的构建块，帮助你定义你的应用的整体行为.

有四个不同类型的应用组件， 每个类型服务于一个不同的目的，并有不同的生命周期，生命周期定义了如何创建和销毁它.

下面是四种应用组件:

Activities活动

一个activity在一个屏幕,显示一个用户接口.比如，一个email应用可能有一个activity，这个 activity用于显示新的email列表.而另一个activity用于写邮件,还有一个activity用于读取邮件.虽然这些activities一起工作于email应用中，形成一个完整的用户体验但每一个部分又是相互独立的.正因如此，不同的应才能启动这些活动的任意一个 (如个email应用允许它).比如,一个照相的应用，能开启一个email应用中写封新邮件的活动，让用户分享一张照片.

一个activity被当作Activity的子类来实现的，在Activities开发指南中,你可以学到更多关于它的使用

Services服务

一个service是长期运行在后台，执行操作的组件，甚至可以为远程进程工作.一个服务不提供用户界面.比如，当用户 在其他应用中时，一个服务可能在后台播放音乐,或者在后台获取数据，这并不影响用户跟其他的活动进行交互操作.其他的组件，比如一个activity，可以启动一个服务，并可以让它运行或者绑定到这个activity，以便与其进行交互操作.

一个服务是作为Service子类来实现的，在Services开发指南中，你能学到更多关于它的使用

Content providers 内容提供者

一个content provider管理共享的应用数据集.你可以把数据存在文件系统中，一个SQLite数据库中，网上，或你应用可以访问的永久存储器中.通过内容提供者，其他的应用可以查询甚至修改数据(如果内容提供者允许的话).比如，Android系统提供一个内容提供者管理用户通信录信息.因此，任何拥用适当权限的应用，可以查询内容提供者的部分来(比如 `ContactsContract.Data`)读取和写入关于某个人的信息.内容提供者对于读取和写入属于你的应用的私有的非共享数据也是非常有用的，比如Note Pad样例应用程序，就使用内容提供者来保存笔记的.

一个内容提供者被当作的子类实现，并且必须实现一套标准的APIs,以让其他的应用能执行交换操作。

参考Content Providers开发指南,以了解更多信息.

Broadcast receivers 广播接收者

广播接收者是一个响应系统范围广播公告（通知）的组件.许多广播信息，都是来源于系统，比如，通知屏幕关闭的公告，电量低，或抓取了一张图片.应用也能发起广播，比如，让其他的应用知道一些数据已下载到设备了，并且他们可以使用了。虽然广播接收者，不能显示用户界面，但当一个广播事件发生时，它们可以创建一个状态通知器，去提醒用户.但更多情况下，一个广播接收者只是一个其他组件,想要做极小量事件的一个"gateway"(途径).举例，它可能发起一个服务，去执行关于某个事件的一些工作.

一个广播接收者，是当作BroadcastReceiver子类被实现的.每个广播接收者都是从Intent对象衍生出来的。更多信息，请参考BroadcastReceiver类

任何一个应用能启动另一个其他应用的组件,是Android系统设计独一无二的方面(aspect).比如，你想要用设备的照相机拍一张图片.其他的应用已经有了这个功能，并且你的应用可以使用它，而不需要你自己去开发一个拍照

相的activity.你并不需要合并(包含)或者甚至是链接camera应中的代码;而只是,简单的启动camera应用中的活动,来拍照就可以了.当拍照完成,甚至把照片返回给你的应用,所以你能使用它。对于用户来讲, camera像是你应用中一部分.当系统开启一个组件时,它会启动那个应用的进程(如果该应用没有运行),并实例化该组件所需要的类.举例,如果你的应用开启一个 camera应用的activity,来拍照,这个activity将运行在属于camera应用的进程中,而不是在你的应用的进程中.因此,不像大多数其他的系统的应用,Android应用,没有单个的入点(比如没有main()函数).

因为系统运行的每个应用,在一个带有文件权限的,独立的进程中,这样限制了对其他应用的访问,你的应用不能直接访问其他应用中的组件.但时, Android系统也能激活其他应用的组件.你必须传一个消息给系统,指定你想要启动的组件,然后系统为你激活这个组件.

激活组件-Activating Components

4个组件中的其中三个组件---activities, services, 和broadcast receivers----是被叫做intent的异步消息激活的.在运行时,Intents把某个的组件与其他的组件互相绑定,而不管这个组件是否属于你的应用还是其他的应用(你可以把它们想像成一个消息,用于请求一个其他组件的动作).

一个intent是一个由Intent创建的对象.该对象定义了一个激活某个特定组件或者某个组件类型的消息,一个intent可以是显示的,同样,也可以是隐式的.

对于activities和services,一个intent(意图)定义了一个要执行的动作(比如: to"view"或"send"些什么),并指定了要采用的URI格式的数据(其中一些,是其他组件启动所需要知道的).比如,一个intent可能传送一个请求给一个 activity,要显示一张图片或打开一个网页.在有些情况,你启动一个activity接收一个结果,这种情况下, activity将在Intent 中返回一个结果.(比如,你可以指示一个intent,让用户取一个人的联系方式,并返回给你,返回的intent中会包含一个指向选定联系方式的 URI.)

对于广播接收者,intent只是定义了一个做为广播的公告.(比如,一个广播指出,设备电池低,它只是包含了一个动作字串,表示"电池低").

其他组件,内容提供者,不会被intents所激活.进一步讲,它是内容解释者(ContentResolver)所请求的目标所激活的. 内容解释者,处理所有与内容

提供者的直接交换.所以组件不需要执行与提供者交换,而是调用ContentResolver对象方法.(这一句不好理解。)为了安全起见,组件请求信息与内容提供者之间有一个抽象层.

下面是激活各种类型组件的几个方法:

- 你可以通过传一个(或者一些要做新的事情)Intent参数给startActivity()或startActivityForResult()(当你想要activity返回一个参数)函数(), 来启动一个activity.
- 你可以传一个Intent给startService()方法, (或给一个新的指令给正在运行的服启), 或者你可以传一个Intent给bindService()方法来邦定到服务.
- 你可以通过使用sendBroadcast(), sendOrderedBroadcast(), 或者sendStickyBroadcast()三种方法来广播一个intent。
- 你可以对ContentResolver调用query()方法, 对内容提供者进行查询

关于使用intents的详细信息, 请看Intents and Intent Filters 文档。在后面的文档中, 也有一些关于激活某个组件的信息Activities, Services, BroadcastReceiver and Content Providers.

清单文件-The Manifest File

在Android系统开启一个应用组件之前, 系统必须通过读取AndroidManifest.xml文件来知道组件的存在.你的应用必须把它所有的组件声明在这个文件中, 并且必须在应用工程的根目录下.

这个manifest文件除了声明组件外, 还处理了许多其他的事情, 比如:

- 指定应用请求的其他权限, 访问网络或访问用户的通信录
- 声明应用要求的最小API Level, 应用使用的是那个API
- 声明应用请求和使用的软硬件特征, 比如照相机, 蓝牙服务, 或多点触模屏
- 应用需要链接的API库, 比如Google Maps library
- 等等

声明组件-Declaring components

manifest文件的主要任务是告诉系统, 应用的组件, 比如, 一个manifest可

以这样声明一个activity：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
            </activity>
            ...
        </application>
    </manifest>
```

在<application> 元素中,android:icon指定应用的icon资源 在<activity> 元素中的,android:name 属性，指定Activity子类的完全类名,android:label 属性，为activity指定一个用户可以见的标签。 你必须这样声明 所有应用的组件：

- <activity> 声明活动的元素
- <service> 声明服务的元素
- <receiver> 声明广播接收者元素
- <provider> 声明内容提供者元素

在你代码中包含的,Activites,services和内容提供者，若没有在manifest中声明，对系统来说是不可见的，即将永远不会运行。但是，广播接收者即可以在manifest中声明，也可以在代码中动态创建(做为BroadcastReceiver对象)并且通过 registerReceiver()方法向系统注册。

了解manifest文件的详细构建过程，请看The AndroidManifest.xml File文档
声明组件功能-**Declaring component capabilities**

就如在上面的Activating Components中所讨论的，你可以用一个Activating Components启动activities,services和broadcast 接收者.你也可以在intent中显式的指定目标组件（使用组件类名）。然而，intent真正强大的是它的intent action.(动作)。通过使用intent动作，你只须简单的描述你要执行的action类型,(并且，可选的与执行动作有关的数据),并且允许系统 在设备上找到一个组件，这样就可以执行那个动作并启动它。如果有多个组件可以执行，intent指定的action，那么用户选择执行那一个.

通过比较设备上的其他应用的manifest文件上的intent filters与接收到

的intent. 系统确定那个组件可以响应一个intent. 当你在你的应用的manifest中声明一个组件时，你可以选择包括intent filters(意图过滤器)，来指定组件的功能，以让其能响应其他应用的intents. 你可以加一个组件声明的元素的子元素<intent-filter>，为你组件声明一个意图过滤器。

比如，一个email应用中，新建email的一个activity可能在它的manifest 中声明了一个意图过滤器，以便能响应"send"意图(为了发送邮件)。然后，在你的应用中的一个activity，创建了一个带有"send" ACTION_SEND的意图。当你调用startActivity()方法，启动该意图过滤器时，系统将其匹配到email应用的"send"活动，并运行它。

关于创建意图过滤器的详细信息，参考Intents and Intent Filters 文档

声明应用需求-Declaring application requirements

有许多设备装了Android，但它们并不提供所有相同的特点和功能。为了避免你的应用，装在一个没有你应用所必特征的设备上。通过在你的 manifest文件中声明软件硬件要求，明了的指出你的应用支持的硬件类型是非常重要的。大多数声明仅仅只是信息，系统并不读取他们，但像Android 市场这样的其他服务，将读取它们，以便让用户在为他们的设备寻找应用时，可以进行筛选。

比如，如果你的应用需要有照相机，并且使用的API是2.1(API Level 7)，你应在你的manifest文件中声明这些要求。这样，那些没有照相机并且Android版本低于2.1的设备，就不能从Android市场上安装你的应用。

但，你也可以声明你的应用使用camera，但不必须要求。那种情况，你的应用必在运行时一个检查，以确定设备是否有一个照相机，如果没有照相机，并禁止与照相相关的功能。

下面是一些重要的设备特性，你在设计和开发应用时必须要考虑的..

- screen size and density 屏幕尺寸与解释度

为了能从它们的屏幕尺寸来分类设备，Android为每个设备定义了两个特性：屏幕尺寸(屏幕的物理尺寸)和解释度(在屏上的像素的物理密度，或者dpi--每英寸的点数)。为了简化屏幕配置的所有不同类型，Android系统把它们分成可选的组，以便更容易定位。

屏幕大小：小，正常，大和极大

屏幕解释度:低解释度， 中解释度， 高解释度， 和极高解释度

默认情况下，你的应用是兼容所有屏幕尺寸和解释度的，因为Android系统对此做了适当的调整，以使得它适合你的UI布局和图像资源

然而，你应为某个屏幕尺寸创建特殊的布局，并为某些解释度提供特定的图像，使用可选的资源，并在你的manifest文件中用<supports-screens>元素声明，以明确指出你的应用支持的屏幕尺寸。

更多信息，参考Supporting Multiple Screens文档

- **Input configurations** 输入配置

许多设备为用提供了一个不同类型输入装置，比如，硬件键盘，轨迹球，five-way导航pad.如果你的应用必须要一个特别的输入硬件，那么你应在你的应用中使用<uses-configuration>元素声明.但时，应用必须要一个特别的输入配置的情况是极少的.

- **Device features** 设备特性

在一个装有Android的设备中，有许多软硬件特性，有可能有，或有可能没有。比如照相机，光敏器件，蓝牙，或某个版本的 OpenGL,或者触模屏的精度.你应该从不假设，在所有的装有Android的设备中某个特点是可用的（除了标准的Android库），所以你应该用 <uses-feature>元素声明你的应用支持的特征.

- **Platform Version** 平台版本

不同的Android设备，经常运行不同的Android平台版本，比如Android1.6或者2.3.每一个成功的版本通常包括在前一个版本中不可用的API。为了指出,那些APIs集是可用的，每个平台版本指定了一个API Level(比如, Android 1.0 is API Level 1 and Android 2.3 is API Level 9).如果你使用的APIs是在1.0版之后，加入到平台的，你应该用<uses-sdk>元素，声明最小API级别，这样就指出了那些 API将被采

用.

为你的应用声明所有必要性的要求非常重要.因为，当你把你的应用发布到Android市场.市场，将用这些声明信息来过滤出，那些应用在每个设备是可用的. 同样，你的应用应该只能在满足所有你应用需求的设备上才可用.

更多关于Android市场如何基于这些需求过滤的，请看Market Filters文档

应用资源-Application Resources

一个Android应用的组成不仅只是代码----它还有与代码独立的资源，比如图像，音频文件，及与应用可显图像任何其他相关的.比如，你应该定义动画，菜单，风格，颜色，和用XML文件定义活动的布局.使用应用资源，能让你的应用在不修改任何代码的情况下容易的升级各种特性---并且通过提供一套可选取的资源--能优化你的应用在各种配置不同的设备中的表现(比如不同的语言和屏幕尺寸).

对于每个包含在你的Android工程中的资源，SDK将其定义成一个唯一的整型ID,这样你就可以在你的代码中或在XML文件中定义的其他资源中引用它.如果你的应用包括一个图片名字是logo.png(保存在res/drawable/目录)，SDK工具将生成一个资源ID命名成R.drawable.logo,你可以用它来引用图片，并插入你的用户界面中

提供与你的代码分开的资源的一个很重要的方面是，使得你能为不同的配置的设备提供可选资源.比如，在XML中定义UI字串，你可以把字串翻译成各种不同的语言并保存在不同的文件中.然后，以基于语言限定词,你可以追加资源目录名(比如res/values-fr/ 用语法语资源)，和用户语言设置,Android系会将相应的资源应用到你的UI中.

Android为你的可选资源，支持许多不同的qualifiers (限定词).限定词是一个包括在你的目录名中的一个简短的字串，是为了定义那些资源将用在，该配置的设备上.再如，由于设备的屏幕的方向和尺寸不同，你通常需要为你的活动定义不同的布局.比如，若设备的屏幕是竖向(高),你可能要一个带有重直button 的布局，当屏幕是横向的 (宽) ，按钮应是水平对齐的.要根据方向来改变布局，你要定义两个不同的布局，并在布局的目录名中使用相应的限定词 (qualifier).然后，系统将自动根据当前的设备朝向来应用相应的

布局.

要详细了解，你的应用中能包含的各种资源，及如何为各种配置的设备创建可选资源，请看**Application Resources**开发指南

来自“[index.php?title=Application_Fundamentals&oldid=14042](#)”



Activities

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：
址：

<http://developer.android.com/guide/topics/fundamentals/activities.html>

翻译：[iceskysl](#)

更新：2012.06.01

后续翻译者：[Albertli](#)

更新：2012.06.13

目录

[[隐藏](#)]

[1 Activity](#)

- [1.1 生成Activity-Creating an Activity](#)
 - [1.1.1 实现用户界面-Implementing a user interface](#)
 - [1.1.2 声明Activity-Declaring the activity in the manifest](#)
 - [1.1.3 使用intent过滤器-Using intent filters](#)
- [1.2 启动Activity-Starting an Activity](#)
 - [1.2.1 启动可以返回结果的 Activity-Starting an activity for a result](#)
- [1.3 关闭Activity-Shutting Down an Activity](#)
- [1.4 管理Activity生命周期-Managing the Activity Lifecycle](#)

- [1.4.1 实现生命周期回调函数-Implementing the lifecycle callbacks](#)
- [1.4.2 保存Activity状态-Saving activity state](#)
- [1.4.3 处理配置更改-Handling configuration changes](#)
- [1.4.4 协调Activity-Coordinating activities](#)

Activity

Activity 是应用程序的组件，它提供了一个屏幕，用户可与之互动，以做一些事情，如拨打电
话，拍照，发送电子邮件，或查
看地图。每个Activity 会提供一个
窗口，在其中绘制它的用户界
面。通常窗口会填满整个屏幕，
但也有可能比屏幕小并且浮动在
其他窗口之上。

应用程序通常是多个松散并相互
绑定的Activity组成。一般，用户
首次启动应用时，将启动一个被
指定为“main”的Activity。每个
Activity都可以启动另一
个Activity，以执行不同的动作。
每次启动一个新的Activity，以前的
Activity停止，但在系统堆栈保
留Activity（“回栈”）。一个新的
Activity启动时，它将Activity压
到栈里面并取得用户的焦点。回
堆栈遵守基本的“后进先出”的堆
栈机制，这样，当用户在
前Activity，按下返回按钮，则弹
出堆栈（并销毁）和恢复以前

快速浏览

- 在程序中，屏幕上的一
个Activity 对应一用户界面
- Activities后台运行时，可以通过
状态恢复返回前台

本文内容

生成Activity

[实现用户界面](#)
[声明Activity](#)

启动Activity

[启动 Activity 的返回结果](#)

关闭Activity 管理Activity生命周期

[实现生命周期回调函数](#)
[保存Activity状态](#)
[处理配置更改](#)
[协调Activity](#)

关键类 Activity 参考 Hello World

的Activity。（回栈在[任务和返回堆栈 - tasks-and-back-stack](#)文件有详细说明。）

Tutorial Tasks and Back Stack

Activity停止是因为一个新的Activity开始，通过活动的生命周期回调方法，通知这种状态发送变化。一个Activity可能会收到多个回调方法，当状态发生变化时.产生,停止,恢复,销毁,这些回调在适当的时机提供给您做相应工作的机会。例如,停止时,你的Activity应该释放大型对象,如网络或数据库连接。恢复Activity时,你可以重新获得必要的资源和恢复被中断的Activity。这些状态转换是所有的Activity程序周期的一部分。

本文件的其余部分将讨论如何建立和使用Activity，包括讨论一个完整的Activity程序周期是如何工作，你能正确的管理各种Activity状态之间的转换。

生成Activity-Creating an Activity

要生成一个Activity，你必须生成一个Activity子类[Activity](#)（或已有的子类）。在你的子类中，创建，停止，恢复或销毁Activity时，你需要实现回调方法，系统调用时，用这些方法表示在Activity生命周期的各种状态之间的转换。两个最重要的回调方法是：

[onCreate\(\)](#)

必须实现这个方法。生成Activity时系统调用。在你实现中，你应该初始化您的Activity中的组件。最重要的是，这是你必须调用[setContentView\(\)](#) 定义Activity的用户界面的布局。

[onPause\(\)](#)

当用户离开你的Activity时候（尽管它并不总是意味着被销毁Activity），系统调用此方法。通常如果你需要保持当前会话的话，你所有的改动都应在这提交（因为用户可能不返回）。

还有其他几个生命周期回调方法，你应该使用这些方法处理Activity与导致使Activity必须停止，甚至销毁的突发中断之间的关系，来提供流畅的用户体验。所有的生命周期回调方法将在稍后讨论，在有关[管理Activity生命周期部分](#)。

实现用户界面-Implementing a user interface

一个Activity的用户界面是由一组按层派生[视图类 - View](#) 的视图对象组成。每个视图控制Activity窗口的特定矩形空间，以响应用户交互。例如，一个视图可能是一个按钮，当用户触摸它启动一个操作。

Android提供了一些现成的视图，你可以用它来设计和组织布局。“Widgets”就是视图，他提供一个可视的可交互的屏幕元素，如按钮，文本字段，复选框，或只是一个图象。“布局”是从[视图组 - ViewGroup](#)派生的，提供了对子视图特殊的布局模式，如线性布局，网格布局，或相对布局的视图组。还可以继承[视图类 - View](#) 和[视图组 - ViewGroup](#)（或现有的子类）来创建自己的Widgets和布局，并将其应用到您的Activity布局中。

最常见的方式来定义一个布局的意见，是一个XML布局文件保存您的应用程序资源。这种方式，你可以单独从源代码中维护你的用户界面设计以及定义Activity的行为。你可以通过[setContentView\(\)](#) 设置您的Activity的UI布局，传入布局的资源ID。或者，你也可以在Activity代码中生成新的[视图 - View](#)，新建立一个[视图 - View](#) 层次结构插入到[视图组 - ViewGroup](#)，然后传入根[视图组 - ViewGroup](#) 给[setContentView\(\)](#) 进行布局。

有关创建用户界面的信息，请参阅[用户界面](#) 文档。

声明Activity-Declaring the activity in the manifest

要在系统中使用Activity，您必须在manifest文件中声明您的Activity。打开你

的manifest文件，并添加 [**<activity-element>**](#) 作为 [**<application>**](#) `<application>` 元素一个子元素。例如：

```
<manifest ... >
    <application ... >
        <activity android:name=".ExampleActivity" />
        ...
    </application ... >
    ...
</manifest >
```

在这个元素可以包含其他一些属性，包括定义属性，如Activity的标签，图标的Activity，或Activity的主题UI风格。属性[**android: name**](#) 是唯一需要的属性，它指定Activity的类名。一旦你发布你的应用程序，你不应该改变这个名字，因为如果这样做，你可能会破坏一些功能，如应用程序的快捷方式，（阅读博客文章，[不要改动 - things-that-cannot-change](#)）。

有关在manifest文件声明Activity的更多信息请查阅 [**<activity>**](#) 元素。

使用intent过滤器-Using intent filters

一个 [**<activity>**](#) 元素也可使用 [**<intent-filter>**](#) 的元素指定的各种 intent过滤器，以说明其他应用程序组件可以如何激活它。

当你使用Android SDK工具为您创建一个新的应用，[**<activity>**](#) 标签下自动包括用于声明Activity响应“main”的动作的intent过滤器和用于启动的“lancher”分类。intent 过滤器看起来像这样：

```
<activity android:name=".ExampleActivity"
    android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

[**<action>**](#) 元素指定应用程序“main”的入口。[**<category>**](#) 元素指定系统的activity列表上应用启动的Activity，（允许用户启动这个Activity）。

如果你打算让您的应用程序成为一个独立的应用，不允许其他应用程序启动应用中的Activity，那你就不需要其他的intent过滤器。如前面示例所

示, 只有一个Activity带有"main"动作和"launcher"类别。你不让其他Activity用于其他应用程序, 那他们不应有任何intent过滤器, 你必须显示的调用intent启动他们 (在下一节讨论)。

但是, 如果您希望您的Activity响应其他应用程序 (和自己) 的隐式intent, 那么你必须在您的Activity中定义额外的intent过滤器。为了响应你需要响应的每种intent, 你必须包括 [<intent-filter>](#), 其中包括一个 [<action>](#) 元素的和一个可选的 [<category>](#) 元素和/或 [<data>](#) 元素。这些元素指定你的Activity可以响应的intent类型。

查看更多关于您的Activity能够响应intent类型的信息, 查看 [intent和intent过滤器 - intents-filters](#) 文档。

启动Activity-Starting an Activity

你就可以通过调用 [startActivity\(android.content.Intent\)](#) [startActivity \(\)](#) 启动另一个Activity, 它传递一个 [Intent](#) 描述你要启动的Activity。intent指定你要启动的Activity, 或描述要执行的动作类型 (系统为您选择合适的Activity, 甚至可能是他应用程序的Activity)。intent也可携带少量数据被用于要启动的Activity。

在自己的应用程序中, 你会经常需要简单地启动一个已知的Activity。你可以使用类名来创建一个intent, 明确定义你要启动的Activity。例如, 如下是一个Activity的启动另一个命名为 SignInActivity 的 Activity:

```
Intent intent = new Intent (this, SignInActivity.class);
startActivity (intent);
```

另外, 您的应用程序可能还需要执行一些动作, 例如发送电子邮件, 文字信息, 或状态更新, 使用Activity中的数据。在这种情况下, 你的应用程序可能无法有其自身的Activity, 执行这些行动, 所以你可以转而利用设备上其他应用程序提供的Activity为您执行。这是intent真正有价值的地方——你可以创建一个描述了要执行的一个动作的intent, 系统从另一个应用程序启动的适当Activity。如果有多个Activity可以处理的intent, 那么用户可以选择使用哪一个。例如, 如果你想允许用户发送电子邮件, 你可以创建以下intent:

```
Intent intent = new Intent( Intent.ACTION_SEND );
intent.putExtra( Intent.EXTRA_EMAIL, recipientArray );
startActivity( intent );
```

EXTRA_EMAIL EXTRA_EMAIL 为发送电子邮件的邮件地址字符串数组，其充当intent的额外数据。当一个电子邮件应用程序响应此intent，它读取额外提供的字符串数组，并把它们放在程序的“to”地址栏。在这种情况下，应用程序启动电子邮件Activity，当用户完成后，恢复到你的Activity。

启动可以返回结果的 **Activity-Starting an activity for a result**

有时，您可能要取得你启动的Activity的结果。在这种情况下，使用Activity的 [startActivityForResult \(\)](#) (而不是 [startActivity \(\)](#))。然后在后续实现了 [onActivityResult \(\)](#) 回调方法的Activity中取得结果，。后续Activity完成后，它会返回一个 [intent](#) 给您 [onActivityResult \(\)](#) 方法。

例如，也许你希望用户选择他们的联系人之一，便于你的Activity可以做一些与该联系人的信息相关的事情。下面是怎样生成这样的intent并处理结果：

```
private void pickContact() {
    //根据CONTENT PROVIDER URI定义，建议一个用于选择联系人得 INTENT
    Intent intent = new Intent( Intent.ACTION_PICK,
    Contacts.CONTENT_URI );
    startActivityForResult( intent, PICK_CONTACT_REQUEST );
}

@Override
protected void onActivityResult( int requestCode, int resultCode,
Intent data ) {
    //如果请求正常 (OK) 并且请求是PICK_CONTACT_REQUEST
    if ( resultCode == Activity.RESULT_OK && requestCode ==
PICK_CONTACT_REQUEST ) {
        //查询 content provider 取得联系人的名字
        Cursor cursor = getContentResolver().query( data.getData(),
        new String[] { Contacts.DISPLAY_NAME }, null, null, null );
        if ( cursor.moveToFirst() ) { // 游标不为空
            int columnIndex =
cursor.getColumnIndex( Contacts.DISPLAY_NAME );
            String name = cursor.getString( columnIndex );
            // 对选中名字的人进行处理...
        }
    }
}
```

```
    }  
}
```

这个例子说明了使用 [onActivityResult \(\)](#) 方法处理Activity的结果的基本逻辑。首要条件是检查请求是否成功和这个结果是响应是否已知，如果是，那么ResultCode会是 [Activity.html#RESULT_OK RESULT_OK](#)，在这种情况下，requestCode匹配 [startActivityForResult \(\)](#) 的第二个参数。从那起，代码在 [intent](#) 中处理取得Activity查询数据的返回结果的 (data 参数)。

[ContentResolver](#) 针对一个content provider 执行了查询，它返回一个[游标 - Cursor](#)，允许查询的数据被读取。欲了解更多信息，请参阅[content providers](#) 文档。

更多使用[intent](#)的信息，请参阅的 [\[intents and intents-filters\]](#) [intent](#)和[intent过滤器](#)文档。

关闭Activity-Shutting Down an Activity

您可以通过调用 [finish\(\)](#) 方法关闭Activity的。你也可以通过调用 [finishActivity\(\)](#) 关闭之前您启动的一个独立的Activity。

注意：在大多数情况下，你应该不需要显示地使用这些方法结束Activity。在下一节讨论有关的Activity生命周期，Android系统为你管理Activity，所以你不需要来自己结束Activity。调用这些方法可对用户体验产生影响，只有在你确定不想让用户返回此Activity实例时使用。

管理Activity生命周期-Managing the Activity Lifecycle

实施生命周期回调方法来管理您的Activity是开发强大和灵活的应用是至关重要的。一个与其关联的其他Activity直接影响其Activity的生命周期，任务和返回堆栈。

一个Activity可以基本上存在三种状态：

恢复

这项Activity是在屏幕前的，并有用户取得其焦点。（此状态，有时也简称为“运行”。）

暂停

另一个Activity在屏幕前，取得焦点，但原来的Activity仍然可见。也就是说，另一个Activity是这个顶部可见，或者是部分透明的或不覆盖整个屏幕。暂停的Activity完全是存在的（Activity对象保留在内存中，它维护所有状态和成员信息，并保持窗口管理器的联系），但在极低的内存的情况下，可以被系统终止。

停止

Activity完全被另一个Activity遮住了（现在Activity是在“background”）。停止Activity也仍然存在（Activity对象保留在内存中，它保持状态和成员信息，但不和窗口管理器有关联）。然而，它已不再是对用户可见，其他地方需要内存时，它可以被系统终止。

如果一项Activity被暂停或停止，该系统可以从内存中删除它，要求它结束（调用它的[finish \(\)](#)方法），或者干脆杀死它的进程。Activity再次打开时（在被结束或被杀死），它必须创造所有的一切。

实现生命周期回调函数-**Implementing the lifecycle callbacks**

当Activity按照如上所述在不同状态转进，出的时，是通过各种回调方法进行通知的。所有的回调方法是钩子，当的Activity状态变化时您可以覆盖他们来做适当的工作时。以下的Activity，包括基本生命周期的每一个方法：

```
public class ExampleActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
```

```

        super.onCreate(savedInstanceState);
        // The activity is being created.
    }

@Override
protected void onStart() {
    super.onStart();
    // The activity is about to become visible.
}

@Override
protected void onResume() {
    super.onResume();
    // The activity has become visible (it is now "resumed").
}

@Override
protected void onPause() {
    super.onPause();
    // Another activity is taking focus (this activity is about
to be "paused").
}

@Override
protected void onStop() {
    super.onStop();
    // The activity is no longer visible (it is now "stopped").
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about to be destroyed.
}
}

```

注意：在做任何事情之前，实现这些生命周期方法，必须始终调用父类的实现，如上面的例子所示。

综合来看，这些方法定义一个Activity的整个生命周期。通过实现这些方法，您可以监视Activity生命周期的三个嵌套循环：

- Activity的整个存在周期发生在 [OnCreate \(\)](#) 调用和 [OnDestroy \(\)](#) 调用之间。Activity调用 [OnCreate \(\)](#) 执行“全局”状态设置（如定义布局），并调用 [OnDestroy \(\)](#) 释放所有剩余资源。例如，如果Activity有一个线程在后台运行，从网络上下载数据，它可能会调用 [OnCreate \(\)](#) 创建该线程，然后由 [OnDestroy \(\)](#) 停止线程的。
- Activity的可见周期发生在 [OnStart \(\)](#) 调用和 [onStop \(\)](#) 调用之间。在这段时间内，用户可以看到屏幕上的Activity，并与它进行交互。例如，一个新的Activity时启动时，[onStop \(\)](#) 被调用，这时Activity不Activity

再可见。这两种方法之间，你可以维持运行所需要的资源，提供给用户。例如，你可以在调用**OnStart ()** 注册 **BroadcastReceiver** 监测影响你用户界面的变化，当用户可以不再看到你内容时，在 **onStop ()** 时注销。在整个存在周期的Activity，Activity在可见和不可见的变化中，系统可能会多次调用**OnStart ()** 和 **OnStop ()**。

- Activity的前台周期发生在 **onResume()** 调用和 **onPause()** 调用之间。在这段时间内，该Activity显示在屏幕上，在所有其他Activity之前，具有用户输入焦点。一个Activity经常在前台后台之间转换，例如，当设备进入睡眠状态，或弹出一个对话框是 **onPause()** 被称调用。因为这种状态转换，在这两种方法中的代码应该是相当轻量级的，以避免缓慢的转换，使用户等待。

图1说明了这些循环和Activity可能采取状态之间转换的路径。矩形代表可以实现在Activity状态转化之间要执行操作的回调方法。



图1.Activity周期。

同样表1列出了所有的生命周期回调方法，描述了每一个回调方法细节和在Activity的整个生命周期内的位置，包括系统完成回调方法后是否能杀死Activity。

表1.Activity周期的回调方法的总结。

方法	说明	之后是否 可Kill?	之后调用
	style="border: 2px #CCC solid; border- style: solid; border-		

	width: 1px; margin:1em 1em 1em 1em; border- collapse:collapse; font- size:1em; empty- <u>onCreate()</u> cells:show" Activity第一次被创建时调用。通常你在此处处理静态设置-比如创建视图，数据绑定到列表等。如果状态可被捕获，一个Bundle包含Activity的以前的状态的对象将传递给这种方法将个（见后面的 保存Activity状态 ）。之后是onStart()	否	onStart()
<u>onRestart()</u>	Activity停止之后，启动之前调用	否	onStart()
<u>onStart()</u>	当Activity将要可见的时候调用，如果之后前台可见，调用 onResume，如果不可见调用 onStop	否	onResume() or onStop()
<u>onResume()</u>	在Activity和将要和用户交互的时候调用，此时Activity位于Activity 栈的栈顶，并且正在获取用户的输入设备，之后是调用OnPause	否	onPause()
	当系统恢复另外一个Activity时调用他，通常用于保存没有保存的		

<u>onPause()</u>	数据，停止消耗CPU的动画处理和其他事情，只有当他返回后，其他的 Activity 才能恢复，因此他调用时间很短。之后如果另外一个Activity返回到前台，将调用onResume, 否者将调用onStop();	是	onResume() or onStop()
<u>onStop()</u>	当Activity对用户不再可见时调用，发生在当另外一个已经存在或者新的Activity恢复并覆盖他的时候，要不就是在自己被销毁的时候，之后如果Activity又恢复到前台并能与用户交互，调用onStart ()，否者调用onDestory ()。	是	onRestart() or onDestory()
<u>onDestory()</u>	当销毁Activity时候调用，这是Activity最后能接受到得调用方法，要不是由Activity已经完成(有 <u>finish()</u> 调用)，或者是系统由于节省资源的需要零时销毁Activity，你可以通过 <u>isFinishing()</u> 方法来判断这两种情况是哪一种。	是	<i>nothing</i>

列标中的“之后是否可Kill？”表示在功能返回后系统可以随时Kill Activity的主进程，不会执行Activity的其他代码。表中有三种方法填的“是”：

(onPause() , onStop() , 和 onDestory())。因为 onPause() 是三

个方法中最先被调用的，Activity创建后，[onPause\(\)](#)是在系统能kill进程之前保证能被调用到得方法——如果系统在紧急情况下必须恢复内存，之后[onStop\(\)](#)和[onDestroy\(\)](#)可能不会被调用，因此，你应该使用[onPause\(\)](#)写入些持久性数据（如存储用户正在编辑的数据）。然而，对于要保存的数据，你应该有选择性，因为任何阻塞方法将阻塞系统转到下一个Activity，降低了用户体验。

“之后是否可Kill？”列中被标记为“否”的方法，表示此时系统会保护Activity的主进程不被kill。因此，Activity在从[onPause\(\)](#)返回到[onResume\(\)](#)被调用的这段时间是可被kill的。在下次[onPause\(\)](#)被调用并返回之前，将不能被kill。

注：对于Activity，表1中的定义，技术上来说并不是“可kill”，Activity仍然可以被系统“kill”，但仅在系统资源匮乏的极端情况下才会发生。Activity何时能被kill，更多信息在[进程和线程 - processes and threads](#) 文档中讨论。

保存Activity状态-Saving activity state

在管理Activity生命周期中提到当Activity暂停或者停止的时候，会保留Activity状态，这是成立的，因为当他暂停或者停止的时候，内存中仍然有[Activity](#)对象，并且包含成员信息和当前的状态，因此，在用户模式下的任何对Activity的操作，在Activity返回到前台的时候将被还原（“恢复状态”）。

然而，系统为了回收资源而销毁[Activity](#)，这时系统不能简单地恢复到之前的完整状态。相反，如果用户返回到Activity，系统必须重新创建[Activity](#)对象。此时，用户并不关系系统销毁Activity，并重新创建它。用户此时更关心Activity是否能恢复到所期望的状态。在这种情况下，要保证有关Activity状态的重要信息是通过实现一个叫做[onSaveInstanceState \(\)](#) 辅助回调函数来完成的，它允许你保存Activity的状态信息。

在Activity变得不太稳定的前，系统调用[onSaveInstanceState \(\)](#)。该

系统传递给其一个 **Bundle**，您可以在其中以名称-值对的形式保存为有关Activity的状态信息，如使用方法 **putString ()** 和 **putInt ()**。然后，如果系统kill你的应用进程，当用户返回Activity，系统重建Activity时，系统会同时传给 **onCreate()** 和 **onRestoreInstanceState() Bundle**。使用这两种方法，你可以从 **Bundle** 提取到保存的状态信息来恢复Activity。如果没有恢复的状态信息，**Bundle**传递的是空（首次创建Activity的情况）。



图2。按照状态的完整性，Activity有两种方式返回到用户的焦点：要么Activity被销毁，然后重新创建Activity，必须要恢复以前保存的状态。或停止Activity，然后恢复，这时Activity保持状态的完整。

注：Activity被销毁之前不保证 **onSaveInstanceState ()** 会调用，因为有情况下，它不需要保存状态（例如，当用户使用“后退”按钮离开Activity的时候，因为用户已明确关闭的Activity）。如果是系统调用 **onSaveInstanceState ()**，那么他会在 **onStop ()** 前也可能在 **onPause()** 前调用。

然而，即使你什么也不做，也不实现 **onSaveInstanceState ()**，某些Activity状态恢复会使用 **Activity**类的默认 **onSaveInstanceState ()** 实现。具体来说，默认实现对于布局中的每个 **视图 - View** 调用相应的**onSaveInstanceState ()** 方法，它允许每个视图提供保存自身的相关信息。几乎在Android框架每一个部件都会适当的实现此方法，这样，当重新创建Activity时候任何UI的变化将自动保存和恢复。例如，**EditText** 控件保存由用户输入和任何文字，**CheckBox**控件保存它是否检查。对于你要保存其状态每一个部件，你所需要做的唯一的工作是提供一个唯一的ID（使用 **layout-resource.html#idvalue android:id** 属性）。一个组件，如果没有一个ID，系统无法保存其状态。

虽然默
认**onSaveInstanceState ()**
实现保存有关Activity UI的有用信
息，你仍可能需要重写它，以保
存更多的信息。例如，

您还可以显式阻止布局视图保存其
状态，通过设置
android:saveEnabled 属性
为“false”或调用
setSaveEnabled () 方法。通常

在Activity中你可能需要在成员变量值发生改变时保存他们。（可能在UI恢复时需要关联这些值，但默认情况下，拥有UI值的成员都不会被还原）。

情况下，你不应该禁他，但如果想使你的Activity恢复状态时有不同UI效果就启用它。

由于[onSaveInstanceState \(\)](#) 的默认实施能保存UI的状态，如果要为保存额外的状态信息而重写此方法，那么在做任何工作之前，你一定要在实现[onSaveInstanceState \(\)](#) 之前调用其超类方法。同样，如果你重写它，你也应该调用 [onRestoreInstanceState \(\)](#) 实现的超类方法，以此默认实现恢复视图状态。

注：因为 [onSaveInstanceState \(\)](#) 不能保证被调用，所以当用户离开Activity，你应该只使用它记录Activity的瞬时状态（UI的状态），你不应该用它来存储持久数据。相反，你应该使用 [onPause\(\)](#) 存储持久数据（如保存到数据库中的数据）。

测试您的应用程序恢复其状态的能力的一个好方法是简单地旋转装置，使屏幕的方向变化。当屏幕方向的变化，系统的销毁并重新创建Activity，对于新的屏幕配置替换资源。因此，当被重建时，您的Activity完全恢复其状态时就非常重要了，因为用户经常在使用应用程序时旋转屏幕。

处理配置更改-**Handling configuration changes**

某些设备配置在运行时可以改变（如屏幕方向，键盘的可用性，和语言）。当这种变化发生时，Android重新运行Activity（系统调用的[onDestroy\(\)](#)，然后立即调用的[onCreate \(\)](#)）。这种行为旨在帮助您用您所提供的（如不同的屏幕方向和大小不同的布局）的替代资源进行应用程序自动重载，以适应新的配置。

如果你正确地设计Activity，以处理由于屏幕方向的变化带来的重新启动并按照如上所述恢复Activity状态，那么您的应用程序在Activity生命周期中对于其他突发事件的应对将更具弹性。

处理重新启动来保存和恢复Activity状态的一个的最佳方式是使用在上一节讨论的[onSaveInstanceState \(\)](#) 和[onRestoreInstanceState \(\)](#)（或[onCreate \(\)](#)）。

更多运行时设备配置的改变，以及如何处理它们信息，请阅读 [处理运行时更改 - runtime-changes](#) 入门。

协调Activity-Coordinating activities

当一个Activity启动另外一个Activity时，他们都经历生命周期的转换。当创建其他Activity时，第一个Activity暂停和停止（不过，如果它仍然在后台可见，它不会停止。）。这些Activity共享保存在磁盘上的数据或其他内容，重要的是要了解在第二个Activity没有被建立之前，第一项Activity是不会完全停止的。而不是，启动第二个Activity进程覆盖第一个Activity进程。

生命周期很好的定义了回调顺序，特别是在同一进程中两个Activity中，一个启动另外一个的时候。如下是当Activity A 启动Activity B是产生的操作顺序：

1. ActivityA执行 [onPause \(\)](#) 方法。

2. ActivityB 按照 [onCreate \(\)](#) , [OnStart\(\)](#) , [onResume \(\)](#) 的顺序执行方法。（ActivityB现在取得用户的焦点。）

3. 然后，如果ActivityA 已不再是显示在屏幕上，它执行方法 [onStop \(\)](#)。

这个可预测的生命周期回调顺序，可让您管理的从一个Activity到另一个Activity的转换信息。例如，如果第一个Activity停止时你必须写数据库，让之后的Activity 读取数据库，那么你应该在 [onPause \(\)](#) 期间写数据库而不是 [onStop \(\)](#)。

来自 "[index.php?title=Activities&oldid=13720](#)"

1个分类: [Android Dev Guide](#)



Fragments

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： sfshine

原文链

接：

<http://developer.android.com/guide/topics/fundamentals/fragments.html>

目录

[[隐藏](#)]

[1 Fragments \(碎片\)](#)

- [1.1 Design Philosophy-设计理念](#)
- [1.2 Creating a Fragment-创建一个Fragment](#)
 - [1.2.1 Adding a user interface-添加一个用户接口](#)
 - [1.2.2 Adding a fragment to an activity-给一个Activity添加一个Fragment](#)
 - [1.2.3 Adding a fragment without a UI-添加一个没有UI的Fragment](#)
- [1.3 Managing Fragments-管理fragment](#)
- [1.4 Performing Fragment Transactions-执行Fragment事务.](#)
- [1.5 Communicating with the Activity-与Activity的通讯](#)
- [1.6 Creating event callbacks to the activity-为Activity创建时间回调](#)
 - [1.6.1 Adding items to the Action Bar-在Action Bar上添加项.](#)
- [1.7 Handling the Fragment Lifecycle-处理Fragment的生命周期](#)
 - [1.7.1 Coordinating with the activity lifecycle-和Activity生命周期的协调](#)

Fragments (碎片)

一个碎片在一个活动中代表一个行为或用户界面的一部分。你可以在一个单一的[活动中](#)组合使用多个碎片以建立一个多窗格的UI，并且可以在多个活动中重用一个碎片。你可以认为是一个拥有独立生命周期、能够独立接受输入事件、并且可以在活动运行时添加或移除的碎片作为一个活动的模块化部分（有点像一个你可以在不同活动中重用的子活动）。

一个碎片必须总是嵌入到一个活动（activity）中，并且它的生命周期直接受到活动得生命周期的影响。例如：当活动暂停或销毁时时，它里边的所有碎片也是如此。然而，当一个活动运行时，（它在 `resumed`(重新开始)[生命周期状态](#)），你可以单独的操作每个碎片，例如添加或移除它们。当你执行了这样的碎片事务，你也可以将它添加到一个后台堆栈所管理的活动——该活动中的每个后台堆栈条目都记录着已发生的片段事务。后台堆栈允许用户通过按返回按钮退一个碎片事务（向后导航）。

When you add a fragment as a part of your activity layout, it lives in a [ViewGroup](#) inside the activity's view hierarchy and the fragment defines its own view layout. You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a `<fragment>` element, or from your application code by adding it to an existing [ViewGroup](#). However, a fragment is not required to be a part of the activity layout; you may also use a fragment without its own UI as an invisible worker for the activity.

当你增加一个Fragment来作为Activity界面布局的一部分时,他存在在Activity视图层里面的一个ViewGroup 中,这个Fragment定义了自己的视图.你可以通过在Activity的布局文件中用`<fragment>`声明这个fragment来把 这个fragment插入你的activity中,或者你可以在应用的代码中把他添加添加到一个现存的ViewGroup中.然而,Fragment不必 是一个activity布局的一部分,你可以使用一个Fragment而不实用他的UI,这样可以让Fragment作为Activity的一个不可见部 分来工作.

This document describes how to build your application to use fragments, including how fragments can maintain their state when added to the

activity's back stack, share events with the activity and other fragments in the activity, contribute to the activity's action bar, and more.

这个文档介绍了增加使用Fragment创建你的应用,包括怎样使fragment在增加到返回栈的时候保持他们的状态,和activity以及该activity的其他fragment的共享事件,组成Activity的ActionBar,等等.

Design Philosophy-设计理念

Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets. Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components. Fragments allow such designs without the need for you to manage complex changes to the view hierarchy. By dividing the layout of an activity into fragments, you become able to modify the activity's appearance at runtime and preserve those changes in a back stack that's managed by the activity.

Android在Android3.0(API等级11)中引入了Fragment,主要是为了在大屏幕上(比如平板)支持更多动态的灵活 的UI设计.因为平板的屏幕比其他手持设备大多了,有更多的空间来组合,交换UI组件.Fragment使你在View层不必进行很复杂变化就可以实现这些设计.通过把Activity的布局分解成很多Fragment,你可以在运行时改动activity的界面并且可以把这些变化保存在activity管理的返回栈中.

For example, a news application can use one fragment to show a list of articles on the left and another fragment to display an article on the right—both fragments appear in one activity, side by side, and each fragment has its own set of lifecycle callback methods and handle their own user input events. Thus, instead of using one activity to select an article and another activity to read the article, the user can select an article and read it all within the same activity, as illustrated in the tablet layout in figure 1.

比如,一个新闻应用可以使用一个Fragment在左边显示一列文章标题而在左边的另一个Fragment显示文章详细内容.这两个 Fragment都在同一个Activity中,他们并排着,每个Fragment有他自己的生命周期回调方法,处理他们各自的输入事件.那么,不需要在一个activity中选择在另一个activity中阅读,用户可以选择一篇文章在同一个activity中阅读这个新闻的内容.如图1所示:

You should design each fragment as a modular and reusable activity component. That is, because each fragment defines its own layout and its own behavior with its own lifecycle callbacks, you can include one fragment in multiple activities, so you should design for reuse and avoid directly manipulating one fragment from another fragment. This is especially important because a modular fragment allows you to change your fragment combinations for different screen sizes. When designing your application to support both tablets and handsets, you can reuse your fragments in different layout configurations to optimize the user experience based on the available screen space. For example, on a handset, it might be necessary to separate fragments to provide a single-pane UI when more than one cannot fit within the same activity.

你应该把fragment设计成模块化的,可复用的Activity组件.就是说,每个Fragment定义了他自己的布局和他自己的,拥有自己生命周期回调的行为,你可以在多个activity中包含一个Fragment,所以你应该把Fragment设计成可以复用的,并且需要避免从一个fragment直接操纵另一个fragment.这一点是非常重要的因为一个fragment模块允许你针对不同的屏幕尺寸变化你的fragment 组合形式.在设计应用来支持平板和手持设备时,你可以在不同的布局配置中重用你的fragment来针对屏幕空间优化用户体验.比如,在一个手持设备上,可能需要分开的fragment来提供一个单独窗口UI而不是使很多fragment在同一个activity中放不开.



Figure 1. An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

图例1 平板上,在一个activity中Fragment怎么定义两个UI模块,但是在手持设备上他们将分开.

For example—to continue with the news application example—the application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article. Thus, the application supports both tablets and handsets by reusing fragments in different combinations, as illustrated in figure 1.

继续以上的新闻为例,当在平板大小的设备上运行的时候,这个应用可以在Activity中嵌入两个Fragment.然而,在手持设备上的时候,由于没有足够的空间盛放这两个fragment,所以ActivityA只显示了其中的一个Fragment(新闻列表),当用户选择新闻标题的时候,他跳转到ActivityB,ActivityB中显示第二个Fragment(新闻详细信息).那么通过复用不同组合的Fragment这个应用就可以同时支持平板和手持设备了,如图1.

For more information about designing your application with different fragment combinations for different screen configurations, see the guide to Supporting Tablets and Handsets.

更多关于使用不同的Fragment组合来设计适应不同屏幕应用的信息,请参阅Supporting Tablets and Handsets一章.

Creating a Fragment-创建一个Fragment

To create a fragment, you must create a subclass of Fragment (or an existing subclass of it). The Fragment class has code that looks a lot like an Activity. It contains callback methods similar to an activity, such as `onCreate()`, `onStart()`, `onPause()`, and `onStop()`. In fact, if you're converting an existing Android application to use fragments, you might simply move code from your activity's callback methods into the respective callback methods of your fragment.

创建一个Fragment,你应该创建一个Fragment的子类(或者他的一个现有子类).Fragment类的代码很像 Activity.它还有和activity相似的回调方法,比如onCreate(), onStart(), onPause(), 和 onStop().实际上,如果你在使用Fragment来转换一个现成的应用,你可能只是简单的从你的activity回调方法中移动代码到 fragment相应的回调方法中.

Usually, you should implement at least the following lifecycle methods: 一般的,你至少应该实现下面的生命周期方法:

onCreate()

The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

创建fragment的时候,系统会调用这个方法.在你实现过程中,当fragment暂停(pause),停止(stop)然后恢复(resume)时,你应该初始化你想要保持的,fragment的必要的组件.

onCreateView()

The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.

在fragment第一次绘制他的用户界面的时候系统会调用这个方法.如果你想为你的fragment绘制界面,你必须从这个方法中返回一个View,这个View是你fragment布局的基础.如果这个Fragment不提供UI,你可以返回空.

onPause()

The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment

is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

系统调用这个方法作为用户离开这个fragment的第一标志(虽然这不总是意味着这个Fragment被摧毁了).通常是你需要做一些改变,这些改变超出了当前的用户会话(因为用户有可能不会回到这个界面来).

Most applications should implement at least these three methods for every fragment, but there are several other callback methods you should also use to handle various stages of the fragment lifecycle. All the lifecycle callback methods are discussed more later, in the section about Handling the Fragment Lifecycle. 大多数应用至少需要对碎片实现这三个方法, 但你也可以使用其他的反馈方式来控制片段生存周期的不同阶段。想获取更多有关生命周期回调方法的详细信息, 请参 照Handling the Fragment Lifecycle部分。

There are also a few subclasses that you might want to extend, instead of the base Fragment class:

也有有些子类(不是基本的Fragment类)你可能想要继承来实现Fragment:

DialogFragment

Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the Activity class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.

显示一个浮动的对话框. 使用这个类来创建一个对话框是和使用对话Helper方法在Activity类中创建对话框都是很好的方法, 因为你可以把Fragment对话框包含在activity管理的Fragment返回栈中, 允许用户返回到关闭的Fragment中.

ListFragment

Displays a list of items that are managed by an adapter (such as a

`SimpleCursorAdapter`), similar to `ListActivity`. It provides several methods for managing a list view, such as the `onListItemClick()` callback to handle click events.

展示一列被adapter(比如`SimpleCursorAdapter`)管理的项,和`ListActivity`很相似.它提供了一些管理一个列表视图的方法,比如处理点击事件的`onListItemClick()`方法.

PreferenceFragment

Displays a hierarchy of Preference objects as a list, similar to `PreferenceActivity`. This is useful when creating a "settings" activity for your application.

用一个列表来显示一组偏好设置对象,类似于`PreferenceActivity`. 在创建设置型的activity时会用到.



Figure 2. The lifecycle of a fragment (while its activity is running).

图2 Fragment的生命周期.

Adding a user interface-添加一个用户接口

A fragment is usually used as part of an activity's user interface and contributes its own layout to the activity.

一个Fragment经常被用作activity界面的一部分,为activity贡献自己的界面.

To provide a layout for a fragment, you must implement the `onCreateView()` callback method, which the Android system calls when it's time for the fragment to draw its layout. Your implementation of this method must return a View that is the root of your fragment's layout.

为了给fragment提供一个布局,你必须实现`onCreateView()`方法,Android系统在Fragment绘制他的界面的时候调用这个方法.你对这个方法的实现必须返回一个View,这个View是你Fragment布局的基础.

Note: If your fragment is a subclass of ListFragment, the default implementation returns a ListView from onCreateView(), so you don't need to implement it. 注意:如果你的Fragment是一个ListFragment类的子类,默认会从onCreateView()返回一个Listview,所以你不需要实现它.

To return a layout from onCreateView(), you can inflate it from a layout resource defined in XML. To help you do so, onCreateView() provides a LayoutInflater object.

为了从onCreateView()方法返回一个布局,你可以用一个xml布局文件来填充它.为了帮助你做这个事情, onCreateView() 方法提供了一个LayoutInflater对象.

For example, here's a subclass of Fragment that loads a layout from the example_fragment.xml file:

比如,这个一个Fragment的子类,他是从example_fragment.xml文件载入的布局:

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment,
                           container, false);
    }
}
```

Creating a layout-创建一个布局

In the sample above, R.layout.example_fragment is a reference to a layout resource named example_fragment.xml saved in the application resources. For information about how to create a layout in XML, see the User Interface documentation.

和上面差不多,R.layout.example_fragment是在系统保存的example_fragment.xml这个布局资源的引用.更多关于使用一个xml文件创建一个布局的信息,参考User Interface 文档.

The container parameter passed to onCreateView() is the parent ViewGroup (from the activity's layout) in which your fragment layout will be inserted. The savedInstanceState parameter is a Bundle that provides data about the previous instance of the fragment, if the fragment is being resumed (restoring state is discussed more in the section about Handling the Fragment Lifecycle).

传递给onCreateView()的容器参数是fragment锁插入的activity的父ViewGroup(来自对应的 activity布局).savedInstanceState的参数是一个提供关于之前Fragment状态数据的Bundle,如果这个 Fragment被恢复了(resume,恢复数据在处理Fragment生命周期这一节有更多介绍)

The inflate() method takes three arguments:

inflate()方法接收三个参数:

- The resource ID of the layout you want to inflate.
- 你想要添加的layout的资源ID.
- The ViewGroup to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going.
- 将作为填充布局的父容器的ViewGroup.传递容器参数是非常重要的,只用这样才能使系统应用布局参数到填充视图的根视图,从而被它的父视图所确定.

- A boolean indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation. (In this case, this is false because the system is already inserting the inflated layout into the container—passing true would create a redundant view group in the final layout.)
- 一个boolean类型的参数,用于在填充时指明填充的布局是否应该附加在ViewGroup(第二个参数)上.(如果系统已经插入这个填充布局到容器了就返回false,如果将要在最终布局中创建一个多余的viewgroup,那就返回true)

Now you've seen how to create a fragment that provides a layout. Next, you need to add the fragment to your activity.

Adding a fragment to an activity-给一个Activity添加一个Fragment

Usually, a fragment contributes a portion of UI to the host activity, which is embedded as a part of the activity's overall view hierarchy. There are two ways you can add a fragment to the activity layout:

一般的一个fragment提供了Activity UI的一部分,他作为Activity全局视图层的一部分而嵌入.有两种方法可以把fragment嵌入到Activity布局中:

***Declare the fragment inside the activity's layout file**-在Activity布局文件中声明Fragment.

In this case, you can specify layout properties for the fragment as if it were a view. For example, here's the layout file for an activity with two fragments:

这样的话,你可以把Fragment当作一个视图,比如,这是一个嵌入两个Fragmet的Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```

    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<fragment android:name="com.example.news.ArticleListFragment"
    android:id="@+id/list"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="match_parent" />
<fragment android:name="com.example.news.ArticleReaderFragment"
    android:id="@+id/viewer"
    android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="match_parent" />
</LinearLayout>

```

The android:name attribute in the <fragment> specifies the Fragment class to instantiate in the layout. <fragment>的android:name属性指定了在布局中要实例化的Fragment类。

When the system creates this activity layout, it instantiates each fragment specified in the layout and calls the onCreateView() method for each one, to retrieve each fragment's layout. The system inserts the View returned by the fragment directly in place of the <fragment> element. 当系统生成这个activity布局时，会把布局中每个特定的片段实例化，然后依次调用onCreateView () 方法以便检索每个片段布局。系统直接 插入片段返回的视图来代替<fragment>元素。

Note: Each fragment requires a unique identifier that the system can use to restore the fragment if the activity is restarted (and which you can use to capture the fragment to perform transactions, such as remove it). There are three ways to provide an ID for a fragment: 在系统创建这个Activity布局的时候,他会实例化每个布局中的每个fragment,调用每个fragment的onCreateView()方法 来取回每个fragment的视图.系统把fragment返回的视图直接插入到<fragment>标签所在的地方。

- Supply the android:id attribute with a unique ID.
- 使用android:id来指定它唯一的ID.
- Supply the android:tag attribute with a unique string.
- 使用android:tag来指定一个唯一的字符串标志

If you provide neither of the previous two, the system uses the ID of the container view.

- 如果上面两个你都不指定,系统会使用容器视图的ID.

- Or, programmatically add the fragment to an existing ViewGroup.
- 或者,机械性的把fragment添加到ViewGroup中.

At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a ViewGroup in which to place the fragment. To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction. You can get an instance of FragmentTransaction from your Activity like this:

在任何你Activity运行的时候,你都可以把fragment添加到Activity的视图中.你只需要指定一个用于盛放 Fragment的ViewGroup.为了让fragment可以被管理(比如添加,删除,替换fragment),你必须使用来自FragmentTransaction的API.你可以像下面这样在Activity中获取一个FragmentTransaction的实例:

```
FragmentManager fragmentManager = getSupportFragmentManager()
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
```

You can then add a fragment using the add() method, specifying the fragment to add and the view in which to insert it. For example:

你可以使用Add()方法添加一个Fragment,指定要添加的Fragment和目标View,如下:

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

The first argument passed to add() is the ViewGroup in which the fragment should be placed, specified by resource ID, and the second parameter is the fragment to add. Once you've made your changes with FragmentTransaction, you must call commit() for the changes to take effect.

`add()`方法中的第一个参数是Fragment所要放置的目标ViewGroup,通过资源ID指定,第二个参数是要添加的Fragment.只要你使用`FragmentTransaction`做了修改,你必须调用`commit()`方法来使修改生效.

Adding a fragment without a UI-添加一个没有UI的Fragment

The examples above show how to add a fragment to your activity in order to provide a UI. However, you can also use a fragment to provide a background behavior for the activity without presenting additional UI.

上面的例子想你展示了怎么添加一个含有UI的Fragment到你的Activity.然而,对于不想增加而外UI的Activity来说,你也可以使用Fragment来进行后台行为.

To add a fragment without a UI, add the fragment from the activity using `add(Fragment, String)` (supplying a unique string "tag" for the fragment, rather than a view ID). This adds the fragment, but, because it's not associated with a view in the activity layout, it does not receive a call to `onCreateView()`. So you don't need to implement that method.

为了添加一个没有UI的Fragment.需要使用`add(Fragment, String)` 方法,其中,你需要为Fragment提供一个字符串的标志而不是一个视图ID.这样增加的Fragment,由于没有涉及到Activity的视图,所以不会调用`onCreateView()`方法.所以你不需要实现这个方法.

Supplying a string tag for the fragment isn't strictly for non-UI fragments—you can also supply string tags to fragments that do have a UI—but if the fragment does not have a UI, then the string tag is the only way to identify it. If you want to get the fragment from the activity later, you need to use `findFragmentByTag()`.

为Fragment提供一个字符串标志不一定只局限于没有UI的Fragment,你也可以为有UI的Fragment指定一个字符串标志,但是如果这个Fragment真的没有UI,那这个字符串标志是确定它的唯一标志.如果你想在后面从Activity中获取到这个fragment, 你需要使用`findFragmentByTag()`方法.

For an example activity that uses a fragment as a background worker, without a UI, see the `FragmentRetainInstance.java` sample.

在`FragmentRetainInstance.java`文件的例子展示了Activity怎么使用一个没有UI的fragment来出来后台工作.

Managing Fragments- 管理fragment

To manage the fragments in your activity, you need to use `FragmentManager`. To get it, call `getFragmentManager()` from your activity.

为了管理你Activity中的fragment,你需要使用`FragmentManager`.你可以通过你Activity中的`getFragmentManager()`来获取它.

Some things that you can do with `FragmentManager` include:

使用`FragmentManager`你可以:

- Get fragments that exist in the activity, with `findFragmentById()` (for fragments that provide a UI in the activity layout) or `findFragmentByTag()` (for fragments that do or don't provide a UI).
- 使用`findFragmentById()`(提供UI的Fragment)或者`findFragmentByTag()`(没有提供UI的Fragment) 获取你Activity存在的Fragment,
- Pop fragments off the back stack, with `popBackStack()` (simulating a Back command by the user).
- 使用`popBackStack()`把Fragment从返回栈中弹出(模拟用户的返回命令).
- Register a listener for changes to the back stack, with `addOnBackStackChangedListener()`.
- 使用`addOnBackStackChangedListener()`方法为返回栈的变化注册监听器.

For more information about these methods and others, refer to the `FragmentManager` class documentation.

请参考文档的 **FragmentManager** 类来查看过于这些方法(还有其他方法)的更多内容.

As demonstrated in the previous section, you can also use **FragmentManager** to open a **FragmentTransaction**, which allows you to perform transactions, such as add and remove fragments.

正如前面的文档所讲的,你也可以使用**FragmentManager**来打开**FragmentTransaction**,**FragmentTransaction**允许你执行添加,删除**Fragment**的事务.

Performing Fragment Transactions-执行Fragment**事务.**

A great feature about using fragments in your activity is the ability to add, remove, replace, and perform other actions with them, in response to user interaction. Each set of changes that you commit to the activity is called a transaction and you can perform one using APIs in **FragmentTransaction**. You can also save each transaction to a back stack managed by the activity, allowing the user to navigate backward through the fragment changes (similar to navigating backward through activities).

在你的**Activity**中使用**Fragment**的最大好处就是可以针对用户的操作,进行对**Fragment**的添加,移除,替换等等其他操作.你提交给**Activity**的每个变化称为一个事务,这些事务你可以使用**FragmentTransaction**的API来实现.你也可以在**Activity**管理的返回栈中保存每个事务,使用户可以在**Fragment**的变化后返回之前的状态(类似于在**Activity**跳转后的返回).

You can acquire an instance of **FragmentTransaction** from the **FragmentManager** like this:

你可以像这样从**FragmentManager**中取得一个**FragmentTransaction**的实例:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
```

Each transaction is a set of changes that you want to perform at the same time. You can set up all the changes you want to perform for a

given transaction using methods such as add(), remove(), and replace(). Then, to apply the transaction to the activity, you must call commit().

每个事务是一系列你想要同时执行的Fragmen的变化.你可以使用像add(),remove(),replace()这样的方法来为一个事务设定你想要执行的操作.为了使Activity的事务生效,你必须执行commit()方法.

Before you call commit(), however, you might want to call addBackStack(), in order to add the transaction to a back stack of fragment transactions. This back stack is managed by the activity and allows the user to return to the previous fragment state, by pressing the Back button.

在你调用commit()方法之前,为了添加这个事务到一个Fragmen事务的返回栈,你可能想要调用addBackStack()方法.这个返回栈被Activity管理,允许用户通过按下返回按键返回之前的Fragmen状态.

For example, here's how you can replace one fragment with another, and preserve the previous state in the back stack:

这里展示了怎么使用一个Fragmen替换另一个,然后在返回栈中返回到之前的状态.

```
// Create new fragment and transaction
// 创建一个新的Fragmen和事务
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction =
getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this
// fragment,
// 使用这个Fragment替换在Fragmen容器中的Fragmet
// and add the transaction to the back stack
// 添加这个事务到返回栈
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
// 提交这个事务.
transaction.commit();
```

In this example, newFragment replaces whatever fragment (if any) is currently in the layout container identified by the R.id.fragment_container ID. By calling addToBackStack(), the replace transaction is saved to the back stack so the user can reverse the transaction and bring back the previous fragment by pressing the Back button.

在这个例子中,新的Fragment替换了R.id.fragment_container ID指定的布局容器中当前存在的fragment(如果存在的话).通过调用addToBackStack()方法,替换事务被保存在了返回栈中,这样用户可以回退这个事务,通过按下返回键返回到以前的fragment.

If you add multiple changes to the transaction (such as another add() or remove()) and call addToBackStack(), then all changes applied before you call commit() are added to the back stack as a single transaction and the Back button will reverse them all together.

如果你在事务中添加了多个变化(比如另一个add()方法或者remove()方法),然后调用了addToBackStack()方法,那在你调用commit()方法之前的所有变化都会作为单独的事务被添加到返回栈中,返回键将会把他们全部回退.

The order in which you add changes to a FragmentTransaction doesn't matter, except:

除了下面这些,其他的情况和你在FragmentTransaction中添加的顺序没有关系

- You must call commit() last
• 你必须在最后调用commit()方法
- If you're adding multiple fragments to the same container, then the order in which you add them determines the order they appear in the view hierarchy
• 如果你在向同一个容器添加多个fragment,那么你添加的顺序决定了他们在视图层出现的顺序.

If you do not call `addToBackStack()` when you perform a transaction that removes a fragment, then that fragment is destroyed when the transaction is committed and the user cannot navigate back to it. Whereas, if you do call `addToBackStack()` when removing a fragment, then the fragment is stopped and will be resumed if the user navigates back.

如果在你执行一个移除所有fragment的事务的时候没有调用`addToBackStack()`方法,那么这个fragment将会在事务提交后被摧毁,用户不能再返回到之前的fragment.如果你在移除fragment的时候调用了`addToBackStack()`方法,那这个 fragment会被停止,并可以在用户按返回的时候恢复.

Tip: For each fragment transaction, you can apply a transition animation, by calling `setTransition()` before you commit. 小贴士:对于每个fragment事务,你可以在提交之前通过调用`setTransition()`来应用一个fragment动画.

Calling `commit()` does not perform the transaction immediately. Rather, it schedules it to run on the activity's UI thread (the "main" thread) as soon as the thread is able to do so. If necessary, however, you may call `executePendingTransactions()` from your UI thread to immediately execute transactions submitted by `commit()`. Doing so is usually not necessary unless the transaction is a dependency for jobs in other threads.

调用`commit()`方法不能立即执行事务而是安排它运行在Activity的UI线程中("主"线程)---如果这线程可以这么做的话. 如果需要,你可以在你UI线程中调用`executePendingTransactions()`方法来直接执行`commit()`方法提交的事务. 这么多一般不必要除非事务依赖于其他线程的工作.

Caution: You can commit a transaction using `commit()` only prior to the activity saving its state (when the user leaves the activity). If you attempt to commit after that point, an exception will be thrown. This

is because the state after the commit can be lost if the activity needs to be restored. For situations in which it's okay that you lose the commit, use `commitAllowingStateLoss()`.

注意:你只可以在Activity保存他状态之前(在用户离开这个Activity的时候)使用`commit()`方法来提交一个事务.如果你在这个时间点之后提交,系统会抛出一个异常.这是因为如果Activity需要恢复,在提交之后的状态可能会丢失.对于允许丢失提交的情况,请使用`commitAllowingStateLoss()`方法.

Communicating with the Activity-与Activity的通讯

Although a Fragment is implemented as an object that's independent from an Activity and can be used inside multiple activities, a given instance of a fragment is directly tied to the activity that contains it.

即使fragment是作为一个object实现的,独立于Activity的并且可以在那多个Activity中使用,但是一个fragment实例还是和它所在的容器有直接的关系.

Specifically, the fragment can access the Activity instance with `getActivity()` and easily perform tasks such as find a view in the activity layout:

特别的,fragment可以通过`getActivity()`方法来访问Activity实例并可以轻易的执行像在activity视图中查找View的任务.

```
View listView = getActivity().findViewById(R.id.list);
```

Likewise, your activity can call methods in the fragment by acquiring a reference to the Fragment from FragmentManager, using `findFragmentById()` or `findFragmentByTag()`. For example:

同样的,使用`findFragmentById()`或`findFragmentByTag()`通过从FragmentManager获取一个对这个Fragment的引用,你的Activity可以调用fragment中的方法

```
ExampleFragment fragment = (ExampleFragment)
```

```
getFragmentManager().findFragmentById(R.id.example_fragment);
```

Creating event callbacks to the activity-为Activity创建时间回调

In some cases, you might need a fragment to share events with the activity. A good way to do that is to define a callback interface inside the fragment and require that the host activity implement it. When the activity receives a callback through the interface, it can share the information with other fragments in the layout as necessary.

在一些情况下,你可能需要一个Fragment和Activity共享事件.一个好的方法是在Fragment中定义一个回调接口然后让承载他的Activity实现它.当Activity通过接口接收到调用时,必要时他可以和视图中的其他Fragment共享信息.

or example, if a news application has two fragments in an activity—one to show a list of articles (fragment A) and another to display an article (fragment B)—then fragment A must tell the activity when a list item is selected so that it can tell fragment B to display the article. In this case, the OnArticleSelectedListener interface is declared inside fragment A:

举个例子,如果一个新的应用在一个Activity中有两个Fragment,一个显示一列文章标题(FragmentA),另一列显示文 章内容(FragmentB),那么在一列被选中的时候,FragmentA必须告诉Activity那一列被选中了,这样Activity就可以告诉 FragmentB显示哪一篇文章.在这种情况下,OnArticleSelectedListener 接口会在FragmentA中声明.

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

Then the activity that hosts the fragment implements the OnArticleSelectedListener interface and overrides onArticleSelected() to notify fragment B of the event from fragment A. To ensure that the host

activity implements this interface, fragment A's onAttach() callback method (which the system calls when adding the fragment to the activity) instantiates an instance of OnArticleSelectedListener by casting the Activity that is passed into onAttach():

然后承载Fragment的Activity实现OnArticleSelectedListener接口并重写onArticleSelected()方法来通知FragmentB响应FragmentA的事件.为了保证这个Activity实现了这个接 口,FragmentA的onAttach()方法(系统在添加Fragment到这个Activity的时候调用)通过把Activity参数传递到onAttach()方法传递实例化一个OnArticleSelectedListener实例.

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement OnArticleSelectedListener");
        }
    }
    ...
}
```

If the activity has not implemented the interface, then the fragment throws a ClassCastException. On success, the mListener member holds a reference to activity's implementation of OnArticleSelectedListener, so that fragment A can share events with the activity by calling methods defined by the OnArticleSelectedListener interface. For example, if fragment A is an extension of ListFragment, each time the user clicks a list item, the system calls onListItemClick() in the fragment, which then calls onArticleSelected() to share the event with the activity:

如果Activity没有实现这个接口,那么Fragment会抛出ClassCastException异常.上面的成功例子 中,mListener成员有一个Activity实现的OnArticleSelectedListener的引用.这样FragmentA可以通过调用OnArticleSelectedListener接口定义的方法来共享事件.比如:如果FragmentA是listFragment的扩展,用户 每次点击list的一项,系统会调用Fragment的onListItemClick()方法,然后调用onArticleSelected() 方法来

和Activity分享事件信息.

```

public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position,
    long id) {
        // Append the clicked item's row ID with the content
        // provider Uri
        Uri noteUri =
        ContentUris.withAppendedId(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host activity
        mListener.onArticleSelected(noteUri);
    }
    ...
}

```

The id parameter passed to onListItemClick() is the row ID of the clicked item, which the activity (or other fragment) uses to fetch the article from the application's ContentProvider.

onListItemClick()方法传递的参数是点击项的行ID,Activity(或Fragment)可以用它来从应用的ContentProvider填充文章信息.

More information about using a content provider is available in the Content Providers document. 更多关于使用content provider的信息请参阅Content Providers文档.

Adding items to the Action Bar-在Action Bar上添加项.

Your fragments can contribute menu items to the activity's Options Menu (and, consequently, the Action Bar) by implementing onCreateOptionsMenu(). In order for this method to receive calls, however, you must call setHasOptionsMenu() during onCreate(), to indicate that the fragment would like to add items to the Options Menu (otherwise, the fragment will not receive a call to onCreateOptionsMenu()).

你的Fragment可以通过实现onCreateOptionsMenu()来为Activity的Options Menu创建菜单项(结果就是形成ActionBar).为了让这个方法接收到调用,你必须在onCreate()方法中调用 setHasOptionsMenu() 方法来表明这个Fragment允许在Options Menu中增加项(否则,Fragment将不能接

收onCreateOptionsMenu()的调用).

Any items that you then add to the Options Menu from the fragment are appended to the existing menu items. The fragment also receives callbacks to onOptionsItemSelected() when a menu item is selected.

你从Fragment 添加到 Options Menu的任何项都是现存菜单项的附加项.在一个菜单项选中的时候,Fragment也接响应onOptionsItemSelected()方法的调用.

You can also register a view in your fragment layout to provide a context menu by calling registerForContextMenu(). When the user opens the context menu, the fragment receives a call to onCreateContextMenu(). When the user selects an item, the fragment receives a call to onContextItemSelected().

你也可以在你的Fragment视图中通过调用registerForContextMenu()方法来注册一个视图,从而提供一个上下文 菜单.当用户打开上下文菜单时,Fragment会接收一个onCreateContextMenu()的调用,当用户选择一项的时候,Fragment 接收一个onContextItemSelected()的调用.

Note: Although your fragment receives an on-item-selected callback for each menu item it adds, the activity is first to receive the respective callback when the user selects a menu item. If the activity's implementation of the on-item-selected callback does not handle the selected item, then the event is passed to the fragment's callback. This is true for the Options Menu and context menus. 注意.即使你的Fragment在每个添加的菜单项接收了一个on-item-selected调用,在用户选择一个菜单项的时候,Activity是第一个接受各自调用的组件.如果Activity实现的on-item-selected调用没有处理选择项后的事件,那这个事件会传递到Fragment 的回调中.这对 Options Menu 和上下文菜单都是适用的.

For more information about menus, see the Menus and Action Bar developer guides. 更多关于菜单的信息,参考Menus and Action Bar一文.

Handling the Fragment Lifecycle-处理Fragment的生命周期



Figure 3. The effect of the activity lifecycle on the fragment lifecycle. 图三 Activity生命周期对Fragment生命周期的影响

Managing the lifecycle of a fragment is a lot like managing the lifecycle of an activity. Like an activity, a fragment can exist in three states:

出来Fragment的生命周期和处理Activity的生命周期很相似.和Activity一样,Fragment的生命周期有以下三个状态:

Resumed

The fragment is visible in the running activity.
Fragment在运行中的Activity中可见

Paused

Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).

另一个Activity在前台或者获得了焦点,但是Fragment所在的Activity仍然可以看到(可能是前台Activity占据了屏幕的一部分或者是半透明的)

Stopped

The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive (all state and member information is retained by the system). However, it is no longer visible to the user and will be killed if the activity is killed.

Fragment不可见.宿主Activity可能已经被停止了或者这个Fragment已经从这个Activity中移除了并被添加到了返回栈.一个停止的Fragment仍然是存活的(所有的状态和成员信息被系统保存着).然而

他不再对Activity可见,如果宿主 Activity被杀死了,他也会被杀死.

Also like an activity, you can retain the state of a fragment using a Bundle, in case the activity's process is killed and you need to restore the fragment state when the activity is recreated. You can save the state during the fragment's onSaveInstanceState() callback and restore it during either onCreate(), onCreateView(), or onActivityCreated(). For more information about saving state, see the Activities document.

和Activity一样,在这个Activity所在的进程被杀死或者你需要在Activity重新创建的时候保存Fragment的状态,你可以用Bundle来做这个工作.你可以在Fragment执行onSaveInstanceState()方法的时候保存它的状态,然后在onCreate()或者onCreateView(),onActivityCreated()方法的时候恢复这些状态.更多关于保存状态的内容参考 Activity文档.

The most significant difference in lifecycle between an activity and a fragment is how one is stored in its respective back stack. An activity is placed into a back stack of activities that's managed by the system when it's stopped, by default (so that the user can navigate back to it with the Back button, as discussed in Tasks and Back Stack). However, a fragment is placed into a back stack managed by the host activity only when you explicitly request that the instance be saved by calling addBackStack() during a transaction that removes the fragment.

Activity和Fragment最大的不同是他们在返回栈中的存在形式.默认的,Activity在停止的时候,是放在一个被系统管理的返回栈中(这样用户可以使用back按钮返回,就像在Tasks and Back Stack一章中谈论的那样).然而在一个移除Fragment的事务中,只有在你通过调用addBackStack()明确的指明这个 Fragment不要被保存,这个Fragment才会被放在被宿主Activity管理的返回栈中.

Otherwise, managing the fragment lifecycle is very similar to managing the activity lifecycle. So, the same practices for managing the activity lifecycle also apply to fragments. What you also need to understand, though, is how the life of the activity affects the life of the fragment.

另外,管理Fragment的生命周期和管理Activity的生命周期很相似.所以,当管理Activity生命周期的方法也适于管理Fragment的生命周期.当然你也需要明确Activity对Fragment生命周期的影响.

Caution: If you need a Context object within your Fragment, you can call `getActivity()`. However, be careful to call `getActivity()` only when the fragment is attached to an activity. When the fragment is not yet attached, or was detached during the end of its lifecycle, `getActivity()` will return null. 注意,如果在你的Fragment中需要一个context对象,你可以调用`getActivity()`.然而,只有这个Fragment附在这个Activity上的时候,才可以调用`getActivity()`.如果Fragment还没有附加在Activity上,或者在最后的生命周期和Activity分离了,那`getActivity()`方法将会返回null.

Coordinating with the activity lifecycle- 和Activity生命周期的协调

The lifecycle of the activity in which the fragment lives directly affects the lifecycle of the fragment, such that each lifecycle callback for the activity results in a similar callback for each fragment. For example, when the activity receives `onPause()`, each fragment in the activity receives `onPause()`.

拥有Fragment的Activity的生命周期会直接影响Fragment的生命周期,每个Activity生命周期方法会影响到每个Fragment.举个例子,当一个Activity执行`onPause()`方法的时候,它里面的每个Fragment也会执行`onPause()`.

Fragments have a few extra lifecycle callbacks, however, that handle unique interaction with the activity in order to perform actions such as build and destroy the fragment's UI. These additional callback methods are:

Fragment有一些额外的生命周期,用来处理和Activity的特殊交换,从而可以执行形如创建和销毁FragmentUI的事情.这些额外的回调方法有:

`onAttach()`

Called when the fragment has been associated with the activity (the Activity is passed in here).

当Fragment和Activity链接起来的时候调用(Activity在这里传送过来).

onCreateView()

Called to create the view hierarchy associated with the fragment.
创建Fragment的视图层.

onActivityCreated()

Called when the activity's onCreate() method has returned.
当Activity的onCreate返回的时候执行.

onDestroyView()

Called when the view hierarchy associated with the fragment is being removed.

当Fragment的试图层被移除的时候执行.

onDetach()

Called when the fragment is being disassociated from the activity.
当Fragment和Activity分离的时候执行.

The flow of a fragment's lifecycle, as it is affected by its host activity, is illustrated by figure 3. In this figure, you can see how each successive state of the activity determines which callback methods a fragment may receive. For example, when the activity has received its onCreate() callback, a fragment in the activity receives no more than the onActivityCreated() callback.

Fragment生命周期的流图,由于被宿主Activity影响,可以用图三表示.在这个表中,你可以知道每个Activity的每个状态是怎样决定一个Fragment收到的回调方法的.比如,当一个Activity收到他的onCreate()方法的时候,他里面的Fragment不会再收到onActivityCreated()方法的回调.

Once the activity reaches the resumed state, you can freely add and

remove fragments to the activity. Thus, only while the activity is in the resumed state can the lifecycle of a fragment change independently.

一旦Activity到达了resume状态,你可以随意添加和移除Activity中的Fragment.当然,只有这个Activity在resume状态的时候,Fragment的生命周期才可以独立的变化.

However, when the activity leaves the resumed state, the fragment again is pushed through its lifecycle by the activity.

然而,当activity离开了resume状态,Fragment会再一次被activity推到它的生命周期中.

==Example-例子==

To bring everything discussed in this document together, here's an example of an activity using two fragments to create a two-pane layout. The activity below includes one fragment to show a list of Shakespeare play titles and another to show a summary of the play when selected from the list. It also demonstrates how to provide different configurations of the fragments, based on the screen configuration.

为了把上面介绍的知识汇总,这里有个使用两个Fragment组成两个视图布局的例子.下面的activity包含两个Fragment,一个用来显示Shakespeare话剧的标题,另一个用来显示选中话剧的简介.也演示了怎么根据屏幕的不同为这两个Fragment提供不同的配置.

Note: The complete source code for this activity is available in FragmentLayout.java. 注意:完整代码在FragmentLayout.java中.

The main activity applies a layout in the usual way, during onCreate():

主activity用平常的方式生成布局,在onCreate()方法的时候:

@Override

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.fragment_layout);
}

```

The layout applied is fragment_layout.xml: fragment_layout.xml如下:

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        class="com.example.android.apis.app.FragmentLayout$TitlesFragment"
        android:id="@+id/titles" android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />

    <FrameLayout android:id="@+id/details" android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent"
        android:background="?"
        android:attr/detailsElementBackground" />

</LinearLayout>

```

Using this layout, the system instantiates the TitlesFragment (which lists the play titles) as soon as the activity loads the layout, while the FrameLayout (where the fragment for showing the play summary will go) consumes space on the right side of the screen, but remains empty at first. As you'll see below, it's not until the user selects an item from the list that a fragment is placed into the FrameLayout.

通过布局文件我们知道,系统在activity载入布局的时候实例化TitlesFragment(话剧的标题),FragmentLayout(显示话剧内容简介的Fragment)占据右边的屏幕但是现在没有内容.就像你下面看到的那样,直到用户选择了标题一个Fragment才会被放到FrameLayout.

However, not all screen configurations are wide enough to show both the list of plays and the summary, side by side. So, the layout above is used only for the landscape screen configuration, by saving it at res/layout-land/fragment_layout.xml.

然而,不是多有的屏幕配置都足够显示这两个Fragment视图.按照res/layout-land/fragment_layout.xml文件,上面的布局只适合横屏.

Thus, when the screen is in portrait orientation, the system applies the following layout, which is saved at res/layout/fragment_layout.xml:

那么当屏幕在竖屏的时候,系统会使用下面的布局,保存在res/layout/fragment_layout.xml中.

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <fragment
        class="com.example.android.apis.app.FragmentLayout$TitlesFragment"
            android:id="@+id/titles"
            android:layout_width="match_parent"
        android:layout_height="match_parent" />
</FrameLayout>
```

This layout includes only TitlesFragment. This means that, when the device is in portrait orientation, only the list of play titles is visible. So, when the user clicks a list item in this configuration, the application will start a new activity to show the summary, instead of loading a second fragment.

这个布局值包含TitlesFragment.这意味着当设备在竖屏的时候,只有话剧的标题是可见的.所以,当用户点击列表的一项的时候,应用将会开始一个新的activity来显示简介而不是载入第二个Fragment.

Next, you can see how this is accomplished in the fragment classes. First is TitlesFragment, which shows the list of Shakespeare play titles. This fragment extends ListFragment and relies on it to handle most of the list view work.

接下来,你将看到这在Fragment类中是怎么实现的.首先是TitleFragment,显示了莎士比亚话剧的标题.这个Fragment继承自ListFragment,可以通过它实现大多数显示列表信息操作.

As you inspect this code, notice that there are two possible behaviors when the user clicks a list item: depending on which of the two layouts is active, it can either create and display a new fragment to show the details in the same activity (adding the fragment to the FrameLayout), or start a new activity (where the fragment can be shown).

正如你看到的那样,注意在用户点击列表响的时候,有两个可能的行为:如果这两个视图存在,将在这个activity中创建并显示一个新的Fragment(把Fragment添加到FragmentLayout中);如果只有一个视图(竖屏),那会启动一个新的activity(Fragment在这个activity中显示).

```

public static class TitlesFragment extends ListFragment {
    boolean mDualPane;
    int mCurCheckPosition = 0;

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Populate list with our static array of titles.
        setListAdapter(new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_activated_1,
            Shakespeare.TITLES));

        // Check to see if we have a frame in which to embed the
        // details
        // fragment directly in the containing UI.
        View detailsFrame =
getActivity().findViewById(R.id.details);
        mDualPane = detailsFrame != null &&
detailsFrame.getVisibility() == View.VISIBLE;

        if (savedInstanceState != null) {
            // Restore last state for checked position.
            mCurCheckPosition =
savedInstanceState.getInt("curChoice", 0);
        }

        if (mDualPane) {
            // In dual-pane mode, the list view highlights the
selected item.

        listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
            // Make sure our UI is in the correct state.
            showDetails(mCurCheckPosition);
        }
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
    }
}

```

```

        outState.putInt( "curChoice" , mCurCheckPosition );
    }

    @Override
    public void onListItemClick( ListView l, View v, int position,
long id ) {
        showDetails( position );
    }

    /**
     * Helper function to show the details of a selected item,
either by
     * displaying a fragment in-place in the current UI, or starting
a
     * whole new activity in which it is displayed.
 */
void showDetails( int index ) {
    mCurCheckPosition = index;

    if ( mDualPane ) {
        // We can display everything in-place with fragments, so
update
        // the list to highlight the selected item and show the
data.
        getListView().setItemChecked( index, true );

        // Check what fragment is currently shown, replace if
needed.
        DetailsFragment details = ( DetailsFragment )

getFragmentManager().findFragmentById( R.id.details );
        if ( details == null || details.getShownIndex() != index )
{
            // Make new fragment to show this selection.
            details = DetailsFragment.newInstance( index );

            // Execute a transaction, replacing any existing
fragment
            // with this one inside the frame.
            FragmentTransaction ft =
getFragmentManager().beginTransaction();
            ft.replace( R.id.details, details );

ft.setTransition( FragmentTransaction.TRANSIT_FRAGMENT_FADE );
            ft.commit();
        }
    } else {
        // Otherwise we need to launch a new activity to display
        // the dialog fragment with selected text.
        Intent intent = new Intent();
        intent.setClass( getActivity(), DetailsActivity.class );
        intent.putExtra( "index" , index );
        startActivity( intent );
    }
}
}

```

The second fragment, DetailsFragment shows the play summary for the

item selected from the list from TitlesFragment:

第二个Fragment, DetailsFragment显示了在TitleFragment中选中的话剧简介.

```

public static class DetailsFragment extends Fragment {
    /**
     * Create a new instance of DetailsFragment, initialized to
     * show the text at 'index'.
     */
    public static DetailsFragment newInstance(int index) {
        DetailsFragment f = new DetailsFragment();
        // Supply index input as an argument.
        Bundle args = new Bundle();
        args.putInt("index", index);
        f.setArguments(args);

        return f;
    }

    public int getShownIndex() {
        return getArguments().getInt("index", 0);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Fragment fragment
    is
    it
    could
    return
    // the view hierarchy; it would just never be used.
    return null;
}

    ScrollView scroller = new ScrollView(getActivity());
    TextView text = new TextView(getActivity());
    int padding =
    (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
    4,
    getActivity().getResources().getDisplayMetrics());
    text.setPadding(padding, padding, padding, padding);
    scroller.addView(text);
    text.setText(Shakespeare.DIALOGUE[getShownIndex()]);
    return scroller;
}
}

```

Recall from the TitlesFragment class, that, if the user clicks a list item and the current layout does not include the R.id.details view (which is where the DetailsFragment belongs), then the application starts the DetailsActivity activity to display the content of the item.

来自TitleFragment的调用,如果用户点击列表项的时候当前布局不包含R.id.details视图(DetailsFragment 所在的视图),那应用将会启动DetailsActivity 来显示选中项的内容简介.

Here is the DetailsActivity, which simply embeds the DetailsFragment to display the selected play summary when the screen is in portrait orientation:

这里是DetailsActivity,在屏幕是竖屏的时候,简单的嵌入在了Fragment中来显示选中的话剧简介.

```
public static class DetailsActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getResources().getConfiguration().orientation
            == Configuration.ORIENTATION_LANDSCAPE) {
            // If the screen is now in landscape mode, we can show
            // the
            // dialog in-line with the list so we don't need this
            // activity.
            finish();
            return;
        }

        if (savedInstanceState == null) {
            // During initial setup, plug in the details fragment.
            DetailsFragment details = new DetailsFragment();
            details.setArguments(getIntent().getExtras());

            getFragmentManager().beginTransaction().add(android.R.id.content,
                details).commit();
        }
    }
}
```

Notice that this activity finishes itself if the configuration is landscape, so that the main activity can take over and display the DetailsFragment

alongside the TitlesFragment. This can happen if the user begins the DetailsActivity while in portrait orientation, but then rotates to landscape (which restarts the current activity).

注意activity会在横屏的时候结束自己,这样主activity可以接管并显示DetailsFragment旁边的TitlesFragment.如果用户在竖屏的时候启动DetailsActivity,然后把设备转到横屏(将会重启当前的activity).

For more samples using fragments (and complete source files for this example), see the API Demos sample app available in ApiDemos (available for download from the Samples SDK component).

更多使用Fragment的例子(包括这个例子的全部代码),请参考API Demo(可以在SDK例子那里下载).

来自 "[index.php?title=Fragments&oldid=13841](#)"



Loaders

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址：<http://developer.android.com/guide/topics/fundamentals/loaders.html>

翻译：[小手冰凉](#)

更新： **2012.07.07**

目录

[[隐藏](#)]

[1 加载器 - Loaders](#)

- [1.1 Loader API Summary](#)
- [1.2 在应用程序中使用Loaders](#)
 - [1.2.1 启动一个Loader](#)
 - [1.2.2 重新启动一个Loader](#)
 - [1.2.3 使用LoaderManager回调](#)
 - [1.2.3.1 onCreateLoader](#)
 - [1.2.3.2 onLoadFinished](#)
 - [1.2.3.3 onLoaderReset](#)
 - [1.2.4 例子](#)
 - [1.2.5 更多的例子](#)

加载器 - Loaders

从Android 3.0开始，Android引入loaders功能，loaders提供了在[activity](#)和[fragment](#)中异步载入数据以及监视数据源的变化的能

力。Loaders的特性如下：

- 在每个Activity和Fragment都可用；
- 实现异步加载数据
- 监控源数据的变化，当数据发生变化的时候获取新的数据；
- 他们最后的装载机光标自动重新连接到配置更改后创建时。因此，他们并不需要重新查询自己的数据。

Loader API Summary

在应用程序中使用Loaders可能会用到一些类和接口，在下表中总结：

Class/Interface	Description
LoaderManager	<p>和Activity和Fragment有关的抽象类，可以用来管理一个或者多个Loader实例。这可以帮助一个应用程序管理长时间运行的方法和Activity或者Fragment的生命周期结合起来。最常用的是一个CursorLoader，对于加载其他类型的数据来写入自己的装载机，应用程序都是免费的。每个activity或者fragment只能有一个LoaderManager.但是一个LoaderManager可以有多个Loaders。</p>
LoaderManager.LoaderCallbacks	<p>客户端和LoaderManager交互的回调接口。例如：你使用onCreateLoader()回调方法来创建一个新的Loader。</p>

Loader	执行异步加载数据的抽象类。是loader的父类。经常使用的是CursorLoader,但是你也可以实现自己的子类。当loaders处于活动状态时，它应该监控其数据的来源，当内容改变时候显示新的结果。
AsyncTaskLoader	抽象的loader提供一个AsyncTask来处理工作。
CursorLoader	AsyncTaskLoader的子类，查询ContentResolver返回一个游标。这个类实现了Loader协议，对于游标查询在一个标准的方式，建立在AsyncTaskLoader基础上，来执行游标查询在后台线程中，以至于不会中断应用程序的UI。 从ContentProvider中实现异步数据加载，而不是通过fragment或者Activity API来执行的管理查询，是最好的方式。

上述表中的类和接口是你在你的应用程序中实现一个加载器的基本组件。你没有必要全部使用到他们为你创建的加载器，但是但你总是需要一个参考 LoaderManager以初始化装载器和一个Loader类的实现，例如CursorLoader。以下部分显示您如何使用这些应用程序中的类和接 口。

在应用程序中使用**Loaders**

本节介绍如何在一个Android应用程序中使用**Loaders**。通常使用装载器的应用程序包括以下内容：

- 一个Activity或者Fragment
- 一个LoaderManager的实例
- 一个CursorLoader通过ContentProvider加载备份数据。另外，你可以实现你自己Loader或AsyncTaskLoader的子类加载一些其他来源的数据。
- LoaderManager.LoaderCallbacks的实现，这是你创建一个新的Loader和管理已经存在的loader的参考
- 显示loader数据的方式，例如，SimpleCursorAdapter。
- 一个数据源，例如：ContentProvider，当使用CursorLoader时。

启动一个**Loader**

LoaderManager通过Activity或者Fragment来管理一个或者多个Loader实例。在一个Activity或者Fragment中仅仅只有一个LoaderManager。

你通常的可以通过Activity的onCreate()方法来初始化一个Loader，或者通过Fragment的onActivityCreated()方法。你可以按照下面的做：

```
// Prepare the loader. Either re-connect with an existing one,
// or start a new one.
getLoaderManager().initLoader(0, null, this);
```

initLoader()方法需要以下几个参数：

- 一个唯一的Id标识，在这个例子中Id是0。
- 构造函数的可选参数（在这里为空）
- LoaderManager.LoaderCallbacks的实现，LoaderManager调用来报告Loader事件。在这个例子中，本类实现了LoaderManager.LoaderCallbacks接口，它传递自身的引用，this。

initLoader()调用，确保Loader被初始化，以及存在。它可能有两种结果：

- 如果通过Id标识的Loader已经存在，上次被创建的Loader将会被重用。
- 如果不存在的，由ID标识的Loader，initLoader () 方法触发

LoaderManager.LoaderCallbacks的onCreateLoader () 方法。这是你实现的代码来实例化并返回一个新的Loader。更多的讨论，请参阅节onCreateLoader。

在这两种情况下，给出的LoaderManager.LoaderCallbacks的实现类和loader有关，当loader的状态发生变化时 将会被调用。如果主叫方在此调用点是在它的启动状态，请求loader已经存在并且产生了数据，这是系统会立刻调用onLoadFinished () 方法。所以你应该准备好这将会发生。

注意虽然initLoader()方法返回了创建的Loader，但也不必捕获参考。Loader Manager会自动管理Loader的使用寿命。必要时Loader Manager会开始或停止加载，并维持Loader及其相关内容的状态。这就意味着你极少会用Loader进行直接交互（如果想参考使用Loader方法来微调的示例，请查阅 LoaderThrottle）。有突发事件时通常使用LoaderManager.LoaderCallbacks方法在加载过程中进行干预。您可以从Using the LoaderManager Callbacks中获取更多信息。

重新启动一个Loader

当你使用initLoader(),它将使用指定Id的已经存在的Loader。如果没有，将会创建一个。但有时你要放弃你的旧数据，并重新开始。

想要丢弃您的旧数据，您使用restartLoader () 方法。例如：当用户的查询状态改变时，实现 SearchView.OnQueryTextListener的类，会重启Loader。Loader需要重新启动，以便它可以使用修改后的搜索过滤器，做一个新的查询：

```
public boolean onQueryTextChanged(String newText) {
    // Called when the action bar search text has changed. Update
    // the search filter, and restart the loader to do a new query
    // with this filter.
    mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
    getLoaderManager().restartLoader(0, null, this);
    return true;
}
```

使用LoaderManager回调

LoaderManager.LoaderCallbacks是一个回调接口，可以让客户端和LoaderManager交互。

Loaders，尤其是CursorLoader，当将要结束的时候，它们被要求需要保存他们的数据。这允许应用程序来保存数据贯穿于 Activity或者fragment的onStop () 和OnStart () 方法,这样当用户返回到应用程序，他们不必等待重新载入数据。当你想知道 合适创建一个新的loader，您可以使用LoaderManager.LoaderCallbacks方法，来告诉应用程序什么时候停止使用 loader的数据。

LoaderManager.LoaderCallbacks包括这些方法：

- `onCreateLoader()` — Instantiate and return a new Loader for the given ID.
- `onLoadFinished()` — Called when a previously created loader has finished its load.
- `onLoaderReset()` — Called when a previously created loader is being reset, thus making its data unavailable.

这些方法在下面的章节将详细介绍。

onCreateLoader

当您尝试访问一个loader（例如，通过loader中的`initLoader ()`方法），它会检查是否存在由指定的ID的loader。如果不存在，它会触发的LoaderManager.LoaderCallbacks的`onCreateLoader ()`方法。

在这个例子中，`onCreateLoader ()` 回调方法创建一个CursorLoader。你必须使用CursorLoader的构造方法构造它，需要ContentProvider执行查询时所需要的很多信息。具体来说，它需要：

- `Uri`-内容检索的URI
- `projection` — 要返回的列元素。如果为空的话，返回所有的列，但是这效率比较低。
- `selection` — 声明返回行的过滤器，格式化为一个SQL WHERE子句（不含WHERE本身）。传递为空，将返回给定的URI的所有行。
- `selectionArgs` —你可以包含? 在selection中，将会被 selectionArgs的值替代，该值将被绑定为字符串。
- `SortOrder` — 如何对列进行排序，格式化为SQL令（不含ORDER本身）BY子句。如果为空将使用默认的排序顺序，这可能是无序的。

例如：

```
// If non-null, this is the current filter the user has provided.
String mCurFilter;
...
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // This is called when a new Loader needs to be created. This
    // sample only has one Loader, so we don't care about the ID.
    // First, pick the base URI to use depending on whether we are
    // currently filtering.
    Uri baseUri;
    if (mCurFilter != null) {
        baseUri = Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
            Uri.encode(mCurFilter));
    } else {
        baseUri = Contacts.CONTENT_URI;
    }
    // Now create and return a CursorLoader that will take care of
    // creating a Cursor for the data being displayed.
    String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL) AND (" +
        + Contacts.HAS_PHONE_NUMBER + "=1) AND (" +
        + Contacts.DISPLAY_NAME + " != '' ))";
    return new CursorLoader(getActivity(), baseUri,
        CONTACTS_SUMMARY_PROJECTION, select, null,
        Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
}
```

onLoadFinished

当先前创建的loader完成了它的加载时，这个方法被调用。这个方法保证在被调用之前，释放loader提供的最近的数据。在这一点上，你应该删除使用的所有的旧数据（因为它会很快被释放），但是你不应该释放自己的数据，loader将会处理它。

一旦知道应用程序不再使用的数据，loader将会释放它。例如：如果数据是从一个CursorLoader的游标，你不应该自己调用 `close()` 方法就可以了。如果光标被放置在一个CursorAdapter的，你应该使用`swapCursor()` 方法，以至于旧的游标没有被关闭。例如：

```
// This is the Adapter being used to display the list's data.
SimpleCursorAdapter mAdapter;
...
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Swap the new cursor in. (The framework will take care of
    // closing the
    // old cursor once we return.)
    mAdapter.swapCursor(data);
}
```

onLoaderReset

先前创建的loader被重置时这个方法将会被调用，这样使得它的数据不可用。该反馈会提醒你何时释放数据，以便于您删除对应的引用。

此实现调用 swapCursor () 方法：

```
// This is the Adapter being used to display the list's data.
SimpleCursorAdapter mAdapter;
...
public void onLoaderReset(Loader<Cursor> loader) {
    // This is called when the last Cursor provided to
onLoadFinished()
    // above is about to be closed. We need to make sure we are no
    // longer using it.
    mAdapter.swapCursor(null);
}
```

例子

作为一个例子，这里完整的实现了一个Fragment，它显示了一个ListView，该ListView包含的查询结果与联系人内容提供者一致。它使用一个CursorLoader在提供者上管理查询。

对于应用程序访问用户的联系人，就像本例所示，它的清单中必须包含READ_CONTACTS许可。

```
public static class CursorLoaderListFragment extends ListFragment
    implements OnQueryTextListener,
LoaderManager.LoaderCallbacks<Cursor> {
    // This is the Adapter being used to display the list's data.
    SimpleCursorAdapter mAdapter;
    // If non-null, this is the current filter the user has
provided.
    String mCurFilter;
    @Override public void onActivityCreated(Bundle
savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        // Give some text to display if there is no data. In a real
        // application this would come from a resource.
        setEmptyText("No phone numbers");
        // We have a menu item to show in action bar.
        setHasOptionsMenu(true);
        // Create an empty adapter we will use to display the loaded
data.
        mAdapter = new SimpleCursorAdapter(getActivity(),
            android.R.layout.simple_list_item_2, null,
            new String[] { Contacts.DISPLAY_NAME,
Contacts.CONTACT_STATUS },
```

```

        new int[] { android.R.id.text1, android.R.id.text2
}, 0);
    setListAdapter(mAdapter);
    // Prepare the loader. Either re-connect with an existing
one,
    // or start a new one.
    getLoaderManager().initLoader(0, null, this);
}
@Override public void onCreateOptionsMenu(Menu menu,
MenuInflater inflater) {
    // Place an action bar item for searching.
    MenuItem item = menu.add("Search");
    item.setIcon(android.R.drawable.ic_menu_search);
    item.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
    SearchView sv = new SearchView(getActivity());
    sv.setOnQueryTextListener(this);
    item.setActionView(sv);
}
public boolean onQueryTextChange(String newText) {
    // Called when the action bar search text has changed.
Update
    // the search filter, and restart the loader to do a new
query
    // with this filter.
    mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;
    getLoaderManager().restartLoader(0, null, this);
    return true;
}
@Override public boolean onQueryTextSubmit(String query) {
    // Don't care about this.
    return true;
}
@Override public void onListItemClick(ListView l, View v, int
position, long id) {
    // Insert desired behavior here.
    Log.i("FragmentComplexList", "Item clicked: " + id);
}
// These are the Contacts rows that we will retrieve.
static final String[] CONTACTS_SUMMARY_PROJECTION = new String[]
{
    Contacts._ID,
    Contacts.DISPLAY_NAME,
    Contacts.CONTEXT_STATUS,
    Contacts.CONTEXT_PRESENCE,
    Contacts.PHOTO_ID,
    Contacts.LOOKUP_KEY,
};
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    // This is called when a new Loader needs to be created.
This
    // sample only has one Loader, so we don't care about the
ID.
    // First, pick the base URI to use depending on whether we
are
    // currently filtering.
    Uri baseUri;
    if (mCurFilter != null) {
        baseUri =
Uri.withAppendedPath(Contacts.CONTENT_FILTER_URI,
                    Uri.encode(mCurFilter));
    } else {
}

```

```

        baseUri = Contacts.CONTENT_URI;
    }
    // Now create and return a CursorLoader that will take care
of
    // creating a Cursor for the data being displayed.
    String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL"
AND (
        + Contacts.HAS_PHONE_NUMBER + "=1) AND (
        + Contacts.DISPLAY_NAME + " != '' ))";
    return new CursorLoader(getActivity(), baseUri,
        CONTACTS_SUMMARY_PROJECTION, select, null,
        Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
}
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    // Swap the new cursor in. (The framework will take care of
closing the
    // old cursor once we return.)
    mAdapter.swapCursor(data);
}
public void onLoaderReset(Loader<Cursor> loader) {
    // This is called when the last Cursor provided to
onLoadFinished()
    // above is about to be closed. We need to make sure we are
no
    // longer using it.
    mAdapter.swapCursor(null);
}
}

```

更多的例子

有几个不同的例子在ApiDemos中，说明如何使用装载机：

- [LoaderCursor](#)
- [LoaderThrottle](#)

来自“[index.php?title=Loaders&oldid=13843](#)”



Tasks and Back Stack

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址：<http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html>

翻译：無語 @ 2012.06.07?

目录

[[隐藏](#)]

1 任务栈和返回堆栈

- [1.1 保存Activity状态](#)
- [1.2 管理多个Task](#)
 - [1.2.1 定义启动模式](#)
 - [1.2.1.1 配置 manifest 清单文件](#)
 - [1.2.1.2 使用 Intent 标识](#)
 - [1.2.2 affinities处理](#)
 - [1.2.3 清除Back Stack](#)
 - [1.2.4 启动Task](#)

任务栈和返回堆栈

一个应用程序通常包含多个Activity. 每个Activity都必须设计成一种特定的操

快速查看

- 所有的Activity 都属于Task
- Task包含了用户操作Activity顺序排列的集合
- Task可以转入后台,并会保存每个Activity 的状态,以便用户运行其它Task时候不会丢失当前的工作.

在本文中 (参见目录)

相关文章

[Multitasking the Android Way](#)

参阅

[Android 设计 : 导航](#)

[应用程序的生命周期视频](#)

[`<activity>` manifest 清单元素](#)

作, 用户可以通过该操作去实现某项功能, 并且操作其他的Activity. 例如.一个电子邮件的应用程序可能有一个Activity, 用于展现出新的电子邮件列表, 当用户选择了一个电子邮件, 就打开一个新的Activity以查看该电子邮件.

一个Activity可以启动设备上的另外的一个应用程序. 例如, 如果您的应用程序想要发送一个电子邮件, 您可以定义一个意图来执行一个“发送”行动, 其中包括一些数据, 如电子邮件地址和消息. 一个Activity在另一个应用程序, 声明本身来处理这种意图然后打开. 在这种情况下. 目的是为了发送电子邮件, 所以电子邮件应用该创建Activity启动器(如果多个活动支持相同的意图, 那么系统让用户选择使用哪一个). 当发送邮件时, 你恢复了的 Activity 和电子邮件的Activity好像是您的应用程序的一部分一样. 即是这些Activity可能来自不同的应用, android系统也会通过相同的任务栈来保存这种无缝的用户体验.

当用户进行某项操作时, 任务栈就收集相互交互的Activity. Activity会被安排在堆栈中(返回堆栈), 堆栈中的Activity会按顺序来重新打开.

一个设备的主屏幕是大多数任务栈的起点. 当用户触摸图标(或者在屏幕上的快捷方式)时, 该应用程序就会到达任务栈的最前面. 如果该应用在任务栈中不存在(应用在近段时间内没有使用过)就会在任务栈中创建一个新的任务, 并将该应用作为“主”Activity放置在根任务栈中.

当前Activity开始时, 新的Activity推入堆栈的顶部和焦点. 以前的Activity仍然在堆栈中, 但已停止. 当 Activity 停止时, 系统会保留其用户界面的当前状态. 当用户按下返回按钮时, 当前Activity就会从堆栈的顶部(当前的Activity就会被销毁)和之前的一个Activity就会恢复(恢复到之前的UI界面). Activity在栈中的顺序永远不会改变, 只会压入和弹出——被当前Activity启动时压入栈顶, 用户用返回键离开时弹出. 这样, back stack 以“后进先出”的方式运行。图1 以时间线表的方式展示了多个Activity切换时对应当前时间点的Back Task状态.



图1.图1所表示的Task中的每个新Activity都会相应在Back Task中增加一项。当用户按下返回键时, 当前的Activity就会被销毁, 而之前一

个Activity就会被恢复。如果用户不停地按下返回键的时候，那么栈中每个Activity都会依次弹出，并且显示前一个Activity，直至用户回到Home屏幕（或者任一启动该Task的Activity，当所有Activity都从栈中弹出后，Task就不再存在。）

Task是一个整体的单位，当用户启动一个新的Task或者通过Home按钮回到Home屏幕的时候，这个TASK就转为“后台”。当Task处于后台时，里面所有的Activity都处于停止状态，但是这个Task的BACK Task仍然完整保留——如图2所示，在其它Task获得焦点期间，这个Task只是失去焦点而已。TASK可以回到前台，以便用户继续之前的操作。例如：当前Task(Task A)的栈中共有3个Activity——下面有两个Activity。这时，用户按下Home键，然后从Application launcher中启动一个新的应用。当Home屏幕出现时，Task A进入后台。当新的应用启动时，系统会为它开启一个Task (Task B)，其中放入新应用中的activity。用完这个应用后，用户再次返回Home屏幕，并选中那个启动Task A的应用。现在，Task A进入前台——栈中的三个Activity仍然完好，位于最顶部的Activity恢复运行。这时候，用户仍然可以切回Task B，通过回到Home屏幕并选择相应图标即可（或者触摸并按住Home键调出最近Task列表并选中）。以下是Android多task的实例。

注意：

Android系统可以在后台同时保存多个Task。但是，假如用户同时运行着多个后台Task，系统可能会销毁后台Activity用于释放内存，这样的情况就会导致Activity状态的丢失。详细情况请参阅[#保存Activity状态](#)。



图2.. 两个Task：Task B 在前台与用户交互，而 Task A 在后台等待唤醒。



图 3. activity A 被实例化多次。

因为Back Stack 中的Activity顺序永远不会改变，如果应用允许某个Activity可以让用户启动多次，则新的实例会压入栈顶（而不是打开之前位于栈顶的Activity）。于是，一个Activity可能会初始化多次（甚至会位于不同

的Task 中) , 如图3所示.如果用户用返回键键返回时, Activity 的每个实例都会按照原来打开的顺序显示出来 (用户界面也都按原来状态显示.当然, 如果你不想让Activity 能被多次实例化, 你可以改变它.具体方法在后面的章节[#管理多个Task](#)中详细说明.

1. 总结Activity和Task的默认行为:

- 当 Activity A 启动 Activity B 时,Activity A 被停止,但系统仍会保存Activity A 状态 (比如滚动条位置和 form 中填入的文字) 如果用户在 Activity B 中按下返回键时,Activity A 恢复运行, 状态也将恢复.
- 当用户按下Home键离开Task 时, 当前 Activity 停止Task 转入后台,系统会保存Task中每个Activity 的状态。如果用户以后通过选中启动该 Task 的图标来恢复Task,Task 就会回到前台, 栈顶的Activity 会恢复运行.
- 如果用户按下返回键,当前Activity 从栈中弹出,并被销毁.栈中前一个Activity恢复运行.当Activity 被销毁时, 系统不会保留Activity 的状态.
- Activity甚至可以在不同的Task中被实例化多次.

导航设计 关于 Android 中应用程序间导航的详情, 请参阅 [Android 导航设计指南](#).

保存Activity状态

如上所述,系统默认会在Activity 停止时保存其状态.这样,当用户返回时,用户的界面能与离开时显示得一样.不过, 你可以,也应该使用用回调方法主动地保存Activity的状态,以便应对Activity被销毁并重建的情况.

当系统停止一个Activity 运行后 (比如启动了一个新Activity 或者Task 转入后台),系统在需要回收内存的时候有可能会完全销毁该Activity.这时,该Activity的状态信息将会丢失.就算这种情况发生,该 Activity 仍然会存在于Back Stack中.但是当它回到栈顶时, 系统将必须重建它 (而不是恢

复).为了避免用户工作内容的丢失，你应通过实现Activity 的 [onSaveInstanceState\(\)](#) 方法来主动保存这些内容.

关于如何保存Activity 状态的详情，请参阅 [Activities](#).

管理多个Task

如上所述,把所有已经启动的Activity 相继放入同一个Task 中以及一个"后入先出"栈, Android 管理Task和Back Stack 的这种方式适用于大多数应用,你也不用去管理你的Activity 如何与Task关联及如何弹出Back Stack .不过,有时候你或许决定要改变这种普通的运行方式.也许你想让某个Activity启动一个新的Task (而不是被放入当前Task中),或者,你想让 activity 启动时只是调出已有的某个实例 (而不是在Back Stack 顶创建一个新的实例) 或者,你想在用户离开Task 时只保留根Activity , 而Back Stack 中的其它Activity 都要清空,

你能做的事情还有很多, 利用 "[<activity>](#) manifest 元素的属性和传入 [startActivity\(\)](#) 的 intent 中的标识即可。

这里, 你可以使用的 [<activity>](#) 属性主要有:

- [taskAffinity](#)
- [launchMode](#)
- [allowTaskReparenting](#)
- [clearTaskOnLaunch](#)
- [alwaysRetainTaskState](#)
- [finishOnTaskLaunch](#)

可用的 intent 标识主要有:

- [FLAG_ACTIVITY_NEW_TASK](#)
- [FLAG_ACTIVITY_CLEAR_TOP](#)
- [FLAG_ACTIVITY_SINGLE_TOP](#)

在下面的一个章节中, 你可以看到怎么样利用这些 manifest 属性 和intent标识来定义Activity 与Task的关联性, 以及 Back Stack 的工作方

式。

警告： 大多数应用不应该改变 Activity 和 Task 默认的工作方式。如果你确定有必要修改默认方式，请保持谨慎，并确保 Activity 在启动和从其它 Activity 用返回键返回时的可用性。请确保对可能与用户预期的导航方式相冲突的地方进行测试。

定义启动模式

启动模式定义了一个新的Activity 实例与当前Task 的关联方式。定义启动模式的方法有两种：

- [使用 manifest 文件](#)

当你在 manifest文件中声明一个Activity 时，可以指定它启动时与Task 的关联方式。

- [使用 Intent 标志](#)

调用 [startActivity\(\)](#) 时，可以在 Intent 中包含一个标识，用于指明新Activity 是如何（是否）与当前Task 相关联。

同样的，如果 Activity A 启动了Activity B，则 Activity B 可以在 manifest 中定义它如何与当之前的Task 关联（如果存在的话），并且，Activity A 也可以要求 Activity B 与当前 task 的关联关系。如果两个Activity都定义了，则 Activity A 的请求（intent 中定义）会比Activity B 的定义（在 manifest 中）优先。

注意： manifest 文件中的某些启动模式在 intent 标识中并不可用，反之亦然，intent 中的某些模式也无法在 manifest 中定义。

配置 manifest 清单文件

在manifest文件中声明Activity 时，你可以使用 [`<activity>`](#) 元素的 [`launchMode`](#) 属性来指定Activity与Task的关系。

[`launchMode`](#)

Activity Task

[`launchMode`](#)

属性指明了启动的方式。属性可设置

四种启动模式：

`"standard"` (默认模式)：“每次访问实例化新的Activity” 默认,系统在启动Activity 的Task 中创建一个新的Activity 实例,并且把 intent 传递路径指向它.该Activity 可以被实例化多次,各个实例可以属于不同的Task ,一个Task 中也可以存在多个实例.

`"singleTop"` “每次访问,看栈顶元素目标对象,是则返回,不再实例化,否则,还是实例化新的Activity.” 如果Activity的一个实例已经存在于当前Task的栈顶, 该系统就会使用[onNewIntent\(\)](#)方法通过intent 传递给已有实例, 而不是创建一个新的Activity 实例.Activity 可以被实例化多次,各个实例可以属于不同的Task,一个Task中可以存在多个实例(但只有Back Stack的Activity 实例不是该Activity 的) . 例如, 假设Task 的 Back Stack 中包含了根Activity A 和 Activities B、C、D (顺序是 A-B-C-D; D在栈顶. 这时候传过来的是启动D的intent,如果D的启动模式是默认的 `"standard"` , 则会启动一个新的实例, 栈的内容就会变为 A-B-C-D-D.但是!!!但是, 如果 D 的启动模式是 `"singleTop"` , 则已有的D的实例会通过[onNewIntent\(\)](#):接收这个 intent , 因为该实例位于栈顶——栈中内容仍然维持 A-B-C-D 不变.当然, 如果 intent 是要启动 B 的,则 B 的一个新实例还是会加入栈中,即使 B 的启动模式是 `"singleTop"` 也是如此.

注意：一个Activity 的新实例创建完毕后，用户可以按返回键返回前一个activity.但是当Activity 已有实例正在处理刚到达的intent 时，用户无法用返回键回到[onNewIntent\(\)](#)中 intent 到来之前的Activity 状态.

`"singleTask"` “保证activity实例化一次,单任务,由此所开启的活动和本活动位于同一task中” 系统将创建一个新的Task , 并把Activity 实例作为根放入其中.但是, 如果Activity 已经在其它Task 中存在实例, 则系统会通过调用其实例的[onNewIntent\(\)](#) 方法把 intent传给已有实例,而不是再创建一个新实例. 此 activity 同一时刻只能存在一个实例.

注意： 虽然Activity启动了一个新的Task , 但用户仍然可以用返回键返回

前一个activity.

`"singleInstance"` "保证Activity实例化一次,单实例,由此所开启的Activity在新的task中,和本活动id不一致." 除了系统不会把其它Activity 放入当前实例所在的 Task 之外, 其它均与 `"singleTask"` 相同,Activity 总是它所在Task 的唯一成员; 它所启动的任何Activity 都会放入其它Task 中.

举一个事例:Android 的浏览器应用就把 web 浏览器Activity 声明为总是在它自己独立的Task 中打开——把 `activity` 设为`singleTask` 模式. 这意味着, 如果你的应用提交 `intent` 来打开 Android 的浏览器, 则其 `activity` 不会被放入你的应用所在的Task 中. 取而代之的是, 或是为浏览器启动一个新的Task; 或是浏览器已经在后台运行, 只要把Task 重新调入前台来处理新 `intent` 就可以.

无论Activity是在一个新的Task 中启动, 还是位于其它已经存在的Task 中, 用户总是可以返回键返回到前一个Activity 中. 但是, 如果你启动了一个启动模式设为`singleTask` 的 `activity`, 且有一个后台 task 中已存在实例的话, 则这个后台 task 就会整个转到前台. 这时, 当前的Back Stack就包含了这个转入前台的Task 中所有的Activity, 位置是在栈顶. 图4所描述的就是这一种场景



图 4. 启动模式为“singleTask”的Activity 如何加入Back Stack 的示意. 如果Activity 已经是在后台 Task 中并带有自己的Back Stack, 则整个后台Back Stack 都会转入前台, 并放入当前Task 的栈顶.

在manifest 文件中使用启动模式的详情, 请参阅[`<activity>`](#)元素文档, 其中详细描述了 `launchMode` 属性及其可用值.

注意: 你使用 `launchMode` 属性为Activity设置的模式可以被启动Activity的intent标识所覆盖, 这将在下一章节具体说明。

使用 Intent 标识

在启动Activity 时, 你可以在传给 `startActivity()` 的 intent 中包含相应标识, 用于修改Activity 与Task 的默认关系. 这个标识可以修改的默认模式包

括：

FLAG_ACTIVITY_NEW_TASK

在新的Task 中启动Activity.如果要启动的Activity 已经运行于某个Task 中,则那个Task 将调入前台中,最后保存的状态也将会恢复,Activity 将在[onNewIntent\(\)](#)中接收到这个新 intent.

这个模式与前一章节所描述述的 "singleTask" [launchMode](#)模式相同.

FLAG_ACTIVITY_SINGLE_TOP

如果要启动的Activity 就是当前Activity (位于Back Stack 顶) ,则已存在的实例将接收到一个[onNewIntent\(\)](#)调用,而不是创建一个Activity 的新实例.

这个模式与前一章节所述的 "singleTop" [launchMode](#)模式相同.

FLAG_ACTIVITY_CLEAR_TOP

如果要启动的Activity 已经在当前Task中运行,则不再启动一个新的实例,且所有在其上面的Activity 将被销毁,然后通过[onNewIntent\(\)](#)传入 intent 并恢复Activity (不在栈顶) 的运行.

此种模式在[launchMode](#)中没有对应的属性值.

`FLAG_ACTIVITY_CLEAR_TOP` 经常与 `FLAG_ACTIVITY_NEW_TASK` 结合起来一起使用.这些标识定位在其它Task中已存在的Activity,再把它放入可以响应 intent的位置上.

注意：如果Activity 的启动模式配置为 "standard" , 它就会先被移除出栈, 再创建一个新的实例来处理这个 intent,因为启动模式为 "standard" 时, 总是会创建一个新的实例。

affinities处理

affinity指明Activity对于哪些Task亲和力更高.默认情况下, 同一个应用中的所有 activity 都拥有同一个affinity 值. 因此, 在同一应用程序中的所有Activity都喜欢在相同的Task.不过, 你可以修改Activity 默认的 affinity 值.不同应用中的Activity 可以共享同一个affinity 值,同一个应用中的Activity也可以赋予不同的Task affinity值.

你可以使用[<activity>](#)元素的[taskAffinity](#) 属性修改Activity的affinity

[taskAffinity](#) 属性是一个字符串值,必须与[<manifest>](#) 元素定义的包名称保证唯一性,因为系统把这个包名称用于标识应用的默认Task affinity值.

下面的两种情况下affinity将会发挥作用:

- 当启动Activity的intent包含了[FLAG_ACTIVITY_NEW_TASK](#)标志.

默认情况下，一个新的Activity将被放入调用[startActivity\(\)](#)

的Activity 所在Task 中,且压入调用者所在的Back Stack 顶栈.不过,如果传给[startActivity\(\)](#) 的 intent 包含了[FLAG_ACTIVITY_NEW_TASK](#) 标识，则系统会查找另一个Task并将新Activity 放入其中.一般情况下会新开一个任务,但并非一定如此.如果一个已有Task的affinity值与新 Activity的相同,则Activity 会放入该Task.如果没有,则会新建一个新Task.

如果这个标志导致Activity启动了一个新的Task并且用户按下Home键离开时,必须采取某种方式让用户能回到此Task.某些应用 (比如通知管理器) 总是让Activity放入其它Task 中启动,而不是放入自己的Task 中,因此,它们总是把 [FLAG_ACTIVITY_NEW_TASK](#) 标识置入传给

[startActivity\(\)](#) 的 intent 中.如果你的Activity 可以被外部应用带此标识来启动,请注意用户会用其它方式返回启动Task , 比如通过应用图标 (Task 的根 Activity 带有一个[CATEGORY_LAUNCHER](#) intent过滤器; 参阅下节[#启动task](#)) .

- 当一个Activity的[allowTaskReparenting](#)属性设置为 "true".

在这种情况下,当某个Task 进入前台时,Activity 的affinity 值又与其相同,则它可以从启动时的Task 移入这个Task 中.

例如:假设某旅游应用中有一个Activity 根据所选的城市来报告天气情况.它的affinity 与同一应用中的其它Activity 一样 (整个应用默认的affinity),且它允许重新指定此属性的归属,当你的另一个Activity 启动此天气预告Activity 时,它会在同一个Task 中启动.然而,当旅游应用的Task 进入前台时,则天气报告Activity将会重新放入其Task中并显示出来.

提示：如果一个 `.apk` 文件中包含了多个“application”，从用户的角度来看，你可能想使用 `taskAffinity` 属性来分配每个“application”中 activity 的 affinity 值。

清除Back Stack

如果用户长时间离开某个 Task，系统将会仅保留一个根 Activity，而把其它 Activity 都清除掉。当用户返回 Task 时，只有根 Activity 会被恢复。系统的这种行为，是因为经过了很长时间后，用户是要放弃之前进行的操作，返回 Task 是为了开始新的操作。

可以使用 Activity 的某些属性来改变这种行为：

[alwaysRetainTaskState](#):

如果 Task 中根 Activity 的此属性设为 `"true"`，则默认的清理方式不会进行。即使过了很长一段时间，Task 中所有的 Activity 也还会保留在栈中。

[clearTaskOnLaunch](#):

如果 Task 中根 Activity 的此属性设为 `"true"`，则只要用户离开并再次返回该 Task，栈就会被清理至根 Activity。也就是说，正好与 [alwaysRetainTaskState](#) 相反。用户每次返回 Task 时看到的都是初始状态，即使只是离开一会儿。

[finishOnTaskLaunch](#)

此属性类似于 [clearTaskOnLaunch](#)，只是它只对一个 Activity 有效，不是整个 Task。这能让任何一个 Activity 消失，包括 根 Activity。如果 Activity 的此属性设为 `"true"`，则只会保留 Task 中当前 session 所涉及的内容。如果用户离开后再返回 Task，它就不存在。

启动Task

你可以通过发送一个指定动作和类别为
`"android.intent.action.MAIN"`、`category` 为

"`android.intent.category.LAUNCHER`" 发送 intent 来指定某个 Activity 为 Task 的入口, 例如:

```
<activity ... >
    <intent-filter ... >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"
/>
    </intent-filter>
    ...
</activity>
```

这种 intent 过滤器将会让此 Activity 的图标和标签作为应用启动图标来显示, 用户可以启动此 Activity, 并且在之后任何时候返回其启动时的 Task.

第二个能力非常重要. 用户必须能离开一个 Task, 之后能再回来使用这个启动 Task 的 Activity. 由于这个原因, 标明 Activity 每次都会启动 Task 的这两种启动模式 "`singleTask`" 和 "`singleInstance`" 应仅用于 `ACTION_MAIN` 和 `CATEGORY_LAUNCHER` 过滤器的 Activity 才能使用. 例如, 如果缺少过滤器会发生怎样的问题: 某个 intent 启动了一个 "`singleTask`" activity, 并新建了一个 Task, 用户在此 Task 中工作了一段时间. 然后他按了 Home 键, Task 就转入后台, 变为不可见状态, 这时用户就无法再回到 Task 了, 因为它在 application launcher 中配置对应的属性.

对于那些你不希望用户能够返回的 Activity, 将 `<activity>` 元素的 `finishOnTaskLaunch` 设置为 "`true`" 即可. 详细请 (参阅 #清除 Back Stack)

来自 "[index.php?title=Tasks_and_Back_Short&oldid=13090](#)"



Services

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：cdkd123

原文链接：<http://docs.eoeandroid.com/guide/components/services.html>

快速预览

- 用户即使切换到另外的应用，服务(service)也会在后台运行。
- 服务允许和多个组件绑定，这样就可以允许多进程和它交互。
- 默认服务被应用的主线程劫持。

目录

[[隐藏](#)]

[1 Services](#)

- [1.1 基础](#)
 - [1.1.1 在manifest中声明一个服务](#)
- [1.2 创建一个启动的服务](#)
 - [1.2.1 继承IntentService类](#)
 - [1.2.2 继承Service类](#)
 - [1.2.3 启动服务](#)
 - [1.2.4 停止服务](#)
- [1.3 创建一个已绑定的服务](#)
- [1.4 发送用户通知](#)
- [1.5 前台运行一个服务](#)
- [1.6 管理服务的生命周期](#)

Services

一个服务即一个应用组件，是可以长期在后台运行，而且不提供用户任何接口。即使启动了其它应用，之前启动的服务仍会继续运行。组件可以绑定服务并与之交互，甚至允许多进程交互(ipc)。例如一个服务可以后台联网，后台播放音乐，后台处理文件输入/输出(I/O),或者后台和内容提供者(content provider)交互。

本质上一个服务有两种类型：

直接启动的服务

应用组件(例如一个Activity)调用startService()方法就可以启动一个服务。一旦启动,服务就在后台无限运行，即使启动它的

组件被销毁。通常，服务启动一个单操作并且不返回结果给调用者。例如，它会通过网络下载、上传文件，当一个操作结束，该服务应该自动结束。

绑定的服务

应用组件调用bindService()绑定服务。绑定的服务提供一个客户端服务器(client)接口允许组件与之交互,发送请求,获得结果，甚至多进程交互执行这些操作。服务和另一个与之绑定的组件运行时间一样长。多个组件只能和一个服务绑定一次，但所有组件取消绑定之后，服务就会销毁。尽管本文档分开讨论两类服务(Started 和Bound)，但服务可以同时用两种方式工作-可以启动之后无限期运行而且允许绑定。这取决于你有木有都实现这两类服务的回调接口:[int, int](#)
[onStartCommand\(\)](#)启动服务[onBind\(\)](#)绑定服务。

不管应用请求的是哪一种服务，任何组件(非本应用的也可以)都可以使用该服务。同样，任何组件(包括其他应用)都可以使用activity,用 Intent启动。但是，你可以在配置文件manifest中把服务声明为私有的，阻止其他应用访问。详见在配置文件manifest中声明一个服务.

注意:服务运行于主线程中-即服务不会自己创建另一个线程(除非你指定)。这意味着如果服务执行任何cpu耗时操作或异步操作(像MP3播放或联网),你应该创建一个新线程执行这类操作,这样,就可以减少死机的风险,主线程就可以专门负责和你的activity交互。

基础

使用线程还是服务? 即使用户没和应用交互,服务仍然会可以在后台运行。因此,你应该在需要时才使用它。如果你不想在主线程执行一个任务,而是仅当用户和应用交互时执行该任务,你应该创建新线程而不是服务。例如,你只想activity运行时才播放音乐,就可以在onCreate()方法里创建一个线程,在onStart()方法里启动该线程,在onStop()方法里关闭它。也可以考虑用AsyncTask或者HandlerThread,代替传统的Thread类。关于进程的更多信息详见-进程和线程文档。记住如果你使用了一个服务,默认它仍然会在主线程里运行,所以,遇到耗时或异步操作,可以创建一个新线程,然后再启动该服务。

要创建一个服务,先写一个子类继承Service。实现该子类时,需重载一些回调函数,这些函数处理一个服务的生命周期的各个关键部分,并为组件提供绑定该服务的机制,你可以有选择的重载这几个回调函数:

[onStartCommand\(\)](#)

当一个组件(例如activity)调用startService()方法,系统就会调用该方法(onStartCommand())启动一个服务。一旦此方法被

调用,服务立即启动,在后台一直执行。如果你实现该方法,该服务做的事情结束后,必须调用[stopSelf\(\)](#)或者[stopService\(\)](#),停止该服务。

[onBind\(\)](#)

当一个组件调用[bindService\(\)](#)想绑定服务,系统就会调用本方

法(`onBind()`)。在你的应用中，你必须为客户端提供和该服务交互的接口，该接口返回一个`IBinder`对象。该方法必须实现。如果不允许绑定这个服务，可以返回`null`即可。

[onCreate\(\)](#)

服务第一次创建，系统都会调用该方法，目的是一次性执行该过程(在调用`onStartCommand()`或`onBind()`方法前)。如果服务已经运行，方法不会被调用。

[onDestroy\(\)](#)

当服务不在使用被销毁时系统会调用该方法。应该实现该方法清除资源(例如线程，注册句柄，接收器等)，该方法最后调用。

如果一个组件调用`startService()`启动一个服务(会导致调用`onStartCommand()`)，那么服务会一直运行，直到自己调用`stopSelf()`或者其他组件调用`stopService()`停止服务。

如果一个组件调用`bindService()`创建服务(`onStartCommand()`没有调用)，那么服务一直运行，直到一个组件绑定他。一旦服务未被客户端绑定，系统会销毁它。

因为一个`activity`获得用户焦点回收系统资源，或者因为内存不够，系统会强行关闭一个服务。如果该服务和该`activity`绑定，那该服务被关闭就比较小，如果服务在后台运行(稍后专门讨论)，该服务几乎不可能被关闭，否则，如果服务已启动，运行了很长时间，随着时间的增加，系统就会降低该服务在后台任务队列中的级别，而且很有可能被干掉。如果一个服务已启动，你应该为该服务设计下重启时的相关处理。如果服务被关闭，当有可用资源时才会重启(尽管这依然依赖`onStartCommand()`函数的返回值，稍后讨论)，关于系统可能销毁服务的更多信息，请参考进程和线程文档资料。

下面，你将会看到怎么创建各种类型的服务以及怎么在组件中使用服务。

在`manifest`中声明一个服务

就像`activities`(和其他组件)，你必须在`manifest`中声明所有服务。

要声明一个服务，在`<application>`元素中添加一个`<service>`子元素即可。例如：

```
<manifest ... >
  ...
  <application ... >
    <service android:name=".ExampleService" />
  ...
</application>
</manifest>
```

定义形如请求启动服务的权限、服务所在的进程之类的属性，可以把这些属性包含在`<service>`标签中。`android:name`属性是必须定义的属性-它指定了服务的类名。一旦发布应用，这个名字就不能改动，因为这样做，你可能破坏某些功能，这些功能可能是某些确定的`intents`所引用的服务所提供的。(请阅读这篇博客-[Things That Cannot Change](#)).

关于声明服务的更多信息请参考引用`<service>`元素.

就像`activity`,一个服务可以通过定义`intent filters`，允许其他组件使用隐式的`intents`激活一个服务。这样做，一个应用中的组件可能启动一个服务，而这个服务中声明的`intent filter`可能匹配另一个应用(通过传递参数给`startService()`函数启动服务)中的`intent`。

如果你只想本地使用服务(其他应用不使用他).你就不必声明任何过滤符.没有这些，你就必须使用一个`intent`启动此服务，而且要指定具体的服务类的名字。下面具体讨论关于如何启动服务：

此外,仅当`android:exported="false"`时你才能确保服务对应用来说是私有的.即使你的服务提供`intent`过滤器，服务仍然私有.

想知道关于创建`intent filters`的更多信息,参考[Intents and Intents Filters](#)。

创建一个启动的服务

低于**android1.6**平台

如果系统版本低于1.6，你需要实现onStart()方法代替onStartCommand()。(在安卓2.0,onStart()不建议使用，更建议用onStartCommand().)

要查看对2.0以下版本的兼容性,请查看onStartCommand()文档

启动一个服务，通过另一个组件调用startService()方法即可.这导致调用onStartCommand()方法.

当一个服务启动,会有个生命周期，它独立于启动它的组件,服务可以后台一直运行,即使启动它的组件被销毁.因此,工作完成，服务应该调用stopSelf()自己关闭自己,或者另外一个组件调用stopService()函数停止该服务.

一个组件,像activity可以通过调用startService()方法启动服务,该方法须传递一个Intent对象,该Intent对象可以包含任何服务要使用的数据.服务在onStartCommand()方法中接受这个Intent对象.

举个例子,一些activity需要保存一些数据到一个在线数据库(online database)。可以保存数据到一个intent对象，然后再把这个intent对象当做参数传给startService()方法,调用startService启动服务。该服务在onStartCommand()方法中接收这个 intent对象，连接到互联网进行数据传输.传输完毕，服务就自动停止销毁.

注意:默认情况下，服务运行于它声明所在的应用进程中，并且在主线程中。所以如果服务做的事情很耗时或者可能导致阻塞操作,这时用户刚好又需要和应用交互,这样肯定导致整个应用性能降低，要避免这样，你应该在服务中新启动一个线程.</pre>

通常,有两种可用于继承的类来创建service:

Service:

这是所有服务类的基类,继承该类,对于在服务中创建新线程很重要.因

为默认服务使用应用的主线程，可能会降低程序的性能.

[IntentService](#):

这是一个[Service](#)的子类,该子类使用线程处理所有启动请求,一次一个.这是不使用服务处理多任务请求的最佳选择.你需要做的只是实现

[onHandleIntent\(\)](#)方法即可.可以为每个启动请求接收到intent,放到后台工作即可.

下面几段描述怎么用上面两个类实现服务.

继承[IntentService](#)类

因为大多数服务不必处理同时发生的多个请求.(多线程方案可能会很危险),所以最好用[IntentService](#)实现该服务.

[IntentService](#)类可以做这些事情:

- 从应用的主线程当中创建一个默认的线程执行所有的intents发送给[onStartCommand\(\)](#)方法,该线程从主线程分离.
- 创建工作队列,每次传递一个intent 给[onHandleIntent\(\)](#)方法实现,所以不必担心多线程.
- 所有的请求被处理后服务停止, 所以你永远都不必调用[stopSelf\(\)](#)函数.
- 默认实现[onBind\(\)](#)方法返回null.
- 默认实现[onStartCommand\(\)](#)方法是发送一个intent给工作队列, 然后发送给[onHandleIntent\(\)](#)方法的实现。

所有这些都基于: 实现[onHandleIntent\(\)](#)方法, 来处理客户端的请求的工作。(因此, 你还需要为服务提供一个小构造器).

这里给出[IntentService](#)类的一个实现:

```
public class HelloIntentService extends IntentService {
    /**
     * A constructor is required, and must call the super
     IntentService(String)
     * constructor with a name for the worker thread.
     */
}
```

```

public HelloIntentService() {
    super("HelloIntentService");
}

/**
 * The IntentService calls this method from the default worker
thread with
 * the intent that started the service. When this method returns,
IntentService
 * stops the service, as appropriate.
*/
@Override
protected void onHandleIntent(Intent intent) {
    // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.
    long endTime = System.currentTimeMillis() + 5*1000;
    while (System.currentTimeMillis() < endTime) {
        synchronized (this) {
            try {
                wait(endTime - System.currentTimeMillis());
            } catch (Exception e) {
            }
        }
    }
}

```

这就行了，一个构造函数，和onHandleIntent()方法的重载。

如果你打算也重载其他回调方法，例如onCreate(), onStartCommand(), 或者onDestroy(), 确保调用父类对应的方法，如此，IntentService实例化的对象才能正常工作。

例如, onStartCommand()必须返回默认实现的方法(意图就是通过它传递给onHandleIntent()的).

```

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting",
    Toast.LENGTH_SHORT).show();
    return super.onStartCommand(intent, flags, startId);
}

```

除了onHandleIntent(), 唯一不需要调用父类对应方法的方法就是onBind()(该方法是设置服务允许绑定时才实现).

接下来，你会看到，其实继承Service类和继承IntentService类如此相似，可能要多费点儿劲，但是如果你同时处理多个服务请求，这些多些的代码

是值得的。

继承Service类

正如之前所述,使用IntentService类,可以简化启动服务的代码.但是, 请求启动服务的是多线程同时请求(代替用一个工作队列处理这种情况),那么你就应该继承Service类处理这种情况.

比较而言, 下面的代码是Service类的子类, 用于执行一些具体的工作, 正如上面使用IntentService类。这样, 对于每个启动服务的请求,就开启一个线程处理该请求。

```

public class HelloService extends Service {
    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // 处理程序从线程中收到的消息
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {
            // 通常我们在这里做一些事情, 例如下载文件。
            // 这里, 我们只让程序睡眠5s
            long endTime = System.currentTimeMillis() + 5*1000;
            while (System.currentTimeMillis() < endTime) {
                synchronized (this) {
                    try {
                        wait(endTime - System.currentTimeMillis());
                    } catch (Exception e) {
                    }
                }
            }
            // 使用startId停止服务, 所以我们在处理其他任务的过程中不会停止该服务。
            stopSelf(msg.arg1);
        }
    }

    @Override
    public void onCreate() {
        // 启动线程开启服务, 注意默认服务通常运行在进程的主线程中, 所以我们创建了另一个线程。
        // 而且我们也不想阻塞程序。另外我们把该线程放在后台运行, 这样cpu耗时操作就不会破坏应用界面。
        HandlerThread thread = new
        HandlerThread("ServiceStartArguments",
                     Process.THREAD_PRIORITY_BACKGROUND);
        thread.start();

        // 获得线程的Looper, 用于我们的Handler
        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }
}

```

```

}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting",
    Toast.LENGTH_SHORT).show();
    // 对于每个启动请求,发送消息用于启动一个任务,传递start ID,这样当完成此任务时就知道谁发出该请求.
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    mServiceHandler.sendMessage(msg);
    // 如果服务已被杀死,执行return后,线程重启.
    return START_STICKY;
}

@Override
public IBinder onBind(Intent intent) {
    // 如果我们不提供服务绑定,返回null
    return null;
}

@Override
public void onDestroy() {
    Toast.makeText(this, "service done",
    Toast.LENGTH_SHORT).show();
}
}

```

如你所见，比起使用IntentService，使用Service是个苦力活.

但是，在onStartCommand()方法里因为是自己处理每个调用,所以你可以同时处理多个请求。例子中虽然没演示,但如果需要，你完全可以为每个请求创建一个线程，按正确的方式处理这些请求.(比起上面的IntentService,Service代替了需要等待上次请求处理完毕的方式。)

注意onStartCommand()方法必须返回一个整型变量.这个变量描述可能已被系统停止的服务，如果被停止，系统会接着继续服务(正如下面讨论，IntentService的实现中默认自动帮你做了个处理，尽管可以修改它),返回值必须是如下几个常量之一:

START_NOT_STICKY

如果系统在onStartCommand()返回后停止服务,系统不会重新创建服务,除非有等待的意图需要处理.如果想避免运行不需要的服务，或者让应用可以轻松的启动未完成的任务，这是一个安全的选择。

START_STICKY

如果系统在onStartCommand()返回后停止服务,系统重新创建服务并且调用onStartCommand()函数,但是不要传送最后一个意图。相反,系统用一个空意图调用onStartCommand(),除非还有想启动服务的意图,这种情况下,这些意图会被传递.这很适合多媒体播放的情况(或类似服务),这种情况下服务不执行命令,但是会无限运行下去,等待处理任务。

START_REDELIVER_INTENT

如果系统在onStartCommand()返回后停止服务,系统使用用最后传递给service的意图,重新创建服务并且调用onStartCommand()方法,任何未被处理的意图会接着循环处理。所以用服务处理像下载之类的可能需要立即重启的任务,非常合适。

关于这些返回值的更多信息,请参考文档for each constant.

启动服务

你可以从一个activity或者其他应用组件通过传递一个intent(指定要启动的服务)给startService()方法.系统调用服务的onStartCommand()方法并且把意图传给此方法.(千万不能直接调用onStartCommand()方法).

下面,我们演示一下用具体的意图传递给方法startService()来启动一个服务:

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

startService()方法立即返回,安卓系统会调用onStartCommand()方法。如果服务没有运行,系统首先调用onCreate()方法,然后在调用onStartCommand()方法。

如果系统不提供绑定,通过传递意图给startService()方法是应用组件和服务唯一的沟通方式。但是如果你想服务发送一个返回结果,可以让客户端使用广播(调用getBroadcast()方法获取)创建一个PendingIntent启动一个服务,然后把该意图传递给要启动的服务,服务就可以使用此广播传递结果了。

并发请求启动服务导致并发响应调用**onStartCommand()**函数,但是,唯一停止服务的方法就是调用**stopSelf()**或者**stopService()**方法。

停止服务

一个启动的服务必须自己管理自己的生命周期.就是说,系统不会停止或销毁服务,除非系统内存被覆盖或者服务在**onStartCommand()**返回之后还在运行。所以,服务自己调用**stopSelf()**或其他组件调用**stopService()**方法都会停止服务.

一旦调用了**stopSelf()**或**stopService()**,系统会尽快销毁服务。

但是,当前如果服务在**onStartCommand()**方法中同时处理多个请求,那么当你正在处理这样的启动请求时,肯定不应该停止服务,因为很有可能在停止服务时刚好收到一个新的请求(在第一个请求的结尾停止服务会终止第二个请求).要避免这个问题,你可以使用**stopSelf(int)**方法确保服务被停止,而且这个服务是最经常启动的。就是说,当调用**stopSelf(int)**,你需要传递给请求启动的ID(这个ID传递给 **onStartCommand()**方法)给对应的服务.如果在可以调用**stopSelf(int)**之前,服务接收到一个新的启动请求,服务就不会匹配,也就不会停止。

注意:当服务完成任务后停止它,非常重要,因为这样做可以避免浪费系统资源和节约电量.如果有必要,其他组件可以调用 **stopService()**方法停止服务。即使你绑定服务,如果该服务收到一个**onStartCommand()**的调用,就必须停止服务。

关于服务的生命周期的更多信息,请参考下面的关于"管理服务的生命周期"(Managing the lifecycle of a Service).</pre>

创建一个已绑定的服务

一个已绑定的服务是指应用组件调用**bindService()**绑定该服务,目的是创建和这个服务的长连接(通常不允许组件调用 **startService()**启动该服务).

当你想和应用组件启动的服务交互,或对其他应用提供部分功能,你应该创

建一个绑定服务。要创建绑定服务，必须实现**onBind()**回调函数，返回一个**IBinder**对象，定义和该服务交互的接口.其他组件就可以调用包括**bindService()**服务接口.该服务就仅对和它绑定的组件起作用，所以当没有组件和服务绑定，系统会自动销毁该服务(当通过调用**onStartCommand()**方法启动一个绑定的服务后，你不必强行关闭它.)

创建一个绑定的服务，第一件必做的事情就是定义接口指定客户端如何和服务交互.这个接口必须返回**IBinder**对象，并且必须从**onBind()**方法中返回，一旦客户端接收到**IBinder**对象，就可以通过它与服务交互。

多客户端可以一次性和服务绑定.当一个客户端和服务交互完毕，就会调用**unbindService()**解除绑定.一旦没有客户端绑定服务，系统会销毁服务。实现一个绑定的服务有多种方式,(有些)具体实现比实现一个启动的服务更复杂,所以针对绑定服务，这里专门列出了一个专题 论述"绑定服务".

发送用户通知

一旦运行，服务可以用**Toast Notifications** 或者**Status Bar Notifications**通知用户一些事情.

一个**toast notification**通常是在当前窗口弹出一个小的消息提示，过一会儿就消失了。而状态条提供了带图标的提示信息,用户可以选择，然后执行一个操作(例如启动**activity**)。

通常一些后台任务完成后用状态条是最合适的(例如文件下载完成),用户可以和它交互,当用户从展开的视图中选择一个状态条 就可以启动一个**activity**.(例如查看下载文件).

请参考**Toast Notifications** 或者**Status Bar Notifications**开发者文档获得更多内容.

前台运行一个服务

前台服务是一种考虑到系统内存不足，但是用户已经意识到而且不想关闭的服务。前台服务必须用状态条做出提示，表示该服务在持续进行中,意思是这种提示除非服务停止或者从所有的前台服务中移除才能消失. 例如,用服务操作音乐播放器播放音乐,应该在前台播放，因为用户明确指出了具体操作(就是播放音乐)。状态条可以包含当前 播放的音乐，并且允许用户启动

一个activity和音乐播放器交互.

要请求服务在前台运行，调用startForeground()方法.这个方法带两个参数:一个整型变量,表示提示信息的唯一标识符,另一个Notification对象,表示状态栏,示例代码:

```
Notification notification = new Notification(R.drawable.icon,
getText(R.string.ticker_text),
System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);
notification.setLatestEventInfo(this,
getText(R.string.notification_title),
getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION, notification);
```

要从前台移除服务，请调用stopForeground().这个方法需要一个布尔参数,表示是否清除状态栏的信息.调用它，本身不会停止服务.但是,服务仍然在后台运行，你突然停止服务,状态栏的提示就会移除。注意:startForeground()和stopForeground()在安卓2.0中介绍过(API 等级5).要在旧平台的前台运行服务,就必须事先调用setForeground(),关于如何向后兼容，请查看startForeground()。

关于notifications的更多信息,查看"创建一个状态栏提示"(Creating Status Bar Notifications).

管理服务的生命周期

服务的生命周期比活动的生命周期简单多了。但是，你应该更多地关注如何创建、销毁服务，因为服务可以在用户没有意识到的情况下在后台运行。

服务的生命周期--从他被创建到销毁-会遵循如下2种方式之一:

启动的服务 当另一个组件调用startService()，服务就被创建,然后一直运行下去，直到服务自己调用stopSelf()方法停止该服务.其他组件也可以通过调用stopService()方法停止一个服务.系统会销毁该服务。

绑定的服务 当另一个组件(客户端)调用bindService()，服务就被创建,客户端然后通过IBinder接口和服务交互.客户端可以调用unbindService()关闭和

服务的连接.多客户端可以绑定到相同的服务，所有的客户端都解除绑定之后，系统销毁该服务.(服务不必像上面那样自己销毁).

这两种方式不是完全不同的，就是说，你绑定一个已经调用startService()启动的服务。例如,一个后台音乐服务可以通过把一个意图传递给startService()启动。稍后，用户可能执行某些操作或获得当前歌曲的信息，一个activity通过调用bindService()绑定某个服务.所有的客户端全都取消绑定之前,调用stopService()或stopSelf()方法不会停止服务.

实现生命周期中的接口 就像activity,服务也有生命周期回到函数,这些函数用于监视改变服务的状态，然后让服务适时执行某些工作。下面的服务就实现了所有这些方法:

```

public class ExampleService extends Service {
    int mStartMode;           // indicates how to behave if the service
is killed
    IBinder mBinder;         // interface for clients that bind
    boolean mAllowRebind;    // indicates whether onRebind should be
used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
{
        // The service is starting, due to a call to startService()
        return mStartMode;
}
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
}
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
}
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
}
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
}
}

```



注意:不像实现activity的生命周期回调方法，你不需要在实现服务的这些方法时调用父类的方法。

通过实现这些接口，你可以监控服务生命周期的两个循环：

服务的整个生命周期是从调用onCreate()开始，直至onDestroy()方法返回结束。就像activity,一个服务是在onCreate()方法中初始化，在onDestroy()方法中释放资源。例如，一个music后台播放的服务，会在onCreate()方法里创建线程播放音乐，在onDestroy()方法里停止该线程。

服务的活动周期是从调用onStartCommand()或者onBind()函数开始，这两个

方法是当意图传给startService()或者bindService()才会被调用。

如果服务已经启动,活动周期和整个生命周期一起结束(即使onStartCommand()函数返回后,服务仍然处于活动状态)。如果服务被绑定,在onUnbind()返回后活动周期结束.

注意:尽管调用stopSelf()或者stopService()可以停止服务,但停止时不会有回调方法被调用.(没有onStop()回调),所以,除非服务绑定客户端,系统会销毁该服务,这时, onDestroy()方法是唯一被回调的方法.

图2展示了服务的两种典型的回调方法流程。尽管一个用startService()创建服务,一个用bindService()创建,但记住,任何服务,不论如何启动,都允许客户端绑定它。所以一个用onStartCommand()(客户端方式调用startService())初始化的服务可以调用onBind()方法(客户端方式就调用bindService())。

想知道更多关于创建带绑定的服务,查看绑定服务,这个文档包含了跟多关于回调onRebind()的内容,该文档包含于管理绑定服务的生命周期.(Managing the Lifecycle of a Bound Service.)

来自 "[index.php?title=Services&oldid=13769](#)"



Bound Services

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/intl/zh-CN/guide/components/bound-services.html>

翻译：--Snowxwyo 2012年7月12日 (四) 14:20 (CST)

目录

- [1 绑定服务-Bound Services](#)
 - [1.1 基础知识-The Basics](#)
 - [1.2 创建绑定服务-Creating a Bound Service](#)
 - [1.2.1 扩展binder类-Extending the Binder class](#)
 - [1.2.2 使用一个消息传递器-Using a Messenger](#)
 - [1.3 绑定到服务-Binding to a Service](#)
 - [1.3.1 补充注释-Additional Notes](#)
 - [1.4 管理绑定服务的生命周期-Managing the Lifecycle of a Bound Service](#)

绑定服务-Bound Services

一个绑定的服务是客户服务器接口上的一个服务器。一个绑定的服务允许组件（如：活动）来绑定一个服务，传送请求，接收响应，甚至执行进程间的通信（IPC）。绑定服务通常只生存在其服务于另一个程序组件时，并

快速浏览

- 绑定服务允许其他组件绑定它，为了能够与其互动，并执行进程间通信

且不会无限期的在后台运行。

这篇文章将向你展示怎么创建一个绑定服务，包括怎么样从其他应用程序组件绑定到服务。然而你也应该查看[服务-Services](#)文档，更多关于普通情况下服务的额外信息，如怎么从一个服务传递通知，设备服务在前台运行等。

基础知识-The Basics

绑定服务是一个[Service](#)类的一个实现，允许其它服务绑定到它并与之互动。为服务提供绑定，你必须实现[onBind\(\)](#)回调方法。这个方法返回一个[IBinder](#)对象，它定义了程序接口，用户可以用其与服务交互。

一个客户可以通过调用[bindService\(\)](#)来绑定到一个服务。当这么做时，必须提供一个[ServiceConnection](#)的实现，它监视着与服务的连接。[bindService\(\)](#)方法立刻返回了一个空值。但是当Android系统在用户和服务器之间创建一个连接时，它将会在[ServiceConnection](#)上调用[onServiceConnected\(\)](#)方法来传

- 当所有的客户端被解绑时，绑定服务被摧毁，除非那个服务还在运行。

本文内容

[基础知识-The Basics](#)

[创建绑定服务-Creating a Bound Service](#)

[扩展binder类-Extending the Binder class](#)

[使用一个消息传递器-Using a Messenger](#)

[绑定到服务-Binding to a Service](#)

[管理绑定服务的生命周期-Managing the Lifecycle of a Bound Service](#)

[关键类](#)

[Service](#)

[ServiceConnection](#)

[IBinder](#)

[范例](#)

[RemoteService](#)

[LocalService](#)

[其它](#)

递 **IBinder** 对象，使用户可以与服务通信。

服务-Services

多个用户可以同时连接到一个服务。但是，系统只以第一个用户绑定时调用 **onBind()** 方法来获得 **IBinder** 对象。然后，系统将同一个 **IBinder** 对象传递给后续绑定的客户，并不会重新调用 **onBind()** 方法。

当最后一个用户从服务上解绑时，系统摧毁这个服务（除非这个服务由 **startService()** 启动）。

当你实现绑定服务时，最重要的就是定义 **onBind()** 回调方法返回的接口。你可以使用有一些不同的方法来定义服务的 **IBinder** 接口，接下来的章节将讨论每一项技术。

创建绑定服务-Creating a Bound Service

当创建一个提供了绑定的服务时，你必须提供一个 **IBinder** 来提供程序接口，用户可以用这个接口与服务交互。有三种方法可以用来定义这个接口：

扩展binder类-Extending the Binder class

如果你的服务只对你自己的应用程序私用，并且作为客户在同一个进程中运行（这种情况很常见），你应该通过扩展 **Binder** 类来创建你的 **onBind()**

绑定到一个启动的服务

如同在 服务-Services 文档中讨论的，你可以创建一个服务，同时是被启动的和被绑定的。也就是说，服务可以通过调用 **startService()** 被启动，其允许服务无限期的运行，同时也允许一个用户通过调用 **bindService()** 方法绑定到服务。

如果你允许服务被启动和被绑定，那么，当服务被启动时，系统在所有用户解绑时不会摧毁这个服务。而是，你必须明确的调

接口并且从一个实例。客户接收[Binder](#)并直接使用它来接入[Binder](#)实现，甚至[Service](#)中可用的公共方法。

当服务对于你的应用程序几乎是运行于后台时，这是首选技术。这种情况下，唯一一个你不建立自己接口的原因是你的服务被其他应用程序所用或使用了多个分开的进程。

[使用一个消息传递器-Using a Messenger](#)

如果你想让你的接口在多个不同的进程间工作，你可以为服务创建一个带有[Messenger](#)的接口。在这种情况下，服务将定义一个[Handler](#)来响应不同类型的[Message](#)对象。这个[Handler](#)是[Messenger](#)的基础，它可以与客户共享一个[IBinder](#)，允许客户使用[Message](#)对象向服务发送指令。以此，客户可以定义一个属于自己的[Messenger](#)，这样，服务就可以把消息传递回来。

这是最简单的执行进程间通信（IPC）的方法，因为[Messenger](#)队列所有的请求到一个单独的线程当中，所以你不需要设计你的服务为线程安全（thread-safe）

用[stopSelf\(\)](#)或[stopService\(\)](#)方法来停止服务。

虽然，通常情况下，你应该实现[onBind\(\)](#)或[onStartCommand\(\)](#)方法，但是，有时必须同时实现两个方法。例如，一个音乐播放器可能会需要无限期的运行，同时提供绑定。通过这种方法，活动可以启动这个服务来播放音乐，并且音乐可以持续播放，即使用户离开了这个应用程序。那么，当用户返回这个应用程序时，活动可以绑定到服务来重新获得播放的控制。

请确认仔细阅读了关于[管理绑定服务生命周期-Managing the Lifecycle of a Bound Service](#)的章节，查看更多关于当添加绑定到一个启动的服务时服务生命周期的信息。

使用AIDL

AIDL (Android接口定义语言-Android Interface Definition Language) 执行了把一个对象分解到操作系统能理解的基元，并安排它们到各个进程间来完成IPC等所有工作。前文中提到的技术，使用一个[Messenger\(消息传送器\)](#)就是基于AIDL和其下面的结构。如前所述，[Messenger](#)创建了一个队列，把所有的请求都放在一个线程中，所以服务一次只接收一个请求。然而，如果想你的服务同时接收多个请求，那么你可以直接创建AIDL。在这种情况下，你的服务必须有能力执行多个纯程，并且为线程安全。

为了直接使用AIDL，你必须创建一个[.aidl](#)文档，其定义了编程接口。Android SDK工具包使用这个文件来生成一个抽象类，并实现了那个接口来处理IPC。你可以在你的服务中扩展使用。

- 注解：大多数应用程序不应该使用AIDL来创建一个绑定服务，因为它可能要求能够支持多线程，其结果将会是很复杂的实现。例如，AIDL不适用于大多数应用程序，这篇文档并不讨论怎样使用。如果你确定你需要AIDL，请见[AIDL](#)文档。

扩展binder类-Extending the Binder class

如果你的服务只被本地应用程序所使用，并且不需要在多个进程间工作，那么你可以实现你自己的[Binder](#)类，让你的客户可以直接接入方法中的公共方法。

- 注解：这个方法只有在客户和服务在同一个应用程序和进程中时才可行，这种情况很常见。例如，这个方法对于一个音乐应用程序将会非常有用，它需要绑定一个活动到它自己的服务用来以后台播放音乐。

以下为如何设定这个**binder**类：

1. 在你的服务中，创建一个[Binder](#)类的实例，实现以下功能之一：

- 包含客户可以调用的公共方法
- 返回当前[Service](#)的实例，其包含了用户可以访问的公共方法
- 或返回这个服务包含的另一个类，并含有客户可以访问的公共方法

2. 从[onBind\(\)](#)回调函数返回这个[Binder](#)的实例。

3. 在客户端，从[onServiceConnected\(\)](#)回调方法接收这个[Binder](#)，并用提供的方法来调用绑定服务。

- 注解：服务和客户端必须在同一个应用程序中的原因是客户端可以计算返回的对象并恰当的调用其APIs。服务和客户端也必须在同一个线程的原因是这种技术不能执行线程间操作。

例如，以下为一个为客户端提供了通过[Binder](#)实现接入服务中方法的服务范例：

```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder. Because we know this
     * service always
     * runs in the same process as its clients, we don't need to
     * deal with IPC.
     */
    public class LocalBinder extends Binder {
```

```

        LocalService getService() {
            // Return this instance of LocalService so clients can
            call public methods
            return LocalService.this;
        }
    }

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

/** method for clients */
public int getRandomNumber() {
    return mGenerator.nextInt(100);
}
}

```

LocalBinder为客户端提供了**getService()**方法来取得当前的**LocalService**实例。这个允许客户端调用服务中的公共方法。例如，客户端可以从服务中调用**getRandomNumber()**。

以下为，当一个按钮被点击时，一个绑定到**LocalService**的活动并调用**getRandomNumber()**方法：

```

public class BindingActivity extends Activity {
    LocalService mService;
    boolean mBound = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        // Bind to LocalService
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Unbind from the service
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }
}

```

```

    /** Called when a button is clicked (the button in the layout
file attaches to
     * this method with the android:onClick attribute) */
public void onButtonClick(View v) {
    if (mBound) {
        // Call a method from the LocalService.
        // However, if this call were something that might hang,
then this request should
        // occur in a separate thread to avoid slowing down the
activity performance.
        int num = mService.getRandomNumber();
        Toast.makeText(this, "number: " + num,
Toast.LENGTH_SHORT).show();
    }
}

/** Defines callbacks for service binding, passed to
bindService() */
private ServiceConnection mConnection = new ServiceConnection()
{
    @Override
    public void onServiceConnected(ComponentName className,
        IBinder service) {
        // We've bound to LocalService, cast the IBinder and get
LocalService instance
        LocalBinder binder = (LocalBinder) service;
        mService = binder.getService();
        mBound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName arg0) {
        mBound = false;
    }
};

```

以上范例中展示了客户端怎样通过使用一个[ServiceConnection](#)的实现和[onServiceConnected\(\)](#)回调方法绑定到一个服务。后续章节中提供了更多的关于绑定到服务流程的信息。

更多实例代码，请见[ApiDemos](#)中[LocalService.java](#)和[LocalServiceActivities.java](#)类。

使用一个消息传递器-Using a Messenger

如果你需要你的服务能够与远程进程通信，那么你可以使用一个[Messenger](#)为你的服务提供接口。这个方法允许你执行进程间通信（IPC）而不需要使用AIDL。

以下为怎么样使用[Messenger](#)的总结：

- 服务实现了一个[Handler](#)，用来接收每一次调用从客户端返回的回调方法。
- [Handler](#)被用来创建一个[Messenger](#)对象（其为[Handler](#)的一个引用）。
- [Messenger](#)创建一个[IBinder](#)，服务从[onBind\(\)](#)方法将其返回给客户端。
- 客户端使用这个[IBinder](#)来实例化这个[Messenger](#)（其引用到服务的[Handler](#)），客户端可以用来向服务发送[Message](#)对象。
- 服务通过它的[Handler](#)接收每一个[Message](#)——更确切的说，是在[handleMessage\(\)](#)方法中接收。

通过这种方法，在服务端没有客户端能调用的“方法”。而是，客户传递“消息”（[Message](#)对象），同时服务在其[Handler](#)中接收。

以下为服务使用一个[Messenger](#)接口的简单范例：

对比AIDL

当你需要执行IPC时，为你的接口使用一个[Messenger](#)比用AIDL实现简单，因为[Messenger](#)把所有对此服务的调用都排在一个队伍中，而单纯的AIDL接口不断的各服务发送请求，这样就要求此服务具备处理多线程的能力。

对于大多数应用程序，服务并不需要执行多线程，所以，使用一个[Messenger](#)允许服务在同一时间只处理一个调用。如果处理多线程对你的服务很重要，那么，你应该使用AIDL来定义你的接口。

```

public class MessengerService extends Service {
    /** Command to the service to display a message */
    static final int MSG_SAY_HELLO = 1;

    /**
     * Handler of incoming messages from clients.
     */
    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_SAY_HELLO:
                    Toast.makeText(getApplicationContext(),
"hello!", Toast.LENGTH_SHORT).show();
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    /**
     * Target we publish for clients to send messages to
     * IncomingHandler.
     */
    final Messenger mMessenger = new Messenger(new
IncomingHandler());
}

/**
 * When binding to the service, we return an interface to our
messenger
 * for sending messages to the service.
 */
@Override
public IBinder onBind(Intent intent) {
    Toast.makeText(getApplicationContext(), "binding",
Toast.LENGTH_SHORT).show();
    return mMessenger.getBinder();
}
}

```

注意[Handler](#)中的[handleMessage\(\)](#)方法，服务在其中接收进入的[Message](#)，基于[what](#)成员，决定下一步的做法。

客户端所需做的只是基于服务返回的[IBinder](#)创建一个[Messenger](#)并使用[send\(\)](#)方法发送一条消息。例如，以下为一个简单的活动范例，其绑定到了服务，并向服务传递了[MSG_SAY_HELLO](#)消息：

```

public class ActivityMessenger extends Activity {
    /** Messenger for communicating with the service. */
    Messenger mService = null;

    /** Flag indicating whether we have called bind on the service.

```

```
/*
    boolean mBound;

    /**
     * Class for interacting with the main interface of the service.
     */
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className,
IBinder service) {
            // This is called when the connection with the service
has been
            // established, giving us the object we can use to
            // interact with the service. We are communicating with
the
            // service using a Messenger, so here we get a client-
side
            // representation of that from the raw IBinder object.
            mService = new Messenger(service);
            mBound = true;
        }

        public void onServiceDisconnected(ComponentName className) {
            // This is called when the connection with the service
has been
            // unexpectedly disconnected -- that is, its process
crashed.
            mService = null;
            mBound = false;
        }
    };

    public void sayHello(View v) {
        if (!mBound) return;
        // Create and send a message to the service, using a
supported 'what' value
        Message msg = Message.obtain(null,
MessengerService.MSG_SAY_HELLO, 0, 0);
        try {
            mService.send(msg);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        // Bind to the service
        bindService(new Intent(this, MessengerService.class),
mConnection,
        Context.BIND_AUTO_CREATE);
    }

    @Override
```

```
protected void onStop() {
    super.onStop();
    // Unbind from the service
    if (mBound) {
        unbindService(mConnection);
        mBound = false;
    }
}
```

注意这个范例中并没有展现服务将怎样响应客户端。如果你希望服务做出反应，那么你需要在客户端创建一个[Messenger](#)当客户端接收到[onServiceConnected\(\)](#)回调方法，它发送一条[Message](#)到服务，其中包括了客户端的[Messenger](#)，其存放在在[send\(\)](#)中[replyTo](#)参数之中。

你可以在[MessengerService.java](#)(服务端)
和[MessengerServiceActivities.java](#) (客户端) 提供双向通信，具体请见相关
范例程序。

绑定到服务 - Binding to a Service

应用程序组件（客户端）可以通过调用[bindService\(\)](#)方法绑定到一个服务。Android系统将调用服务的[onBind\(\)](#)方法，返回一个[IBinder](#)用来与服务交互。

这个绑定是不同步的。[bindService\(\)](#)立即返回，但并没有返回[IBinder]给用户。为了接收[IBinder](#)，客户端必须创建一个[ServiceConnection](#)的实例，并传递给[bindService\(\)](#)。[ServiceConnection](#)包含了一个回调方法，系统调用它来传递[IBinder](#)。

- 注解：只有活动（activities），服务（services），和内容提供者（content providers）可以绑定到一个服务——你不能从一个广播接收器（broadcast receiver）绑定到一个服务。

所以，从你的客户端绑定到服务，你必须：

1. 实现ServiceConnection。

你的实现必须复写两个回调方法：

onServiceConnected()

系统调用这个方法来传递由服务端的onBind()方法返回的IBinder。

onServiceDisconnected()

当与服务的连接发生了不可预期的丢失，Android系统调用这个方法，例如，服务发生了冲突或被杀死。在服务被解绑时并不会调用这个方法。

2. 调用bindService()，传递ServiceConnection的实现。

3. 当系统调用onServiceConnected()回调方法时，你可以开始使用由接口定义的方法调用服务。

4. 与服务解决连接，调用unbindService()。

当客户端被摧毁，它将会从服务解绑，但你应该始终在结果与服务的交互或当你的活动暂停时解绑，这样系统可以在不被使用时关闭。

(后续文章将更加具体讨论适当的绑定和解绑时间。)

例如，以下的范例小片断，使客户端连接到使用前文讨论的扩展binder类-Extending the Binder class方法创建的服务，所以，唯一需要做的就是把返回的IBinder传递给LocalService类，并请求LocalService的实例：

```
Bound Services - eoeAndroid wiki
LocalService mService;
private ServiceConnection mConnection = new ServiceConnection() {
    // Called when the connection with the service is established
    public void onServiceConnected(ComponentName className, IBinder
service) {
        // Because we have bound to an explicit
        // service that is running in our own process, we can
        // cast its IBinder to a concrete class and directly access
it.
        LocalBinder binder = (LocalBinder) service;
        mService = binder.getService();
        mBound = true;
    }

    // Called when the connection with the service disconnects
unexpectedly
    public void onServiceDisconnected(ComponentName className) {
        Log.e(TAG, "onServiceDisconnected");
        mBound = false;
    }
};
```

使用这个[ServiceConnection](#),客户端可以通过把其传递到[bindService\(\)](#)来绑定到一个服务。如：

```
Intent intent = new Intent(this, LocalService.class);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

- [bindService\(\)](#)的第一个参数，是一个[Intent](#)，其明确的指出了被绑定服务的名字（[intent](#)被认为可以暗示性的指出）。
- 第二个参数是[ServiceConnection](#)对象。
- 第三个参数是一个标志位，指示了绑定的选项。通常情况下应该为[BIND_AUTO_CREATE](#)，为了能够在服务不存在的情况下创建一个服务。其它可选值有[BIND_DEBUG_UNBIND](#)和[BIND_NOT_FOREGROUND](#)，或0表示什么都没有。

补充注释-Additional Notes

以下为绑定到一个服务时的几个重要的注意事项：

- 你应该始终捕获[DeadObjectException](#)异常，这个异常在连接断开时被抛出。在这远程方法唯一会抛出的异常。
- 对象为进程引用数。
- 在客户端生命周期的创建和摧毁时刻，你应该成对使用绑定和解绑。例如：
 - 当你的活动可见时，如果你需要做的只是与服务交互，你应该在[onStart\(\)](#)中绑定，在[onStop\(\)](#)中解绑。
 - 如果你希望即使活动在后台停止时也接收响应，那么你可以在[onCreate\(\)](#)方法中绑定，并在[onDestroy\(\)](#)方法中解绑。需要注意的是，这个实现使得你的活动需要在所有服务运行的时候占用该服务（即使在后台运行时）。所以，如果服务存在于另一个进程中，那么你增加了进程的比重，因此，系统也将更有可能性将其杀死。

- 注解：通常情况下，你不应该在活动的[onResume\(\)](#)和[onPause\(\)](#)方法中绑定与解绑，因为这些回调方法发生在生命周期转换的时候，你应该把此期间的发生的进程减小到最少。同样，如果有多个活动绑定到同一个服务，两个活动间必然存在着转换，当当前的活动在下一个活动绑定（在resume期间）之前，进行解绑（在pause期间），服务可能会被摧毁或重新创建。（这个活动转换中，关于活动如何协调其生命的情况，在[Activities](#)）文档中做了详细的表述。

更多的关于怎么样绑定服务的范例代码，请见[ApiDemos](#)中的[RemoteService.java](#)类。

管理绑定服务的生命周期-**Managing the Lifecycle of a Bound Service**

当一个服
务从所有
客户端解



图1.服务的生命周期被启动，并允许绑定。

绑，Android系统将将其摧毁（除非这个活动被[onStartCommand\(\)](#)启动）。这样，如果你的服务只是纯粹一个绑定服务，那么你不需要自己管理其生命周期——Android系统将根据其是否被绑定到客户端，来管理其生命周期。

然而，如果你选择实现[int, int\) onStartCommand\(\)](#)回调方法，那么你必须明确的停止这个服务，因为服务现在被认为已被启动。在这种情况下，服务将一些运行，直到它被自身的[stopSelf\(\)](#)方法停止，或其它活动组件调用[stopService\(\)](#)方法，而不考虑其是否绑定到客户端。

此外，如果你的服务被启动，并接收绑定，那么当系统调用了[onUnbind\(\)](#)方法，你可以有选择性的返回true，如果你希望下一次一个客户端绑定到服务时接收一个[onRebind\(\)](#)方法调用（而不是接收一个[onBind\(\)](#)调用）。[onRebind\(\)](#)返回一个空值，但客户端依然可以接到[onServiceConnected\(\)](#)回调方法中的[IBinder](#)。图1展示了这种服务生命周期的逻辑。

更多关于一个启动的服务的信息，请见[Services](#)文档。

来自“[index.php?title=Bound_Services&oldid=13714](#)”

Android Interface Definition Language (AIDL)

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：sfshine

原文链接：<http://developer.android.com/intl/zh-CN/guide/components/aidl.html>

目录

[[隐藏](#)]

[1 Android接口定义语言\(AIDL\)](#)

- [1.1 声明一个AIDL接口](#)
 - [1.1.1 1. 创建一个.aidl](#)
 - [1.1.2 2. Implement the interface](#)
 - [1.1.3 3. Expose the interface to clients-为客户端公开接口](#)
- [1.2 Passing Objects over IPC](#)
- [1.3 调用一个IPC方法](#)

Android接口定义语言(AIDL)

Android接口定义语言(以下简称AIDL)和其他您使用过得IDL差不多，他使您可以定义应用程序的接口。通过这个接口，客户端和服务器之间可以顺利的进行进程间通讯（IPC）。在android设备上，一个进程的不能通过正常的方式访问其他进程的内存，也就是说，必须把需要通信的对象翻译成 操作系统可以识别的原语，通过这些原语才能穿越这些对象的外表，从而得到该物体的内部信息。如果使用用代码来做这些事情，那将非常枯燥和乏味，因此，android为您提供了AIDL。

注意：只有您允许来自不同应用的客户端访问您的IPC服务并且您希望在服务中处理多线程，使用AIDL才是必要的。如果您不需要使用并发的IPC访问不同的应用，您应该通过[继承Binder](#)来创建您的接口，或者，如果您确实需要使用IPC，但是不需要处理多线程，那请[继承Messenger](#)来实现您的接口。总之，在实现一个AIDL之前，请确保您已经理解了[Bound Services](#)。

在您开始设计您的AIDL接口的时候，请注意AIDL接口的调用都是直接的函数调用。请不要想当然的认为这个调用是发生在线程里。具体的情况会取决于这个调用发生在本地进程还是远程进程。描述如下：

- 发生在本地进程的调用会在发生这个调用的线程里执行。如果这是您的主UI线程，这个线程将会持续的在AIDL接口接口中执行。如果这是其他的线程，那这个线程将会是在服务中执行您的代码的那个。那么，只有本地线程在访问服务，您才能完全控制是那些线程在这个服务里发生（但是如果是这种情况，那您不应该使用AIDL，而是通过[继承Binder](#)创建一个接口）。
- 发生在远程的调用是在该平台所维护的，您的进程内部的一个线程池中发送的。您必须准备好从未知的线程里接收调用，与此同时，多重调用也会发生。换句话说，AIDL接口的实现必须是完全线程安全的。
- 单向 (oneway) 的关键字传输决定了远程调用的行为。使用时，一个远程调用不会被阻塞，它只是发送需要交流的数据然后立即返回。这个接口的实现最终就像一个来自[Binder](#)线程池的调用一样作为一个普通的远程调用而被接受。如果单向是在本地被调用的，那他将不会有任何影响，而这个调用是一直同步的。

声明一个AIDL接口

您必须在一个.aidl格式的文件里使用java程序语言的语法声明AIDL接口，并且把它放到承载这个服务的应用和绑定这个服务的应用的源代码的文件夹(文件夹src/)下。

当您创建一个含有.aidl格式的文件的应用时,Android SDK工具会基于这

个.aidl文件自动生成一个**IBinder**的接口并且把它保存到工程的gen文件夹下。这个服务必须正确的继承**IBinder**接口。这样客户程序就可以绑定这个服务，从而可以从**IBinder**中调用方法来执行IPC。

使用一下步骤通过AIDL建立一个边界服务：

1. [创建一个.aidl](#):

这个文件使用签名方法定义一个一应用程序接口。

2. [继承这个接口](#):

这个Android SDK工具使用Java应用程序语言基于您的.aidl文件生成一个接口。这个接口有一个叫**Stub**的内部抽象类，这个类继承自**Binder**并且实现您的AIDL接口中的方法。您必须继承这个**Stub**类并且实现这些方法。

3. [Expose the interface to clients](#)

把接口暴露给客户端：继承一个[Service](#)并且重写[onBind\(\)](#)来返回您实现的**Stub**类。

注意:为了避免打断使用您服务的其他应用，在您第一次打开AIDL接口的时候，您对AIDL接口的任何改变必须保持向后兼容。这就是说，您必须保证对原始接口的支持，因为为了可以访问您的服务接口，您的.aidl文件已经被复制到了其他应用程序里面。

1. 创建一个.aidl

AIDL使用一个简单的语法，让你一个接口与一个或多个参数和返回值的方法，可以申报。参数和返回值可以是任何类型，甚至其他AIDL生成的接口。 AIDL使用一种简单的语法使您使用一个或者多个可以输入参数输出返回值的方法来声明一个接口。这些参数和返回值可以是任何数据类型，甚至是AIDL生成的接口。 您必须使用java语言构建一个.aidl文件。每个.aidl文件必须定义一个单独的接口，并且使用确定的声明和方法签名。

By default, AIDL supports the following data types:

详细来说，AIDL支持下面的数据类型：

- 所有的java语言的基本数据类型，比如int, long, char, Boolean 等等。
- [String 字符串](#)
- [CharSequence 字符数组](#)
- [List](#)

[List](#)中的所有元素必须是[list](#)所支持的数据类型中的一种，或者是您已经声明的其他AIDL接口或Parcelables数组之一。一个List有时可能被当作一个“通用”类来使用（比如List<String>）。这些类具体实现的时候，对方接收的通常是[ArrayList](#)，即使该方法继承[List](#)的接口生成的。

- [Map](#)

在[map](#)中的所有元素必须是[list](#)所支持的数据类型中的一种，或者是您已经声明的其他AIDL接口或Parcelables数组之一。一般的Map，（比如形如Map<String, Integer>一类）是不被 支持的。这些类具体实现的时候，对方接收的通常是[HashMap](#)，即使该方法是继承[Map](#)的接口生成的。

如果需要增加上面没有列出的数据类型，您必须使用import语句引入该类型，即使他们作为您的接口被定义在同一个包。

When defining your service interface, be aware that:

当定义您的服务接口的时候，请注意：

- Methods can take zero or more parameters, and return a value or void.
- 方法可以不使用参数也可以使用多个参数，可以返回一个值或者不返回值。
- All non-primitive parameters require a directional tag indicating which way the data goes. Either in, out, orinout (see the example below).
- 所有的非原始参数需要一个方向标签，来指示数据的去向：是in out 或者inout的一种（请看下面的例子）。

Primitives are in by default, and cannot be otherwise.

原语默认是in，并且不能是其他的。

Caution: You should limit the direction to what is truly needed, because marshalling parameters is expensive.

注意：您应该尽量使方向标签指示到真正需要的地方，因为编组参数会耗费很多的系统资源。

- All code comments included in the .aidl file are included in the generated [IBinder](#) interface (except for comments before the import and package statements).
- 所有包含在.aidl文件的代码注释会被包含在生成的IBinder接口中（在引入和声明包之前的注释除外）。
- Only methods are supported; you cannot expose static fields in AIDL. Here is an example .aidl file:
- 只支持方法，您不必期望在AIDL中暴露静态方法，这是一个.aidl文件的例子：

```
// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import statements

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service, to do evil things
with it. */
    int getPid();

    /** Demonstrates some basic types that you can use as parameters
     * and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean aBoolean, float
aFloat,
                    double aDouble, String aString);
}
```

Simply save your .aidl file in your project's src/ directory and when you build your application, the SDK tools generate the [IBinder](#) interface file in your project's gen/ directory. The generated file name matches the .aidl file name, but with a .java extension (for example, IRemoteService.aidl results in IRemoteService.java).

建立您的工程时，您只需要把.aidl文件保存到src/目录里，SDK工具会在您的gen、文件夹下自动生成[IBinder](#)接口文件。生成的文件名和.aidl文件名相符合，但是使用.java的扩展名（例如，如果文件是IRemoteService.aidl则生成的是IRemoteService.java）。

If you use Eclipse, the incremental build generates the binder class almost immediately. If you do not use Eclipse, then the Ant tool generates the binder class next time you build your application—you should build your project with ant debug (or ant release) as soon as you're finished writing the .aidl file, so that your code can link against the generated class.

如果您使用Eclipse，增量构建几乎即时地生成绑定者类。如果你不使用Eclipse，那么Ant工具在你下次构建你的应用程序时生成绑定着类——你应该用ant debug命令（甚至ant release命令）构建你的工程，只要你完成编写.aidl文件时，以使你的代码可以链接到被生成的类。

2. Implement the interface

实现该接口

When you build your application, the Android SDK tools generate a .java interface file named after your .aidlfile. The generated interface includes a subclass named Stub that is an abstract implementation of its parent interface (for example, YourInterface.Stub) and declares all the methods from the .aidl file.

在您创建应用的时候，AndroidSDK工具会生成一个以您.aidl文件命名的.java文件。生成的接口包括一个名为Stub的子类，这个子类是他的父接口的一个抽象实现（例如YourInterface.Stub），并且声明.aidl文件里的所有方法。

Note: Stub also defines a few helper methods, most notably asInterface(), which takes an [IBinder](#)(usually the one passed to a client's [android.os.IBinder](#)) [onServiceConnected\(\)](#) callback method) and returns an instance of the stub interface. See the section [Calling an IPC Method](#) for more details on how to make this cast.

注意：Stub也定义一些辅助方法，尤其是 `asInterface()`，它需要一个[IBinder](#)（通常是传递给客户端的一个回调方法。）并且返回一个stub接口的实例。更多细节参考[Calling an IPC Method](#)这一节。

To implement the interface generated from the .aidl, extend the generated [Binder](#) interface (for example, `YourInterface.Stub`) and implement the methods inherited from the .aidl file.

要实现.AIDL里面定义的接口，就要继承已经生成的[Binder](#)接口（例如 `YourInterface.Stub`）并且继承.aidl文件的方法。

Here is an example implementation of an interface called `IRemoteService` (defined by the `IRemoteService.aidl` example, above) using an anonymous instance:

这里是一个使用继承`IRemoteService`（使用上面例子的`IRemoteService.aidl`定义）接口的例子，这个`IRemoteService`接口使用一个匿名实例调用`IRemoteService`。

```
private final IRemoteService.Stub mBinder = new
IRemoteService.Stub() {
    public int getPid(){
        return Process.myPid();
    }
    public void basicTypes(int anInt, long aLong, boolean aBoolean,
        float aFloat, double aDouble, String aString) {
        // Does nothing
    }
};
```

Now the `mBinder` is an instance of the `Stub` class (a [Binder](#)), which defines the RPC interface for the service. In the next step, this instance is exposed to clients so they can interact with the service.

`mBinder` 是`Stub`类的一个实例(一个[Binder](#))，它定义了服务的RPC接口。下一步，这个实例暴露给客户端，从而使客户端可以和服务进行交互。

There are a few rules you should be aware of when implementing your AIDL interface:

当继承AIDL接口的时候，有许多规则需要注意：

- Incoming calls are not guaranteed to be executed on the main thread, so you need to think about multithreading from the start and properly build your service to be thread-safe.
- 来自外部的调用不保证在主线程中一定执行，所以您需要一开始就考虑多线程，合理的创建您的服务保证线程安全。
- By default, RPC calls are synchronous. If you know that the service takes more than a few milliseconds to complete a request, you should not call it from the activity's main thread, because it might hang the application (Android might display an "Application is Not Responding" dialog)—you should usually call them from a separate thread in the client.
- RPC调用默认是同步的。如果您知道一个服务需要耗费很多时间来完成请求，那请不要从activity的主线程里调用，因为这样会是应用没有响应（Android可能会显示一个“应用程序没有响应”（ANR）对话框）。通常你应该从客户端的另一个线程里调用他们。
- No exceptions that you throw are sent back to the caller.
- 抛出的异常不要返回给调用者。

3. Expose the interface to clients- 为客户端公开接口

Once you've implemented the interface for your [service](#), you need to expose it to clients so they can bind to it. To expose the interface for your service, extend Service and implement [onBind\(\)](#) to return an instance of your class that implements the generated Stub (as discussed in the previous section). Here's an example service that exposes the IRemoteService example interface to clients.

一旦您完成了对接口的实现，你需要向客户端公开该实现，这样客户端就可以使用它了。这要发布一个[Service](#)，请继承[Service](#)并实现[onBinder\(\)](#)方法来返回您的一个被实现的类的实例，该类继承自生成的Stub类（前面已经叙述过）。下面是一个例子，其中Service向客户端公了 IRemoteService接口。

```
public class RemoteService extends Service {
    @Override
    public void onCreate() {
```

```

        super.onCreate();
    }

@Override
public IBinder onBind(Intent intent) {
    // Return the interface
    return mBinder;
}

private final IRemoteService.Stub mBinder = new
IRemoteService.Stub() {
    public int getPid(){
        return Process.myPid();
    }
    public void basicTypes(int anInt, long aLong, boolean
aBoolean,
                           float aFloat, double aDouble, String aString) {
        // Does nothing
    }
};
}

```

Now, when a client (such as an activity) calls [android.content.ServiceConnection, int\)bindService\(\)](#) to connect to this service, the [client'sonServiceConnected()] callback receives the mBinder instance returned by the service's onBind() method.

当一个客户端（比如一个activity调用）[android.content.ServiceConnection, int\) bindService\(\)](#)方法来和这个服务连接时，这个客户端的[android.os.IBinder\) onServiceConnected\(\)](#) 方法接收mBinder实例并且通过服务的[onBind](#)方法返回。

The client must also have access to the interface class, so if the client and service are in separate applications, then the client's application must have a copy of the .aidl file in its src/ directory (which generates the android.os.Binder interface—providing the client access to the AIDL methods).

客户端也必须访问接口类，所以如果客户端和服务分属两个不同的应用，那客户端应用必须复制.aidl文件到他的src/目录（这个目录生成android.os.Binder接口，提供客户端访问的AIDL接口）。

When the client receives the IBinder in the onServiceConnected() callback, it must callYourServiceInterface.Stub.asInterface(service) to cast the returned parameter toYourServiceInterface type. For example:

当客户端在[android.os.IBinder\) onServiceConnected\(\)](#) 回调方法中接收[IBinder](#)时，它必须调用YourServiceInterface.Stub.asInterface(service)来与您的YourServiceInterface 接口类型保持一致。例如：

```
IRemoteService mIRemoteService;
private ServiceConnection mConnection = new ServiceConnection() {
    // Called when the connection with the service is established
    public void onServiceConnected(ComponentName className, IBinder
service) {
        // Following the example above for an AIDL interface,
        // this gets an instance of the IRemoteInterface, which we
can use to call on the service
        mIRemoteService = IRemoteService.Stub.asInterface(service);
    }

    // Called when the connection with the service disconnects
unexpectedly
    public void onServiceDisconnected(ComponentName className) {
        Log.e(TAG, "Service has unexpectedly disconnected");
        mIRemoteService = null;
    }
};
```

For more sample code, see the [RemoteService.java](#) class in [ApiDemos](#). 更多代码，请看[ApiDemos](#)的[RemoteService.java](#)。

Passing Objects over IPC

If you have a class that you would like to send from one process to another through an IPC interface, you can do that. However, you must ensure that the code for your class is available to the other side of the IPC channel and your class must support the [Parcelable](#) interface.

Supporting the [Parcelable](#) interface is important because it allows the Android system to decompose objects into primitives that can be marshalled across processes.

如果你的类中有的需要通过IPC接口从一个进程发送到另一个进程，您可以这么做。然而，你必须确保类代码可以被其他的IPC接收端所使用。您的类必须支持[Parcelable](#) 接口。支持[Parcelable](#) 接口非常重要因为Android系统需要分解对象为可以穿越进程的原语。

To create a class that supports the [Parcelable](#) protocol, you must do the following:

创建一个支持[Parcelable](#) 协议的类你需要这么做：

1. Make your class implement the Parcelable interface. 确保您的类继承了Parcelable 接口

2. Implement [int\) writeToParcel](#), which takes the current state of the object and writes it to a [Parcel](#).

实现public void [int\) writeToParcel](#)(Parcel out), 该方法可以将当前对象的状态写入[parcel](#).

3. Add a static field called CREATOR to your class which is an object implementing the [Parcelable.Creatorinterface](#).

向类中添加一个静态成员,名为CREATOR。该对象实现了[Parcelable.Creator](#)接口.

4. Finally, create an .aidl file that declares your parcelable class (as shown for the Rect.aidl file, below).

最好，创建一个声明您的Parcelable 类的.aidl文件（就像下面的Rect.aidl文件那样）。

If you are using a custom build process, do not add the .aidl file to your build. Similar to a header file in the C language, this .aidl file isn't compiled. 如果您在使用自定义的进程，不要在里面加入.aidl文件。这和C语言的头文件和相似，.aidl文件不会被编译。

AIDL uses these methods and fields in the code it generates to marshall and unmarshall your objects.

AIDL将使用代码中生成的这些方法和成员来伪装或解读对象。

For example, here is a Rect.aidl file to create a Rect class that's [Parcelable](#): 这里有一个Rect.aidl文件创建一个有[Parcelable](#) 类型Rect类的例子

```
package android.graphics;
// Declare Rect so AIDL can find it and knows that it implements
```

```
// the parcelable protocol.
parcelable Rect;
```

And here is an example of how the [Rect](#) class implements the Parcelable protocol.

这个例子说明了[Rect](#)类如何实现了[Parcelable](#)协议.

```
import android.os.Parcel;
import android.os.Parcelable;

public final class Rect implements Parcelable {
    public int left;
    public int top;
    public int right;
    public int bottom;

    public static final Parcelable.Creator<Rect> CREATOR = new
Parcelable.Creator<Rect>() {
        public Rect createFromParcel(Parcel in) {
            return new Rect(in);
        }

        public Rect[] newArray(int size) {
            return new Rect[size];
        }
    };

    public Rect() {
    }

    private Rect(Parcel in) {
        readFromParcel(in);
    }

    public void writeToParcel(Parcel out) {
        out.writeInt(left);
        out.writeInt(top);
        out.writeInt(right);
        out.writeInt(bottom);
    }

    public void readFromParcel(Parcel in) {
        left = in.readInt();
        top = in.readInt();
        right = in.readInt();
        bottom = in.readInt();
    }
}
```

The marshalling in the Rect class is pretty simple. Take a look at the other methods on [Parcel](#) to see the other kinds of values you can write to a Parcel.

Rect类中的伪装是相当简单的。仔细看看[Parcel](#)中的其他方法，你会看到其他各种值你都可以写进Parcel.

Warning: Don't forget the security implications of receiving data from other processes. In this case, the Rect reads four numbers from the Parcel, but it is up to you to ensure that these are within the acceptable range of values for whatever the caller is trying to do. See Security and Permissions for more information about how to keep your application secure from malware.

警告:不要忘了考虑从其他进程接收数据时的安全性。在本例中，Rect将从Parcel中读四个数字，而你的工作则是确保这些都在可接受的值得范围内而不管调用者想要干什么。AndRoid中的安全和访问许可中有更多关于如何确保应用程序安全的信息。 Calling an IPC Method

调用一个**IPC**方法

Here are the steps a calling class must take to call a remote interface defined with AIDL:

调用类调用AIDL接口的步骤:

1. Include the .aidl file in the project src/ directory.

引入项目src/目录的.aidl文件

2. Declare an instance of the [IBinder](#) interface (generated based on the AIDL).

声明一个[IBinder](#)的实例（基于AIDL生成）。

3. Implement [ServiceConnection](#).

实现[ServiceConnection](#)。

4. Call [android.content.ServiceConnection, int\) Context.bindService\(\)](#) , passing in your [ServiceConnection](#) implementation.

调用[android.content.ServiceConnection, int\) Context.bindService\(\)](#),并在[ServiceConnection](#)实现中进行传递.

5. In your implementation of [android.os.IBinder\) onServiceConnected\(\)](#) , you will receive an [IBinder](#) instance (called service). Call YourInterfaceName.Stub.asInterface((IBinder)service) to cast the returned parameter toYourInterface type.

在您实现[android.os.IBinder\) onServiceConnected\(\)](#) 时，您将会收到一个[IBinder](#)实例(被调用的Service)，调用YourInterfaceName.Stub.asInterface((IBinder)service) 并将参数转换为YourInterface类型。

6. Call the methods that you defined on your interface. You should always trap [DeadObjectException](#) exceptions, which are thrown when the connection has broken; this will be the only exception thrown by remote methods.

调用接口中定义的方法。,你应该捕捉[DeadObjectException](#)异常，该异常在连接断开时被抛出。它只会被远程方法抛出。

7. To disconnect, call [Context.unbindService\(\)](#) with the instance of your interface.

断开连接，调用接口实例中的 [Context.unbindService\(\)](#) 方法。

A few comments on calling an IPC service: 调用IPC服务时候的一些经验：

- Objects are reference counted across processes.-跨进程时，对象会被统计。
- You can send anonymous objects as method arguments.-匿名对象可以通过方法参数发送。

For more information about binding to a service, read the [Bound Services](#) document.

关于绑定服务的更多信息请参阅 [Bound Services](#) 的文档。

Here is some sample code demonstrating calling an AIDL-created service, taken from the Remote Service sample in the ApiDemos project.

这是一些展示调用基于AIDL创建服务的简单例子，来自ApiDemo项目的远程服务模块。

```

public static class Binding extends Activity {
    /** The primary interface we will be calling on the service. */
    IRemoteService mService = null;
    /** Another interface we use on the service. */
    ISecondary mSecondaryService = null;

    Button mKillButton;
    TextView mCallbackText;

    private boolean mIsBound;

    /**
     * Standard initialization of this activity. Set up the UI,
     then wait
     * for the user to poke it before doing anything.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.remote_service_binding);

        // Watch for button clicks.
        Button button = (Button) findViewById(R.id.bind);
        button.setOnClickListener(mBindListener);
        button = (Button) findViewById(R.id.unbind);
        button.setOnClickListener(mUnbindListener);
        mKillButton = (Button) findViewById(R.id.kill);
        mKillButton.setOnClickListener(mKillListener);
        mKillButton.setEnabled(false);

        mCallbackText = (TextView) findViewById(R.id.callback);
        mCallbackText.setText("Not attached.");
    }

    /**
     * Class for interacting with the main interface of the service.
     */
    private ServiceConnection mConnection = new ServiceConnection()
    {
        public void onServiceConnected(ComponentName className,
                                      IBinder service)
        {
            // This is called when the connection with the service
            has been
            // established, giving us the service object we can use
            to
            // interact with the service. We are communicating with
        }
    };
}

```

```

our           // service through an IDL interface, so get a client-
side          // representation of that from the raw service object.
mService = IRemoteService.Stub.asInterface(service);
mKillButton.setEnabled(true);
mCallbackText.setText("Attached.");

// We want to monitor the service for as long as we are
// connected to it.
try {
    mService.registerCallback(mCallback);
} catch (RemoteException e) {
    // In this case the service has crashed before we
    // could even
    // do anything with it; we can count on soon being
    // disconnected (and then reconnected if it can be
    // restarted)
    // so there is no need to do anything here.
}

// As part of the sample, tell the user what happened.
Toast.makeText(Binding.this,
R.string.remote_service_connected,
        Toast.LENGTH_SHORT).show();
}

public void onServiceDisconnected(ComponentName className) {
    // This is called when the connection with the service
has been
    // unexpectedly disconnected -- that is, its process
crashed.
    mService = null;
    mKillButton.setEnabled(false);
    mCallbackText.setText("Disconnected.");

    // As part of the sample, tell the user what happened.
    Toast.makeText(Binding.this,
R.string.remote_service_disconnected,
        Toast.LENGTH_SHORT).show();
};

/**
 * Class for interacting with the secondary interface of the
service.
*/
private ServiceConnection mSecondaryConnection = new
ServiceConnection() {
    public void onServiceConnected(ComponentName className,
        IBinder service) {
        // Connecting to a secondary interface is the same as
any
        // other interface.
        mSecondaryService =
ISecondary.Stub.asInterface(service);
        mKillButton.setEnabled(true);
    }

    public void onServiceDisconnected(ComponentName className) {
        mSecondaryService = null;
    }
}

```

```

        }
    }

    private OnClickListener mBindListener = new OnClickListener() {
        public void onClick(View v) {
            // Establish a couple connections with the service,
            binding
            // by interface names. This allows other applications
            to be
            // installed that replace the remote service by
            implementing
            // the same interface.
            bindService(new Intent(IRemoteService.class.getName()),
                mConnection, Context.BIND_AUTO_CREATE);
            bindService(new Intent(ISecondary.class.getName()),
                mSecondaryConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
            mCallbackText.setText("Binding.");
        }
    };

    private OnClickListener mUnbindListener = new OnClickListener() {
        public void onClick(View v) {
            if (mIsBound) {
                // If we have received the service, and hence
                registered with
                // it, then now is the time to unregister.
                if (mService != null) {
                    try {
                        mService.unregisterCallback(mCallback);
                    } catch (RemoteException e) {
                        // There is nothing special we need to do if
                        the service
                        // has crashed.
                    }
                }
                // Detach our existing connection.
                unbindService(mConnection);
                unbindService(mSecondaryConnection);
                mKillButton.setEnabled(false);
                mIsBound = false;
                mCallbackText.setText("Unbinding.");
            }
        }
    };

    private OnClickListener mKillListener = new OnClickListener() {
        public void onClick(View v) {
            // To kill the process hosting our service, we need to
            know its
            // PID. Conveniently our service has a call that will
            return
            // tell us that information.
            if (mSecondaryService != null) {
                try {
                    int pid = mSecondaryService.getPid();
                    // Note that, though this API allows us to
request to

```

will
IDs you
means only
additional
packages
kill each

```

        // kill any process based on its PID, the kernel
        // still impose standard restrictions on which
        // are actually able to kill. Typically this
        // the process running your application and any
        // processes created by that app as shown here;
        // sharing a common UID will also be able to
        // other's processes.
Process.killProcess(pid);
mCallbackText.setText("Killed service
process.");
```

the

```

} catch (RemoteException ex) {
    // Recover gracefully from the process hosting
    // server dying.
    // Just for purposes of the sample, put up a
    Toast.makeText(Binding.this,
                    R.string.remote_call_failed,
                    Toast.LENGTH_SHORT).show();
}
```

}

```

// -----
// Code showing how to deal with callbacks.
// -----
```

```

/**
 * This implementation is used to receive callbacks from the
remote
 * service.
 */
private IRemoteServiceCallback mCallback = new
IRemoteServiceCallback.Stub() {
    /**
     * This is called by the remote service regularly to tell us
about
     * new values. Note that IPC calls are dispatched through a
thread
     * pool running in each process, so the code executing here
     * NOT be running in our main thread like most other things
     * to update the UI, we need to use a Handler to hop over
there.
     */
    public void valueChanged(int value) {
        mHandler.sendMessage(mHandler.obtainMessage(BUMP_MSG,
value, 0));
    }
};
```

```
private static final int BUMP_MSG = 1;

private Handler mHandler = new Handler() {
    @Override public void handleMessage(Message msg) {
        switch (msg.what) {
            case BUMP_MSG:
                mCallbackText.setText("Received from service: "
+ msg.arg1);
                break;
            default:
                super.handleMessage(msg);
        }
    }
}
```

来自“[index.php?](#)

[title=Android_Interface_Definition_Language_\(AIDL\)&oldid=8725](#)



Content Providers

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/providers/content-providers.html>

翻译：gale56

更新：2012.06.05

目录

[[隐藏](#)]

[1 Content Providers](#)

Content Providers

Content providers是用来管理对结构化数据集进行访问的一组接口。这组接口对数据进行封装，并提供了用于定义数据安全的机制。Content providers是一个进程使用另一个进程数据的标准接口。

当要使用content provider访问数据时，我们需要在应用程序的Context中使用ContentResolver对象作为客户端，同provider进行通信。与provider对象通信的ContentResolver对象是ContentProvider类的一个实例。provider对象接收从客户端发来的数据，执行请求的动作并返回结果。

如果你不打算同其他应用程序共享数据，就没必要实现provider。但是，如果希望在自己的应用程序中搜索建议的功能，就需要实现自己的provider。同样的，如果希望在自己的应用程序和其他的应用程序间拷贝粘贴复杂的数据或文件，也需要实现自己的provider。

Android系统本身也通过content providers来管理数据，如音频，视频，图像，个人联系信息等。我们可以在android.provider包的参考文档中看到这些providers列表。在一定条件下，这些providers能够访问任何Android应

用程序。

接下来，将就content providers的以下题目做详细说明：

Content Provider Basics

如何访问内容提供商中的数据表中的数据的组织时

Creating a Content Provider

：如何创建自己的内容提供者

Calendar Provider

：如何访问日历提供的Android平台的一部分

Contacts Provider

：如何访问联系人提供的Android平台的一部分。

来自“[index.php?title=Content_Providers&oldid=10917](#)”



Content Provider Basics

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Gale56 & playboyanta123

原文链接：<http://developer.android.com/intl/zh-CN/guide/topics/providers/content-provider-basics.html>

目录

[[隐藏](#)]

[1 Content Provider 基本介绍 - Content Provider Basics](#)

- [1.1 概述-Overview](#)
 - [1.1.1 访问 provider - Accessing a provider](#)
 - [1.1.2 内容 URI - Content URIs](#)
- [1.2 从Provider中检索数据 - Retrieving Data from the Provider](#)
 - [1.2.1 获得读访问权限-Requesting read access permission](#)
 - [1.2.2 构建查询程序 - Constructing the query](#)
 - [1.2.2.1 防止恶意输入 - Protecting against malicious input](#)
 - [1.2.3 显示查询结果 - Displaying query results](#)
 - [1.2.4 从查询结果中获得数据 - Getting data from query results](#)
- [1.3 Content Provider 权限-Content Provider Permissions](#)
- [1.4 插入，更新和删除数据 - Inserting, Updating, and Deleting Data](#)
 - [1.4.1 插入数据](#)
 - [1.4.2 更改数据](#)
 - [1.4.3 删除数据](#)
- [1.5 Provider的数据类型](#)
- [1.6 Provider的其它访问形式](#)
 - [1.6.1 批处理访问](#)
 - [1.6.2 通过intent访问数据](#)
 - [1.6.2.1 获取临时的访问权限](#)

1.6.3 使用其它的应用程序

- [1.7 契约类](#)
- [1.8 引用MIME类型](#)

Content Provider 基本介绍 - Content Provider Basics

Content provider管理对于中央数据库的访问。而provider是Android应用程序的一部分，通常每个provider自己提供界面同数据交互。然而，content providers最主要的目的的是为了让其他应用程序使用provider的客户端对象访问provider。所以，providers和provider客户端共同提供了一个一致的标准数据接口用来处理进程间通信和安全的数据访问。

本节主要介绍以下几个方面的内容：

- Content providers是如何工作的。
- 如何使用API从content provider中接收数据。
- 在content provider中如何使用API进行数据的插入，更新或删除。
- Providers提供的其他API功能，以便更好的使用providers。

概述-Overview

Content provider以类似于关系数据库中一个或多个表的方式给外部的应用程序提供数据。一行代表provider收集的一些类型数据的一个实例，一列中的每一行代表数据中某个类型的一个实例。

例如，用户字典就是Android平台内置的providers中的一个，用来保存用户希望保存的非标准词的拼写。表1展示了provider表中数据可能的存储方式：

表1：用户字典表的例子

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3

const	user1	255	pt_BR	4
int	user5	100	en_UK	5

在表1中，每行代表了在标准字典中可能找不到的一个词的实例。每一列表明这个词的一些数据，例如 它第一次出现时的语言环境。列标题是存储在provider中的列名。为了找到一行的语言环境，你就需要索引到locale这个列所指向的内容。对于此 provider来说，_ID列作为主键提供自动索引。

注：主键对于provider来说不是必须的，即使有主键，provider也不必一定要使用_ID作为主键的列名。但是，如果希望将provider的数据绑定到ListView上，就要使用_ID作为一列的名字。这些会在显示查询结果的小结中详细介绍。

访问 provider - Accessing a provider

应用程序使用[ContentResolver](#)客户端对象来访问content provider的数据。[ContentResolver](#)对象与[ContentProvider](#)的一个具体子类的实例拥有相同名字的接口。[ContentResolver](#)的方法提供了基本的'CRUD'（创建，检索，更新和删除）数据存储的功能。

客户端应用程序进程中的[ContentResolver](#)对象和我们自己的应用程序中的[ContentProvider](#)对象会自动处理进程间通信。[ContentProvider](#)也会作为其存储的数据和数据以表的形式展现之间的抽象层。

注：为访问provider，应用程序通常需要在manifest文件中添加特定的权限。这些将在[Content Provider 权限](#)小结中详细介绍。

举例来说，为了从用户字典的Provider中获得单词和它们出现的语言环境列表，就需要调用 ContentResolver.query()方法。query()方法会调用在用户字典的Provider中定义的 ContentProvider.query()方法。下面这行代码展示了ContentResolver.query()是如何调用的：

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,           // 词表的内容URI
    mProjection,                            // 每行中返回数据的列的名称,
    mSelectionClause,                      // null表示返回所有列的数据。
    mSelectionArgs,                         // 过滤条件
    mSortOrder);                          // 过滤条件的参数
                                            // 返回行的排序方式
```

表2介绍了函数query(Uri,projection,selection,selectionArgs,sortOrder)的参数同SQL SELECT语句之间的关系:

表2：Query()函数与SQL的查询语句间的对应关系

query() argument	SELECT keyword/parameter	Notes
Uri	FROM table_name	provider中的Uri相当于表名。
projection	col,col,col,...	projection 是一个列名的数组，规定了查询结果中每行应该包含哪些列。
selection	WHERE col = value	selection 指定了符合条件的行。
selectionArgs	(No exact equivalent. Selection arguments replace ? placeholders in the selection clause.)	
sortOrder	ORDER BY col,col,...	sortOrder 指定了在返回结果Cursor中行排序的规则。

内容 URI - Content URIs

内容URI是provider中用来唯一标识数据的URI。内容URIs包括整个provider的符号名称（它的权威）和指向表的名字（路径）。当调用客户端的方法来访问provider中的一个表时，这个表的内容URI会作为这个方法一个参数。

在前面的代码中，常量CONTENT_URI包含用户词典中的词表的内容URI。[ContentResolver](#)对象解析出URI的权威，并将其与系统已知provider的权威表比较，从而获取对应的provider。然后，[ContentResolver](#)就可以将查询参数发送给正确的provider。

[ContentProvider](#)根据内容URI中的路径部分选择需要访问的表。Provider通

常为每个表都公开一个路径。

在前面的程序中，“词”表的完整URI如下所示：

```
content://user_dictionary/words
```

这里的user_dictionary是provider的权威，words是表的路径。content://（大纲）是一定要有的，作为内容URI的唯一标识。

许多provider允许通过在URI结尾追加一个ID值来访问表中的单独一行。例如，从用户词典中接收_ID为4的那行数据，就要使用如下的内容URI：

```
Uri singleUri =  
ContentUri.withAppendedId(UserDictionary.Words.CONTENT_URI, 4);
```

当我们希望查询多行数据，然后更新或删除其中一行的数据时，就会经常用到id值。

注：Uri和Uri.Builder类包含了从字符串构建Uri对象的一些方法。ContentUris包含了将id值追加到URI结尾的方法。前面的程序段使用withAppendedId()方法将id追加到用户词典的内容URI后面。

从Provider中检索数据 - Retrieving Data from the Provider

本节介绍了如何从provider中检索数据，使用用户词典的Provider来举例说明。

为了方便起见，本节中的代码会在“UI线程”中调用ContentResolver.query()函数。但是，在实际代码中，应该将查询操作放到一个单独的线程中异步执行。一种方法就是使用CursorLoader类，这个类的详细说明可以在Loaders手册中查到。此外，例子代码仅仅是一部分，它们并没有显示完整的应用程序。

为了从provider中检索数据，请遵循以下的基本步骤：

1. 获得provider的读访问权限。
2. 定义发送给provider的查询代码。

获得读访问权限-Requesting read access permission

为了从provider中查询数据，应用程序必须有此provider的读访问权限。该权限只能在应用程序的manifest文件中指定，而不能在运行时获得，使

用`<uses-permission>`标签，并指明该provider定义的此权限的确切名称。当在应用程序的manifest文件中指定了该标签，实际上是要求为您的应用程序赋予此权限。当用户安装你的应用时，会隐式的赋予此权限。

要知道你使用的provider的读访问权限和其他权限的确切名字，请参考provider的文档。

在[Content Provider Permissions](#)中详细说明了访问provider的各个权限的作用。

用户词表的Provider在它的manifest文件中定义

了`android.permission.READ_USER_DICTIONARY`权限，所以如果一个应用程序希望从provider中读取信息，就必须获得此权限。

构建查询程序 - Constructing the query

从provider中检索数据的下一步就是构建查询程序。程序的第一部分定义了访问用户词典provider所需要的一些变量：

```
// projection 定义了返回的结果中需包含的列。
String[] mProjection =
{
    UserDictionary.Words._ID,           // Contract class constant for the
_ID column name
    UserDictionary.Words.WORD,         // Contract class constant for the
word column name
    UserDictionary.Words.LOCALE       // Contract class constant for the
locale column name
};

// 定义一个字符串用来包含selection的条件。
String mSelectionClause = null;

// 初始化数组包含selection的参数。
String[] mSelectionArgs = {" "};
```

接下来的程序以用户词典的Provider为例，显示如何使用[ContentResolver.query()]。Provider客户端查询类似于SQL查询，它包含一组返回的列值，一组查询条件和排序规则。

查询返回的列的集合被称作一个**projection**（即变量mProjection）。

检索指定行的条件表达式被分成selection语句和selection参数。selection语句是由逻辑和布尔表达式，列名和值（变量mSelection）组成的。如果指定的是替换参数? 而不是值，查询方法会从selection参数数组（变量mSelectionArgs）中检索对应的值。

下一段程序中，如果用户没有输入任何单词，selection语句被设置成null，查询会返回provider中所有的单词。如果用户输入一个单词，查询语句被设置

成 `UserDictionary.Words.Word + " = ?"`, 同时 `selection` 参数数组中的第一个元素被设置成用户输入的单词。

```

/*
 * 这里定义了只有一个元素的字符串数组用来包含 selection 的参数。
 */
String[] mSelectionArgs = { "" };

// 从UI获得单词
mSearchString = mSearchWord.getText().toString();

// 这里记着要添加错误检查。

// 如果输入的单词为空, 返回所有词。
if (TextUtils.isEmpty(mSearchString)) {
    // 设置selection语句, 如果为空返回所有单词
    mSelectionClause = null;
    mSelectionArgs[0] = "";

} else {
    // 构建selection语句来匹配用户输入的单词
    mSelectionClause = " = ?";

    // 将用户输入的单词放到selection的参数列表中
    mSelectionArgs[0] = mSearchString;
}

// 查询表, 并返回一个Cursor对象
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,           // 单词表的URI
    mProjection,                                // 返回结果中每行包含的列
    mSelectionClause,                           // 用户输入的单词或null
    mSelectionArgs,                             // 用户输入的字符串或null
    mSortOrder);                               // 返回行的排序规则

// 一些provider在查询发生错误后会返回null, 其他的会抛出异常
if (null == mCursor) {
    /*
     * 这里需要添加你的错误处理程序, 可以调用
     * android.util.Log.e() 函数输出日志
     *
     */
// 如果Cursor为空, 表明provider中没有匹配条件的结果
} else if (mCursor.getCount() < 1) {

    /*
     * 添加代码通知用户查询未成功。这些不一定是错误。可以供用户选择是插入
     * 新的一行还是重新输入检索词。
     */
}

} else {
    // 插入对查询结果的操作
}

```

此查询类似于下面的SQL语句：

```
SELECT _ID, word, frequency, locale FROM words WHERE word = <userinput> ORDER BY word ASC;
```

在这条SQL语句中用实际的列名替换了contract类的常量。

防止恶意输入 - Protecting against malicious input

如果content provider管理的数据保存在SQL数据库中，在数据库中包含了外部不信任的数据，可能会引起SQL注入的风险。

考虑下面的selection语句：

```
// 构建一个selection语句直接将用户输入连接到列名后面
String mSelectionClause = "var = " + mUserInput;
```

如果我们这样写程序，将允许用户把恶意的SQL语句连接到你的SQL语句中。例如，用户可以在变量mUserInput中输入“nothing; DROP TABLE *;”，会导致selection语句变为var = nothing; DROP TABLE *;。因为selection语句会作为SQL语句，所以这条语句可能会导致provider删除底层SQLite数据库中的所有表（除非provider建立了捕获SQL注入的机制）。

为了避免SQL注入，在selection语句中使用?作为替代参数，并使用一个单独的selection参数数组。当使用参数的时候，用户的输入被直接添加到查询中，而不是被解释为SQL语句的一部分。因为它不是作为SQL处理，用户输入不能注入恶意的SQL。使用selection语句参数而不是直接将用户的输入连接起来的方式如下：

```
// 构建一个用参数替代的selection语句
String mSelectionClause = "var = ?";
```

用如下方式建立selection的参数数组：

```
// 定义一个包含selection的数组
String[] selectionArgs = {" "};
```

以如下方式将一个值放入selection的参数数组中：

```
// 将用户的输入添加到selection的参数数组中
selectionArgs[0] = mUserInput;
```

一个使用`?`作为替代参数并使用了`selection`参数数组的`selection`语句是使用`selection`的首要方式，即使provider并不是基于SQL数据库。

显示查询结果 - Displaying query results

客户端的`ContentResolver.query()`方法会返回一个`Cursor`对象，包含了满足查询条件的行及由查询的`projection`指定的列。一个`Cursor`对象为它所包含的行和列提供了随机读取的访问。使用`Cursor`的方法，可以遍历查询结果中的所有行，确定每列的数据类型，获取一列的数据，及检验结果的其他属性。当provider的数据改变时，有些`Cursor`实现了自动更新的功能，或者当`Cursor`改变时，会触发一个监听对象的方法，或者两者兼而有之。

注：provider可能会基于查询对象本身而限制对一些列的访问。例如，联系人的Provider会限制对同步适配器的一些列的访问，所以这些列将不会返回给activity或service。

如果没有行能匹配`selection`条件，provider会返回一个`Cursor.getCount()`数量为0的`Cursor`对象（一个空的cursor）。

如果发生内部错误，查询的结果取决于具体的provider。有的会返回`null`，有的会抛出异常

因为`Cursor`是所有行的列表，所以显示`Cursor`内容的一个好的方式是使用`SimpleCursorAdapter`将其连接到`ListView`上。

下面的代码延续了前面的程序。创建了一个`SimpleCursorAdapter`对象，其中包含了查询结果的`Cursor`，并将此对象设置为一个`ListView`的适配器：

```
// Defines a list of columns to retrieve from the Cursor and load
into an output row
String[] mWordListColumns =
{
    UserDictionary.Words.WORD,      // Contract class constant containing
    the word column name
    UserDictionary.Words.LOCALE   // Contract class constant containing
    the locale column name
};

// Defines a list of View IDs that will receive the Cursor columns
for each row
int[] mWordListItems = { R.id.dictWord, R.id.locale };

// Creates a new SimpleCursorAdapter
mCursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(),           // The application's
    Context object
    R.layout.wordlistrow,            // A layout in XML for one
    row in the ListView
```

```

mCursor,                                // The result from the
query                                     // A string array of column
mWordListColumns,                         names in the cursor
mWordListItems,                           // An integer array of view
IDs in the row layout
0);                                         // Flags (usually none are
needed)

// Sets the adapter for the ListView
mWordList.setAdapter(mCursorAdapter);

```

注：若要用[Cursor](#)来备份[ListView](#)，就需要此cursor包含一个名为[_ID](#)的列。正因为这样，前面对单词表的查询会收到[_ID](#)这一列，即使[ListView](#)并没有显示该列。此限制条件也说明了为什么大多数providers会在它们的每个表中包含一个[_ID](#)列。

从查询结果中获得数据 - Getting data from query results

我们还可以将查询结果用于别的任务，而不是仅仅将其显示出来。例如，你可以从用户词典中检索出拼写，然后在其他的provider中查找它们。为了实现这点，你需要在[Cursor](#)中遍历所有行：

```

// Determine the column index of the column named "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
 * Only executes if the cursor is valid. The User Dictionary Provider
returns null if
 * an internal error occurs. Other providers may throw an Exception
instead of returning null.
 */

if (mCursor != null) {
    /*
     * Moves to the next row in the cursor. Before the first movement
in the cursor, the
     * "row pointer" is -1, and if you try to retrieve data at that
position you will get an
     * exception.
    */
    while (mCursor.moveToNext()) {
        // Gets the value from the column.
        newWord = mCursor.getString(index);

        // Insert code here to process the retrieved word.

        ...
    } // end of while loop
} else {

```

```
// Insert code here to report an error if the cursor is null or
the provider threw an exception.
}
```

[Cursor](#)实现了包含各种“get”的方法用来从对象中获得不同类型的数据。例如，上面的程序中使用[Cursor.getString\(\)](#)方法。也可以使用[getType\(\)](#)方法获得一个值，显示该列的数据类型。

Content Provider 权限-Content Provider Permissions

提供provider的应用程序可以指定访问权限，以便其他应用能够访问provider提供的数据。此权限可以保证用户能够知道应用程序中的那些数据可以访问。基于provider提供的说明，其他的应用程序可以根据自身的需求来请求权限去访问 provider。最后，用户在安装此应用时会看到该应用请求获得的权限。

如果包含provider的应用没有指定任何权限，其他的应用程序是无法访问该provider的数据的。但是包含provider应用的其他组件拥有对该provider的完全读写权限，无论是否指定了权限。

正如上面所说，用户字典的Provider要[android.permission.READ_USER_DICTIONARY](#)权限来控制对其数据的访问。此provider还提供了[android.permission.WRITE_USER_DICTIONARY](#)权限控制对数据的插入，更新和删除。

为获得访问provider的权限，应用程序在manifest文件中需要使用[`<uses-permission>`](#)标签。当Android Package Manager 安装应用时，用户必须批准应用程序的所有权限请求。如果用户允许了，Package Manager会继续安装流程；如果用户不允许，Package Manager会终止安装。

下面的[`<uses-permission>`](#)标签请求获取用户词典Provider的读权限：

```
<uses-permission
    android:name="android.permission.READ_USER_DICTIONARY">
```

在[安全性和权限](#)中会详细介绍provider访问权限的影响。

插入，更新和删除数据 - Inserting, Updating, and Deleting Data

同从provider获取数据的方式一样，我们也要使用在provider的客户端和[ContentProvider](#)之间的交互来修改数据。调用[ContentResolver](#)的方法并传入参数，最终会调用[ContentProvider](#)的对应方法。provider和provider客户端

会自动处理安全性和进程间通信。

插入数据

你可以调用[ContentResolver.insert\(\)](#)方法往provider中插入数据。该方法会往provider中插入一行新的数据并返回该数据的URI。下面的例子演示了如何往User Dictionary Provider中插入一个新单词：

```
// Defines a new Uri object that receives the result of the insertion
Uri mNewUri;

...
// Defines an object to contain the new values to insert
ContentValues mNewValues = new ContentValues();

/*
 * Sets the values of each column and inserts the word. The arguments
 * to the "put"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
UserDictionary.Word.CONTENT_URI,      // the user dictionary content URI
mNewValues                          // the values to insert
);
```

这行新的数据会成为一个单一的[ContentValues](#)对象，它跟每一行的cursor返回的数据是一样的。这个对象的每个字段不需要有相同的数据类型，如果你不想给某个字段设定一个值，你可以调用[ContentValues.putNull\(\)](#)把它设为null。这条数据中并没有插入_ID字段，那是因为它会自动的被增加到数据中。provider会给每一行数据赋予一个唯一的_ID，而它往往就被看作数据库表中的主键。newUri会返回content URI以区分这条新增加的数据，跟下面的格式一样：

```
content://user_dictionary/words/<id_value>
```

<id_value> 是这条新数据的_ID的值。大多数的provider都会自动的检测出这种形式的content URI，然后在特定的行上执行请求的操作。调

用[ContentUris.parseLong\(\)](#)可以从返回的[Uri](#)中获取_ID的值。

更改数据

更改数据时，你要用更新后的[ContentValues](#)对象和选择条件，就像你做插入操作和查询操作那样。在客户端你要使用[ContentResolver.update\(\)](#)方法。你仅仅只需要把你想要更改的字段的添加到[ContentValues](#)对象中就可以了。如果你想删除某个字段的内容，可以把它设为null。下面的代码把所有行的环境语言字段由“en”改为null。返回的值是被更改的行数：

```
// Defines an object to contain the updated values
ContentValues mUpdateValues = new ContentValues();

// Defines selection criteria for the rows you want to update
String mSelectionClause = UserDictionary.Words.LOCALE + " LIKE ?";
String[] mSelectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int mRowsUpdated = 0;

...

/*
 * Sets the updated value and updates the selected words.
 */
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
UserDictionary.Words.CONTENT_URI, // the user dictionary contentURI
mUpdateValues // the columns to update
mSelectionClause // the column to select on
mSelectionArgs // the value to compare to
);
```

当你调用[ContentResolver.update\(\)](#)方法的时候你应该对用户的输入进行检查。要想了解更多关于这方面的信息，请阅读[Protecting against malicious input](#)部分。

删除数据

删除数据与检索数据很类似：你用特定的选择条件选出你要删除的数据，然后客户端的方法会返回删除的行数。下面的代码删除了appid字段的值为“user”的所有数据，并返回了删除的行数。

```
// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};
```

```
// Defines a variable to contain the number of rows deleted
int mRowsDeleted = 0;

...
// Deletes the words that match the selection criteria
mRowsDeleted = getContentResolver().delete(
UserDictionary.Words.CONTENT_URI,           // the user dictionary content URI
to select on      mSelectionArgs          // the value to
compare to        );

```

当你调用`IContentResolver.delete()`方法的时候你应该对用户的输入进行检查。要想了解更多关于这方面的信息，请阅读[Protecting against malicious input](#)部分。

Provider的数据类型

Content providers可以提供许多种不同的数据类型。User Dictionary Provider只能提供文本类型的数据，而provider还可以提供下面的数据格式：integer long integer (long) floating point long floating point (double) 除此之外，Provider经常使用的数据类型是Binary Large OBject (BLOB)，它是一个长度为64KB的byte类型的数组。你可以查阅[Cursor](#)类的"get"方法了解所有可用的数据类型。

Provider的说明文档中列出了每个字段的数据类型。User Dictionary Provider的数据类型也在其相关类的参考文献 [1] ([Contract Classes](#)描述了所有的合同类)。你也可以调用[Cursor.getType\(\)](#)方法来鉴定其数据类型。

Provider也支持content URI自定义的MIME数据类型信息。你可以使用MIME类型的信息来检测你的应用程序是否能处理provider所提供的数据，或者在基于MIME类型的基础上选择一种处理方式。当你使用的provider中包含复杂的数据结构或者文件时，你往往需要MIME类型的数据。例如，Contacts Provider中的[ContactsContract.Data](#)表就使用了MIME类型来标注存储在每一行中的contact数据的类型。调用[ContentResolver.getType\(\)](#)可以得到MIME类型对应的content URI。[MIME Type Reference](#)章节介绍了标准的和自定义的MIME的类型的组成结构。

Provider的其它访问形式

Provider的其它访问形式在应用程序开发过程中也是很重要的：[Batch](#)

[access](#): 你可以调用 [ContentProviderOperation](#)类中的方法创建批处理访问，然后调用 [ContentResolver.applyBatch\(\)](#) 来应用它们。 异步查询：你应该在一个单独的线程中执行查询操作。使用 [CursorLoader](#) 对象是这种操作的一种方式。[Loaders](#)的引导例子演示了怎么操作这个对象。通过[intent](#)访问数据：尽管你不能直接发送一个[intent](#)给一个provider，但是你可以发送一个[intent](#)给provider的应用程序，这样做往往是修改provider中数据最好的方式。下面的内容介绍了批处理访问数据和通过[intent](#)修改数据。

批处理访问

当你要往provider中插入大量的数据，或者在相同的方法调用中往不同的表插入数据，或者执行进程间的数据访问的一组操作（原子操作）的情况下，批处理访问是非常有用的。要在"batch mode"中访问provider，你要创建一个[ContentProviderOperation](#)对象的数组，然后通过[ContentResolver.applyBatch\(\)](#)来把他们分派到content provider中。你应该把content provider的authority传到这个方法中，而不是一个特定的content URI，它允许这个数组中的每一个[ContentProviderOperation](#)对象能对不同的表进行操作。调用[ContentResolver.applyBatch\(\)](#)方法会以数组的形式返回结果。[ContactsContract.RawContacts](#)类的介绍中包含的代码演示了批处理插入数据操作。在[Contact Manager](#) 小应用程序的ContactAdder.java 源文件中包含了批处理操作的例子。

通过[intent](#)访问数据

[Intent](#)提供了间接访问content provider 的方法。在你的应用程序没有访问provider的权限的时候，你也可以通过以下方式让你的用户访问provider中的数据，要么用[intent](#)从有访问provider的应用程序中得到返回的结果，要么激活应用程序的访问权限以便让用户能在其中进行操作。

获取临时的访问权限

尽管你没有合适的访问权限，你也可以访问content provider 中的数据，发送一个[intent](#)到有访问权限的应用程序中，然后接收这个包含"URI"权限的[intent](#)。这个特定的content URI 的权限将会持续到接收它的activity停止为止。具有持久访问权限的应用程序会给作为结果返回的[intent](#)中的临时权限设置一个标志：

- Read permission:[FLAG_GRANT_READ_URI_PERMISSION](#)

Write permission:[FLAG_GRANT_WRITE_URI_PERMISSION](#)

提示：如果content URI中包含了authority，那么这些标志不会对content provider 给予正常的读和写访问权限。只有URI本身才有这个访问权限。

使用辅助应用程序来显示数据 如果你的应用程序有访问权限，你仍然想通过intent把数据传给别的应用程序。例如，Calendar应用程序会接收[ACTION_VIEW](#) intent，它包含了一个特定的日期或者事件。这样就不需要你创建自己的UI而可以显示Calendar的信息。想了解关于这个特性更多信息，可以去查看关于[Calendar Provider](#)的指导。接收你发送的Intent的应用程序并不一定要是与provider有关联的应用程序。例如，你可以从Contact Provider中检索一个联系人，然后发送一个包含这个content URI的[ACTION_VIEW](#) intent到图片浏览器中显示这个联系人的图片。

Provider会在manifest文件中定义content URI的URI权限，用[`<provider>`](#)元素的[`android:grantUriPermission`](#) 属性，就像[`<provider>`](#)元素的子元素[`<grant-uri-permission>`](#)一样。URI的权限机制在[Security and Permissions](#)中的"URI Permissions"部分有更加详细的说明。

例如，尽管你没有[READ_CONTACTS](#) 权限，你仍然可以从Contacts Provider中检索某个联系人的信息。你可能会想用应用程序给联系人中的某个他/她发送电子生日贺卡。与其请求[READ_CONTACTS](#) 权限，你更应该让你的用户通过你的应用程序来获取访问所有联系人的所有信息的权限。按照下面的步骤，你可以达到这个目的：

1. 你的应用程序可以调用[startActivityForResult\(\)](#)方法发送一个包含[ACTION_PICK](#) 活动和 "contacts" MIME类型的[CONTENT_ITEM_TYPE](#)。
2. 因为这个intent与intent filter中的People 应用程序的 "selection" 活动相匹配，所以这个activity将会在前景中显示。
3. 在选择activity中，用户会选择要更新的联系人。之后，这个选择activity会调用方法[setResult\(resultcode, intent\)](#)创建一个intent然后返回到你的应用程

序中。这个intent会包含用户所选择的联系人的content URI以及额外的标志`FLAG_GRANT_READ_URI_PERMISSION`。这些标志会给你的应用程序授予URI权限以便你的应用程序可以通过content URI读取所需要数据。这个选择activity会调用`finish()`方法然后把控制权返还给你的应用程序。

4. 你的activity返回到前景之后，系统会调用你的activity的`onActivityResult()`方法。这个方法会收到People应用程序中选择activity返回的intent。

5. 通过返回的intent中的content URI，你可以从Contacts Provider中读取联系人联系人的信息，即便你在你的manifest文件中并没有要求永久的读取provider的权限。然后你可以得到联系人的生日信息或者他/她的电子邮件地址，并发送电子贺卡。

使用其它的应用程序

让用户能修改你没权限访问的数据的一个简单的方法就是激活有权限的应用程序，然后让用户可以在其中操作。例如，当Calendar应用程序接收到一个`ACTION_INSERT`intent，它就会让你激活这个应用程序的UI。你也可以通过这个intent传入额外的数据，让应用程序用来预填充UI. 由于反复产生的事件都会有复杂执行过程，所以将事件加入到Calendar Provider中的首选方式是用`ACTION_INSERT`激活Calendar 应用程序，然后让用户在其中加入事件。

契约类

Contract 类会定义一些常量，比如content URI，字段名，intent actions以及content provider的其它功能来帮助应用程序工作。Contract 类并没有被自动的包含在provider中；provider的开发人员需要定义它们，然后把他们提供给其他开发者们使用。Android平台包含了许多 provider，并在`android.provider`包中有相应的contract类。例如，User Dictionary Provider包含了一个契约类`UserDictionary`，并包含了不变的content URI和字段名。`UserDictionary.Words.CONTENT_URI`常量定义了"word"表的content URI。`UserDictionary.Words`类中也包含了字段名常量，在下面的演示代码中就用到了这种情况。例如，一个查询条件可以像下面那样定义：

```
String[] mProjection = {
    UserDictionary.Words._ID,
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCAL
```

};

Contacts Provider 能使用的另外一个contract类是 [ContactsContract](#)。这个类的说明文档中包含了代码演示的例子。 [ContactsContract.Intents.Insert](#)就是 [ContactsContract](#)的一个子类，它也是一个包含了intent和intent数据常量的contract 类。

引用MIME类型

Content provider 能返回标准的MIME media类型，也能返回自定义的MIME类型的字符串。 MIME类型的格式如下：

type/subtype

例如，众所周知的MIME类型 `text/html` 是由 `text`类型和 `html`子类型组成的。如果provider 把这种类型返回给URI，那就意味着用这个URI作为查询条件得到的结果是包含HTML标签的text类型。自定义的MIME类型的字符串，也被叫做"vendor-specific" MIME类型，还有更多复杂的类型和子类型。这种类型一般是下面这样的

`vnd.android.cursor.dir`

来返回多行记录，或者下面的

`vnd.android.cursor.item`

来返回单个记录。其子类型是特定的provider。Android 内置的provider 通常有一个简单的子类型。例如，当“联系人”应用程序创建一个电话号码时，它会在该行设置如下的MIME类型：

`vnd.android.cursor.item/phone_v2`

注意，这时候子类型的值就是 `phone_v2`。其他provider的开发人员可以根据provider 的权限和表名创建自己的子类型模式。例如，可以考虑在让某个provider 包含列车时刻表。那个这个provider 的权限就是`com.example.trains`，并且它包含了Line1,Line2,Line3表。这个content URI

`content://com.example.trains/Line1`

是给Line1表的反馈，provider 会返回MIME类型

vnd.android.cursor.dir/vnd.example.line1

这个content URI

content://com.example.trains/Line2/5

是给Line2表第5行数据的反馈， provider也是返回MIME类型

vnd.android.cursor.item/vnd.example.line2

大部分content provider 为它们使用的MIME类型定义了contract 类常量。例如， Contacts Provider 中的contract 类 [ContactsContract.RawContacts](#)，就为一个单一的联系人的MIME类型定义了[CONTENT_ITEM_TYPE](#) 常量。 单行数据的content URI在[Content URIs](#) 部分做了介绍。

来自“[index.php?title=Content_Provider_Basics&oldid=11886](#)”



Creating a Content Provider

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Jpengfly

原文链接：<http://developer.android.com/guide/topics/providers/content-provider-creating.html>

目录

[[隐藏](#)]

[1 创建一个Content Provider](#)

- [1.1 Before You Start Building\(开始创建之前\)](#)
- [1.2 Designing Data Storage\(设计数据存储\)](#)
 - [1.2.1 Data design considerations \(数据设计注意事项\)](#)
 - [1.2.2 Designing Content URIs\(设计Content URIs\)](#)
 - [1.2.3 Designing an authority \(设计一个身份认证\)](#)
 - [1.2.4 Designing a path structure \(设计一个路径结构\)](#)
 - [1.2.5 Handling content URI IDs \(处理content URI IDs\)](#)
 - [1.2.6 Content URI patterns\(Content URI模型\)](#)
- [1.3 \(Implementing the ContentProvider Class\) 实现ContentProvider类](#)
 - [1.3.1 Required methods \(请求方法\)](#)
 - [1.3.2 实现query\(\)方法](#)
 - [1.3.3 实现insert \(\) 方法](#)
 - [1.3.4 实现delete \(\) 方法](#)
 - [1.3.5 实现update \(\) 方法](#)
 - [1.3.6 实现onCreate \(\) 方法](#)
- [1.4 实现ContentProvider的MIME类型](#)
 - [1.4.1 MIME类型表](#)
 - [1.4.2 文件的MIME类型](#)
- [1.5 Implementing a Contract Class\(实现一个Contract类\)](#)
- [1.6 Implementing Content Provider Permissions\(实现Content Provider Permissions\)](#)
 - [1.6.1 Implementing permissions\(实现权限\)](#)
- [1.7 The <provider> Element\(<provider> 元素\)](#)
- [1.8 Intents and Data Access\(意图和数据访问\)](#)

创建一个Content Provider

Content Provider管理中央存储库的数据的访问, 你在Android程序中实现一个或者多个Provider, 连同清单文件中的元素. 其中一个类实现了ContentProvider子类, 这个子类是你的Provider和其他程序之间的接口. 尽管Content Provider意味着让数据对其他程序可见, 当然, 你也可以在你的程序里, 让用户查询和修改由Provider管理的数据.

这个主题的其他部分是创建Content Provider的基本步骤以及一些API的使用.

Before You Start Building(开始创建之前)

在你开始创建一个Provider之前, 做如下:

1. 确定你需要一个Content Provider. 如果你需要提供一个或多个以下特征, 虚拟需要创建Content Provider:

- 你希望向其他程序提供复杂的数据或者文件
- 你希望用户从你的程序复制复杂数据到其他程序
- 你希望提供用搜索引擎框架提供自定义搜索提示

如果用途完全只在你的程序内部, 你就不需要Provider去用SQLite数据库.

2. 如果你还没有准备好, 阅读 Content Provider Basics(Content Provider基础)学习更多关于Provider的知识

接下来, 遵循以下步骤来构建你的Provider:

1. Design the raw storage for your data. A content provider offers data in two ways:

文件数据

数据通常装入文件, 诸如图片, 音频, 或者视频. 将文件存储在你的私人空间. 作为其他程序请求你的文件的响应, 你的Provider提供一个文件的handle.

"格式化"数据

数据通常放入数据库, 数组, 或者类似结构. 将数据存储在一个表格, 匹配表的行和列. 一行表示一个实体, 如一个人或者一个清单条目. 一列表示实体的一些数据, 如人的名字, 或者条目的价格. 储存这种数据的常见方式是用SQLite 数据库, 但你可以使用任何类型的持久性存储.

2. 定义ContentProvider类的具体实现及其所需的方法。这个类是你的数据和Android系统的其余部分之间的接口。对于这个类的详细信息, 请参阅 [Implementing the](#)

ContentProvider Class 的部分。

3. 定义Provider的授权字符串，其内容的URI，和列名。如果你想Provider的程序处理的Intent，也要定义 Intent行动，额外的数据，和标志。还可以定义您将需要的应用程序要访问数据的权限。你应该考虑将所有这些值作为一个单独的contract类中的常量定义；之后，你将这个类公开给其他开发者。有关URI的更多信息，请参阅部分设计内容的URI。欲了解更多有关意图的信息，请参阅 [Intents and Data Access](#)。

4. 添加其他可选件，如样本数据或AbstractThreadingSyncAdapter的实现，能实现Provider和云端数据的同步

Designing Data Storage(设计数据存储)

Content Provider是用结构化格式保存的数据接口。在你创建接口之前，你必须确定如何存储数据。你可以将数据储存成任何形式，然后设计接口在必要时读写数据。

这是一些Android中的一些数据存储技术：

- Android系统提供SQLite数据库的API，用于Android自己的Provider存储面向表结构的数据。SQLiteOpenHelper类是用来访问数据库的基类。

记住，你不必用一个数据库去实现你的存储库。一个Provider外在表现为一组表格，就像关系型数据库，但是这个并不是Provider内部实现的必要部分。

- 对于存储文件数据，Android有多样的面向文件的APIs，要学习更多关于文件存储的信息，参阅Data Storage（数据存储）主题。如果你设计的Provider 提供媒体相关的数据，如音频和视频，你可以让Provider结合表数据结构数据和文件数据。
- 对于网络型数据，用java.net和android.net的类。你也可以将网络数据同步到本地存储，如数据库中，然后提供表数据或者文件。Sample Sync Adapter示例程序展示这种类型的同步。

Data design considerations (数据设计注意事项)

这里有一些建议供你设计Provider数据结构：

- Table数据常有一个主键列，Provider为每行提供一个唯一的数值。你可以使用这个值来联接一行到其他表中的相关行(使用它作为一个“外键”)。尽管你可以使用这个列的任何名字，使用BaseColumns_ID是个好的选择，因为联接到Provider查询一个ListView的结果，需要一个检索列有name_ID。
- 如果你想提供的位图图像或其他非常大的文件型数据块，将数据存储在文件，然后由它间接地提供，而不是直接存储在一个表。如果你这样做，你需要告诉你的Provider的用户，他们需要使用一个ContentResolver文件的方法来访问数据。
- 使用二进制大对象 (BLOB) 数据类型存储数据，不同大小有不同的数据结构。例如，你可以使用一个BLOB列存储protocol buffer (协议缓冲区) 或JSON结构。

您还可以使用一个**BLOB**实现一个单一模式表。在这种类型的表中，你定义一个主键列，一个**MIME**类型的列，以及一个或多个普通的**BLOB**列。在“**MIME**类型”列中的值表示在**BLOB**列中的数据的含义。这使您可以在同一表存储不同类型的行。Contacts Provider的“数据”表ContactsContract.Data架构的独立表的一个例子。就是schema-independent table（单一模式表）的一个实例。

Designing Content URIs(设计Content URIs)

Content URI是在Provider中标识数据的URI。Content URIs包含整个Provider的符号名称（它的签名）以及表或者文件的名称（路径）。选配的id部分指向表中的独立的行。每个访问ContentProvider的方法的数据以一个Content URI作为一个参数；这允许你指定表，行以及文件的访问。

Designing an authority (设计一个身份认证)

Provider通常有个单独的身份认证，作为其Android内部名称。为了避免与其他Provider冲突，你应该用完整的互联网域名（反转）作为你Provider认证的基础。因为这一建议也适用Android的包名，您可以定义provider身份认证作为包名的扩展。例如，如果你的Android包名是com.example.<appname>，你应该设置Provider的身份认证为com.example.<appname>.provider.

Designing a path structure (设计一个路径结构)

开发者通常在身份认证后附加指向单独表的路径的方式创建Content URIs。例如，如果你有两个表，table1和table2，你结合前面的例子的认证生成Content URIs com.example.<appname>.provider/table1 and com.example.<appname>.provider/table2。路径不局限于单一的segment，没有必要产生每个级别的路径表。

Handling content URI IDs (处理content URI IDs)

按照惯例，通过接收一个带有行ID在URI末尾的content URI，providers提供访问一个表中单独的行。按照惯例，providers匹配ID值与表的ID列，并执行对行的访问请求的匹配。

本惯例有助于为应用程序访问Provider提供通用的设计模式。应用程序针对Provider做一个查询，然后用 CursorAdapter在 ListView中显示结果Cursor。CursorAdapter的定义需要 Cursor中的一列作为_ID.

然后用户选择UI中显示的一行来查看或者修改数据。应用程序从由ListView支持的Cursor获取对应的行。为这行取得_ID值，将之添加到content URI，发送访问请求到Provider。

Content URI patterns(Content URI模型)

为了帮你选择引入的content URI采取的动作，Provider API引入了个方便的类UriMatcher。将content URI模型映射到数值。您可以使用整数值在一个switch语句选择

所需的行动为内容的URI或URI相匹配的特定模式。

Content URI模型用通配符匹配Content URIs:

- 匹配字符串中有效字符的长度
- 匹配字符串中数字字符的长度

涉及便携Content URI处理的例子，考虑一个认证

为com.example.app.provider的Provider，它识别下面指向表的Content URIs:

- content://com.example.app.provider/table1: A table called table1.
- content://com.example.app.provider/table2/dataset1: A table called dataset1.
- content://com.example.app.provider/table2/dataset2: A table called dataset2.
- content://com.example.app.provider/table3: A table called table3.

这个Provider也识别这样的Content URIs，将行号加在它们后面，例如：

content://com.example.app.provider/table3/1 表示table3的第一行

以下的Content URI模型也适用: content://com.example.app.provider/*

匹配Provider中的任意Content URI

content://com.example.app.provider/*/table2/*:

匹配表中dataset1和dataset2的Content URI，但是不匹配table1或table3的Content URI。

content://com.example.app.provider/table3/#:匹配table3中单一行的Content URI，如：content://com.example.app.provider/table3/6表示第6行。

下面的代码段显示UriMatcher中的方法如何使用。这段代码处理整个table的URIs，不同与单一行的URIs，通过使用Content URI模型content://<authority>/<path>用于表，content://<authority>/<path>/<id>对于单一行。

addURI()方法映射一个authority和路径到一个整数。方法android.content.UriMatcher#match(Uri) match(){} 返回URI的整数值。用switch语句选择在查询整个表或者查询一条记录。

```
public class ExampleProvider extends ContentProvider {
...
    // Creates a UriMatcher object.
    private static final UriMatcher sUriMatcher;
...
    /*
     * The calls to addURI() go here, for all of the content URI patterns that
     * the provider
     *      * should recognize. For this snippet, only the calls for table 3 are shown.
     */
...
}
```

```

/*
 * Sets the integer value for multiple rows in table 3 to 1. Notice that no
wildcard is used
 * in the path
 */
sUriMatcher.addURI( "com.example.app.provider" , "table3" , 1 );

/*
 * Sets the code for a single row to 2. In this case, the "#" wildcard is
used. "content://com.example.app.provider/table3/3" matches, but
"content://com.example.app.provider/table3 doesn't.
*/
sUriMatcher.addURI( "com.example.app.provider" , "table3/#" , 2 );
...
// Implements ContentProvider.query()
public Cursor query(
    Uri uri,
    String[] projection,
    String selection,
    String[] selectionArgs,
    String sortOrder) {
...
    /*
     * Choose the table to query and a sort order based on the code returned
for the incoming
     * URI. Here, too, only the statements for table 3 are shown.
     */
    switch ( sUriMatcher.match(uri)) {

        // If the incoming URI was for all of table3
        case 1:

            if (TextUtils.isEmpty(sortOrder)) sortOrder = "_ID ASC";
            break;

        // If the incoming URI was for a single row
        case 2:

            /*
             * Because this URI was for a single row, the _ID value part is
             * present. Get the last path segment from the URI; this is the
_ID value.
             * Then, append the value to the WHERE clause for the query
             */
            selection = selection + " _ID = " uri.getLastPathSegment();
            break;

        default:
            ...
            // If the URI is not recognized, you should do some error
handling here.
    }
    // call the code to actually do the query
}

```

另一个类，ContentUris，提供了个便利的方法，用于处理Content URIs的id部分。类Uri和Uri.Builder包含便利的方法，解析存在的Uri对象和创建新的。

(Implementing the ContentProvider Class) 实现ContentProvider类

ContentProvider实例管理通过从其他应用程序请求的处理的方式访问一个结构化的数据集。所有形式的访问ContentResolver最终调用,然后调用一个具体的方法来访问的ContentProvider。

Required methods (请求方法)

ContentProvider的抽象类定义了六个抽象方法,您必须实现的作为你的自己的具体子类的一部分。除了onCreate(), 其他的这些方法调用一个客户程序尝试访问你的ContentProvider

query()

从您的Provider检索数据。使用参数选择表去查询,返回行和列,按顺序排列结果。返回数据作为Cursor对象。

insert()

插入一个新行到您的Provider。使用参数来选择目标表及获得的列值使用。为新插入的行返回一个Content URI。

update()

在你的Provider更新现有的行。使用参数选择表和行更新并得到更新的列值。返回更新的行数。

delete()

从你的Provider删除行。使用参数选择要删除的表和行。返回删除的行数。

getType()

返回Content URI对应的IME类型。这个方法在Implementing Content Provider MIME Types部分做了更详细的阐述。

onCreate()

初始化您的provider。Android系统一创建你的Provider就调用这个方法。注意直到ContentResolver对象试图访问时,Provider才创建。

注意,这些方法与ContentResolver中同样命名方法有相同的签名。

这些方法的实现应该遵循以下原则:

- 除了onCreate(), 所有的这些方法可能被多个线程同时调用,所以他们必须是线程安全的。学习更多关于多线程, 查看Processes and Threads专题
- 避免在onCreate()做耗时的操作。延迟初始化任务,直到他们真正需要。部分 Implementing the onCreate() method章节更细致的讨论了这个问题。
- 尽管您必须实现这些方法,除了返回预期的数据类型, 你的代码可以不做任何事。例如,您可能想要阻止其他应用程序将数据插入某些表。要做到这一点,您可以忽略该insert()调用和返回0。

实现**query()**方法

`ContentProvider.query()`方法必须返回一个`Cursor`对象,或者如果它失败了,抛出一个异常。如果您使用的是一个`SQLite`数据库作为数据存储库,您可以简单地返回一个`SQLiteDatabase`游标`query()`方法的类`SQLiteDatabase`。如果查询不匹配任何行,您应该返回一个游标的实例源`()`方法返回`0`。如果在查询过程发生了一个内部错误,你只需要返回`null`。

如果您没有使用`SQLite`数据库作为数据存储,使用其中一个`Cursor`的具体的子类。例如,`MatrixCursor`类实现一个`cursor`中的每一行当成一个数组对象。这个类,使用`addRow()`来添加一个新行。

记住,Android系统必须能够联系各个进程之间的异常。Android可以为下列异常进行有效的查询异常处理:

- `IllegalArgumentException`(如果你的provider接收一个无效的content URI,可以选择抛出这个异常)
- `NullPointerException`

实现**insert()**方法

`insert()`方法添加一个新行到相应的表,使用在`ContentValues`的`??`参数值。如果列名不在`ContentValues``??`参数中,你可能想在您的provider代码,或在您的数据库架构提供了它的默认值。

此方法应该返回新行的content URI。要构造这个,使用`withAppendedId()`追加新行的`_id`(或其他主键)值到表的content URI。

实现**delete()**方法

`delete()`方法实际并没有从您的数据存储中删除的行。如果您使用的是同步适配器到provider。你应该考虑用一个“`delete`”标记标记一个被删除的行,而不是完全的删除。同步适配器可以检查要被删除的行,从provider删除之前,将他们从服务器中删除。

实现**update()**方法

`update()`方法需要同`insert()`方法相同的`ContentValues``??`的参数,和`deleted()`和`ContentProvider.query()`用过相同的`selection` 和 `selectionArgs`参数。这可能让你复用这些方法之间的代码。

实现**onCreate()**方法

启动Provider时,Android系统调用的`onCreate()`。您应该在方法中只执行快速初始化任务,并推迟创建数据库和数据加载,直到provider实际收到的数据的请求。如果你在`onCreate()`做冗长的任务,你将减缓您provider的启动。从而这会减慢从provider到其他应用程序的响应。

例如，如果您使用的是SQLite数据库，你可以在ContentProvider.onCreate()中创建一个新SQLiteOpenHelper对象()，然后首次打开数据库时创建SQL表。为了推动这项工作，你第一次调用getWritableDatabase()，它会自动调用SQLiteOpenHelper.onCreate()方法。

以下两个片段展示ContentProvider.onCreate()和SQLiteOpenHelper.onCreate()之间的相互作用。第一个片段是ContentProvider.onCreate()的实现：

```
public class ExampleProvider extends ContentProvider {
    /*
     * Defines a handle to the database helper object. The MainDatabaseHelper
     * class is defined
     * in a following snippet.
     */
    private MainDatabaseHelper mOpenHelper;

    // Defines the database name
    private static final String DBNAME = "mydb";

    // Holds the database object
    private SQLiteDatabase db;

    public boolean onCreate() {
        /*
         * Creates a new helper object. This method always returns quickly.
         * Notice that the database itself isn't created or opened
         * until SQLiteOpenHelper.getWritableDatabase is called
         */
        mOpenHelper = new SQLiteOpenHelper(
            getContext(),           // the application context
            DBNAME,                 // the name of the database)
            null,                  // uses the default SQLite cursor
            1                      // the version number
        );
        return true;
    }

    ...
    // Implements the provider's insert method
    public Cursor insert(Uri uri, ContentValues values) {
        // Insert code here to determine which table to open, handle error-
        // checking, and so forth
        ...
        /*
         * Gets a writeable database. This will trigger its creation if it
        doesn't already exist.
         */
        db = mOpenHelper.getWritableDatabase();
    }
}
```

下一个片段是SQLiteOpenHelper.onCreate()的实现，包含一个辅助类：

```
... A string that defines the SQL statement for creating a table
```

```

private static final String SQL_CREATE_MAIN = "CREATE TABLE " +
    "main " +                                     // Table's name
    "(" +                                         // The columns in the table
    " _ID INTEGER PRIMARY KEY, " +
    " WORD TEXT" +
    " FREQUENCY INTEGER " +
    " LOCALE TEXT )";
...
/***
 * Helper class that actually creates and manages the provider's underlying data
repository.
 */
protected static final class MainDatabaseHelper extends SQLiteOpenHelper {

    /**
     * Instantiates an open helper for the provider's SQLite data repository
     * Do not do database creation and upgrade here.
     */
    MainDatabaseHelper(Context context) {
        super(context, DBNAME, null, 1);
    }

    /**
     * Creates the data repository. This is called when the provider attempts to
open the
     * repository and SQLite reports that it doesn't exist.
     */
    public void onCreate(SQLiteDatabase db) {
        // Creates the main table
        db.execSQL(SQL_CREATE_MAIN);
    }
}

```

实现**ContentProvider**的**MIME**类型

ContentProvider类有两个方法返回的**MIME**类型：

get`Type` ()

你必须在所有的**provider**中实现的方法之一。

get`StreamTypes` ()

如果您的**provider**提供文件，你会希望实现。

MIME类型表

get`Type` () 方法返回一个**MIME**格式的**String** 描述内容的**URI**参数返回的数据类型。开放的参数可以是一个模式，而不是一个特定的**URI**;在这种情况下，你应该返回类型与内容的**URI**模式相匹配的相关数据。

对于常见的数据类型，如文本，HTML或JPEG的，**get`Type`** () 应返回的数据的标准**MIME**类型。这些标准类型的完整列表可以在IANA **MIME Media Types**网站上获得。

对于**content URI**，指向表中的数据一行或多行，**get`Type`** () 应该返回一个**Android**的厂商

特定的MIME格式的MIME类型：

- 类型部分:vnd
- 子类型部分:

如果URI模型是一个单一的行:android.cursor.item/

如果URI模式是多行:android.cursor.dir/

- Provider指定部分: vnd.<name>.<type>

您提供<name>和<TYPE>。<name>的值应该是全球唯一的，对于对应的URI模型,<type>值应该唯一。贵公司的名称或您的应用程序的 Android包名称的某些部分是<name>的一个不错的选择。<TYPE>的一个不错的选择是与URI相关联的表的标识的字符串。

例如,如果一个Provider的认证是com.example.app.provider,并将一个表名为table1,为table1中的多个行的MIME类型是:

```
vnd.android.cursor.dir/vnd.com.example.provider.table1
```

对于单一行的table1,MIME类型是:

```
vnd.android.cursor.item/vnd.com.example.provider.table1
```

文件的MIME类型

如果您的provider提供文件，实现getStreamTypes () 。该方法返回一个MIME类型的文件，这个文件是你的provider可以返回一个给定的content URI的String数组。你应该过滤你提供的MIME类型过滤器参数，这样你只返回那些客户端要处理的MIME类型。

例如，考虑provider提供的照片图像，如:jpg, PNG, gif格式的文件。如果应用程序通过过滤器的字符串image/* (这是一个“图片”) 调用ContentResolver.getStreamTypes () ，然后的ContentProvider.getStreamTypes () 方法返回数组:

```
{ "image/jpeg", "image/png", "image/gif" }
```

如果应用程序仅在jpg文件感兴趣，那么它可以调用过滤字符串* \ JPEG

ContentResolver.getStreamTypes () , ContentProvider.getStreamTypes () 应返回:

```
{ "image/jpeg" }
```

如果您的provider没有提供任何过滤字符串要求的MIME类型，getStreamTypes () 应该返回null。

Implementing a Contract Class(实现一个Contract类)

Contract类是一个属于provider的public final类，它包含常量定义的URI，列名，MIME类型，和其他元数据。这个类建立的供应商和其他应用程序之间的contract，以确保即使有

改变 URI的实际值，列名，等等，provider也可以正确访问，。

Contract类可以帮助开发人员，因为它通常有助记符名称的常量，所以provider不太可能使用不正确的值的列名或URIs。因为 它是一个类，它可以包含的Javadoc文档。集成开发环境，如Eclipse可以自动完成从Contract类和显示常数的Javadoc的常量名。

provider不能从您的应用程序访问的Contract类的类文件，但他们可以从您提供的.jar文件静态编译到他们的应用程序，。

ContactsContract类和嵌套类Contract类的例子。

Implementing Content Provider Permissions(实现Content Provider Permissions)

Android系统的所有方面的权限和访问中在主题Security and Permissions进行了详细介绍。主题Data Storage还介绍了不同类型的存储安全和有效的权限。总之，要点如下：

- 默认情况下，在存储设备的内部存储的数据文件对您的应用程序和Provider是私有的。
- 您所创建的SQLiteDatabase数据库对您的应用程序和Provider是私有的。
- 默认情况下，你保存到外部存储的数据文件是公开的。你不能使用一个Content Provider，以限制访问外部存储中的文件，因为其他应用程序可以使用其他API读写它们。
- 打开或创建你内部储存设备上的文件或SQLite数据库，调用这个方法能给其他程序读或者写他们的方法。如果你使用内部文件或者数据库作为你的Provider的存储器，给它“可读”或“可写”访问。在 manifest为您的Provider设置的访问权限将不能保护您的数据。内部存储器访问文件和数据库的访问权限是“private”，你的 Provider的存储器不应该改变。

如果您要使用的content provider的权限，以控制对数据的访问，那么你应该将数据存储在内部文件，SQLite数据库，或“云”（例如，在远程服务器上）的数据，你应该保持文件和数据库对您的应用程序私有。

Implementing permissions(实现权限)

所有的应用程序可以读取或写入您的Provider，即使底层数据是私人的，因为默认情况下，你的Provider没有的权限集。要改变这种状况，在您的manifest文件中设置权限，用<provider>元素的属性或者子属性。你可以为以下情况设置权限，适用整个provider，或者特定的表，或者单一的记录，或者前面3项。

你用一个或多个mainifest文件中的<permission>元素定义你Provider的属性。为了使您的Provider有特有的限定名，使用Java风格的作用域为android: name属性。例如，指定读权限的 com.example.app.provider.permission.READ_PROVIDER.

以下列表说明Provider的权限范围，最开始的适用于整个Provider，然后变得更加细化。更加细化的权限覆盖大范围的权限：

单一读写Provider级别权限

一个权限控制读写访问整个Provider,指定了android:权限属性的<provider>元素。

分开的读,写Provider级别权限

读权限和写权限为整个Provider。您可以在<provider>元素中指定Android: readPermission和androidwritePermission属性。他们覆盖了android:permission的权限。

路径级别权限

读,写,或读/写权限作用与您Provider中的Content URI。你指定你想控制每个URI与"的<provider>元素<path-permission>子元素。对于每个您指定的Content URI,你可以指定一个读/写权限,读权限,写权限,或所有三个。读取和写入权限覆盖读/写权限。此外,路径级别的权限覆盖Provider级别的权限。

临时权限

权限级别授予临时访问的应用程序,即使应用程序没有通常所需的权限。临时访问功能减少了应用程序要求在其清单的权限的数量。当你打开临时权限,只有需要Provider的“永久性”的权限,不断访问你的所有数据。

考虑实施时,你要允许外部图像浏览器应用程序,以显示您提供的照片附件的电子邮件服务提供商和应用,你需要的权限。为了让图像浏览器,没有必要设立临时的权限为内容的照片的URI需要的权限,访问。设计您的电子邮件应用程序,这样,当用户要显示一张照片,应用程序发送一个含有照片的内容 URI和权限标志的图像浏览器的意图。图像浏览器就可以查询您的电子邮件提供商检索照片,即使观众不正常读取为您提供的权限。

要打开的临时权限,或者设置了<provider>元素的android:grantUriPermissions属性,或将一个或多个<grant-uri-permission>子元素添加到您的<provider>元素。如果您使用的临时权限,你必须调用Context.revokeUriPermission () 每当你从Provider删除Content URI支持,Content URI拥有一个临时的权限。

您的Provider有多少是由访问该属性的值决定。如果该属性设置为true,则系统将给予临时许可给您的整个Provider,覆盖您的Provider级别或路径级别的权限。

如果这个标志设置为false,那么您必须添加<grant-uri-permission>子元素到你的<provider>元素。每个子元素指定Content URI或URI临时授权访问。

要指定临时访问权限给应用程序,意图必须包含的FLAG_GRANT_READ_URI_PERMISSION或FLAG_GRANT_WRITE_URI_PERMISSION标志,或两者兼而有之。这些用setFlags () 方法设置。

如果 `android:grantUriPermissions` 属性没有显示，假定为 `false`.

The `<provider>` Element(`<provider>` 元素)

像Activity和Service组件一样，ContentProvider的子类必须定义在其应用的manifest文件中，使用的`<provider>`元素。Android系统得到元素的以下信息：

认证(`android:authorities`)

符号名称，标识系统内的整个provider。这个属性在章节 [Designing Content URIs](#) 中做了更详细的描述。

Provider类的名字 (`android:name`)

实现ContentProvider的类。这个类在章节 [Implementing the ContentProvider Class](#) 中做了更详细的描述。

权限

指定权限的属性，其他应用程序必须为了访问Provider的数据：

- `android:grantUriPermissions`: 临时权限标志。
- `android:permission`: 单一的provider范围读/写权限
- `android:readPermission:Provider` 范围读权限
- `android:writePermission:Provider` 范围写权限

权限及其相应的属性在[Implementing Content Provider Permissions](#)章节中做了更详细的描述。

启动和控制属性

这些属性决定如何以及何时启动Android系统Provider，该Provider的过程特性，以及其他运行时设置：

- `android:enabled`: 标志允许系统开启Provider.
- `android:exported`: 标志允许其他程序使用这个Provider
- `android:initOrder:Provider` 应该开始的顺序，涉及相同进程中的其他Provider.
- `android:multiProcess:Provider`: 标志允许系统作为调用客户端开启同一个进程中的Provider.
- `android:process:Provider`: 应该运行的进程的名字.
- `android:syncable:Provider`: 标志表明Provider的数据是同服务器同步的.

这个属性是完全记录在dev指南主题的`<provider>`元素。

信息属性

provider的可选icon和label

• **android:icon:**一个绘图资源包含Provider的图标。icon显示应用程序列表中的下一个provider标签在Setting > Apps > All.

- **android:label:**描述provider或它们数据或两者的信息标签.这个标签的列表中显示的应用程序中 Settings > Apps > All.

这个属性在dev指南主题的<provider>元素中有完整的记录.

Intents and Data Access(意图和数据访问)

应用程序可以通过Intent间接的访问ContentProvider。应用程序不调用任何ContentResolver或 ContentProvider的方法。相反，它发送一个启动活动的意图，这往往是Provider的自己的应用程序的一部分。目标Activity是负责在UI上检索和显示的数据。根据Activity的Intent，目标Activity也可能提示用户修改Provider数据。目标活动在UI中显示的意图，也可能包含“额外”的数据，然后在改变Provider的数据之前，用户可以有选择性的改变数据。

你可能想使用意图访问确保数据的完整性。你的Provider可能会依赖于数据的插入，更新和删除按照严格定义的业务逻辑。如果是这样的情况下，允许其他应用程序直接修改您的数据可能会导致无效的数据。如果你想要开发者使用意图访问，确保完全的记录它。向他们解释为何使用自己的应用程序的 UI的意图存取比用他们的代码修改数据好。

处理传入的意图修改您Provider数据的意图，处理其他的意图没有什么不同。你可以了解更多有关使用Intent的主题 Intents and Intent Filters。

来自 "[index.php?title=Creating_a_Content_Provider&oldid=8748](#)"



Calendar Provider

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

[链接标题](#) 负责人：孙希凯

社区ID：sunxikai

完成时间：8月1日

原文链接：<http://developer.android.com/guide/topics/providers/calendar-provider.html>

目录

[[隐藏](#)]

- [1 Calendar Provider](#)
 - [1.1 Basics](#)
 - [1.2 Calendars Table](#)
 - [1.2.1 Querying a calendar](#)
 - [1.2.2 Modifying a calendar](#)
 - [1.2.3 Inserting a calendar](#)
 - [1.3 Events Table](#)
 - [1.3.1 Adding Events](#)
 - [1.3.2 Updating Events](#)
 - [1.3.3 Deleting Events](#)
 - [1.3.4 Adding Attendees](#)
 - [1.4 Reminders Table](#)
 - [1.4.1 Adding Reminders](#)
 - [1.5 Instances Table](#)
 - [1.5.1 Querying the Instances table](#)
 - [1.6 Calendar Intents](#)

- [1.6.1 Using an intent to insert an event](#)
- [1.6.2 Using an intent to edit an event](#)
- [1.6.3 Using intents to view calendar data](#)
- [1.7 Sync Adapters](#)

Calendar Provider

日历提供器是用来存放用户的日历事件的。日历提供器提供接口并且允许你执行查询，插入，更新和删除操作，日历，事件，与会者，提醒，等等。日历提供器的接口可以被应用程序和同步适配器使用。规则取决于什么类型的执行者正在做出的请求。这个文档侧重于作为一个应用程序如何来使用日历提供器提供的接口。想知道同步适配器的不同，请参考同步适配器([Sync Adapters](#))。

通常想要读或者写日历数据，应用程序清单必须包含适当的描述用户权限的声明。为了更加容易展示常用的操作，日历提供器提供了可以设置Intent的操作，就像日历中描述的的Intents一样。这些Intents可以满足用户去对日历应用程序进入插入，查看和编辑操作。用户与日历应用程序交互，然后返回到原来的应用程序。因此，您的应用程序既不需要请求权限许可，也不需要创建应用界面来编辑或者创建活动。

Basics

[Content providers](#)用来存储数据，让日历应用程序方便可用。内容提供器提供的机器人平台（包括日历提供器）通常暴露数据作为一套表的基础上的关系数据库模型，其中每一行是一个记录，每一列的数据的一种特殊类型和意义。通过日历提供器的API，应用和同步适配器可以读/写访问有保持用户的日历数据的数据库表。

每一个内容提供器有一个公开的URI(被包装成一个开放的对象)它作为一个唯一标示数据集。一个内容提供器控制多个数据集(多表)为每一个使用者提供一个独立的URI。提供器的所有URIs都以"content://"为开头。对应的标示数据被内容提供器控制着。日历提供器定义所有类别(表)中的URI作为常量。

这些URIs的形式就是<class>.CONTENT_URI. 例如, [Events.CONTENT_URI](#).

图1显示的图形表示的是日历提供的数据模型。它显示的主表和领域，其链接到对方。



图1. 日历提供器的数据模型。

用户可以有多个日历，并且不同的日历可以与不同类型的账户关联(谷歌日历，交换，等等)。这日历契约定义了数据模型的日历和事件的相关信息。此数

据存储在多个表，列表如下。

Table (Class)	Description
CalendarContract.Calendars	本表保存日历的具体信息。在该表中每一行为一个单一的日历包含细节信息，如名称，颜色，同步信息，等等。
CalendarContract.Events	这表具有事件的具体信息。在本表中每一行都是具有单个事件信息--例如，事件，地点，开始时间，结束时间，等。事件可以发生一次性或可发生多次。与会者，提醒，并扩展属性存储在不同的表。他们各有一个可以用来引用表中 _id 对应事件的 event_id 。
CalendarContract.Instances	此表为每一个事件保存开始和结束的时间。本表的每一行代表一个单一事件的发生。一次性事件有一个1 : 1的映射的实例事件。周期性活动，多行是自动生成对应于多发生的事件。
CalendarContract.Attendees	此表保存活动的参加者(客)的信息。每一行代表一个单一的客人的事件。它指定类型的客人与客人参加反应的事件。
CalendarContract.Reminders	此表保存警报/通知数据。每一行代表一个单一的警报事件。事件可以有多个提醒。每个事件的最大提醒数量是通过 MAX_REMINDERS 指定的，这是一套由同步适配器设定，并且它拥有给定的日历。提醒指定的分钟前的事件和方法的决定用户将被如何提醒。

日历提供器的接口的设计是灵活的，强大的。同时，重要的是提供了一个良好的用户体验和保护日历的完整性及日历的数据。为此，这里有一些事情在使用API的时候一定要记住：

- 插入，更新，并查看日历事件。你需要适当的权限来从日历事件提供器对日历的数据进行直接插入，修改，和阅读。然而，如果你没有建立一个正式的日历应用程序或同步适配器，就没必要要求这些权限。你可以使用机器人日历程序支持的意图来代替去对应用程序执行读取和写入操作。当你使用意图的时候，您的应用程序会调用日历程序中已经填充在表格中的所要执行的操作。当它们完成之后，他们返回到你的应用程序。设计您的应用程序通过日历执行公共操作，您需要为用户提供一个一致的，强大的用户界面。这是推荐的方法。更多信息，请参考[Calendar Intents](#)。

- 同步适配器。一个同步器可以同步用户的设备与另一个服务器或数据源日历数据。在[CalendarContract](#)。日历和[Calendar events](#)。事件表，有列是保留给同步适配器使用的。供应商和应用程序不应修改。事实上，他们是不可见的，除非他们是作为一个同步适配器。更多关于同步适配器的信息，请参考[Sync Adapters](#)。

User Permissions

应用程序必须在应用程序清单中添加[READ_CALENDAR](#) 权限，来读取日历数据。它必须包含[WRITE_CALENDAR](#)权限来删除，插入和更新日历数据：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>
    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WRITE_CALENDAR" />
    ...
</manifest>
```

Calendars Table

表[CalendarContract.Calendars](#)包含每个日历的细节。下面的日历列对于应用程序和同步适配器来说是可以写入的。想知道支持域的一个完整的列表，请参考[CalendarContract.Calendars](#)。

Constant	Description
NAME	日历的名字
CALENDAR_DISPLAY_NAME	这个日历展示给用户看的名字
VISIBLE	一个布尔值，指示是否被选中要显示的日历。值为0表示不应该显示此日历相关联的事件。值为1表示应显示此日历相关联的事件。此值会影响在 CalendarContract.Instances 表中行的下一代。
SYNC_EVENTS	一个布尔值，指示是否应同步日历，并将事件存储在设备上。值为0表示不同步这个日历或存储设备上

的事件。值为1表示同步此日历的事件，并在设备上存储的事件。

Querying a calendar

这里有一个例子来说明，如何来取得一个特定用户的所拥有的日历。简单来说，在这个例子中，查询操作显示在用户界面线程（“主线程”）。在实践中，这项工作应在异步线程，而不是在主线程。想知道更多的探讨，请参考装载机([Loaders](#))。如果你不仅仅是读取数据，还要进行修改，请参考[AsyncQueryHandler](#)”。

```
// Projection array. Creating indices for this array instead of doing
// dynamic lookups improves performance.
public static final String[] EVENT_PROJECTION = new String[] {
    Calendars._ID, // 0
    Calendars.ACCOUNT_NAME, // 1
    Calendars.CALENDAR_DISPLAY_NAME, // 2
    Calendars.OWNER_ACCOUNT // 3
};

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_ACCOUNT_NAME_INDEX = 1;
private static final int PROJECTION_DISPLAY_NAME_INDEX = 2;
private static final int PROJECTION_OWNER_ACCOUNT_INDEX = 3;
```

在例子的下一部分，你可以构建你的查询。选择指定查询的条件。在这个例子中，查询正在寻找有ACCOUNT_NAME的日历。“sampleuser@google.com”，这个账户类型(ACCOUNT_TYPE “com.google”),和拥有者账户(OWNER_ACCOUNT “sampleuser@google.com”)。如果你想看到所有用户浏览，日历，不仅是日历的用户拥有，省略的OWNER_ACCOUNT。查询返回一个Cursor对象，你可以用它来遍历数据库查询返回的结果集。更多关于查询内容提供器的探讨，请参考[ContentProviders](#)。

```
// Run query
Cursor cur = null;
ContentResolver cr = getContentResolver();
Uri uri = Calendars.CONTENT_URI;
String selection = "(" + Calendars.ACCOUNT_NAME + " = ?) AND (" +
    + Calendars.ACCOUNT_TYPE + " = ?) AND (" +
    + Calendars.OWNER_ACCOUNT + " = ?)";
String[] selectionArgs = new String[] { "sampleuser@gmail.com", "com.google",
    "sampleuser@gmail.com" };
// Submit the query and get a Cursor object back.
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs, null);
```

下一节使用游标遍历结果集。它使用常量，例如开始返回的每个字段的值。

```
// Use the cursor to step through the returned records
while (cur.moveToFirst()) {
    long calID = 0;
    String displayName = null;
    String accountName = null;
    String ownerName = null;

    // Get the field values
    calID = cur.getLong(PROJECTION_ID_INDEX);
    displayName = cur.getString(PROJECTION_DISPLAY_NAME_INDEX);
    accountName = cur.getString(PROJECTION_ACCOUNT_NAME_INDEX);
    ownerName = cur.getString(PROJECTION_OWNER_ACCOUNT_INDEX);

    // Do something with the values...
    ...
}
```

Modifying a calendar

要执行一个更新的日历，你可以提供的日历 [_id](#) 作为附加标识的 [URI\(long\) \(withAppendedId\(\)\)](#)，或作为第一选择的项目。应该开始选择“[_ID = ?](#)”，和第一 [selectionArg](#) 应该是 [\[1\]](#) 日历。你也可以通过编码的 URI 中的 ID 来做更新。这个例子改变日历的显示名称使用 [\(long\) \(withAppendedId\(\)\)](#) 方法：

```
private static final String DEBUG_TAG = "MyActivity";
...
long calID = 2;
ContentValues values = new ContentValues();
// The new display name for the calendar
values.put(Calendars.CALENDAR_DISPLAY_NAME, "Trevor's Calendar");
Uri updateUri = ContentUris.withAppendedId(Calendar.CONTENT_URI, calID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);
```

Inserting a calendar

日历设计是用同步适配器来管理，所以你可以用同步适配器来插入一个日历。在大多数情况下，应用程序只能肤浅的改变，如改变显示名称日历。如果应用程序需要创建一个本地日历，它可以执行同步适配器插入日历，使用 [ACCOUNT_TYPE_LOCAL ACCOUNT_TYPE](#)。 [ACCOUNT_TYPE_LOCAL](#) 是一个不与相关设备账户相关的，特殊的帐户类型的日历。这种类型的日历不与服务同步，更多关于同步适配器的探讨，请参考 [Sync Adapters](#)。

Events Table

[CalendarContract.Events](#)

[WRITE_CALENDAR](#)

表包含对个别事件的细节。对于添加，更新或删除事件的操作，应用程序必须包括
程序和同步适配器可以对以下事件列进行写操作。需要查看支持字段的完整列表，请参考[CalendarContract.Events](#)。

权限在其清单文件中。应用

Constant	Description
CALENDAR_ID	属于该_id的日历事件
ORGANIZER	电子邮件的组织者(拥有者)的事件。
TITLE	时间的标题
EVENT_LOCATION	事件发生的地方
DESCRIPTION	事件的描述
DTSTART	时间是事件开始于协调世界时(UTC，协调互联网纪时，又称互联网纪标准时间，为加特林威治标准时间的新名，避免惟独彼方的感觉。简称UTC，从英文“Universal Time, Coordinated”来)的时间以来的毫秒时代。
DTEND	时间是事件结束于协调世界时(UTC，协调互联网纪时，又称互联网纪标准时间，为加特林威治标准时间的新名，避免惟独彼方的感觉。简称UTC，从英文“Universal Time, Coordinated”来)的时间以来的毫秒时代。
EVENT_TIMEZONE	事件的时区
EVENT_END_TIMEZONE	事件结束的时区

DURATION	事件活动期间是以rfc5545为格式。例如，一个价值“pt1h”状态，事件应该持续一个小时，而一个价值“p2w”则表明了2周的时间。
ALL_DAY	值为1表示此事件占据了整整一天，由当地的时区定义。值为0表明它是一个普通的事件，可能会启动并在在一天内任何时间结束。
RRULE	复发事件格式规则。例如，“频率=每周数= 10; WKST=苏”。你可以在这里找到更多的例子。
RDATE	复发事件的日期。您通常使用的RRULE一起的RDATE定义一个重复出现的总集。更多的讨论，请参阅RFC5545规范。
AVAILABILITY	如果此事件计数为忙碌的时候，或者是空闲时间是可以预定的。
GUESTS_CAN MODIFY	无论来访者是否可以修改事件。
GUESTS_CAN_INVITE_OTHERS	无论来访者是否可以访问其它来访者。
GUESTS_CAN_SEE_GUESTS	无论来访者是否可以查看来访者清单。

Adding Events

当您的应用程序中插入一个新的事件，我们建议您使用一个[INSERT](#)意图，就像在使用如何使用意图插入事件中描述的那样。不过，如果你需要，你可以直接插入事件。本节介绍如何做到这一点。下边是用意图插入事件的规则：

- 您必须包括CALENDAR_ID和DTSTART。

- 你必须包括一个**EVENT_TIMEZONE**。为了得到一个系统的安装时间区域ID，使用`getAvailableIDs()`的名单。注意，如果你使用插入事件在这种情况下，这个规则并不适用，这个在如何使用意图进行插入事件中有详细讲解。默认时区提供的意图中所描述的**INSERT**意图，通过插入一个事件。
- 对于非经常性事件，你必须包括**DTEND**。
- 对于反复出现的事件，你必须包括时间间隔除了**RRULE**或**RDATE**外。请注意，如果你通过**INSERT**意图来插入事件将不适用这条规则，说明使用意图插入插入一个事件，在这种情况下，你可以使用一个的**RRULE**，在与**dtstart**和**dtend**一起和日历应用程序的转换它自动工期。

这里是一个插入事件的例子。这简洁的UI线程正在执行。在实践中，插入和更新应在异步线程移动到后台线程的行动。有关详细信息，请参阅**AsyncQueryHandler**。

```
long calID = 3;
long startMillis = 0;
long endMillis = 0;
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 9, 14, 7, 30);
startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 9, 14, 8, 45);
endMillis = endTime.getTimeInMillis();
...

ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Events.DTSTART, startMillis);
values.put(Events.DTEND, endMillis);
values.put(Events.TITLE, "Jazzercise");
values.put(Events.DESCRIPTION, "Group workout");
values.put(Events.CALENDAR_ID, calID);
values.put(Events.EVENT_TIMEZONE, "America/Los_Angeles");
Uri uri = cr.insert(Events.CONTENT_URI, values);

// get the event ID that is the last element in the Uri
long eventId = Long.parseLong(uri.getLastPathSegment());
//
// ... do something with event ID
//
```

注意：请仔细看这个例子是如何捕捉在事件发生后所创建的ID的。这是最简单的方式来获得一个事件ID。你经常需要执行其他操作，例如日历，添加与会者或事件提醒的事件ID。

Updating Events

当您的应用程序希望允许用户编辑事件，我们建议您使用一个编辑的意图，就像在使用的意图编辑事件这部分当中描述的那样。不过，如果你需要，你可

以直接编辑事件。执行一个事件的更新，你可以提供该事件的`_id`或者附加标识的URI`(long) (withAppendedId())`，或作为第一选择的项目。应该开始选择“`_ID = ?`”，和第一`selectionArg`应该是`_id`事件。你也可以选择使用没有ID的更新。下面是一个更新事件的例子。它使用`(long) withAppendedId()`方法改变了事件的标题：

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 188;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri updateUri = null;
// The new title for the event
values.put(Events.TITLE, "Kickboxing");
myUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().update(updateUri, values, null, null);
Log.i(DEBUG_TAG, "Rows updated: " + rows);
```

Deleting Events

无论是作为附加标识的`URI_id`，或通过使用标准的选择，您都可以通过他们来删除事件。如果使用附加的ID，你也不能做一个选择。有两个版本：删除应用程序和同步适配器。应用程序删除，删除列设置为1。这个标志告诉同步适配器，该行已被删除，这个删除应传播到服务器。同步适配器删除连同其相关的所有数据从数据库中删除的事件。这里有一个通过其`_id`去删除应用中的事件的一个例子：

```
private static final String DEBUG_TAG = "MyActivity";
...
long eventID = 201;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
Uri deleteUri = null;
deleteUri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
int rows = getContentResolver().delete(deleteUri, null, null);
Log.i(DEBUG_TAG, "Rows deleted: " + rows);
```

==Attendees Table==

`CalendarContract.Attendees`表的每一行代表一个单独的与会者或做客事件。调用查询方法返回一个与会者为在事件与给予`EVENT_ID`的名单。此`EVENT_ID`，必须符合特定事件的`_id`。下表列出了可写的领域。当插入一个新的与会者，您必须包括除`ATTENDEE_NAME`的所有与会者。

Constant	Description
<code>EVENT_ID</code>	事件的ID。

ATTENDEE_NAME	与会者的ID。
ATTENDEE_EMAIL	与会者的电子邮件地址。
ATTENDEE_RELATIONSHIP	与会者与事件的关系，它们中的一个： <ul style="list-style-type: none">• RELATIONSHIP_ATTENDEE• RELATIONSHIP_NONE• RELATIONSHIP_ORGANIZER• RELATIONSHIP_PERFORMER• RELATIONSHIP_SPEAKER
ATTENDEE_TYPE	与会者的类型，它们中的一个： <ul style="list-style-type: none">• TYPE_REQUIRED• TYPE_OPTIONAL
ATTENDEE_STATUS	与会者出席的状态，它们中的一个： <ul style="list-style-type: none">• ATTENDEE_STATUS_ACCEPTED• ATTENDEE_STATUS_DECLINED• ATTENDEE_STATUS_INVITED• ATTENDEE_STATUS_NONE• ATTENDEE_STATUS_TENTATIVE

Adding Attendees

下面是一个例子，增加了一个单一的与会者事件。注意，需要EVENT_ID：

```

long eventID = 202;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Attendees.ATTENDEE_NAME, "Trevor");
values.put(Attendees.ATTENDEE_EMAIL, "trevor@example.com");
values.put(Attendees.ATTENDEE_RELATIONSHIP, Attendees.RELATIONSHIP_ATTENDEE);
values.put(Attendees.ATTENDEE_TYPE, Attendees.TYPE_OPTIONAL);
values.put(Attendees.ATTENDEE_STATUS, Attendees.ATTENDEE_STATUS_INVITED);
values.put(Attendees.EVENT_ID, eventID);
Uri uri = cr.insert(Attendees.CONTENT_URI, values);

```

Reminders Table

[CalendarContract.Reminders](#)表的每一行代表一个对事件单独的提醒。通过EVENT_ID调用查询方法可以返回一个对事件进行提醒的提醒列表。作为提醒，下表列出了可写字段。插入一个新的提醒时，必须包括所有这些。请注意，同步适配器指定它们在[CalendarContract.Calendars](#)表中支持的提醒。详情请查看[ALLOWED_Reminders](#)。

Constant	Description
EVENT_ID	事件的ID。
MINUTES	在事件被解除前的分钟数。
METHOD	报警方法，如在服务器上设置，它们中的一个： <ul style="list-style-type: none"> • METHOD_ALERT • METHOD_DEFAULT • METHOD_EMAIL • METHOD_SMS

Adding Reminders

这个例子增加了一个对事件的提醒。这个提醒在事件前15分钟后发出。

```
long eventID = 221;
...
ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Reminders.MINUTES, 15);
values.put(Reminders.EVENT_ID, eventID);
values.put(Reminders.METHOD, Reminders.METHOD_ALERT);
Uri uri = cr.insert(Reminders.CONTENT_URI, values);
```

Instances Table

[CalendarContract.Instances](#)表持有一个发生了的事件的开始和结束时间。这个表的每一行代表一个单独事件的发生。实例表不可写，并且它只提供了一种方法来查询发生了的事件的。下表列出了一些领域，你可以查询一个实例。请注意时区是由[KEY_TIMEZONE_TYPE](#)和[KEY_TIMEZONE_INSTANCES](#)定义的。

Constant	Description
BEGIN	实例的开始时间，以UTC毫秒计。
END	实例的结束时间，以UTC毫秒计。
END_DAY	该实例的朱利安年底的一天，相对日历的时区来说。
END_MINUTE	在日历的时区午夜测量的实例年底分钟。
EVENT_ID	此实例对应的_ID事件。
START_DAY	朱利安开始一天实例，相对日历的时区。

START_MINUTE

实例的启动的时分测量午夜相对日历的时区。

Querying the Instances table

查询实例表，你需要指定一个URI查询范围。在这个例子中，[CalendarContract.Instances](#)获得通过其实施的[CalendarContract.EventsColumns](#)界面的标题字段。换句话说，返回的标题是通过数据库视图，而不是通过查询的原料[CalendarContract.Instances](#)表。

```

private static final String DEBUG_TAG = "MyActivity";
public static final String[] INSTANCE_PROJECTION = new String[] {
    Instances.EVENT_ID,           // 0
    Instances.BEGIN,              // 1
    Instances.TITLE               // 2
};

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_BEGIN_INDEX = 1;
private static final int PROJECTION_TITLE_INDEX = 2;
...

// Specify the date range you want to search for recurring
// event instances
Calendar beginTime = Calendar.getInstance();
beginTime.set(2011, 9, 23, 8, 0);
long startMillis = beginTime.getTimeInMillis();
Calendar endTime = Calendar.getInstance();
endTime.set(2011, 10, 24, 8, 0);
long endMillis = endTime.getTimeInMillis();

Cursor cur = null;
ContentResolver cr = getContentResolver();

// The ID of the recurring event whose instances you are searching
// for in the Instances table
String selection = Instances.EVENT_ID + " = ?";
String[] selectionArgs = new String[] { "207" };

// Construct the query with the desired date range.
Uri.Builder builder = Instances.CONTENT_URI.buildUpon();
ContentUris.appendId(builder, startMillis);
ContentUris.appendId(builder, endMillis);

// Submit the query
cur = cr.query(builder.build(),
    INSTANCE_PROJECTION,
    selection,
    selectionArgs,
    null);

while (cur.moveToNext()) {

```

```

String title = null;
long eventID = 0;
long beginVal = 0;

// Get the field values
eventID = cur.getLong(PROJECTION_ID_INDEX);
beginVal = cur.getLong(PROJECTION_BEGIN_INDEX);
title = cur.getString(PROJECTION_TITLE_INDEX);

// Do something with the values.
Log.i(DEBUG_TAG, "Event: " + title);
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(beginVal);
DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
Log.i(DEBUG_TAG, "Date: " + formatter.format(calendar.getTime()));
}
}

```

Calendar Intents

您的应用程序并不需要权限来读取和写入日历数据。相反，它可以使用机器人的日历应用程序所支持的意图，来关闭该应用程序的读写操作。下表列出了由日历提供支持的意向：

Action	URI	Description	Extras
VIEW	content://com.android.calendar/time/<ms_since_epoch> 您也可以参考到与CalendarContract.CONTENT_URI的URI。对于使用这种意图的一个例子，请参阅 Using intents to view calendar data 。	通过<ms_since_epoch>，以指定的时间打开日历。	无。
VIEW	content://com.android.calendar/events/<event_id>。您也可以参考到与Events.CONTENT_URI的URI。对于使用这种意图的一个例子，请参阅 Using intents to view calendar data 。	通过具体的<event_id>来查看对应的事件。	CalendarContract.EXTRA_EVENT_BEGIN_TIME。 CalendarContract.EXTRA_EVENT_END_TIME。
EDIT	content://com.android.calendar/events/<event_id>。您也可以参考到与Events.CONTENT_URI的URI。使用这种	通过具体的<event_id>来编辑对	CalendarContract.EXTRA_EVENT_BEGIN_TIME。 CalendarContract.EXTRA_EVENT_END_TIME。

	意图的一个例子，请参阅 Using an intent to edit an event 。	应的事件。	
EDIT INSERT	content://com.android.calendar/events。您也可以参考到与Events.CONTENT_URI的URI。使用这种意图的一个例子，请参阅 Using an intent to edit an event 。	创建一个事件。	任何额外的在表中列出。

下表列出了日历提供支持的额外的意图：

Intent Extra	Description
Events.TITLE	事件的名称。
CalendarContract.EXTRA_EVENT_BEGIN_TIME	事件开始是以划时代的毫秒时间计算。
CalendarContract.EXTRA_EVENT_END_TIME	事件结束是以划时代的毫秒时间计算。
CalendarContract.EXTRA_EVENT_ALL_DAY	一个布尔值，指示事件是整天。值可以是真或假。
Events.EVENT_LOCATION	事件的位置。
Events.DESCRIPTION	事件的详细描述。
Intent.EXTRA_EMAIL	这些受访问的电子邮件地址，时一个以逗号分隔的列表。
Events.RRULE	事件再次发生的规则。

Events.ACCESS_LEVEL	无论这个事件是私有的还是共有的。
Events.AVAILABILITY	如果此事件计数，可以被预定在空闲的时间或者忙碌的时间。

以下各节描述如何使用这些意图。

Using an intent to insert an event

使用INSERT意向，让您的应用程序的手关闭的事件插入任务日历本身。用这种方法，您的应用程序甚至并不需要在其清单文件中包含有WRITE_CALENDAR许可。当用户运行一个使用了此方法的应用程序。应用程序把它们发送日历程序来完成事件的添加。在INSERT意图使用额外的字段预先填充在日历事件的细节的一种形式。然后，用户可以取消的事件，需要编辑的形式，或保存在他们的日历事件。下面是一个代码段，计划2012年01月19日的事件，从7:30时至上午8时30分请注意以下有关此代码段：它指定的URI Events.CONTENT_URI。它采用了CalendarContract.EXTRA_EVENT_BEGIN_TIME和CalendarContract.EXTRA_EVENT_END_TIME额外的字段预先填充的形式与事件的时间。这些时代的价值观，必须从时代的UTC毫秒计。它采用的Intent.EXTRA_EMAIL额外的字段提供一个被邀请者的逗号分隔的列表，来指定电子邮件地址。

```
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 0, 19, 8, 30);
Intent intent = new Intent(Intent.ACTION_INSERT)
    .setData(Events.CONTENT_URI)
    .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis())
    .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis())
    .putExtra(Events.TITLE, "Yoga")
    .putExtra(Events.DESCRIPTION, "Group class")
    .putExtra(Events.EVENT_LOCATION, "The gym")
    .putExtra(Events.AVAILABILITY, Events.AVAILABILITY_BUSY)
    .putExtra(Intent.EXTRA_EMAIL, "rowan@example.com,trevor@example.com");
startActivity(intent);
```

Using an intent to edit an event

你可以直接更新事件，像更新事件中描述的那样。但是使用的编辑意图，允许应用程序没有权限来编辑的日历应用程序中的事件。当用户编辑完他们的日历中的事件，他们返回到原来的应用程序。这里是一个例子的意图设置为指定事件的新称号，并允许用户编辑在日历事件。

```
long eventID = 208;
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent = new Intent(Intent.ACTION_EDIT)
    .setData(uri)
    .putExtra(Events.TITLE, "My New Title");
startActivity(intent);
```

Using intents to view calendar data

日历提供器提供了两种不同的方法来使用视图意图：

- 要打开一个特定的日期的日历。
- 要查看事件。

下面是一个例子，演示如何打开一个特定的日期的日历：

```
// A date-time specified in milliseconds since the epoch.
long startMillis;
...
Uri.Builder builder = CalendarContract.CONTENT_URI.buildUpon();
builder.appendPath("time");
ContentUris.appendId(builder, startMillis);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(builder.build());
startActivity(intent);
```

下面是一个例子，说明如何打开查看事件：

```
long eventID = 208;
...
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, eventID);
Intent intent = new Intent(Intent.ACTION_VIEW)
    .setData(uri);
startActivity(intent);
```

Sync Adapters

在应用程序和同步适配器如何访问应用程序日历提供器上只有一些微小的差别：

- 同步适配器，需要指定它是一个同步适配器设置CALLER_IS_SYNCADAPTER为true。
- 一个同步适配器需要提供一个ACCOUNT_NAME和一个ACCOUNT_TYPE座位在URI中的查询参数。
- 同步适配器比一个应用程序或者组件有更多的写访问。例如，一个应用程序只能修改日历的一些特点，如它的名称，显示名称，可见性设置，以及是否同步日历，。相比之下，同步适配器可以访问不仅那些列，但如日历的颜色，时区，访问级别，地点，等许多人。然而，同步适配器对指定的ACCOUNT_NAME和ACCOUNT_TYPE是有限制的。

这里是一个辅助方法，您可以使用同步适配器返回的URI：

```
static Uri asSyncAdapter(Uri uri, String account, String accountType) {  
    return uri.buildUpon()  
        .appendQueryParameter(android.provider.CalendarContract.CALLER_IS_SYNCADAPTER, "true")  
        .appendQueryParameter(CalendarContract Calendars.ACCOUNT_NAME, account)  
        .appendQueryParameter(CalendarContract Calendars.ACCOUNT_TYPE, accountType).build();  
}
```

示例实现一个同步适配器（不包括专门涉及到日历），请参阅[SampleSyncAdapter](#)。

[acb](#)

来自“[index.php?title=Calendar_Provider&oldid=13780](#)”



Contacts Provider

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：tiandiruan 原文链

接：<http://docs.eoeandroid.com/guide/topics/providers/contacts-provider.html>

目录

[[隐藏](#)]

[1 联系人提供者](#)

[2 联系人提供者组织结构图](#)

[3 原联系人](#)

- [3.1 重要的原联系人列](#)

- [3.2 须知](#)

- [3.3 原联系人数据的源](#)

[4 数据](#)

- [4.1 描述性列名](#)

- [4.2 通用类型的列名称](#)

- [4.3 特定类型的列名称](#)

- [4.4 特定类型的列名类](#)

- [4.5 联系](#)

[5 其他联系人提供者功能](#)

- [5.1 联系人组](#)

- [5.2 联系人图片](#)

联系人提供者

联系人提供者是Android中一个强大并且灵活的组件，管理系统通讯录的数据。联系人提供者是系统的联系人应用程序的数据源，可以在自己的应用程序访问数据及在系统与在线服务之间传输数据。 提供者可以容纳一个范围广泛的数据

源并且试图管理尽可能多的联系人的数据，这样导致它的组织是复杂的。正因为如此，提供者的**API**包含了一组广泛的协议类和接口来促进数据的恢复和修改。

本指南将介绍一下内容：

- 。基本提供者的结构
- 。如何从提供者中检索数据
- 。如何修改提供者中的数
- 。如何写一个同步适配器为服务器中的数据同步到联系人提供者中

本指南假设你知道基本的**Android**内容提供者。学习更多的**Android**提供者，请阅读内容提供者的基本操作指南。样品同步适配器示例应用程序是使用同步适配器传输数据在联系人提供者和为谷歌Web服务主办的一个示例应用程序之间联系的一个例子。

联系人提供者组织结构图

联系人提供者是**Android**中一个内容提供者的组成部分，它维护关于一个联系人的三种类型的数据，其中每一联系个对应于由提供者提供的表，如在图1中所示的三种类型：

图1.联系人提供者表的结构

三个表通常被称为由其协议类的名称。类定义常量为内容的URI、列名和列值为表所用

ContactsContract.Contacts表

Contacts表包含了不同的联系人的记录

ContactsContract.RawContacts表

RawContacts表是联系人的数据集合，指定用户账号和类型

ContactsContract.Data表

Data表是存储具体的联系人信息，包括邮件、电话号码等

协议类**ContactsContract**为代表的其他表是内容提供者使用在系统的联系人或电话应用程序来管理其业务或支持特定功能的辅助表。

原联系人

Raw Contacts存储了用户所在服务器的账号和账号类型，由于联系人提供者允许多个在线服务作为一个联系人的数据源，同时也允许用户使用同一个服务但是不同的账号来存储联系人信息。

大部分**raw contacts**的数据并没有直接存储在**ContactsContract.RawContacts**表中，而是以一行或多行的形式存储在 **ContactsContract.RawContacts**表中，每一行有一个列**Data.RAW_CONTACT_ID**包含了一个指向 **ContactsContract.RawContacts**表的列**RawContacts._ID**，也就是说 **ContactsContract.RawContacts**表存储的是引用。

重要的原联系人列

表1中列举出**ContactsContract.RawContacts**表中重要的列名。请阅读表格下方的须知：

表1：行对应的重要的列名

须知

下面是**ContactsContract.RawContacts**表中重要的须知：

- 。账号不是存储在**ContactsContract.RawContacts**表的行中。而是存储在**ContactsContract.Data** 表中
的**ContactsContract.CommonDataKinds.StructureName**行中。一个账号只有一种类型存在于 **ContactsContract.Data**表中
- 。警告：为了能够在行中使用自己的账户数据，你必须先注册**AccountManager**. 要做到这一点，提示用户 添加用户类型和用户账号到账户列表中。如果你不这样做的话，联系热提供者会自动删除你的行信息。

例如，如果你希望你的应用程序对数据进行关联来实现基于Web服务的域名
为**com.example.dataservice**和为您服务的用户账号类型是

becky.sharp@dataservice.example.com，用户必须在你的应用程序可以添加行的时候添加账号类型 (com.example.dataservice) 和账号名称 (becky.smart@dataservice.example.com)。你可以在用户文档中解释这个需求，或者提示用户添加类型和名称，或两者。在下一章节中会详细的描述用户账号类型和用户账号名称。

原联系人数据的源

为了理解用户如何操作，要考虑一个用户"Emily Dickinson"在设备上是如何运行的，它在设备上定义了三个用户账号：

- emily.dickinson@gmail.com
- emilyd@gmail.com
- Twitter account "belle_of_amherst"

用户可能在账户设置中同步设置三个账号的信息。

假设Emily Dickinson打开一个浏览器窗口，以emily.dickinson@gmail.com登录到Gmail，打开账号，并且添加"Thomas Higginson"。后来，以emilyd@gmail.com登录到Gmail，并且发送一份邮件给Thomas Higginson，可以自动添加为联系人。还可以在Twitter上添加colonel_tom(Thomas Higginson在Twitter上的ID号)为联系人。

联系人提供者为同一个用户创建了三个账号的结果：

- Thomas Higginson的一个账号为emily.dickinson@gmail.com，用户类型是Google.
- Thomas Higginson的第二个账号为emilyd@gmail.com，用户类型是Google.尽管用户的名称与前一个是相同的，因为一个用户可以添加不同的用户账号。
- Thomas Higginson的第三个账号为belle_of_amherst，用户类型是Twitter。

数据

如前所述，用户数据存储在ContactsContract.Data行链接到_ID行的值。这允许

一个单一的用户有多个相同类型的数据，如 电子邮件地址和电话号码。例如，Thomas Higginson有一个家庭电子邮箱地址thigg@gmail.com和一个工作电子邮箱地址thomas.higginson@gmail.com，联系人提供者会把这两个电子邮箱地址存储在同一个行中并且链接给同一个用户。

不同的数据类型存储在一张表中，名字、电话号码、电子邮箱、邮政地址或者照片等都存储在ContactsContract.Data表中，为了方便管理数据，ContactsContract.Data表有一些描述性的列，还有其他一些一般的列。描述性列的内容不管数据的类型都有同样的意思，但是一般性的列的内容就会依据不同的数据类型有不同的意思。

描述性列名

下面是一些描述性列名的例子：

- RAW_CONTACT_ID

和RawContact的_ID对应的值

- MIMETYPE

该行的数据类型表示为一个自定义类型，联系人提供者将这种自定义类型定义在ContactsContract.CommonDataKinds子类中。这些自定义类型是开源的，可以被联系人提供者用于任何应用程序或同步适配器中

- IS_PRIMARY

如果这种类型的数据发生一次以上的联系，IS_PRIMARY列标志的数据行包含这种原始类型的数据。例如，如果用户 long-presses电话号码联系并选择默认设置，然后将ContactsContract.Data行中包含多少IS_PRIMARY列设置成一个的非零值。

通用类型的列名称

从DATA1到DATA15有15个字段来存储值，从SYN1到SYN4有四个字段来存储值并应用于同步适配器中。字段值常量不管包含什么类型的数据都是可用的。

DATA1一般是索引列，Contacts Provider通常用这个索引列来进行目标查询。例如，在邮件行中，这一列存储实际的邮件地址。

另外，DATA15是用来存储BLOB(Binary Large Object)类型数据的，比如用户头像。

特定类型的列名称

为方便操作这些特定类型的行，Contacts Provider还提供了一种定义在ContactsContract.CommonDataKinds子类当中的特定类型的类的名称常量。这种常量是同一列名称下的不同常量名称，有助于用户访问行中数据所定义的一种特殊类型的数据。

例如，ContactsContract.CommonDataKinds.Email类定义的特定类型的列的名称为ContactsContract.Data行中特定类型的Email.CONTENT_ITEM_TYPE。该类包含固定地址的电子邮件地址栏。电子邮件的地址的实际价值是“data1”，这是列的常用名称。

注意：

不要添加您自己的自定义数据到包含有一个预先定义的MIME类型列ContactsContract.Data表中。如果你这样做，你可能会丢失数据或导致provider故障。例如，你不应该添加MIME类型的Email.CONTENT_ITEM_TYPE行(它在DATA1中包含一个用户名，而不是一个电子邮件地址)。如果您使用自己的自定义MIME类型的行，那么你可以自由定义自己的特定类型的列名称并且使用他。

图2显示了如何描述在ContactsContract.Data行中的列和数据列，以及特定类型的列名“覆盖”如何通用列名

{图2} 

特定类型的列名类

表2列出了最常用的特定类型的列名类：

Mapping class	Type of data	Notes
ContactsContract.CommonDataKinds.StructuredName	The name data for the raw contact	A raw contact has only one of

	associated with this data row.	these rows.
ContactsContract.CommonDataKinds.Photo	The main photo for the raw contact associated with this data row.	A raw contact has only one of these rows.
ContactsContract.CommonDataKinds.Email	An email address for the raw contact associated with this data row.	A raw contact can have multiple email addresses.
ContactsContract.CommonDataKinds.StructuredPostal	A postal address for the raw contact associated with this data row.	A raw contact can have multiple postal addresses.
ContactsContract.CommonDataKinds.GroupMembership	An identifier that links the raw contact to one of the groups in the	Groups are an optional feature of an account type and account name. They're described in more

Contacts
Provider.detail in
the
section
Contact
groups.

联系

联系人提供者会把所有账户类型和名称结合起来形成联系。这样便于显示和修改所有用户数据。联系人提供者管理新联系人的创建，并与现有联系人汇总。应用程序和同步适配器都不允许添加联系人。

联系人提供者创建了新联系人以便响应添加不符合现有联系人的新raw联系。如果现有联系人的数据以这种方式变化，即不再匹配先前连接的联系，联系人也会采取这种做法。如果应用程序或者同步适配器创建了新的不匹配现有联系人的联系人，那么新联系人会汇总到现有联系中。

其他联系人提供者功能

除了在前面章节中描述的主要特点外，联系人提供者还提供了这些有用的功能和联系人数据：

- 。联系人组
- 。图片功能

联系人组

联系人提供者可以有选择性的标注相关的联系人组数据的集合。如果服务器要保持与用户账号相关联的组的账号类型，同步适配器为账号类型在联系人提供者 和服务器之间进行数据组数据的传输。当用户添加一个新的联系人到服务器，并将这个联系人放到一个新的组中，同步适配器一定会将这个新组放至 `ContactsContract.Groups` 表中。组和组中的原联系人使用 `ContactsContract.CommonDataKinds.GroupMembership` 表中的 MIME 类型存储在 `ContactsContract.Data` 表中。

如果你设计一个从服务器到联系人提供者的联系人数据的同步适配器，并且不使用组，那么你必须告诉提供者使你的数据可见。在执行用户添加账户到设备上，更新 `ContactsContract.Settings` 行时，联系人提供者会添加这个用户账号。

在这行当中，设置 `Settings.UNGROUPED_VISIBLE` 列为1.当你进行这些操作时，尽管你不使用组，联系人提供者始终会使联系人数据可见的。

联系人图片

来自“[index.php?title=Contacts_Provider&oldid=13850](#)”



Intents and Intent Filters

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/guide/topics/intents/intents-filters.html>

原编辑者：我是一棵菠菜

现编辑者：--[Snowxwyo](#) 2012年7月17日 (二) 16:00 (CST)

-

目录

[[隐藏](#)]

[1 Intents与Intent过滤器-Intents and Intent Filters](#)

- [1.1 Intent对象-Intent Objects](#)
 - [1.1.1 组件名-Component name](#)
 - [1.1.2 动作-Action](#)
 - [1.1.3 数据-Data](#)
 - [1.1.4 分类-Category](#)
 - [1.1.5 Extras](#)
 - [1.1.6 标志位-Flags](#)

[1.2 Intent解决方法-Intent Resolution](#)

- [1.2.1 Intent过滤器-Intent filters](#)

- [1.2.1.1 Action test](#)
- [1.2.1.2 分类测试-Category test](#)
- [1.2.1.3 数据测试-Data test](#)
- [1.2.1.4 一般案例-Common cases](#)
- [1.2.1.5 使用intent匹配-Using intent matching](#)

- [1.3 记事本案例-Note Pad Example](#)

Intents与Intents过滤器-Intents and Intent Filters

Android中的三个核心组件activities,services,broadcast receivers是通过Intent彼此联系、触发的。Intent是同一个或不同的应用中的组件之间的消息传递的媒介。Intent本身是一种‘被动’的数据结构，它抽象地描述了即将被执行的动作又或是在广播中已经发生的并且正在被通知的某个事件的描述。下面列出几种为每一种组件传递Intent的独立机制：

- [Context.startActivity\(\)](#)和[int\) Activity.startActivityForResult\(\)](#)可以通过Intent启动一个activity或者通过目前的activity去做新的事情
([Activity.setResult\(\)](#)可以通过Intent把信息传递给调用了[startActivityForResult\(\)](#)方法的activity)
- [Context.startService\(\)](#)可以通过Intent初始化一个service或者传递指令给一个已经工作的service.相似的，[Context.bindService\(\)](#)可以通

本文内容

[Intent对象-Intents Objects](#)

[Intent解决方法-Intent Resolution](#)

[Intent过滤器-Intent filters](#)

[一般案例-Common cases](#)

[使用Intent匹配-Using intent matching](#)

过Intent在发起呼叫的组件和目标service之间建立连接。Intent可以任意的初始化还没有运行的service.

- 任意的广播方法 (例如[Context.sendBroadcast\(\)](#),
[Context.sendOrderedBroadcast\(\)](#)或[Context.sendStickyBroadcast\(\)](#))
 可以通过Intent把消息传递给Broadcast Receiver。很多广播事件发起于系统级别的代码。

在以上的每一种情况中，Android系统都会找到合适的activity,service或者一套的broadcast receivers去响应Intent,必要的时候也可以通过Intent初始化他们。广播事件中的Intent只是传递给broadcast receivers.[startActivity\(\)](#)只是把Intent传递给activity，而不是service或是broadcast receiver，以此类推。以上我们知道这些传递消息的系统是没有任何重叠的。

本文档从描述Intent本身开始。接下来将描述Android用来匹配Intent和组件的规则-他是如何解决哪一个组件应该接受哪一个 Intent的事情。因为Intent并不明确地指定目标组件，所以传递的过程要通过intent filters的过滤以便传递给那些潜在的目标中合适的目标。

[记事本案例-Note Pad Example](#)

关键类

[Intent](#)

[IntentFilter](#)

[BroadcastReceiver](#)

[PackageManager](#)

Intent对象-Intent Objects

[Intent](#)本身是一个消息的集合。他包含那些传递给接收Intent的组件的信息 (such as the action to be taken and the data to act on) 和传递给Android系统的信息 (例如应该绑定到Intent上的组件的类别和如何启动目标activity的指南) 主要的，他包含了以下几点：

组件名-Component name

应该处理Intent的组件名(The name of the component that should handle the intent.)。这一段所描述的就是关于[ComponentName](#)的--他是目标组件的类名（例如"com.example.project.app.FreneticActivity"）和在manifest文件中注册过的包名（例如"com.example.project"）的联合体。组件名字中所包含的包名的部分不需要必须与manifest文件中的包名相匹配。

组件的名字是非强制性的。如果他是固定的，那么Intent就会被传递给指定名字的类的实例，如果他不是固定的，那么Android就会通过Intent中的其他信息找到合适的目标--可以查看本文档以后的提到的[Intent解决方法-Intent Resolution](#)。

组件名字可以通过[setComponent\(\)](#), [setClass\(\)](#)或[setClassName\(\)](#)来设置，通过[getComponent\(\)](#)来读取。

动作-Action

一个命名了将要被执行的动作的字符串，或在广播intents事件中，已经发生并被报告的动作。Intent类定义了许多动作常量，包括如下：

常量	目标组件	行为动作
ACTION_CALL	activity	初始化一次电话呼叫
ACTION_EDIT	activity	提供可编辑的数据
ACTION_MAIN	activity	作为程序入口启动，没有输入或输出

ACTION_SYNC	activity	同步服务端和移动端的数据
ACTION_BATTERY_LOW	broadcast receiver	电量低的警告
ACTION_HEADSET_PLUG	broadcast receiver	耳机已经插入或拔出设备
ACTION_SCREEN_ON	broadcast receiver	屏幕已经被打开
ACTION_TIMEZONE_CHANGED	broadcast receiver	时区设置已经发生变化

你可以访问[Intent](#)类的定义查看一系列的代表一般行为动作的常量。其余行为动作的定义可以在Android API文档中的其他地方找到。你也可以在应用中自定义这样的常量串，这些常量要以包名作为前缀，例如"`com.example.project.SHOW_COLOR`".

`action`的名字能够很好的说明intent有着怎样的机构--特别是[数据-Data](#)和[Extras](#)--就像方法的名字决定了参数和返回值。所以，使用一个明确的`action`的名字是一个很好的主意。另外，要为你的[Intent](#)定义一个完整的协议，而不是孤立的定义`action`。

我们用[setAction\(\)](#)来设置Intent中的`action`并用[getAction\(\)](#)来读取。

数据-Data

数据运行的URI和其MIME类型。不同的action被配与不同的data说明。例如，如果是ACTION_EDIT,那么他的data就包含提供编辑功能的文本的URI。如果是ACTION_CALL,那么他的data就是号码的URI--tel:。相似，如果是ACTION_VIEW并且data是http: URI,那么接收Intent的activity将下载并显示URI所指向的内容。

当为某个组件匹配一个可以处理数据的Intent的时候，通常除了要了解Data的URI以外，重要的是要知道Data的类型 (MIME type)。例如，一个可以展示图片的组件不应该被调用来播放音频。

在很多情况下，Data的类型可以从URI中推测出来，特别是URI所展示的内容：指出了Data被用在什么位置及被哪种content provider控制（参考[separate discussion on content providers](#)）。但是Data的类型也可以在Intent中明确的设定。[setData\(\)](#)方法设置Data的URI，[setType\(\)](#)设置Data的类型 (MIME type)，[setDataAndType\(\)](#)两者一起设置，[getData\(\)](#)读取URI，[getType\(\)](#)读取类型

分类-Category

Category是这样一个String：他包含了需要处理Intent的组件的种类的信息。很多Category的描述能够放在Intent里。就像Action那样，Intent也定义了一些Category常量，如下图表

常量	意义
CATEGORY_BROWSABLE	通过链接，目标activity可以安全地被浏览器引用来展示数据，例如一张图片或一个e-mail
CATEGORY_GADGET	activity可以嵌入另一个activity中来操控gadget
CATEGORY_LAUNCHER	activity能够成为任务的初始化activity and is listed in the top-level

	application launcher
CATEGORY_HOME	activity可以展示主页面--当设备第一次打开或当Home button被按下时候
CATEGORY_PREFERENCE	目标activity是优先选择的activity

参考[Intent](#)可以查看全部Category的列表

[addCategory\(\)](#)方法可以把一个Category放入到Intent中, [removeCategory\(\)](#)可以删除之前加入的Category, [getCategories\(\)](#)可以得到目前在Intent中所有的Category

Extras

Extras是传递给目标组件的键值对信息。就像一些action匹配着特别的data uri,一些action匹配着特别的Extras。例如ACTION_TIMEZONE_CHANGED匹配着"time-zone"指示新时区的信息, ACTION_HEADSET_PLUG匹配着"state"标识耳机设备是否插好的信息, 除此还有"name"显示耳机设备的类型。如果你是创造一个SHOW_COLOR的action, 颜色值将被设置在一个键-值对信息中。

Intent有一系列的[put..\(\)](#)方法用以向Extras中插入各种类型的值并且也有一系列的[get..\(\)](#)方法来取出数值。这些方法相对应的存在于[Bundle](#)类中。事实上, Extras也可以使用[putExtras\(\)](#)和[getExtras\(\)](#)来操作数据。

标志位-Flags

Flags有着很多种类。很多用来通知Android System如何运行一个activity (例如某个activity应该属于哪

个任务) 和运行以后如何处理 (例如, flag是否属于当前活动activity) 。所有这些flag都是在Intent中定义的。

Android system 和平台本身的应用会使用Intent发送系统本身的广播并且激活系统定义的组件。如何构造一个Intent并且激活一个系统组件, 请参考[list of intents](#)。

Intent解决方法-Intent Resolution

Intent可以被分为两组:

- **Explicit intents** 通过名字指定目标组件。由于组件的名字通常并不被其他应用的开发者所知道, explicit intents通常会用在应用内部的信息传递中, 例如一个activity调用一个附属的service或者是一个姊妹activity。
- **Implicit intents** 没有指明目标 (没有目标组件的名字) 。这种Intent通常是用来激活其他应用里的组件。

Android把explicit intent传递给指定的目标类。在Intent中没有什么比指定目标组件的名字更重要的了, 因为它决定了是哪个组件来接收intent。

对于implicit intent就要使用另一套策略了。在没有指定目标类的情况下, Android System必须找到最合适的组件去处理intent--单个的activity或者是service又或者是一套的broadcast receivers。为了实现以上, Android会把intent的内容拿来和与可能合适的组件相关联的intent过滤器和结构相比较。过滤器对外声明了组件的作用并且限定了组件能处理的intent的界限。intent过滤器使组件可以接受implicit intents成为了可能。如果一个组件没有intent过滤器, 那么他只能接收explicit intent。拥有intent过滤器的组件可以接收两种intent。

Intent过滤器要对intent的三个内容进行校验

action

data(both URI and data type)

category

Extra和Flag这一方面并不发挥作用。

Intent过滤器-Intent filters

activity, service 和 broadcast receiver为了通知系统他们可以处理哪些不为五隐含的intent，通常他们都有一个或是更多的intent过滤器。每个过滤器描述了组件的一个能力也限定一些组件可以接收的intent.实际上过滤器是筛选组件想要的intent，而不是筛选出不需要的intent--仅仅是不需要的隐含的 intent。显性的intent总是被传递给指定的目标，无论过滤器里设置了什么条件。

一个组件会把过滤器们分开，让他们各自去做自己的事情。例如， NoteEditor activity拥有两个过滤器，一个是为了启动用于查看和编辑的note,另一个是为了启动一个新的，空白的note用于填写和保存。（所有的Note Pad's过滤器在[记事本案例-Note Pad Example](#)都有描述）。

一个Intent过滤器就是[IntentFilter](#)类的一个实例。然而，由于Android System必须在加载组件之前知道这个组件的功能，所以Intent过滤器通常不在java代码里设置，而是在AndroidManifest.xml利用[`<intent-filter>`](#)来设置。（一个例外可能就是broadcast receivers的过滤器，他们是通过[Context.registerReceiver\(\)](#)动态注册的，他们作为IntentFilter对象被直接创建）。

在Intent中filter与action, data, category是平行关系。一个implicit

过滤器和安全-Filters and security

一个Intent过滤器的安全性不可靠。当它打开一个组件准备接收确定类型的隐含intents时，它并不能阻止目标组件中的显性intents。即使一个过滤器阻止了一个组件被要

intent要想通过intent filter必须要在action, data, category三个方面通过校验。如果其中一个没有通过, Android System就不会把intent传递给组件。然而,由于一个组件拥有很多的intent过滤器,所以只要通过其中一个就可以了。

求处理的确定动作和数据源,别人依然可以使用不同的动作和数据源绑定到一个显示的intent上,并将其命名为与目标组件相同的名字。

检验的细节如下描述:

Action test

manifest文件中在[<intent-filter>](#)元素下,列出了[<action>](#)子元素。例如:

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    .
    .
</intent-filter>
```

如范例所展示,当一个Intent对象只命名了一个动作,过滤器可能会列举多个。列表不能为空;一个过滤器至少需要包含一个[<action>](#)元素,否则它不会匹配任何intents。

为了通过这个测试,Intent对象中所指定的动作必须与过滤器中所列举的动作之一匹配。如果这个对象或过滤器没有指定一个动作,其结果如下:

- 如果过滤器没有列举任何动作,那么就没有动作能够用来与intent相匹配,所以,所有的intents都不能通过测试。没有intents可以通过过滤器。
- 另一方面,一个Intent对象没有指定任何动作,将自动通过测试——只要过滤器含有至少一个动作。

分类测试-Category test

一个[<intent-filter>](#)元素，同样也把分类当作子元素列举，如：

```
<intent-filter . . . >
    <category android:name= "android.intent.category.DEFAULT" />
    <category android:name= "android.intent.category.BROWSABLE" />
    .
    .
</intent-filter>
```

注意，上文中讨论的动作和分类常量并不适用于manifest文件。而是使用了完全字符串值。例如，上面范例中的"abdriud,ubtebt,category.BROWSABLE"字符串对应了前文中讨论到的CATEGORY_BROWSABLE常量。相似地，"android.intent.action.EDIT"字符串对应了ACTION_EDIT常量。

对于一个intent，想要通过分类测试，intent对象中的每一个分类都必须对应过滤器中的一个分类。过滤器可以列举额外的分类，但不能忽略intent中的任何分类。

因此，从理论上来说，不考虑过滤器中的值，一个没有分类的intent对象应该始终能通过这个测试。大部分情况下是对的。然而，有一个例外，Android把所有传递给[startActivity\(\)](#)的隐性intents都看作是至少有一个分类："android.intent.category.DEFAULT"（即CATEGORY_DEFAULT常量）。所以，想要接收隐性intents对象的活动必须在intent过滤器中包含"android.intent.category.DEFAULT"（过滤器中含有"android.intent.action.MAIN"和"android.intent.category.LAUNCHER"设定的为异常。它们标记了活动开始新任务，并显示在启动屏幕上。它们可以在分类列表中包括"android.intent.category.DEFAULT"，但不需要这么做。）见下文中[使用Intent匹配-Using intent matching](#)，更多关于这种过滤器的信息。

数据测试-Data test

如同动作与分类，intent过滤器中的数据指定也包含在了一个子元素当中。并且，在这种情况下，子元素可以多次出现，或不出现。例如：

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    .
    .
</intent-filter>
```

每一个[<data>](#)元素可以指定一个URI和一个数据类型（MIME媒体类型）。有多个属性——scheme，host，port和port——组成了URI的每一个部分：

scheme://host:port/path

例如，在以下的URI中，

content://com.example.project:200/folder/subfolder/etc

其中，scheme为“content”，host为“com.example.project”，port是“200”，path为“folder/subfolder/etc”。host与port一起，组成了URI权限，如果没有指定host，那么port也将被无视。

这些属性都是可选的，但它们之间并不相互独立：为了有意义的授权，scheme必须被指定。为了使路径（path）有意义，scheme和权限（authority）都必须被指定。

当Intent对象中的URI与过滤器中指定的URI对比时，只对比在过滤器中提及的部分。例如，如果过滤器只指定了数据类型，所有拥有这个scheme的URIs都能匹配。如果过滤器指定了一个scheme和权限（authority），但没有路径（path），所有拥有相同scheme和权限（authority）的URIs将匹配，不考虑其路径（path）。如果过滤器指定了scheme，权限（authority）和路径（path），那么，只有在URIs相同的这三个属性时才匹配。然而，过滤器中的路径指定可以包含通用符，来要求路径的部分匹配。

<data>元素中的type属性指定了数据的MIME类型。在过滤器中，它比URI更常见。Intent对象和过滤器都可以使用“*”通用符作为子类型域——如，“text/*”或“audio/*”——指定任意一个匹配的子类型。

数据测试同时对比Intent对象中和过滤器中所指定的URI和数据类型。规则如下：

- a.一个既不包含URI又没有指定数据类型的Intent对象只有在过滤器同样什么都没指定的情况下才能通过测试。
- b.任意一个只包含了URI但没有数据类型（并且无法从URI中推断出其类型）只有在这个URI与过滤器中的一个URI相匹配，并且同样没有指定数据类型的情况下通过测试。当URI为mailto:和tel:时即为这种情况，并不能确切的知道数据类开。
- c.Intent对象中包含了一个数据类型但没有URI时，只有在过滤器列举了相同的数据类型，并且类似有没有指定URI时通过测试。
- d.Intent对象同时包含了一个URI和一个数据类型（如数据类型可以由URI推断出）只有在该类型与过滤器中所列类型匹配时才能通过测试。它将通过URI部分的测试，要么URI与过滤器的中某个匹配，要么其包含了一个content:或file:URI并且过滤器没有指定一个URI。换句话说，如果过滤器中听指定了数据类型，那么一个组件默认的支持>content:和file:数据。

如果一个intent可以通过多个活动和服务的过滤器，那么用户将需要选择激活哪个组件。如果找不到目标，刚会抛出一个异常。

一般案例-Common cases

在上一个Data test提到的最后一条 (d) ，表达出一种期望：组件可以从文件或content provider中获得数据。因此，他们的过滤器只需列出数据的类型而不需要给content: schemes和file: schemes明确命名。下面是一个经典的例子。一个<data>元素，告诉Android，组件可以从content provider获得图片数据并且显示：

```
<data android:mimeType="image/*" />
```

因为大多数可用的数据是由content provider配与的，过滤器只指定数据类型而不指定URI可能成为最常见的。

另一种通用的配置是过滤器指定scheme和数据类型。例如，一个<data>元素，告诉Android，组件可以从网络获得视频数据并显示：

```
<data android:scheme="http" android:type="video/*" />
```

考虑一下当点击一个链接时，浏览器会做什么。首先，它会尝试去展示数据（因为他能够连接到网页）。如果它不能展示数据，它会把implicit intent与scheme和数据类型放在一起并且尝试启动一个可以完成任务的activity。如果没有合适的activity,它会要求下载器下载数据。把应用放在content provider的控制下，这样会有一个潜在的activities池响应调度。

大多数应用还可以重新启动，在不需要引入任何特殊的数据的情况下。初始化应用的Activities拥有action过滤器"android.intent.action.MAIN"。如果他们出现在应用的启动器中，那么他们还必须声明"android.intent.category.LAUNCHER"

```
<intent-filter . . . >
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

使用**intent**匹配-Using intent matching

Intents与intent filters匹配，不仅仅是为了找到激活的目标组件，而且也为了找到关于设备上的组件的相关信息。例如，Android系统填充应用程序启动器，顶层屏幕显示了用户可以启动的应用程序，通过寻找所有在intent filters中指定了"android.intent.action.MAIN"动作和"android.intent.category.LAUNCHER"分类（如前文中所展示的）的活动。然后将这些活动的图标和标签显示在启动器中。相似地，通过寻找filter中的"android.intent.category.HOME"来发现主屏幕上的应用。

你的应用程序可以使用intent匹配，也是通过类似的方法。[PackageManager](#)拥有一系列的[query...\(\)](#)方法，这些方法可以用来返回所有可以接收一个特定intent的组件，和相似的一系列[resolve...\(\)](#)方法用来决定最佳的组件来响应这个intent。例如，[queryIntentActivities\(\)](#)返回了一个所有能通过intent的活动的列表，[queryIntentService\(\)](#)类似地返回了一个服务列表。但这两个方法都不激活组件；他们只是列举了所有可以响应的组件。对于广播接收器也有类似的方法，[queryBroadcastReceivers](#)。

记事本案例-Note Pad Example

记事本案例（Note Pad Example）应用程序使用户能够浏览一个记事本列表，查看列表中的单独项目，编辑，和

添加新的项目到列表。本章节着重于manifest文档中intent filters的声明。（如果你处于离线，并使用SDK，你可以在`<sdk>/samples/NotePad/index.html`中找到关于这个范例应用程序，包括manifest文件的原代码。如果你是在线观看的本文档，所有源码文件在[Tutorials and Sample Code](#)中。）

在manifest文件中，记事本应用程序声明了三个活动，每一个至少含有一个intent filter。它同时还声明了一个内容提供者（content provider）来管理记事本数据。以下为manifest中的完整代码：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.notepad">
    <application android:icon="@drawable/app_notes"
        android:label="@string/app_name" >
        <provider android:name="NotePadProvider"
            android:authorities="com.google.provider.NotePad" />
        <activity android:name="NotesList" android:label="@string/title_notes_list" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="android.intent.action.EDIT" />
                <action android:name="android.intent.action.PICK" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.GET_CONTENT" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
            </intent-filter>
        </activity>
        <activity android:name="NoteEditor"
            android:theme="@android:style/Theme.Light"
            android:label="@string/title_note" >
            <intent-filter android:label="@string/resolve_edit">
```

```

<action android:name="android.intent.action.VIEW" />
<action android:name="android.intent.action.EDIT" />
<action android:name="com.android.notepad.action.EDIT_NOTE" />
<category android:name="android.intent.category.DEFAULT" />
<data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.INSERT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
</activity>

<activity android:name="TitleEditor"
          android:label="@string/title_edit_title"
          android:theme="@android:style/Theme.Dialog">
    <intent-filter android:label="@string/resolve_title">
        <action android:name="com.android.notepad.action.EDIT_TITLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.ALTERNATIVE" />
        <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
        <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
    </intent-filter>
</activity>

</application>
</manifest>

```

第一个活动，`NoteList`，通过其操作于一个记事本路径（记事本列表）而不是某个单独的笔记的实际区别于其他活动。通常情况下，它将作为用户的首选接口服务于应用程序之中。通过它定义的三个intent filters，它可以完成以下三件事：

```

<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>

```

这个过滤器申明了记事本应用程序的主接入点。标准的MAIN动作是一个不需要任何intent信息的接入点（例如，

不需要指定数据），**LAUNCHER**分类描述了这个接入点应该被列举在应用程序启动器之中。

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.PICK" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

这个过滤器申明活动可以在记事本路径下可做的事情。它允许用户查看并编辑这个路径（通过**VIEW**和**EDIT**动作），或是从路径中选取一个特定的笔记（通过**PICK**动作）。

<data>元素中的**mimeType**属性指定了这些动作所操作的数据类型。它指示了这个活动可以零个或多个项目上从一个内容提供者接收Cursor（**vnd.android.cursor.dir**），这个内容提供者拥有记事本的数据（**vnd.google.note**）。启动活动的Intent对象会包含一个**content:URI**指定了这种类型的活动应该打开的额外数据。

在过滤器中也含有**DEFAULT**分类。其存在的原因为[Context.startActivity\(\)](#)和[Activity.startActivityForResult\(\)](#)方法把所有的intents看作是包含了**DEFAULT**分类——只有两个例外：

- Intents明确的指出了目标活动
- 组成了**MAIN**动作和**LAUNCHER**分类的Intents

因此，**DEFAULT**分类是所有过滤器所要求的——除了那些包含了**MAIN**动作和**LAUNCHER**分类的过滤器。（在有明确的intents时，Intent过滤器并不被查寻）。

```
<intent-filter>
    <action android:name="android.intent.action.GET_CONTENT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

这个过滤器描述了活动可以返回一个用户所选择的笔记而不需要指定用户选择的笔记的路径。**GET_CONTENT**动作类似于**PICK**动作。在两个案例中，活动都返回了用户选择的笔记的URI。（在每一个案例中，它被返回给了活动并调用[startActivityForResult\(\)](#)来启动NoteList活动。）然而，这种情况下调用者指定了渴望的数据类型而不是用户将要取得数据的路径。

数据类型，**vnd.android.cursor.item/vnd.google.note**，指示了活动可以返回的数据类型——每个单独的笔记对应一个URI。通过返回的URI，调用者可以从握有笔记本数据的内容提供者（**vnd.google.note**）得到一个一一对应的Cursor。

换句话说，对于前文中提到的**PICK**动作，数据类型指示了活动能够像用户展示的数据类型。对于**GET_CONTENT**过滤器，它指示了活动可以返回给调用者的数据类型。

拥有了这些能力，以下的intents能够将其分解给NotesList活动：

动作：**android.intent.action.MAIN**

启动一个没有指明数据的活动

动作: `android.intent.action.MAIN`

分类: `<android.intent.category.LAUNCHER>`

启动一个没有数据被选择的活动。这个是启动器实际上用来填充其顶级列表的。所有在过滤器中匹配这个动作和分类的活动将被添加到这个列表中。

动作: `android.intent.action.VIEW`

数据: `content://com.google.provider.NotePad/notes`

要求活动在`content://com.google.provider.NotePad/notes`路径下的所有笔记的列表。用户可以从列表中选择一个笔记，并且该活动将把此项目的URI返回给启动NoteList活动的活动。

动作: `android.intent.action.GET_CONTENT`

数据类型: `vnd.android.cursor.item/vnd.google.note`

要求活动支持笔记本数据的单个项目。

第二个活动, NoteEditor, 向用户使展示了单独一个笔记的键入并允许其被编辑。如两个intent过滤器所描述的, 它可以做两个事情:

```
<intent-filter android:label="@string/resolve_edit">
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
```

```

<action android:name="com.android.notebook.action.EDIT_NOTE" />
<category android:name="android.intent.category.DEFAULT" />
<data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>

```

第一个，也是唯一的目的，这个activity是想让用户和记事本进行互动--用户既可以查看又可以编辑它（`EDIT_NOTE`是`EDIT`的同义词）。Intent将包含匹配类型的数据的URI`vnd.android.cursor.item/vnd.google.note`--他是指向一个单独的，具体的记事本的URI。他通常是被`PICK`或者`GET_CONTENT`返回的URI。

像以前一样，上面的过滤器列出了`DEFAULT`category,使activity能够被没有具体指明`NoteEditor`类的intent所调用

```

<intent-filter>
<action android:name="android.intent.action.INSERT" />
<category android:name="android.intent.category.DEFAULT" />
<data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>

```

这个activity的第二个目的是让用户创建一个新的记事本。Intent将包含匹配类型的数据的URI`vnd.android.cursor.dir/vnd.google.note`--他是指向记事本存在的路径的URI。

鉴于这些能力，下列intents将解决`NoteEditor` activity：

`action:android.intent.action.VIEW`

`data:content://com.google.provider.NotePad/notes/ID`

要求activity按照ID显示记事本的内容（关于细节：`content:` URIs是如何指定组中的个体，请看[Content Providers](#)）

`action: android.intent.action.INSERT`

data: content://com.google.provider.NotePad/notes

要求activity通过content://com.google.provider.NotePad/notes创建一个新的，空白的记事本并且允许用户去编辑，如果用户保存，那么它的URI将返回给调用者。

最后的activity，TitleEditor，让用户可以编辑记事本的题目。这个可以通过直接引入activity实现，而不需要通过intent过滤器。但是我们在这里展示如何how to publish alternative operations on existing data:

```
<intent-filter android:label="@string/resolve_title">
    <action android:name="com.android.notepad.action.EDIT_TITLE" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

这个intent过滤器使用了传统的action,"com.android.notepad.action.EDIT_TITLE"。他必须是由一个具体的记事本引用（数据类型是vnd.android.cursor.item/vnd.google.note）,就像是之前的View和EDIT。然而，这里的activity显示的是记事本的题目而不是内容。

除了支持常用的DEFAULT category,题目编辑器还支持两种标准的categories:ALTERNATIVE和SELECTED_ALTERNATIVE。这些categories指定了那些以菜单形式展示给用户的activities (很像LAUNCHER category 指定了应用启动器一样)。注意过滤器还要提供明确的标签 (通过android:label="@string/resolve_title") 以便能更好的控制用户所能看到的一切。 (关于这些categories和创建菜单项，请查看[PackageManager.queryIntentActivityOptions\(\)](#)和[Menu.addIntentOptions\(\)](#))

鉴于这些能力，下列intent将解决TitleEditor activity :

action: com.android.notepad.action.EDIT_TITLE

data: content://com.google.provider.NotePad/notes/ID

要求activity按照ID显示记事本的标题，并允许用户编辑标题。

来自“[index.php?title=Intents_and_Intent_Filters&oldid=13776](#)”



Processes and Threads

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html>

翻译：[Suchang1123](#)

更新： 2012.06.05

目录

[[隐藏](#)]

[1 android中进程与线程 - Processes and Threads](#)

- [1.1 进程 - Processes](#)
 - [1.1.1 进程的生命期](#)
- [1.2 线程 - Threads](#)
 - [1.2.1 工作线程 - Worker threads](#)
 - [1.2.2 使用-AsyncTask](#)
 - [1.2.3 线程安全的方法 - Thread-safe methods](#)
- [1.3 进程间通信 - Interprocess Communication](#)

android中进程与线程 - Processes and Threads

当一个应用程序开始运行它的第一个组件时，Android会为它启动一个Linux进程，并在其 中执行一个单一的线程。默认情况下，应用程序所有的组件均在这个进程的这个线程中运行。然而，你也可以安排组件在其他进程中运行，而且可以为任意进程衍生 出其它线程。

下面将介绍如何在android系统中使用线程和进程.

进程 - Processes

默认情况下，同一应用程序的所有组件运行在同一进程中。不过，如果你需要控制某个组件属于哪个进程，也可以通过修改manifest文件来实现。

manifest文件中的所有支持android:process属性的那些项，例如[\[活动\]](#) [\[activity\]](#), [\[服务\]](#) [\[service\]](#), [\[接收者\]](#), 和[\[内容提供\]](#) [\[provider\]](#) 都可以指定一个进程，这样这些组件就会在指定的进程中运行。你可以设置这个属性使每个组件运行于其自己的进程或只是其中一些组件共享一个进程。你也可以设置android:process以使不同应用的组件们可以运行于同一个进程—这样需要这些应用共享同一个用户ID并且有相同的数字证书。

[\[application\]](#)元素也支持android:process属性,用于为所有的组件指定一个默认值.

Android系统可能在某些时刻决定关闭一个进程，比如内存很少了并且另一个进程更迫切的需要启动时。进程被关闭时，其中的组件们都已被销毁。如果重新需要这些组件工作时，进程又会被创建出来。

当决定关闭哪些进程时，Android系统会衡量进程们与用户的紧密程度。例如，比起一个具有可见的 [activity](#)的进程，那些所含activity全部不可见的进程更容易被关闭。如何决定一个进程是否被关闭，取决于进程中运行的组件们的状态。决定关闭进程的规则将在下面讨论。

进程的生命期

Android系统会尽量维持一个进程的生命，直到最终需要为新的更重要的进程腾出内存空间。为了决定哪个进程该终止，系统会根据运行于进程内的组件的和组件的状态把进程置于不同的重要性等级。当需要系统资源时，重要性等级越低的先被淘汰。

重要性等级被分为 5 个档。下面列出了不同类型的进程的重要性等级(第一个进程类型是最重要的，也是最后才会被终止的)

1 前台进程

用户当前正在做的事情需要这个进程。如果满足下面的条件，一个进程就被认为是前台进程：

1)这个进程拥有一个正在与用户交互的Activity(这个Activity的onResume() 方法被调用)。

2)这个进程拥有一个绑定到正在与用户交互的activity上的Service。

3)这个进程拥有一个前台运行的Service — service调用了方法startForeground()。

4)这个进程拥有一个正在执行其任何一个生命周期回调方法(onCreate(), onStart(), 或onDestroy()) 的Service。

5)这个进程拥有正在执行其onReceive()方法的BroadcastReceiver。

通常，在任何时间点，只有很少的前台进程存在。它们只有在达到无法调合的矛盾时才会被终止——如果内存太小而不能继续运行时。通常，到了这时，设备

就达到了一个内存分页调度状态，所以需要终止一些前台进程来保证用户界面的反应。

2 可见进程

一个进程不拥有运行于前台的组件，但是依然能影响用户所见。满足下列条件时，进程即为可见：

1)这个进程拥有一个不在前台但仍可见的Activity(它的onPause()方法被调用)。例如当一个前台activity启动一个对话框时，就出了这种情况。

2)这个进程拥有一个绑定在前台(或者可见)Activity的服务。一个可见的进程是极其重要的，通常不会被终止，除非内存不够，需要释放内存以便前台进程运行。

3 服务进程

一个进程不在上述两种之内，但它运行着一个被startService()所启动的service。

尽管一个服务进程不直接影响用户所见，但是它们通常做一些用户关心的事情(比如播放音乐或下载数据)，所以除非系统没有足够的空间运行前台进程和可见进程时才会终止一个服务进程。

4 后台进程

一个进程拥有一个当前不可见的activity(activity的onStop()方法被调用)。

这样的进程们不会直接影响到用户体验，所以系统可以在任意时刻杀了它们从而为前台、可见、以及服务进程们提供存储空间。通常有很多后台进程在运行。它们被保存在一个LRU(最近最少使用)列表中来确保拥有最近刚被看到的activity的进程最后被杀。如果一个activity正确的实现了它的生命周期方法，并保存了它的当前状态，那么杀死它的进程将不会对用户的可视化体验造成影响。因为当用户返回到这个activity时，这个activity会恢复它所有的可见状态。

5 空进程

一个没有任何active组件的进程。

保留这类进程的唯一理由是高速缓存，这样可以提高下一次一个组件要运行它时的启动速度。系统经常为了平衡在进程高速缓存和底层的内核高速缓存之间的整体系统资源而终止它们。

根据进程中当前活动的组件的重要性，Android会把进程按排在其可能的最高级别。例如，如果一个进程拥有一个service和一个可见的activity，进程会被定为可见进程，而不是服务进程。

另外，如果被其它进程所依赖，一个进程的级别可能会被提高——一个服务于其它进程的进程，其级别不可能比被服务进程低。因为拥有一个service的进程比拥有一个后台activitie的进程级别高，所以当一个activity启动一个需长时间执行的操作时，最好是启动一个服务，而不是简单的创建一个工作线程。尤其是当这个操作可能比activity的生命还要长时。例如，一个向网站上传图片的activity，应

该启动一个service，从而使上传操作可以在用户离开这个activity时继续在后台执行。使用一个service保证了这个操作至少是在"服务进程"级别，而不管activity是否发生了什么不幸。这同样是广播接收者应该使用service而不是简单地使用一个线程的理由。

线程 - Threads

当一个应用被启动，系统创建一个执行线程，叫做"main"。这个线程是十分重要的，因为它主管向用户界面控件派发事件。其中包含绘图事件。它也是你的应用与界面工具包（`android.widget`和`android.view`包中的组件）交互的地方。于是main线程也被称为界面线程。

系统不会为每个组件的实例分别创建线程。所有运行于一个进程的组件都在界面线程中被实例化，并且系统对每个组件的调用都在这个线程中派发。因此，响应系统调用的方法(比如报告用户动作的`onKeyDown()`或一个生命周期回调方法)永远在界面线程中进行。

例如，当用户触摸屏幕上的一个按钮时，你的应用的界面线程把触摸事件派发给控件，然后控件设置它的按下状态再向事件队列发出一个自己界面变得无效的请求，界面线程从队列中取出这个请求并通知这个控件重绘它自己。

当你的应用在响应用户交互时需执行大量运算时，这种单线程的模式会带来低性能，除非你能正确的优化你的程序。如果所有事情都在界面线程中发生，执行比如网络连接或数据库请求这样的耗时操作，将会阻止整个界面的响应。当线程被阻塞时，就不能派发事件了，包括绘图事件。从用户的角度看，程序反应太慢了。甚至更糟的是，如果界面线程被阻塞几秒钟（5秒左右），用户就抱怨说程序没反应了，用户可能会退出并删掉你的应用。

此外，Android界面不是线程安全的。所以你绝不能在一个工作线程中操作你的界面—你只能在界面线程中管理你的界面。所以，对于单线程模式有两个简单的规则：

1. 不要阻塞界面线程
2. 不要在界面线程之外操作界面。

工作线程 - Worker threads

由于上述的单线程模式，不要阻塞你的界面线程以使你的应用的界面保持响应是非常重要的，那么如果你有不能很快完成的任务，你应把它们放在另一个线程中执行(后台线程或工作线程)。

例如，下面是的代码是响应click事件，在另外一个线程中下载一个图片并在一个ImageView中显示它：

```

public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b =
loadImageFromNetwork(<nowiki>"http://example.com/image.png"</nowiki>);
            mImageView.setImageBitmap(b);
        }
    }).start();
}

```

粗略看来，这样能很好的工作，因为它创建了一个新线程来进行网络操作。然而它违反了第二条规则：不要在界面线程之外操作界面。这段代码在工作线程中修改了ImageView。这会导至未定义的异常出现，并且难以调试追踪。

为了能改正这个问题，Android提供了很多从其它线程来操作界面的方法。下面是可用的方法：

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)
- View.postDelayed(Runnable, long)

例如，你可以用View.post(Runnable)来修正上面的问题：

```

public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
loadImageFromNetwork(<nowiki>"http://example.com/image.png"</nowiki>);
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}

```

现在这个实现终于是线程安全的了：网络操作在另一个线程中并且ImageView在界面线程中改变。

然而，由于操作复杂性的增长，这样的代码就变得复杂并难以维护，为了处理更复杂的交互，你可能需要在线程中用到Handler对象来处理从UI线程中传递的消息。但最好的解决办法是继承AsyncTask类，这样可以使得线程和UI交互这一类任务变得更轻松。

使用-**AsyncTask**

AsyncTask可以让程序进行异步工作，它在一个线程中执行某些操作，之后将结果返回给UI线程。

使用AsyncTask类时，你需要继承AsyncTask类并实现doInBackground()回调方法。要更新UI界面，需要实现onPostExecute()，并从doInBackground()方法中获得结果，最后，你可以在UI线程中调用execute()方法来执行操作，这样就可以安全的更新UI界面。

例如，我们用AsynTask来实现上面的示例：

```
public void onClick(View v) {
    new DownloadImageTask().execute(<nowiki>"http://example.com/image.png" </nowiki>);
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** 该方法运行在后台线程中。参数来自AsyncTask.execute()方法，  
     * 这里将主要负责执行那些很耗时的后台处理工作。 */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** 后台的计算结果将通过该方法传递到UI 线程，并且在界面上展示给用户 */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

因为AsyncTask将工作分成了两部分，UI线程和工作线程都做自己应该做的那部分任务，因此我们就可以简单的实现一个安全的UI更新任务了。

更多关于AsyncTask的说明请参看AsyncTask类的参考，这里简单介绍下它是如何工作的：

- AsyncTask定义了三种泛型类型 Params, Progress和Result。Params是启动任务执行的输入参数，比如HTTP请求的URL。Progress是后台任务执行的百分比。Result是后台执行任务最终返回的结果，比如String, Integer等。
- doInBackground()会在工作线程中自动执行。
- onPreExecute(), onPostExecute(), and onProgressUpdate()这三个方法都是在UI线程中调用。
- doInBackground()方法返回的值会传递给onPostExecute()方法。
- 你可以在doInBackground()方法中随时调用publishProgress()方法以此在UI线程中更新执行进度。
- 你可以随时在任何线程中取消任务。

Caution: Another problem you might encounter when using a worker thread is unexpected restarts in your activity due to a runtime configuration change (such as when the user changes the screen orientation), which may destroy your worker thread. To see how you can persist your task during one of these restarts and how to properly cancel the task when the activity is destroyed, see the source code for the Shelves sample application.

线程安全的方法 - **Thread-safe methods**

在某些情况下，你实现的方法可能会被多个线程调用，因此要保证该方法是线程安全的。

一些远程调用的方法——例如绑定服务。当我们在同一个进程中调用了某一个正在运行的**IBinder**中的 方法，这个方法会运行在调用者的线程中。然而，当调用另一个进程中的方法时，该方法会在系统为这个进程开启的线程池中的某一个线程中执行。例如，绑定服务要在UI线程中调用，**onBind()**返回的对象会在线程池中被调用。因为一个服务可以有多个客户端，多个池线程可以在同一时间进行相同的**IBinder** 方法。因此，实现**IBinder**方法必须是线程安全的。

同样内容提供者可以接收其他进程的数据请求。虽然**ContentResolver**和**ContentProvider**类隐藏了进程间通信管理的细节，响应请求的**ContentProvider**方法--**query()**, **insert()**, **delete()**, **update()**, 和**getType()**--从内容提供者进程的线程中调用，而不是UI线程。因为这些方法可以从任意数量线程同时调用，他们也必须安全实施。

进程间通信 - **Interprocess Communication**

Android提供了进程间通信 (IPC) 机制，使用远程过程调用 (RPC) ，其中一个方法被称为活动(Activity)或其他应用程序组件，但任何返回给调用者的结果都在另一个进程中执行。这就需要在系统级别从本地进程和远程进程的地址空间中调用方法和参数，并将返回值按相同的方法传递回来。android系统能完美支持进程间通信(IPC)，这样你就可以专注于制定和实现RPC编程接口。

要实现进程间通信(IPC)，应用程序必须绑定到一个服务，使用**bindService ()**。欲了解更多信息，请参阅服务(Service)开发指南。

来自 "[index.php?title=Processes_and_Threads&oldid=13851](#)"

1个分类: [Android Dev Guide](#)



Permissions

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/topics/security/permissions.html>

作者：smilysas

更新时间：2012年7月11日

目录

[[隐藏](#)]

[1 权限许可-Permissions](#)

- [1.1 安全体系架构-Security Architecture](#)
- [1.2 应用程序数字签名-Application Signing](#)
- [1.3 用户ID和文件访问-User IDs and File Access](#)
- [1.4 使用权限-Using Permissions](#)
- [1.5 声明和实施许可-Declaring and Enforcing Permissions](#)
 - [1.5.1 在AndroidManifest.xml中声明权限设置-Enforcing Permissions in AndroidManifest.xml](#)
 - [1.5.2 发送广播时实施许可-Enforcing Permissions when Sending Broadcasts](#)
 - [1.5.3 其他权限设置-Other Permission Enforcement](#)
- [1.6 URI许可-URI Permissions](#)

权限许可-Permissions

本文档介绍了应用程序开发人员如何使用由Android提供的安全功能。
在Android 开放源代码项目AOSP(Android Open Source Project)中提供了更

一般的Android安全性概述。

Android是一种特权分隔的操作系统，在Android上运行的每个应用程序都具有各自独立的系统标识（Linux用户ID和组ID）。系统各部分有不同的身份标识。因此，Linux上运行的各个应用程序相互独立且与系统无关。

Android的“权限许可”机制通过限定特定的进程能够执行的指定操作和限定对每一个资源点对点的访问的URI许可来提供附加细粒度的安全功能

安全体系架构-Security Architecture

Android安全体系架构设计的核心是在默认情况下没有任何一个程序可以执行对其他程序、操作系统或者用户有害的操作，包括读写用户的隐私数据（例如联系人或者电子邮件），读写其他程序的文件，进行网络访问或者唤醒设备等等。

由于内核让每个应用程序运行在独立的沙盒中，应用程序必须通过声明所需要而沙盒没有提供的权限来明确的分配资源和数据。Android没有采用会使用户体验复杂并且不利于安全的动态授权机制。应用程序静态的声明他们所需要的权限，在程序安装时Android系统会提示用户同意它们获取这些权限。

沙盒程序独立于生成普通应用程序的机制。特别地，Dalvik虚拟机不是一个安全的边界，任何的应用程序都能够运行本地代码（参照Android NDK）。所有类型的应用程序——java、native和混合的——均用相同的方式置以相同的安全等级在沙盒中运行。

应用程序数字签名-Application Signing

所有的Android应用程序（apk文件）都必须使用一个开发人员掌握私钥、用于识别应用程序作者的证书进行签名。该证书要求很宽松，并不需要由专门的证书颁发机构进行签名，Android应用程序可以使用自签名的证书。Android证书的目的是区分应用程序的作者，可以允许操作系统授予或者拒绝应用程序使用签名级别的权限和操作系统授予或者拒绝应用程序请求和其他应用程序相同的Linux身份。

用户ID和文件访问-User IDs and File Access

在安装的时候，Android会给每个程序分配一个不同的Linux用户身份(**UID**)。软件在设备上的生命周期中这个身份标识保持恒定不变。在不同的设备上，相同的软件可能会有一个不同的**UID**；重要的是在给定的设备上不同的包是不同的**UID**。

因为安全是在进程级别上实现的，两个软件包的代码在同一个进程中不能够同时正常运行，他们必须以不同的Linux用户运行。可以在每个程序包的**AndroidManifest.xml**中将**manifest**标签的**shareUserId**属性分配相同的用户**ID**，把两个应用程序看作拥有同样的用户**ID**和文件权限的同一个应用程序。为了保持安全，只有具有相同签名（请求的**sharedUserId**也相同）的应用程序才会分配相同的用户**ID**。

任何由应用程序存储的数据将被赋予应用程序的用户**ID**，正常情况不能被其它应用程序访问。当使用 `getSharedPreferences(String, int)`, `openFileOutput(String, int)`, 或 `openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)` 创建一个新的文件时，可以使 `MODE_WORLD_READABLE` 或 `MODE_WORLD_WRITEABLE` 标记允许其他应用程序来读/写文件。设置这些全局的读写权限标记后，该文件仍然为创建文件的应用程序所拥有，任何其他应用程序可以看到它。

使用权限-Using Permissions

一个Android应用程序没有任何权限，这意味着它不能做任何会对用户体验或设备上的任何数据造成不利影响的操作。要利用设备的保护功能，必须在 **AndroidManifest.xml** 文件中的一个或多个**<uses-permission>** 标签上声明应用程序所需要的权限。

例如，一个监控接收短信的应用程序需要作如下声明：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <package android:name="com.android.app.myapp" >
        <uses-permission android:name="android.permission.RECEIVE_SMS" />
        ...
    </manifest>
```

在安装应用程序时，安装者要基于应用程序的签名在交互时对应用程序所需的权限进行授权。应用程序运行时不会进行权限检查：它要么在安装后

被给予一个特殊的权限，并且可以使用它期望的权限，要么就不被授予权限，任何使用这些权限的操作都会在没有用户提示的情况下自动失败。

通常如果请求权限失败应用程序会抛出一个**SecurityException**异常，但是也有特例。例如，**sendBroadcast(Intent)**函数在所有数据被投递到接收者时检查权限，当函数返回后，不会对数据的权限进行检查，也不能接收到任何权限异常。约大多数情况下，权限异常会记录在日志中。

所有Android系统提供的权限可以在**Manifest.permission**中找到。任何应用程序也可以定义并执行其自己的权限，所以本文不是一个对于所有可能的权限的全面的清单。

在程序执行的任何阶段程序都可以定义并执行其自己的权限：

- 当调用系统调用时，阻止应用程序执行特定的功能。
- 当启动一个Activity时，阻止应用程序启动其他应用程序的Activity。
- 在发送或接受广播时，控制谁可以接受你的广播或者谁可以向你发送广播。
- 谁可以访问或操作一个特定的内容提供者。
- 绑定或者启动一项服务。

声明和实施许可-Declaring and Enforcing Permissions

为了执行你自己的权限，你必须首先在你的**AndroidManifest.xml**中使用一个或多个**<permission>**标签声明它们。

例如，一个应用程序想要控制谁能够开始它的一个Activity，可以声明如下：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >
    <permission
        android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>

```

<protectionLevel>属性是必选的，它告诉系统当其它应用程序需要该权限时怎样通知用户或者谁可以使用该权限。

<permissionGroup>属性是可选的，用于帮助系统展示权限给用户。通常会将它设置为在 `android.Manifest.permission_group` 中的一个标准的系统组，在极少数情况下也可以由自己进行定义。最好是优先使用现有的组，便于简化权限UI展示给用户。

请注意，这两个标签和描述应提供许可。用户在查看的权限列表(`android:label`)或单个权限(`android:description`)的细节时，这些内容会被展现。标签应该简洁的介绍权限保护的关键功能。用几个简单的句子描述拥有该权限可以做什么。惯例是用两个句子，第一句描述权限，第二句警告用户当授权该权限后会发生什么。

这里是一个`CALL_PHONE`权限的标签和描述的例子：

```
<string name="permlab_callPhone">directly call phone numbers
</string> <string name="permdesc_callPhone">Allows the
application to call phone numbers without your intervention.
Malicious applications may cause unexpected calls on your phone
bill. Note that this does not allow the application to call
emergency numbers.</string>
```

可以通过`shell`命令 `adb shell pm list permissions`来查看现在系统上的权限定义。特别地，`-s`选项可以用简单的表格形式来给用户呈现权限。

```
$ adb shell pm list permissions -s
All Permissions:
Network communication: view Wi-Fi state, create Bluetooth
connections, full Internet access, view network state
Your location: access extra location provider commands, fine (GPS)
location,
mock location sources for testing, coarse (network-based) location
Services that cost you money: send SMS messages, directly call phone
numbers
...
```

在 `AndroidManifest.xml` 中声明权限设置 - **Enforcing Permissions in `AndroidManifest.xml`**

进入系统或应用程序的组件的高级别权限可以在AndroidManifest.xml中实现。所有这些都可以通过在相应的组件中包含 android:permission 属性，以使其被用以控制进入的权限。

Activity权限（应用于<activity>标签）用于限制谁才可以启动相关的Activity。该权限在运行 Context.startActivity() 和 Activity.startActivityForResult() 期间被检查；如果调用方不具有相应必需的权限，那么将会从此次调用中抛出 SecurityException 异常。

Service权限（应用于<service>标签）用于限制谁才可以启动或绑定该service。在运行 Context.startService()，Context.stopService() 和 Context.bindService() 调用的时候会进行权限检查。如果调用方没有所需的权限，则会抛出一个 SecurityException 异常。

BroadcastReceiver许可（应用于<receiver>标签）用于限制谁可以向相关的接收器发送广播。权限检查会在 Context.sendBroadcast() 返回后当系统去发送已经提交的广播给相应的 Receiver 时进行。最终，一个 permission failure 不会再返回给调用方一个异常，只是不会去实现该 Intent 而已。同样地， Context.registerReceiver() 也可以通过自己的 permission 用于限制谁可以向一个在程序中注册的 receiver 发送广播。另一种方式是，一个 permission 也可以提供给 Context.sendBroadcast() 用以限制哪一个 BroadcastReceiver 才可以接收该广播。（见下文）

ContentProvider许可（应用于<provider>标签）用于限制谁才可以访问 ContentProvider 提供的数据。（Content providers 有一套额外的安全机制叫做 URI permissions，这些在稍后讨论。）不像其他组件，它有两个单独的权限属性，你可以设置： android:readPermission 用于限制谁能够读， android:writePermission 用于限制谁能够写。需要注意的是如果 provider 同时需要读写许可，只有写许可的情况下并不能读取 provider 中的数据。当你第一次检索内容提供者和当完成相关操作时会进行权限检查。（假如没有任何权限则会抛出 SecurityException 异常。）使用 ContentResolver.query() 需要持有读权限；使用 ContentResolver.insert()，ContentResolver.update()，ContentResolver.delete() 需要写权限。在所有这些情况下，没有所需的权限将会导致抛出 SecurityException 异常。

发送广播时实施许可-**Enforcing Permissions when Sending Broadcasts**

除了之前说过的权限（用于限制谁可以发送广播给相应的BroadcastReceiver），还可以在发送广播的时候指定一个许可。在调用Context.sendBroadcast()的时候使用一个permission string，你就可以要求接收器的宿主程序必须有相应的权限。

值得注意的是接收器和广播都可以要求许可。当这种情况发生时，这两种permission检查都需要通过后才会将相应的intent发送给相关的目的地。

其他权限设置-**Other Permission Enforcement**

在调用service的过程中可以设置更加细化的许可。这是通过Context.checkCallingPermission()方法来完成的。调用的时候使用一个想得到的permission string，返回给调用方一个整数判断是否具有相关权限。需要注意的是这种情况只能发生在来自另一个进程的调用，通常是一个service发布的IDL 接口或者是其他方式提供给其他的进程。

Android提供了很多其他有效的方法用于检查许可。如果有另一个进程的pid，可以通过 Context.checkPermission(String, int, int)去检查该进程的权限设置。如果有另一个应用程序的包名，可以直接用PackageManager.checkPermission(String, String)来确定该包是否已经拥有了相应的权限。

URI许可-**URI Permissions**

到目前为止我们讨论的标准的permission系统对于内容提供者 (content provider) 来说是不够的。一个内容提供者可能想保护它的读写权限，而同时与它对应的直属客户端也需要将特定的URI传递给其它应用程序以便对该URI进行操作。一个典型的例子是邮件应用程序的附件。访问邮件需要使用permission来保护，因为这些是敏感的用户数据。然而，如果有一个指向图片附件的URI需要传递给图片浏览器，那个图片浏览器是不会有访问附件的权利的，因为它不可能拥有所有的邮件的访问权限。

针对这个问题的解决方案就是per-URI permission：当启动一个activity或者给一个activity返回结果的时候，调用方可以设置Intent.FLAG_GRANT_READ_URI_PERMISSION和/或Intent.FLAG_GRANT_WRITE_URI_PERMISSION。这赋予接收活动

(activity) 访问该意图 (Intent) 指定的 URI的权限，而不论它是否有权限进入该意图对应的内容提供者。

这种机制允许一个通常的能力-风格 (capability-style) 模型，以用户交互（如打开一个附件，从列表中选择一个联系人）来驱动细化的特别授权。这是实现减少应用程序所需要的权限而只留下和程序行为直接相关的权限时很关键的一步。

这些URI permission的获取需要内容提供者（包含那些URI）的配合。强烈建议在内容提供者中实现这种功能，并通过`android:grantUriPermissions`或者`<grant-uri-permissions>`标签来声明支持。

更多的信息可以参考`Context.grantUriPermission()`,
`Context.revokeUriPermission()`和`Context.checkUriPermission()`函数。

来自 "[index.php?title=Permissions&oldid=6119](#)"



App Widgets

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： `◇Aduil ...

原文地址

址：<http://developer.android.com/guide/topics/appwidgets/index.html>

目录

[[隐藏](#)]

[1 微小的应用程序视图-App Widgets](#)

- [1.1 基础知识-The Basics](#)
- [1.2 Declaring an App Widget in the Manifest](#)
- [1.3 Adding the AppWidgetProviderInfo Metadata](#)
- [1.4 Creating the App Widget Layout](#)
 - [1.4.1 Adding margins to App Widgets](#)
- [1.5 Using the AppWidgetProvider Class](#)
 - [1.5.1 Receiving App Widget broadcast Intents](#)
- [1.6 Creating an App Widget Configuration Activity](#)
 - [1.6.1 Updating the App Widget from the configuration Activity](#)
- [1.7 Setting a Preview Image](#)
- [1.8 Using App Widgets with Collections](#)
 - [1.8.1 Sample application](#)
 - [1.8.2 Implementing app widgets with collections](#)
 - [1.8.3 Manifest for app widgets with collections](#)
 - [1.8.4 Layout for app widgets with collections](#)
 - [1.8.5 AppWidgetProvider class for app widgets with collections](#)
 - [1.8.6 RemoteViewsService class](#)

[1.8.7 RemoteViewsFactory interface](#)

- [1.8.8 Adding behavior to individual items](#)
- [1.8.9 Setting up the pending intent template](#)
- [1.8.10 Setting the fill-in Intent](#)
- [1.8.11 Keeping Collection Data Fresh](#)

微小的应用程序视图-App Widgets

App Widgets是一个应用程序的小视图，可以嵌入到其他应用程序

(如主屏幕)并且能够定期更新。你可以发布一个应用程序的App Widget，而这些视图称为窗口的用户界面。一个应用程序里，可以支持其他应用程序的App Widgets称为App Widget的主机。下面的截图是显示音乐的App Widget。



该文档将介绍如何在应用程序里发布使用App Widget。

基础知识-The Basics

要创建一个App Widget，您需要了解以下几点：

[AppWidgetProviderInfo](#) 对象：

描述了一个App Widget的元数据，如在App Widget的布局，更新频率，和AppWidgetProvider类。都应

快速浏览

- App Widgets提供一些用户直接从主屏幕访问您的应用程序功能(而不需要启动一个Activity)
- App Widgets支持了一种特殊的广播接收机处理AppWidget生命周期

在本文档

[基础知识-The Basics](#)

[Declaring an App Widget in the Manifest](#)

[Adding the AppWidgetProviderInfo Metadata](#)

[Creating the App Widget Layout Using the AppWidgetProvider Class](#)

[Receiving App Widget broadcast Intents](#)

[Creating an App Widget Configuration Activity](#)

在XML中定义。

[AppWidgetProvider](#)类的实现：

定义一个基于广播事件与App Widget的接口方法。通过它，您将收到广播对App Widget进行更新，启用，禁用和删除。

视图布局：

在XML中初步定义App Widget布局。

此外，还可以实现App Widget可配置的[Activity](#)。当用户添加您的App Widget，并允许他或她在创建时修改设置时启动这个可配置的Activity。

以下各节描述如何设置这些组件。

[Declaring an App Widget in the Manifest](#)

首先，在您的应用程序的[AndroidManifest.xml](#)文件中声明[AppWidgetProvider](#)类。例如：

```
<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>元素需要的android: name属性，在App Widget中指定使用[AppWidgetProvider](#)。intent-filter元素必须包含一个与<action>元素的android: name属性。此属性指定[AppWidgetProvider](#)接

[Updating the App Widget from the configuration Activity](#)

[Setting a Preview Image Using App Widgets with Collections](#)

[Sample application](#)
[Implementing app widgets with collections](#)
[Keeping Collection Data Fresh](#)

关键类

[AppWidgetProvider](#)
[AppWidgetProviderInfo](#)
[AppWidgetManager](#)

参见

[App Widget Design Guidelines](#)
[Introducing home screen widgets and the AppWidget framework](#)

受ACTION_APPWIDGET_UPDATE广播。这是唯一的广播，你必须明确声明。在AppWidgetManager自动发送其他App widget广播AppWidgetProvider是必要的。<meta-data>元素指定的AppWidgetProviderInfo的资源，需要以下属性：

- android: name - 指定的元数据的名称。使用`android.appwidget.provider`识别作为AppWidgetProviderInfo描述符的数据。
- android:resource - 指定AppWidgetProviderInfo的资源位置。

Adding the AppWidgetProviderInfo Metadata

AppWidgetProviderInfo定义App Widget的基本素质，例如其最小的布局尺寸，其初始布局资源，如何经常更新App Widget，和（可选）配置Activity，在创建时发起。在XML资源文件中定义AppWidgetProviderInfo对象，使用`<appwidget-provider>`元素和在项目的RES / XML/文件夹中保存。

例如：

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

以下是`<appwidget-provider>`属性的摘要：

- App Widget默认的情况下`minWidth`和`MinHeight`属性值指定消耗最小的空间，AppWidgets默认主页的屏幕位置，在其窗口基础上的单元格有一个明确高度和宽度的网格。如果一个App Widget的最小宽度或高度值不匹配单元格的尺寸，则App Widget尺寸四舍五入到最接近的单元格大小。

在App Widget Design Guidelines里了解更多关于App Widget的窗口大小。

注：为了使App Widget更容易在跨设备中携带，App Widget的最小尺寸不应大于4×4单元格

- **minResizeWidth**和**minResizeHeight**属性指定App Widget的绝对最小尺寸。这些值应该指定尺寸下，否则应用程序组件将无法辨认或以其他方式使用。在android3.1，允许用户使用这些属性调整控件大小，可能是小于默认尺寸界定的**minWidth**和**MinHeight**属性。
- 在[App Widget Design Guidelines](#)里了解更多关于App Widget的窗口大小。

updatePeriodMillis属性定义，往往在App Widget框架，应从[Appwidgetprovider](#)请求通过调用**onUpdate()**回调方法更新。实际上不能保证更新的准确性，我们建议尽可能的少更新或者不超过每小时一次，以此来节省电池。你也可以在 **configuration-some**中允许用户调整频率，比如证券报价机，一些用户可能想要15分钟更新一次，一些则想要每天只更新四次

注：如果该设备是睡着的，而它是一个更新的时间（定义**updatePeriodMillis**）时，则该设备将被唤醒以执行更新。如果你不超过每小时更新一次，这可能不会对电池寿命造成重大的问题。但是，如果您需要频繁更新或你并不需要更新，而设备是睡着的，你就可以根据报警代替执行，则不会唤醒设备执行更新。要做到这一点，设置一个Intent，当您的AppWidgetProvider收到的报警时，使用[AlarmManager](#)。设置报警类型有[ELAPSED_REALTIME](#)或RTC，这在收到报警时，该设备被唤醒。然后设置**updatePeriodMillis**为零（"0"）。

- **initialLayout**的属性指向布局资源，它定义了App Widget的布局。
- 在[Activity](#)启动时对**configure**属性进行配置定义，用户添加App Widget，以便他或她配置App Widget属性。这是可选的（阅读下面创建一个App Widget配置的Activity）。
- 在配置**previewImage**属性后将指定一个App Widget图标是什么样子，当选择这个App Widget时用户可以进行预览。如果没有提供图标，用户却认为launcher是您的应用程序图标。这个字段对应**android:previewImage**进行在**<receiver>**元素的**AndroidManifest.xml**文件中。在android3.0有相

关介绍中。更多的讨论使用`previewImage`，请参阅设置预览图标。

- 在Android 3.0的介绍，该`autoAdvanceViewId`属性指定的App Widget子视图的视图ID，应该是auto-advanced部件的主机。
- 在Android 3.1的介绍，该`resizeMode`属性指定其中一个可以调整规则的部件。您可以使用此属性使主屏幕小部件的`resizeMode`，水平，垂直，或两轴。用户`touch-hold`一个小工具，以显示其调整手柄，然后拖动水平和/或垂直手柄，改变布局网格的大小。`resizemode`属性值包括`"horizontal"`, `"vertical"`, 和`"none"`。声明一个部件作为`resizeable`横向、纵向、以及“横向|垂直”的值。

参考`AppWidgetProviderInfo`类查看更多信息，关于被`AppWidgetProviderInfo`类的`<appwidget-provider>`元素接收的标示。

Creating the App Widget Layout

必须为你的App widget定义初始布局的XML和将其保存在项目的`res/layout/`目录中。使用下列的视图对象可以设计你的App widget，但在你开始设计你的App widget之前，请阅读和理解App widget的设计准则。如果你熟悉XML的布局，创建App widget的布局很简单。然而，你们必须知道App widget的布局都是基于`RemoteViews`，不支持各种布局或视图控件。一个`RemoteViews`对象(and, consequently, an App Widget)可以支持以下布局类：

`FrameLayout`
`LinearLayout`
`RelativeLayout`

及以下的小部件类：

`AnalogClock`
`Button`
`Chronometer`
`ImageButton`
`ImageView`
`ProgressBar`
`TextView`
`ViewFlipper`

ListView
GridView
StackView
AdapterViewFlipper

这些类的子类都不支持的。

Adding margins to App Widgets

widget通常不应该扩展到屏幕边缘,不应与其他widget视图共同刷新,所以你应该增加边缘围绕你的widget框架。作为Android 4.0,App widget提供了小部件之间的自动填充框架和App widget的包围盒,以便用户在主屏幕更好的调整其他小部件和图标。使用这种推荐应用程序的targetSdkVersion设置14或更高。

它很容易编写一个布局,适用于早期版本的平台,具有自定义边距,在Android4.0并没有额外的边距或者更高:

1. 设置应用程序的targetsdkversion值为14或更高。
2. 创建一个如下布局,引用其他尺寸资源:

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/widget_margin">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:background="@drawable/my_widget_background">
        ...
    </LinearLayout>
</FrameLayout>
```

3. 创建两个维度的资源,一个在res/values/提供pre-Android 4.0自定义边距,一个在res/values-v14/没有为Android4.0小部件提供额外的填充:

res/values/dimens.xml:

```
<dimen name="widget_margin">8dp</dimen>
```

res/values-v14/dimens.xml:

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

另一个选择是在默认情况下简单地构建额外的边距在你的nine-patch背景的assets,而没有边距在API级别14或之后并提供不同的nine-patches。

Using the AppWidgetProvider Class

appwidgetprovider作为一个方便的类来继承broadcastreceiver处理App widget广播。

AppWidgetProvider接收事件广播，如App widget进行更新时，相关的App widget进行删除，启用和禁用。这些广播事件发生时，AppWidgetProvider将调用以下的方法：

onUpdate()

这种在AppWidgetProviderInfo由updatePeriodMillis属性定义的时间间隔叫做AppWidget更新。（请参阅上面添加AppWidgetProviderInfo元数据）这种方法也被称为用户添加App widget，所以它应该进行必要的设置，如视图定义事件处理程序，如果有必要，应启动临时服务。不过，如果你已经宣布配置的Activity，这种方法则不叫用户添加App widget，而是后续更新调用。配置的Activity完成后，它的作用就是执行第一次更新。（参见下面创建一个AppWidget配置的Activity。）

onAppWidgetOptionsChanged()

当窗口部件首次放置时会调用此函数，部件需要调整时也会调整此函数。可以使用此回调根据部件尺寸范围来显示或隐藏内容。调用getAppWidgetOptions()可以获取尺寸范围，此函数会返回包含下列内容的Bundle：

- OPTION_APPWIDGET_MIN_WIDTH--包含当前宽度下界，dp单位。

你必须在AndroidManifest<receiver>元素里声明广播接收器使用的您的AppWidgetProvider类的实现（参见在上述清单声明的一个AppWidget）。

- `OPTION_APPWIDGET_MIN_HEIGHT`--包含当前高度下界, dp单位。
- `OPTION_APPWIDGET_MAX_WIDTH`--包含当前宽度上界, dp单位。
- `OPTION_APPWIDGET_MAX_HEIGHT`--包含当前高度上界, dp单位。

API级别16引入了这一回调。要实施此回调, 请确保应用程序并不依赖于它, 因为旧设备不会进行调用。

onDeleted(Context, int[])

一个App Widget从App Widget主机中删除。

onEnabled(Context)

首次创建当一个实例App widget。例如, 如果用户添加了两个实例App widget, 这也只能称为第一次创建。如果你需要打开一个新的数据库或进行其他的设置, 只需要执行一次App widget实例, 那么在这个地方实例是非常好的。

onDisabled(Context)

这是最后一个实例时调用App widget, 从应用程序中删除主机中的App widget。使用`onDisabled(Context)`方法进行清理, 比如删除临时数据库。

onReceive(Context, Intent)

这是呼吁每个广播之前, 上面的每个回调方法。你通常不需要实现这个方法, 因为默认的`AppWidgetProvider`实现过滤器所有App widget广播, 并适当调用上面的方法。

注：在Android1.5中，有一个已知的问题，就是`onDeleted()`方法将不会被调用。为解决此问题，您可以实现`onReceive()`方法来接收`onDeleted()`方法的回调。

当每个App widget添加到一个主机时，最重要的是`AppWidgetProvider``onUpdate()`方法回调(除非你使用一个配置的Activity)。如果你的App widget接受任何用户交互事件，在回调时，你需要注册事件来处理程序。如果你的App widget不能创建临时文件或数据库，或者执行其他的工作，需要清理，您需要定义`onUpdate()`方法，它可能是唯一的回调方法。例如，如果你想要一个App widget窗口按钮，当点击时启动一个Activity，你可以使用以下实现的`AppWidgetProvider`：

```
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);
        }
    }
}
```

这appwidgetprovider只定义`onupdate()`方法，其目的是定义一个`PendingIntent`启动一个Activity和附加到App widget使用`setOnPendingIntent (int, PendingIntent)`方法的按钮。注意，

在`appwidgetids`它包括一个循环遍历每个条目，这是一个数组的`id`标识，确定每个App widget的创建。这样，如果用户创建多个实例的App widget，然后他们都同时更新。然而，只有一个`updateperiodmillis`时间表将管理所有的App widget。例如，如果更新计划被定义为每两个小时，在第一个后添加第二个实例的App widget并每小时更新一次，那么它们都将被定义为第一个而第二个更新周期会被忽略（他们会每两小时更新，而不是每隔一小时）。

注：由于`AppWidgetProvider`是继承`BroadcastReceiver`，不能保证你的进程保持运行后的回调方法的返回（见 `BroadcastReceiver`广播周期的信息）。如果您的App widget设置过程可能要花费几秒（也许同时执行Web请求），并且要求你的进程仍在继续，可以考虑在`OnUpdate()`方法中启动一个 Service。从内的服务，您可以执行更新自己的App widget，不用担心由于`AppWidgetProvider`关闭应用程序而出现"Application Not Responding"的(ANR)错误。参看Wiktionary sample's AppWidgetProvider一个App widget使用后台服务的示例。

也参考`ExampleAppWidgetProvider.java`的示例类。

Receiving App Widget broadcast Intents

`AppWidgetProvider`仅仅是一个方便的类。如果你想直接接收App widget广播，你可以实现自己的`BroadcastReceiver`或覆盖的`onReceive(Context, Intent)`方法。你需要知道以下四个意图：

- ACTION_APPWIDGET_UPDATE
- ACTION_APPWIDGET_DELETED
- ACTION_APPWIDGET_ENABLED
- ACTION_APPWIDGET_DISABLED

Creating an App Widget Configuration Activity

如果你想设定的用户，当他或她增加了一个新的App widget时，你可以创建一个App widget配置Activity。当前的Activity将自动启动的App widget的主机，并允许用户在创建时配置App widget的颜色，大小，更新周期或其他功能的设置。这个配置的Activity应该在Android manifest文件中声明是

一个标准的Activity。然而,它将发起这个App widget主机ACTION_APPWIDGET_CONFIGURE的一个Action,所以Activity需要接受这种意图。例如:

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

此外, Activity必须在AppWidgetProviderInfo XML文件中声明android:configure属性(参见前面添加AppWidgetProviderInfo的元数据)。例如,可配置的Activity声明如下:

```
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    ... >
</appwidget-provider>
```

注意,这个Activity是声明完全限定的命名空间,因为它是从包范围之外引用的。

你需要的是开始使用一个配置Activity。现在你所需要的是实际的Activity。然而,当你实现Activity有两个重要的事情要记住:

- 这个App widget主机调用配置Activity, 而配置Activity应该总是返回一个结果码。这个结果码应该包括App widget ID所通过的意图发起这一Activity(这个意图作为EXTRA_APPWIDGET_ID进行额外保存)。
- 当创建App widget时OnUpdate () 方法将不会被调用 (配置Activity启动时, 系统将不发送ACTION_APPWIDGET_UPDATE广播) 。

这是配置Activity的作用, 请求从App widget首次创建AppWidgetManager时更新。然而, onUpdate()方法将呼吁后续更新,它只是跳过第一次。

请参阅以下部分中的一个例子, 从配置和更新的App widget的代码片段中如何返回结果。

Updating the App Widget from the configuration Activity

当一个App widget使用配置Activity，这个Activity配置完成后负责更新App widget。你可以从appwidgetmanager通过请求直接更新。

以下总结了正确的更新App widget并关闭该配置Activity：

1. 第一步,把App widget的ID通过意图发起这一Activity:

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

2. 第二步,执行你的App widget配置.

3. 第三部,当配置完成后,通过调用getInstance(context)获得一个实例的AppWidgetManager:

```
AppWidgetManager appWidgetManager =
AppWidgetManager.getInstance(context);
```

4. 第四步,通过调用updateAppWidget(int,RemoteViews)更新App widget与RemoteViews布局:

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

5. 最后,创建返回的意图, 其设置Activity的结果码, 并终止该Activity:

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

提示:当你的配置Activity首次打开时,在该Activity使用RESULT_CANCELED设置结果码。这样,如果用户选择退出之前能够达到目的,App widget通知配置被取消,并不会再添加。

请参见ApiDemos ExampleAppWidgetConfigure.java的示例类。

Setting a Preview Image

Android 3.0引入了previewImage字段,它指定一个Appwidget预览。这个预览是给用户显示小部件的选择器。如果没有提供这个字段,则这个Appwidget的图标只用于预览。

如何在XML中指定此设置:

```
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:previewImage="@drawable/preview" >
</appwidget-provider>
```

为了帮助您的Appwidget (指定在previewImage领域) 创建预览图像,在Android模拟器中包含一个应用程序被称为“小工具预览”。要创建预览图像,则要启动该应用程序,选择您的应用程序的Appwidget,并设置您希望的预览图像,然后将它保存,并将其放置在您的应用程序的绘制资源。

Using App Widgets with Collections

Android 3.0引入了Appwidget集合。这些类型的Appwidget使用RemoteViewsService显示集合,支持了远程数据.比如从一个 content provider,它所提供的数据RemoteViewsService提出Appwidget通过使用以下视图类型,我们称之为“集合视图.”

ListView

一个视图来显示项目在一个垂直滚动列表项。例如,查看Gmail的Appwidget。

GridView

一个视图来显示项目在双向滚动的网格。例如,查看书签的Appwidget。

StackView

一堆卡片视图(有点像一盒名片), 用户可在其中滑动前插卡上或下

看到上一张或下一张卡.例如,YouTube和书籍的Appwidget。

AdapterViewFlipper

一个适配器支持简单的ViewAnimator不会苟同,在两个或多个视图。只有一个孩子会显示。

如上所述,这些集合视图由远程数据集合进行显示。这意味着他们使用一个Adapter以他们的数据来绑定他们的View。一个Adapter从一组数据为单独的视图对象绑定到个别项目。因为这些集合视图是由Adapter,Android框架必须包括额外的架构来支持他们的使用在Appwidget。一个应用程序上下文中的小部件,而适配器是一个RemoteViewsFactory所取代,这是一个简单的瘦包装器的Adapter接口。当请求一个特定的项目在收集、RemoteViewsFactory创建并返回条目的集合作为一个RemoteViews对象。为了将一个集合视图在你的Appwidget,您必须实现RemoteViewsService和RemoteViewsFactory。

RemoteViewsService是一个服务,允许远程适配器请求RemoteViews对象。RemoteViewsFactory是一个接口之间的适配器集合视图(如列ListView,GridView,等等)和底层的数据视图。从StackView Widget sample,下面是一个示例样板代码使用来实现这个服务和接口:

```
public class StackWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new
StackRemoteViewsFactory(this.getApplicationContext(), intent);
    }
}

class StackRemoteViewsFactory implements
RemoteViewsService.RemoteViewsFactory {

    //... include adapter-like methods here. See the StackView Widget
    //sample.

}
```

Sample application

这一节的代码摘录来自StackView Widget sample:

这个示例包括一堆10视图,它显示值"0!"到"9!"的样例Appwidget,其具有这些主要的行为:

- 用户可以垂直上抛视图的应用程序窗口显示下一个或前视图。这是一个内置的Stack View行为。
- 在无需任何用户交互,Appwidget通过其视图自动进步序列,就像一个幻灯片。这是由于设置`android:autoAdvanceViewId= "@+id/stack_view"`在`res / xml / stackwidgetinfo.xml`文件。这个设置适用于视图ID(在本例中是视图ID的堆栈视图)。
- 如果用户触摸的顶视图, Appwidget显示Toast消息“感动视图n”, 其中n是触摸的视图索引 (位置) 。这是如何实现的更多讨论, 请参阅 [Adding behavior to individual items](#)。

Implementing app widgets with collections

实现一个Appwidget集合,您遵循相同的基本步骤用于实现任何Appwidget。以下部分描述了您需要执行额外的步骤来实现一个Appwidget集合。

Manifest for app widgets with collections

除了在[Declaring an App Widget in the Manifest](#)列出的要求,有可能对应用程序组件绑定到你的RemoteViewsService集合,必须声明服务在你的Manifest文件与 `BIND_REMOTEVIEWS`许可。这可以防止其他应用程序从自由访问您的应用程序小部件的数据。例如,当创建一个Appwidget,使用RemoteViewsService填充一个集合视图,清单条目可能看起来像这样:

```
<service android:name="MyWidgetService"
...
    android:permission="android.permission.BIND_REMOTEVIEWS" />
```

`android:name = " MyWidgetService"`是指你RemoteViewsService的子类。

Layout for app widgets with collections

主要的要求Appwidget布局的XML文件,它包含一个集合的观点:`ListView`, `GridView`, `StackView`,或`AdapterViewFlipper`。这是`widget_layout.xml`的StackView Widget sample:

```

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

<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <StackView
        xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/stack_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:loopViews="true" />
    <TextView
        xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/empty_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:background="@drawable/widget_item_background"
            android:textColor="#ffffffff"
            android:textStyle="bold"
            android:text="@string/empty_view_text"
            android:textSize="20sp" />
</FrameLayout>

```

需要注意的是空视图，必须收集兄弟姐妹的集合视图，这空视图表示空状态。

除了布局文件为整个Appwidget,您必须创建另一个布局文件,它定义了布局中每一项的集合(例如,每本书的藏书布局)。例如,StackView Widget sample只有一个布局文件,widget_item.xml,因为所有项目使用相同的布局。但WeatherList Widget示例有两个布局文件:dark_widget_item.xml和light_widget_item.xml。

AppWidgetProvider class for app widgets with collections

与常规Appwidget,大部分代码在您的AppWidgetProvider子类中一般会在onUpdate()方法。在您的实现的主要区别为 onUpdate()方法在创建一个Appwidget集合,您必须调用setRemoteAdapter()方法。这告诉集合视图从哪里得到数据。然后 返回你的RemoteViewsService可以实现RemoteViewsFactory,和小部件可以提供适当的数据。当你调用这个方法,您必须通过一个intent指向您的实现和应用程序的小部件ID RemoteViewsService指定Appwidget更新。

例如,这里的StackView的小部件示例实现了onUpdate()回调方法来设

置RemoteViewsService作为远程适配器的Appwidget集合:

```

public void onUpdate(Context context, AppWidgetManager
appWidgetManager,
int[] appWidgetIds) {
    // update each of the app widgets with the remote adapter
    for (int i = 0; i < appWidgetIds.length; ++i) {
        // Set up the intent that starts the StackViewService, which
will
        // provide the views for this collection.
        Intent intent = new Intent(context,
StackWidgetService.class);
        // Add the app widget ID to the intent extras.
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
appWidgetIds[i]);

        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
        // Instantiate the RemoteViews object for the App Widget
layout.
        RemoteViews rv = new RemoteViews(context.getPackageName(),
R.layout.widget_layout);
        // Set up the RemoteViews object to use a RemoteViews
adapter.
        // This adapter connects
        // to a RemoteViewsService through the specified intent.
        // This is how you populate the data.
        rv.setRemoteAdapter(appWidgetIds[i], R.id.stack_view,
intent);

        // The empty view is displayed when the collection has no
items.
        // It should be in the same layout used to instantiate the
RemoteViews
        // object above.
        rv.setEmptyView(R.id.stack_view, R.id.empty_view);

        //
        // Do additional processing specific to this app widget...
        //

        appWidgetManager.updateAppWidget(appWidgetIds[i], rv);
    }
    super.onUpdate(context, appWidgetManager, appWidgetIds);
}
}

```

RemoteViewsService class

如上所述,你RemoteViewsFactory
RemoteViewsService子类提供了用
于填充远程集合视图。具体来
说,您需要执行以下步骤:

持久化数据

你不能依赖一个单一实例的服务,或任
何它所包含的数据。你不应在你
的RemoteViewsService存储任何数

1. RemoteViewsService子类。 通过RemoteViewsService是服务远程适配器可以请求RemoteViews。

2. 在你RemoteViewsService子类, 包括一个类, 它实现

了RemoteViewsFactory接口。 RemoteViewsFactory是一个接口的适配器之间一个远程集合视图(如列ListView, GridView, 等等)和底层的数据视图。 您的实现负责做出一个RemoteViews对象中每个项目的数据集。 这个接口是一个瘦包装器的Adapter。

主要内容是其RemoteViewsFactory的RemoteViewsService实现, 如下所述。

RemoteViewsFactory interface

您的自定义类, 它实现了RemoteViewsFactory接口提供了Appwidget的数据项的集合。 要做到这一点, 它结合了你的Appwidget条目XML布局文件与一个源的数据。 这个数据源可以是任何内容, 从数据库到一个简单的数组。

在StackView Widget sample, 数据源是WidgetItems数组。 这个RemoteViewsFactory函数作为一个适配器胶数据到远程集合视图。

最重要的两个方法, 你需要为你的RemoteViewsFactory子类实现是onCreate()和getViewAt()。

在系统调用onCreate () 方法中创建您的factory。 这是你建立任何连接和/或游标到数据源。 例如, StackView Widget sample使用onCreate()来初始化WidgetItem对象的数组。 当你的Appwidget是活跃的, 系统访问这些对象使用数组中的索引位置和它们所包含的文本显示。 以下是摘自StackView Widget sample的RemoteViewsFactory实现, 展示了部分的onCreate()方法:

```
class StackRemoteViewsFactory implements
RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 10;
    private List<WidgetItem> mWidgetItemList = new
ArrayList<WidgetItem>();
    private Context mContext;
    private int mAppWidgetId;

    public StackRemoteViewsFactory(Context context, Intent intent) {
        mContext = context;
```

据(除非它是静态的)。 如果你想让你的Appwidget的数据持久化, 最好的方法是使用一个ContentProvider使其数据持续超越过程中的生命周期

```
mAppWidgetId =
intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                    AppWidgetManager.INVALID_APPWIDGET_ID);
}

public void onCreate() {
    // In onCreate() you setup any connections / cursors to your
    // data source. Heavy lifting,
    // for example downloading or creating content etc, should
    // be deferred to onDataSetChanged()
    // or getViewAt(). Taking more than 20 seconds in this call
    // will result in an ANR.
    for (int i = 0; i < mCount; i++) {
        mWidgetItem.add(new WidgetItem(i + " !"));
    }
    ...
}
```

RemoteViewsFactory的方法getgetViewAt()返回一个RemoteViews对象对应于指定position的数据的数据集。以下是StackView Widget sample的RemoteViewsFactory实现:

```
public RemoteViews getViewAt(int position) {
    // Construct a remote views item based on the app widget item
    // XML file,
    // and set the text based on the position.
    RemoteViews rv = new RemoteViews(mContext.getPackageName(),
R.layout.widget_item);
    rv.setTextViewText(R.id.widget_item,
mWidgetItem.get(position).text);

    ...
    // Return the remote views object.
    return rv;
}
```

Adding behavior to individual items

上面的小节将向您展示如何绑定数据到您的应用程序小部件集合。但是，如果你想添加动态行为的集合视图中的各个项目吗？

Using the AppWidgetProviderClass 所描述的那样，您通常使用setOnClickPendingIntent() 来设置一个 对象的点击行为如造成一个按钮以启动一个Activity。但这种方法是不允许在一个单独的子视图收集项目(澄清一下，在Gmail的Appwidget你可以使用 setOnClickPendingIntent() 来设置一个全局按钮启动应用程序，例如启动应用程序，而不是在个别的清单项目)。

相反,添加集合中的个别项目的点击行为,您可以使
用setOnClickFillInIntent ()。这需要建立起一个悬而未决的意图模板为您的
集合视图,然后通过你的 RemoteViewsFactory集合设置一个代替者意图中
的每一项。

本节使用StackView Widget sample 来描述如何将行为添加到个别项目。
在StackView Widget sample ,如果用户接触到顶视图,这个应用程序小部件
显示Toast消息“感动视图n,n是指数(位置)的感动视图。它是如何工作的:

- 在StackWidgetProvider(一个AppWidgetProvider子类)创建一个悬而未决
的意图,有一个自定义的动作称TOAST_ACTION。
- 当用户触摸一个视图,目的是发射和它广播TOAST_ACTION。
- 这广播是拦截了StackWidgetProvider的onReceive()方法,并Appwidget的
触摸视图显示Toast消息。数据的收集项目通过remoteviewsservice提供了
remoteviewsfactory。

注意:StackView Widget sample 使用一个广播,但在此类场景通常一
个Appwidget将简单地启动一个Activity。

Setting up the pending intent template

这个StackWidgetProvider(AppWidgetProvider子类)设置一个悬而未决的
意图。个人项目的集合不能成立自己悬而未决的意图。相反作为一个整体的
收集设置了一个挂起的意图模板,个别项目设置填充的一个项目,通过项
目的基础上,创造出独特的行为意图。

当用户触摸一个视图而发送广播时,这个类也可进行接收广播。它处理这
个事件在其onReceive()方法。如果意图的行动是TOAST_ACTION。,在当
前视图Appwidget显示Toast消息。

```
public class StackWidgetProvider extends AppWidgetProvider {
    public static final String TOAST_ACTION =
"com.example.android.stackwidget.TOAST_ACTION";
    public static final String EXTRA_ITEM =
"com.example.android.stackwidget.EXTRA_ITEM";
    ...
    // Called when the BroadcastReceiver receives an Intent
    broadcast.
    // Checks to see whether the intent's action is TOAST_ACTION. If
    it is, the app widget
```

```

// displays a Toast message for the current item.
@Override
public void onReceive(Context context, Intent intent) {
    AppWidgetManager mgr =
        AppWidgetManager.getInstance(context);
    if (intent.getAction().equals(TOAST_ACTION)) {
        int appWidgetId =
            intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                AppWidgetManager.INVALID_APPWIDGET_ID);
        int viewIndex = intent.getIntExtra(EXTRA_ITEM, 0);
        Toast.makeText(context, "Touched view " + viewIndex,
        Toast.LENGTH_SHORT).show();
    }
    super.onReceive(context, intent);
}

@Override
public void onUpdate(Context context, AppWidgetManager
appWidgetManager, int[] appWidgetIds) {
    // update each of the app widgets with the remote adapter
    for (int i = 0; i < appWidgetIds.length; ++i) {

        // Sets up the intent that points to the
        StackViewService that will
        // provide the views for this collection.
        Intent intent = new Intent(context,
        StackWidgetService.class);
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
        appWidgetIds[i]);
        // When intents are compared, the extras are ignored, so
        we need to embed the extras
        // into the data so that the extras will not be ignored.

        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
        RemoteViews rv = new
        RemoteViews(context.getPackageName(), R.layout.widget_layout);
        rv.setRemoteAdapter(appWidgetIds[i], R.id.stack_view,
        intent);

        // The empty view is displayed when the collection has
        no items. It should be a sibling
        // of the collection view.
        rv.setEmptyView(R.id.stack_view, R.id.empty_view);

        // This section makes it possible for items to have
        individualized behavior.
        // It does this by setting up a pending intent template.
        Individuals items of a collection
        // cannot set up their own pending intents. Instead, the
        collection as a whole sets
        // up a pending intent template, and the individual
        items set a fillInIntent
        // to create unique behavior on an item-by-item basis.
        Intent toastIntent = new Intent(context,
        StackWidgetProvider.class);
        // Set the action for the intent.
        // When the user touches a particular view, it will have
        the effect of
        // broadcasting TOAST_ACTION.
        toastIntent.setAction(StackWidgetProvider.TOAST_ACTION);
}

```

```

toastIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
appWidgetIds[i]);

intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
PendingIntent toastPendingIntent =
PendingIntent.getBroadcast(context, 0, toastIntent,
                           PendingIntent.FLAG_UPDATE_CURRENT);
rv.setPendingIntentTemplate(R.id.stack_view,
toastPendingIntent);

        appWidgetManager.updateAppWidget(appWidgetIds[i], rv);
    }
super.onUpdate(context, appWidgetManager, appWidgetIds);
}
}

```

Setting the fill-in Intent

你的RemoteViewsFactory咋每个项目集合中必须设置一个代替者意图。这使得它可以区分一个给定的项目的个别点击动作。填充的意图，然后结合pendingintent模板来确定最终的意图将被执行时单击该项目。

```

public class StackWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new
StackRemoteViewsFactory(this.getApplicationContext(), intent);
    }

class StackRemoteViewsFactory implements
RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 10;
    private List<WidgetItem> mWidgetItemList = new
ArrayList<WidgetItem>();
    private Context mContext;
    private int mAppWidgetId;

    public StackRemoteViewsFactory(Context context, Intent intent) {
        mContext = context;
        mAppWidgetId =
intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                    AppWidgetManager.INVALID_APPWIDGET_ID);
    }

    // Initialize the data set.
    public void onCreate() {
        // In onCreate() you set up any connections / cursors to
your data source. Heavy lifting,
        // for example downloading or creating content etc,
should be deferred to onDataSetChanged()
        // or getViewAt(). Taking more than 20 seconds in this
call will result in an ANR.
        for (int i = 0; i < mCount; i++) {
            mWidgetItemList.add(new WidgetItem(i + " !"));
        }
    }
}

```

```

    }

    ...

    // Given the position (index) of a WidgetItem in the array,
use the item's text value in
    // combination with the app widget item XML file to
construct a RemoteViews object.
    public RemoteViews getViewAt(int position) {
        // position will always range from 0 to getCount() - 1.

        // Construct a RemoteViews item based on the app widget
item XML file, and set the
        // text based on the position.
        RemoteViews rv = new
RemoteViews(mContext.getPackageName(), R.layout.widget_item);
        rv.setTextViewText(R.id.widget_item,
mWidgetItems.get(position).text);

        // Next, set a fill-intent, which will be used to fill
in the pending intent template
        // that is set on the collection view in
StackWidgetProvider.
        Bundle extras = new Bundle();
        extras.putInt(StackWidgetProvider.EXTRA_ITEM, position);
        Intent fillInIntent = new Intent();
        fillInIntent.putExtras(extras);
        // Make it possible to distinguish the individual on-
click
        // action of a given item
        rv.setOnClickListener(R.id.widget_item,
fillInIntent);

        ...
        // Return the RemoteViews object.
        return rv;
    }
}

```

Keeping Collection Data Fresh

下图说明了发生在一个App Widget的使用集合更新时发生的流量。它显示了如何在App Widget的代码交互的RemoteViewsFactory，以及如何触发更新：



一个特性的Appwidget,使用集合是能够给用户提供最新的内容。例如,考虑Android 3.0 Gmail的Appwidget,它为用户提供了一个了解他们的收件箱。要实现这一点,您需要能够触发你的RemoteViewsFactory。 和集合视图获取和显示新的数据。你达到这个与AppWidgetManager调用notifyAppWidgetViewDataChanged()。这个调用会导致回调到你的onDataSetChanged RemoteViewsFactory()方法,它给了你机会去获取任

何新的数据。注意,您可以执行运算操作同步在onDataSetChanged() 回调。你是保证这个调用将元数据或视图之前完成从**RemoteViewsFactory**抓取数据。此外,您可以执行运算操作在getViewAt()方法。如果这个调用需要很长时间,加载视图(指定的**RemoteViewsFactory**的getLoadingView()方法)将显示在相应的位置,直到它返回集合视图。

来自“[index.php?title=App_Widgets&oldid=14016](#)”



Android Manifest

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

编辑者： eoe耗子

更新时间： 2012.07.27

目录

[[隐藏](#)]

[1 The AndroidManifest.xml File](#)

- [1.1 manifest文件结构](#)
- [1.2 文件约定](#)
- [1.3 文件特性](#)
 - [1.3.1 Intent过滤器](#)
 - [1.3.2 图标和标签](#)
 - [1.3.3 权限](#)
 - [1.3.4 库](#)

The AndroidManifest.xml File

每个应用程序在其根目录下必定有一个AndroidManifest.xml文件(文件名必须是这个)。这个manifest文件向android系统列出了应用程序的必要信息，有了这些信息，系统才能运行应用程序。除此之外，manifest还有以下作用：

- 列举了应用程序的java包。包名是识别应用程序的唯一标志。
- 描述了应用程序的组件——活动、服务、广播接收器和内容提供器。列

举了实现每个组件的类，并给出可能的值（例如，类能处理的[intent](#)信息）。这些声明使Android系统了解了应用程序包含的组件及其运行条件。

- 确定了主导应用程序组件的进程。
- 声明了应用程序拥有的权限，使其可以使用API保护的内容，与其他应用程序进行交互。
- 同时，也声明了其他应用程序与该应用程序组件交互所需要的权限。
- 列举了为应用程序运行时提供性能和其他信息的[Instrumentation](#)类的声明。这些声明只有在开发和测试的时候才会在manifest中使用，发布的时候需要将这些声明删除。
- 声明了应用程序支持的Android API的最低等级。
- 列举了应用程序必须链接的库。

manifest文件结构

下图显示了manifest文件的整体结构以及可以使用的所有元素。每一个元素及其所有属性都在另一个单独的文件中显示。该图按元素的字母顺序排序，点击图中元素的名称或者任何提到该元素的地方，即可查看其详细信息。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
    </application>
</manifest>
```

```
<activity-alias>
    <intent-filter> . . . </intent-filter>
    <meta-data />
</activity-alias>

<service>
    <intent-filter> . . . </intent-filter>
    <meta-data />
</service>

<receiver>
    <intent-filter> . . . </intent-filter>
    <meta-data />
</receiver>

<provider>
    <grant-uri-permission />
    <meta-data />
</provider>

<uses-library />

</application>

</manifest>
```

下面按字母顺序列出了manifest文件中的所有元素。这些已经是所有的合法元素，用户不能添加自定义的元素或属性。

[<action>](#)
[<activity>](#)
[<activity-alias>](#)
[<application>](#)
[<category>](#)
[<data>](#)
[<grant-uri-permission>](#)
[<instrumentation>](#)
[<intent-filter>](#)
[<manifest>](#)
[<meta-data>](#)
[<permission>](#)
[<permission-group>](#)
[<permission-tree>](#)
[<provider>](#)
[<receiver>](#)
[<service>](#)
[<supports-screens>](#)

[<uses-configuration>](#)
[<uses-feature>](#)
[<uses-library>](#)
[<uses-permission>](#)
[<uses-sdk>](#)

文件约定

有些约定和规则适用于manifest文件中的所有元素和属性：

元素

只有[<manifest>](#)和[<application>](#)元素是必须的，而且只能出现一次。大多数其他元素都可以出现多次或者不出现——尽管要实现有意义的功能时，有些元素是必不可少的。

如果一个元素包含某些东西，那一定是包含其他元素。所有的值都必须通过属性设置，而不是元素中的字符。

同等级的元素通常没有固定顺序。例如：[<activity>](#), [<provider>](#), 和[<service>](#)可以按任何顺序混合出现。（[<activity-alias>](#)通常要遵守这个规则：必须在以其作别名的原[<activity>](#)之后出现。）

属性

从正式意义上来说，所有的属性都是可选的。然而，为了实现功能，元素的某些属性是必须要指明的。可以将这份文档作为指南。对于真正可选的属性，意味着在未指明的时候会有一个默认值或默认状态。除了[<manifest>](#)根元素的某些属性，其他所有的属性都以 `android:` 为前缀——例如，`android:alwaysRetainTaskState`。因为前缀是通用的，所以当引用属性的名字的时候文档会自动将其忽略。

声明类名

很多元素都对应着java对象，包括应用程序元素本身（[<application>](#)）及其主要的组件——活动（[<activity>](#)），服务（[<service>](#)），广播接收器（[<receiver>](#)）和内容提供器（[<provider>](#)）。

如果定义一个子类，比如用户通常会定义的组件子类

（[Activity](#), [Service](#), [BroadcastReceiver](#), [ContentProvider](#)），通常由`name`属性声明。这个名字必须包括完整的包名。例如，[Service](#)的子类必须按如下方法定义：

```
<manifest . . . >
  <application . . . >
    <service android:name = "com.example.project.SecretService" .
```

```

    . . .
    </service>
    . . .
</application>
</manifest>
```

但是，作为简写，如果字符串的第一个字符为一个点，那么字符串将会被添加到应用程序的包名（由[manifest](#)的[package](#)属性指定）之后，下面的赋值跟上面是一样的效果：

```

<manifest package="com.example.project" . . . >
    <application . . . >
        <service android:name=".SecretService" . . . >
            . . .
        </service>
    . . .
</application>
</manifest>
```

当使用组件时，`android`会实例化命名的子类。如果子类没有指定，将会实例化基类。

多个值

如果元素可以赋多个值，通常会重复进行。例如，一个[intent filter](#)可以列举多个[action](#)：

```

<intent-filter . . . >
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    . . .
</intent-filter>
```

资源值

某些属性的值可以展示给用户看——例如，活动的标签和图标。这些属性的值应该本地化，因此可以从一个资源或主题中获取。资源值通常按以下格式表示，`@[package:]type:name`

其中，如果资源跟应用程序在同一个包中，那么包名可以省略。`type`是资源的类型——例如“`string`”或者“`drawable`”——`name`是识别具体资源的标识。例如：

```
<activity android:icon="@drawable/smallPic" . . . >
```

从主题中获取的值可以用类似的方法表示，只不过首字符使用`?而不是@:`

```
? [package:]type:name
```

字符串值

当属性值是一个字符串的时候，需要使用双反斜杠('\\')来转义字符——例如'\\n'代表换行，或者'\\uxxxx'代表Unicode字符。

文件特性

以下内容描述了manifest文件中表现出来的Android特性：

Intent过滤器

应用程序的核心组件（活动、服务、广播接收器）由intent激活。intent是一组描述要做的动作的信息（一个[intent](#)对象）——包括依赖的数据、执行动作的组件种类和其他相关的说明。Android选用对应的组件来响应intent，需要的时候实例化组件，并将其传给intent对象。

组件通过[intent](#)过滤器表明了自身的能力——能响应的[intent](#)的种类。由于android系统在运行之前必须明确组件能够处理哪些[intent](#)，因此可以在manifest文件中声明[intent](#)过滤器。一个组件可以有多个不同的过滤器，代表不同的能力。

显式地指定目标组件的[intent](#)将激活该组件；过滤器并不起作用。但是，如果一个[intent](#)没有通过名字指定目标，那么只有在通过组件的某一过滤器时才会激活组件。

更多关于[intent](#)对象及其过滤器的信息，参见独立的文档，[intent及其过滤器](#)。

图标和标签

很多元素都有icon和label属性，用于给用户显示一个小图标或者一个文本。有些元素还有一个description属性，用于在屏幕上显示更长的扩展信息。例如，[<permission>](#)元素以上三个属性都有，所以当用户要求为应用程序授予权限的时候，将会看到代表权限的图标、权限的名称，以及描述权限作用的信息。

任何情况下，在容器元素中设定的图标和标签，将会成为其子元素的默认图标和标签。因此，在[application](#)元素中设定的图标和标签，将会是应用程序中每个组件的默认设置。类似的，组件的图标和标签——例

如，[activity](#)元素——将会是该组件的每一个[intent-filter](#)元素的默认设置。如果[application](#)元素设置了一个标签，但是activity和intent过滤器没有设置，那么应用程序标签将同时适用于activity和intent过滤器。

不管组件何时展现给用户，intent过滤器设置的图标和标签都代表实现该过滤器功能的组件。例如，设置

了“`android.intent.action.MAIN`”和“`android.intent.category.LAUNCHER`”的过滤器代表了由这个activity来启动应用程序——也就是说，应用程序启动的时候，这个activity最先显示。因此，启动画面显示的图标和标签就是过滤器中设置的图标和标签。

权限

权限是访问设备上的代码或数据的严格限制。这个限制用于保护关键代码和数据，以免被滥用导致出现错误或影响用户体验。

每个权限都有一个唯一的标记。通常这个标记表示被限制的行为。例如，以下是android中定义的一些权限：

`android.permission.CALL_EMERGENCY_NUMBERS`

`android.permission.READ_OWNER_DATA`

`android.permission.SET_WALLPAPER`

`android.permission.DEVICE_POWER`

一个特性最多由一个权限控制。

如果应用程序需要访问有权限保护的特性，必须在manifest中用[`<uses-permission>`](#)元素声明该应用程序需要使用该权限。然后，安装应用程序的时候，安装程序将检查用户签署的应用程序证书的权限，或者，在某些情况下，询问用户，从而决定是否需要给应用程序授予要求的权限。如果授权成功了，那么应用程序就可以使用被保护的特性了。如果授权失败，那么应用程序尝试访问被保护的特性的時候，将会直接失败，而不通知用户。

应用程序也可以使用权限保护自己的组件（活动、服务、广播接收器、内容提供器）。android定义的所有权限（在[android.Manifest.permission](#)中列出的）或者其他应用程序声明的权限，都可以使用。或者应用程序也可以定义自己的权限。新的权限通过[`<permission>`](#)元素声明。例如，可以按照以下方法保护activity：

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . .
/>
```

```

<uses-permission android:name="com.example.project.DEBIT_ACCT"
/>

<application . . .
    <activity android:name="com.example.project.FreneticActivity"
        android:permission="com.example.project.DEBIT_ACCT"
        . . . >
    . . .
</activity>
</application>
</manifest>

```

注意，在这个例子中，`DEBIT_ACCT`权限不仅使用`<permission>`元素声明，同时也需要使用`<uses-permission>`声明。应用程序的其他组件想要运行被保护的`activity`的时候，就需要使用该权限，即使该`activity`是被应用程序本身保护的。

还是这个例子，如果`permission`属性已经在其他地方声明过了（比如`android.permission.CALL_EMERGENCY_NUMBERS`中），那么就无需再用`<permission>`元素声明了，但是`<uses-permission>`元素的声明还是需要的。

`<permission-tree>`元素为代码中需要使用的权限组定义了一个命名空间。`<permission-group>`元素为权限组定义了一个标签（无论权限是在`manifest`中使用`<permission>`元素声明的，还是在其他地方声明的）。区别只是展现给用户的时候，权限是如何组合的。`<permission-group>`元素不会指明权限属于哪一个组，只给出了组的名字。把组名添加到`<permission>`元素的`<permission-group>`属性就可以把该权限分配到该组了。

库

每一个应用程序都会连接到`android`的默认库，这个库中包含了编译应用程序需要的基础包（包中含有常用的类，如`Activity`, `Service`, `Intent`, `View`, `Button`, `Application`, `ContentProvider`等）。

但是，有些包有自己的库。如果应用程序要使用这些包中的代码，就必须显式地链接这些包的库。在`manifest`文件中必须包含独立的`<uses-library>`元素，将需要的库全部列举出来。（库的名称可查阅`package`文档。）

来自“[index.php?title=Android_Manifest&oldid=13240](#)”



User Interface

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： 枫露之茗

原文链接：<http://developer.android.com/guide/topics/ui/index.html>

User Interface

你应用程序的用户接口就是用户能看见和交互的东西。Android提供了各种各样已经建立好的UI组件，例如构造好的布局对象，允许为你应用程序建立图形化的用户接口的UI控制器。Android也提供了其他特殊接口的UI模型，例如：dialogs, notifications, 和 menus。



[OVERVIEW>>](#)

BLOG ARTICLES-博客文章

- 跟菜单按钮说再见

根据冰激凌三明治带来的许多装置，为了提升Android用户的体验，将你的设计移植到动作条上是非常重要的。

- 新的布局小部件：Space和GridLayout

冰激凌三明治(ICS)炫耀出两款通过大的陈列控件：Space和GridLayout，已经设计好来支持富有用户接口的模型的新的小部件。

- 调整工作条

通过在你的目标应用程序中使用动作条，你将会给用户一个与application交互的熟悉的方法。

- 与ViewPager关联的水平视图

无论你是一个Android开发新手还是一个经验丰富的开发人员，对与你实现设置滚动水平视图都不会花太长的时间。

TRAINING-训练

- 实现有效的导航

这个类显示着怎样为你等级的制度的应用程序计划出更高级别的屏幕，然后为了允许用户有效的、直觉的研究你的内容，选择导航更接近的形式。

- 为多点屏幕设计

Android提供了各种各样不同屏幕大小装置类型，涉及范围从小的手机到大的电视机。这个类显示了你怎么实现一个为许多屏幕配置的用户接口。

- 提高布局表现力

布局是一个能够直接影响用户体验的Android的应用程序的核心部分，如果简单的实现，那么在应用程序显示UI的时候，你的布局会导致内存紧缺。这个类显示你怎么样避免这些问题。

来自“[index.php?title=User_Interface&oldid=8290](#)”

UI Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： 枫露之茗

原文链接：<http://developer.android.com/guide/topics/ui/overview.html>

UI Overview-UI综述

所有在Android管理中的用户接口元素是通过View和ViewGroup对象来建立的。一个view就是一个可以在与用户交互的屏幕上绘制一些动东西的对象,一个ViewGroup就是一个为了定义布局接口的而装载其他View(和ViewGroup)的对象.

Android提供了一系列为你提供普通的输入控制控件 (如：按钮和文本域) 和各种布局模式 (例如：线性布局和相对布局) 的View和ViewGroup的子类。

User Interface Layout- 用户接口布局

你应用程序的每一个组件的用户接口是用一些具有继承关系View和ViewGroup对象来定义的，如下图1所示。每一个视图组都是一个能够组织子视图的可见的容器，然而子视图可能放置一些控件和来自UI部分的小部件。这个具有继承关系的树可以与你想的一样简单（但是表现地越简明越好）。



图1： 定义在UI布局中的继承视图树的图解

为了申明布局，你可以用代码实例化一些视图对象和开始建一个树，简单而有效的方法就是在你的xml布局文件中定义，xml为布局提供了一个个性化、可读性好的结构，类似于HTML.

The name of an XML element for a view is respective to the Android class it represents. So a <TextView> element creates a TextView widget in your UI, and a <LinearLayout> element creates a LinearLayout view group.

例如，一个包含文本视图和按钮的简单垂直布局如下图所示：

```
<?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" >
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button" />
</LinearLayout>
```

当你在你的应用程序中加载布局资源的时候，Android会把每一个布局文件中的节点初始化成在运行时，用来定义扩展的行为、查询对象状态和修改布局文件的对象。

更多关于创建UI布局的指南，查阅[XML Layouts](#)。

User Interface Components-用户接口组件

你不需要建立所有UI使用的View和ViewGroup对象。Android提供了许多能够提供标准的那些你需要简单定义内容 的UI布局的应用组件。这些每一个UI组件都有一组在他们文件中描述的独特的APIs，例如：Action Bar, Dialogs, 和 Status Notification。

来自“[index.php?title=UI_Overview&oldid=8056](#)”

Layouts

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： eoe_zvivi521

原文地址：<http://developer.android.com/guide/topics/ui/declaring-layout.html>

目录

[[隐藏](#)]

[1 Layouts](#)

- [1.1 写一个XML](#)

[2 加载XML资源](#)

- [2.1 属性](#)
 - [2.1.1 ID](#)
 - [2.1.2 布局参数](#)
- [2.2 布局位置](#)
- [2.3 大小、padding和margins](#)
- [2.4 一般布局](#)
- [2.5 使用适配器构建布局](#)
 - [2.5.1 用数据填充适配器视图](#)
 - [2.5.2 处理click事件](#)

Layouts

布局为 UI 供了一个可视化的结构，比如对于一个 activity 或者 app widget 的 UI。你可以用两种方式声明布局：

- 在**XML**中声明**UI**元素。Android提供了简单的 XML 元素和 View 类以及

子类对应，就像布局和**widgets**一样。

- 在运行时实例化布局元素。应用能够通过编程创建显示对象和显示组对象(并且操作他们的属性)

Android framework提供了非常灵活的方法来声明和管理应用UI。例如，可以在**XML**中声明默认布局，屏幕元素会根据它们的属性显示。接下来可以在应用中增加代码修改屏幕对象的状态，也可以在运行时修改在**XML**中声明的对象。在**XML**中声明UI的好处是，可以更好地区分显示和控制这些行为的代码。UI描述与应用代码无关，也就是说可以修改和调整UI布局但是不用修改源代码以及重新编译。例如，能够为不同的屏幕目标、不同的设备屏幕大小、不同的语言创建不同的**XML**布局文件。另外，在**XML**中声明布局使得UI更容易可视化，这样更容易调试问题。就其本身而言，这个文档主要用于教会你如何在**xml**中声明布局。如果您对运行时对象实例化感兴趣，那么请参考[viewgroup](#) 以及[view](#)类说明。一般来说，**xml**声明UI元素的词汇和类的命名以及方法名密切相关，元素根据类名、属性名根据方法名来命名。实际上，能猜到什么**XML**属性对应一个类的方法，或者能够猜到哪个类对应给定的**XML**元素，这往往是直接的对应。但是，注意并不是所有的词汇都是等同的。在某些情况下，有的命名有些许不同。例如，**EditText**元素有个**text**属性对应**EditText.setText()**方法。

提示：在[Common Layout Objects](#)学习更多不同的布局类型。在[hello views](#) 查看教程指南教程的集合。

写一个**XML**

使用**android**的**XML**词汇，可以快速的设计UI布局和它们包含的屏幕元素。同样，创建**web**页面用**html**使用一系列元素。每一个布局文件必须包含一个根元素。这个根元素必须是一个**View** 或者**ViewGroup**对象。定义了根元素，可以添加任意的布局对象或者**widgets**作为子元素，构建一个**View**层次定义布局。例如，这是一个**XML**布局文件使用了纵向的[线性布局](#)来排列一个[TextView](#)和[Button](#)：

```
<?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" >
<TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
<Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>

```

XML声明完布局后，用.xml的扩展名来保存这个文件，在android工程/res/layout/目录下，它会编译。我们接下来会讨论这里显示的每个属性。

加载XML资源

当编译应用的时候，每一个XML布局文件都被编译到view资源中。要在[Activity.onCreate\(\)](#)回调实现里面加载布局资源。通过调用[setContentView\(\)](#)来加载，按照R.layout.layout_file_name的格式，来传递布局资源。例如，XML布局被保存为main_layout.xml，可以在[activity](#)这样加载：

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}

```

Activity里面的onCreate()回调方法是在activity launched时，被android framework调用的。（在[Activities](#)文档里面，看生命周期的讨论）

属性

每一个View和ViewGroup对象支持他们各自的XML属性。有些属性是特定的View对象（例如，TextView支持textSize属性）但是这些属性被扩展这个类的所有View对象继承。有些属性是对所有的View对象都适用的，因为它们是从根View类继承下来的（像id属性）。其它的属性被认为是布局参数，这些属性是被该对象的父对象ViewGroup定义的，用来描述特定的View对象布局目标。

ID

每一个可视对象都可能有一个数字ID和它相关，在树中唯一的标示这个可视对象。当程序编译完，这个ID被认为是一个数字，但是ID在XML布局文件里面是通过给id属性用string类型赋值的。这是一个XML针对所有可视对象的通用属性([view类](#)定义的)，你会经常用到它。ID语法包含在XML标签内：

```
android:id="@+id/my_button"
```

这个@符号在字符串开头表明xml解析器会解析和扩展剩余的ID字符串，并把它定义为ID资源。“+”表示这是一个新的资源名字，要创建并且增加的我们的资源中(在R.java文件里)。Android framework层也提供了一部分ID资源。当遇到android 资源ID，你不需要“+”，但是要加上android包名命名空间，如下所示：

```
android:id="@+id/empty"
```

Android的包命名空间中，我们现在引用android.R资源类的ID,而不是本地的资源类中引用。为了创建views，并在应用中使用，常用的模式如下：

1. 在布局文件中创建view/widget，并为他们分配一个唯一的ID

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text" />
```

2. 接下来创建view对象实例，从布局中找到它(一般在[OnCreate\(\)](#)方法中)

```
Button myButton = (Button) findViewById(R.id.my_button);
```

如果创建[RelativeLayout](#)在一个相对布局里面，为每一个视图对象定义ID是非常重要的。兄弟视图能够定义他们的布局相对于另一个兄弟view，这些视图是用唯一的ID标示的。

一个ID在整个树中不一定要求唯一，但是你搜索的部分树应该是唯一的(经常是整个树，所以最好的办法是在整个树中是唯一的)

布局参数

命名为 `layout_something` 的 XML 布局属性都是为 `view` 在其所在的 `view group` 中定义了合适的布局参数。

每一个`viewgroup`类都是继承了[viewgroup.LayoutParams](#)。整个子类包含适当的类型能够在适当的视图组为每一个子`view`定义尺寸和位置。在图一中能够看出，父视图组为每一个子`view`定义了布局参数（包括子视图组）。



图1. 带布局参数的可视化视图层

每一个布局参数子类拥有自己的句法赋值。每一个子元素必须根据父元素来定义恰当的布局参数，尽管父元素也为它的子元素定义了不同的布局参数。所有的可视组包含宽度和高度（）。每一个视图都要定义。需要布局参数也包含可选边缘和边界。可以指定宽度和高度使用精确的测量工具，并不一定经常想这么做。往往，会使用这些常量来设置宽度和高度。

- `wrap_content` 会根据内容自动调整视图到合适的空间。
- `Fill_parent`(在API Level 8命名为`match_parent`)会告诉你的视图变成和父视图组允许的范围一样大。

一般说来，不推荐使用用绝对单位像是像素来指定布局的宽度和高度。而使用相对的测量工具像是与密度无关的单位(`dp`)，`wrap_content`，或者`fill_parent`是一种较好的方法，因为它能够保证您的应用程序在不同屏幕大小的设备上都能够正常显示。公认的测量类型在[Available Resources](#)文档中定义。

布局位置

一个视图的几何形状是一个矩形。一个视图有一个坐标、用`left`和`top`参数表示，和两个维度，用宽度和高度表示。坐标和纬度的单位是像素。

调用[getLeft\(\)](#) 和[getTop\(\)](#)方法能够得到视图的坐标。前者返回`left`、或者`X`，表示视图的矩形坐标。。后者返回`top`，或者`Y`，表示视图的矩形坐标。这些方法都返回视图相对于父视图的相对坐标。比如，如果`getLeft()`返回`20`，这表示当前视图在父视图左侧边缘向右`20`个像素的处。

另外，提供了许多便利的方法避免了不必要的计算，像是[getRight\(\)](#)

和[getBottom\(\)](#) 这些方法返回视图矩形的右侧和底侧边缘。例如，调用[getRight\(\)](#)和进行这个计算是一样的:[getLeft\(\) + getWidth\(\)](#).

大小、padding和margins

视图大小使用宽度和高度表示的。一个视图实际上有两对宽度和高度值。

第一对是测量得到的宽度和高度。这些尺寸定义了一个视图想在父视图中多大。测量尺寸能够通过调用[getMeasuredWidth\(\)](#) 和[getMeasuredHeight\(\)](#) 得到。

第二对被简单的成为宽度和高度，有时成为绘制的宽度和高度。这些尺寸定义了在绘制时和布局后视图在屏幕上的实际大小。这些参数(不一定非要)或许和测量得到的宽度和高度不同。这些参数能够通过调用[getWidth\(\)](#)和[getHeight\(\)](#) 得到。

为了测量尺寸，需要考虑padding。padding 表示视图的左侧、上侧、右侧和下侧部分。padding能够用来偏移视图中的内容、通过指定一定数量的像素。比如，左侧padding是2，会让视图内容偏移视图左边缘2个像素。padding能够通过使用[int, int, int\)](#) [setPadding\(int,int,int,int\)](#)方法进行设置，调用[getPaddingLeft\(\)](#), [getPaddingTop\(\)](#),[getPaddingRight\(\)](#) 和[getPaddingBottom\(\)](#)进行查询。

尽管视图能够定义padding，它不提供任何margins的支持。view group提供了这样的支持。更多信息请参考[ViewGroup](#)和[ViewGroup.MarginLayoutParams](#).

dimesions详细信息，请查看[Dimension Values](#).

一般布局

[ViewGroup](#)的每个子类提供了唯一的方法来显示视图。。。接下来是在android 平台常用一些的布局类型。注意：尽管为了UI设计，可以在一个布局里面放置一个或者多个布局，但是应该力求让布局层次尽可能的少。如果视图层次很少会绘制的很快(一个广度视图层次比深度层次视图好很多)

[线性布局](#)  这个布局是让其孩子组织成一个单一的水平或垂直行。如果窗口长度超出了屏幕，它会自动创建滚动条。

[相对布局](#)  让你能够指定子对象之间的相对位置(孩子A在孩子B的左侧)或者和父对象之间的相对位置(和父对象顶端对齐)

[页面视图](#)  显示web页面

使用适配器构建布局

如果布局是动态的或者非预定义的，可以在运行时使用一个布局子类[AdapterView](#)来填充布局。[AdapterView](#)类的子类使用一个适配器将数据绑定到它的布局。适配器表现为数据源和[AdapterView](#)布局之间的中间人-适配器检索数据(从像是数组或者数据库查询这样的数据源)并将它转换成可以添加到[AdapterView](#)布局视图中的条目。通用的适配器布局包括：

[List View](#)



显示滚动的列的列表

[Grid View](#)



显示滚动的网格的行和列

用数据填充适配器视图

可以填充一个[AdapterView](#)，如[ListView](#)和[GridView](#),通过将[AdapterView](#)实例绑定到一个适配器上，这个适配器从外部数据源检索数据并为每个数据条目创建一个视图。Android提供了许多适配器子类用来检索各种各样的数据并为[AdapterView](#)构建视图。最常用的通用适配器是：

ArrayAdapter

当数据源是数组的时候，使用这个适配器。默认情况

下， **ArrayAdapter**为每个数组元素创建一个视图，通过在每个元素调用 [toString\(\)](#) 后，将数据存放在 [TextView](#) 里面。

例如，如果您想将一个字符串数组显示在 [ListView](#) 中，使用构造器为每个字符串和字符串数组指定布局初始化一个 [ArrayAdapter](#)：

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, myStringArray);
```

构造器参数：

- 应用[上下文](#)
- 一个针对字符串数组中的字符串都有一个[TextView](#)的布局
- 字符串数组

然后只需要在 [ListView](#) 上调用

[SetAdapter\(\)](#):

```
ListView listView = (ListView) findViewById(R.id.listView);
listView.setAdapter(adapter);
```

为了调整元素外观，您可以为了数组中的对象重写 [toString\(\)](#) 函数。或者，为每个元素创建非 [TextView](#) 的视图(比如，想获取每个数组元素的 [ImageView](#))扩展 [AdapterArray](#)，重写 [android.view.View](#), [android.view.ViewGroup](#) [getView\(\)](#) 方法返回想要的视图类型。

[SimpleCursorAdapter](#)

当数据来源是 [Cursor](#) 时，则使用这个适配器。在使

用 [SimpleCursorAdapter](#) 时，必须为 [Cursor](#) 的每一行数据指定使用的布局，也必须为 [Cursor](#) 的每一栏指定要使用布局中的哪个控件。比如，创建一个包含用户名称和电话号码的列表。首先执行 [Cursor](#) 数据库的查询，返回的是 [Cursor](#) 一行用户的信息，其中有用户的名称和电话号码等信息；然后创建一个字符串数组用于指定将显示 [Cursor](#) 的哪一列，创建一个 [integer](#) 数组为每一列指定相应的视图来放置数据。

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
    ContactsContract.CommonDataKinds.Phone.NUMBER};
```

```
int[] toViews = {R.id.display_name, R.id.phone_number};
```

初始化SimpleCursorAdapter时，传入的布局参数和两个数组对Cursor每个结果都是适用的：

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
    R.layout.person_name_and_number, cursor, fromColumns,
    toViews, 0);
ListView listView = getListView();
listView.setAdapter(adapter);
```

SimpleCursorAdapter接下来为Cursor每一行使用提供的布局来创建视图，通过将每个fromColumns元素插入到toViews指定的视图。

在应用程序的生命周期中，如果适配器对应数据被改变了，应该调用notifyDataSetChanged()。这会通知相应的视图数据变化了、视图会自我刷新。

处理**click**事件

您可以通过实现**AdapterView.OnItemClickListener**接口，来处理**AdapterView**每个项目的**click**事件。例如：

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new
OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int
position, long id) {
        // Do something in response to the click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```

来自“[index.php?title=Layouts&oldid=13729](#)”



Linear Layout

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

翻译作者：小手冰凉

预计时间：2012-8-4

原文链

接：<http://developer.android.com/guide/topics/ui/layout/linear.html>

线性布局

线性布局是一个视图组，它所有的子视图都在一个方向对齐，水平或者垂直。你可以指定布局的方向通过`android:orientation` 属性。

线性布局的所有子视图排列都是一个靠着另一个，因此垂直列表每行仅仅有一个子视图，不管有多宽。水平列表只能有一行的高度（最高子视图的高度加上边距距离）。线性布局对于每一个子视图涉及到边缘在子视图和权重（左边或者右边以及中间对齐）之间。



布局权重

线性布局支持给个别的子视图设定权重，通过`android:layout_weight`属性。就一个视图在屏幕上占多大的空间而言，这个属性给其设定了一个重要的值。一个大的权重值，允许它扩大到填充父视图中的任何剩余空间。子视图可以指定一个权重值，然后视图组剩余的其他的空间将会分配给其声明权重的子视图。默认的权重是0.

例如，如果有三个文本框，其中两个声明的权重为1，另外一个没有权重，没有权重第三个文本字段不会增加，只会占用其内容所需的面积。其他两个同样的会扩大以填补剩余的空间，在三个文本域被测量后。如果第三个字段，然后给定的权重为2（而不是0），那么它现在的声明比其他的更重要，所以它得到一半的总的剩余空间，而前两个平均分配剩余的。

例子

```
<?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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

例子运行的结果如下图所示：



如果想更详细的了解线性布局的每个子视图的可用属性，可以参考**LinearLayout.LayoutParams**

来自“[index.php?title=Linear_Layout&oldid=13853](#)”



Relative Layout

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

翻译作者：小手冰凉

预计时间：2012-8-4

原文链

接：<http://developer.android.com/guide/topics/ui/layout/relative.html>

相对布局

RelativeLayout顾名思义，相对布局，在这个容器内部的子元素们可以使用彼此之间的相对位置（例如，在某个视图左边`left-of`）或者和容器间的相对位置（例如，与父视图左对齐，底部对齐或者居中等）来进行定位。



RelativeLayout（相对布局）是一个为用户界面设计，非常强大的工具。因为它可以消除嵌套视图组，并保持你的布局层次更简洁，从而提高性能。如果你发现自己使用多个嵌套的**LinearLayout**组，您可能能够取代单一**RelativeLayout**。

定位视图

相对布局可以让它的子视图指定自己的相对于父视图的位置或者视图元素之间的相对位置（通过指定的**ID**）。你可以使两个元素右边界对齐，或者使一个视图在另一个视图下方，或者使视图在屏幕居中偏左等等。默认情况下，

所有的子视图在布局的左上角。所以你必须通过使用布局属性**RelativeLayout.LayoutParams**中各种不同的可用属性值来定义每个视图的位置。

相对布局视图的一些可用属性包括：

- **android:layout_alignParentTop**

如果设置为“true”,使这一视图的顶部边缘匹配父类的顶部边缘

- **android:layout_centerVertical**

如果“true”,设置此子视图在父视图中垂直居中。

- **android:layout_below**

设置此视图的上边缘位于通过资源ID指定的视图的下方。

- **android:layout_toRightOf**

设置此视图的左边缘位于通过资源ID指定的视图的右方。

这仅仅是几个例子，所有的布局属性我们可以在**RelativeLayout.LayoutParams**中找到。

每个布局属性的值既可以是boolean类型的值来确定布局相对于父布局的位置，也可以是某个子视图的ID，来指定布局相对于这个子视图的位置。

在你的xml布局文件中，依赖于其他视图的布局可以在声明的时候没有顺序。例如：

你可以声明“View1”在“View2”的下方，即使View2是在视图层次结构中最后一个被声明的。下面的例子演示了这种情况。

例子

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
file:///D:/guide/RelativeLayout[2015/9/23 19:14:48]
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
<EditText
    android:id="@+id/name"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/reminder" />
<Spinner
    android:id="@+id/dates"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/name"
    android:layout_alignParentLeft="true"
    android:layout_toLeftOf="@+id/times" />
<Spinner
    android:id="@+id/times"
    android:layout_width="96dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/name"
    android:layout_alignParentRight="true" />
<Button
    android:layout_width="96dp"
    android:layout_height="wrap_content"
    android:layout_below="@+id/times"
    android:layout_alignParentRight="true"
    android:text="@string/done" />
</RelativeLayout>
```

例子运行的结果如图所示：



来自 "[index.php?title=Relative_Layout&oldid=9065](#)"

List View

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

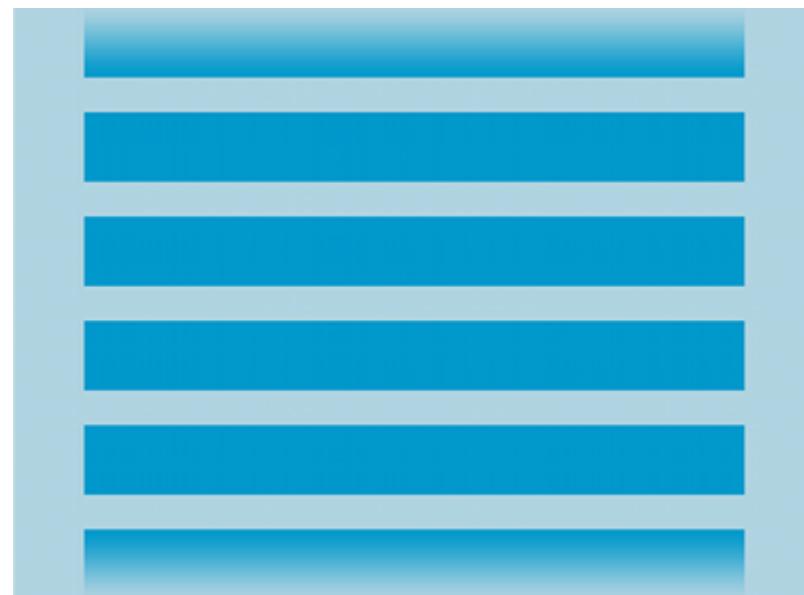
<http://docs.eoeandroid.com/guide/topics/ui/layout/listview.html>

作者：smilysas

更新时间：2012年7月28日

List View

列表视图是一个成列显示滚动项的视图组合。成列的项都通过[Adapter](#)]被自动插入到列表中，其中，[Adapter](#)适配器可以从数组或者数据库查询中提取出内容并转化成列表视图中的项。



使用一个加载者

使用一个`CursorLoader`是避免一个异步任务查询光标`Cursor`时阻塞程序主线程的标准途径。当`CursorLoader`收到一个`Cursor`结果，`LoaderCallbacks`会收到一个对`onLoadFinished()`的回调，这时可以利用新的`Cursor`和列表视图更新`Adapter`并显示结果。

虽然`CursorLoader`函数在Android3.0(API级别 11)中才第一次引入，程序可以通过引入`Support Library`使用它们来支持运行Android 1.6及以上的设备。

要查看更多关于利用`Loader`异步加载数据的信息，请参看[Loaders](#)

例子：

下面的例子是把`ListView`作为唯一默认布局元素的活动`ListActivity`。它完成向`Contacts Provider`查询姓名和电话号码清单的功能。

为了使用`CursorLoader`向列表动态加载数据，这个活动实现了`LoaderCallbacks`接口。

```
public class ListViewLoader extends ListActivity
    implements LoaderManager.LoaderCallbacks<Cursor> {

    // This is the Adapter being used to display the list's data
    SimpleCursorAdapter mAdapter;

    // These are the Contacts rows that we will retrieve
    static final String[] PROJECTION = new String[]
    {ContactsContract.Data._ID,
        ContactsContract.Data.DISPLAY_NAME};

    // This is the select criteria
    static final String SELECTION = "( (" +
        ContactsContract.Data.DISPLAY_NAME + " NOTNULL) AND ( " +
        ContactsContract.Data.DISPLAY_NAME + " != '' ))";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Create a progress bar to display while the list loads
        ProgressBar progressBar = new ProgressBar(this);
        progressBar.setLayoutParams(new
        LayoutParams(LayoutParams.WRAP_CONTENT,
                    LayoutParams.WRAP_CONTENT, Gravity.CENTER));
        progressBar.setIndeterminate(true);
        getListView().setEmptyView(progressBar);

        // Must add the progress bar to the root of the layout
        ViewGroup root = (ViewGroup)
```

```

findViewById( android.R.id.content );
    root.addView( progressBar );

        // For the cursor adapter, specify which columns go into
which views
        String[] fromColumns = { ContactsContract.Data.DISPLAY_NAME };
        int[] toViews = { android.R.id.text1 }; // The TextView in
simple_list_item_1

        // Create an empty adapter we will use to display the loaded
data.
        // We pass null for the cursor, then update it in
onLoadFinished()
        mAdapter = new SimpleCursorAdapter( this,
            android.R.layout.simple_list_item_1, null,
            fromColumns, toViews, 0 );
        setListAdapter( mAdapter );

        // Prepare the loader. Either re-connect with an existing
one,
        // or start a new one.
        getLoaderManager().initLoader( 0, null, this );
    }

    // Called when a new Loader needs to be created
public Loader<Cursor> onCreateLoader( int id, Bundle args ) {
    // Now create and return a CursorLoader that will take care
of
    // creating a Cursor for the data being displayed.
    return new CursorLoader( this,
        ContactsContract.Data.CONTENT_URI,
            PROJECTION, SELECTION, null, null );
}

    // Called when a previously created loader has finished loading
public void onLoadFinished( Loader<Cursor> loader, Cursor data ) {
    // Swap the new cursor in. (The framework will take care of
closing the
    // old cursor once we return.)
    mAdapter.swapCursor( data );
}

    // Called when a previously created loader is reset, making the
data unavailable
public void onLoaderReset( Loader<Cursor> loader ) {
    // This is called when the last Cursor provided to
onLoadFinished()
        // above is about to be closed. We need to make sure we are
no
        // longer using it.
    mAdapter.swapCursor( null );
}

@Override
public void onListItemClick( ListView l, View v, int position,
long id ) {
    // Do something when a list item is clicked
}
}

```

注意：因为这个例子要向[Contacts Provider](#)请求查询数据，程序需要在制作清单文件中请求[READ_CONTACTS](#)权限：

```
<uses-permission  
    android:name="android.permission.READ_CONTACTS" />
```

来自“[index.php?title=List_View&oldid=9556](#)”



Grid View

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

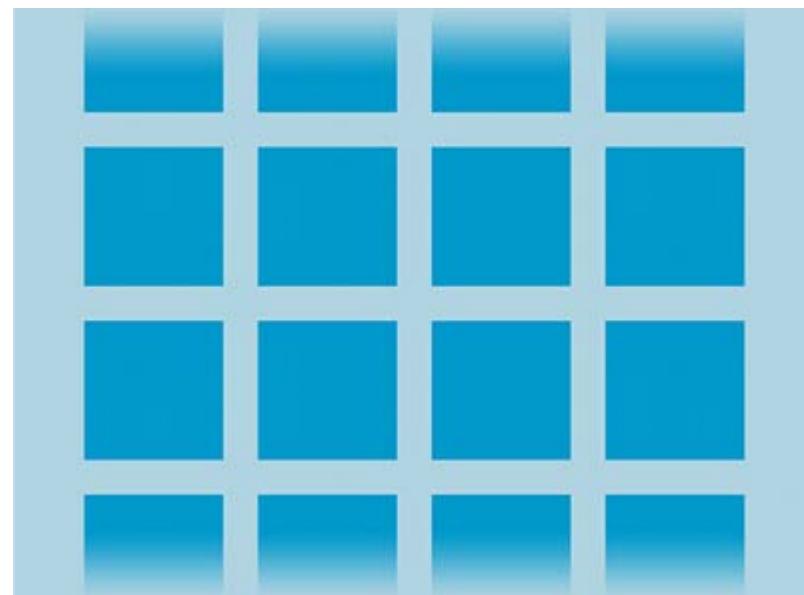
接：<http://developer.android.com/guide/topics/ui/layout/gridview.html>

作者：smilysas

更新时间：2012年8月2日

Grid

[GridView](#)是一个在可滚动的二维网格空间中显示项目的[ViewGroup](#)。元件会使用[ListAdapter](#)自动插入网格布局中。



例子

在本教程中，将创建一个缩略图网格。当一个元件被选中，会弹出显示该图像的位置的消息框。

1、首先创建一个名为HelloGridView的工程

2、找出一些将要使用的的图像，或者从样例图像中下载[

http://developer.android.com/shareables/sample_images.zip]。将准备好的图像放在工程的res/drawable/目录下。

3、打开 res/layout/main.xml文件，并插入以下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

创建的[GridView](#)将填满整个屏幕。这些属性的含义都很明显。更多关于属性的信息，请参阅的[GridView](#)的参考。

4、打开 HelloGridView.java 并在其中的[onCreate\(\)](#)函数中插入以下代码：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position,
            Toast.LENGTH_SHORT).show();
        }
    });
}
```

当main.xml文件完成内容布局以后，[GridView](#)会被[findViewById\(int\)](#)方法从

布局中捕捉到。[setAdapter\(\)](#)方法设置一个自定义的适配器

(ImageAdapter) 作为被显示在网格中的元件的源。ImageAdapter会在下一步中创建。

[setOnItemClickListener\(\)](#)会传递一个新的

[AdapterView.OnItemClickListener](#)消息以便于响应网格元件被选中的事件。匿名实例定义了由[<?>, android.view.View, int, long\)](#) [onItemClick\(\)](#)回调函数弹出一个[Toast](#)消息框显示所选中的网格元件的位置索引号 (索引号从零开始计算。在实际程序中可以通过位置索引号获取其全尺寸图像以备其他用途)。

5、创建一个扩展[BaseAdapter](#)并调用ImageAdapter的新类：

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the
    // Adapter
    public View getView(int position, View convertView, ViewGroup
parent) {
        ImageView imageView;
        if (convertView == null) { // if it's not recycled,
initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85,
85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }
        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
}
```

```
// references to our images
private Integer[] mThumbIds = {
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7,
    R.drawable.sample_0, R.drawable.sample_1,
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7,
    R.drawable.sample_0, R.drawable.sample_1,
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7
};

}
```

首先，实例化一些继承自[BaseAdapter](#)的必要函数。构造函数和[getCount\(\)](#)不用多解释。通常情况下，[getItem\(int\)](#)应该返回一个适配器中指定位置的真实对象，但是在本例中这点被忽略了。同样的，[getItemId\(int\)](#)应该返回元件的真实编号，但是本例中不需要这样。

第一个必须的方法是[android.view.View, android.view.ViewGroup](#) [getView\(\)](#)。这个方法为每一个加入到ImageAdapter的图像创建一个新的[View](#)视图。当调用它时，一个[View](#)视图对象会被传入并且是可重复使用的(在被调用至少一次以后)，所以需要确认对象是否为空。如果为空，就要实例化一个[ImageView](#)并根据要呈现的图像设置属性参数。

[setLayoutParams\(ViewGroup.LayoutParams\)](#) 设置视图的高度和宽度，这样可以确保不论原图像的大小如何都能适当的调整大小和裁减。

[setScaleType\(ImageView.ScaleType\)](#) 声明了图像将依照中心进行裁减(如果需要的话)。

[int, int, int\) setPadding\(int, int, int, int\)](#) 定义了各边如何进行填充。(需要注意的是，如果图像有不同的纵横比，那么当图像不匹配[ImageView](#)给定的尺寸时，较少的填充就会导致图像更多的裁减)。

如果传给[android.view.View, android.view.ViewGroup](#) [getView\(\)](#)的[View](#)视图不为空，则本地的[ImageView](#)会由可重复使用的[View](#)初始化。

在[android.view.View, android.view.ViewGroup](#) [getView\(\)](#)方法的最后，被

传入的position参数会用于从被作为[ImageView](#)资源的mThumbIds数组中选择图像。

剩下的就是定义绘画资源的mThumbIds数组。

6、运行程序。

可以通过调整[GridView](#)和[ImageView](#)的元素体验其使用方法。例如，使用[setAdjustViewBounds\(boolean\)](#)]而不使用[setLayoutParams\(ViewGroup.LayoutParams\)](#)。

来自“[index.php?title=Grid_View&oldid=9560](#)”



Input Controls

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/ui/controls.html>

译文地址：[Input_Controls](#)

翻译： fvn

更新日期：2012-07-28

输入控件 - Input Controls

输入控件是应用程序中用户接口的一种交互式组件。Android提供了大量的可供人们在UI中使用的控件，比如按钮、文本区域、(带滑块的)进度条、复选框、缩放按钮以及切换按钮等等。



在UI中增加输入控件就如同在XML布局中增加XNL元素一样简单。举例来说，下面为一个带有文本区域和按钮的布局。

```
<?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="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button android:id="@+id/button_send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
```

每个输入控件都支持一个特定的输入事件，如当用户输入文本或触摸一个按钮，这样你就可以处理事件。

常见控件 - Common Controls

下面是一些常见的控件的清单，您可以在您的应用程序中使用它们。点击下面的链接，以了解每个控件更多的使用。

注意：Android提供了比这里列出来的更多的控件，可以通过浏览android.widget包发现更多的控件。如果您的应用程序需要特定的输入控件，你可以建立自己的自定义组件。

Control Type	Description	Related Classes
按键	一个按钮，可以被用户按下或点击，以执行一个动作。	< Button >
文本区域	一个可编辑的文本区域。你可以使用AutoCompleteTextView小部件来创建一个文本输入部件，以提供自动完成建议	< EditText > , < AutoCompleteTextView >
复选框	一个可以由用户切换的ON/OFF开关。当提供给用户一组不互斥的可选项时，你应该使用复选框。	< CheckBox >

单选按钮	与复选框类似，但一组里只能选择一个选项。	< RadioGroup > < RadioButton >
开关按钮	带有亮度指示的ON/OFF的按钮	< ToggleButton >
下拉列表	一个下拉列表，允许用户从一组数据中选择一个值。	< Spinner >
选择器	一个对话框供用户通过使用向上/向下按钮或手势从一系列值中选择一个值。使用 DatePicker 部件来输入日期（月，日，年）或使用 TimePicker 部件输入时间（小时，分钟，上午/下午），这将被自动的格式化为用户区域的值。	< DatePicker >, < TimePicker >

来自“[index.php?title=Input_Controls&oldid=9450](#)”

Buttons

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址：<http://developer.android.com/guide/topics/ui/controls/button.html>

译文地址：[Buttons](#)

翻译： fvn

更新日期： 2012-07-28

目录

- [1 按键 - Buttons](#)
 - [1.1 点击事件的响应 - Responding to Click Events](#)
 - [1.1.1 使用监听器 - Using an OnClickListener](#)
 - [1.2 按钮的样式 - Styling Your Button](#)
 - [1.2.1 无边框按钮 - Borderless button](#)
 - [1.2.2 定制背景 - Custom background](#)

按键 - Buttons

按钮包括文字或者图标，或者两者兼而有之，当用户触摸到按钮时就会触发事件。



取决于你需要按钮有文本、图标或两者兼而有之，您可以以三种方式创建

按钮布局:

需要有文字的按钮，使用<[Button](#)>类：

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

需要有图标的按钮，使用<[ImageButton](#)>类：

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

需要有文字和图标的按钮，使用具有`android:drawableLeft`属性的<[Button](#)>类：

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

点击事件的响应 - Responding to Click Events

当用户点击一个按钮时，<[Button](#)>对象就会收到一个单击事件。

为了定义一个按钮的点击事件处理程序，在XML布局中为<`buttonandroid:onClick>属性。这个属性的值必须是你要调用的方法响应点击事件的名称。使用这个布局的<Activity>必须执行相应的方法。`

例如，这里有一个使用<[android:onClick](#)>属性的按钮的布局：

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

在使用这个布局的<[Activity](#)>中，利用下面的方法处理点击事件：

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

在<[android:onClick](#)>属性声明的方法必须完全按照上面显示。具体来说，该方法必须：

是公共的

返回void

定义一个<[View](#)>作为其唯一的参数（点击时将会看作这个<[View](#)>）

使用监听器 - Using an OnClick Listener

您也可以声明单击事件处理程序，而不是在XML布局中。这可能是必要的，如果你在运行时实例化<[Button](#)>，或者你需要在一<[Fragment](#)>子类中声明单击事件。

为了声明事件处理程序，创建一个<[View.OnClickListener](#)>对象，并通过调用<[setOnClickListener \(View.OnClickListener\)](#)>分配给按钮。例如：

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

按钮的样式 - Styling Your Button

按钮的外观（背景图片和字体）可能会因为机器不同而有所不同，因为不同厂家的设备的输入控件的默认样式往往不同。

您可以控制控件使用适用于整个应用程序的样式。例如，要确保所有运行Android4.0甚至更高版本的设备在您的应用程序使用Holo主题，需要在manifest的<[application](#)>元素中声

明android:theme="@android:style/Theme.Holo"。通过阅读博客文章得知，Holo Everywhere使用了Holo的主题，同时支持低版本的设备。

为了给按钮定制不同的背景，指定`<android:background>`属性为可绘制或彩色的资源。另外，您可以使用一种类似于HTML的样式来定义按钮的样式，可以定义多种属性，如背景、字体、大小等等。关于应用样式的更多信息，请参阅[Styles and Themes](#)。

无边框按钮 - Borderless button

一种有用的设计是无边框按钮。无边框按钮与基本按钮相似，但是无边框按钮没有无边框或背景，但在不同状态如点击时，会改变外观。

要创建一个无边框“按钮”，为按钮应用`<borderlessButtonStyle>`样式。例如：

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    style="?android:attr/borderlessButtonStyle" />
```

定制背景 - Custom background

如果你想真正重新定义按钮的外观，你可以指定一个自定义的背景。而不是提供一个简单的位图或颜色，然而，你的背景应该是一个状态列表资源，取决于按钮的当前状态而改变外观，。

您可以在一个XML文件中定义状态列表，定义三种不同的图像或颜色，用于不同的按钮状态。

要为按钮的背景创建一个状态列表资源：

创建三个按钮背景位图以表示默认、按下和选中的按钮状态。

为了确保图像适合不同大小的按钮，以[9-patch](#)的格式创建图像。

位图放到你工程的res/drawable/ directory。确保每个位图命名正确能够反映它们分别代表的按钮状态，如button_default.9.png、button_pressed.9.png以及button_focused.9.png。

在res/drawable/ directory下创建一个新的XML文件（命名类似于button_custom.xml）。使用下面的XML：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_pressed"
          android:state_pressed="true" />
    <item android:drawable="@drawable/button_focused"
          android:state_focused="true" />
    <item android:drawable="@drawable/button_default" />
</selector>
```

这定义一个单一的绘制资源，将基于按钮的当前状态改变其图像。

第一个<item>定义了按下按钮（激活）时使用的位图。

第二个<item>定义了按钮按下时（按钮高亮时，使用轨迹球或方向键）使用的位图。

第三个<item>定义了默认状态下的按钮（既不是按下也不是选中）使用的位图。

注意：<item>元素的顺序是重要的。当图像可用时，按顺序遍历<item>元素，以确定哪一个适合当前按钮状态。因为默认的位图在最后，当android:state_pressed 和 android:state_focused 的值都为false时才会被应用。

现在这个XML文件代表一个单一的绘制资源，当<Button>引用它作为背景，图像将基于按钮的三种状态而改变。

然后，只需应用此XML文件作为按钮的背景：

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    android:background="@drawable/button_custom" />
```

对于此XML语法的更多详细信息，包括如何定义一个非使能、不确定的或其

他的按钮状态，阅读<[State List Drawable](#)>。

来自“[index.php?title=Buttons&oldid=8777](#)”

Text Fields

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Monica

主任务原文链

接：<http://developer.android.com/guide/topics/ui/controls/text.html>

目录

[[隐藏](#)]

[1 Text Fields-文本框](#)

- [1.1 Specifying the Keyboard Type-指定键盘类型](#)
 - [1.1.1 Controlling other behaviors-控制其他行为](#)
- [1.2 Specifying Keyboard Actions-指定键盘操作](#)
 - [1.2.1 Responding to action button events-响应按钮事件](#)
 - [1.2.2 Setting a custom action button label-设置文本框标签](#)
- [1.3 Adding Other Keyboard Flags-添加其他键盘标志](#)
- [1.4 Providing Auto-complete Suggestions-提供自动完成建议](#)

Text Fields-文本框

A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

文本框允许用户在应用程序中输入文本。它们可以是单行的，也可以是多行的。点击文本框后显示光标，并自动显示键盘。除了输入，文本框还包

含其它操作，比如文本选择（剪切，复制，粘贴）以及数据的自动查找功能。

You can add a text field to you layout with the `EditText` object. You should usually do so in your XML layout with a `<EditText>` element. 你可以使用`EditText`对象在布局中添加一个文本字段， android里的写法通常是在XML布局文件中添加`<EditText>`元素



Specifying the Keyboard Type-指定键盘类型

Text fields can have different input types, such as number, date, password, or email address. The type determines what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.

文本字段可以有不同的输入类型，如数字，日期，密码，或电子邮件地址。类型确定文本框内允许输入什么样的字符，可能会提示虚拟键盘调整其布局来显示最常用的字符。

You can specify the type of keyboard you want for your `EditText` object with the `android:inputType` attribute. For example, if you want the user to input an email address, you should use the `textEmailAddress` input type:

你可以在`EditText`对象使用`Android:inputType`属性指定输入类型的键盘，例如：你想输入一个电子邮件地址上的用户，`inputType`属性应为`textEmailAddress`:



图1:默认的文字输入类型。

```
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

There are several different input types available for different situations. You can find them all listed with the documentation for `android:inputType`

针对不同的情况有几种不同的输入类型。你可以找到所有的文件中列出的`android:inputType`属性

"text"普通文本键盘

"textEmailAddress"带有@字符的普通文本键盘

"textUri"带有/字符的普通文本键盘

"number"基本数字键盘

"phone"手机式键盘



图2:textEmailAddress输入类型。

Tip: To allow users to input long strings of text with line breaks, use the "textMultiLine" input type. By default, an EditText object is restricted to one line of text and scrolls horizontally when the text exceeds the available width.

提示：为了让用户输入长文本字符串时换行，使用的“textMultiLine”属性。默认情况下，一个编辑文本对象仅限于一行文本和水平滚动文本时超过可用宽度。



图3:手机输入类型。

Controlling other behaviors-控制其他行为

android: inputType还允许您指定操作行为，如在某些键盘上是否要利用所有新词，或使用自动完成和拼写建议功能。

inputType的属性允许按位组合，让您可以一次指定一个键盘布局和一个或多个操作行为。

下面是一些定义键盘行为的普通输入值：

- "textCapSentences"每个句子首字母大写
- "textCapWords"每个词语均大写，适用于标题或人名
- "textAutoCorrect"纠正常见单词的错误拼写
- "textPassword"输入的字符转出点
- "textMultiLine"允许用户输入包括换行符（回车）在内的长字符串

例如，你如何收集邮政地址，利用每一个字，并禁用文字的行为：

```
<EditText
    android:id="@+id/postal_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/postal_address_hint"
    android:inputType="textPostalAddress|
        textCapWords|
        textNoSuggestions" />
```

All behaviors are also listed with the android:inputType documentation.
还列出了所有的行为与Android的：inputType相关的文件。

Specifying Keyboard Actions-指定键盘操作

In addition to changing the keyboard's input type, Android allows you to specify an action to be made when users have completed their input. The action specifies the button that appears in place of the carriage return

key and the action to be made, such as "Search" or "Send."

除了改变键盘的输入类型，当用户完成输入时，android允许你指定特殊的按钮进行相应的操作，如把回车键作为“搜索”或“发送”操作。



图4：如果你声明的Android imeOptions = “actionSend” , 键盘包括发送的行动。

You can specify the action by setting the android:imeOptions attribute.

For example, here's how you can specify the Send action:

您可以通过android:imeOptions属性设置指定的动作。例如，这里你可以指定发送的行为：

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```

If you do not explicitly specify an input action then the system attempts to determine if there are any subsequent android:focusable fields. If any focusable fields are found following this one, the system applies the {@code actionNext} action to the current EditText so the user can select Next to move to the next field. If there's no subsequent focusable field, the system applies the "actionDone" action. You can override this by setting the android:imeOptions attribute to any other value such as "actionSend" or "actionSearch" or suppress the default behavior by using the "actionNone" action.

如果你不明确指定一个输入动作，然后系统将尝试确定是否有任何后续的android:focusable属性动作。如果发现了有 android:focusable属性动作，那么这个系统适用于在当前的EditText的 {@code actionNext} 行动，使用户可以选择“下一步”或移动到下一个字段。如果是没有后续的focusable属性，那该系统适用“actionDone”动作，你也可以通过设置Android:imeOptions属性使系统更改到其它值，

如“actionSend”或“actionSearch”或禁止使用“actionNone”动作的默认行为。

Responding to action button events-响应按钮事件

If you have specified a keyboard action for the input method using android:imeOptions attribute (such as "actionSend"), you can listen for the specific action event using an TextView.OnEditorActionListener. The TextView.OnEditorActionListener interface provides a callback method called onEditorAction() that indicates the action type invoked with an action ID such as IME_ACTION_SEND or IME_ACTION_SEARCH.

如果您已指定键盘采用Android: imeOptions属性 (“actionSend”等) 的操作方法，你可以使用 TextView.OnEditorActionListener监听事件行为。TextView.OnEditorActionListener接口提供了一个回调方法onEditorAction ()，它通过输入的动作ID，如IME_ACTION_SEND或IME_ACTION_SEARCH行为调用相关的动作类型方法。

For example, here's how you can listen for when the user clicks the Send button on the keyboard:

例如，你可以监听用户点击键盘上的发送按钮：

```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionBarId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            // Send the user message
            handled = true;
        }
        return handled;
    }
});
```

Setting a custom action button label-设置文本框标签

If the keyboard is too large to reasonably share space with the underlying application (such as when a handset device is in landscape orientation) then fullscreen ("extract mode") is triggered. In this mode, a labeled action button is displayed next to the input. You can customize the text of this button by setting the android:imeActionButton attribute:

如果键盘太大，系统将会合理分担余下的应用程序（例如，当手机设备是横向的）空间，然后全屏（“提取模式”）被触发。在这种模式下，将在输入框旁边显示按钮，你可以通过设置imeActionButton属性来定制按钮的文本。

```
<EditText
    android:id="@+id/launch_codes"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/enter_launch_codes"
    android:inputType="number"
    android:imeActionButton="@string/launch" />
```



Figure 5. A custom action label with android:imeActionButton
android:imeActionButton属性案例

Adding Other Keyboard Flags-添加其他键盘标志

In addition to the actions you can specify with the android:imeOptions attribute, you can add additional flags to specify other keyboard behaviors. All available flags are listed along with the actions in the android:imeOptions documentation.

你可以指定Android: imeOptions属性添加另外的行为，你可以添加额外的标志，以指定其他键盘行为。在Android: imeOptions文件列出所有可用的标志。

For example, figure 5 shows how the system enables a fullscreen text field when a handset device is in landscape orientation (or the screen

space is otherwise constrained for space). You can disable the fullscreen input mode with flagNoExtractUi in the android:imeOptions attribute, as shown in figure 6.

例如，图5显示的是当手机设备在横向屏幕上使用全屏文本字段时（或在屏幕上限制键盘空间）。您可以在Android: imeOptions属性设置flagNoExtractUi去禁用全屏输入模式，如图6所示。



Figure 6. The fullscreen text field ("extract mode") is disabled with android:imeOptions="flagNoExtractUi".

android:imeOptions="flagNoExtractUi"的提取模式

Providing Auto-complete Suggestions-提供自动完成建议

If you want to provide suggestions to users as they type, you can use a subclass of EditText called AutoCompleteTextView. To implement auto-complete, you must specify an (@link android.widget.Adapter) that provides the text suggestions. There are several kinds of adapters available, depending on where the data is coming from, such as from a database or an array.

如果你想向用户键入提供建议，您可以使用EditText的子类AutoCompleteTextView控件。为了实现自动完成，你必须指定一组（@link android.widget.Adapter）文字提供建议。有几种可用的适配器，匹配数据项，如从数据库或一个数组获取所需要匹配值。



Figure 7. Example of AutoCompleteTextView with text suggestions
AutoCompleteTextView控件案例

The following procedure describes how to set up an AutoCompleteTextView that provides suggestions from an array, using ArrayAdapter:

下面的过程介绍了如何设置一个AutoCompleteTextView并使用ArrayAdapter适配器配置里面的所需的数组：

1.Add the AutoCompleteTextView to your layout. Here's a layout with only the text field:

添加AutoCompleteTextView到您的布局。这里是只是一个文本字段的布局

```
<?xml version="1.0" encoding="utf-8"?>
<AutoCompleteTextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

2.Define the array that contains all text suggestions. For example, here's an array of country names that's defined in an XML resource file (res/values/strings.xml):

定义一个数组，里面包含数组所需的子值。例如，这里是一个国家的名字，这是定义在一个XML资源文件 (res/values/strings.xml) 数组：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Afghanistan</item>
        <item>Albania</item>
        <item>Algeria</item>
        <item>American Samoa</item>
        <item>Andorra</item>
        <item>Angola</item>
        <item>Anguilla</item>
        <item>Antarctica</item>
        ...
    </string-array>
</resources>
```

3.In your Activity or Fragment, use the following code to specify the adapter that supplies the suggestions:

在Activity中或Fragment中，建议使用下面的代码操作适配器

```
// Get a reference to the AutoCompleteTextView in the layout
```

```
AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.autocomplete_country);
// Get the string array
String[] countries =
getResources().getStringArray(R.array.countries_array);
// Create the adapter and set it to the AutoCompleteTextView
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, countries);
textView.setAdapter(adapter);
```

Here, a new ArrayAdapter is initialized to bind each item in the COUNTRIES string array to a TextView that exists in the simple_list_item_1 layout (this is a layout provided by Android that provides a standard appearance for text in a list).

代码中，创建一个countries的数组，并将ArrayAdapter适配器初始化，且将其绑定到simple_list_item_1文件布局中（这是由Android提供一个文本框绑定数据的文本列表）。

Then assign the adapter to the AutoCompleteTextView by calling setAdapter().

然后AutoCompleteTextView属性调用setAdapter () 方法，将适配器添加其中。

来自 "[index.php?title=Text_Fields&oldid=13855](#)"



Checkboxes

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Monica

原文链

接：<http://developer.android.com/guide/topics/ui/controls/checkbox.html>

CheckBox

Checkboxes allow the user to select one or more options from a set.

Typically, you should present each checkbox option in a vertical list.

复选框允许用户从列表中选择一个或多个选项。通常，你应该在垂直的列表中显示每一个选项。



To create each checkbox option, create a CheckBox in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

当你要创建一个复选框时，你就必须要在你的布局文件中创建一个CheckBox字段。因为一组复选框选项允许用户选择多个选项，而且每个复选框分开管理，你必须为每一个选项注册点击监听器。

Responding to Click Events- 响应单击事件

When the user selects a checkbox, the CheckBox object receives an on-click event.

当用户选择一个复选框中的选项时，该复选框对象接收onClick事件

To define the click event handler for a checkbox, add the android:onClick attribute to the <CheckBox> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

定义一个复选框的Click事件处理程序，需要在XML布局文件中的<CheckBox>元素中添加android: onClick属性。这个属性的值必须是你调用的方法响应click事件的名称。Activity界面将会响应该事件方法。

For example, here are a couple CheckBox objects in a list:

例如，这里有几组Checkbox对象列表：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked" />
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked" />
</LinearLayout>
```

Within the Activity that hosts this layout, the following method handles the click event for both checkboxes:

在Activity中，将会响应Checkboxes字段中设置的方法

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
```

```

boolean checked = ((CheckBox) view).isChecked();

// Check which checkbox was clicked
switch(view.getId()) {
    case R.id.checkbox_meat:
        if (checked)
            // Put some meat on the sandwich
        else
            // Remove the meat
        break;
    case R.id.checkbox_cheese:
        if (checked)
            // Cheese me
        else
            // I'm lactose intolerant
        break;
    // TODO: Veggie sandwich
}
}

```

The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must:
在方法中你必须声明android:onClick属性的方法，你在android声明：
onClick属性必须严格按照上面显示的属性。具体来说，该方法必须：

- Be public
- 公共的
- Return void
- 返回值为空
- Define a View as its only parameter (this will be the View that was clicked)
- 定义一个视图作为其唯一的参数（这将被点击查看）

Tip: If you need to change the radio button state yourself (such as when loading a savedCheckBoxPreference), use the `setChecked(boolean)` or `toggle()` method.

提示：如果你需要改变自己的单选按钮状态（如当加载一个保存CheckBoxPreference），使用`setChecked (boolean)` 或`toggle ()` 方法。

Radio Buttons

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：sumakira

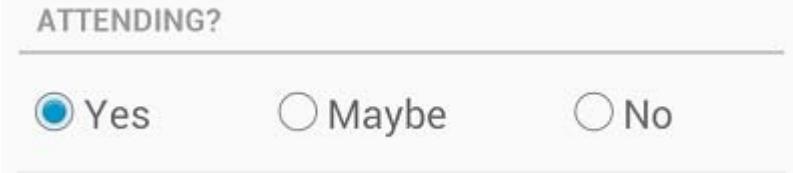
原文链

接：

<http://developer.android.com/guide/topics/ui/controls/radiobutton.html>

Radio Buttons 单选按钮

单选按钮允许用户从一系列选项中选择其中某个选项。如果认为有必要让用户看到所有并列的可选项，并且各个选项中只能有一个被选择，那么单选按钮是个好选择。如果没有必要显示所有的并列选项，那么可以



用[spinner](#)下拉列表代替。

创建单个选项之前，需要在布局文件中创建选项按钮[RadioButton](#)。然而，因为单选按钮之间是互斥的，它们需要被放在同一个选项组[RadioGroup](#)里。这样系统会认为在同一个选项组里每次只能有一个选项被选择。

响应点击事件

当用户选择其中一个选项后，相应的选项按钮对象会收到on-click的事件。

通过在XML布局文件中为<RadioButton>元素添加[android:onClick](#)属性，可

以为按钮定义点击事件的响应处理器。属性的值必须要和你想调用来自响应这个点击事件的方法名保持一致。所在的[Activity](#)需要实现此响应方法。

例如：这有一对[RadioButton](#)选项按钮对象：

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked" />
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked" />
</RadioGroup>
```

注意：[RadioGroup](#) 是 [LinearLayout](#) 的子类，因此默认情况下它拥有垂直方向的布局属性。

在包含这种布局的[Activity](#)内，下面的方法可以捕获并处理这两个单选按钮的点击事件

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = (RadioButton) view.isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
            break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
            break;
    }
}
```

在[android:onClick](#)属性中定义的方法必须有如上方法的特点。尤其特别指出，此方法必须：

- 访问权限是Public
- 返回类型为void
- 定义一个View为其唯一参数（这个View是要被点击的那一个）

提示：如果你想自己改变单选按钮的状态（例如加载一个已保存过的CheckBoxPreference），那么你可以使用 setChecked(boolean) 或者 toggle() 方法

来自“[index.php?title=Radio_Buttons&oldid=13773](#)”



Toggle Buttons

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：sumakira

分任务原文链接

一： <http://developer.android.com/guide/topics/ui/controls/togglebutton.html>

Toggle Buttons 开关按钮

开关按钮允许用户在两种状态间进行切换以改变设置效果。

你可以使用[ToggleButton](#)对象把一个基本的开关按钮添加到布局文件。Android 4.0 (API 级别14) 引进了另外一种叫做[Switch](#)的开关按钮，这种按钮提供了滑动控制的效果，你可以使用[Switch](#)对象添加到布局文件来实现。



[ToggleButton](#)和[Switch](#)控件都是[CompoundButton](#)组合按钮的子类并且有着相同的功能，所以你可以用同样的方法来实现他们功能。

响应点击事件

当用户选择[Toggle Buttons](#)和[Switch](#)时，相应的对象就会接受到一个点击事件。

在你的XML布局文件中的<ToggleButton>或<Switch>元素里添加[android:onClick](#)属性,可以定义这个点击事件的响应操作。属性的值必须要和你想调用来响应这个点击事件的方法名保持一致。所在的[Activity](#)需要实现此响应方法。

例如，这里有一个[ToggleButton](#)的例子并且设置了[android:onClick](#)属性：

```
<ToggleButton
    android:id="@+id/togglebutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Vibrate on"
    android:textOff="Vibrate off"
    android:onClick="onToggleClicked" />
```

在持有这个布局的Activity里，下边的方法处理点击事件：

```
public void onToggleClicked(View view) {
    // Is the toggle on?
    boolean on = ((ToggleButton) view).isChecked();

    if (on) {
        // Enable vibrate
    } else {
        // Disable vibrate
    }
}
```

在[android:onClick](#)属性中定义的方法必须有如上方法的特点。

尤其特别指出，此方法必须：

- 访问权限是Public
- 返回类型为void
- 定义一个[View](#)为其唯一参数（这个[View](#)是要被点击的那个）

提示：如果你需要自己改变按钮的状态，那么你可以使用[setChecked\(boolean\)](#)或者[toggle\(\)](#)方法改变状态。

使用**OnCheckedChangeListener**监听器

你也可以通过代码来声明点击事件的处理器来代替在XML布局里进行设置。如果你在运行时去实例化一个[ToggleButton](#)或者[Switch](#)对象，又或者你需要在[Fragment](#)的子类里定义点击动作时，这么做就显得十分必要了。

要用代码来声明点击事件的处理器，创建一个[CompoundButton.OnCheckedChangeListener](#)对象，并且通过调

用[setOnCheckedChangeListener\(CompoundButton.OnCheckedChangeListener\)](#)

将它绑定到按钮上。例如：

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

来自“[index.php?title=Toggle_Buttons&oldid=13774](#)”



Spinners

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

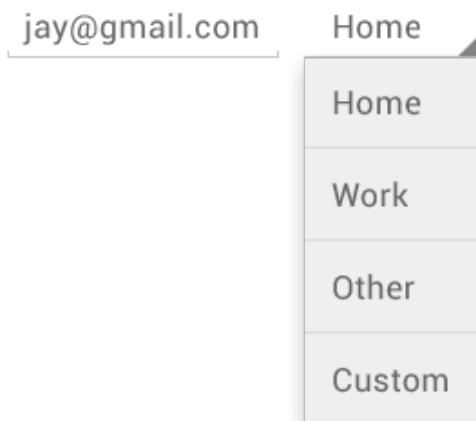
负责人: sumakira

分任务原文链接

二: <http://developer.android.com/guide/topics/ui/controls/spinner.html>

Spinners

Spinners提供一条快速的方式来从一系列的选项中选取某值。默认的状态下，一个Spinner只显示当前被选择的值。通过触屏点击Spinner可以显示一个dropdown的下拉列表菜单并且显示出来了所有可选的值，从这个列表中，用户可以选取一个新的值。



你可以添加一个[spinner](#)对象到布局文件中。这通常都是通过在XML的布局文件中的<Spinner>元素中进行设置来实现的。例如：

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

要加入选项列表到spinner的话，你要先在你的[Activity](#)或者是[Fragment](#)的代码里指定一个[SpinnerAdapter](#)。

添加用户选项给**Spinner**

提供给**Spinner**的选项可以是源自任何地方，但是必须要通过[SpinnerAdapter](#)来提供。比如说如果选项可以放在一个array里，那就通过[ArrayAdapter](#)来提供数据，在比如说如果选项是来自数据库，那就可以选用[CursorAdapter](#)来提供。

举例，如果提供给**Spinner**的可选项已经是事先决定好的，那可以通过在[string resource file](#)字符串资源文件中定义一组string array字符串数组

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

通过使用这样一个数组，在加上把下边这段代码应用到[Activity](#)或者[Fragment](#)里来提供给spinner一个实例化的[ArrayAdapter](#)数组适配器

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

可以用[int, int\) createFromResource\(\)](#)函数从字符串数组中来创建[ArrayAdapter](#)数组适配器。第三个参数是定义了被选项如何显示在spinner控件中布局的资源文件。[simple_spinner_item](#)布局文件是由平台提供的默认布局，除非自己定义了spinner的样式布局文件。

之后需要调用[setDropDownViewResource\(int\)](#)函数来指定适配器来显示spinner的选项列表 ([simple_spinner_dropdown_item](#)是平台定义的另外一个标准的布局)

调用[setAdapter\(\)](#)函数来绑定适配器给Spinner

响应用户选项

当用户从drop-down下拉列表中选择其中一个选项，[Spinner](#) 对象将会收到一个on-item-selected事件。

要给spinner定义选择的动作事件，继承[AdapterView.OnItemSelectedListener](#)接口和实现相应的[onItemSelected\(\)](#)回调函数。例如，这有一个继承了此接口的[Activity](#)：

```
public class SpinnerActivity extends Activity implements
OnItemSelectedListener {
    ...

    public void onItemSelected(AdapterView<?> parent, View view,
        int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

[AdapterView.OnItemSelectedListener](#)需要实现[onItemSelected\(\)](#)函数和[onNothingSelected\(\)](#)回调函数

之后你要通过调用实现[setOnItemSelectedListener\(\)](#)来实现调用接口：

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

如果你的[Activity](#) 或者[Fragment](#) 继承了[AdapterView.OnItemSelectedListener](#)接口（如上所示），你可以传递this来做为参数。

来自 "[index.php?title=Spinners&oldid=9073](#)"



Pickers

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址：<http://developer.android.com/guide/topics/ui/controls/pickers.html>

译文地址：[Pickers](#)

翻译： fvn

更新日期： 2012-07-30

目录

- [1 选择器 - Pickers](#)
- [2 创建一个时间选择器 - Creating a Time Picker](#)
 - [2.1 继承DialogFragment，创建时间选择器 - Extending DialogFragment for a time picker](#)
 - [2.2 显示时间选择器 - Showing the time picker](#)
- [3 创建数据选择器 - Creating a Date Picker](#)
 - [3.1 继承DialogFragment，创建数据选择器 - Extending DialogFragment for a data picker](#)
 - [3.2 显示数据选择器 - Showing the data picker](#)

选择器 - Pickers

Android给用户提供了选择时间或日期的对话框控件。每个选择器提供了选择时间（小时，分钟，上午/下午）或日期（月，日，年）的控件。使用这

些选择器有助于确保用户可以选择一个有效的、格式正确的时间或日期，并自动调整到用户的所在区域。



我们建议您使用**DialogFragment**承载每个时间或日期选择器。

DialogFragment能够自动管理对话框的生命周期，并且允许您在不同的布局配置中显示选择器，如在手机上的基本对话框，或大屏幕上的布局嵌入的一部分。

虽然**DialogFragment**首次加入是在Android 3.0平台（API级别11），如果您的应用程序支持的Android版本低于3.0，甚至是Android 1.6版本，你也可以使用一个向后兼容的支持库来使用**DialogFragment**类。

注意：下面的代码显示了如何创建和使用一个时间和日期选择器，它使用了**DialogFragment**的支持库API。如果应用程序的minSdkVersion是11或更高，你可以直接使用**DialogFragment**的平台版本。

创建一个时间选择器 - Creating a Time Picker

为了使用**DialogFragment**来显示**TimePickerDialog**，你需要定义一个片段类，它扩展了**DialogFragment**，并且通过onCreateDialog（）方法返回一个**TimePickerDialog**。

注意：如果您的应用程序支持低于Android3.0的版本，确保你的Android项目已经设置了支持库。

继承**DialogFragment**， 创建时间选择器 -

Extending DialogFragment for a time picker

为了定义一个用于TimePickerDialog的DialogFragment，你必须：

定义onCreateDialog()的方法以实例化TimePickerDialog

实现TimePickerDialog.OnTimeSetListener接口，以便当用户设置时间时实现回调。

下面是一个例子：

```
public static class TimePickerFragment extends DialogFragment
    implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour,
minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int
minute) {
        // Do something with the time chosen by the user
    }
}
```

要知道参数信息，请查看TimePickerDialog类。

现在所需要做的就是将fragment在activity中实例化。

显示时间选择器 - Showing the time picker

一旦你定义了一个如上所示的DialogFragment，您可以通过实例化DialogFragment并调用Show()来显示时间选择器。

例如，这里有一个按钮，当点击的时候，就会调用方法来显示对话框：

```
<Button
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />
```

当用户点击这个按钮时，就会调用下面的方法：

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

此方法在DialogFragment实例化后就调用show()。show()方法需要实例化FragmentManager，每个fragment都要有一个独一无二的tag name。

注意：如果您的应用程序支持低于Android3.0版本的，需要调用getSupportFragmentManager () 实例化FragmentManager。另外，还要确保显示时间选择器的activity要继承FragmentActivity而不是标准的Activity。。

创建数据选择器 - Creating a Date Picker

创建DatePickerDialog 类似于创建TimePickerDialog。唯一的区别是为你为fragment而创建的对话框不同。

为了使用DialogFragment来显示DatePickerDialog，你需要定义一个继承DialogFragment的fragment类并通过onCreateDialog()方法返回DatePickerDialog。

注意：如果您的应用程序支持低于Android3.0的版本，确保你的Android项目已经设置了支持库。

继承DialogFragment， 创建数据选择器 -

Extending DialogFragment for a data picker

为了定义一个用于DatePickerDialog的DialogFragment，你必须：

定义onCreateDialog()的方法以实例化DatePickerDialog

实现 DatePickerDialog.OnDateSetListener 接口，以便当用户设置时间时实现回调。

下面是一个例子：

```
public static class DatePickerFragment extends DialogFragment
    implements DatePickerDialog.OnDateSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year,
month, day);
    }
    public void onDateSet(DatePicker view, int year, int month, int
day) {
        // Do something with the date chosen by the user
    }
}
```

要知道参数信息，请查看DatePickerDialog类。

现在所需要做的就是将fragment在activity中实例化。

显示数据选择器 - Showing the data picker

一旦你定义了一个如上所示的DialogFragment，您可以通过实例化DialogFragment并调用Show()来显示数据选择器。

例如，这里有一个按钮，当点击的时候，就会调用方法来显示对话框：

```
<Button
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/pick_date"
    android:onClick="showDatePickerDialog" />
```

当用户点击这个按钮时，就会调用下面的方法：

```
public void showDatePickerDialog (View v) {
    DialogFragment newFragment = new DatePickerFragment ();
    newFragment.show (getSupportFragmentManager (), "datePicker");
}
```

此方法在DialogFragment实例化后就调用show()。show()方法需要实例化FragmentManager，每个fragment都要有一个独一无二的tag name。

注意：如果您的应用程序支持低于Android3.0版本的，需要调用getSupportFragmentManager () 实例化FragmentManager。另外，还要确保显示时间选择器的activity要继承FragmentActivity而不是标准的Activity。。

来自“[index.php?title=Pickers&oldid=9453](#)”

Input Events

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/ui/ui-events.html>

翻译者： jcccn

更新时间： 2012年7月19日

目录

[[隐藏](#)]

[1 Input Events](#)

- [1.1 事件监听器](#)
- [1.2 事件处理器](#)
- [1.3 触摸模式](#)
- [1.4 处理焦点](#)

Input Events

在Android中有多种方法可以用来拦截用户与程序的交互事件。如果想处理用户界面中触发的事件，可以通过从用户交互的View捕获事件来实现。View这个类提供了这些方法。

在用来构成布局的各种View类中，我们可以看到有几个用于UI事件的公共回调方法。当这些对象中有用户行为产生时，Android框架就会调用相应的回调方法。例如，当一个view(比如一个按钮Button)被触摸了，那么它的onTouchEvent()方法就会被调用。但是，为了拦截到这个事件，你必须扩展这个类并重写这些方法。然而，为了处理这样一个事件就扩展每一个View对象是不实际的。这就是为什么View类还提供了包含一些很方便定

义的回调方法的嵌套接口的原因。这些叫做事件监听器(event listener)接口就是你拦截UI交互事件的入口。

虽然多数情况下使用事件监听器来监听用户交互，但是有时候为了创建一个自定义的组件，也需要扩展一个View类。比如你可能想要扩展一个Button类来使它更好用。这种情况下，你就可以使用事件处理器(event handler)来定义默认的事件行为。

事件监听器

一个事件监听器是View类中一个包含单一回调方法的接口。当注册了监听器的View发生了跟监听器对应的UI交互事件时，Android框架就会调用这些回调方法。

事件监听器接口中包含了以下回调方法：

onClick()

来自View.OnClickListener接口。当用户触摸了一个对象（在触摸模式下），或者通过导航键或轨迹球使它获得焦点后再按下兼容"enter"的按键或是按下轨迹球，这个方法被调用。

onLongClick()

来自View.OnLongClickListener接口。当用户在一个对象上触摸并按住不放（触摸模式下），或者通过导航键或轨迹球使它获得焦点后再按下兼容"enter"的按键或是轨迹球并按住不放（一秒钟时间），这个方法被调用。

onFocusChange()

来自View.OnFocusChangeListener。当用户使用导航键或者轨迹球导航到一个对象或离开一个对象时，这个方法被调用。

onKey()

来自View.OnKeyListener接口。当用户让焦点落在一个对象上后按下或释放设备上的一个键时，这个方法被调用。

onTouch()

来自**View.OnTouchListener**接口，当用户执行一个触屏事件的动作时，包括按下动作、释放动作，或者任何在屏幕上的手势（当然必须在对象所在的区域内），这个方法被调用。

onCreateContextMenu()

来自**View.OnCreateContextMenuListener**接口。当一个上下文菜单被创建（长按后的结果）时，这个方法被调用。关于上下文菜单的更多讨论，请参考关于[菜单 - Menus](#)的开发指南。

这些方法是各自接口的唯一成员。如果要定义这些方法来处理事件，就要在**Activity**中实现这些嵌套接口，或者定义一个匿名内部类。然后，把你实现的实例传递给对应的**View.set...Listener()**方法。（例如，以实现的**OnClickListener**作为参数调用 **setOnClickListener()**方法）。

下面的例子演示了如何给一个**Button**注册一个**on-click**监听器：

```
// 创建一个实现了OnClickListener的匿名内部类的对象
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ...
    // 从布局中找到需要的按钮
    Button button = (Button) findViewById(R.id.corky);
    // 用上面实现的对象注册点击监听器
    button.setOnClickListener(mCorkyListener);
    ...
}
```

你可能也发现把**OnClickListener**作为**Activity**的一部分来实现会更方便。这样可以避免额外的类加载和对象内存分配。如：

```
public class ExampleActivity extends Activity implements
OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        ...
        Button button = (Button) findViewById(R.id.corky);
        button.setOnClickListener(this);
    }
}
```

```
// 实现OnClickListener的回调
public void onClick(View v) {
    // 这里做按钮点击后需要做的事情
}
...
```

注意上例中的`onClick()`回调方法没有返回值，但是有些其他的事件监听器方法必须要返回一个布尔值。具体情况取决于事件。以下是针对部分情况进行说明：

- `onLongClick()` - 这个方法返回一个布尔值，用来指明这个事件是否已经被你消耗(使用)了而不应进一步传递。也就是说，如果返回`true`，表明你已经处理了这个事件，并且事件应该就此停止；如果返回`false`，表明你没有处理这个事件，并且/或者这个事件应该继续传递给其他的`on-click`监听器来处理。
- `onKey()` - 这个方法返回一个布尔值，用来指明这个事件是否已经被你消耗了而不应进一步传递。也就是说，如果返回`true`，表明你已经处理了这个事件，并且事件应该就此停止；如果返回`false`，表明你没有处理这个事件，并且/或者这个事件应该继续传递给其他的`on-key`监听器来处理。
- `onTouch()` - 这个方法返回一个布尔值，用来指明你的监听器是否消耗这个事件。重要的是这个事件可能包含多个彼此跟随的动作。因此，如果在接收到按下动作事件时返回了`false`，表明你不使用这个事件，并且对这个事件中按下动作的后续动作也不感兴趣了。这样的话，你将不会再为这个事件中的任何其他动作调用这个方法，比如手势或者结束动作事件。

记住，硬件按键事件始终是发送给当前焦点所在的`View`。它们从`View`层次结构的最顶层开始从上而下派发，直到到达合适的目标。如果你当前的`View`（或`View`的子视图）有焦点，那么通过`dispatchKeyEvent()`方法你能够看到事件的派发路线。通过`View`捕获按键事件的另一个方法是，你可以在`Activity`内部通过`onKeyDown()`和`onKeyUp()`两个方法来接受所有的按键事件。

而且，当考虑到程序的文本输入的时候，请注意许多设备只有软键盘输入法。这些输入法不要求必须有实体按键；一些设备可能使用语音输入、手写输入等等。即使一个输入法展现了像实体键盘一样的界面，通常它也不会

触发**onKeyDown()**一类的事件。你永远不应该让你的UI依赖特定的按键来实现操作，除非你想将你的程序限制在拥有实体键盘的设备上。尤其，不要依赖在这些按键上按下**return**键来确认输入；而应该用**IME_ACTION_DONE**一类的动作来表示输入法完成了程序中你想要的动作，以便程序用应有的方式来改变UI。不用去猜测软键盘工作的方式，只相信它会为程序提供格式化的文本就够了。

注意：Android会首先调用事件处理器，然后再调用类定义的合适的默认处理程序。这样，如果这些事件监听器返回了**true**，那么事件向其他事件监听器的传递会被中止，**View**中默认的事件处理程序的回调也会被阻止。因此，当你却确信要中止一个事件的时候才返回**true**。

事件处理器

如果你要从**View**构建一个自定义的组件，你可以定义几个回调方法作为默认的事件处理器。在[自定义组件 - Custom Components](#)文档中，你可以学到一些用来处理事件的常用回调，包括：

- **onKeyDown(int, KeyEvent)** - 当新的按键事件产生的时候调用
- **onKeyUp(int, KeyEvent)** - 当一个按键松开事件产生的时候调用
- **onTrackballEvent(MotionEvent)** - 当轨迹球滚动的时候调用
- **onTouchEvent(MotionEvent)** - 当触屏动作产生时调用
- **onFocusChanged(boolean, int, Rect)** - 当**View**获取或者失去焦点的时候调用

另外还有一些需要注意的方法，它们不属于**View**类，但是能够直接影响到你处理事件的方式。因此，当你管理布局中更复杂的事件的时候，考虑下面这些方法：

- **Activity.dispatchTouchEvent(MotionEvent)** - 这个方法允许**Activity**在触屏事件被传递到**window**之前拦截它们。
- **ViewGroup.onInterceptTouchEvent(MotionEvent)** - 这个方法允许**ViewGroup**查看传递到它的子**view**的触屏事件。
- **ViewParent.requestDisallowInterceptTouchEvent(boolean)** - 在**View**

父 中调用这个方法来表明它不应该使用
用onInterceptTouchEvent(MotionEvent)来拦截触屏事件。

触摸模式

当用户使用方向键或者轨迹球浏览到一个用户界面的时候，就需要把焦点交给可交互的对象（比如按钮）以便用户能够看到这些对象能够接受输入。但是，如果一个设备支持触摸，并且用户通过触摸来实现交互，那么就不必高亮显示这些对象或者把焦点交给某个View了。这样，就有了一个名为触摸模式的交互模式了。

对一个支持触摸的设备，一旦用户触摸了屏幕，设备就进入了触摸模式。从此，只要一个View的isFocusableInTouchMode()方法返回值为true，那它就是可以获得焦点的，比如一个文本编辑框。其他可以触摸的View比如按钮，在被触摸的时候是不会获得焦点的；它们仅仅是在按下的时候触发一下它们on-click监听器。

每当用户点击了一个方向键或者滚动轨迹球的时候，设备就将退出触摸模式，并寻找一个View让其获得焦点。现在，用户就恢复到了不需要触摸屏的交互模式了。

触摸模式的状态是整个系统维护的（所有的窗口和活动视图）。要查看当前设备所处的触摸模式状态，可以调用isInTouchMode()方法得到当前设备是否处于触摸模式。

处理焦点

Android框架会处理焦点的移动来响应用户的输入，包括当View被删除或隐藏，或者新的View变得可见时的焦点变化。View通过isFocusable()方法来表明它们是否希望获得焦点。通过调用setFocusable()方法来设置一个View能否获得焦点。在触摸模式下，你通过isFocusableInTouchMode()方法查询一个View是否允许获得焦点。当然你可以通过调用setFocusableInTouchMode()方法来改变设置。

焦点的移动是基于一种在特定方向查找最近的可接受焦点的View的算法。在某些不常见的情况下，这种默认的算法找到的焦点可能与开发者的期望表现不一致。这时，你就要在下面的布局文件中的xml属性来精确指定焦点的移动：nextFocusDown, nextFocusLeft, nextFocusRight, 以及

nextFocusUp。给将要失去焦点的View添加一个上面的属性，在属性值中定义下一个将要获得焦点的View的id。比如：

```
<LinearLayout
    android:orientation="vertical"
    ...
    >
    <Button android:id="@+id/top"
        android:nextFocusUp="@+id/bottom"
        ...
        />
    <Button android:id="@+id/bottom"
        android:nextFocusDown="@+id/top"
        ...
        />
</LinearLayout>
```

通常，在这样一个垂直布局中，从第一个按钮向上浏览不会到任何地方，同样从第二个按钮向下浏览也不会到任何地方。现在，**top**按钮定义下一个向上导航的焦点获得者为**bottom**按钮（反之亦然），这样焦点就可以在这两个按钮之间上下循环了。

如果你习惯在UI中声明一个默认拥有焦点的View（通常不这样做），就需要在布局声明文件中给这个View添加一个 **android:focusable** 属性并把这个属性值设置为true。你也可以在触摸模式中使用 **android:focusableInTouchMode** 属性来定义默认拥有焦点的View。

要强制让某个View获得焦点，调用它的**requestFocus()**方法。

就像上面“事件监听器”章节讨论的一样，使用**onFocusChange()**方法来监听焦点变化事件（当一个View获取到或者失去焦点的时候会发出通知）。

来自“[index.php?title=Input_Events&oldid=8781](#)”



Menus

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文：<http://developer.android.com/guide/topics/ui/menus.html>

翻译： [Gavin Zhuang](#)

更新： 2012.06.08

目录

[[隐藏](#)]

[1 菜单-Menus](#)

- [1.1 在XML中定义菜单](#)
- [1.2 创建一个选项菜单](#)
 - [1.2.1 单击事件处理](#)
 - [1.2.2 运行时改变菜单项](#)
- [1.3 创建上下文菜单](#)
 - [1.3.1 创建浮动的上下文菜单](#)
 - [1.3.2 使用上下文动作模式](#)
- [1.4 创建弹出菜单](#)
 - [1.4.1 处理单击事件](#)
- [1.5 创建菜单组](#)
 - [1.5.1 使用可选的菜单项](#)
- [1.6 基于一个Intent添加菜单选项](#)
 - [1.6.1 允许你的活动添加到其他菜单](#)

菜单-Menus

在许多不同类型的应用中，菜单通常是一种用户界面组件。

为了提供给用户熟悉且一致的体验，你需要使用菜单API来展示用户动作和你应用中的其他选项。

从安卓3.0系统（API level 11）开始，安卓设备已经不再需要提供专用的菜单按键。基于这种变化，安卓应用需要远离原

来所依赖的传统**6**选项菜单盘，取而代之的是提供一个动作条来显示普通用户的动作。

虽然设计方案和用户使用菜单选项的方式已经改变，但是从语义上定义的一套动作和选项仍然是基于菜单**API**的。这份指导书将介绍在所有版本的安卓系统中如何去创建三个基本类型的菜单和动作：

选项菜单和动作条

选项菜单对于一个应用的菜单项来说是首要的。你放置其中的动作一般是可以影响整个应用的，例如“搜索”、“写邮件”和“设置”。

假如你为**2.3**或者更低版本的安卓系统开发应用，那么用户可以通过点击菜单按钮来显示选项菜单盘。

在安卓**3.0**或者更高的系统中，选项菜单中的选项作为屏幕上动作项和溢出的选项采用动作条显示。从安卓**3.0**开始，菜单按键是不被赞成的（一些设备一个也没有），所以你需要改为使用动作条来提供动作和其他选项的入口。

请查看关于[创建选项菜单](#)的章节。

上下文菜单和上下操作模式

上下文菜单是一种浮动的菜单，是在当用户在一个元件上执行长按动作时显示的。

当开发平台为安卓**3.0**或者更高的时候，你需要使用上下文操作模式来使所选的内容产生动作。这种模式显示的动作项会影响到在屏幕顶部条上选定的内容，并允许用户选择多项。

请查看关于[创建上下文菜单](#)的章节。

弹出窗口菜单

弹出窗口菜单显示一列被锚记为调用菜单列表的列表项。它很好的提供了一个涉及到具体内容或者提供一个命令的第二部分选项的溢出操作。在弹出菜单中的动作不会直接影响到相应的内容，这就是上下文操作所想要的。

[请查看创建弹出菜单的章节](#)

在XML中定义菜单

针对所有的菜单类别，安卓系统都提供了一个标准的XML格式来定义菜单项。你可以在一个XML菜单资源中定义一个菜单和它的所有选项，取代了在activity代码中建立菜单。你可以接着在你的活动中或者代码段中扩展菜单资源（载入它作为一个菜单对象）。

使用菜单资源是一个很好的惯例，主要有以下几个原因：

- 它更容易在XML中形象化菜单结构；
- 它把菜单的内容从你应用的行为代码中脱离出来；
- 它允许你创建交替的菜单结构以适应不同平台版本，屏幕大小，和其他利用应用资源框架的结构。

定义一个菜单，需要在你项目的res/menu/目录下创建一个XML文件以及使用下面这些元件创建菜单：

<menu>

定义一个菜单作为菜单项的容器。`<menu>`必须作为文件的根结点，这样才能容纳一个或多个`<item>`和`<group>`元件。

<item>

创建一个在菜单中表示一个单独的选项的菜单项。这个元件可能需要包含`<menu>`网来创建一个子菜单。

<group>

`<item>`元件中可选且不可见的容器。它允许你去把菜单项归类，所以它们可以分享特性例如激活状态和可视状态。更多的信息，请查看创建菜单组这一章节。

这里是一个命名为 game_menu.xml 的菜单实例：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/new_game"
      android:icon="@drawable/ic_new_game"
      android:title="@string/new_game"
      android:showAsAction="ifRoom" />
<item android:id="@+id/help"
      android:icon="@drawable/ic_help"
      android:title="@string/help" />
</menu>
```

<item>元件支持多种属性，你可以用来定义一个项的样式和行为。菜单上的选项包含了以下属性：

android:id

菜单项唯一的的ID资源，当用户选中这个选项时允许应用通过这个ID来识别这个菜单项。

android:icon

索引一个图片资源作为该项的图标。

android:title

索引一个字符串作为该项的标题

android:showAsAction

载明该项作为一个行为项什么时候和怎样显示在动作条中。

这些是你需要使用的最重要属性，但是还有更多可用的属性。关于所有支持的属性的信息请查阅菜单资源文档。

你可以通过增加一个**<menu>**元素作为**<item>**的子项，给任意菜单的项增加子菜单（除了子菜单本身以外）。当你的应用有大量的功能被组织成主题形式，例如电脑应用程序的菜单栏的选项（文件，编辑，查看等等）时子菜单是非常有用的。

例如：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

在你的应用中使用菜单，你可以使用[MenuInflater.inflate\(\)](#)寻找需要的菜单资源文件（将

文档资源转换成一个可编程的对象）。在接下来的章节，你将看到怎样为每个菜单类型定位菜单文件。

创建一个选项菜单

选项菜单包含了动作以及其他与当前活动上下文相关的选项,例如"搜索","撰写邮件"以及"设置"等.

在你选项菜单中的选项出现在屏幕中的位置时根据你开发应用的版本而定的:

- 如果你开发的应用是基于Android 2.3.x (API级别10) 或者更低的,那么当用户点击菜单按钮时你的选项菜单的内容出现在屏幕的底部。例如图像1.当菜单打开时,首先看到的是菜单图标的部分,且最多可容纳六个菜单项。如果你的菜单包含了多于六的菜单项,那么Android放置六个选项目且其余的放入到溢出的菜单中,用户可以通过选择“更多”选项来打开。
- 如果你开发的应用是基于Android 3.0 (API级别11) 或者更高的,那么菜单选项中的选项可以添加到动作条中。默认情况下,系统会放置所有选项在动作溢出栏中,用户可以打开在动作条右边的动作溢出图标(或者如果设备菜单按钮可用的话,用户可以点击它)。为了能够快速的访问重要的动作,你可以在相应的<item>元件中添加
`android:showAsAction="ifRoom"`使一些选项出现在动作条上(如图2)。

更多关于动作项和其他动作条行为的信息可以查看[Action Bar](#)说明。

注意: 即使你的开发没有基于Android 3.0或者更高版本,你也可以创建你自己的动作条布局取得相似的效果。例如你如何支持老版本的Android使用动作条,可以查看[ActionBar Compatibility](#) 的实例。



图像1 安卓2.3系统中浏览器的选项菜单

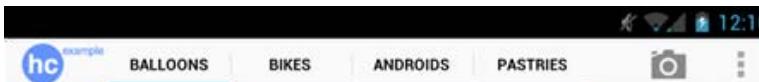


图2 Honeycomb Gallery应用上的动作条,显示了导航标签和

相机动作项 (加上了动作溢出按钮)

从你的活动 (Activity) 子类或者片段 (Fragment) 子类你可以声明选项菜单的选项。加入你的活动和片段都声明了选项菜单的选项，那么它们将被集合在UI界面中。活动的选项先显示，然后才是每个片段按顺序添加到活动中。如果需要，你还可以在你需要移动的每个<item>元素中添加 `android:orderInCategory` 的属性重新按次序添加菜单项。

去指定一个活动的选项菜单，需要覆写 `onCreateOptionsMenu()` 这个方法（片段提供它们自己的 `onCreateOptionsMenu()` 回调方法）。在这种方法下，你可以导入你的菜单资源（定义在XML文件中）到 `Menu` 提供的回调方法中。例如：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

你也可以使用 `add()` 方法添加菜单项以及使用 `findItem()` 方法取回选项，使用 `MenuItem APIs` 修改它们的属性。

如果你已经开发了基于Android 2.3.x或者更低版本的应用，那么当用户第一次打开菜单时系统可以调用 `onCreateOptionsMenu()` 去创建选项菜单。

如果你已经开发的应用是基于Android 3.0或者更高版本的，当活动 (Activity) 启动时系统可以调用 `onCreateOptionsMenu()` 在动作条上显示选项。

单击事件处理

当用户从选项菜单中选择一个选项时（包括动作条中的动作选项），系统将会调用你的活动中的 `onOptionsItemSelected()` 方法。这种方法是通过 `MenuItem` 选择的。你可以通过调用 `getItemId()` 来识别选项，那么它会返回一个菜单项中特定的 `ID`（定义在菜单资源中的 `android:id` 或者通过 `add()` 方法赋予其一个整型数）。你可以对已知的菜单项来匹配这个 `ID` 去执行适当的动作。例如：

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
// Handle item selection
switch (item.getItemId()) {
    case R.id.new_game:
        newGame();
        return true;
    case R.id.help:
        showHelp();
        return true;
    default:
        return super.onOptionsItemSelected(item);
}
```

当你成功的处理了一个菜单项，则会返回true值。如果你不能处理一个菜单项，那么你需要调用[onOptionsItemSelected\(\)](#)基类来实现（默认实现方法返回false值）

如果你的活动（Activity）包含片段（Fragment），那么系统首先会为活动调用[onOptionsItemSelected\(\)](#)，然后才是每个片段（每个片段已经按顺序添加）直到有一个返回true值，或者所有的片段都被调用了。

提示：Android 3.0在XML中的菜单项中为你增加了定义点击行为的能力，使用`android:onClick`属性即可。属性的值必须使用菜单的活动中定义的方法的名称。当系统调用这个方法的时候，方法必须是公共的且接受单独的`MenuItem`参数。且它是通过菜单项选择的。更多相关信息和例子，请查看[Menu Resource](#)文档。

提示：如果你的应用包含多个活动且其中有一部分提供相同的选项菜单，你可以创建一个活动只实现[onCreateOptionsMenu\(\)](#)和[onOptionsItemSelected\(\)](#)这两个方法。然后每个需要使用这个选项菜单的活动继承这个类。这样的话，你可以管理一部分代码来处理菜单动作且每个子类都集成菜单的动作。如果你想要给子活动增加菜单项，只要在这个活动中覆写[onCreateOptionsMenu\(\)](#)这个方法即可。调用`super.onCreateOptionsMenu(menu)`那么原菜单项就被创建了，然后使用`menu.add()`添加新的菜单项.当然你也可以为个别的菜单项复写基类的动作。

运行时改变菜单项

在系统调用[onCreateOptionsMenu\(\)](#)之后，它保留了你填充菜单中的一个实例且不会再调用[onCreateOptionsMenu\(\)](#)，除非菜单出于某种原因失效了。然而，你可以使[onCreateOptionsMenu\(\)](#)去创建一个有效的菜单状态且不能在活动的生命周期内改变。

如果你想要在活动的生命周期内基于事件修改选项菜单，你需要在[onPrepareOptionsMenu\(\)](#)方法中实现。只有目前存在菜单对象这个方法才能通过，那么你就可以修改它，例如增加，移除或者使选项失效。（片段也可以提供一个[onPrepareOptionsMenu\(\)](#)的回调）。

在Android 2.3.x或者更低的版本中，当用户每次打开选项菜单（通过点击菜单按钮）时系统都会调用`onPrepareOptionsMenu()`。

在Android 3.0或者更高版本中，当菜单项提交在动作条时选项菜单被认为是始终打开的。当一个事件发生且你想要执行一个菜单的更新，那么你必须调用`invalidateOptionsMenu()`这个方法去反馈系统调用的`onPrepareOptionsMenu()`方法。

注意：你永远不应该改变基于在选项菜单中目前处于焦点View的的选项。当其处于触点状态（用户没有使用循迹球或者d-pad），视图不能处于焦点状态，所以你永远不能在选项菜单中使用焦点来改变选项。假如你想要为视图（View）提供一个上下文敏感的菜单项，请使用上下文菜单。

创建上下文菜单

一个上下文菜单可以提供影响一个特殊选项或者UI中上下文的框架的动作。你可以为任何界面提供一个上下文菜单，但是它们通常在、或者用户可以在每个选项中直接执行的动作的其他界面分类中使用。

这里有两种方法提供一个上下文动作：

- 在浮动的上下文菜单中。当用户在一个声明支持上下文菜单的界面中执行一个长点击（按住并保持），那么这个菜单将作为一个菜单项浮动列表显示（类似对话框）。用户可以每次在一个选项上执行一个上下文动作。
- 在上下文动作模式下。这种模式是系统实现的动作模式，在屏幕顶部显示上下文动作条影响所选选项的动作选项的模式。当这种模式被激活，用户可以一次在一个动作中执行多个选项（如果你的应用允许这样）。

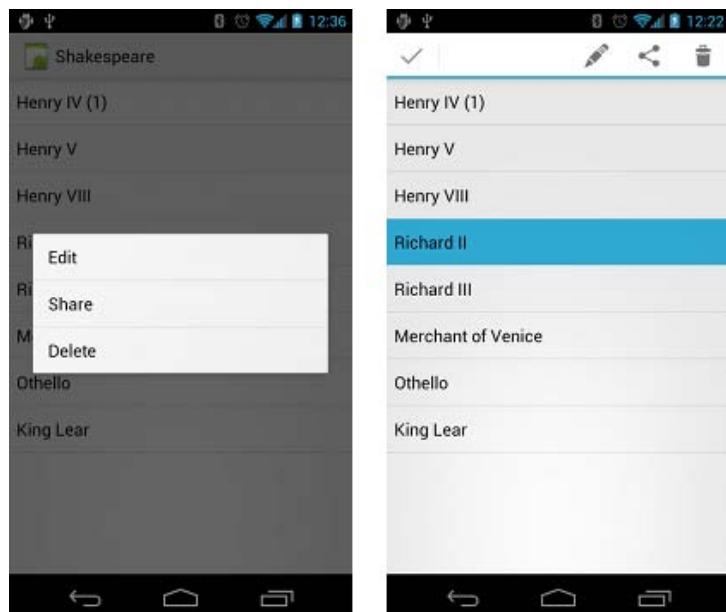


图 3. 浮动上下文菜单截图（左边）和上下文动作条（右边）。

注意：上下文动作模式只有在Android 3.0或者更高版本可用且当可用时是首选的显示上下文动作的技术。如果你的应用支持的版本低于3.0，那么你需要在这些设备中使用浮动的上下文菜单。

创建浮动的上下文菜单

为了提供一个浮动的上下文菜单：

1、通过调用[registerForContextMenu\(\)](#)来注册上下文菜单相关的视，并在视图中通过它。

如果你的活动使用了[ListView](#)或者[GridView](#)且你想要每个选项都提供一个相同的上下文菜单，那么需要通过调用[ListView](#)或者[GridView](#)中的[registerForContextMenu\(\)](#)为一个上下文菜单注册所有的选项。

2、在你的活动(Activity)或者片段(Fragment)实现[onCreateContextMenu\(\)](#)的方法。

当注册时接收到一个长点击事件，那么系统将会调用你的[onCreateContextMenu](#)方法。这是你定义菜单项的地方，通常通过导入一个菜单资源。例如：

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

[MenuInflater](#)允许你从一个菜单资源导入上下文菜单。这种回调方法的参数包含了用户选择的视图和提供关于被选项额外信息的[ContextMenu.ContextMenuInfo](#)对象。假如你的活动有若干个提供不同上下文菜单的视图，你需要使用这些参数来确定被导入的上下文菜单。

3、实现[onContextItemSelected\(\)](#)。

当用户选择一个菜单项时，系统调用这个方法则你可以执行相应的动作。例如：

```
@Override
public boolean onContextItemSelected(MenuItem item) {
```

```

AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
switch (item.getItemId()) {
    case R.id.edit:
        editNote(info.id);
        return true;
    case R.id.delete:
        deleteNote(info.id);
        return true;
    default:
        return super.onContextItemSelected(item);
}
}

```

`getItemId()`方法为被选的菜单项查询ID，这些ID是你在XML文件中使用`android:id`属性分配给每个菜单项的。如在XML中定义一个菜单的章节中所讲的。

当你成功的操作了一个菜单项，将返回`true`值，如果你不能操作菜单项，你需要通过基类来实现菜单项。如果你的活动包含片段（Fragment），那么活动将首先接收到这个回调。当不能操作时通过调用基类，系统将会在每个片段中使用各自的回调方法过滤掉这个事件，一次一个（每个片段都已经被按顺序添加了）直到返回`true`或者`false`已经返回了（默认活动和`android.app.Fragment`实现时返回`false`，当不能操作时你总是需要调用基类）。

使用上下文动作模式

上下文动作模式是系统实现的动作模式，重点是在执行上下文动作的用户交互。当一个用户通过选择一个选项使这个模式启用，那么上下文动作条将会出现在呈现用户可执行的当前被选选项的屏幕的顶部。当这种模式被启用，那么用户可以选择多个选项（如果你允许这样的话），取消多个选项以及继续在活动中导航（你允许的尽可能多的）。当用户通过点击返回按钮取消选择所有选项或者选择动作条坐标完成动作，则动作模式失效且上下文动作条消失。

注意：上下文动作条没有必要与动作条相关联。它们是独立运行的，即时上下文动作条显示超过动作条的位置。

如果你正在开发基于Android 3.0或者更高版本的应用，通常你需要使用上下文动作模式代替浮动上下文菜单来呈现上下文动作。

为提供上下文的视图，通常你需要调用上下文的动作模式如以下两个事件：

- 用户在界面中执行长点击。
- 用户选择一个复选框或者视图中类似的UI组件。

如何使你的应用调用上下文动作模式且根据你的设计为每个动作定义行为。基本上有两种设计：

- 对于个别的任意视图的上下文动作
- 对于在ListView或者GridView的选项组的一些上下

文动作（允许用户选择多个选项并执行它们所有的动作）。

接下来的部分是描述各个方案所需的设置。

为个别视图启用上下文动作模式

假如仅仅当用户选择特殊视图时你想要调用上下文动作模式，那么你应该：

- 1、实现**ActionMode.Callback**接口。在它的回调方法中，你可以为上下文动作条指定动作，相应在动作选项上点击事件，以及为动作模式操作其他生命周期事件。
- 2、当你想要显示一个动作条（例如当用户长点击视图）时可以调用**startActionMode()**方法。

例如：

- 1、实现**ActionMode.Callback**接口：

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {
    // Called when the action mode is created; startActionMode() was called
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate a menu resource providing context menu items
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    // Called each time the action mode is shown. Always called after
    // onActionMode, but
    // may be called multiple times if the mode is invalidated.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }

    // Called when the user selects a contextual menu item
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_share:
                shareCurrentItem();
                mode.finish(); // Action picked, so close the CAB
                return true;
            default:
                return false;
        }
    }

    // Called when the user exits the action mode
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        mActionMode = null;
    }
};
```

请注意，这些回调事件几乎和选项菜单回调一模一样的，除了这些还可以通过与事件相关的**ActionMode**对象。你可以使用**ActionMode APIs**来使**CAB**各种变化，例如使用**setTitle()**和**setSubtitle()**来修改标题和子标题（所有被选中的选项都可用）。

同时也请注意上述的例子在当动作模式被注销时设定了**mActionMode**变量为空。下一步，你将会看到它是如何初始化以及如何在你可用的活动或片段中保存成员变量。

2、当需要时调用startActionMode()**去启用上下文动作模式，例如相应一个视图中的长点击：**

```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Called when the user long-clicks on someView
    public boolean onLongClick(View view) {
        if (mActionMode != null) {
            return false;
        }

        // Start the CAB using the ActionMode.Callback defined above
        mActionMode = getActivity().startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

当你调用**startActionMode()**，系统返回**ActionMode**被创建。通过保存这个成员变量，你可以在相应其他事件时改变上下文动作条。在上述例子中，**ActionMode**是被用于确保**ActionMode**实例不会被重建，在它已经激活的情况下，且在动作模式开始的时候检查成员是否为空。

在**ListView**或者**GridView**中启用一批上下文动作

如果在**ListView**或者**GridView**（或者另一个**AbsListView**的扩展类）中你有一个选项分类，且想要允许用户去执行一批动作，你需要：

- 实现**AbsListView.MultiChoiceModeListener**接口且使用**setMultiChoiceModeListener()**把它设定给一个视图组。在监听回调的方法的时候，你可以为上下文动作条指定动作，在动作项中相应点击事件，或者操作来自**ActionMode.Callback**接口的其他回调。
- 使用**CHOICE_MODE_MULTIPLE_MODAL**调用**setChoiceMode()**。

例如：

```
ListView listView = getListView();
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position,
                                         long id, boolean checked) {
```

```

    // Here you can do something when items are selected/de-selected,
    // such as update the title in the CAB
}

@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    // Respond to clicks on the actions in the CAB
    switch (item.getItemId()) {
        case R.id.menu_delete:
            deleteSelectedItems();
            mode.finish(); // Action picked, so close the CAB
            return true;
        default:
            return false;
    }
}

@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    // Inflate the menu for the CAB
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.context, menu);
    return true;
}

@Override
public void onDestroyActionMode(ActionMode mode) {
    // Here you can make any necessary updates to the activity when
    // the CAB is removed. By default, selected items are
deselected/unchecked.
}

@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    // Here you can perform updates to the CAB due to
    // an invalidate() request
    return false;
}
);
}

```

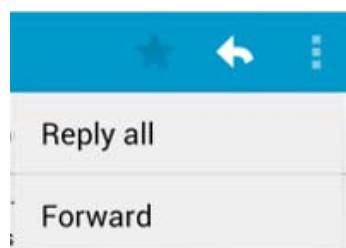
就是这样，现在当用户通过长点击选择一个选项时，系统将会调用`onCreateActionMode()`方法且为指定的动作显示一个上下文动作条。当上下文动作条可见时，用户可以选择附加的选项。

在上下文动作提供共同的动作选项的一些情况下，你可能想要添加一个复选框或者相似的允许用户选择的选项的UI组件，因为它们也许不能通过长点击行为发现。当一个用户选择复选框时，你可以通过使用`setItemChecked()`为各自的选项列表设定选择状态来调用上下文动作模式。

创建弹出菜单

弹出菜单是一个形式上的菜单标记在View上面。它出现在定位的窗口之下，如果那里有空间的话，或者在其上面。它对以下这些是有用的：

- 为关联到特殊内容的动作提供一个溢出模式的菜单（例如Gmail的邮件头部，如图4所示）。



注意：这是不同于会影响到所选内容的普通动作的上下文菜单。为了使动作能够影响所选内容，使用上下文动作模式或者浮动上下文菜单。

图像 4. 在Gmail应用里的弹出菜单，从右上角浮出。

- 提供一个命令句的第二部分（例如一个标记为"Add"的按钮，使用不同"Add"选项可产生一个弹出菜单）。
- 提供一个类似不保留持续选项的Spinner的下拉菜单。

注意：弹出菜单在API 11或者更高的版本可用。

如果你在XML中定义你的菜单，这里介绍了你怎样去显示弹出菜单：

- 1、使用其结构实例化一个弹出菜单，可以获取当前应用的Context和被标记的菜单的View。
- 2、使用MenuInflater导入你的菜单资源到菜单对象，通过PopupMenu.getMenu()来返回菜单对象。在API 14 及高于14的，你可以用PopupMenu.inflate()来导入。
- 3、调用PopupMenu.show().

例如，给一个按钮设定其android:onClick属性来显示弹出菜单：

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"

    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

在activity中显示弹出菜单如下：

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

在API 14 或者更高的，你可以组合两行
由PopupMenu.inflate() 导入的菜单。

当用户选择一个选项或者触摸菜单以外的区域菜单将消失。你
可以使用PopupMenu.OnDismissListener来监听菜单消失事件。

处理单击事件

当用户选择一个菜单项去执行一个动作时，你必须调用setOnMenuItemClickListener() 实现PopupMenu.OnMenuItemClickListener接口且用PopupMenu来注册它。当用户选择一个选项时，系统将在你的界面中调用onMenuItemClick() 函数来回调。

例如：

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

创建菜单组

菜单组是一个分享某些特征的菜单项集合，使用菜单组，你可以：

- 使用setGroupVisible()显示或隐藏所有选项；
- 使用setGroupEnabled()让所有选项有效或无效；
- 使用setGroupCheckable()指定是否所有选项可选。

在你的菜单资源或者使用add()方法指定的菜单组ID中，你可以在<group>元件里嵌套<item>元件来创建菜单组。

这里有一个包含菜单组的菜单资源例子：

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

file:///D:/guide/Menus[2015/9/23 19:15:18]

```

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/menu_save"
          android:title="@string/menu_save" />
    <group android:id="@+id/group_delete">
        <item android:id="@+id/menu_archive"
              android:title="@string/menu_archive" />
        <item android:id="@+id/menu_delete"
              android:title="@string/menu_delete" />
    </group>
</menu>

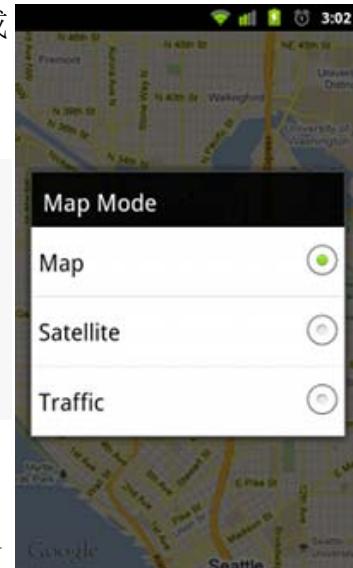
```

这些在组里的选项与第一个选项一样的级别显示——在菜单里的所有三个选项是同级别的。然而，你可以通过查询组的ID地址和使用上面列出的方法来修改组里面两个选项的特征。系统还从未分离过组选项。例如，假如你为每一个选项声明`android:showAsAction="ifRoom"`，那么他们将同时出现在动作条上或者动作溢出上。

使用可选的菜单项

一个菜单可以被当做选项开关的接口，独立选项使用复选框，或者互相排斥的选项。组使用单选按钮。图像5显示的是一个使用可选的单选按钮的子菜单。

注意：在图标菜单里的菜单项（来自选项菜单）不能显示复选框或者单选按钮。假如你选择让图标菜单里的选项可选，那么你需要在每次状态改变时手动的改变图标或者文字来表示选择状态。



你可以在`<item>`元素使用`android:checkable`属性来定义一个独立菜单项的可选行为，或者在`<group>`元素里使用`android:checkableBehavior`属性来为整个组定义。例如，这个菜单组里的所有选项使用单选按钮且可选：

```

<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item android:id="@+id/red"
              android:title="@string/red" />
        <item android:id="@+id/blue"
              android:title="@string/blue" />
    </group>
</menu>

```

图像5 子菜单选项可选的截图

`android:checkableBehavior`属性可设定为：

`single`

在菜单组中只有一个选项能被选中（单选按钮）

all

所有选项都可被选（复选框）

none

没有选项可选

你可以在`<item>`元素里使用`android:checked`属性来定义选项的默认选择状态或者在代码中使用`setChecked()`方法来改变状态。

当一个可选的项被选中，系统将调用选项各自的回调方法（例如`onOptionsItemSelected()`）。在这里你必须设定复选框的状态，因为复选框或者单选按钮无法自动改变它们的状态。你可以使用`isChecked()`查询选项的当前状态，以及使用`setChecked()`来设定它选中状态。例如：

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.vibrate:
        case R.id.dont_vibrate:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

假如你不通过这种方式来设定选中状态，那么选项的可视状态（复选框或者单选按钮）在用户选中它时将不会改变。当你设定了状态，`activity`将维持选项的选中状态以至于当用户较迟打开菜单，选中状态你设定为可见。

注意：可选菜单项是被拟用于每个会话基础上的，且不能在应用销毁后保留。假如你想为用户保存应用设定信息，你需要使用**Shared Preferences**保存数据。

基于一个**Intent**添加菜单选项

一些时候你可能想要使用`intent`来载入一个活动（无论这个活动是在你的应用中或者其他应用中）。当你知道你想要使用的`intent`且有一个可以开启这个`Intent`的具体菜单项时，你可以在合适的选项被选中回调方法时使用`startActivity()`来开启这个`Intent`（例如

onOptionsItemSelected() 回调)。

然而，假如你不确定用户的设备是否包含有处理这个Intent的应用，却还添加了一个菜单项来调用它，这将导致生成一个无功能的菜单项，因为这个Intent可能无法在这个活动中解决。为了解决这个问题，安卓系统可以允许你在设备发现发现了可以处理你的Intent的活动时动态的添加菜单选项。

基于可接受一个Intent的活动的添加菜单方式：

- 1、使用CATEGORY_ALTERNATIVE 和/或者 CATEGORY_SELECTED_ALTERNATIVE 定义一个intent，可以添加其他任意需求。
- 2、调用Menu.addIntentOptions()方法。安卓系统可以搜寻到任何能够执行这个Intent的应用，且将它们添加到你的菜单中。

假如系统中没有安装可以满足Intent的应用，那么将没有菜单选项可以被添加。

注意：CATEGORY_SELECTED_ALTERNATIVE是被用来处理当前在屏幕上被选中的元素的。所以，它只能在采用onCreateContextMenu()创建菜单时使用。

例如：

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    // 创建一个描述所有需要被执行的需求的Intent，包含在我们的菜单中。
    // 提供的应用必须包含Intent.CATEGORY_ALTERNATIVE类的值。
    Intent intent = new Intent(null, dataUri);
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
    // 利用可接受的提供应用来搜寻和填充菜单
    menu.addIntentOptions(
        R.id.intent_group, // 添加新的新的菜单项的菜单组
        0, // 唯一的选项ID (无)
        0, // 目的选项 (无)
        this.getComponentName(), // 当前活动的名字
        null, // 位于第一位的具体选项 (无)
        intent, // 一句前面描述的需求定义的Intent
        0, // 控制选项的其他标志 (无)
        null); // 具体项目相关的MenuItem阵列 (无)
    return true;
}
```

为了让每个活动都能被找到，那么需要提供一个匹配已定义的Intent的容器、一个被添加进来的菜单项、使用在intent容器中类似菜单项名字的 android:label 的值，以及类似菜单项图标的应用图标。addIntentOptions()方法将返回被添加进来的选项的数量。

注意：当你调用addIntentOptions()，它将覆盖菜单组指定的所有菜单项。

允许你的活动添加到其他菜单

你可也可以为其他应用提供你的Activity(活动)中的服务,所以你的应用也可以被包含在其他应用的菜单中 (反之亦可) 。

为了能够被添加到其他应用得菜单中, 通常你需要定义一个intent (意图) 过滤器, 但是必须要包含 [CATEGORY_ALTERNATIVE](#) 和/或者 [CATEGORY_SELECTED_ALTERNATIVE](#) 的intent (意图) 过滤器类别的值。例如:

```
<intent-filter label="@string/resize_image">
    ...
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    ...
</intent-filter>
```

在[Intents and Intents Filters](#)文档中, 你可以阅读到更多关于如何书写intent(意图)过滤器的内容。

对于使用这种技术的示例应用, 可以查看 [Note Pad](#) 示例代码。

来自“[index.php?title=Menus&oldid=13598](#)”



Dialogs

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

- [1 对话框](#)
 - [1.1 对话框](#)
 - [1.2 创建对话片段](#)
 - [1.3 构建警告对话框](#)
 - [1.3.1 添加按钮](#)
 - [1.3.2 添加列表](#)
 - [1.3.3 添加持久多选或单选列表](#)
 - [1.3.4 创建自定义布局](#)
 - [1.4 Passing Events Back to the Dialog's Host](#)
 - [1.5 显示对话框](#)
 - [1.6 显示对话框全屏或作为嵌入式片段](#)
 - [1.6.1 用对话框在大屏幕上显示活动](#)
 - [1.7 关闭对话框](#)

对话框

对话框是一个提示用户做出决定或输入其它信息的小窗口。同时对话框不会填满屏幕。

对话框

如果想要设计包括语言建议在内的对话，请阅读对话框设计指南信息。



Dialog类是创建对话框的基类。但是，您通常不应该直接实例化一个对话框。相反，你应该使用下面的子类中的一个：

AlertDialog（弹出对话框）

对话框可以显示一个标题和最多三个按钮，还包括可选的项目列表以及自定义布局。

DatePickerDialog（日期选择对话框）及TimePickerDialog（时间选择对话框）

允许用户选择一个日期或者时间的对话框。

这些类定义对话框的风格和结构，但你要使用**DialogFragment**作为对话框的容器。**DialogFragment**类提供了创建对话框并管理其外观需要的所有控件，而不用再调用**Dialog**的方法。

使用**DialogFragment**管理对话框可以确保它正确处理生命周期事件，比如用户按下返回按钮或旋转屏幕等。**DialogFragment**类还允许对话框的用户界面就像传统片段那样，作为一个更大的用户界面内的可嵌入组件重新使用。

本指南中的以下各节介绍如何将**DialogFragment**与**AlertDialog**对象进行组合。如果你想创建日期或时间选择器，应该改为阅读选择器指南。

注意：由于**DialogFragment**类最初添加了Android3.0（API级别11），该文件描述了如何使用该公司所提供支持库的**DialogFragment**类。把该库添加到应用，你可以使用**DialogFragment**并运行Android1.6或更高版本设备的各种API。如果应用程序支持的最低版本是API级别11或更高，那么你可以使用**DialogFragment**的框架版本，但要注意本文件中的链接都是支持API库的。当使用支持库时，请确保您导入的是`android.support.v4.app.DialogFragment`类，而不是`android.app.DialogFragment`。

创建对话片段

你可以进行包括自定义布局和对话框设计指南在内的各种对话框设计，其中对话框设计指南可以通过扩展**DialogFragment**并创建在**onCreateDialog ()**回调方法中的**AlertDialog**获得。

例如，下方提供了一个在**DialogFragment**中管理的基本**AlertDialog**：

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new
        AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
            .setPositiveButton(R.string.fire, new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int
        id) {
                // FIRE ZE MISSILES!
            }
        })
        .setNegativeButton(R.string.cancel, new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int
        id) {
                // User cancelled the dialog
            }
        });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

现在，当你创建这个类以及**show ()**该对象的实例时，出现的对话框如图1。

下一节将介绍更多使用**AlertDialog.Builder API**来创建对话框的信息。

基于对话复杂程度，可以实现**DialogFragment**中包括所有基本片段生命周期方法在内的其他回调方法。

构建警告对话框

该**AlertDialog**类允许你建立各种对话框的设计和往往是你唯一需要的对话框类。如图2中，警告对话框有三个区域：

1.

标题。此为可选项。应该仅当内容区域被详细信息、列表或者自定义布局占据时使用。如果需要陈述简单的消息或问题（例如图1对话框），则不需要标题。

2. 内容区域。可以显示消息，列表或者其他自定义布局。
3. 动作按钮。一个对话框的操作按钮不能超过3个。



`AlertDialog.Builder`类提供了API，可以使用包括自定义布局在内的内容来创建`AlertDialog`。要建立`AlertDialog`：

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
// 2. Chain together various setter methods to set the dialog
// characteristics
builder.setMessage(R.string.dialog_message)
    .setTitle(R.string.dialog_title);
// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

下列主题显示如何使用`AlertDialog.Builder`定义各种对话框属性。

添加按钮

为了添加图2所示按钮，调用`setPositiveButton()` and `setNegativeButton()`方法：

```
AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
// Add the buttons
builder.setPositiveButton(R.string.ok, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User clicked OK button
    }
});
builder.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User cancelled the dialog
    }
});
// Set other dialog properties
...
```

```
// Create the AlertDialog
AlertDialog dialog = builder.create();
```

set...Button()方法需要按钮标题（由字符串资源提供）
和DialogInterface.OnClickListener（当按下按钮时进行相应操作）。

可以添加下列三种不同的操作按钮：

- 肯定

使用此按钮来接受并继续进行操作（"OK"操作）。

- 否定

使用此按钮来取消操作。

- 中立

当用户不希望继续操作时使用此按钮，但并不一定取消操作。例如操作可能显示“稍后提醒”。

AlertDialog只能添加各类型按钮各一个。也就是说不能添加两个肯定按钮。

添加列表

AlertDialog API有三种类型可用列表：

- 传统单选项列表
- 永久单选项列表（单选按钮）
- 永久多选项列表（复选框）



要创建图3所示单选项列表，请使用setItems() 方法：

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new
    AlertDialog.Builder(getActivity());
    builder.setTitle(R.string.pick_color)
```

```
Dialogs - eoeAndroid wiki
        .setItems(R.array.colors_array, new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int
which) {
                // The 'which' argument contains the index position
                // of the selected item
            }
        });
    return builder.create();
}
```

因为列表在对话框内容区域显示，对话框不能同时显示消息和列表，你应该为setTitle()对话框设置一个标题。要指定列表项目，可以调用setItems()来传递数组。或者可以使用setAdapter()指定列表。这样可以使用ListAdapter返回动态数据列表。

如果用ListAdapter返回列表，则要使用Loader以便内容加载异步。

添加持久多选或单选列表

要添加多项选择（复选框）或者单项选择（单选按钮）列表，分别使用setMultiChoiceItems()或者setSingleChoiceItems()方式。



例如这里显示了如何创建图4所示保存 ArrayList 所选项目的多选列表：

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mSelectedItems = new ArrayList(); // Where we track the
selected items
    AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
    // Set the dialog title
    builder.setTitle(R.string.pick_toppings)
    // Specify the list array, the items to be selected by default
(null for none),
    // and the listener through which to receive callbacks when
items are selected
    .setMultiChoiceItems(R.array.toppings, null,
        new
DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int
which,
                boolean isChecked) {
                if (isChecked) {
                    // If the user checked the item, add it to
the selected items
            }
        }
    });
    return builder.create();
}
```

```

        mSelectedItems.add(which);
    } else if (mSelectedItems.contains(which)) {
        // Else, if the item is already in the array,
remove it
    }
}
// Set the action buttons
.setPositiveButton(R.string.ok, new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int id) {
results somewhere
        // User clicked OK, so save the mSelectedItems
the dialog
        ...
    }
})
.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int id) {
        ...
    }
});
return builder.create();
}

```

尽管传统列表和单选列表都提供了“单选”操作，如果要坚持用户的选择您可以使用`setSingleChoiceItems()`。如果再次打开对话框，就应该指示用户当前选择，然后创建单选按钮列表。

创建自定义布局

如果想要对话框自定义布局，可以创建一个布局并调用`AlertDialog.Builder`对象的`setView()`添加到`AlertDialog`。

默认情况下自定义布局填充对话框窗口，但是仍可以使`AlertDialog.Builder`方法来添加按钮和标题。



例如下面是图5对话框的布局文件：`res/layout/dialog_signin.xml`

```
<LinearLayout
```

```

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >
<ImageView
        android:src="@drawable/header_logo"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:scaleType="center"
        android:background="#FFFFBB33"
        android:contentDescription="@string/app_name" />
<EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/username" />
<EditText
        android:id="@+id/password"
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:fontFamily="sans-serif"
        android:hint="@string/password" />
</LinearLayout>

```

为了扩展DialogFragment布局，使用getLayoutInflator()来获得LayoutInflater同时调用inflate()（第一个参数是布局资源ID，第二个参数是布局父类视图）。然后调用setView()来进行对话框布局。

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
    // Get the layout inflator
    LayoutInflator inflater = getActivity().getLayoutInflator();
    // Inflate and set the layout for the dialog
    // Pass null as the parent view because its going in the dialog
layout
    builder.setView(inflater.inflate(R.layout.dialog_signin, null))
    // Add action buttons
        .setPositiveButton(R.string.signin, new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                // sign in the user ...
            }
}

```

```
Dialogs - eoeAndroid wiki
    })
        .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        LoginDialogFragment.this.getDialog().cancel();
    }
});
return builder.create();
}
```

提示：如果想要自定义对话框，就用活动来作为对话框，而不是使用对话框API。简单创建一个活动并在<activity>清单元素设置主题为Theme.Holo.Dialog。

```
<activity android:theme="@android:style/Theme.Holo.Dialog" >
```

Passing Events Back to the Dialog's Host

当用户触摸对话框操作按钮之一或者选择列表中的项目时，DialogFragment可能会执行必要操作，但是往往回向活动或片段传递事件。要做到这一点，就需要定义每种类型的单击事件的方法接口。然后在要接收操作事件的主机组件实现该接口。

例如DialogFragment定义了提供返回主机活动的事件的界面：

```
public class NoticeDialogFragment extends DialogFragment {
    /* The activity that creates an instance of this dialog fragment must
     * implement this interface in order to receive event callbacks.
     * Each method passes the DialogFragment in case the host needs
     * to query it. */
    public interface NoticeDialogListener {
        public void onDialogPositiveClick(DialogFragment dialog);
        public void onDialogNegativeClick(DialogFragment dialog);
    }

    // Use this instance of the interface to deliver action events
    NoticeDialogListener mListener;

    // Override the Fragment.onAttach() method to instantiate the
    NoticeDialogListener
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        // Verify that the host activity implements the callback
        interface
```

```

try {
    // Instantiate the NoticeDialogListener so we can send
events to the host
    mListener = (NoticeDialogListener) activity;
} catch (ClassCastException e) {
    // The activity doesn't implement the interface, throw
exception
    throw new ClassCastException(activity.toString()
        + " must implement NoticeDialogListener");
}
}
...
}

```

The activity hosting the dialog creates an instance of the dialog with the dialog fragment's constructor and receives the dialog's events through an implementation of the `NoticeDialogListener` interface:

```

public class MainActivity extends FragmentActivity
    implements
NoticeDialogFragment.NoticeDialogListener{
    ...

    public void showNoticeDialog() {
        // Create an instance of the dialog fragment and show it
        DialogFragment dialog = new NoticeDialogFragment();
        dialog.show(getSupportFragmentManager(),
"NoticeDialogFragment");
    }

    // The dialog fragment receives a reference to this Activity
    // through the
    // Fragment.onAttach() callback, which it uses to call the
    // following methods
    // defined by the NoticeDialogFragment.NoticeDialogListener
    // interface
    @Override
    public void onDialogPositiveClick(DialogFragment dialog) {
        // User touched the dialog's positive button
        ...
    }

    @Override
    public void onDialogNegativeClick(DialogFragment dialog) {
        // User touched the dialog's negative button
        ...
    }
}

```

因为主机活动实现了由 `onAttach()` 执行的回调方式 `NoticeDialogListener`，对话框片段可以使用接口回调方法来传递事件。

```

public class NoticeDialogFragment extends DialogFragment {
    ...
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Build the dialog and set up the button click handlers
        AlertDialog.Builder builder = new
        AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
            .setPositiveButton(R.string.fire, new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int
id) {
                // Send the positive button event back to the
host activity
                mListener.onDialogPositiveClick(NoticeDialogFragment.this);
            }
        })
            .setNegativeButton(R.string.cancel, new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int
id) {
                // Send the negative button event back to the
host activity
                mListener.onDialogNegativeClick(NoticeDialogFragment.this);
            }
        });
        return builder.create();
    }
}

```

显示对话框

当想要显示对话框时，创建DialogFragment实例并调用show()，传递FragmentManager和对话框片段的标签名称。

可以通过调用 FragmentActivity的getSupportFragmentManager()或者 Fragment的getFragmentManager()。例如：

```

public void confirmFireMissiles() {
    DialogFragment newFragment = new FireMissilesDialogFragment();
    newFragment.show(getSupportFragmentManager(), "missiles");
}

```

第二个参数"missiles"是系统用来必要时保存和恢复片段状态的标记名称。标签还可以调用findFragmentByTag()来获得控制代码。

显示对话框全屏或作为嵌入式片段

这种情况下不能使用`AlertDialog.Builder`或其他`Dialog`对话来建立对话框。如果想要 嵌入`DialogFragment`，就必须定义对话框UI，然后加载`onCreateView()`回调的布局。下面是作为对话或者嵌入片段的`DialogFragment`例子（使用名为`purchase_items.xml`的布局）：

```
public class CustomDialogFragment extends DialogFragment {
    /** The system calls this to get the DialogFragment's layout,
     * regardless
     * of whether it's being displayed as a dialog or an embedded
     * fragment. */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout to use as dialog or embedded fragment
        return inflater.inflate(R.layout.purchase_items, container,
false);
    }

    /** The system calls this only when creating the layout in a
     * dialog. */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // The only reason you might override this method when using
        // onCreateView() is
        // to modify any dialog characteristics. For example, the
        // dialog includes a
        // title by default, but your custom layout might not need
        // it. So here you can
        // remove the dialog title, but you must call the superclass
        // to get the Dialog.
        Dialog dialog = super.onCreateDialog(savedInstanceState);
        dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
        return dialog;
    }
}
```

这里显示了一些代码，可以决定是否根据屏幕尺寸用对话或全屏UI来显示片段：

```
public void showDialog() {
    FragmentManager fragmentManager = getSupportFragmentManager();
    CustomDialogFragment newFragment = new CustomDialogFragment();

    if (mIsLargeLayout) {
        // The device is using a large layout, so show the fragment
        // as a dialog
        newFragment.show(fragmentManager, "dialog");
    } else {
```

```
// The device is smaller, so show the fragment fullscreen
FragmentManager transaction =
fragmentManager.beginTransaction();
    // For a little polish, specify a transition animation
transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
    // To make it fullscreen, use the 'content' root view as the
// container
    // for the fragment, which is always the root view for the
activity
        transaction.add(android.R.id.content, newFragment)
            .addToBackStack(null).commit();
    }
}
```

欲查询执行片段事务的更多信息，请参见片段指南。

这一事例中`mIsLargeLayout boolean`指定当前设备是否要使用应用程序大布局设计。设置这种`boolean`的最好方法就是用不同屏幕大小的替代资源值来声明`bool`资源值。例如，有适用于不同屏幕尺寸的`bool`资源的两个版本：

`res/values/bools.xml`

```
<!-- Default boolean values -->
<resources>
    <bool name="large_layout">false</bool>
</resources>
```

`res/values-large/bools.xml`

```
<!-- Large screen boolean values -->
<resources>
    <bool name="large_layout">true</bool>
</resources>
```

接着使用活动`onCreate()`方法可以初始化`mIsLargeLayout`值。

```
boolean mIsLargeLayout;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mIsLargeLayout = getResources().getBoolean(R.bool.large_layout);
}
```

用对话框在大屏幕上显示活动

想要仅当在大屏幕时显示对话框活动，可以在`<activity>`清单元素中应用`Theme.Holo.DialogWhenLarge`主题：

```
<activity android:theme="@android:style/Theme.Holo.DialogWhenLarge" >
```

请参阅样式和主题指南，以便查找用主题定义活动样式的更多信息。

关闭对话框

当用户触摸`AlertDialog.Builder`创建的任何操作按钮时，系统会关闭对话框。

当用户触摸对话框（除单选按钮和复选框之外）列表中的项目时，系统也会关闭对话框。否则可以在`DialogFragment`调用`dismiss()`来关闭对话框。

如果对话框消失时需要执行某些操作，就可以在`DialogFragment`实现`onDismiss()`方法。

可以取消对话框。如果用户按下返回按钮，触摸对话框区域外的屏幕就会进行这一操作。如果显性调用对话框的`cancel()`（比如相对应对话框的“取消”按钮）。

来自“[index.php?title=Dialogs&oldid=13856](#)”

Action Bar

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/topics/ui/actionbar.html#Adding>

作者： tmacbo

更新时间：

目录

- [1 Action Bar](#)
 - [1.1 添加操作栏](#)
 - [1.1.1 移除操作栏](#)
 - [1.2 添加操作项](#)
 - [1.2.1 选择你的操作项](#)
 - [1.2.2 使用分裂的操作栏](#)
 - [1.3 使用应用程序图标来导航](#)
 - [1.3.1 向应用程序上级页面导航](#)
 - [1.4 添加操作视窗](#)
 - [1.4.1 处理可折叠的操作视窗](#)
 - [1.5 添加一个操作提供器](#)
 - [1.5.1 使用ShareActionProvider类](#)
 - [1.5.2 创建一个定制的操作提供器](#)
 - [1.6 添加导航选项标签](#)
 - [1.7 添加下拉式导航](#)
 - [1.8 设置操作栏的样式](#)
 - [1.8.1 普通的外观](#)
 - [1.8.2 操作项样式](#)
 - [1.8.3 导航选项标签样式](#)
 - [1.8.4 下拉列表样式](#)

- [1.8.5 高级样式](#)

Action Bar

操作栏是一个窗口功能用于确定应用程序和用户的位置，并提供给用户操作和导航模式。如果需要突出当前用户的操作或导航，应该使用操作栏，因为操作栏为用户提供了一个一致的接口，这个接口跨应用程序和系统，并且不同尺寸的屏幕适配操作栏的外观。你可以通过[ActionBar API](#)来控制动作栏的行为和可视性，这个API被添加在Android 3.0(API级别为11)。操作栏设计的初衷是：

- 提供一个专门的空间来确定应用程序的标识和用户的位置。
- 这是在应用程序图标或者是左侧的标志以及Activity的标题帮助下完成的。如果当前视图的导航标签被标识，例如当前选项卡选中，你可能会选择删除该活动名称。
- 提供一致的导航和视图细化到不同的应用程序中。
- 操作栏提供了内置选项卡导航来进行切换。它还提供了一个下拉列表中，可以用来替代导航模式或用来完善当前视图(比如按照不同的标准来排序列表)。
- 突出活动的关键动作(如“搜索”、“创建”、“共享”，等等。)，便于用户在一个可预测的方法。
- 对于关键用户操作，你可以通过将项目从选项菜单直接在操作栏定义为[操作项](#)来提高访问速度。操作项也可以提供一个“操作窗口”，它用一个嵌入式部件来提供更多的及时活动的操作。它提供了一个更直接的行动行为的嵌入式部件。没有改进成操作项的菜单项在溢出菜单中还是有效的，用户既可以使用设备上的菜单按钮(设备上有按钮的时候)，也可以使用操作栏中的溢出菜单按钮(当设备上不包含菜单按钮时)来显示这些操作项目。



图1、Action Bar来源于[Honeycomb](#)的app库，从左边开始，依次为logo，导航标签与操作项(在右边插入溢出菜单按钮)。

注意有关Action Bar设计准则，可以阅读Android文档[Action Bar](#)的设计指南。

添加操作栏

从Android3.0 (API级别11) 开始，ActionBar包括在所有Activity中使用的[Theme.Holo](#)主题（或是继承Activity的一个子类），这是当[targetSdkVersion](#)或[\[1\]](#)属性设置为“11”或更高时程序默认的主题。例如：

```
<manifest ... >
    <uses-sdk android:minSdkVersion="4"
              android:targetSdkVersion="11" />
    ...
</manifest>
```

在这个例子中，应用程序设置的最低版本的API等级为4 (Android 1.6)，但它目标API级别为11 (Android 3.0)。通过这样设置，当应用程序运行在Android 3.0或更高版本上时，该系统适用于全息每个Activity的主题，因此，每一个Activity包括ActionBar。

如果你想使用[ActionBar](#)的API，比如添加导航模式和修改操作栏样式，你应该设置[minSdkVersion](#)为“11”或是更高的版本。如果你想你的应用程序支持旧版本的Android，有很多办法可以让低版本的[ActionBar](#)的API在支持API级别为11或更高的设备上，同时仍运行旧版本。参看[sidebox](#)保持向后兼容的信息。

移除操作栏

如果你不想为一个特定的Activity设置ActionBar，设置Activity主题为[Theme.Holo.NoActionBar](#)。例如：

```
<activity android:theme="@android:style/Theme.Holo.NoActionBar" >
```

您还在运行时通过调用[hide\(\)](#)隐藏ActionBar。例如：

```
ActionBar actionBar = getSupportActionBar();
actionBar.hide();
```

当ActionBar隐藏，系统的Activity调整布局来填补所有可用屏幕空间。你可以通过调用[show\(\)](#)显示ActionBar。隐藏和删除操作栏可能会使Activity重新调整布局，重新使用ActionBar所占用的空间。如果你的活动经常隐藏和显示操作栏

(如在Android应用程序库)，你可能想用叠加模式。叠加模式布局在Activity的顶部，而不是在屏幕空间上的Action Bar。这样，你的布局可以在Action Bar隐藏和重新出现时保持不变。要启用覆盖模式，创建Activity主题并且将[android:windowActionBarOverlay](#)属性值设置为true。欲了解更多信息，请参阅样式的Action Bar章节。

提示：如果你有一个删除了操作栏的定制化的Activity主题，它把[android:windowActionBar](#)样式属性设置为false。但是，如果你使用了删除操作栏的一个主题，那么，创建窗口将不允许操作栏再显示，因此，你不能在以后给这个Activity添加操作栏---因为getActionBar()方法将返回null。

添加操作项

有时你可能想让用户从选项菜单中直接访问项目。要做到这一点，你可以声明该菜单项为Action Bar中的一个“action item”。一个“action item”包括一个图标和/或文字标题。如果一个菜单项不作为一个“action item”，系统会将菜单项放置在溢出菜单。溢出菜单显示设备菜单“按钮（如果设备提供）或在操作栏中的按钮（如果设备不提供“菜单”按钮）。

首次启动Activity时，系统通过在activity调用onCreateOptionsMenu（）方法来填充action bar和溢出菜单（overflow menu）。在菜单开发指南中讨论的，它是在这个回调方法，你应该夸大一个XML定义菜单项的菜单资源。例如：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_activity, menu);
    return true;
}
```



图2两个action item的图标和文字标题，以及溢出菜单按钮。

在XML文件中，你可以通过声明[android:showAsAction="ifRoom"](#)成为这个条目

的元素，从而使一个菜单项变为action item。通过这种方式，当有可用空间时，菜单项才会出现在action item的快速访问栏。如果没有足够的空间，该item将出现在溢出菜单。

如果你的菜单项同时提供标题和图标--同时具有Android:title和android:icon属性，action item默认只会显示图标。但如果你想显示文字标题，必须添加“withText”到Android:showAsAction属性中。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/ic_menu_save"
          android:title="@string/menu_save"
          android:showAsAction="ifRoom|withText" />
</menu>
```

注：“withText”值应是以一个操作栏的提示文本的标题出现。但如果一个图标无法使用或者空间受限，action bar的标题可能无法显示。

当用户选择一个action item，activity调用一个OptionsItemSelected () 方法，通过Android:id属性获取的ID来接收在选项菜单中的所有item中相同的回调。这一点很重要，你总是为每个菜单项定义android: title，即使你不在显示的action item声明标题，原因有三：

- 如果在action bar中没有足够的空间提供给action item，该菜单项出现溢出“菜单中”，而且只有标题显示。
- 屏幕阅读器为视障用户读出菜单项的标题。
- 如果action item只有图标，用户可以长按item显示的工具提示，显示action item的标题。

Android的图标始终是可选的。但对于图标设计建议，详情参考Action Bar图标设计指引。

注：如果您添加了菜单项，从一个片段，通过片段类的onCreateOptionsMenu，回调然后系统调用各自onOptionsItemSelected () 的方法，当用户选择该片段片段的项目之一。

然而，该活动得到一个机会来处理事件，所以系统调用活动`onOptionsItemSelected()`的片段，然后再调用相同的回调函数。

你也可以定义一个item“总是”为action item，以避免当空间有限时被放到溢出菜单中去。但在大多数情况下，不应该设置“always”这个值来强制使一个item出现在action bar中。然而，然而，当提供的是一个“活动视图（action view）”而不是提供“溢出菜单”中的默认动作时，你可能需要这个item总是出现。要注意的是过多的action item，会导致创建出来的UI杂乱不堪，并且在窄屏幕的设备上会出现布局问题。最好使用“ifRoom”，而不是要求一个item出现在action bar中，但在没有足够的空间时，应当允许系统将它移动到溢出菜单。对于有关创建选项菜单定义action item的详细信息，请参阅“菜单”开发人员指南。

选择你的操作项

你应该应通过评估的几个关键特性，仔细从选项菜单中选出action item。在一般情况下，每个action item，至少是下列操作之一：

- 1、常用性：用户70%的时间需要访问或需要连续多次使用。常用性例子：在消息应用程序和“搜索”谷歌播放的“新信息”。
- 2、重要性：用户能够很容易地发现，或者如果不经常使用，在少数情况用户确实需要它的时候，可以毫不费力地执行，这一点是很重要的。重要性例子：“加入网络”Wi-Fi设置“切换到相机”在库应用程序。
- 3、典型性：这是一个通常在类似的应用程序的操作栏中提供的行动，因此，用户希望自己找到它。典型性例子：电子邮件或社会的应用，“刷新”和“新接触”在应用程序。

如果你认为四个以上菜单项可以合理的作为action item，那么你应该仔细考虑其相对水平的重要性，并尽量设置不超过四个的菜单项的action item（这样设置“ifRoom”这个值，在一些空间有限的小屏幕上，系统把一些菜单项放到溢出菜单背面）。即使是宽屏幕上，你也不要创建一个杂乱的UI的action item，冗长得看起来像一个桌面工具栏，应该要使action item的数量保持到最低限度。此外，下列行为不应该出现行动项目中：设置，帮助，反馈，或查找相似，始终将这些放在溢出菜单中。

注意：请记住，并非所有的设备都提供了一个搜索专用硬件按钮，因此，如果是应用程序的重要功能，它应该始终作为一个action item（通常作为第一

个item，特别是如果你提供的是一个action view）。

菜单项对比其他应用程序的控制 作为一般规则，在选项菜单中（更不用说action item）的所有项目应该有一个应用程序的全局影响力，而不是只影响一小部分的接口。例如，如果你有一个多窗格的布局和一个窗格显示一个视频，而另一个列出的所有视频，视频播放器的控件应出现在包含视频窗格（而不是在操作栏），而action bar可能会提供action item来共享视频或保存视频到收藏夹列表中。所以，在决定是否应该设置一个菜单项的action item前，先确保该项目当前activity是否有一个全局范围。如果没有，那么你应该设置一个按钮放置在适当的范围内的布局中。

使用分裂的操作栏

当您的应用程序上运行Android 4.0系统（API 14级）或更高级别时，还有一个额外的模式可称action bar为“split action bar”。当在一个狭窄的屏幕运行启用split action bar时，会在屏幕的底部出现一个action bar显示所有action item。分裂action bar用来分开action item，确保分配合理数量的空间来在一个狭窄的屏幕上显示所有的action item，而空间留给顶端的导航和标题元素。使用 split action bar，只需添加uiOptions="splitActionBarWhenNarrow"，到你的<activity>或<application>清单元素。

要知道Android在各种不同的方式，根据当前的屏幕大小调整操作栏的外观。采用分体式操作栏只是一个选项，您可以启用允许操作栏，以进一步为不同的屏幕尺寸，优化用户体验。这样做，你也可以让操作栏可以折叠成主要的操作栏导航标签。也就是说，如果你在你的动作条中使用的导航标签，一旦操作项狭窄的屏幕上分离，导航标签可以融入的主要操作栏，而不是被分隔成的“折叠的操作栏”。具体来说，如果你禁用操作栏中的图标和标题

`(setDisplayShowHomeEnabled (false)`

和`setDisplayShowTitleEnabled (false))`，然后将主要动作栏的导航标签收合，如图3中的第二个设备。



图3。模拟栏左侧的导航标签的分裂行动;与应用程序图标和标题右侧的禁用。

使用应用程序图标来导航

默认情况下，应用程序图标显示在操作栏的左边。你能够把这个图标当做操作项来使用。应用程序应该在这个图标上响应以下两个操作之一：

- 返回应用程序的“主”Activity；
- 向应用程序上级页面导航。

当用户触摸这个图标时，系统会调用Activity带有`android.R.id.home` ID的`onOptionsItemSelected()`方法。在这个响应中，你既可以启动主Activity，也可以返回你的应用程序结构化层次中用户上一步操作的界面。

如果你要通过应用程序图标的响应来返回主Activity，那么就应该在Intent对象中包括`FLAG_ACTIVITY_CLEAR_TOP`标识。用这个标记，如果你要启动的Activity在当前任务中已经存在，那么，堆栈中这个Activity之上的所有的Activity都有被销毁，并且把这个Activity显示给用户。添加这个标识往往是重要的，因为返回主Activity相当与一个回退的动作，因此通常不应该再创建一个新的主Activity的实例，否则，最终可能会在当前任务中产生一个很长的拥有多个主Activity的堆栈。

例如，下例的`onOptionsItemSelected()`方法实现了返回应用程序的主Activity的操作：

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            // app icon in action bar clicked; go home
            Intent intent = new Intent(this, HomeActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

在用户从另一个应用程序进入当前Activity的情况下，你可能还想要添加`FLAG_ACTIVITY_NEW_TASK`标识。这个标识确保在用户返回主页或上级页面时，新的Activity不会被添加到当前的任务中，而是在属于你自己的应用程序的任务中启动。例如，如果用户通过被另一个应用程序调用的Intent对象启动了你的应用程序中的一个Activity，那么选择操作栏图标来返回主页或上级页面时，`FLAG_ACTIVITY_CLEAR_TOP`标识会在属于你的应用程序的任务中启动这个Activity（不是当前任务）。系统既可以用这个新的Activity做根Activity来启动

一个新的任务，也可以把存在后台的拥有这个Activity实例的一个既存任务带到前台来，并且目标Activity会接受onNewIntent()回调。因此，如果你的Activity要接收另一个应用程序的Intent对象，那么通常应该给这个Intent对象添加FLAG_ACTIVITY_NEW_TASK标识，如：

```
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |  
Intent.FLAG_ACTIVITY_NEW_TASK);
```

注意：如果你要使用应用图标来返回主页，要注意从Android4.0（API级别14）开始，必须通过调用setHomeButtonEnabled(true)方法确保这个图标能够作为一个操作项（在以前的版本，默认情况下，这个图标就能够作为一个操作项）。

向应用程序上级页面导航

作为传统的回退导航（把用户带回任务历史中的前一个窗口）的补充，你能够让操作栏图标提供向上级页面导航的功能，它应用把用户带回到你的应用程序的上级页面。例如，当前页面时你的应用程序层次比较深的一个页面，触摸应用程序图标应该返回返回上一级页面（当前页面的父页面）。 

图4. Email应用程序的标准图标（左）和向上导航图标（右）。系统会自动添加向上指示。

例如，图5演示了当用户从一个应用程序导航到一个属于不同应用程序的Activity时，“回退”按钮的行为。 

图5. 在从People（或Contacts）应用程序进入Email应用程序之后，回退按钮的行为。

但是，如果在编辑完邮件之后，想要停留在Email应用程序中，那么向上导航就允许你把用户导航到Email应用程序中编辑邮件页面的上级页面，而不是返回到前一个Activity。图6演示了这种场景，在这个场景中，用户进入到Email应用程序后，不是按回退按钮，而是按操作栏图标来向上导航。



图6. 从People应用进入Email应用后，向上导航的行为。

要是应用程序图标能够向上导航，就要在你的ActionBar中调用`setDisplayHomeAsUpEnabled(true)`方法。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);
    ActionBar actionBar = getActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
    ...
}
```

当用户触摸这个图标时，系统会调用带有`android.R.id.home` ID的`onOptionsItemSelected()`方法。

请集中要在Intent对象中使用`FLAG_ACTIVITY_CLEAR_TOP`标识，以便你不会这个父Activity存在的情况下，再创建一个新的实例。例如，如果你不使用`FLAG_ACTIVITY_CLEAR_TOP`标识，那么向上导航后，再按回退按钮，实际上会把用户带到应用程序的下级页面，这是很奇怪的。

注意：如果有很多用户能够到达应用程序中当前Activity的路径，那么，向上图标应该沿着当前Activity的实际启动路径逐步的向会导航。

添加操作视窗

操作视窗是作为操作项目按钮的替代品显示在操作栏中的一个可视构件。例如，如果你有一个用于搜索的可选菜单项，你可以用`SearchView`类来替代操作栏上的搜索按钮，如图7所示：



图7. 折叠（上）和展开（下）的搜索视窗的操作栏

要个菜单资源中的一个项目声明一个操作视窗，你既可以使`android:actionLayout`属性也`android:actionViewClass`属性来分别指定一个布局资源或要使用的可视构件类。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_search"
          android:title="@string/menu_search"
          android:icon="@drawable/ic_menu_search"
          android:showAsAction="ifRoom|collapseActionView"
          android:actionViewClass="android.widget.SearchView" />
</menu>
```

android:showAsAction属性也可包含“collapseActionView”属性值，这个值是可选的，并且声明了这个操作视窗应该被折叠到一个按钮中，当用户选择这个按钮时，这个操作视窗展开。否则，这个操作视窗在默认的情况下是可见的，并且即便在用于不适用的时候，也要占据操作栏的有效空间。

如果需要给操作视窗添加一些事件，那么就需要在onCreateOptionsMenu()回调执行期间做这件事。你能够通过调用带有菜单项ID的findItem()方法来获取菜单项，然后再调用getActionView()方法操作视窗中的元素。例如，使用以下方法获取上例中的搜索视窗构件。

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options, menu);
    SearchView searchView = (SearchView)
        menu.findItem(R.id.menu_search).getActionView();
    // Configure the search info and add any event listeners
    ...
    return super.onCreateOptionsMenu(menu);
}
```

处理可折叠的操作视窗

操作视窗让你在不改变Activity或Fragment的情况下，就可以给用户提供快捷的访问和丰富的操作。但是，默认情况下让操作视窗可见可能不太合适。要保证操作栏的空间（尤其是在小屏幕设备上运行时），你能够把操作视窗折叠进一个操作项按钮中。当用户选择这个按钮时，操作视窗就在操作栏中显示。被折叠的时候，如果你定义了android:showAsAction="ifRoom"属性，那么系统可能会把这个项目放到溢出菜单中，但是当用户选项了这个菜单项，它依然会显示在操作栏中。通过给android:showAsAction属性添加“collapseActionView”属性值，你能够让操作视窗可以折叠起来。

因为在用户选择这个项目时，系统会展开这个操作视窗，所以你没有必要在onOptionsItemSelected()回调方法中响应这个菜单项。在用户选择这个菜单项时，系统会依然调用onOptionsItemSelected()方法，但是除非你在方法中返

回了true（指示你已经替代系统处理了这个事件），否则系统会始终展开这个操作视窗。

当用户选择了操作栏中的“向上”图标或按下了回退按钮时，系统也会把操作视窗折叠起来。

如果需要，你能够在代码中通过在expandActionView()和collapseActionView()方法来展开或折叠操作视窗。

注意：尽管把操作视窗折叠起来是可选的，但是，如果包含了SearchView对象，我们推荐你始终把这个视窗折叠起来，只有在需要的时候，由用户选择后才把它给展开。在提供了专用的“搜索”按钮的设备上也要小心了，如果用户按下了“搜索”按钮，那么也应该把这个搜索视窗给展开，简单的重写Activity的onKeyUp()回调方法，监听KEYCODE_SEARCH类型的按键事件，然后调用expandActionView()方法，就可以把操作视窗给展开。

如果你需要根据操作视窗的可见性来更新你的Activity，那么你可以定义一个OnActionExpandListener事件，并且用setOnActionExpandListener()方法来注册这个事件，然后就能在操作视窗展开和折叠时接受这个回调方法了，如：

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options, menu);
    MenuItem menuItem = menu.findItem(R.id.actionItem);
    ...
    menuItem.setOnActionExpandListener(new OnActionExpandListener() {
        @Override
        public boolean onMenuItemActionCollapse(MenuItem item) {
            // Do something when collapsed
            return true; // Return true to collapse action view
        }
        @Override
        public boolean onMenuItemActionExpand(MenuItem item) {
            // Do something when expanded
            return true; // Return true to expand action view
        }
    });
}

```

添加一个操作提供器

与操作视窗类似，操作提供器（由**ActionProvider**类定义的）用一个定制的布局代替一个操作项目，它还需要对所有这些项目行为的控制。当你在操作栏中给一个菜单项声明一个操作项目时，它不仅要一个定制的布局来控制这个菜单项的外观，而且当它在显示在溢出菜单中时，还要处理它的默认事件。无论是在操作栏中还是在溢出菜单中，它都能够提供一个子菜单。

例如，**ActionProvider**的扩展类**ShareActionProvider**，它通过在操作栏中显示一个有效的共享目标列表来方便共享操作。与使用传统的调用**ACTION_SEND**类型Intent对象的操作项不同，你能够声明一个**ShareActionProvider**对象来处理一个操作项。这种操作提供器会保留一个带有处理**ACTION_SEND**类型Intent对象的应用程序的下拉列表，即使这个菜单项显示在溢出菜单中。因此，当你使用像这样的操作提供器时，你不必处理有关这个菜单项的用户事件。

要给一个操作项声明一个操作提供器，就要在菜单资源中对应的<item>元素中定义**android:actionProviderClass**属性，提供器要使用完整的类名。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_share"
          android:title="@string/share"
          android:showAsAction="ifRoom"

        android:actionProviderClass="android.widget.ShareActionProvider" />
    ...
</menu>
```

在这个例子中，用**ShareActionProvider**类作为操作提供器，在这里，操作提供器需要菜单项的控制，并处理它们在操作栏中的外观和行为以及在溢出菜单中的行为。你必须依然给这个菜单项提供一个用于溢出菜单的文本标题。

尽管操作提供器提供了它在溢出菜单中显示时所能执行的默认操作，但是**Activity**（或**Fragment**）也能够通过处理来自**onOptionsItemSelected()**回调方法的点击事件来重写这个默认操作。如果你不在这个回调方法中处理点击事件，那么操作提供器会接收**onPerformDefaultAction()**回调来处理事件。但是，如果操作提供器提供了一个子菜单，那么**Activity**将不会接收**onOptionsItemSelected()**回调，因为子菜单的显示替代了选择时调用的默认菜单行为。

使用**ShareActionProvider**类

如果你想要在操作栏中提供一个“共享”操作，以充分利用安装在设备上的其他

应用程序（如，把一张图片共享给消息或社交应用程序使用），那么使用**ShareActionProvider**类是一个有效的方法，而不是添加一个调用**ACTION_SEND**类型Intent对象的操作项。当你给一个操作项使用**ShareActionProvider**类时，它会呈现一个带有能够处理**ACTION_SEND**类型Intent对象的应用程序的下拉列表（如图8所示）。



图8. Gallery 应用截屏，用**ShareActionProvider**对象展开显示共享目标。创建子菜单的所有逻辑，包括共享目标的封装、点击事件的处理（包在溢出菜单中的项目显示）等，都在**ShareActionProvider**类中实现了---你需要编写的唯一的代码是给对应的菜单项声明操作提供器，并指定共享的Intent对象。

默认情况，**ShareActionProvider**对象会基于用户的使用频率来保留共享目标的排列顺序。使用频率高的目标应用程序会显示在下来列表的上面，并且最常用的目标会作为默认共享目标直接显示在操作栏。默认情况下，排序信息被保存在由**DEFAULT_SHARE_HISTORY_FILE_NAME**指定名称的私有文件中。如果你只使用一种操作类型的**ShareActionProvider**类或它的子类，那么你应该继续使用这个默认的历史文件，而不需要做任何事情。但是，如果你使用了不同类型的多个操作的**ShareActionProvider**类或它的子类，那么为了保持它们自己的历史，每种**ShareActionProvider**类都应该指定它们自己的历史文件。给每种**ShareActionProvider**类指定不同的历史文件，就要调用**setShareHistoryFileName()**方法，并且提供一个XML文件的名字（如，`custom_share_history.xml`）

注意：尽管**ShareActionProvider**类是基于使用频率来排列共享目标的，但是这种行为是可扩展的，并且**ShareActionProvider**类的扩展能够基于历史文件执行不同的行为和排序。

要添加**ShareActionProvider**对象，只需简单的给**android.actionProviderClass**属性设定**android.widget.ShareActionProvider**属性值就可以了。唯一要做的事情是定义你要用于共享的Intent对象，你必须先调用**getActionProvider()**方法来获取跟菜单项匹配的**ShareActionProvider**对象，然后调用**setShareIntent()**方法。

如果对于共享的Intent对象的格式依赖与被选择的菜单项，或其他的在Activity生存周期内改变的变量，那么你应该把**ShareActionProvider**对象保存

在一个成员属性里，并在需要的时候调用`setShareIntent()`方法来更新它。如：

```

private ShareActionProvider mShareActionProvider;
...

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    mShareActionProvider = (ShareActionProvider)
menu.findItem(R.id.menu_share).getActionProvider();

    // If you use more than one ShareActionProvider, each for a
different action,
    // use the following line to specify a unique history file for each
one.
    //
mShareActionProvider.setShareHistoryFileName("custom_share_history.xml");

    // Set the default share intent
    mShareActionProvider.setShareIntent(getDefaultShareIntent());

    return true;
}
// When you need to update the share intent somewhere else in the app,
call
// mShareActionProvider.setShareIntent()

```

上例中`ShareActionProvider`对象处理所有的跟这个菜单项有关的用户交互，并且不需要处理来自`onOptionsItemSelected()`回调方法的点击事件。

创建一个定制的操作提供器

当你想要创建一个有动态行为和在溢出菜单中有默认图标的操作视窗时，，继承`ActionProvider`类来定义这些行为是一个比好的的方案。创建自己的操作提供器，提供一个有组织的可重用的组件，而不是在`Fragment`或`Activity`的代码中处理各种操作项的变换和行为。

要创建自己的操作提供器，只需简单的继承`ActionProvider`类，并且实现合适的回调方法。你应该实现以下重要的回调方法：`ActionProvider()`

这个构造器把应用程序的`Context`对象传递个操作提供器，你应该把它保存在一个成员变量中，以便其他的回调方法使用。

OnCreateActionView()

这是你给菜单项定义操作视窗的地方。使用从构造器中接收的`Context`对象，获取一个`LayoutInflater`对象的实例，并且用`XML`资源来填充操作视窗，然后注册事件监听器。如：

```

public View onCreateActionView() {
    // Inflate the action view to be shown on the action bar.
    LayoutInflater layoutInflater = LayoutInflater.from(mContext);
    View view = layoutInflater.inflate(R.layout.action_provider, null);
    ImageButton button = (ImageButton) view.findViewById(R.id.button);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Do something...
        }
    });
    return view;
}

```

onPerformDefaultAction()

在选中溢出菜单中的菜单时，系统会调用这个方法，并且操作提供器应该这对这个选中的菜单项执行默认的操作。

但是，如果你的操作提供器提供了一个子菜单，即使是溢出菜单中一个菜单项的子菜单，那么也要通过onPrepareSubMenu()回调方法来显示子菜单。这样onPerformDefaultAction()在子菜单显示时就不会被调用。

注意：实现了onOptionsItemSelected()回调方法的Activity或Fragment对象能够通过处理item-selected事件（并且返回true）来覆盖操作提供器的默认行为，这种情况下，系统不会调用onPerformDefaultAction()回调方法。

添加导航选项标签

当你想要在一个Activity中提供导航选择标签时，使用操作栏的选项标签是一个非常好的选择（而不是使用TabWidget类），因为系统会调整操作栏选项标签来适应不同尺寸的屏幕的需要---在屏幕足够宽的时候，导航选项标签会被放到主操作栏中；当屏幕太窄的时候，选项标签会被放到一个分离的横条中，如图9和图10所示。



图9. Honeycomb Gallery应用程序中的操作栏选项标签的截图



图10. 在窄屏设备上被堆放在操作栏中的选项标签的截图

要使用选项标签在Fragment之间切换，你必须在每次选择一个选项标签时执行一个Fragment事务。如果你不熟悉如何使用FragmentTransaction对象来改变Fragment，请阅读Fragment开发指南。

首先，你的布局必须包含一个用于放置跟每个Fragment对象关联的选项标签的ViewGroup对象。并且要确保这个ViewGroup对象有一个资源ID，以便你能够在选项标签的切换代码中能够引用它。另外，如果选项标签的内容填充在Activity的布局中（不包括操作栏），那么Activity不需要任何布局（你甚至不需要调用setContentView()方法）。相反，你能够把每个Fragment对象放到默认的根ViewGroup对象中，你能够用android.R.id.content ID来引用这个ViewGroup对象（在Fragment执行事务期间，你能够在下面的示例代码中看到如何使用这个ID的）。

决定了Fragment对象在布局中的显示位置后，添加选项标签的基本过程如下：

1. 实现ActionBar.TabListener接口。这个接口中回调方法会响应选项标签上的用户事件，以便你能够切换Fragment对象；
2. 对于每个要添加的选项标签，都要实例化一个ActionBar.Tab对象，并且调用setTabListener()方法设置ActionBar.Tab对象的事件监听器。还可以用setText()或setIcon()方法来设置选项标签的标题或图标。
3. 通过调用addTab()方法，把每个选项标签添加到操作栏。

在查看ActionBar.TabListener接口时，注意到回调方法只提供了被选择的ActionBar.Tab对象和执行Fragment对象事务的FragmentTransaction对象--没有说明任何有关Fragment切换的事。因此。你必须定义自己的每个ActionBar.Tab之间的关联，以及ActionBar.Tab所代表的适合的Fragment对象（为了执行合适的Fragment事务）。依赖你的设计，会有几种不同的方法来定义这种关联。在下面的例子中，ActionBar.TabListener接口的实现提供了一个构造器，这样每个新的选项标签都会使用它自己的监听器实例。每个监听器实例都定义了几个在对应Fragment对象上执行事务时必须的几个成员变量。

例如，以下示例是ActionBar.TabListener接口的一种实现，在这个实现中，每个选项标签都使用了它自己的监听器实例：

```
public static class TabListener<T extends Fragment> implements
ActionBar.TabListener {
    private Fragment mFragment;
    private final Activity mActivity;
```

```

private final String mTag;
private final Class<T> mClass;

/** Constructor used each time a new tab is created.
 * @param activity The host Activity, used to instantiate the
fragment
 * @param tag The identifier tag for the fragment
 * @param cls The fragment's Class, used to instantiate the
fragment
 */
public TabListener(Activity activity, String tag, Class<T> cls) {
    mActivity = activity;
    mTag = tag;
    mClass = cls;
}

/* The following are each of the ActionBar.TabListener callbacks */

public void onTabSelected(Tab tab, FragmentTransaction ft) {
    // Check if the fragment is already initialized
    if (mFragment == null) {
        // If not, instantiate and add it to the activity
        mFragment = Fragment.instantiate(mActivity,
mClass.getName());
        ft.add(android.R.id.content, mFragment, mTag);
    } else {
        // If it exists, simply attach it in order to show it
        ft.attach(mFragment);
    }
}

public void onTabUnselected(Tab tab, FragmentTransaction ft) {
    if (mFragment != null) {
        // Detach the fragment, because another one is being
attached
        ft.detach(mFragment);
    }
}

public void onTabReselected(Tab tab, FragmentTransaction ft) {
    // User selected the already selected tab. Usually do nothing.
}
}

```

警告：针对每个回调中的Fragment事务，你都不必调用commit()方法---系统会调用这个方法，并且如果你自己调用了这个方法，有可能会抛出一个异常。你也不能把这些Fragment事务添加到回退堆栈中。

在这个例子中，当对应的选项标签被选择时，监听器只是简单的把一个Fragment对象附加（attach()方法）到Activity布局上---或者，如果没有实例化，就会创建这个Fragment对象，并且把它添加（add()方法）到布局中（android.R.id.content ViewGroup的一个子类），当这个选项标签解除选择

时，对应的Fragment对象也会被解除与布局的依附关系。

ActionBar.TabListener的实现做了大量的工作，剩下的事情就是创建每个ActionBar.Tab对象并把它添加到ActionBar对象中，另外，你必须调用setNavigationMode(NAVIGATION_MODE_TABS)方法来让选项标签可见。如果选项标签的标题实际指示了当前的View对象，你也可以通过调用setDisplayShowTitleEnabled(false)方法来禁用Activity的标题。

例如，下面的代码使用上面定义的监听器在操作栏中添加了两个选项标签。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Notice that setContentView() is not used, because we use the root
    // android.R.id.content as the container for each fragment

    // setup action bar for tabs
    ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
    actionBar.setDisplayShowTitleEnabled(false);

    Tab tab = actionBar.newTab()
        .setText(R.string.artist)
        .setTabListener(new TabListener<ArtistFragment>(
            this, "artist", ArtistFragment.class));
    actionBar.addTab(tab);

    tab = actionBar.newTab()
        .setText(R.string.album)
        .setTabListener(new TabListener<AlbumFragment>(
            this, "album", AlbumFragment.class));
    actionBar.addTab(tab);
}
```

注意：以上有关ActionBar.TabListener的实现，只是几种可能的技术之一。在API Demos应用中你能够看到更多的这种样式。

<http://developer.android.com/resources/samples/ApiDemos/src/com/example>

如果Activity终止了，那么你应该保存当前选择的选项标签的状态，以便当用户再次返回时，你能够打开合适的选项标签。在保存状态的时刻，你能够用getSelectedNavigationIndex()方法查询当前的被选择的选项标签。这个方法返回被选择的选项标签的索引位置。

警告：保存每个**Fragment**所必须的状态是至关重要的，因为当用户用选项标签在**Fragment**对象间切换时，它会查看**Fragment**在离开时样子。

注意：在某些情况下，Android系统会把操作栏选项标签作为一个下拉列表来显示，以便确保操作栏的最优化显示。

添加下拉式导航

作为**Activity**内部的另一种导航（或过滤）模式，操作栏提供了内置的下拉列表。下拉列表能够提供**Activity**中内容的不同排序模式。

启用下拉式导航的基本过程如下： 1. 创建一个给下拉提供可选项目的列表，以及描画列表项目时所使用的布局； 2. 实现**ActionBar.OnNavigationListener**回调，在这个回调中定义当用户选择列表中一个项目时所发生的行为； 3. 用**setNavigationMode()**方法该操作栏启用导航模式，如：

```
ActionBar actionBar = getActionBar();
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
```

Note: You should perform this during your activity's onCreate() method.

4. 用**setListNavigationCallbacks()**方法给下拉列表设置回调方法，如：

```
actionBar.setListNavigationCallbacks(mSpinnerAdapter,
mNavigationCallback);
```

这个方法需要**SpinnerAdapter**和**ActionBar.OnNavigationListener**对象。

That's the basic setup. However, implementing the **SpinnerAdapter** and **ActionBar.OnNavigationListener** is where most of the work is done. There are many ways you can implement these to define the functionality for your drop-down navigation and implementing various types of **SpinnerAdapter** is

beyond the scope of this document (you should refer to the `SpinnerAdapter` class reference for more information). However, below is a simple example for a `SpinnerAdapter` and `ActionBar.OnNavigationListener` to get you started (click the title to reveal the sample).

Example `SpinnerAdapter` and `OnNavigationListener`

设置操作栏的样式

如果你对应用程序中的可视构件进行了定制化的设计，那么你可能也会要对操作栏做一些重新设计，以便跟应用程序的设计匹配。要这样做的话，需要使用Android的样式与主题框架中的一些特殊的样式属性来重新设置操作栏的样式。

注意：改变外观的背景图片依赖与当前按钮的状态（选择、按下、解除选择），因此你使用的可描画的资源必须是一个可描画的状态列表。

警告：对于你提供的所有可描画的背景，要确保使用NinePatch类型可描画资源，以便允许图片的拉伸。NinePatch类型的图片应该比40像素高30像素宽的图片要小。

普通的外观

`android:windowActionBarOverlay`

这个属性声明了操作栏是否应该覆盖Activity布局，而不是相对Activity的布局位置的偏移。这个属性的默认值是`false`。通常，在屏幕上，操作栏需要它自己的空间，并且把剩下的空间用来填充Activity的布局。当操作栏四覆盖模式时，Activity会使用所有的有效空间，系统会在Activity的上面描画操作栏。如果你想要在操作栏隐藏和显示时，布局中的内容保持固定的尺寸好位置，那么这种覆盖模式是有用的。你也可能只是为了显示效果来使用

它，因为你可以给操作栏设置半透明的背景，以便用户依然能够看到操作栏背后的Activity布局。

注意：默认情况下，Holo主题会用半透明背景来描画操作栏。但是，你能够用自己的样式来修改它，并且默认的情况下，DeviceDefault主题在不同的设备上可能使用不透明的背景。

覆盖模式被启用时，Activity布局不会感知到操作栏覆盖在它的上面，因此，在操作栏覆盖的区域，最好不要放置一些重要的信息或UI组件。如果适合，你能够引用平台的 actionBarSize 值来决定操作栏的高度，例如，在 XML 布局文件中引用这个值。

```
<SomeView
    ...
    android:layout_marginTop= "?android:attr/actionBarSize" />
```

你还能够用 getHeight() 方法在运行时获取操作栏的高度。如果在 Activity 生存周期的早期调用这个方法，那么在调用时所反映的操作栏的高度可能不包括被堆叠的操作栏（因为导航选项标签）。要看如何在运行时判断操作栏总的高度（包括被堆叠的操作栏），请看 Honeycomb Gallery 示例应用中的 TitlesFragment 类。

<http://developer.android.com/resources/samples/HoneycombGallery/index.html>

操作项样式

android:actionButtonStyle

给操作项按钮定义样式资源。

android: actionBarItemBackground

给每个操作项的背景定义可描画资源（被添加在 API 级别 14 中）。

android: itemBackground

给每个溢出菜单项的背景定义可描画资源。

android: actionBarDivider

给操作项之间的分隔线定义可描画资源（被添加在API级别14中）

android:actionMenuTextColor

给显示在操作项中文本定义颜色。

android:actionMenuTextAppearance

给显示在操作项中文本定义样式资源。

android: actionBarWidgetThem

给作为操作视窗被填充到操作栏中的可视构件定义主题资源（被添加在API级别14中）。

导航选项标签样式

android: actionBarTabStyle

给操作栏中的选项标签定义样式资源。

android: actionBarTabBarStyle

给显示在导航选项标签下方的细条定义样式资源。

android: actionBarTabTextStyle

给导航选项标签中的文本定义样式资源。

下拉列表样式

android: actionBarDropDownStyle

给下拉导航列表定义样式（如背景和文本样式）。

如，下例XML文件中给操作栏定义了一些定制的样式：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActivityTheme"
parent="@android:style/Theme.Holo">
        <item
name="android:actionBarTabTextStyle">@style/CustomTabTextStyle</item>
```

```

<item
    name="android:actionBarDivider">@drawable/ab_divider</item>
<item
    name="android:actionBarItemBackground">@drawable/ab_item_background</item>

</style>

<!-- style for the action bar tab text -->
<style name="CustomTabTextStyle"
parent="@android:style/TextAppearance.Holo">
    <item name="android:textColor">#2456c2</item>
</style>
</resources>

```

注意：一定要在<style>标签中声明一个父主题，这样定制的主题可以继承所有没有明确声明的样式。在修改操作栏样式时，使用父主题是至关重要的，它会让你能够简单的覆写你想要改变的操作栏样式，而不影响你不想修改的样式（如文本的外观或操作项的边缘）。

你能够在清单文件中把定制的主题应用到整个应用程序或一个单独的Activity对象，如：

```
<application android:theme="@style/CustomActivityTheme" ... />
```

高级样式

如果需要比上述属性更高级的样式，可以在Activity的主题中包含`android: actionBarStyle`和`android: actionBarSplitStyle`属性。这两个属性的每一个都指定了另一种能够给操作栏定义各种属性的样式，包括带有`android: background`、`android: backgroundSplit`、`android: backgroundStacked`属性的不同背景。如果要覆盖这些操作栏样式，就要确保定义一个像`Widget.Holo.ActionBar`这样的父操作栏样式。

例如，如果要改变操作栏背景，你可以使用下列样式：

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- the theme applied to the application or activity -->
    <style name="CustomActivityTheme"
parent="@android:style/Theme.Holo">
        <item name="android: actionBarStyle">@style/MyActionBar</item>
        <!-- other activity and action bar styles here -->
</style>

```

```
<!-- style for the action bar backgrounds -->
<style name="MyActionBar"
parent="@android:style/Widget.Holo.ActionBar">
    <item name="android:background">@drawable/ab_background</item>
    <item
name="android:backgroundStacked">@drawable/ab_background</item>
    <item
name="android:backgroundSplit">@drawable/ab_split_background</item>
</style>
</resource>
```

来自“[index.php?title=Action_Bar&oldid=7961](#)”

Notifications

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/topics/ui/notifiers/index.html>

翻译： [Gavin Zhuang](#)

更新： 2012.06.08

目录

- [1 通知 - Notifications](#)
 - [1.1 Toast 通知](#)
 - [1.2 状态栏通知](#)
 - [1.3 对话框通知](#)

通知 - Notifications

在某些情况下，可能需要你去通知用户发生在你应用中的事件，其中一些事件需要用户响应，有的则不需要。例如：

- 当一个事件完成时（比如保存一个文件），则需要显示一个简短的消息来确认保存成功。
- 假如应用正在后台运行且需要用户注意，那么该应用需要创建一个通知以方便用户做出响应。

假如应用正在执行某个动作（比如正在载入一个文件）且需要用户等待，那么该应用需要显示一个旋转的进度条来表示这个过程。

以上这些通知任务，每一个都可以用不同的技术来实现：

- [Toast通知](#)，是从后台启用一个简短的消息；
- [状态栏通知](#)，是来自后台的持续提醒且需要用户响应；
- [对话框通知](#)，是一种与Activity（活动）相关的通知。

这份文档包括了这些通知用户的相关技术的介绍以及完整文档的链接。

Toast 通知

Toast通知是一种浮现
在屏幕上层的消息提
醒，它只填充消息所
需要的空间，而当前
正在运行的活动仍然
保持其自身的可见性
和交互性。这种通知
自动淡入淡出且不接
受交互事件，因为它是
是由后台服务创建的，
所以即时应用不可见了
它仍然能够显示。



当你完全将注意力集
中在屏幕上时，那
么Toast通知是最好的
提示简短消息的方式

(例如文件保存成功提醒)。这种通知不接受用户的交互事件，但假如你想让用户去响应和做出动作，你可以考虑使用[状态栏通知](#)来代替。

详细信息，请参阅[Toast通知](#)。

状态栏通知

状态栏通知是将图标添加到系统的状态栏（带有一个可选的滚动文本消息），同时将扩展信息添加到“通知”窗口。当用户选择这个扩展信息时，安卓设备将触发一个由通知定义的Intent（通常是载入一个活动）。你还可以为这个通知配置声音、震动以及闪光灯来提醒用户。



这种模式的通知是在当你的应用运行在后台服务中且需要用户注意到这个事件时使用的。假如你需要提

醒用户正在发生的事
件，且这个事件正持
续进行时，你可以考
虑使用[对话框通知](#)来
代替。

详细信息，请参阅[状
态栏通知](#)。

对话框通知



对话框通常是一种显
示在当前活动之上的
小窗口，这时候下层
的活动将失去焦点，
且对话框可以接受任
何形式的用户交互方
式。一般来说对话框
是用于通知或者用于
直接关系到应用进程
的短期活动。

当你需要显示一个进
度条或者一个需要用
户确认的短消息（例
如带有“确定”和“取
消”按钮的提醒）时，
你可以利用对话框来
实现。你还可以把它
当作应用的用户界面
和除了通知以外其他
目的的交互元件。所

有可用类型的对话框
(包括用于通知) 的
完整讨论, 请参阅[对话框](#)。

[↑返回顶部](#)

[←返回用户接口](#)

来自“[index.php?title=Notifications&oldid=8801](#)”

Toast Notifications

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/topics/ui/notifiers/toasts.html>

编辑者：[IBrave](#)

更新时间：6月14日

目录

[[隐藏](#)]

[1 Toast通知 - Toast Notifications](#)

- [1.1 Toast基础](#)
- [1.2 Toast定位](#)
- [1.3 创建自定义Toast视图](#)

Toast通知 - Toast Notifications

Toast通知是在窗口表面弹出的一个简短的小消息。它只填充消息所需要的空
间，并且用
户当前

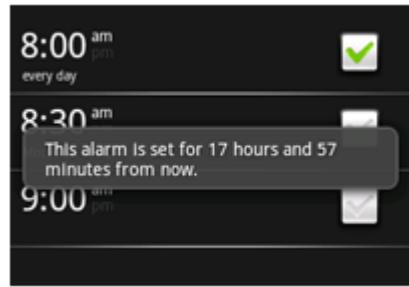
快速浏览

toast是一个在屏幕上显示片刻的提示消息，但是toast不能获得焦点(或者暂停当前运行的活动(activity)),所以它不能接受用户的输入。

你可以自定义包括图像的toast布

的Activity依然保持可见性和交互性。这种通知可自动的淡入淡出，且不接受用户的交互事件。

右图显示的是闹钟的Toast例子，一旦某个闹钟被打开，就会显示一条表示你对其设置的Toast通知。



Toast通知能够被[Activity](#)或[Service](#)创建并显示。如果你创建了一个源自Service的Toast通知，它会显示在当前的Activity最上层。

如果用户需要对通知做出响应，请考虑使用[状态栏通知\(Status Bar Notification\)](#)。

Toast基础

首先，用某个[makeText\(\)](#)方法实例化一个[Toast对象](#)。这个方法有三个参数：1.应用程序的上下文[Context](#)、2.要显示的文本消息；3.Toast通知持续显示的时间。这个方法将会返回一个合适的且被初始化的Toast对象。你可以用[show\(\)方法](#)显示Toast通知，例子如下：

```
<font color="purple">Context</font> context =  
getApplicationContext();
```

局(layout)。

文档内容

- 1. [Toast 基础](#)
- 2. [Toast 定位](#)
- 3. [创建自定义Toast视图](#)

[主要的类](#)

[Toast](#)

```

<font color="purple">CharSequence</font> text = <font
color="green">"Hello toast!"</font>;
<font color="blue">int</font> duration = <font
color="purple">Toast</font>.LENGTH_SHORT;

<font color="purple">Toast</font> toast = <font
color="purple">Toast</font>.makeText(context, text, duration);
toast.show();

```

这个示例给你演示了大多数Toast通知需要的内容，有了这些应该很少需要用到其他的内容。但是，你想要把Toast通知放到不同的位置显示，甚至要使用自己的布局来替代那个简单的文本消息。下一节，将向你介绍如何实现这些想法。

提示：你也可以用链式组合方法写且避免创建Toast对象，向下面这样：

```
Toast.makeText(context, text, duration).show();
```

Toast定位

标准的Toast通知水平居中显示在屏幕底部附近，可以通过setGravity(int, int, int)方法来重新设置显示位置。这个方法有三个参数： 1. [Gravity常量](#)（详细参照Gravity类）； 2.X轴偏移量； 3.Y轴偏移量。

例如：如果你想让Toast通知显示在屏幕的左上角，可以这样设置setGravity(int ,int ,int)方法：

```
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);
```

如果想让位置向右移，可以增加第二个参数的值，要向下移动，可以增加最后一个参数的值。

创建自定义Toast视图

如果一个简单的文本消息不能满足现实的需要，你可以给Toast通知创建一个自定义的布局(layout)。要创建一个自定义的布局(layout)，可以在XML文件或程序代码中定义一个View布局，然后把(根)View对象传递给setView(View)方法。

例如，你可以用下面的XML文件创建一个如图所示的Toast通知视图。

```
<LinearLayout xmlns:android=
```

```

"http://schemas.android.com/apk/res/android"

android:id="@+id/toast_layout_root"

android:orientation="horizontal"

android:layout_width="fill_parent"

android:layout_height="fill_parent"
    android:padding="10dp"
    android:background="#DAAA"
    >
<ImageView android:id="@+id/image"

android:layout_width="wrap_content"

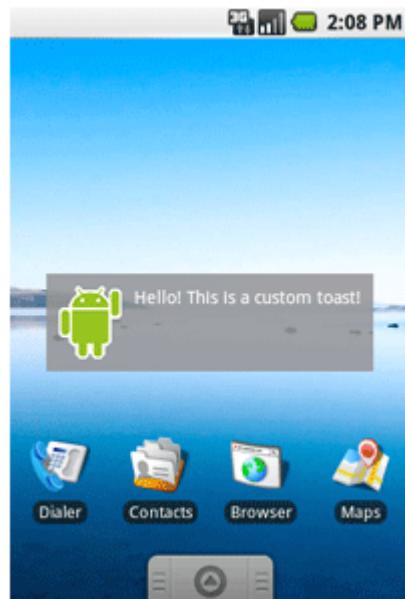
android:layout_height="fill_parent"

android:layout_marginRight="10dp"
    />
<TextView android:id="@+id/text"

android:layout_width="wrap_content"

android:layout_height="fill_parent"
    android:textColor="#FFF"
    />
</LinearLayout>

```



注意:LinearLayout元素的ID属性值是“toast_layout_root”。你必须使用这个ID的把XML的定义填充到布局(layout)中，方法如下：

```

LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout,
    (ViewGroup)
findViewById(R.id.toast_layout_root));

ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Hello! This is a custom toast!");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();

```

首先，用getLayoutInflator()方法或getSystemService()方法获取LayoutInflater对象，然后使用inflate(int, ViewGroup)方法把XML定义填

充到布局(layout)中，这个方法的第一个参数是布局(layout)资源的ID，第二个参数是被填充的布局(layout)View对象，本例中是root View对象。你能够使用这个被填充的布局来查找更多的View对象，以便获取和定义ImageView和TextView元素的内容。最后，用Toast(Context)方法创建一个Toast对象，并设置一些Toast的属性，如Gravity常量和持续显示的时间等。然后调用setView(View)方法，把它传递给要填充的布局(layout)对象。然后调用show()方法显示自定义的Toast。

注意：除非你要用setView(View)方法定义布局(layout)，否则不要使用公共的Toast类构造器。如果不使用自定义的布局(layout)，必须使用makeText(Context, int, int)方法来创建Toast

来自“[index.php?title=Toast_Notifications&oldid=8807](#)”



Status Notifications

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：Kobehonghai

原文地址

址<http://developer.android.com/guide/topics/ui/notifiers/notifications.html>

目录

- [1 状态通知--Status Notifications](#)
- [2 基础-The Basics](#)
- [3 响应通知-Responding to Notifications](#)
- [4 管理通知-Managing your Notifications](#)
- [5 创建通知-Creating a Notification](#)
- [6 更新通知-Updating the notification](#)
- [7 添加声音-Adding a sound](#)
- [8 添加振动-Adding vibration](#)
- [9 添加闪灯-Adding flashing lights](#)
- [10 更多特性-More features](#)
- [11 自定义通知的布局-Creating a Custom Notification Layout](#)

状态通知--Status Notifications

一个状态通知添加一个图标到系统的状态栏（带有可选的ticker-text消息）和通知消息到通知窗口。当用户选择了一个通

快速预览

知，Android会启动一个[Intent](#)，这个Intent是由[Notification](#)定义的（通常是为了启动一个[Activity](#)）你也可以自定义通知来通知用户，例如用声音文件，振动或者是闪动的通知灯。

当一个后台服务需要通知用户一个需要反应的事件的时候，状态通知就要被用到了。后台服务永远不需要启动一个activity与用户互动，而是应该创建一个状态通知--当用户选择的时候可以启动一个activity。

如图1 在状态栏的左边有一个通知的图标



图1 状态栏上的图标

如图2 显示在通知窗口里的信息



图2 通知窗口

设计-**Notification Design**

想了解设计准则，请阅读[Android Design's Notifications](#)指南

一个状态通知允许应用程序通知用户一个事件但不扰乱他们的当前活动

- 你可以把一个intent和通知绑定在一起，这样当用户点击通知选项时系统就可以进行初始化

本文内容

[基础-The Basics](#)

[响应通知-Responding to Notifications](#)

[管理通知-Managing your Notifications](#)

[创建通知-Creating a Notification](#)

[更新通知-Updating the notification](#)

[添加声音-Adding a sound](#)

[添加振动-Adding vibration](#)

[添加闪灯-Adding flashing lights](#)

[更多特性-More features](#)

[自定义通知的布局-Creating a Custom Notification Layout](#)

关键类

[Notification](#)

[NotificationManager](#)

请参阅

[Android Design: Notifications](#)

基础-The Basics

一个[Activity](#)或者[Service](#)可以初始化一个状态通知。然而，因为一个activity只有运行在前台并且它所在的窗口获得焦点的时候，它才能执行动作，所以你通常需要用service来创建通知。当用户正在使用其他应用或者设备待机时，通知可以由后台创建。为了创建通知，你必须用到两个类：[Notification](#)和[NotificationManager](#)。

使用[Notification](#)类的一个实例去定义状态通知的属性，如图标，通知信息和一些其他的设定，如播放的声音。[NotificationManager](#)是系统服务，它来执行和管理所有的状态通知。你不需要直接初始化[NotificationManager](#)。为了把你的通知给它，你必须使用[getSystemService\(\)](#)检索到指向[NotificationManager](#)的引用，然后，当你要通知用户的时候，使用[android.app.Notification.notify\(\)](#)把你的[Notification](#)传给它。

按如下方式创建状态通知：

1. 得到指向[NotificationManager](#)的引用：

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(ns);
```

2. 初始化[Notification](#)：

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Hello";
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText,
when);
```

3. 定义通知的信息和[PendingIntent](#)：

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
```

```
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);

notification.setLatestEventInfo(context, contentTitle, contentText,
contentIntent);
```

4. 把[Notification](#)传递给[NotificationManager](#):

```
private static final int HELLO_ID = 1;

mNotificationManager.notify(HELLO_ID, notification);
```

响应通知-**Responding to Notifications**

关于通知，用户体验的核心围绕在怎样使它与应用程序的界面相结合。你必须正确地去实现它以便于为你的应用带来统一的用户体验。

日历应用和电子邮件的两个典型的通知例子：活动即将开始的通知；新邮件到来的通知。他们所展现的是两个被推荐的处理通知的形式：或者启动一个与应用无关的独立的activity，或者调用一个应用的全新的实例。

接下来的场景展示了activity栈应该如何在这两种典型的通知流里面工作，首先来操作日历的通知：

1. 用户正在日历应用里创建一个新的活动。他们意识到需要复制一部分电子邮件消息到这个活动中来。
2. 用户选择Home>Email
3. 当操作Email应用时，用户收到从日历应用那里收到一个即将开始的会议的通知。
4. 用户选择了通知，他们开始关注那个日历应用所展现出来的简要的会议信息。
5. 用户已经确定了他们即将有个会议，所以他们按了返回键。他们现在返

回了Email应用--接到通知的地方。

操作电子邮件的通知：

1. 目前用户正在电子邮件应用里撰写邮件，需要核对一下日期。
2. 用户选择Home>Calendar (日历应用)
3. 而在日历应用中，他们收到电子邮件关于一个新的讯息的通知。
4. 用户选择了通知，这使他们回到了电子邮件展示详细信息的页面。这取代了他们以前所做的（写信），但这信件仍然保存在草稿箱中。
5. 用户按下了Back键，来到了信件列表页面，然后按下Back键返回日历应用。

在电子邮件那样风格的通知中，按下通知所调出的UI以一种形式展示出通知的应用。例如，当电子邮件应用通过通知来到前台展示的时候，他展示出的是一个列表还是一个具体信件信息要依赖于新邮件的数量，是多封还是一封。为了达到这种效果，我们需要用一个新的activity栈所展现出的新通知的状态来完全代替当下的任何一种状态。

下面的代码举例说明了显示这种通知。最关键的是[makemessageintentstack\(\)](#)方法，这个方法构建了一个由intent组成的数据-代表了在这种状态下，应用的新的activity栈（如果你正在使用片段(fragments)，你可能需要初始化你的片段和应用的状态，从而按下Back键时UI会返回到前一次的状态），其中的核心是[Intent.makeRestartActivityTask\(\)](#)，它在栈里构建了根activity，并使用了合适的标志(flag)，如[Intent.FLAG_ACTIVITY_CLEAR_TASK](#)

```
/**
 * This method creates an array of Intent objects representing the
 * activity stack for the incoming message details state that the
 * application should be in when launching it from a notification.
 */
static Intent[] makeMessageIntentStack(Context context, CharSequence
from,
    CharSequence msg) {
    // A typical convention for notifications is to launch the user
deeply
    // into an application representing the data in the
notification; to
```

```

// accomplish this, we can build an array of intents to insert
the back
// stack stack history above the item being displayed.
Intent[] intents = new Intent[4];

// First: root activity of ApiDemos.
// This is a convenient way to make the proper Intent to launch
and
// reset an application's task.
intents[0] = Intent.makeRestartActivityTask(new
ComponentName(context,
    com.example.android.apis.ApiDemos.class));
}

// "App"
intents[1] = new Intent(context,
com.example.android.apis.ApiDemos.class);
intents[1].putExtra("com.example.android.apis.Path", "App");
// "App/Notification"
intents[2] = new Intent(context,
com.example.android.apis.ApiDemos.class);
intents[2].putExtra("com.example.android.apis.Path",
"App/Notification");

// Now the activity to display to the user. Also fill in the
data it
// should display.
intents[3] = new Intent(context, IncomingMessageView.class);
intents[3].putExtra(IncomingMessageView.KEY_FROM, from);
intents[3].putExtra(IncomingMessageView.KEY_MESSAGE, msg);

return intents;
}

/**
 * The notification is the icon and associated expanded entry in the
 * status bar.
 */
void showAppNotification() {
    // look up the notification manager service
    NotificationManager nm =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    // The details of our fake message
    CharSequence from = "Joe";
    CharSequence message;
    switch ((new Random()).nextInt()) % 3) {
        case 0: message = "r u hungry? i am starved"; break;
        case 1: message = "im nearby u"; break;
        default: message = "kthx. meet u for dinner. cul8r"; break;
    }

    // The PendingIntent to launch our activity if the user selects
this
    // notification. Note the use of FLAG_CANCEL_CURRENT so that,
if there
    // is already an active matching pending intent, cancel it and
replace
    // it with the new array of Intents.
    PendingIntent contentIntent = PendingIntent.getActivity(this,
0,
    makeMessageIntentStack(this, from, message),

```

```

Status Notifications - eoeAndroid wiki
PendingIntent.FLAG_CANCEL_CURRENT);

    // The ticker text, this uses a formatted string so our message
    // could be localized
    String tickerText =
getString(R.string.imcoming_message_ticker_text, message);

    // construct the Notification object.
    Notification notif = new Notification(R.drawable.stat_sample,
tickerText,
        System.currentTimeMillis());

    // Set the info for the views that show in the notification
    // panel.
    notif.setLatestEventInfo(this, from, message, contentIntent);

    // We'll have this notification do the default sound, vibration,
    // and led.
    // Note that if you want any of these behaviors, you should
    // always have
    // a preference for the user to turn them off.
    notif.defaults = Notification.DEFAULT_ALL;

    // Note that we use R.layout.incoming_message_panel as the ID
    // for
    // the notification. It could be any integer you want, but we
    // use
    // the convention of using a resource id for a string related to
    // the notification. It will always be a unique number within
    // your
    // application.
    nm.notify(R.string.imcoming_message_ticker_text, notif);
}

```

在日历应用风格的通知中，被通知调用的UI是一个专用的activity，而不是应用流程中的一部分。例如，当用户接受到日历应用的通知时，点击查看，触发了一个特殊的activity来显示即将开始的活动的列表-这个界面只能由通知调用，而不能由日历应用中的界面进入。

这种通知的代码是很简单的，就像上面的代码，但是PendingIntent只能用于单个的activity,我们那个专门的通知activity。

```

/**
 * The notification is the icon and associated expanded entry in the
 * status bar.
 */
void showInterstitialNotification() {
    // look up the notification manager service
    NotificationManager nm =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    // The details of our fake message
    CharSequence from = "Dianne";
    CharSequence message;

```

```

switch ((new Random()).nextInt()) % 3) {
    case 0: message = "i am ready for some dinner"; break;
    case 1: message = "how about thai down the block?"; break;
    default: message = "meet u soon. dont b late!"; break;
}

// The PendingIntent to launch our activity if the user selects
this
// notification. Note the use of FLAG_CANCEL_CURRENT so that,
if there
// is already an active matching pending intent, cancel it and
replace
// it with the new Intent.
Intent intent = new Intent(this,
IncomingMessageInterstitial.class);
intent.putExtra(IncomingMessageView.KEY_FROM, from);
intent.putExtra(IncomingMessageView.KEY_MESSAGE, message);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
    intent, PendingIntent.FLAG_CANCEL_CURRENT);

// The ticker text, this uses a formatted string so our message
could be localized
String tickerText =
getString(R.string.imcoming_message_ticker_text, message);

// construct the Notification object.
Notification notif = new Notification(R.drawable.stat_sample,
tickerText,
    System.currentTimeMillis());

// Set the info for the views that show in the notification
panel.
notif.setLatestEventInfo(this, from, message, contentIntent);

// We'll have this notification do the default sound, vibration,
and led.
// Note that if you want any of these behaviors, you should
always have
// a preference for the user to turn them off.
notif.defaults = Notification.DEFAULT_ALL;

// Note that we use R.layout.incoming_message_panel as the ID
for
// the notification. It could be any integer you want, but we
use
// the convention of using a resource id for a string related to
// the notification. It will always be a unique number within
your
// application.
nm.notify(R.string.imcoming_message_ticker_text, notif);
}

```

然而这还不够。通常Android认为一个应用里的所有activity都是这个应用界面流程的一部分，所以像这样简单地调用activity能够导致：使这个activity被混入应用的返回栈中。为了让它正确的工作，在manifest声明的

时候，`android:launchMode="singleTask"`, `android:taskAffinity=""`和`android:excludeFromRecents="true"`是必须要设置的。例子如下：

```
<activity android:name=".app.IncomingMessageInterstitial"
    android:label="You have messages"
    android:theme="@style/ThemeHoloDialog"
    android:launchMode="singleTask"
    android:taskAffinity=""
    android:excludeFromRecents="true">
</activity>
```

你必须要小心翼翼地通过这个初始的activity来调用其他的activity，因为它不是应用的最上层，也没有在最近出现过，而且它需要被通知调用，任何时候。最好的方法就是确保任何被它调用的activity都是在它自己的任务中被调用的。当这么做的时候，必须要小心，确保这个新的任务和当前存在于应用中的任务能够很好的交互。这在本质上与之前提到的电子邮件风格的通知，切换到主界面的情况一样。想想之前的`makeMessageIntentStack()`，处理一个点击事件然后看起来是这样的：

```
/**
 * Perform a switch to the app. A new activity stack is started,
 * replacing whatever is currently running, and this activity is finished.
 */
void switchToApp() {
    // We will launch the app showing what the user picked. In this
    // simple
    // example, it is just what the notification gave us.
    CharSequence from =
    getIntent().getCharSequenceExtra(IncomingMessageView.KEY_FROM);
    CharSequence msg =
    getIntent().getCharSequenceExtra(IncomingMessageView.KEY_MESSAGE);
    // Build the new activity stack, launch it, and finish this UI.
    Intent[] stack = IncomingMessage.makeMessageIntentStack(this,
    from, msg);
    startActivities(stack);
    finish();
}
```

管理通知-Managing your Notifications

[NotificationManager](#)是系统服务，负责管理所有的通知。你必须使

用[getSystemService\(\)](#)得到它的引用。如下：

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(ns);
```

当你要发表通知时，要使用[notify\(int, Notification\)](#)把[Notification](#)传递给[NotificationManager](#)。方法的第一个参数是通知的唯一的ID，第二个参数就是[Notification](#)对象。**ID**唯一地标识了通知是来自你的应用程序。如果你需要更新通知或者（如果你的应用管理不同种类的通知）当用户通过通知中定义的[intent](#)返回到你的应用时，选择合适的[action](#)，那么**ID**就是必需的。

当用户点击选择通知后，为了清空通知，给你的[Notification](#)添加一个标志"**"FLAG_AUTO_CANCEL"**"，你也可以使用[cancel\(int\)](#)手动清空它，参数是通知的**ID**，或者使用[cancelAll\(\)](#)清空所有通知。

创建通知-Creating a Notification

[Notification](#)对象定义了通知的细节信息和其他提醒的设置，例如声音和闪灯。

一个状态通知需要以下：

- 状态栏的图标
- 标题和信息，除非你定义一个[custom notification layout](#)
- [PendingIntent](#),当通知被点击选择时将会被触发。

可选的设定包括：

- 标题栏的滚动文本
- 提醒的声音
- 振动的设定

- 闪屏的设定

新通知的starter-kit包括[Notification\(int, CharSequence, long\)](#)构造函数和[setLatestEventInfo\(Context, CharSequence, CharSequence, PendingIntent\)](#)方法。这些为通知定义了所有的需求。下面是一个基本的通知：

```

int icon = R.drawable.notification_icon;           // icon from
resources
CharSequence tickerText = "Hello";                // ticker-text
long when = System.currentTimeMillis();            // notification time
Context context = getApplicationContext();          // application
Context
CharSequence contentTitle = "My notification";    // message title
CharSequence contentText = "Hello World!";         // message text

Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);

// the next two lines initialize the Notification, using the
configurations above
Notification notification = new Notification(icon, tickerText,
when);
notification.setLatestEventInfo(context, contentTitle, contentText,
contentIntent)

```

更新通知-Updating the notification

当事件继续出现在你的应用中的时候，你就可以更新状态通知的信息。例如，当上一个条短信被阅读之前又来了一条新短信，这时短信应用会更新已存在的通知，给未读信息条数加一。这个更新通知的实践要比添加一个新通知好得多，因为它避免了在通知窗口里遇到的混乱。

因为每个通知都是由[NotificationManager](#)定义的，有着唯一的ID，你可以通过调用[setLatestEventInfo\(\)](#)来修改通知，然后再调用一次[notify\(\)](#).

你可以修改对象的每个属性（除了[Context](#)和通知的标题和内容）。你应该随时修改文本信息当调用[setLatestEventInfo\(\)](#)并且[contentTitle](#)和[contentText](#)都有新值的时候，然后调用[notify\(\)](#)(当然，如

果你已经创建了[custom notification layout](#),那么更新标贴和文本是不起作用的)

添加声音-Adding a sound

你可以使用默认的声音文件或者自定义的声音文件来提醒用户。

为了使用默认声音，要给defaults属性赋值"DEFAULT_SOUND":

```
notification.defaults |= Notification.DEFAULT_SOUND;
```

为了给你的通知使用不同的声音，需要把声音文件的Uri传递给sound属性。如下：

```
notification.sound =
Uri.parse("file:///sdcard/notification/ringer.mp3");
```

在下面的例子里，我们从内部的[MediaStore's ContentProvider](#):

```
notification.sound =
Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

在上面的例子中，数字6是媒体文件的ID，而且被加在了[Uri](#)的后面。如果你不知道确切的ID，你必须使用[ContentResolver](#)在[MediaStore](#)要查询一下。你可以查看[Content Providers](#)文档，来了解如何使用ContentResolver。

如果你希望提示音可以反复的播放，直到用户对通知做出了反应或者通知被取消了，可以把[FLAG_INSISTENT](#)赋值给flags属性。

注意：如果defaults属性的值是[DEFAULT_SOUND](#),那么无论设置什么声音都不会有效果的，仍只会播放默认的声音。

添加振动-Adding vibration

你可以用默认的振动方式或自定义的振动方式来提示用户。

默认的方式，要用到[DEFAULT_VIBRATE](#)

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
```

自定义的方式，要是定义一个long型数组，赋值给vibrate属性：

```
long[] vibrate = {0, 100, 200, 300};  
notification.vibrate = vibrate;
```

long型的数组定义了交替振动的方式和振动的时间（毫秒）。第一个值是指振动前的准备（间歇）时间，第二个值是第一次振动的时间，第三个值又是间歇的时间，以此类推。振动的方式任你设定。但是不能够反复不停。

注意：如果defaults属性的值是[DEFAULT_VIBRATE](#)，那么无论设置什么振动都不会有效果的，仍只会以默认的方式振动。

添加闪灯-Adding flashing lights

使用闪灯来提示用户，你可以使用默认的也可以自定义

默认的方式，[DEFAULT_LIGHTS](#)

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
```

可以自定义灯光的颜色和闪动的方式。`ledARGB`属性是定义颜色的，`ledOffMS`属性是定义灯光关闭的时间（毫秒），`ledOnMs`是灯光打开的时间（毫秒），还要给`flags`属性赋值为[FLAG_SHOW_LIGHTS](#)

```
notification.ledARGB = 0xff00ff00;  
notification.ledOnMS = 300;  
notification.ledOffMS = 1000;  
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

上面的例子中，绿色的灯亮了300毫秒，暗了1秒，，，，如此循环。

更多特性-More features

你可以使用[Notification](#)和标志（flags）来为自己的通知做定制。下面是一些有用的特性：

[FLAG_AUTO_CANCEL](#)标志

使用这个标志可以让在通知被选择之后，通知提示会自动的消失。

[FLAG_INSISTENT](#)标志

使用这个标志，可以让提示音循环播放，知道用户响应。

[FLAG_ONGOING_EVENT](#)标志

使用这个标记，可以让该通知成为正在运行的应用的通知。这说明应用还在运行-它的进程还跑在后台，即使是当应用在前台不可见（就像音乐播放和电话通话）。

[FLAG_NO_CLEAR](#)标志

使用这个标志，说明通知必须被清除，通过"Clear notifications"按钮。如果你的应用还在运行，那么这个就非常有用了。

[number](#)属性

这个属性的值，指出了当前通知的数量。这个数字是显示在状态通知的图标上的。如果你想使用这个属性，那么当第一个通知被创建的时候，它的值要从1开始。而不是零。

[iconLevel](#)

这个属性的值，指出了[LevelListDrawable](#)的当前水平。你可以通过改变它的值与LevelListDrawable定义的drawable相关联，从而实现通知图标在状态

栏上的动画。查看[LevelListDrawable](#)可以获得更多信息。

查看[Notification](#)可以了解更多。

自定义通知的布局-Creating a Custom Notification Layout

默认的，通知窗口里的通知会包括标题和消息文本两部分。它们是通过[setLatestEventInfo\(\)](#)定义了contentTitle和contentText来实现的。然而，你也可以使用[RemoteViews](#)为通知界面定义一个布局。它看起来与默认的布局很像，但实际上是由XML创建的。



图3 自定义布局

为了自定义通知的布局，首先实例化[RemoteViews](#)创建一个布局文件，然后把[RemoteViews](#)传递给[Notification](#)的contentView属性。

通过下面的例子可以更好的理解：

1. 创建一个XML布局文件 custom_notification.xml:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp" >
    <ImageView android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true"
        android:layout_marginRight="10dp" />
    <TextView android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/image"
        style="@style/NotificationTitle" />
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/image"
        android:layout_below="@+id/title"
```

```
    style="@style/NotificationText" />
</RelativeLayout>
```

注意那两个[TextView](#)的style属性。在定制的通知界面中，为文本使用style文件进行定义是很重要的，因为通知界面的背景色会因为不同的硬件，不同的os版本而改变。从android2.3(API 9)开始，系统为默认的通知界面定义了文本的style属性。因此，你应该使用style属性，以便于在android2.3或更高的版本上可以清晰地显示你的文本，而不被背景色干扰。

例如，在低于android2.3的版本中使用标准文本颜色，应该使用如下的文件res/values/styles.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="NotificationText">
        <item name="android:textColor">?
    android:attr/textColorPrimary</item>
    </style>
    <style name="NotificationTitle">
        <item name="android:textColor">?
    android:attr/textColorPrimary</item>
        <item name="android:textStyle">bold</item>
    </style>
    <!-- If you want a slightly different color for some text,
        consider using ?android:attr/textColorSecondary -->
</resources>
```

然后，在高于android2.3的系统中使用系统默认的颜色，如下文件res/values-v9/styles.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="NotificationText"
parent="android:TextAppearance.StatusBar.EventContent" />
    <style name="NotificationTitle"
parent="android:TextAppearance.StatusBar.EventContent.Title" />
</resources>
```

现在，当运行在2.3版本以上时，在你的定制界面中，文本都会是同一种颜色-系统为默认通知界面定义的颜色。这很重要，能保证你的文本颜色是高亮的，即使背景色是意料之外的颜色，你的文本页也会作出适当的改变。

2. 现在，在应用的代码中，使用RemoveViews方法定义了图片和文本。然

后把RemoveViews对象传给contentView属性。例子如下：

```
RemoteViews contentView = new RemoteViews(getApplicationContext(),
R.layout.custom_notification_layout);
contentView.setImageResource(R.id.image,
R.drawable.notification_image);
contentView.setTextViewText(R.id.title, "Custom notification");
contentView.setTextViewText(R.id.text, "This is a custom layout");
notification.contentView = contentView;
```

如上所示，把应用的包名和布局文件的ID传给RemoteViews的构造器。然后分别使用[setImageviewResource\(\)](#)和[setTextViewText\(\)](#)定义ImageView和TextView的内容。最后，吧RemoteViews对象传递给通知的contentView属性。

3. 因为当你在使用定制界面时，不需要使用[setLatestEventInfo\(\)](#)，你必须为通知定义一个intent，并赋值给contentIntent属性，如下代码：

```
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);
notification.contentIntent = contentIntent;
```

4. 发送通知：

```
mNotificationManager.notify(CUSTOM_VIEW_ID, notification);
```

[RemoteViews](#)类还包括一些方法可以让你轻松地在通知界面的布局里添加[Chronometer](#)和[ProgressBar](#)。如果要为你的通知界面做更多的定制，请参考[RemoteViews](#)。

特别注意：当创建一个自定义通知布局，你必须特别小心，以确保布局能适当地在不同的设备上显示。而这个建议适用于所有视图布局创建，不单单是通知界面布局，在这种情况下（通知界面）尤其重要，因为布局的空间是非常有限的。所以不要让你的自定义布局太复杂并确保在不同的配置上测试它。

来自“[index.php?title=Status_Notifications&oldid=8809](#)”

Search

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

Search Overview

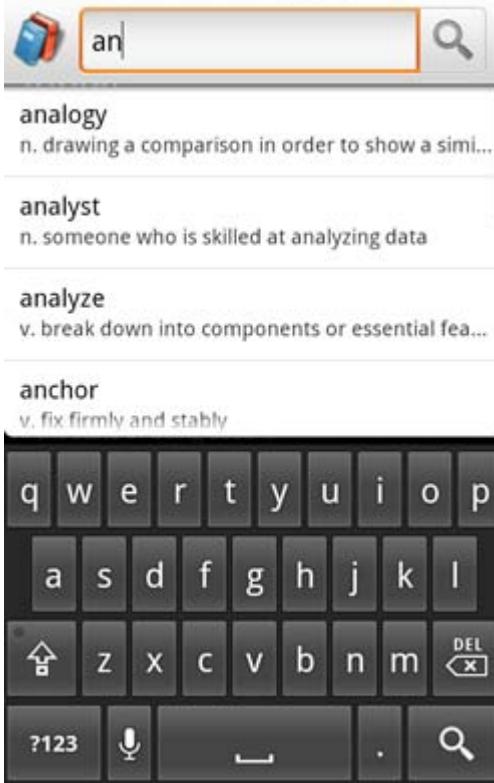
原文地址：<http://docs.eoeandroid.com/guide/topics/search/index.html>

翻译：张译成

更新：2012/9/14

搜索概述

在Android平台上搜索是一个核心的用户特色。用户应该能够搜索任何对它们可用的数据，无论内容位于设备或网络上。为了创建一个一致的用户搜索体验，Android平台提供了一个搜索框架帮助你的应用程序实现搜索功能。搜索框架提供了两种模式的搜索输入：一个在屏幕的顶部搜索对话框或搜索小部件(SearchView)，您可以将其嵌入到你的activity布局。在这两种情况下，Android系统将通过传递一个查询到特定的activity协助你的搜索实现。你还可以启用或搜索对话框或小部件来提供搜索建议用户类型。下图显示了一个示例搜索对话框和可选的搜索建议。



一旦以设置好了搜索对话或者搜索小部件，你就可以：

- 可以使用声音搜索
- 通过最近的搜索词提供搜索建议
- 提供自定义搜索建议,以匹配您的应用程序数据的实际结果
- 在系统范围的快速搜索框中提供你应用程序的搜索建议

注意：搜索框架不为你的数据提供API。搜索的话，你需要适合你的数据的API。例如，如果你的数据在SQLite数据库中，你应该使用[android.database.sqlite](#)达到搜索的目的。同时,也不能保证每个设备提供了一个专门的搜索按钮在您的应用程序中来调用搜索界面。当使用搜索对话框或一个自定义的接口,您必须提供一个搜索按钮在你的UI,激活了搜索界面。更多信息请看[Invoking the search dialog](#)。

下面的文档帮助你怎么样使用android平台的框架实现搜索功能：

Creating a Search Interface

怎么样设置你的应用程序使用搜索对话框或者小部件。

Adding Recent Query Suggestions

怎么样通过先前的查询提供搜索建议。

Adding Custom Suggestions

怎么样基于你应用程序的数据和提供搜索建议并且提供给系统的快速搜索框。

Searchable Configuration

搜索配置文件的参考文档(管其他文件也讨论配置文件的特殊行为)。

保护用户隐私

当你在应用程序中实现搜索功能,采取措施来保护用户的隐私。许多用户认为他们在电话上的行为-包括搜索是私人信息。为了保护每个用户的隐私,你应该遵守以下原则:

- 不要不私人信息传送到服务器, 如果必须的话, 不要保存下来

个人信息是任何可以识别你的用户的信息,比如他们的姓名、电子邮件地址、账单信息,或其他可以合理地链接到这些信息的数据。如果您的应用程序在一个服务器的协助下实现搜索,避免发送个人信息连同搜索查询。例如,如果你正在寻找企业近一个邮政编码,你不需要发送用户ID;只发送zip代码到服务器。如果你必须把个人信息,你应该不需要进行日志记录。如果你必须记录它,保护这些数据非常谨慎,尽快将它抹去。

- 向用户提供一个可以清楚历史搜索的途径

当用户输入的时候搜索框架帮助您的应用程序提供特定于上下文的建议。有时这些建议都是基于之前的搜索或用户在前面的采取的其它行动。用户可能不希望对先前的搜索被泄露给其他设备用户,例如,如果用户与朋友共享设备。如果您的应用程序提供了建议会之前的搜索活

动,你应该实现允许用户清楚搜索 历史的办法。如果你正在使用[SearchRecentSuggestions](#)你可以简单的调用[clearHistory\(\)](#)方法。如果您正在实现定制的建议,你需要在你的content provider提供一个类似的“清除历史”方法,使用户可以执行。

来自“[index.php?title=Search&oldid=11634](#)”



Creating a Search Interface

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/guide/topics/search/search-dialog.html>

翻译： --[Snowxwyo](#) 2012年7月10日 (二) 08:20 (CST)

[搜索-Search:](#)

目录

- [1 创建一个搜索接口-Creating a Search Interface](#)
 - [1.1 基础-The Basics](#)
 - [1.2 创建一个可搜索的配置-Creating a Searchable Configuration](#)
 - [1.3 创建一个可搜索活动-Creating a Searchable Activity](#)
 - [1.3.1 声明一个可搜索活动-Declaring a searchable activity](#)
 - [1.3.2 执行一个搜索-Performing a search](#)
 - [1.3.2.1 接收请求-Receiving the query](#)
 - [1.3.2.2 搜索数据-Searching your data](#)
 - [1.3.2.3 呈现结果-Presenting the results](#)
 - [1.4 使用搜索对话框-Using the Search Dialog](#)
 - [1.4.1 调用搜索对话框Invoking the search dialog](#)
 - [1.4.2 搜索对话框对活动生命周期的影响-The impact of the search dialog on your activity lifecycle](#)
 - [1.4.3 传递搜索上下文数据-Passing search context data](#)
 - [1.5 使用搜索小插件-Using the Search Widget](#)
 - [1.5.1 配置搜索小插件-Configuring the search widget](#)
 - [1.5.2 搜索小插件的其它功能-Other search widget features](#)
 - [1.5.3 同时使用小插件和对话框-Using both the widget and the dialog](#)
 - [1.6 添加语音搜索-Adding Voice Search](#)
 - [1.7 添加搜索建议-Adding Search Suggestions](#)

创建一个搜索接口-Creating a Search

Interface

当你准备为你的应用程序添加搜索功能时，Android会帮你助实现用户接口，这个接口是一个搜索对话框，显示在活动窗口的顶端，或是一个你可以插入到你的布局之中的搜索小插件。搜索对话框和小插件都可以把的搜索请求传递给你的应用程序中一个指定的活动。通过这种方法，用户可以任何一个搜索对话框或小插件可用的活动中发起一个搜索，并且系统将会启动一个适应的活动来实现这个搜索并呈现其结果。

其它搜索对话框和小插件的可用功能有：

- 语音搜索
- 基于最近查询的搜索建议
- 在你的应用程序数据中，初建结果相匹配的搜索建议

这个向导将向你展现怎么样设置你的应用程序来提供一个搜索接口，由Android系统帮助传递搜索请求，使用搜索对话框或搜索插件。

基础-The Basics

在你开始之前，你应该先决定你是要用搜索对话框或是搜索小插件来实现你的搜索接口。这两个方法提供了相同的搜索功能，但是存在一点小区别：

- 搜索对话框是一个由Android系统控制

快速浏览

- Android系统从搜索对话框或小插件向你指点的用来执行搜索和呈现结果的活动发送搜索请求
- 为了快速访问，你可以把搜索小插件放在动作栏中，作为一个“动作视图”

本文内容：

[基础-The Basics](#)

[创建一个可搜索的配置-Creating a Searchable Configuration](#)
[创建一个可搜索的活动-Creating a Searchable Activity](#)

[声明一个可搜索的活动-Declaring a searchable activity](#)

[使用搜索对话框-Using the Search Dialog](#)

[调用搜索对话框Invoking the search dialog](#)

[搜索对话框对活动生命周期的影响-The impact of the search dialog on your activity lifecycle](#)

[传递搜索上下文数据-Passing search context data](#)

[使用搜索小插件-Using the Search Widget](#)

[配置搜索小插件-Configuring](#)

的UI部件。当被用户激活时，搜索对话框出现在活动的顶部，如图1所示。

Android系统控制了搜索对话框中的所用事件。当用户提交一个请求时，系统将请求传递给你指定的用来处理搜索的活动。当用户输入时，对话框也可以提供搜索建议。

- 搜索小插件是[SearchView](#)的实例话应用，你可以放在布局中的任意位置。默认的，搜索小插件表现的如同一个标准的[EditText](#)小插件，并不做其它任何事情，不过你可以更改其配置，让Android系统所有的输入事件，传递请求给适当的活动，并且提供搜索建议（与搜索对话框相同）。然而，搜索小插件只在Android 3.0 (API Level 11) 以及更高版本中可用。
- 注解：如果你希望，你可以自己处理所有的用户输入，使用不同的回调方法和监听器。然而，这个文档专注于如何整合搜索小插件与系统，达到辅助搜索的实现。如果你希望自己处理所有的用户输入，请阅读[SearchView](#)参考文档以及其嵌套接口。

[the search widget](#)

[搜索小插件的其它功能-Other search widget features](#)

[同时使用小插件和对话框-](#)

[Using both the widget and the dialog](#)

[添加语音搜索-Adding Voice Search](#)

[添加搜索建议-Adding Search Suggestions](#)

关键类：

[SearchManager](#)

[SearchView](#)

下载：

[search_icons.zip](#)

其他资源：

[添加最近请求建议-Adding Recent Query Suggestions](#)

[添加定制建议-Adding Custom Suggestions](#)

[可搜索的配置-Searchable Configuration](#)

当用户从搜索对话框或一个搜索小插件执行

一个搜索时，系统创建了一个[Intent](#)对象，并保存了用户的请求。然后，系统启动了你声明的用来处理搜索的活动（“可搜索的活动<searchable activity>”），并把intent对象传递给它。给你的应用程序设置这种辅助搜索方法，以下是你需要做的：

- 一个可搜索的配置
一个XML文件，为搜索对话框或小插件配置了一些设定。它包括了一些功能设定，如语音搜索，搜索建议，以及搜索框提示文本。
- 一个可搜索的活动

一个[活动-Activity](#)，用来接收搜索请示，搜索你的数据并显示搜索结果。

图1 应用程序的搜索对话框截图

- 一个搜索接口，提供了以下两者之一：
- ◦ 搜索对话框
默认的，搜索对话框是被隐藏的，但当用户按了设备的搜索按键（当可用时）或是其他用户接口中的按键时，它会出现在屏幕的顶部。
- ◦ 或是[SearchView](#)小插件
使用搜索小插件允许你把搜索框放置于活动的任意位置。而不用放在活动的布局之中，然而，一般情况下使用一个[操作栏-Action Bar](#)中的操作视图将更加便于用户使用。

本文的其余部分将向你展示怎么创建可搜索的配置，可搜索的活动，以及使用搜索对话框或搜索小插件实现一个搜索接口。

创建一个可搜索的配置-Creating a Searchable Configuration

首先，你需要一个叫做可搜索的配置的XML文件。它配置了特定UI方面的搜索对话框或小插件，并且定义了例如建议和语音搜索等功能是怎么样实现的。这个文件传统中被命名为[searchable.xml](#)，并且必须保存在[res/xml/](#)路径下。

- 注解：系统使用这个文件来创建一个[SearchableInfo](#)对象的实例，但是，你并不能在运行时自己创建这个对象－而应该是在XML中申明这个可搜索的配置。

这个可搜索的配置文件必须包含[`<searchable>`](#)这个元素作为其根结点，并指明一个或多个属性。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint" >
</searchable>
```

android:label是唯一一个必要的属性。它指向了一个字符串资源，这个字符串应该是应用程序的名字。这个标签只有在你为快速搜索框启用了搜索建议时才会对用户可见。这时，这个标签将出现在系统设置中的可搜索项目列表中。

虽然不要求，但我们建议你总是设置一个**android:hint**属性，用户输入请求之前，在搜索框中提供一个可提示性的字符串。这个提示很重要，因为它为用户搜索提供了重要的线索。

- 小贴士：为了和其它Android应用程序保持一致，你应该为你的**android:hint**字符串使用如下格式："Search <content-or-product>"。如，"Search songs and artists"或"Search YouTube"。

<searchable>元素可接受几种属性。然而，你并不需要其中大多数的属性，直到你添加一些特性，如[搜索建议-Search Suggestions](#)和[语音搜索-Voice Search](#).更多关于可搜索的配置文件信息，请见[可搜索配置-Searchable Configuration](#)引用文档。

创建一个可搜索活动-Creating a Searchable Activity

一个可搜索的活动是你应用程序中的一个[Activity](#), 它执行一个基于请求字符串的搜索，并呈现搜索结果。

当用户在搜索对话框或小插件中执行一个搜索时，系统将启动你的可搜索活动，同时把搜索请求通过intent对象的[ACTION_SEARCH](#)动作传送给它。你的活动通过intent对象的[QUERY](#)获得请求，然后搜索你的数据并呈现结果。

因为你在应用程序中的任何一个活动中包含搜索对话框或小插件，所以系统必须知道哪一个活动是你的可搜索活动，这样他才能够准备的传递搜索请求。所以，你必须首先在Android的manifest文件中申明你的可搜索活动。

声明一个可搜索活动-Declaring a searchable activity

如果你还没有，那么先创建一个[活动-Activity](#)来执行搜索并呈现结果。你目前还不需要实现搜索功能—只是创建一个你可以用来在manifest中申明的活动。放在manifest的**<activity>**元素中：

1. 在[<intent-filter>](#)元素中申明这个活动用来接收[ACTION_SEARCH](#) intent对象。
2. 在[<meta-data>](#)元素中指定一个用来使用的可搜索配置。

例如：

```
<application ... >
    <activity android:name=".SearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
            android:resource="@xml/searchable" />
    </activity>
    ...
</application>
```

[<meta-data>](#)元素必须包括[android:name](#)属性，并赋值"[android.app.searchable](#)"，和[android:resource](#)属性，使用一个可搜索配置文件的引用（在这个例子中，指的是[res/xml/searchable.xml](#)文件）。

- 注解：[<intent-filter>](#)并不需要一个使用[DEFAULT](#)值的[<category>](#)元素（常见于[<activity>](#)元素之中），因为系统使用了其组件名，能把[ACTION_SEARCH](#) intent对象准确的传递给你的可搜索活动。

执行一个搜索-Performing a search

当你在manifest中申明了你的可搜索活动后，在可搜索活动中执行一个搜索包括以下三个步骤：

1. [接收请求-Receiving the query](#)
2. [搜索数据-Searching your data](#)
3. [呈现结果-Presenting the results](#)

一般来说，你的搜索结果应该呈现在[ListView](#)控件中，所以，你可能会希望你的可搜索活动继承[ListActivity](#)。它包含了一个默认的布局文件，其中有一个单独的[ListView](#)控件，并提供了几个方便的方法来使用[ListView](#)控件。

接收请求-Receiving the query

当用户从搜索对话框或小插件执行一个搜索时，系统将启动你的可搜索活动，并把[ACTION_SEARCH](#) intent对象传递给它。这个intent对象携带了[QUERY](#) extra字符串中的搜索请求。当活动启动时，你必须检查这个intent对象，并获得这个字符串。以下例子为当你的可搜索活动启动时将怎样获得搜索请求

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.search);

    // Get the intent, verify the action and get the query
    Intent intent = getIntent();
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        doMySearch(query);
    }
}
```

[QUERY](#)字符串始终包含了[ACTION_SEARCH](#) intent对象。在这个例子中，请求被取得，并传递给一个本地方法[doMySearch\(\)](#)，在这个方法中实现了实际的搜索操作。

搜索数据-Searching your data

对于你的应用程序，存储和搜索数据的过程是唯一的。你可以通过多种方法来实现储存和搜索数据，但这个向导并没有教你怎么样存储并搜索数据。你需要根据你的需求和数据格式认真考虑怎么样存储和搜索数据。然而，以下是一些你可以会用到的小提示：

- 如果你的数据储存在设备的SQLite数据库中，呈现一个全文本搜索（使用[FTS3](#)，而不是[LIKE](#)请求），通过文本数据，能够提供一个更加稳健的搜索，并且可以显著的提高搜索时间。关于[FTS3](#)的信息请见[sqlite.org](#)，关于Android中SQLite的信息请见[SQLiteDatabase](#)类。同样可见[Searchable Dictionary](#)范例应用程序，其完整的展示了SQLite如何通过[FTS3](#)实现搜索。
- 如果你的数据存储在网络上，那么搜索的表现将会由用户的数据连接所控制。如果你想在搜索结果返回之前显示一个旋转的进度条，请见[android.net](#)中网络APIs的引用和[Creating a Progress Dialog](#)中关于怎样显示一个进度条的相关信息。

忽略数据的存储位置和搜索方式，我们建议你使用[Adapter](#)把搜索结果返回给可搜索活动。通过这种方法，你可以非常简单的把所有的搜索结果呈现在一个[ListView](#)。如果你的

关于Adapters (适配器)

数据来自于SQLite数据库请求，你可以使用[CursorAdapter](#)把结果应用于[ListView](#)。如果你的数据来自于其它格式类型，那么你可以创建一个扩展的[BaseAdapter](#)。

呈现结果-Presenting the results

如前所述，推荐的用来展示搜索结果的UI为[ListView](#)，所以你可能会想要你的可搜索活动继承[ListActivity](#)。然后你可以调用一个[setListAdapter\(\)](#)方法，把已绑定了数据的[Adapter](#)传递给它。

更多关于在列表中呈现结果的帮助，请见[ListActivity](#)文档。

也可见[Searchable Dictionary](#)范例，其完整的呈现了怎样创建一个SQLite数据库并使用[Adapter](#)为[ListView](#)提供结果。

使用搜索对话框-Using the Search Dialog

搜索对话框在屏幕上端提供了一个浮动的搜索框，并在其左侧使用了应用程序的图标。用户输入时，搜索对话框可以提供搜索建议，当用户执行一个搜索时，系统把搜索请求传给一个可搜索活动，并执行搜索。然而，如果你在为Android 3.0设备开发应用程序，你应该考虑使用搜索小插件来代替搜索对话框（见侧边栏）。

默认的，搜索对话框始终保持隐藏，直到用户激活它。如果用户的设备有一个搜索按钮，点击这个按钮将会默认的激活搜索对话框。你的应用程序也可根据要求，通过调用[onSearchRequested\(\)](#)

一个[Adapter](#)把一组数据中的每一个项目绑定成了一个[View](#)对象。当一个[Adapter](#)应用于一个[ListView](#)，每一片数据都被当作是一个单独的view被插入列表中。[Adapter](#)只是一个接口，所以如[CursorAdapter](#)（从一个[Cursor](#)绑定数据）等实现是必需的。如果现有的实现没有能为你的数据服务的，那么你可以从[BaseAdapter](#)创建一个你自己的实现。安装所有的API level 4 SDK样例包，可以查阅所有原始版本的可搜索字典，其创建了一个可定制的adapter，从一个文件中读取数据。

我应该使用搜索对话框还是小插件？

答案主要基于你是否为Android 3.0 (API level 11或更高版本) 设备做开发，因为[SearchView](#)小插件是在Android 3.0中加入的。所以，如果你的应用程序是为低于3.0版本的Android设备所开发的，搜索小插件将不做为一个备选项，你应该使用搜索对话框来实现你的搜索接口。

用 方法来激活搜索对话框。但是，这些在你没有为活动启动搜索对话框之前都是无效的。

启动搜索对话框，你必需向系统指出哪个可搜索活动应该从搜索对话框接收搜索请求，来执行搜索。例如，在前一个关于[创建一个可搜索活动-Creating a Searchable Activity](#)的章节中，一个名叫SearchableActivity被创建。如果你想要另一个分开的活动，取名OtherActivity，来显示搜索对话框并把搜索传递给SearchableActivity。你必需要在manifest中申明SearchableActivity为OtherActivity中搜索对话框所使用的可搜索活动。

如果你在为Android 3.0或更高版本做开发，那么，其将由你的需求所决定。多数情况下，我们建议你使用搜索小插件作为Action Bar（动作条）中的“Action View（动作视图）”。所以，你可能会想把搜索小插件放在活动而已中的某个位置。并且，如果其它都失败了，你还可以使用搜索对话框如果你倾向于把搜索框隐藏。事实上，在某些情况下，你应该会想要同时提供对话框和小插件。更多关于小插件的情况，请移步[Using the Search Widget](#)。

为一个活动的搜索对话框申明一个可搜索活动，在各自的活动元素[<activity>]中添加一个[<meta-data>]元素。这个[<meta-data>]必需包括android:value属性来指定可搜索活动类的名字和android:name属性，并使用"android.app.default_searchable"作为其值。

例如，以下为同时为可搜索活动，SearchableActivity，和另一个活动，OtherActivity，其通过搜索对话框使用了SearchableActivity来执行搜索：

```
<application ... >
    <!-- this is the searchable activity; it performs searches -->
    <activity android:name=".SearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
                  android:resource="@xml/searchable" />
    </activity>

    <!-- this activity enables the search dialog to initiate searches
        in the SearchableActivity -->
    <activity android:name=".OtherActivity" ... >
        <!-- enable the search dialog to send searches to SearchableActivity --
    ->
        <meta-data android:name="android.app.default_searchable"
                  android:value=".SearchableActivity" />
    </activity>
    ...
</application>
```

因为OtherActivity现在包含了一个[<meta-data>](#)元素，用来申明哪一个可搜索活动用来

执行搜索，这个活动已经启用了搜索对话框。当用户使用这个活动时，设备的搜索按钮（如果可用）和[onSearchRequested\(\)](#)方法将激活搜索对话框。当用户执行搜索时，系统启动[SearchableActivity](#)并把[ACTION_SEARCH](#) intent对象。

- 注解：可搜索活动默认自带了搜索对话框，并不需要把这个申明添加到[SearchableActivity](#)中。

如果你想给应用程序中的每一个活动都提供搜索对话框，把以上提到的[`<meta-data>`](#)元素作为[`<application>`](#)元素的子元素插入，而不用在每一个[`<activity>`](#)中都添加一遍。通过这个方法，每一个活动都继承了这个值，提供搜索对话框，并把搜索传递给同一个可搜索活动。（如果你有多个可搜索活动，你可以复写默认的可搜索活动，在每一个活动中放置一个不同的[`<meta-data>`](#)申明）。

现在，搜索对话框已对你的活动启用，你的应用程序已经准备好可以执行搜索了。

调用搜索对话框 **Invoking the search dialog**

在前文中已经提到，设备的搜索按钮可以打开搜索对话框，只要当前的活动在[manifest](#)中申明了可用的搜索活动。

然而，一些设备并没有一个特定的搜索按钮，所以，你并不应该假设搜索按钮激活对话框始终可行。当使用搜索对话框时，你应该始终为你的[UI](#)提供另一个搜索按钮，可以调用[onSearchRequested\(\)](#)方法来激活搜索对话框。

例如，你要么在[Options Menu](#)中提供一个菜单项，要么在活动的布局中添加一个按钮来调用[onSearchRequested\(\)](#)方法，并激活搜索。[search_icons.zip](#)文件包含了为中等和高密度屏幕准备的图标，你可以用作搜索菜单项或按钮的图标（低密度屏幕把hdpi图压缩一半）。

你也可启用"type-to-search (键入搜索)"功能，这个功能在用户开始敲击键盘时激活搜索对话框。你可以在活动的[onCreate\(\)](#)方法中通过调用[setDefaultKeyMode\(DEFAULT_KEYS_SEARCH_LOCAL\)](#)来启动"type-to-search (键入搜索)"功能。

搜索对话框对活动生命周期的影响-**The impact of the search dialog on**

your activity lifecycle

搜索对话框是一个漂浮于屏幕顶端的[对话框-Dialog](#)。它并不影响活动栈，所以当搜索对话框出现时，没有任何生命周期方法（如[onPause\(\)](#)）被调用。活动只是失去了输入焦点，因为输入焦点被交给了搜索对话框。

如果你想在搜索对话框被激活时收到通知，复写[onSearchRequested\(\)](#)方法。当系统调用这个方法时，将出现提示，你的活动失去了输入焦点，并交给了搜索对话框，所以你可以用来对事件做适当的处理工作（如暂停一个游戏）。除非你[传递搜索前景数据](#)（将在下文中讨论），你应当通过调用其父类实现来结束这个方法。如：

```
@Override
public boolean onSearchRequested() {
    pauseSomeStuff();
    return super.onSearchRequested();
}
```

如果用户通过返回按键取消了搜索，搜索对话框被关闭，活动重新获得输入焦点。当搜索对话框关闭时，你可以使用[setOnDismissListener\(\)](#)和/或[setOnCancelListener\(\)](#)方法来注册一个通知。你应该只需要注册[OnDismissListener](#)，因为它在每次搜索对话框关闭时都会被调用。[OnCancelListener](#)只有在用户明确的退出搜索对话框时才附属于这个事件，所以在搜索执行时被不会被调用（在这种情况下，搜索对话框将自然消失）。

如果当前的活动并不是可搜索活动，那么，在用户执行一个搜索时正常的活动生命周期事件将被触发（当前活动接收[onPause\(\)](#)等方法，如[Activities](#)文档中所表述的）。然而，如果当前活动为可搜索活动，那么以下两者之一将发生：

- 默认的，可搜索活动接收[ACTION_SEARCH](#) intent对象，调用[onCreate\(\)](#)方法，并且一个新的活动实例被添加到活动栈的顶端。现在就有两个可搜索活动的实例在活动栈中（所以点击返回按钮将返回到前一个可搜索活动实例，而不是当前存大的可搜索活动）。
- 如果你设定[android:launchMode](#)为singleTop，那么可搜索活动接收[ACTION_SEARCH](#) intent对象，调用[onNewIntent\(Intent\)](#)活动，传递新的[ACTION_SEARCH](#) intent对象。例如，以下为，在可搜索活动启动模式为"singleTop"时，你应该怎么样处理这种情况：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.search);
    handleIntent(getIntent());
}
```

```

@Override
protected void onNewIntent( Intent intent ) {
    setIntent( intent );
    handleIntent( intent );
}

private void handleIntent( Intent intent ) {
    if ( Intent.ACTION_SEARCH.equals( intent.getAction() ) ) {
        String query = intent.getStringExtra( SearchManager.QUERY );
        doMySearch( query );
    }
}

```

与这个章节中关于[Performing a Search](#)范例中的代码对比，所有处理搜索intent对象的代码现在都包括在了[handleIntent\(\)](#)方法中，所以，[onCreate\(\)](#)和[onNewIntent\(\)](#)都可以执行这个方法。

当系统调用[onNewIntent\(Intent\)](#)方法时，活动被没有被重新启动，所以[getIntent\(\)](#)方法返回了与[onCreate\(\)](#)方法接收到的相同的intent对象。这也是为什么你应该在[onNewIntent\(Intent\)](#)中调用[setIntent\(Intent\)](#)方法（这样，活动保存的intent对象被更新了，以防将来你调用[getIntent\(\)](#)）。

第二个场景使用"singleTop"启动模式是一般作法，因为，一旦搜索完成，其可为用户提供一个很好的来执行附加搜索，并且，如果你的应用程序创建了多个可搜索活动实例将是一个不好的体验。所以，我们建议我们建议你在应用程序的manifest中设置可搜索活动为"singleTop"启动模式。如：

```

<activity android:name=".SearchableActivity"
          android:launchMode="singleTop" >
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    <meta-data android:name="android.app.searchable"
              android:resource="@xml/searchable" />
</activity>

```

传递搜索上下文数据-Passing search context data

在一些情况下，你可以为每一个搜索，在可搜索活动中为搜索请求添加必要的精练提纯。然则，如果你想要基于用户执行搜索的活动来精练搜索条件，你可以在系统传递给可搜索活动的intent对象中添加一些额外的数据。你可以在[APP_DATABundle](#)中传递额外的数据，其包含在[ACTION_SEARCH](#) intent对象中。

传递这种类型数据给可搜索活动，在用户可执行搜索的活动中复写[onSearchRequested\(\)](#)方法，使用额外数据创建一个[Bundle](#)对象，并调用[startSearch\(\)](#)方法来激活搜索对话框。例如：

```

@Override
public boolean onSearchRequested() {
    Bundle appData = new Bundle();
    appData.putBoolean(SearchableActivity.JARGON, true);
    startSearch(null, false, appData, false);
    return true;
}

```

返回“true”，表明你已经成功的处理了这个回调事件，并调用了[startSearch\(\)](#)方法激活搜索对话框。一旦用户确认了一个请求，它将伴随着你添加的数据一同传递给可搜索活动。你可以从[APP_DATA Bundle](#)中提取额外的数据，来精练搜索。如：

```

Bundle appData = getIntent().getBundleExtra(SearchManager.APP_DATA);
if (appData != null) {
    boolean jargon = appData.getBoolean(SearchableActivity.JARGON);
}

```

- 注意：永远不要在[onSearchRequested\(\)](#)回调方法外调用[startSearch\(\)](#)方法。为了激活搜索对话框，始终调用[onSearchRequested\(\)](#)方法。否则，[onSearchRequested\(\)](#)不被调用并且定制（如上例中添加的appData）将丢失。

使用搜索小插件-Using the Search Widget

[SearchView](#)小插件

在Android 3.0和更高版本中可用。如果你在为Android 3.0设备开发应用程序，并且决定使用搜索小插件，我们建议你把搜索小插件当作[动作条中的动作视图](#)插入，而不是使用搜索对话框（也不是把搜索小插件入在活动布局中）。例如，如图2展示了动作条中的搜索小搜索。



图2.在动作条中作为“动作视图”的[SearchView](#)小插件。

搜索小插件提供了与搜索对话框相同的功能。当用户执行一个搜索时，它将启动适当的活动，并提供了搜索建议和语音搜索。

- 注解：当你把搜索小插件当作一个动作视图使用时，你依然需要支持使用搜索对话框，以防搜索小插件不适合动作条。见后续关于[Using both the widget and the dialog](#)的章节。

配置搜索小插件-Configuring the search widget

如前所述，在创建了[可搜索配置](#)和[可搜索活动](#)之后，你需要为每一个[SearchView](#)启用辅助搜索。你可以通过调用[setSearchableInfo\(\)](#)并传递表示了可搜索配置的[SearchableInfo](#)对象来实现。

你可以通过[SearchManager](#)调用[getSearchableInfo\(\)](#)方法来获得一个[SearchableInfo](#)的引用。

例如，如果你使用一个[SearchView](#)作为[Action Bar](#)中的一个动作视图，你应该在[onCreateOptionsMenu\(\)](#)回调方法中启用小插件：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the options menu from XML
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager = (SearchManager)
        getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView = (SearchView)
        menu.findItem(R.id.menu_search).getActionView();

    searchView.setSearchableInfo(searchManager.getSearchableInfo(getApplicationContext()));
    searchView.setIconifiedByDefault(false); // Do not iconify the widget;
                                         // expand it by default

    return true;
}
```

这是所有你需要的。搜索小插件已配置，并且系统将把搜索请求传递给可搜索活动。你可以为搜索小插件启用[搜索建议](#)。

- 注解：如果你想自己处理所有的输入，你可以使用一些回调方法和事件监听器来实现。更多信息请见[SearchView](#)引用文档和其适当事件监听器的嵌套接口。

更多关于动作条中的动作视图的信息，请阅读[Action Bar](#)开发向导（其包含了添加一个搜索小插件作为动作视图的范例代码）。

搜索小插件的其它功能-Other search widget features

[SearchView](#)小插件允许了一些你可能想要的附加功能：

一个确认按钮

默认的，没有按钮来确认一个搜索请求，所以用户必须点击键盘上的“返回”键来发起一个搜索。你可以通过调用[setSubmitButtonEnabled\(true\)](#)来添加一个“确认”按钮。

精练请求搜索建议

当你启用搜索建议时，通常情况下，你期望用户简单的选择一个建议，但他们也许会想要精练建议的搜索请求。你可以通过调用[setQueryRefinementEnabled\(true\)](#)方法，在每一个建议边上添加一个按钮，其在搜索框中插入了为用户精练所用的建议。

开关搜索框可见性的能力

默认情况下，搜索小插件是“图标化的”，即它由一个搜索图标代表（一个放大镜），当用户点击时将放大来展示搜索框。如前所述，你可以通过默认方式显示搜索框，通过调用[setIconifiedByDefault\(false\)](#)。你也能通过调用[setIconified\(\)](#)来切换搜索小插件的外观。

[SearchView](#)类中还有一些其他的APIs,允许你定制搜索小插件。然而，大部分只适用于你自己处理所有用户输入，而不是使用Android系统来传递搜索请求和显示搜索建议。

同时使用小插件和对话框-Using both the widget and the dialog

如果你把搜索小插件当作[动作视图-action view](#)插入到动作条中，并且让其在动作条时显示“如果有空间”（通过设定`android:showAsAction="ifRoom"`），那么，有一定机会搜索小插件不会作为一个动作视图出现，但是，菜单选项将会出现在溢出菜单中。例如，当你的应用程序在一个更小的屏幕上运行，状态条中可能没有足够的空间来显示搜索小插件和其它的动作项目或导航元素，所以，菜单项将会代替其显示在溢出菜单中。当显示在溢出菜单时，这个项目就如同普通菜单项一样工作，并且不会显示动作视图（搜索小插件）。

出下这种情况，，当用户从溢出菜单选择你用来存储搜索小插件的菜单项应该激活搜索对话框。为了实现这个功能，你必须实现[onOptionsItemSelected\(\)](#)方法来处理搜索菜单项并且通过调用[onSearchRequested\(\)](#)方法来开启搜对话框。

更多关于动作条中项目怎么工作和怎样处理这种情形的信息，请见[Action Bar](#)开发向导。

同样可见[Searchable Dictionary](#)中一个同时使用了对话框的小插件的范例。

添加语音搜索-Adding Voice Search

你可以在搜索小插件听对话框中加入语音搜索功能，通过在搜索配置中添加`android:voiceSearchMode`属性。这个方法添加了一个语音搜索按钮，启动一个语音提示信息。当用户结束语音，转录搜索请求被传给了可搜索活动。

例：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer" >
</searchable>
```

`showVoiceSearchButton`值用来启用语音搜索，第二个值，`launchRecognizer`，指定了语音搜索按钮应该启动一个识别器，将转录的文本返回给搜索活动。

你可以提供其它的属性来指定语音搜索行为，如希望使用的语言和最大的结果返回数。更多关于可用属性的信息，请见[Searchable Configuration](#)。

- 注解：仔细考虑语音搜索是否适用于你的应用程序。所有由语音搜索按钮执行的搜索将立即传递给搜索活动，并不留机会给用户回顾转录的请求。有效的测试语音识别，并保证其正确理解用户在应用程序中确认的请求的类型。

添加搜索建议-Adding Search Suggestions

当用户输入时，在Android系统的帮助下，搜索对话框和小插件都可以搜索搜索建议。系统管理着搜索列表并且处理用户选择一个建议时的事件。



图3. 使用定制搜索建议的搜索对话框

你可以提供两种类型的搜索建议：

近期请求搜索建议

这些建议是用户近期在应用程序中使用的搜索请求。

见[Adding Recent Query Suggestions](#)。

定制搜索建议

这些搜索建议是你从你的数据资源中提供的，用来帮助用户快速选择正确的拼写或项目进行搜索。图3展示了一个为字典定制搜索建议的例子——用户可以选择一个建议，并立即进入其解释。

见[Adding Custom Suggestions](#)

[返回搜索](#)

来自“[index.php?title=Creating_a_Search_Interface&oldid=4719](#)”

1个分类:

- [Android Dev Guide](#)

Adding Recent Query Suggestions

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/search/adding-recent-query-suggestions.html>

翻译：[futurexiong](#)

更新：2012.07.02

目录

- [1 添加最近查询建议项](#)
 - [1.1 基础知识](#)
 - [1.2 创建一个内容提供者](#)
 - [1.3 修改搜索配置](#)
 - [1.4 保存查询](#)
 - [1.5 清除建议项数据](#)

添加最近查询建议项

当使用Android搜索对话框或者搜索小工具时，你可以提供基于最近搜索查询的搜索建议项。举个例子，如果一个用户之前搜索过 "puppies,"那么一旦他或她开始键入同一个查询的时候这个查询将作为一个建议项展示出来。图1展示了一个带有最近查询建议项的搜索对话框的例子。

本文内容

[基础知识](#)

[创建一个内容提供者](#)

在你开始这么做之前，你需要在你的应用中为基本的搜索实现搜索对话框或者一个搜索小工具。如果你还没有实现，请看[Creating a Search Interface](#)。

基础知识

最近搜索建议项只是被保存下来的搜索。当用户选择了其中一个建议项，你的搜索Activity会收到一个带有该Activity已经处理过的搜索查询的[ACTION_SEARCH](#)意图（如在[Creating a Search Interface](#)描述的那样）。

要提供搜索建议项，你需要：

- 实现一个搜索的Activity，如在[Creating a Search Interface](#)描述的那样。
- 创建一个继承自[SearchRecentSuggestionsProvider](#)的内容提供者并在你的应用清单中声明它。
- 修改搜索配置中有关提供搜索建议项的内容提供者的信息。
- 每次搜索执行的时候把查询内容保存到你的内容提供者中。

正如Android系统显示搜索对话框那样，它也同样在对话框或者搜索小工具下显示搜索建议项。你所需要做的是提供一个系统能从中取得建议项的源。

当系统识别到你的Activity是用于搜索的并且提供了搜索建议项，一旦用户开始键入查询的时候以下步骤就会发生：

- 系统取得搜索查询文本（不管当前键入了什么）并对包含你搜索建议项的内容提供者进行一次查询。
- 你的内容提供者返回一个[Cursor](#)，该Cursor指向所有与查询文本匹配的搜索建议项。
- 系统以列表形式展现该Cursor提供的建议项。

[修改搜索配置](#)

[保存查询](#)

[清除建议项数据](#)

关键类

[SearchRecentSuggestions](#)

[SearchRecentSuggestionsProvider](#)

参考

[Searchable Configuration](#)



图1。搜索对话框的截图，该对话框带有最近搜索建议项。

一旦最近搜索建议项展示出来，以下事情可能会发生：

1. 如果用户键入了另外一个关键字，或者以任何形式改变了查询内容，系统将会重复上述步骤并更新搜索建议项列表。
2. 如果用户执行了搜索，这些建议项将会被忽略，并且系统会使用一般的[ACTION_SEARCH](#)意图将该搜索传递到你的搜索Activity中。
3. 如果用户选择了其中一个建议项，一个[ACTION_SEARCH](#)意图将会被传递到你的搜索Activity中，使用被选中的建议文本作为查询。

你的内容提供者继承的[SearchRecentSuggestionsProvider](#)类自动地完成了上述的工作，所以实际上只有很少的代码需要你去编写。

创建一个内容提供者

为了最近搜索建议项你所需的那个内容提供者必须是[SearchRecentSuggestionsProvider](#)的一个实现。这个类几乎为了做了所有的事情。你所需要做的只是写一个执行一行代码的类构造器。举个例子，这是一个作为最近搜索建议项的内容提供者的一个完整的实现。

```
public class MySuggestionProvider extends
SearchRecentSuggestionsProvider {
    public final static String AUTHORITY =
"com.example.MySuggestionProvider";
    public final static int MODE = DATABASE_MODE_QUERIES;

    public MySuggestionProvider() {
        setupSuggestions(AUTHORITY, MODE);
    }
}
```

[setupSuggestions\(\)](#)方法的调用传递了搜索鉴权的名字和数据库模式。搜索鉴权可以是任何独特的字符串，但最好的做法是使用你内容提供者的全路径（提供者的类名跟随在包名后面；举个例子，“com.example.MySuggestionProvider”）。数据库模式必须包含[DATABASE_MODE_QUERIES](#)并可以选择性的包含[DATABASE_MODE_2LINES](#)，这个模式在你的建议项表中增加了另一列，允许你为每个建议项提供第二行文本。举个例子，如果你想为每个建议项提供两行文本：

```
public final static int MODE = DATABASE_MODE_QUERIES |
DATABASE_MODE_2LINES;
```

现在在你的应用程序清单中使用与你[SearchRecentSuggestionsProvider](#)类（和搜索配置）中同样的鉴权字符串来声明你的内容提供者。举例：

```
<application>
    <provider android:name=".MySuggestionProvider"
               android:authorities="com.example.MySuggestionProvider" />
    ...
</application>
```

修改搜索配置

要配置的让系统去使用你的建议项提供者，你需要在你搜索配置的[`<searchable>`](#)元素内添加 `android:searchSuggestAuthority` 和 `android:searchSuggestSelection` 属性。举个例子：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="com.example.MySuggestionProvider"
    android:searchSuggestSelection=" ? " >
</searchable>
```

`android:searchSuggestAuthority`的值应该是你内容提供者的全路径名，这个名字必须完全匹配在内容提供者中使用的鉴权（上面例子中的**AUTHORITY**字符串）。`android:searchSuggestSelection`的值必须是前缀为一个空格的一个单一的问号，“?”，这是SQLite选择参数的一个简单的占位符（会被用户输入的查询文本自动替换）。

保存查询

要填充你的最近查询的集合，添加每个你搜索Activity接收到的查询到你的[SearchRecentSuggestionsProvider](#)中。要做这个事情，每次你的搜索Activity接收到一个查询的时候创建一个[SearchRecentSuggestions](#)的实例并调用[saveRecentQuery\(\)](#)。举个例子，这里展示了你可以怎样在你搜索Activity的[onCreate\(\)](#)方法期间保存查询。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    file:///D:/guide/Adding_Recent_Query_Suggestions[2015/9/23 19:15:34]
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.main);

Intent intent = getIntent();

if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
    String query = intent.getStringExtra(SearchManager.QUERY);
    SearchRecentSuggestions suggestions = new
SearchRecentSuggestions(this,
    MySuggestionProvider.AUTHORITY,
    MySuggestionProvider.MODE);
    suggestions.saveRecentQuery(query, null);
}
}

```

[SearchRecentSuggestionsProvider](#)构造函数需要与你在内容提供者中声明的一样的鉴权和数据库模式。[saveRecentQuery\(\)](#)取搜索查询字符串作为第一个参数并可选的用第二个参数来包含建议项的第二行数据（或者空）。第二个参数仅在你用[DATABASE_MODE_2LINES](#)来为你的搜索建议项启用双行模式的情况下使用。如果你启用了双行模式，那么当系统在寻找匹配的建议项时，这个第二行也会被用作查询文本的匹配。

清除建议项数据

为了保护用户的隐私，你应该总是提供一个方法给用户清楚最近搜索建议项。要清楚查询历史，调用[clearHistory\(\)](#)。举个例子：

```

SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,
    HelloSuggestionProvider.AUTHORITY,
    HelloSuggestionProvider.MODE);
suggestions.clearHistory();

```

选择一个“清除搜索历史”的菜单项或设置项或按钮来执行这段代码。你也应该提供一个确认对话框来确认用户是否真要删除他们的搜索历史。

来自“[index.php?title=Adding_Recent_Query_Suggestions&oldid=8815](#)”

1个分类：

- [Search](#)

Adding Custom Suggestions

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/search/adding-custom-suggestions.html>

翻译：[futurexiong](#)

更新：2012.07.13

目录

- [1 添加自定义建议项](#)
 - [1.1 基础知识](#)
 - [1.2 修改搜索配置](#)
 - [1.3 创建一个Content Provider](#)
 - [1.3.1 处理建议项请求](#)
 - [1.3.1.1 从Uri中获取请求文本](#)
 - [1.3.1.2 从selection参数中获取请求文本](#)
 - [1.3.2 创建建议表](#)
 - [1.4 为建议项声明一个Intent](#)
 - [1.4.1 声明意图操作](#)
 - [1.4.2 声明意图数据](#)
 - [1.5 处理Intent](#)
 - [1.6 改写查询文本](#)

添加自定义建议项

当使用Android搜索对话框或者搜索部件时，你可以提供自定义搜索建议项，这些建议项从你的应用程序数据中创建出来。举个例子，你的应用程序是一个字典，你可以把字典中那些与当前输入文本相匹配的单词作为建议项。那些是最有价值的建议项，因为你可以高效地预测到用户所需并提供访问它的实例。图1展示了一个带自定义建议项的搜索对话框。一旦你提供了自定义搜索建议项，你同样可以使它们对系统范围的**Quick Search Box**有效，提供在你应用程序外访问你内容的方式。在你开始使用这个教程去增加自定义建议项之前，你需要已经实现了你应用程序搜索所需的搜索对话框和搜索部件。如果还没有，请看[Creating a Search Interface](#)。

基础知识

当用户选择了一个自定义建议项，Android系统会发送一个[Intent](#)到你的搜索Activity中。鉴于一个普通的搜索查询发送一个带[ACTION_SEARCH](#)action的intent，你反而可以限定你的自定义建议项使用[ACTION_VIEW](#)（或者其它intent action），并可以包含跟选中建议项相关的数据。回到字典的例子上来，当用户选择了一个建议项，你的应用程序可以马上打开那个单词的定义，而不是在字典中搜索那些匹配项。要提供自定义建议项，要做以下事情：

- 实现一个基本的搜索Activity，正如[Creating a Search Interface](#)所描述的那样。
- 修改搜索配置中关于提供自定义建议项的content provider的信息。
- 为你的建议项创建一个表（比如在一个[SQLiteDatabase](#)中），并用所需要的列来格式化这个表。
- 创建一个[Content Provider](#)来访问你的建议项表并在你的manifest文件中声明这个provider。
- 定义当用户选择一个建议项时将会被发送的[Intent](#)的类型（包括自定义的action和自定义的数据）。

就像Android系统显示搜索对话框那样，它同样可以显示你的搜索建议项。你所需要的是一個能从中取得你建议项的content provider。如果你对创建content provider不熟悉，在你继续往下做前阅读[Content Providers](#)开发教

程。当系统识别到你的**Activity**是可用于搜索的并提供了搜索建议项，当用户键入查询时以下步骤将会发生：

1. 系统会取得搜索查询文本（不管当前键入了什么）并对管理你的建议项的**content provider**执行一次查询。
2. 你的**content provider**返回一个指向跟搜索查询文本相关的所有建议项的**Cursor**。
3. 系统展示这个**Cursor**提供的建议项的列表。



一旦自定义建议项被展示出来，以下情况可能会发生：

- 如果用户键入另外一个关键字，或者以任何方式改变了查询内容，上述的步骤将会被重复，并且建议项列表也会得到适当的更新。
- 如果用户执行了搜索，建议项会被忽略并且系统会使用一般的**ACTION_SEARCH****intent**来将搜索传递到你的搜索**Activity**中去。
- 如果用户选择了一个建议项，一个携带自定义**action**和自定义**data**的**intent**将会被发送到你的搜索**Activity**中，这样你的应用程序就可以打开建议的内容。

修改搜索配置

要支持自定义建议项，在你搜索配置文件的**<searchable>**元素下加入**android:searchSuggestAuthority**属性。举个例子：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"

    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"

</searchable>
```

你可能需要一些额外的属性，这取决于你关联到每个建议项的**intent**类型和你想如何将查询格式化至你的**content provider**。其他的可选属性会在下一部分中讨论。

创建一个Content Provider

为自定义建议项创建一个content provider需要先掌握有关content provider的知识，这些知识涵盖在[Content Provider](#)开发指南当中。就一个自定义建议项的content provider大部分而言跟任意其他的content provider都是一样的。但是，对你提供的每个建议项来说，[Cursor](#)中各行必须包含指定的列，这些列是系统能够理解的并使用来格式化建议项的。

当用户开始往搜索对话框和搜索部件中键入内容时，系统会在每次一个字母被键入时调用query()来查询你建议项的content provider。在你query()方法的实现中，你的content provider必须查询你的建议项数据并返回一个[Cursor](#)，这个Cursor指向你觉得是好建议的那些行。有关为自定义建议项创建一个content provider的详细信息会在以下两部分讨论：

[处理建议项请求](#)

系统如何发送请求到你的content provider以及如何处理这些请求

[建立一个建议项的表](#)

如何在每个查询都返回的[Cursor](#)中定义系统期望的列名

处理建议项请求

当系统从你的content provider中请求建议项的时候，会调用你content provider的query()方法。你必须实现这个方法来查询建议项数据并返回一个指向你认为有关的建议项的[Cursor](#)。

以下是系统传递给你query()方法的参数概要(按顺序排列)：

uri

总是一个content类型的[Uri](#)，格式如下：

```
content://your.authority/optional.suggest.path/SUGGEST_URI_PATH_QUE
```

系统默认的行为是传递这个URI并在URI后面拼接上查询文本。比如：

```
content://your.authority/optional.suggest.path/SUGGEST_URI_PATH_QUE
```

末尾的查询文本是使用**URI**编码规则编码过的，所以你可能需要在执行查询之前解码它。

只有当你在你的搜索配置文件中为**android:searchSuggestPath**属性设置了路径的时候才需要**optional.suggest.path**这个部分。只有多个搜索**activity**需共用同一个**content provider**时，才需要用到这个部分，这种情况下，你需要区分建议项请求的来源。

注意:[SUGGEST_URI_PATH_QUERY](#)并不属于**URI**提供的字符串，而是你要指向此路径所需的常量。

projection

总为**null**

selection

该值由你搜索配置文件中的**android:searchSuggestSelection**属性提供，如果你没声明**android:searchSuggestSelection**这个属性则该值为**null**。更多使用信息请参见下文[get the query](#)。

selectionArgs

如果你在你的搜索配置中声明了**android:searchSuggestSelection**的属性，那么这个数组类型的参数的第一个(也只有一个)元素包含了搜索请求文本。如果没声明的话，则这个参数为**null**。更多使用信息请参见下文[get the query](#)。

sortOrder

总为**null**

系统可以使用两种方式来发送你的请求。默认的方式是把请求文本包含在作为uri参数传递的**content URI**的最末尾。但是，如果你搜索配置文件的**android:searchSuggestSelection**属性包含了**selection**的值，那么系统将会使用**selectionArgs**字符串数组的第一个元素来传递请求文本。这两种方式都会在后面说明。

从**Uri**中获取请求文本

请求文本默认地是附在uri参数(一个Uri对象)的最后一段上。在这种情况下要获取请求文本，简单的使用[getLastPathSegment\(\)](#)方法就可以了。比如：

```
String query = uri.getLastPathSegment().toLowerCase();
```

这个方法返回Uri的最后一段，也就是用户输入的请求文本。

从selection参数中获取请求文本

相对于使用Uri这种方式获取请求文本，你可能会觉得让你的[query\(\)](#)方法接收它执行查询所需的所有内容会更为合理，并且你希望selection和selectionArgs能携带合适的值。在这种情况下，用你SQLite语句的selection字符串来添加android:searchSuggestSelection到你的搜索配置文件中。selection字符串中包含一个作为占位符的问号 ("?") 来代表实际要搜索的请求文本。

例如下面显示了如何运用 android:searchSuggestSelection属性来创建全文搜索语句：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"

    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:searchSuggestSelection="word MATCH ?" >
</searchable>
```

运用这一配置，query()方法提供了“word MATCH ?”选择参数和selectionArgs参数用于搜索查询。当传递这些参数到SQLite方法时，对应参数会合并在一起（问号替换为查询文本）。如果选择接受这种查询方式，就要向查询文本添加通配符，并附加给selectionArgs参数。

上述例子的另一个新特性是android:searchSuggestIntentAction，它定义了当用户选择建议时发送给每一意向的操作。

创建建议表

当你用Cursor向系统返回建议时，系统预计每行的特定列。所以不管是在设

备的SQLite数据库，web服务器的数据库还是其他格式数据库来存储建议数据，都要格式化建议并且用光标来呈现。系统给出几列，但只有两个是必需的。

_ID

各个建议唯一的整型行ID。系统需要此ID在ListView提出建议。

SUGGEST_COLUMN_TEXT_1

该字符串可以作为一个建议。

下列几行都是可选的（而且大多数会在下面章节进一步讨论）：

SUGGEST_COLUMN_TEXT_2

字符串。如果Cursor包含这一字符串，那么所有建议都会在two-line格式中提供。此列中的字符串显示为主要文本下方的文本中第二个较小行。它显示为NULL或者空用来表示没有二次文本。

SUGGEST_COLUMN_ICON_1

可绘制资源，内容或文本URI字符串。如果Cursor包含这一字符串，那么所有建议都会显示为左侧可绘制图标的图标加文本格式。显示为NULL或零用来表示此行中没有图标。

SUGGEST_COLUMN_ICON_2

可绘制资源，内容或文件URI字符串。如果Cursor包含此列，那么所有建议都会显示为右侧图标的图标加文本格式。显示为NULL或零用来表示此行中没有图标。

SUGGEST_COLUMN_INTENT_ACTION

意图操作字符串。如果该列存在并且包含给定行的值，当生成建议意图时会使用定义的操作。如果没有提供该元素，则从搜索配置的`android:searchSuggestIntentAction`域采取操作。如果所有建议的操作均相同，指定使用`android:searchSuggestIntentAction`的操作并忽略此列会更加有效。

SUGGEST_COLUMN_INTENT_DATA

数据URI字符串。如果此列存在并且包含给定行的值，当生成建议意图时可以使用此数据。如果没有提供这一元素，就要从搜索配置的 android:searchSuggestIntentData 域来获取数据。如果不提供资源，意图数据字段为空。如果所有建议的数据相同，或者可以使用常量和特定ID来描述，那么使用 android:searchSuggestIntentData 来指定并忽略此项会更有效。

SUGGEST_COLUMN_INTENT_DATA_ID

URI路径字符串。如果此列存在并包含给定行的值，那么"/"和此值要附加到意图中的数据字段。

SUGGEST_COLUMN_INTENT_EXTRA_DATA

任意数据。如果该列存在并包含给定行的值，那么这就是生成建议意图时的额外数据。如果没有提供，那么意图的额外数据字段为空。

SUGGEST_COLUMN_QUERY

如果此列存在而且给定行存在该元素，形成建议查询时可以使用这一数据。如果建议操作是 ACTION_SEARCH，可以选择其他方式。

SUGGEST_COLUMN_SHORTCUT_ID

仅当提供快速搜索框建议时使用。此列表示 搜索建议是否应作为快捷方式存储，以及是否要进行验证。当用户点击快速搜索框的建议时会形成快捷方式。如果缺失的话，结果会存储为不再更新的快捷方式。如果设置为 SUGGEST_NEVER_MAKE_SHORTCUT，结果就不会存储为快捷方式。

SUGGEST_COLUMN_SPINNER WHILE_REFRESHING

仅当提供快速搜索框的建议时使用。此列指定了在快速搜索栏刷新快捷方式时应该显示 spinner 而不是 SUGGEST_COLUMN_ICON_2 图标。

为建议项声明一个 Intent

当用户从搜索对话框或者窗口小部件下方的列表选择建议时，系统会发送搜索活动的自定义意图。您必须定义意图的操作和数据。

声明意图操作

自定义建议中最常见的意图操作是 ACTION_VIEW，想要打开联系人信息或者网页时非常合适。

根据所有建议是否要使用相同意图操作，可以通过两种方式定义操作：

A. 使用搜索配置文件的 android:searchSuggestIntentAction 属性来定义所有建议的操作。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"

    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.Intent.action.VIEW" >
</searchable>
```

B. 使用 SUGGEST_COLUMN_INTENT_ACTION 列来定义个人建议的操作。

把 SUGGEST_COLUMN_INTENT_ACTION 列添加到建议表，并且在每个建议中放置所用操作（比如 "android.Intent.action.VIEW"）。

同时可以把这两种技术结合起来。you can include the android:searchSuggestIntentAction attribute with an action to be used with all suggestions by default, 然后通过在 SUGGEST_COLUMN_INTENT_ACTION 列来声明不同的操作以便覆盖原操作。如果不包括 SUGGEST_COLUMN_INTENT_ACTION 列的值，就要使用 android:searchSuggestIntentAction 属性提供的意图。

声明意图数据

当用户选择建议时，搜索活动会使用定义的操作来接收意图，但是意图必须携带数据以便确定要选择的建议。具体来说，数据应该是每个建议唯一的标识，比如 SQLite 表中的行 ID。接收意图时可以使

用getData()或getDataString()来检索附加数据。

You can define the data included with the intent in two ways: A.在SUGGEST_COLUMN_INTENT_DATA中定义每个建议的数据。

用每一行固有数据来填充SUGGEST_COLUMN_INTENT_DATA栏，这样可以为建议表中的各意图提供所有必要数据信息。此列的数据会按照定义连接到意图。然后就可以用getData()或getDataString()进行检索。

B.把数据URI分成两部分：所有建议的通用部分和各建议的特定部分。把这些部分分别放入可搜索配置的android:searchSuggestIntentData属性和建议表的SUGGEST_COLUMN_INTENT_DATA_ID列。

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint"

    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
        android:searchSuggestIntentAction="android.intent.action.VIEW"
        android:searchSuggestIntentData="content://com.example/databale"
>
</searchable>
```

处理Intent

既然提供了自定义搜索建议，就需要用户选择建议后由搜索活动来控制意向。这是除处理ACTION_SEARCH之外搜索活动做的事情。下面示例显示了活动onCreate()回调时如何处理意图：

```
Intent intent = getIntent();
if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
    // Handle the normal search query case
    String query = intent.getStringExtra(SearchManager.QUERY);
    doSearch(query);
} else if (Intent.ACTION_VIEW.equals(intent.getAction())) {
    // Handle a suggestions click (because the suggestions all use
    ACTION_VIEW)
    Uri data = intent.getData();
    showResult(data);
}
```

改写查询文本

如果用户使用定向控制通过建议列表进行导航，查询文本默认情况下不会更新。但是可以暂时改写文本框显示的查询文本。这样用户就可以看到建议正在进行的查询，然后选择搜索框并且编辑查询。

可以用下列方式来查询文本：

- a、用"queryRewriteFromText"值为搜索配置添加`android:searchMode`属性。这种情况建议`SUGGEST_COLUMN_TEXT_1`的内容被用来重写查询文本。
- b、用"queryRewriteFromData"值 向搜索配置添加`android:searchMode`属性。这种情况下建议`SUGGEST_COLUMN_INTENT_DATA` 的内容被用来重写查询文件。这只能用于URI或者其他用户可见的数据格式，比如HTTP URL。内部URI模式不应该以这种方式来重写查询。
- c、提供建议表`SUGGEST_COLUMN_QUERY`唯一的查询文本字符串。如果当前栏包含了当前建议的值，就可以用来改写查询文本。

来自 "[index.php?title=Adding_Custom_Suggestions&oldid=13861](#)"

1个分类:

- [开发指南 - Android API Guides](#)

Searchable Configuration

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/search/searchable-config.html>

翻译： futurexiong

更新： 2012.08.11

搜索配置

为了在Android系统的协助下(把搜索查询传递到Activity中并提供搜索建议项)实现搜索，你的应用必须以一个XML文件的形式提供给系统一个搜索配置。这一页将从搜索配置的语法以及使用上来描述它。更多如何为你的应用实现搜索功能的信息，请从关于[Creating a Search Interface](#)的开发指南开始阅读。

文件路径：

`res/xml/filename.xml`

Android使用文件名作为资源ID。

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="string resource"
    android:hint="string resource"
    android:searchMode=[ "queryRewriteFromData" | "queryRewriteFromText" ]
    android:searchButtonText="string resource"
    android:inputType="inputType"
    android:imeOptions="imeOptions"
    android:searchSuggestAuthority="string"
    android:searchSuggestPath="string"
    android:searchSuggestSelection="string"
    android:searchSuggestIntentAction="string"
    android:searchSuggestIntentData="string"
    android:searchSuggestThreshold="int"
    android:includeInGlobalSearch=[ "true" | "false" ]
    android:searchSettingsDescription="string resource"
    android:queryAfterZeroResults=[ "true" | "false" ]
    android:voiceSearchMode=[ "showVoiceSearchButton" | "launchWebSearch" | "useSearchBar" ]>
```

```

"launchRecognizer"]
    android:voiceLanguageModel=[ "free-form" | "web_search" ]
    android:voicePromptText="string resource"
    android:voiceLanguage="string"
    android:voiceMaxResults="int"
>
<actionkey
    android:keycode="KEYCODE"
    android:queryActionMsg="string"
    android:suggestActionMsg="string"
    android:suggestActionMsgColumn="string" >
</searchable>

```

元素：

<searchable>

定义所有Android系统用于提供辅助搜索的搜索配置。

属性：

android:label

字符串资源。(必须的。)你应用的名称。它应该跟你manifest文件中[<activity>](#)或者[<application>](#)元素中android:label属性的名称一样。这个标签只有当你设置android:includeInGlobalSearch为true的时候才对用户可见，在这种情况下，这个标签作为系统搜索设置中的一个可搜索项用来辨别你的应用。

android:hint

字符串资源。(推荐的。)当搜索框文本区域没有文本输入时显示的文本。它提示用户什么内容是可搜索的。为了跟其他Android应用保持一致性，你应该用"搜索 <内容-或者-产品>"这种格式来格式化android:hint的字符串。比如，"搜索歌曲或者艺术家"或者"搜索YouTube"。

android:searchMode

关键字。设置额外的模式来控制搜索的表现。当前可用的模式定义了当自定义建议项获取到焦点时搜索文本该如何被改写。以下的模式值是可接受的：

Value	Description
"queryRewriteFromText"	使用 SUGGEST_COLUMN_TEXT_1 这一列的值来改写搜索文本。
"queryRewriteFromData"	使用 SUGGEST_COLUMN_INTENT_DATA 这一列的值来改写搜索文本。这应该仅用于

当[SUGGEST_COLUMN_INTENT_DATA](#)里面的值对用户的检查和编辑是合适的，通常HTTP URI的就是这样。

详见[Adding Custom Suggestions](#)中改写搜索文本的相关讨论。

android:searchButtonText

字符串资源。显示在搜索按钮上的文本。按钮默认的展示一个搜索图标(放大镜)，这对国际化来说是理想的，所以你不应当使用这个属性来改变按钮的图标，除非按钮执行的行为是搜索以外的东西(比如Web浏览器中的一个URI请求)。

android:inputType

关键字。定义了使用的输入法的类型(比如软键盘的类型)。对于大多数对期望输入的文本没有限制的搜索来说，你不需要用到这个属性。
在[inputType](#)里查看这个属性的合法值。

android:imeOptions

关键字。为输入法提供额外的选项。对于大多数对期望输入的文本没有限制的搜索来说，你不需要用到这个属性。默认输入法的执行键值是"actionSearch"(在软键盘上提供一个"搜索"按钮来代替回车符)。
在[imeOptions](#)里查看这个属性的合法值。

搜索建议项的属性

如果你定义了一个content provider来生成搜索建议项，你需要定义额外的属性来配置与content provider的通讯。当提供搜索建议项的时候，你需要以下[`<searchable>`](#)属性的一部分：

android:searchSuggestAuthority

字符串。(提供搜索建议项必须的。)这个值必须跟Android manifest中[`<provider>`](#)元素里[android:authorities](#)属性提供的鉴权字符串一致。

android:searchSuggestPath

字符串。这个路径被用作建议项查询Uri的一部分，位于Uri的前缀和authority之后，但在标准的建议项路径之前。只有当你仅用一个content provider来处理不同类型的建议项(比如不同的数据类型)时会要用到这个属性，并且当你接收到这些建议项查询的时候你要有方式来区分

它们。

android:searchSuggestSelection

字符串。这个值将作为selection参数传递到你的查询函数中。通常这对你的数据库来说是一个WHERE语句，并且应该包含一个单独的问号，这个问号是用户实际键入查询文本的占位符(比如"query=?")。当然你也可以通过在selectionArgs参数中使用任意非空值来触发查询文本的传递(然后忽略selection参数)。

android:searchSuggestIntentAction

字符串。当用户点击自定义搜索建议项时默认使用的intent action(比如"android.intent.action.VIEW")。如果这个值没有被选中的建议项覆盖(通过SUGGEST_COLUMN_INTENT_ACTION这一列)，当用户点击一个建议项的时候这个值将会被赋给Intent的action字段。

android:searchSuggestIntentData

字符串。当用户点击自定义搜索建议项时默认使用的intent data。如果这个值没有被选中的建议项覆盖(通过SUGGEST_COLUMN_INTENT_DATA这一列)，当用户点击一个建议项的时候这个值将会被赋给Intent的数据字段。

android:searchSuggestThreshold

整型。触发建议项查询的最小字符数。仅保证系统在输入字符数小于这个阀值的情况下不会查询你的content provider。默认值是0。

想要了解更多关于以上搜索建议项的属性，[Adding Recent Query Suggestions](#)和[Adding Custom Suggestions](#)的开发指南。

Quick Search Box属性

要让你的自定义搜索建议项对Quick Search Box可用，你需要以下<searchable>属性的一部分：

android:includeInGlobalSearch

布尔值。(对在Quick Search Box里提供搜索建议项来说是必须的。)如果你想让你的建议项被包括在可全局访问的Quick Search Box里，把这个值设为"true"。在你的建议项能在Quick Search Box里显示之前，用户仍然必须在Quick Search Box的设置里面启用你的应用作为一个可搜索项。(译者注：就是说设置这个值为"true"的情况下用户还要在Quick Search Box的设置里面勾选上你的应用，这样才能在Quick Search Box里显示你的建议项。)

android:searchSettingsDescription

字符串。为你提供给Quick Search Box的搜索建议项提供一个简要说明，它会显示在(Quick Search Box设置里面)你应用对应的可搜索项条目中。你的描述应该扼要地描述出什么内容是可搜索的。比如，用“艺术家，唱片，专辑”来描述音乐应用可搜索的内容，或者用“已保存的笔记”来描述记事本应用可搜索的内容。

android:queryAfterZeroResults

布尔值。如果你希望对之前返回0个结果的搜索的超集仍然调用你的content provider，设置这个值为“true”。比如你的content provider对“bo”返回0个结果，那么它会对“bob”重新进行查询。如果这个设为“false”，在这个单独的会话里超集将会被忽略 (“bob”不会调用一个新的查询)。这仅在搜索对话框的生存期或者使用search widget的activity的生存期中持续(当搜索对话框或者activity重新打开的时候，“bo”将再次查询你的content provider)。默认值是false。

语音搜索属性

要启用语音搜索，你需要以下<searchable>属性的一部分：

android:voiceSearchMode

关键字。(对提供语音搜索功能来说是必要的属性。)用指定的模式启用语音搜索。(设备可能不提供语音搜索功能，在这种情况下这些标志将不起作用。)以下是可接受的模式值：

Value	Description
"showVoiceSearchButton"	如果语音搜索在设备上可用，那么显示一个语音搜索的按钮。如果设置了这个模式，那么“launchWebSearch”和“launchRecognizer”两者之一也必须同时被设置(用 字符分割开来)。
"launchWebSearch"	语音搜索按钮把用户直接带到一个内建的语音web搜索activity。大多数应用不需要这个标志，因为它将用户带离了调用搜索的activity。
"launchRecognizer"	语音搜索按钮把用户直接带到一个内建的录音activity。这个activity提示用户说话，解码

语音文本，讲得到的查询文本转发到搜索activity，就好像用户在search UI输入查询文本然后按下搜索按钮一样。

android:voiceLanguageModel

关键字。语音识别系统应该使用的语言模式。以下是可接受的模式值：

Value	Description
"free_form"	为口述的查询使用"自由形式"("free-form")的语音识别。这主要是为英语优化的。这是默认值。
"web_search"	为更短的，类似检索的短语使用网页检索词识别。这种模式支持的语言比"free-form"的更多。

你也可以在[EXTRA_LANGUAGE_MODEL](#)里查看更多信息。

android:voicePromptText

字符串。显示在语音输入框里的附加信息。

android:voiceLanguage

字符串。期望的语言，用[Locale](#)里面的字符串常量来表示(比如"de"表示德国或者"fr"表示法国)。只有期望的语言跟[Locale.getDefault\(\)](#)的当前值不一样的时候才会需要这个属性。

android:voiceMaxResults

整型。强制返回结果的最大个数，其中包含用于[ACTION_SEARCH](#)intent的主查询的"最佳"结果，这个结果总是会被提供。这个值必须大于等于1。使用[EXTRA_RESULTS](#)从intent里面获取结果。如果没有提供该值，语音识别程序将决定返回多少个结果。

<actionkey>

为搜索action指定一个设备的按键和行为。当触摸设备屏幕上的按钮时，搜索action会根据当前查询文本或者获得焦点的建议项来触发一个特定的行为。比如联系人应用在提供了联系人建议项的时候，按下Call按键会提供一个搜索action来初始化一个通话，通话的对象就是当前获得焦点的联系人建议项所

对应的联系人。

不是所有的action键值在任何设备上都是可用的，也不是所有的键值都允许用这种方式来覆盖。比如"Home"键不能被用作 action key并且按下它的时候必须返回home界面。同时要确保不要为用于输入查询文本的按键定义action key。这从根本上限定了action keys只能用于拨号键和菜单键。同时也要注意action keys通常不容易被发现，所以你不应该把它们作为核心用户功能提供出来。

你必须通过指定android: keycode键值来指定一个按键，并且指定其他三个属性中至少一个的值来定义搜索action。

属性：

android: keycode

字符串。(必须的。) [KeyEvent](#) 中定义的键值(比如"KEYCODE_CALL")，用来代表你所希望响应的那个action key。这个键值被附加到将被传递到你搜索activity中的[ACTION_SEARCH](#)intent中。要查看这个键值，使用[getIntExtra\(SearchManager.ACTION_KEY\)](#)。不是所有的按键都支持搜索action，因为他们很多都被用作输入，导向和系统功能。

android: queryActionMsg

字符串。当用户正在输入查询文本时按下action key所发送的action信息。这个值被附加到[ACTION_SEARCH](#)intent中，这个intent将会被系统传递到你的搜索activity中去。要查看这个值，使用[getStringExtra\(SearchManager.ACTION_MSG\)](#)。

android: suggestActionMsg

字符串。当一个建议项获得焦点时按下action key所发送的action信息。这个值被附加到intent中，系统将这个intent(使用你为之前为建议项定义的action)传递到你的搜索activity中去。要查看这个值，使用[getStringExtra\(SearchManager.ACTION_MSG\)](#)。这仅用于你所有的建议项都支持这个action key的情况下。如果不是所有建议项都能处理同一个action key的话，你应该下面的android: suggestActionMsgColumn属性替代。

android: suggestActionMsgColumn

字符串。你content provider中的列名，用于指定当一个建议项获得焦点时用户按下action key所发送的对应于该action key的action信息。这个属性能让你在逐个建议项的基础上操控action key，这是因为与使用android: queryActionMsg属性为所有的建议项定义一个action信息不同，使用这个属性你content provider每个条目都提供自身对应的一

个action信息。

首先，你必须在你content provider中定义一列来让每个建议项都提供一个action信息，然后在这个属性中提供这个列名。系统查看你的建议项cursor，使用这个属性提供的字符串找到你action信息的那一列，然后从cursor中找出action信息的字符串。那个被查出来的字符串会被附在系统传递给你的搜索 activity的intent(这个intent使用你之前为建议项定义的action)中。要查看这个字符串，使用getStringExtra(SearchManager.ACTION_MSG)。如果选中的建议项对应的数据不存在，那么这个action key将会被忽略。

示例：

保存为res/xml/searchable.xml的XML文件：

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:searchSuggestAuthority="dictionary"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:includeInGlobalSearch="true"
    android:searchSettingsDescription="@string/settings_description" >
</searchable>
```

来自“[Index.php?title=Searchable_Configuration&oldid=8302](#)”

1个分类：[Search](#)



Drag and Drop

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

Drag and Drop

原文链接：<http://developer.android.com/guide/topics/ui/drag-drop.html>

翻译：落落琪琪

更新：2012.07.10

目录

[[隐藏](#)]

[1 拖放 - Drag and Drop](#)

- [1.1 概述](#)
 - [1.1.1 拖放过程](#)
 - [1.1.2 拖动事件监听器和回调方法](#)
 - [1.1.3 拖动事件](#)
 - [1.1.4 拖动阴影](#)
- [1.2 设计一个拖放操作](#)
 - [1.2.1 开始一个拖动动作](#)
 - [1.2.2 回应一个拖动的开始](#)
 - [1.2.3 在拖动过程中处理事件](#)
 - [1.2.4 回应一个释放动作](#)
 - [1.2.5 回应一个拖动的结束](#)
 - [1.2.6 回应拖动事件：一个例子](#)

拖放 - Drag and Drop

使用Android的拖放框架，允许用户通过一个图形化的拖放动作，把数据从当前布局中的一个视图上转移到另一个视图上。这个框架包含了一个拖动事件类，拖动监听器和一些辅助的方法和类。

虽然这个框架主要是为了数据的移动而设计的，但是你可以将这些移动的数据提供给其他的UI操作使用。例如：你可以创建一个当用户把一个彩色图标拖到另一个彩色图标上时，将颜色混合起来的应用。接下来本文将描述关于这个拖放框架的数据移动的内容。

概述

当用户执行一些被当作是开始拖动数据的信号的手势时，一个拖放动作就开始了。作为回应，你的应用程序告诉系统拖动动作开始了。系统回调你的应用程序 获取一个代表数据正在被拖动的图形。当用户的手指将这个代表图形（一个拖动阴影）移动到当前布局上时，系统分别发送拖动事件给拖动事件监听器对象，与布局 中的[View](#)相联系的拖动事件回调方法。一旦用户释放这个拖动阴影，系统就结束拖动操作。

你可以通过在一个类中实现[View.OnDragListener](#)，创建拖动事件监听器。然后通过视图对象的[setOnDragListener\(\)](#)方法，为视图设置一个拖动事件监听器对象。每个视图对象都可以有一个[onDragEvent\(\)](#) 回调方法。以上两个方法在[拖动事件监听器和回调方法](#) 中会详细介绍。

注意：为了简便起见，在接下来的章节中，把程序接收拖动事件称为“拖动事件监听器”，尽管事实上它也有可能是一个回调方法。

当开始一个拖动，就同时包括了正在拖动的数据，以及描述这些数据的元数据作为系统回调的一部分。在拖动过程中，系统会发送拖动事件给布局中的每个视图的拖动事件监听器或回调方法。这些监听器或回调方法可以使用元数据来决定他们是否想要接收那些被拖动的数据。如果用户将数据拖动到一个视图对象上，并且该视图对象的监听器或回调方法事先已经告诉过系统想要接收拖动的数据，那么系统就会把这些数据发送给拖动事件中的监听器或回调方法。

你的应用程序通过调用[startDrag\(\)](#)方法告诉系统开始一个拖动，也就是告诉系统可以开始发送拖动事件了。[startDrag\(\)](#)也会发送你正在拖动的数据。

你可以调用当前布局中任意一个相关联视图的[startDrag\(\)](#)方法。系统只会利用视图对象获得进入你布局中的全局设置的权限。

一旦你的应用程序调用[startDrag\(\)](#)方法，剩下的过程就是使用系统发送给布局中的视图对象的事件。

拖放过程

拖放过程包括以下四个步骤或状态：

开始

为了响应用户开始拖动的手势，你的应用程序通过调用 [startDrag\(\)](#) 方法告诉系统开始一个拖动动作。[startDrag\(\)](#) 的参数提供被拖动的数据，描述被拖动数据的元数据以及一个绘制拖动阴影的回调方法。

系统的首次回应是通过回调你的应用程序去获得一个拖动阴影。然后将这个拖动阴影显示在设备上。

接着，系统发送一个操作类型为 [ACTION_DRAG_STARTED](#) 的拖动事件给当前布局中的所有视图对象的拖动事件监听器。为了继续接收拖动事件，包括一个可能的拖动事件，拖动事件监听器必须返回 `true`。这样就在系统中注册了一个监听器。只有被注册过的监听器才继续接收拖动事件。这时候，监听器也可以改变他们的视图对象的外观，来表明监听器可以接收一个拖放事件。

如果拖动事件监听器返回值为 `false`，那么在当前操作中就接收不到拖动事件，直到系统发送一个操作类型为 [ACTION_DRAG_ENDED](#) 的拖动事件。通过发送 `false`，监听器告诉系统它对拖动操作不感兴趣，并且不想接收被拖动的数据。

继续

用户继续拖动。当拖动阴影和视图对象的边界框相交，系统会发送一个或多个拖动事件给视图对象的拖动事件监听器(如果该事件监听器已经被注册为接收事件)。作为回应，监听器可以选择改变响应拖动事件的视图对象的外观。例如，如果事件表明阴影已经进入了视图的边界框 (操作类型为 [ACTION_DRAG_ENDED](#))，那么监听器就可以高亮视图以作出回应。

释放

用户在可以接收数据的视图的边界框内释放拖动阴影。系统发送一个操作的类型为 [ACTION_DROP](#) 拖动事件给视图对象的监听器。这个拖动事件包括调用 [startDrag\(\)](#) 方法传给系统的数据。如果接收释放动作的代码执行成功，那么这个监听器会被期望返回 `true` 给系统。

注意，这一步只会在用户在监听器被注册为接收拖动时事件的视图的边界框内释放这个拖动阴影的情况下才会发生。如果用户在其他情况下释放这个拖动阴影，[ACTION_DROP](#) 的拖动事件就不会被发送。

终止

在用户释放拖动阴影并且系统发送出一个操作类型为 [ACTION_DROP](#) 的拖动事件 (如果有必要的话) 之后，系统发送出一个操作类型为 [ACTION_DRAG_ENDED](#) 的拖动事件来表明这个拖动操作已经结束了。不管用户在哪里释放这个拖动阴影，这个步骤都会发生。这个事件会发送给每一个被注册为接收拖动事件的监听器，即使这个监听器已经接收了 [ACTION_DROP](#) 事件。

上面四步中的每一步都会在[设计一个拖放操作](#)一文中详细地介绍。

拖动事件监听器和回调方法

一个视图通过实现了[View.OnDragListener](#) 的拖动事件监听器或通过它自己的[onDragEvent\(DragEvent\)](#) 回调方法来接收拖动事件。当系统调用这个方法或监听器时，系统传递给他们一个[DragEvent](#)对象。

在大多数情况下，你可能会想要使用监听器。当你设计界面时，通常不会继承视图类，但使用回调方法时，为了要重写这个方法，就会迫使你去继承视图类。相比之下，你还可以实现一个监听器类，然后在几个不同的视图对象中使用它。你也可以将这个监听器类作为一个匿名内部类去实现。调用[setOnDragListener\(\)](#)这个方法，就可以为一个视图对象设置监听器。

视图对象可以同时有一个监听器和一个回调方法。如果在这种情况下，系统会首先调用监听器。除非监听器返回的是false，要不然系统不会去调用回调方法。

[onDragEvent\(DragEvent\)](#)方法和[View.OnDragListener](#)的结合跟触屏事件的[onTouchEvent\(\)](#)与[View.OnTouchListener](#)的结合是相似的。

拖动事件

系统以[DragEvent](#)对象的形式发送一个拖动事件。这个对象包括一个告诉监听器在拖放事件中正在发生什么的事件类型。这个对象还包括其他依赖这个事件类型的数据。

监听器调用[getAction\(\)](#)这个方法就可以获得这个事件类型。在[DragEvent](#)中，还有其他六个可能的变量，分别在表1中列出。

[DragEvent](#)对象还包括你的应用程序在调用[startDrag\(\)](#)这个方法的时候要传递给系统的那些数据。在这些数据中，有些数据只是对某些特定的事件类型有效。对每个事件类型有效的数据都列在表2中。在[设计一个拖放操作](#) 中详细描述了那些有效的事件。

表1.DragEvent的事件类型

getAction()的值	意义
ACTION_DRAG_STARTED	在应用程序调用 startDrag() 并获得一个拖动阴影之后，视图对象的拖动事件监听器就会接收到这个事件类型的事件。
ACTION_DRAG_ENTERED	当拖动阴影刚刚进入视图的边界框范围时，视图的拖动事件监听器就会接收到这个action类型的事件。这是当拖动阴影进入视图的边界框范围时监听器所接收到的第一个事件操作类型。如果监听器还

行继续为拖动阴影进入视图边界框范围之这个动作接收拖动事件的话，那么必须返回true给系统。

[ACTION_DRAG_LOCATION](#)

当拖动阴影还在视图的边界框范围内，视图的拖动事件监听器就会在接收到[ACTION_DRAG_ENTERED](#)事件之后接收到这个操作类型的事件。

[ACTION_DRAG_EXITED](#)

当视图的拖动事件监听器接收到[ACTION_DRAG_ENTERED](#)这个事件，并且至少接收到一个[ACTION_DRAG_LOCATION](#)事件，那么在用户把拖动阴影移除视图的边界框范围之后，该监听器就会在接收到这个操作类型的事件。

[ACTION_DROP](#)

当用户在视图对象上释放拖动阴影时，该视图对象的拖动事件监听器就会接收到这个类型的拖动事件。这个操作类型只会发送给在回应[ACTION_DRAG_STARTED](#)类型的拖动事件中返回true的那个视图对象的监听器。如果用户释放拖动阴影的那个视图没有注册监听器，或者用户在当前布局之外的任何对象上释放了拖动阴影，那么这个操作类型就不会被发送。

如果释放动作顺利，监听器应该返回true，否则应该返回false。

[ACTION_DRAG_ENDED](#)

当系统结束拖动动作时，视图对象的拖动事件监听器就会接收到这个类型的拖动事件。这种操作类型不一定是前面有一个[ACTION_DROP](#)事件。如果系统发送一个[ACTION_DROP](#)，并接收到一个[ACTION_DRAG_ENDED](#)操作类型，并不意味着拖动事件的成功。监听器必须调用[getResult\(\)](#)方法来获取在回应[ACTION_DROP](#)事件中返回的结果。如果[ACTION_DROP](#)事件没有被发送，那么[getResult\(\)](#)就返回false。

表2.

<code>getAction()</code> 的值	<code>getClipDescription()</code> 的值	<code>getLocalState()</code> 的值	<code>getX()</code> 的值	<code>getY()</code> 的值	<code>getClipData()</code> 的值	<code>getResult()</code> 的值

ACTION_DRAG_STARTED	X	X	X			
ACTION_DRAG_ENTERED	X	X	X	X		
ACTION_DRAG_LOCATION	X	X	X	X		
ACTION_DRAG_EXITED	X	X				
ACTION_DROP	X	X	X	X	X	
ACTION_DRAG_ENDED	X	X				X

如果一个方法不包含对某个特定的操作类型有效的数据，那么就会根据该方法的返回值类型返回null或0.

拖动阴影

在拖动过程中，系统会显示一张用户拖动的图片。对数据移动而言，这张图片代表着那些正在被移动的数据。对其他操作而言，这张图片代表着拖动操作的某些环节。

这张图片就被叫做是一个拖动阴影。你可以通过你声明的[View.DragShadowBuilder](#)对象的方法去创建它，然后当你使用[startDrag\(\)](#)方法的一部分，系统调用你定义的[View.DragShadowBuilder](#)里面的回调方法去获取一个拖动阴影。

[View.DragShadowBuilder](#)类有两个构造函数：

[View.DragShadowBuilder\(View\)](#)

这个构造函数接受你的应用程序中的任意一个视图对象。它将视图对象存储[View.DragShadowBuilder](#)对象中，因此在回调过程中，你可以去访问这个构造方法，为你构造一个拖动阴影。构造方法不必和用户选择开始一个拖动的视图对象（如果有的话）相关联。

如果你使用这个构造方法，不必去继承[View.DragShadowBuilder](#)类或覆盖它的方法。默认情况下，你会得到一个与你作为参数传递的那个视图有相同外表的拖动阴影，并且该拖动阴影会居中位于用户接触的屏幕上。

[View.DragShadowBuilder\(\)](#)

如果你使用这个构造方法，在[View.DragShadowBuilder](#)对象中没有一个视图对象是有效的（这个字段被设置为null）。如果使用该构造函数，你不必继承[View.DragShadowBuilder](#)类或覆盖它的方法，你可以得到一个不可见的的拖动阴影。系统不会给出一个错误。

[View.DragShadowBuilder](#)类有两个方法：

[onProvideShadowMetrics\(\)](#)

系统在调用了[android.view.View.DragShadowBuilder.java.lang.Object.int\) startDrag\(\)](#)这个方法之后，立刻回调用这个方法。用这个方法给系统发送拖动阴影的规模和接触点。这个方法有两个参数：

`dimensions`

一个[Point](#)对象。拖动阴影的宽为`x`，高为`y`。

`touch_point`

一个[Point](#)对象。这个接触点应该是拖动过程中，在用户手指之下的拖动阴影的位置。它的X轴坐标为`x`，Y轴坐标为`y`。

[onDrawShadow\(\)](#)

在调用了[onProvideShadowMetrics\(\)](#)方法之后，系统立刻调用[onDrawShadow\(\)](#)这个方法来获取拖动阴影。这个方法只有一个参数，一个[Canvas](#)对象，该对象是系统利用你提供给[onProvideShadowMetrics\(\)](#)方法里面的参数构造出来的。利用它可以在提供给你的[Canvas](#)对象中绘制你的拖动阴影。

设计一个拖放操作

这个章节会一步一步说明如何开始一个拖动，如何在拖动过程中回应事件，如何回应一个拖动事件以及如何结束一个拖放操作。

开始一个拖动动作

用户用一个拖动的手势开始一个拖动，通常是一个在视图对象上的长按动作。作为回应，应做到以下几点：

1. 必要时，为那些已经移动的数据创建一个[ClipData](#) 和[ClipData.Item](#)对象。作为ClipData这个对象的一部分，在ClipData中提供存储在[ClipDescription](#)对

象中的元数据。因为一个拖放动作不能代表数据的移动，你可能想要使用null来代替一个实际的数据。

比如：下面这个代码片段说明了如何通过创建一个包含了ImageView的标志或标签的ClipData对象，来回应在ImageView上的一个长按动作。以下就是这些片段，第二个片段说明了如何重写View.DragShadowBuilder这个类中的方法。

```
// Create a string for the ImageView label
private static final String IMAGEVIEW_TAG = "icon bitmap"

// Creates a new ImageView
ImageView imageView = new ImageView(this);

// Sets the bitmap for the ImageView from an icon bit map (defined elsewhere)
imageView.setImageBitmap(mIconBitmap);

// Sets the tag
imageView.setTag(IMAGEVIEW_TAG);

...

// Sets a long click listener for the ImageView using an anonymous listener object that
// implements the OnLongClickListener interface
imageView.setOnLongClickListener(new View.OnLongClickListener() {

    // Defines the one method for the interface, which is called when the View is long-clicked
    public boolean onLongClick(View v) {

        // Create a new ClipData.
        // This is done in two steps to provide clarity. The convenience method
        // ClipData.newPlainText() can create a plain text ClipData in one step.

        // Create a new ClipData.Item from the ImageView object's tag
        ClipData.Item item = new ClipData.Item(v.getTag());

        // Create a new ClipData using the tag as a label, the plain text MIME type, and
        // the already-created item. This will create a new ClipDescription object within the
        // ClipData, and set its MIME type entry to "text/plain"
        ClipData dragData = new ClipData(v.getTag(), ClipData.MIMETYPE_TEXT_PLAIN, item);

        // Instantiates the drag shadow builder.
        View.DragShadowBuilder myShadow = new MyDragShadowBuilder(imageView);

        // Starts the drag
        v.startDrag(dragData, // the data to be dragged
                    myShadow, // the drag shadow builder
                    null, // no need to use local data
                    0 // flags (not currently used, set to 0)
        );
    }
})
```

2.下面这个代码片段定义了myDragShadowBuilder。它为拖动一个TextView创建了一个小的灰色矩形框拖动阴影。

```

private static class MyDragShadowBuilder extends View.DragShadowBuilder {
    // The drag shadow image, defined as a drawable thing
    private static Drawable shadow;

    // Defines the constructor for myDragShadowBuilder
    public MyDragShadowBuilder(View v) {
        // Stores the View parameter passed to myDragShadowBuilder.
        super(v);

        // Creates a draggable image that will fill the Canvas provided by the system.
        shadow = new ColorDrawable(Color.LTGRAY);
    }

    // Defines a callback that sends the drag shadow dimensions and touch point back to the
    // system.
    @Override
    public void onProvideShadowMetrics (Point size, Point touch)
        // Defines local variables
        private int width, height;

        // Sets the width of the shadow to half the width of the original View
        width = getView().getWidth() / 2;

        // Sets the height of the shadow to half the height of the original View
        height = getView().getHeight() / 2;

        // The drag shadow is a ColorDrawable. This sets its dimensions to be the same as the
        // Canvas that the system will provide. As a result, the drag shadow will fill the
        // Canvas.
        shadow.setBounds(0, 0, width, height);

        // Sets the size parameter's width and height values. These get back to the system
        // through the size parameter.
        size.set(width, height);

        // Sets the touch point's position to be in the middle of the drag shadow
        touch.set(width / 2, height / 2);
    }

    // Defines a callback that draws the drag shadow in a Canvas that the system constructs
    // from the dimensions passed in onProvideShadowMetrics().
    @Override
    public void onDrawShadow(Canvas canvas) {
        // Draws the ColorDrawable in the Canvas passed in from the system.
        shadow.draw(canvas);
    }
}

```

注意：记住你不必去继承[View.DragShadowBuilder](#)。构造方法[View.DragShadowBuilder\(View\)](#)会创建一个默认的拖动阴影，这个拖动阴影与传递给它的View参数一样大，并且位于以接触点为中心的位置。

回应一个拖动的开始

在拖动过程中，系统将拖动事件分配给当前布局中的视图对象的拖动事件监听器。监听器应该调用[getAction\(\)](#)这个方法获取操作类型。在一个拖动开始时，这个方法返回[ACTION_DRAG_STARTED](#)。

作为回应一个操作类型为 [ACTION_DRAG_STARTED](#)的事件，监听器应该做到以下几点：

1. 调用[getClipDescription\(\)](#)方法获取[ClipDescription](#)。使用在[ClipDescription](#) 中的MIME类型的方法查看监听器是否接收被拖动的数据。如果拖放操作没有代表数据的移动，那么这个步骤就不是必须的。
2. 如果监听器可以接收一个拖动，它必须返回true。这样会告诉系统继续发送拖动事件给监听器。如果监听器不接收一个拖动，就会返回false，系统就会停止发送拖动事件直到它发送[ACTION_DRAG_ENDED](#)。

注意对于[ACTION_DRAG_STARTED](#)事件，以下这些[DragEvent](#)的方法都是无效的：[getClipData\(\)](#)、[getX\(\)](#)、[getY\(\)](#)和[getResult\(\)](#)。

在拖动过程中处理事件

在拖动过程中，作为回应[ACTION_DRAG_STARTED](#)拖动事件，监听器返回true来继续接受拖动事件。监听器在拖动过程中接收到的拖动事件类型取决于拖放阴影的位置以及监听器视图的可见性。

在拖动过程中，监听器首先使用拖动事件来决定是否应该改变他们的视图的外观。

在拖动过程中，[getAction\(\)](#)返回以下三个变量中的一个：

- [ACTION_DRAG_ENTERED](#): 当接触点（屏幕上位于用户手指下的那个点）进入监听器的视图的边界框范围内时监听器会接收到这个事件。
- [ACTION_DRAG_LOCATION](#): 一旦监听器接收到[ACTION_DRAG_LOCATION](#)事件，在它接收到[ACTION_DRAG_EXITED](#)事件之前，接触点每移动一次，它都会接收到一个新的[ACTION_DRAG_LOCATION](#)事件。方法[getX\(\)](#)和[getY\(\)](#)会返回接触点的X轴和Y轴的坐标。
- [ACTION_DRAG_EXITED](#): 在拖动阴影不再位于监听器视图的边界框范围之内时，这个事件会被发送给以前接收到[ACTION_DRAG_ENTERED](#)事件的监听器。

监听器不必对这些操作类型中的任意一个作出反应。如果监听器返回一个值给系统，它会被忽略掉。下面是应对这些动作类型的一些准则：

- 在回应[ACTION_DRAG_ENTERED](#)或者[ACTION_DRAG_LOCATION](#)时，监听器可以通过改变视图的外观来表明它将要接收到一个拖动。
- 具有[ACTION_DRAG_LOCATION](#)操作类型的事件包含了对[getX\(\)](#)和[getY\(\)](#)方法有效的数据，相应的接触点的位置。监听器可能可以使用这些信息来改变在接触点的视图的部分的外观。监听器也可以用这些信息来决定用户想要释放拖动阴影的精确位置。
- 在回应[ACTION_DRAG_EXITED](#)时，监听器应该重置它在回应[ACTION_DRAG_ENTERED](#)或[ACTION_DRAG_LOCATION](#)中应用的任何外观的变化。这

是在向用户表明视图不再是一个临近被释放的目标。

回应一个释放动作

当用户在应用程序的视图上释放拖动阴影时，并且该视图会事先报告是否可以接收被拖动的内容，系统将拖动事件分发给那个含有 [ACTION_DROP](#) 操作类型的视图。监听器应做到以下几点：

1. 调用 [getClipData\(\)](#) 方法获取最初在 [startDrag\(\)](#) 方法中应用的 [ClipData](#) 对象，并储存之。如果拖放操作没有代表数据的移动，这些都不是必须的。
2. 监听器应返回 [true](#) 来表明释放动作已顺利完成，如果没有完成的话，则返回 [false](#)。这个被返回的值成为 [ACTION_DRAG_ENDED](#) 事件中 [getResult\(\)](#) 方法的返回值。

需要注意的是，如果系统没有发出 [ACTION_DROP](#) 事件，那么 [ACTION_DRAG_ENDED](#) 事件中 [getResult\(\)](#) 方法的返回值就为 [false](#)。

对于 [ACTION_DROP](#) 事件来说，在释放动作的瞬间，[getX\(\)](#) 和 [getY\(\)](#) 方法使用接收释放动作的视图上的坐标系统，返回拖动点的 X 轴和 Y 轴的坐标。

系统允许用户在监听器不接收拖动事件的视图上释放拖动阴影。系统允许用户在应用程序 UI 的空区域或者应用程序之外的区域释放拖动阴影。在以上例子中，系统虽然会发送 [ACTION_DRAG_ENDED](#) 事件，但是不会发送一个 [ACTION_DROP](#) 事件。

回应一个拖动的结束

用户释放了拖动阴影后，系统会立即给应用程序中所有的拖动事件监听器发送 [ACTION_DRAG_ENDED](#) 类型的拖动事件，表明拖动动作结束了。

每个监听器都应该做下列事情：

1. 如果监听器在操作期间改变了 View 对象的外观，那么应该把 View 对象重置为默认的外观。这是对用户可见的操作结束的指示。
2. 监听器能够可选的调用 [getResult\(\)](#) 方法来查找更多的相关操作。如果在响应 [ACTION_DROP](#) 类型的事件中监听器返回了 [true](#)，那么 [getResult\(\)](#) 方法也会返回 [true](#)。在其他的情况下，[getResult\(\)](#) 方法会返回 [false](#)，包括系统没有发出 [ACTION_DROP](#) 事件的情况。
3. 监听器应该给系统返回 [true](#)。

回应拖动事件：一个例子

所有的拖动事件都会被拖动事件的回调方法或监听器所接收。以下代码片段是一个简单的在监听器中对拖动事件作出反应的示例。

```
// Creates a new drag event listener
mDragListen = new myDragEventListener();
view imageView = new ImageView(this);
```

```
// Sets the drag event listener for the View
imageView.setOnDragListener(mDragListen);
...
protected class myDragEventListener implements View.OnDragEventListener {
    // This is the method that the system calls when it dispatches a drag event to the
    // listener.
    public boolean onDrag(View v, DragEvent event) {
        // Defines a variable to store the action type for the incoming event
        final int action = event.getAction();
        // Handles each of the expected events
        switch(action) {
            case DragEvent.ACTION_DRAG_STARTED:
                // Determines if this View can accept the dragged data
                if (event.getClipDescription().hasMimeType(ClipDescription.MIMETYPE_TEXT_PLAIN)) {
                    // As an example of what your application might do,
                    // applies a blue color tint to the View to indicate that it can accept
                    // data.
                    v.setColorFilter(Color.BLUE);
                    // Invalidate the view to force a redraw in the new tint
                    v.invalidate();
                    // returns true to indicate that the View can accept the dragged data.
                    return(true);
                } else {
                    // Returns false. During the current drag and drop operation, this View will
                    // not receive events again until ACTION_DRAG_ENDED is sent.
                    return(false);
                }
            break;
            case DragEvent.ACTION_DRAG_ENTERED: {
                // Applies a green tint to the View. Return true; the return value is ignored.
                v.setColorFilter(Color.GREEN);
                // Invalidate the view to force a redraw in the new tint
                v.invalidate();
                return(true);
            break;
            case DragEvent.ACTION_DRAG_LOCATION:
                // Ignore the event
                return(true);
        }
    }
}
```

```
break;

case DragEvent.ACTION_DRAG_EXITED:
    // Re-sets the color tint to blue. Returns true; the return value is ignored.
    v.setColorFilter(Color.BLUE);

    // Invalidate the view to force a redraw in the new tint
    v.invalidate();

    return(true);

break;

case DragEvent.ACTION_DROP:
    // Gets the item containing the dragged data
    ClipData.Item item = event.getClipData().getItemAt(0);

    // Gets the text data from the item.
    dragData = item.getText();

    // Displays a message containing the dragged data.
    Toast.makeText(this, "Dragged data is " + dragData, Toast.LENGTH_LONG);

    // Turns off any color tints
    v.clearColorFilter();

    // Invalidates the view to force a redraw
    v.invalidate();

    // Returns true. DragEvent.getResult() will return true.
    return(true);

break;

case DragEvent.ACTION_DRAG_ENDED:
    // Turns off any color tinting
    v.clearColorFilter();

    // Invalidates the view to force a redraw
    v.invalidate();

    // Does a getResult(), and displays what happened.
    if (event.getResult()) {
        Toast.makeText(this, "The drop was handled.", Toast.LENGTH_LONG);
    } else {
        Toast.makeText(this, "The drop didn't work.", Toast.LENGTH_LONG);
    }

    // returns true; the value is ignored.
    return(true);

break;
```

```
// An unknown action type was received.  
default:  
    Log.e( "DragDrop Example" , "Unknown action type received by OnDragListener." );  
    break;  
}  
};  
};
```

来自“[index.php?title=Drag_and_Drop&oldid=8829](#)”



Accessibility

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Shepherd1st

翻译原

文：<http://developer.android.com/guide/topics/ui/accessibility/index.html>

Accessibility

很多Android用户都有一定的障碍，假如要求他们用不同的方式和Android设备交互，这些能阻止他们很充分的看或者使用触摸屏。这些包括，视觉，身体或者年龄相关的障碍。

Android提供了更人性化的特性和服务来帮助这些用户更容易的操作他们的设备，包括文字识别到语音，触摸反馈，D-pad导航来增加他们的体验。

Android应用程序开发者可以利用这些服务的优势，来让他们的应用程序更人性化，并且建立了他们自己的人性化服务。

下面的几个主题可以教你怎么使用Android应用框架使你的应用程序更人性化。

[应用程序的访问 - Making Applications Accessible](#)

开发经验和API特性来确保你的应用程序更容易

[建立可访问的服务 - Building Accessibility Services](#)

如何使用API特性来构建服务，使其他应用程序对于用户更容易。

来自“[index.php?title=Accessibility&oldid=8831](#)”



Making Applications Accessible

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：

<http://developer.android.com/guide/topics/ui/accessibility/apps.html#directional-control>

编辑者：todaay110120

目录

- [1 Making Applications Accessible\(应用程序访问\)](#)
 - [1.1 标志用户界面元素](#)
 - [1.2 启用焦点导航](#)
 - [1.2.1 启用视图焦点](#)
 - [1.2.2 焦点控制命令](#)
 - [1.3 创建可访问自定义视图](#)
 - [1.3.1 处理定向控制器的点击](#)
 - [1.3.2 实现可访问的API函数方法](#)
 - [1.3.3 发送可访问性的事件](#)
 - [1.3.4 填充可访问性的事件](#)
 - [1.3.5 提供自定义辅助内容](#)
 - [1.3.6 处理自定义触摸事件](#)

Making Applications Accessible(应用程序访问)

当用户激活设备上的在Android系统中构建的应用程序的具有可访问的功能和服务时，都可以通过可视化，物理或者与年龄有关的残疾方式来与用户进行交互。然而，你应该对应用程序的可访问性进行必要的修改优化。并且同时确保能让所有用户得到舒适的使用体验。

特别地当你使用基于框架的用户界面组件时，要确保所有用户比较容易地访问你的应用程序。如果你只是使用标准的组件来构建你的应用程序，那么以下有几个步骤能够确保你的程序具有可访问性：

1. 标志你的 `ImageButton`, `ImageView`, `EditText`, `CheckBox`控件和通过使
用[android:contentDescription](#)属性来控制其他用
户界面。

2. 通过使用定向控制器让你的所有用户界面元素
具有可访问性，例如：使用轨迹球或者方向键。

3. 通过开启你的可访问性服务来测试你的应用程
序，如通过触摸来反馈和搜索，和尝试只是使用
定向控制器来使用你的应用程序。

创建扩展于视图类的自定义控件的开发者有必要的责任确保所有使用者能够访问他们开发的组
件。本文档同样讨论了如何使自定义视图控制兼容的辅助功能服务。

标志用户界面元素

许多用户界面控件依赖使用视觉提示，来告知用户其具体的含义。例如：一个笔记应用程序可能会使用一个带有加号标志图片的图像按钮来提示用户可以通过它添加一个新的笔记。或者，一个编辑框组件可能会有一个标签在旁边来表明它的用途。此时当一个视力有问题的用户来访问你的应用程序，那么上面的视图提示通常是没有用的。

为了提供有关界面控件的文本信息（来代替视图提示），可使用
[android:contentDescription](#)属性。在这个属性中你提供的文本在屏幕上是不可见

在本文档中

标签用户界面元素

启用焦点导航

- 启用视图焦点
- 焦点控制命令

创建可访问自定义视图

- 处理定向控制器的点击
- 实现可访问的API函数方
法
- 发送可访问性的事件
- 填充可访问性的事件

可访问性测试

- 声音反馈测试
- 焦点导航测试

的，但如果用户启用了无障碍服务，提供声音提示，那么属性中的描述就能够朗读给用户。

在你的应用程序的用户界面中为每个 `ImageButton`, `ImageView`, `EditText`, `CheckBox` 控件设置[android:contentDescription](#) 属性，和任何其他的输入控件，可能需要其他信息的用户无法看到该控件。

例如，下面的`ImageButton`设置的`add_note`字符串资源，可以作为一个英文界面的“添加注释”定义为加号按钮，代码内容描述：

```
<font color="purple"><ImageButton</font>
    android:id=<font color="green">"@+id/add_note_button"</font>
    android:src=<font color="green">"@drawable/add_note"</font>
    android:contentDescription=<font
color="green">"@string/add_note"/></font>
| style=" width:250px;text-align:center; " |
| }
```

基于语音的辅助服务，包括描述，当用户将焦点移动到这个按钮或在按钮上悬停，可以表明“添加注释”的含义。

注意：对于文本框胡字段,提供一个[android:hint](#)属性,以帮助用户了解哪些内容是需要的。

启用焦点导航

焦点导航，是让残疾用户通过用户界面使用定向控制器控制。定向控制器可以是物理的，如点击轨迹球，方向键（D垫）或箭头键，`tab`键，通过带有一个附加的键盘键导航或软件应用程序，如由眼睛来控制的键盘，来提供了一个屏幕上的方向控制。

定向控制器是许多用户的导航的主要手段。验证所有的用户界面（UI）控制在您的应用程序访问，并且不使用触摸屏而点击中心按钮定向控制器（或“确定”按钮）相比在触摸屏上触摸控制，一样具有相同的效果。有关测试定向控制的信息，请参阅[\[焦点导航测试\]](#)。

启用视图焦点

用户界面元素使用[android:focusable](#)属性设置为true的定向控制器，此时用户界面元素是访问。此设置允许用户集中使用定向控制的元素，然后与它进行交互。Android框架提供的用户界面控件默认情况下是可定焦的和通过改变控件的外观可直观表明其焦点。

Android提供了几个API，让你配置用户界面控件是否可定焦，甚至可要求控件给予焦点：

- [setFocusable\(\)](#)
- [isFocusable\(\)](#)
- [requestFocus\(\)](#)

在开发时有时候是默认情况下不聚焦的，你可以通过设置XML布局文件中的[android:focusable](#) 属性为true或使用[setFocusable \(\)](#) 方法 来获取控件的焦点。

焦点控制命令

当用户导航中使用任何定向控制器，焦点是通过从一个用户界面元素（视图）传递到另一个用户界面元素，这由焦点命令来确定其指向。焦点移动命令是根据一种算法，意在在某一特定方向上寻找相邻的元素。在极少数情况下，默认的算法可能不匹配具体的命令，从而不符合你的用户界面。在这些情况下，可以提供明确的命令来覆盖具体布局文件中的命令，可以使用下列的XML属性：

[android:nextFocusDown](#)

当用户导航为向下时定义下一个视图接收焦点。

[android:nextFocusLeft](#)

当用户导航为向左时定义下一个视图接收焦点

[android:nextFocusRight](#)

当用户导航为向右时定义下一个视图接收焦点。

[android:nextFocusUp](#)

当用户导航为向上时定义下一个视图接收焦点。

下面的示例XML布局展示了两个可定焦的用户界面元素, `android:nextFocusDown` 和 `android:nextFocusUp` 属性被显式地设置。 `TextView` 位于右边的 `EditText`。然而, 通过这些属性设置, `TextView` 元素现在可以做到当按向下箭头时当前焦点是 `EditText` 元素:

```
<LinearLayout android:orientation="horizontal"
    ...
    <EditText android:id="@+id/edit"
        android:nextFocusDown="@+id/text"
        ...
    />
    <TextView android:id="@+id/text"
        android:focusable="true"
        android:text="Hello, I am a focusable TextView"
        android:nextFocusUp="@+id/edit"
        ...
    />
</LinearLayout>
```

修改焦点命令时, 确保从每个用户界面控件的所有方向上和当向后退方向导航时(返回到原来的界面), 程序都能如预计一样运行。

注意: 您在用户界面组件运行时修改焦点命令, 可使用方法
如[setNextFocusDownId\(\)](#) 和 [setNextFocusRightId\(\)](#)。

创建可访问自定义视图

如果您的应用程序需要使用一个定制的视图组件, 你必须做一些额外的工作来确保您的自定义视图是可访问的。为确保您的视图的可访问性有以下几个主要要点:

- 处理定向控制器的点击
- 实现可访问的API函数方法
- 发送特定于您的自定义视图的 [AccessibilityEvent](#) 对象
- 使用[AccessibilityEvent](#) 和 [AccessibilityNodeInfo](#) 来设置你的视图

处理定向控制器的点击

在多数的设备, 单击视图利用定向控制器发送一个带有[KEYCODE_DPAD_CENTER](#) 的按键事件到当前具有焦点的视图。所有标准

的Android的视图已经适当地处理[KEYCODE_DPAD_CENTER](#)。当构建一个定制的视图控,确保这个事件的产生的效果跟触摸触摸屏上的视图的效果一样。

你的自定义控件处理[KEYCODE_ENTER](#)事件时应该和处理[KEYCODE_DPAD_CENTER](#)一样。这种方法更便于与一个使用全键盘的用户进行交互操作。

实现可访问的**API**函数方法

在您的应用程序中可访问性事件用视觉界面组件来标示用户交互信息。这些信息是由使用在这些事件中的信息产生反馈,并当用户启用了可访问性服务时产生补充提示的可访问性服务来处理的。作为Android4.0(API级别14)和更高的方法,用于生成可访问性事件都被扩展以提供更详细的资料,超越了在Android 1.6(API级别4)中引进的[AccessibilityEventSource](#)接口。扩展可访问性方法属于视图类的一部分以及同样作为[View.AccessibilityDelegate](#)类的一部分。这个方法如下:

[sendAccessibilityEvent\(\)](#)

(API级别4)在一个视图中当用户执行任务时这种方法被调用。事件根据一个用户操作类型进行分类,例如类型视图点击。你通常不需要实现这个方法,除非您正在创建一个自定义的视图。

[sendAccessibilityEventUnchecked\(\)](#)

(API级别4)使用这种方法调用的代码时需要直接控制检查可访问性被启用了设备([AccessibilityManager.isEnabled\(\)](#))。如果你实现这个方法,你必须假定调用方法已经检查了和可访问性已经被启用了,并且结果是true。你通常不需要实现这个方法特定于一个自定义的视图。

[dispatchPopulateAccessibilityEvent\(\)](#)

(API级别4)当您的自定义视图生成一个可访问性的事件时系统会调用这个方法,。作为API级别的14,默认为该视图实现调用[onPopulateAccessibilityEvent\(\)](#)方法,然后对该视图的每个子视图实现[dispatchPopulateAccessibilityEvent\(\)](#)方法。为了在修订的Android 4.0版本之前(API级别14)支持可访问性服务你必须为您的自定义视图覆盖这个方法和使用[getText\(\)](#)来输入描述性文本。

[onPopulateAccessibilityEvent\(\)](#)

(API级别14)此方法设置输出你的视图的AccessibilityEvent文本。这种方法如果是一个视图的子视图也会被调用,并它生成一个可访问性的事件。

注意:在该方法中修改文本之外的附加属性可能会以其他方式覆盖属性的设置。所以,虽然你可以使用此方法 修改可访问性事件的属性,但您应该只限制这些更改只作用于文本内容和仅使用由[onInitializeAccessibilityEvent\(\)](#)方法来修改事件的其他属性。

注意:如果您要求实现这一事件完全重写输出文本却不允许其他部件的布局来修改其内容,那么就不要调用该方法的超类方法来实现在你的代码。

[onInitializeAccessibilityEvent\(\)](#)

(API级别14)系统调用这个方法来获取视图状态的额外信息,除了文本内容。如果您的自定义视图由一个简单的文本框或按钮提供了互动的控制,您应该重写这个方法和用这个方法设置这个视图额外的信息到的事件中,如提供用户交互或反馈的口令字段类型,复选框类型或声明。如果你覆盖这个方法,您必须调用它的超类实现方法,然后只修改那些超类尚未设置的属性。

[onInitializeAccessibilityNodeInfo\(\)](#)

API级别14)这个方法来提供易访问性服务的视图状态的信息。默认的视图实现集和一组标准的视图属性,但是如果您的自定义视图由一个简单的文本框或按钮来提供了互动的控制,您应该重写这个方法和由该方法设置你的视图额外的信息到的AccessibilityNodeInfo对象中。

[android.view.accessibility.AccessibilityEvent\) onRequestSendAccessibilityEvent\(\)](#)

(API级别14)当一个视图的子视图生成AccessibilityEvent时系统调用这个方法。这个步骤允许父视图修改可访问性的事件和其他信息。只是如果您的自定义视图有子视图和如果父视图可提供上下文信息到可访问性的事件那么你应该实现这个方法,这样做的可访问性服务就很有用的。

为了在一个自定义视图中支持这些易访问性方法,您应该采取下列的一种方法:

- 如果你的应用程序目标安卓4.0(**API级别14**)或更高的系统中,就直接在您的自

定义视图类中重写并实现上面列出的可访问性方法。

- 如果您的自定义视图的目的就是要兼容安卓1.6(API级别4)及以上,在你的项目中添加[Android支持库](#),版本5或更高,。然后,在您的自定义视图类,调用[android.support.v4.view.AccessibilityDelegateCompat](#))
[ViewCompat.setAccessibilityDelegate\(\)](#)方法来实现可访问性上面的方法。对于这种方法的一个示例,请参阅Android支持库(版本5或更高)[AccessibilityDelegateSupportActivity](#)例子
 在(<sdk>/extras/android/support/v4/samples/Support4Demos/)

在任何一种情况下,为您的自定义视图类您应该执行下面的可访问性方法:

- [dispatchPopulateAccessibilityEvent\(\)](#)
- [onPopulateAccessibilityEvent\(\)](#)
- [onInitializeAccessibilityEvent\(\)](#)
- [onInitializeAccessibilityNodeInfo\(\)](#)

更多信息实现这些方法,请参阅 [Populating Accessibility Events](#)。

发送可访问性的事件

根据您的自定义视图的特性,它可能在不同时间或事件需要发送AccessibilityEvent对象而不是由默认来实现。视图类提供了一个默认方法来实现这些事件类型:

- 从API级别4:
 - TYPE_VIEW_CLICKED
 - TYPE_VIEW_LONG_CLICKED
 - TYPE_VIEW_FOCUSED
- 从API级别4:
 - TYPE_VIEW_SCROLLED
 - TYPE_VIEW_HOVER_ENTER
 - TYPE_VIEW_HOVER_EXIT

注意:Hover事件与通过触摸功能的探索关联,利用这些事件作为触发器伟输入用户界面元素提供声音提示。

一般来说,每当你自定义视图的内容有变化时你应该发送一个**AccessibilityEvent**事件。例如,如果您正在实现一个自定义的滑动条,可以让用户选择一个数字值按下左边或者右边的箭头,这时您的自定义视图应该发出一个[TYPE VIEW_TEXT_CHANGED](#)类型的事件来查看滑块值是否发生变化。下面的示例代码演示了使用[sendAccessibilityEvent\(\)](#)方法来说明这个事件:

```
@Override
public boolean onKeyDown ( int keyCode, KeyEvent event ) {
    if ( keyCode == KeyEvent.KEYCODE_DPAD_LEFT ) {
        mCurrentValue--;
        sendAccessibilityEvent ( AccessibilityEvent.TYPE_VIEW_TEXT_CHANGED );
        return true;
    }
    ...
}
```

填充可访问性的事件

每个[AccessibilityEvent](#)有描述当前状态的视图的一组必需的属性。这些属性包括诸如视图类名称、内容描述和检查状态。对于每个事件类型的特定的性能要求,都在[AccessibilityEvent](#)参考文档中进行了描述。为视图实现提供了一些默认的属性值。在这些属性值中,包括自动提供的类名和事件的时间戳。如果你正在创建一个自定义视图组件,您必须提供一些视图有关的信息内容和特点。这些信息可能是简单的按钮的标签,而且您想要添加到事件中的信息可能还包括其他的状态信息。

为一个带自定义视图的可访问性服务提供信息的最低要求是实现[dispatchPopulateAccessibilityEvent\(\)](#)方法。系统为一个[AccessibilityEvent](#)调用这个方法来请求信息,使您的自定义视图兼容Android 1.6系统(API级别4)和更高的可访问性服务。下面的示例代码展示了该方法的一个基本的实现。

```
@Override
public void dispatchPopulateAccessibilityEvent ( AccessibilityEvent event )
{
    super.dispatchPopulateAccessibilityEvent ( event );
    // Call the super implementation to populate its text to the event,
    // which
    // calls onPopulateAccessibilityEvent() on API Level 14 and up.

    // In case this is running on a API revision earlier than 14, check
    // the text content of the event and add an appropriate text
    // description for this custom view:
    CharSequence text = getText ();
    if ( !TextUtils.isEmpty ( text ) ) {
        event.getText ().add ( text );
    }
}
```

Android 4.0系统(API级别14)和较高,推荐使用**onPopulateAccessibilityEvent()**和**onInitializeAccessibilityEvent()**方法来填充或修改一个**AccessibilityEvent**事件中的信息。专门使用**onPopulateAccessibilityEvent()**方法用来为该事件添加或修改文本的内容,这是通过可访问性服务变转换成声音提示,比如反馈。使用**onInitializeAccessibilityEvent()**方法来填充关于事件的其他信息,比如视图的选择状态。

此外,您还应该实现**onInitializeAccessibilityNodeInfo()**方法。通过这个方法来填充**AccessibilityNodeInfo**对象,易访问性服务根据使用此对象来分析视图的层次,接收了这一事件后生成一个可访问性事件,获得一个更详细的上下文信息,并提供适当的反馈给用户。

下面的示例代码显示了如何通过使用[android.support.v4.view.AccessibilityDelegateCompat](#)
[ViewCompat.setAccessibilityDelegate\(\)](#)重写这三种方法。注意,此示例代码要求添加API级别4的Android支持库(版本5或更高)到您的项目。

```
ViewCompat.setAccessibilityDelegate(new AccessibilityDelegateCompat() {
    @Override
    public void onPopulateAccessibilityEvent(View host, AccessibilityEvent event) {
        super.onPopulateAccessibilityEvent(host, event);
        // We call the super implementation to populate its text for the
        // event. Then we add our text not present in a super class.
        // Very often you only need to add the text for the custom view.
        CharSequence text = getText();
        if (!TextUtils.isEmpty(text)) {
            event.getText().add(text);
        }
    }
    @Override
    public void onInitializeAccessibilityEvent(View host,
        AccessibilityEvent event) {
        super.onInitializeAccessibilityEvent(host, event);
        // We call the super implementation to let super classes
        // set appropriate event properties. Then we add the new property
        // (checked) which is not supported by a super class.
        event.setChecked(isChecked());
    }
    @Override
    public void onInitializeAccessibilityNodeInfo(View host,
        AccessibilityNodeInfoCompat info) {
        super.onInitializeAccessibilityNodeInfo(host, info);
        // We call the super implementation to let super classes set
        // appropriate info properties. Then we add our properties
        // (checkable and checked) which are not supported by a super
        class.
        info.setCheckable(true);
        info.setChecked(isChecked());
        // Quite often you only need to add the text for the custom view.
        CharSequence text = getText();
    }
})
```

```

    if ( !TextUtils.isEmpty( text ) ) {
        info.setText( text );
    }
}

```

针对在Android 4.0(API级别14)和更高上的应用程序,这些方法可以直接在您的自定义视图类中实现。这种方法的另一个例子,请参阅Android支持库(版本5或更高)的示例, [AccessibilityDelegateSupportActivity](#)样本在(`<sdk>/extras/android/support/v4/samples/Support4Demos/`)。

注意:编写Android 4.0之前您可能会发现实现可访问性信息的自定义视图,它描述了使用`dispatchPopulateAccessibilityEvent()`方法`AccessibilityEvents`填充。作为Android 4.0的发布,然而,推荐的方法是使用`onPopulateAccessibilityEvent()`和`onInitializeAccessibilityEvent()`方法。

提供自定义辅助内容

安卓4.0 (API级别14) 框架得到改善, 可以访问服务来检查用户界面组件的层次结构视图。此增强功能允许访问服务来提供更丰富的上下文信息。还有一些情况, 可访问服务不能从视图层次获取充分信息。例如具有两个或者更多独立点击区域的自定义界面控件, 比如日历控件。这种情况下服务不能获取足够信息, 因为可点击分段不是视图层次结构的一部分。



在图1所示例子中, 整个日历作为单一视图实现。因此如果不采取其他措施, 辅助服务就不能获取视图内容和用户选择的足够信息。例如, 如果用户点击了包含17的日期, 访问框架只接收日历控件的描述信息。这种情况TalkBack辅助服务会简单宣布“日历”或者稍好一些“五月日历”, 用户不禁疑惑选择了哪天。

为了提供辅助服务的足够内容信息, 框架提供了一种方法来指定虚拟视图层次结构。虚拟视图层次结构为应用程序开发者提供了辅助服务的互补视图层次结构, 这样会更紧密地匹配屏幕上的实际信息。这种方法使辅助服务提供了更多有用的内容信息。

另一种需要虚拟视图层次结构的情况就是, 用户界面包含一组与功能密切相关的控件(视图), 控件的操作会影响一个或多个元素的内容。这种情况下辅助服务无法获得足够信息, 因为控件操作改变了另一控件的内容, 而这些控件的关系可

能并不明显。为了处理这种情况，把相关控件通过视图组合起来，并且提供虚拟视图层次结构来清楚代表控件信息和操作。

为了提供虚拟视图层次结构，在自定义视图或者视图组改写getAccessibilityNodeProvider()方法，并返回AccessibilityNodeProvider实现。使用ViewCompat.getAccessibilityNodeProvider()支持库，并提供AccessibilityNodeProviderCompat实现，就可以实现与安卓1.6及更高版本兼容的虚拟视图层次结构。

处理自定义触摸事件

自定义视图控件可能需要非标准的触摸行为。例如自定义控件可以使用onTouchEvent(MotionEvent)监听方式来检测 ACTION_DOWN 和ACTION_UP事件，并触发一个特殊事件。为了维持辅助服务的兼容性，处理自定义click事件的代码必须满足以下条件：

1. 为解释点击操作生成适当的AccessibilityEvent。
2. 启用辅助服务来为不能使用触摸屏的用户执行自定义单击操作。

要有效处理这些需求，代码应该重写performClick()方法。当检测到自定义点击操作时，代码应该调用performClick()。下列代码示例显示了这一模式：

```
class CustomTouchView extends View {
    public CustomTouchView(Context context) {
        super(context);
    }

    boolean mDownTouch = false;

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        super.onTouchEvent(event);

        // Listening for the down and up touch events
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                mDownTouch = true;
                return true;

            case MotionEvent.ACTION_UP:
                if (mDownTouch) {
                    mDownTouch = false;
                    performClick(); // Call this method to handle the
response, and
services to
cannot
                }
        }
    }

    protected void performClick() {
        // thereby enable accessibility
        // perform this action for a user who
        // click the touchscreen.
    }
}
```

```
        }
    }

    return false; // Return false for other touch events
}

@Override
public boolean performClick() {
    // Calls the super implementation, which generates an
    // AccessibilityEvent
    // and calls the onClick() listener on the view, if any
    super.performClick();

    // Handle the action for the custom click here

    return true;
}
}
```

来自“[index.php?title=Making_Applications_Accessible&oldid=13862](#)”

Building Accessibility Services

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/topics/ui/accessibility/services.html#manifest>

编辑者：IBrave

目录

[[隐藏](#)]

[1 Building Accessibility Services\(建立可访问性服务\)](#)

- [1.1 Manifest声明和权限](#)
 - [1.1.1 可访问性服务声明](#)
 - [1.1.2 可访问性服务配置\(configuration\)](#)
- [1.2 注册辅助活动](#)
- [1.3 AccessibilityService 方法](#)
- [1.4 获得事件的详细信息](#)
- [1.5 采取用户操作](#)
 - [1.5.1 监听手势](#)
 - [1.5.2 使用辅助操作](#)
 - [1.5.3 使用焦点类型](#)

[2 示例代码](#)

Building Accessibility Services(建立可访问性服务)

一个可访问性服务，是一个为增强用户界面并帮助残疾用户的应用程序，或者用户可能无法完全与设备的交互。例如：用户正在开车、照顾一个小孩或者参加一个非常吵闹的聚会，那么用户就有可能需要添加额外的或者可替代的用户反馈方式。

主题

Manifest声明和权限

- [可访问性服务声明](#)

- 可访问性服务配置

AccessibilityService方法

获得事件细节

示例代码

主要的类

AccessibilityService

AccessibilityServiceInfo

AccessibilityEvent

AccessibilityRecord

AccessibilityNodeInfo

同时要看

Implementing Accessibility

Android提供了标准的可访问性服务，包括反馈，还有开发者可创建和发布他们自己的服务。这个文档解释了建立一个可访问性服务的基础知识。

这种构建和部署可访问性服务的技能被引入**Android 1.6 (API 级别 4)**并且在**Android 4.0 (API 级别 14)**中得到显著的改进。这个**Android**支持库伴随着**Android 4.0**的发布也更新到可以提供支持以前**Android 1.6**的增强可访问性的特性。开发者的目標是广泛兼容可访问性的服务被鼓励使用在支持库中和在**Android 4.0**介绍更先进的可访问特性的开发。

Manifest声明和权限

提供可访问性的应用程序，为了被**Android**系统当作一个可访问性服务，则必须在他们的应用程序**manifests**文件中包含特殊的声明。这段解释了所需和可选的可访问性服务。

可访问性服务声明

为了应用程序具有可访问性服务，应用程序必须在其**manifest**文件中的**application**单元(element)中包含**service**单元(而不是**activity**单元)。除此之外，在**service**单元中也必须包括一个可访问性服务意图的筛选程序，如下例所示：

```
<application>
    <service <font color="purple">android:name=</font><font
color="green">".MyAccessibilityService" </font>
        <font color="purple">android:label=</font><font
color="green">"@string/accessibility_service_label" </font>>
        <intent-filter>
            <action <font color="purple">android:name=</font><font
color="green">"android.accessibilityservice.AccessibilityService" </font> />
        </intent-filter>
    </service>
</application>
```

在**Android 1.6 (API 等级 4)**或者更高级中部署所有可访问性的服务都需要这些声明。

可访问性服务配置(**configuration**)

可访问性服务还必须提供一个，指定能够处理和添加额外信息服务的可访问事件类型的配置。这个可访问性服务的配置包含在**AccessibilityServiceInfo**类中。在运行的时候，你的服务可以用这个类的接口和**setServiceInfo()**建立并设置一个配置。然而，用这种方法不是所有的配置选项是有用的。

从Android4.0开始，你可以在你的**manifest**文件中包含一个引用配置文件的**<meta-data>**，这样才允许你设置所有你的可访问性服务选项的完整范围，如下例所示：

```
<service android:name=".MyAccessibilityService">
    ...
    <meta-data
        android:name="android.accessibilityservice"
        android:resource="@xml/accessibility_service_config" />
</service>
```

meta-data单元引用的一个**XML**文件，这个文件应该创建在应用程序的**resource**路径(**<project_dir>/res/xml/accessibility_service_config.xml**)下面代码 展示的是服务配置文件内容的示例：

```
<accessibility-service
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/accessibility_service_description"
    android:packageName="com.example.android.apis"
    android:accessibilityEventTypes="typeAllMask"
    android:accessibilityFlags="flagDefault"
    android:accessibilityFeedbackType="feedbackSpoken"
    android:notificationTimeout="100"
    android:canRetrieveWindowContent="true"

    android:settingsActivity="com.example.android.accessibility.ServiceSettingsActivity"
/>
```

更多关于可以被用在可访问性服务配置文件的**XML**属性信息，可以根据下面这些链接，参考相关文档。

- [android:description](#)
- [android:packageName](#)
- [android:accessibilityEventTypes](#)
- [android:accessibilityFlags](#)
- [android:accessibilityFeedbackType](#)
- [android:notificationTimeout](#)
- [android:canRetrieveWindowContent](#)
- [android:settingsActivity](#)

更多的关于可以在运行时动态地设置配置的设置信息，可以参考[AccessibilityServiceInfo](#)文档。

注册辅助活动

辅助服务的配置参数中最重要的功能之一是允许你指定服务可以处理哪些类型的辅助事件。指定该信息可以让辅助活动相互合作，并且允许开发人员来灵活处理具体应用的特定事件类型。事件过滤会包含以下标准：

Package names-希望服务来处理辅助事件，从而指定相应应用程序的包名。如果忽略该参数，辅助服务适用于任何应用程序的服务辅助事件。这一参数会在辅助服务配置文件中设置，可以使用`android:packageNames`属性作为逗号分隔列表，或者使用`AccessibilityServiceInfo.packageNames`成员设置。

事件类型-指定所处理辅助事件的类型。此参数可以在辅助服务配置文件中设置，使用`android:accessibilityEventTypes`属性作为|字符分隔的列表（比如`accessibilityEventTypes="typeViewClicked|typeViewFocused"`），或者使用`AccessibilityServiceInfo.eventTypes`成员进行设置。

设置辅助服务时，请认真考虑服务可以处理的事件然后进行注册。由于用户可以同时激活一个以上的辅助服务，服务不能占用未处理的事件。请记住其他服务可以处理这些事件，以便改善用户体验。

AccessibilityService 方法

一个提供可访问性服务的应用程序必须继承`AccessibilityService`类，并且重写这个类的方法。这些方法是按从服务程序开始(`onServiceConnected()`)的时候这些方法(`onAccessibilityEvent()`, `onInterrupt()`)就一直运行到服务关掉(`onUnbind()`)的Android系统调用的顺序呈现的。

- `onServiceConnected()`--(可选的)当这个方法成功连接到你的可访问性服务，系统将调用这个方法。用这个方法可以一次性的设置你的服务，包括连接到用户反馈系统服务，例如音频的管理或者设备振动器。如果你想在运行的时候设置你的服务配置或者一次性的调整，这个是很方便定位哪个服务系统调用`setServiceInfo()`。
- `onAccessibilityEvent()`--(必须有)当这个方法检测到一个与你可访问性服务指定的事件过滤参数相匹配的可访问性事件(`AccessibilityEvent`)系统将调用这个方法回应。例如:在一个应用程序中，当用户点击按钮或者一个用户界面启动，你的哪个可访问性服务将提供反馈。当这种情况发生，系统将调用与`AccessibilityEvent`相关的服务方法，这样你才可以做出响应(`interpret`)并给用户提供反馈。在你的服务生命周期期间，这个方法可以被多次调用。
- `onInterrupt()`--(必须有)当系统想中断你的服务系统提供的反馈，这个方法就会被调用。通常是对用户采取行动做出响应，例如:移动焦点到一个不同的用户控制界面而不是你当前提供反馈的那个界面。在你的服务生命周期期间，这个方法可以被多次调用。
- `onUnbind()`--(可选的)当系统想关闭这个可访问性服务，这个方法就会被调用。用这个方法可以一次性的关闭程序，包括取消使用者的反馈系统服务的分配，例如:音频管理或者设备振动器。

这些回调的方法提供了你的可访问性服务的基本构架。由你决定怎样处理Android系统以`AccessibilityEvent`对象形式提供的数据和怎样提供给用户反馈。

获得事件的详细信息

Android系统通过[AccessibilityEvent](#)对象把关于用户界面交互的信息 提供给可访问性服务。在Android4.0之前，这些有用的信息在可访问性事件中，但另一方面提供了大量有用的有关用户可选的用户界面控件的详细信息，典型的是提供了有限的上下关联信息。在许多情况下，这些缺失的前后关联的信息，对理解这些可选管理控件的含义有可能是很重要的。在一个界面中，上下关联是至关重要的一个典型的例子是日历或日程计划。如果一个用户选择了周一到周五的“下午4点”时间列表，并且可访问性服务将通知“下午4点”，但是没有明确这是某个月的哪个周五和周一，很难将理想的信息反馈给用户。在这种情况下，对于想安排一次会议的人，这个用户界面控制的上下关联是 至关重要的。

Android4.0显著地拓展了一个可访问性服务能获得有关通过基于视图底层的可访问性服务的用户界面交互的大量信息。视图分层结构可由 包含组件(它的父类)的用户界面组件和可被组件(它的子类)包含的用户界面元素组成。用这种方式，Android系统可以提供更多有关允许可访问性服务提 供更多有用反馈给用户的可访问性事件的详细信息。

一个可访问性服务获取有关一个用户接口事件利用[AccessibilityEvent](#)事件通过系统向服务器请求返回一个[onAccessibilityEvent\(\)](#)回 调方法的信息 。这个可访问性服务对象提供了关于事件的详细细节。包括这个类型的对象作用，其描述文本和其他细节。

从Android4.0(并且在支持库中支持以前版本 的accessibilityeventcompat对象)开始，你可以获得关于事件用这些调用的额外信息：

- [AccessibilityEvent.getRecordCount\(\)](#) 和 [getRecord\(int\)](#)--这些方法允许你检索[AccessibilityRecord](#)对象集，这有助于[AccessibilityEvent](#)通过系统传递给你，这样才可以提供更多的有关可访问性服务的上下文。
- [AccessibilityEvent.getSource\(\)](#)--这方法返回一个[AccessibilityNodeInfo](#)对象。这些对象允许你索取来自可访问性事件的父类和子类组件和追查他们的内容和状态以便提供

重点:从[AccessibilityEvent](#)调查这个完整视图层次的能力可能曝光你的可访问性服务的私人用户信息。由于这个原因,你的服务必须请求这种可通过可访问性服务配置XML[service configuration XML](#)文件的访问级别,包括canRetrieveWindowContent属性并设置为true。如果在你的服务配置xml文件中不包括此设置,将不能成功调用[getSource\(\)](#)。

采取用户操作

从安卓4.0 (API级别14) 开始，辅助服务可以代表用户，包含改变输入焦点和选择 (激活) 用户界面元素。在安卓4.1 (API级别16) 操作范 围扩展到包含滑动列表和文本字段交互。辅助活动可以采取全局操作，比如导航至主屏幕，按返回按钮，打开通知屏幕和最近应用程序列表。安卓4.1还包括新型 焦点 Accessibility Focus，这样所有可见元素由辅助服务进行选择。

监听手势

辅助服务可以监听特定手势并代表用户采取操作进行回应。把功能添加到安卓4.1（API级别16），就需要辅助服务通过触碰功能来请求激活搜索。通过把服务AccessibilityServiceInfo示例的标志成员设置为FLAG_REQUEST_TOUCH_EXPLORATION_MODE，如下例所示可以激活服务。

```
public class MyAccessibilityService extends AccessibilityService {
    @Override
    public void onCreate() {
        getServiceInfo().flags =
            AccessibilityServiceInfo.FLAG_REQUEST_TOUCH_EXPLORATION_MODE;
    }
    ...
}
```

一旦服务通过触摸请求激活搜索，用户必须允许启动该功能。当激活该功能时，服务通过onGesture()回调方法来接收辅助手势通知，并且采取操作进行回应。

使用辅助操作

辅助服务可以代表用户采取操作，以便与应用程序进行简单有效的互动。

为了采取代表用户的操作，辅助服务必须注册才能从许多应用程序接收事件，同时把service configuration file中的android:canRetrieveWindowContent设置为true，这样可以获取查看应用程序内容的许可。当服务接收到事件时，就可以使用getSource()检索AccessibilityNodeInfo对象。服务使用AccessibilityNodeInfo对象来探索视图层次结构，这样可以决定采取哪种操作，然后使用performAction()实现操作。

```
public class MyAccessibilityService extends AccessibilityService {
    @Override
    public void onAccessibilityEvent(AccessibilityEvent event) {
        // get the source node of the event
        AccessibilityNodeInfo nodeInfo = event.getSource();

        // Use the event and node information to determine
        // what action to take

        // take action on behalf of the user
        nodeInfo.performAction(AccessibilityNodeInfo.ACTION_SCROLL_FORWARD);

        // recycle the nodeInfo object
        nodeInfo.recycle();
    }
    ...
}
```

performAction()方法允许服务在应用程序中进行操作。如果服务需要执行全局操作，例如导航到主屏幕，按返回按钮，打开通知屏幕或近期应用程序列表，然后使用performGlobalAction()。

使用焦点类型

安卓4.1 (API级别16) 引入了用户界面焦点新类型，称为辅助焦点。辅助服务可以使用此焦点类型来选择任意可见用户界面元素并进行操作。这种焦点方式不同于著名的输入焦点，当用户输入字符，按 Enter键或者推D-pad控件的中央按钮时，输入焦点会确定屏幕用户界面元素的接收输入。

辅助焦点是完全分离且独立于输入焦点的。实际上用户界面的元素可能具有输入焦点，而另一个元素具有辅助对焦。辅助焦点的目的是为屏幕上任意可见元素交互方法提供辅助服务，而不管从系统角度来看元素是否是输入可聚焦。可以通过测试辅助手势来参阅辅助焦点。欲了解测试该功能的更多信息，请参阅测试手势导航。

示例代码

API演示项目包含两个例子，这两个例子可以作为生成可访问性服务的起点。[\(<sdk>/samples/<platform>/ApiDemos/src/com/example/android/apis/accessibility\)](<sdk>/samples/<platform>/ApiDemos/src/com/example/android/apis/accessibility):

- [ClockBackService](#)——这个服务是基于原始[AccessibilityService](#)的实现和可以被用来作为发展基础的可访问性服务的基础,兼容安卓1.6(API级别4)和更高的。
- [TaskBackService](#)——这个服务是基于增强可访问性APIs，在Android 4.0(API级别14)中介绍的APIs。不过,您可以使用Android支持库([Support Library](#)) 中等效的支持包类来代替在最新的API级别中介绍的类 (例如, [AccessibilityRecord](#),[AccessibilityNodeInfo](#))。用等效的支持包类 (例如, [AccessibilityRecordCompat](#),[AccessibilityNodeInfoCompat](#)) 可使这个示例处理API版本兼容Android1.6(API级别4)。

来自 "[index.php?title=Building_Accessibility_Services&oldid=13865](#)"



Styles and Themes

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：
址：

<http://developer.android.com/guide/topics/ui/themes.html#DefiningStyles>

翻译： [红色沙漠](#)

更新： 2012.6.12

目录

[[隐藏](#)]

[1 样式和主题 - Styles and Themes](#)

- [1.1 定义样式 - Defining Styles](#)
 - [1.1.1 继承 - Inheritance](#)
 - [1.1.2 样式属性 - Style Properties](#)
- [1.2 样式和主题应用到UI - Applying Styles and Themes to the UI](#)
 - [1.2.1 为视图套用样式 - Apply a style to a View](#)
 - [1.2.2 在Activity或应用程序中应用主题 - Apply a theme to an Activity or application](#)
 - [1.2.3 选择基于平台版本的主题 - Select a theme based on platform version](#)
- [1.3 使用平台的样式和主题 - Using Platform Styles and Themes](#)

样式和主题 - Styles and Themes

style是用于指定[View](#)或window的外观和格式的一系列属性的集合。style可以指定高（height）、填补（padding）、字体颜色、字体大小、背景颜色等等属性。style定义在不同于用来设置布局的XML资源中。

Android中的Styles与网页设计中的层叠样式表有着相似的原理——允许你将设计从内容中分离出来。

例如，使用一个style，你可以将下面这个布局：

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

变成这样：

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

所有与style相关的属性从XML布局中移出，放到一个名为CodeFont的style定义中，通过style属性应用。你将在以下章节中看到此类style的定义。

theme是一个应用于整个Activity或应用中，而不是某一个单独的View（正如上面的例子）。当一个style被作为theme来应用时，Activity或应用中的每个View都将应用支持的每个style属性。例如，你能把CodeFont style作为theme应用于一个Activity，那么这个Activity中所有文本都将是绿色等宽字体。

定义样式 - Defining Styles

创建一套style,需保存一个XML文件到你的工程的res/values/ 目录下。这个XML文件的名称可以随便定义，但必须使用.xml作为后缀，且要保存在res/values/ 文件夹中。

这个XML文件的根节点必须是<resources>。

为每个要创建的style，添加一个用来唯一标识此style的name的<style>元素到文件中（这个属性是必需的）。然后为style的每个属性添加一个<item> 元素，其包含一个声明style属性的name 和一个使用的值（这个属性是必需的）。这个<item> 的值可以是一个关键字符串、十六进制颜色、到另一个资源类型的引用或其他值，取决于style的属性。这里有一个单独style的例子：

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont"
parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item
name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>

```

每个`<resources>`元素的子节点在编译时都被转换为一个应用程序资源对象，其可通过`<style>`元素的`name`属性的值来引用。这个示例中`style`可以通过`@style/CodeFont`在一个XML布局中引用（正如上面的介绍）。

在`<style>`元素中的`parent`属性是可选的，用来指定另一个`style`资源的ID，前者继承后者的所有属性。你可以复写继承的`style`属性，如果你想要那样做。

记住，你想要用作一个Activity或应用theme的`style`，与在XML中定义一个View的`style`方法是一样的。一个如同上面那样定义的`style`可以应用于一个View的`style`，或是整个Activity或应用的theme。稍后讨论如何将一个`style`应用于一个View或一个应用theme中。

继承 - Inheritance

`<style>`元素的`parent`属性让你能够从指定的`style`中继承属性。你可以通过这种途径从一个现有的`style`中继承属性，然后定义你想改变或添加的属性。你可以从你自己创建的`style`或平台内创建的`style`中继承。（参阅[Using Platform Styles and Themes](#)，以获取关于继承Android平台预定义的`style`的信息。）例如，你可以继承Android平台默认文本外观并修改：

```

<style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>

```

如果你想要继承你自己定义的`style`，你不必使用`parent`属性，而是将你想通过继承创建的新`style`的`name`前加上要继承的`style`的`name`，使用一个句点。例如，创建一个继承前面定义的`CodeFont`的`style`，把颜色改为红色，你可以像这样编写新的`style`：

```
<style name="CodeFont.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

注意在`<style>` 标签中没有`parent` 属性，因为`name` 属性以`CodeFont` 起始（你已经创建的`style`），这个`style`继承所有`style`属性。这个`style`复写`android:textColor` 属性将文本设置为红色。你可以通过`@style/CodeFont.Red` 引用这个新`style`。

你可以像这样继续继承很多次，只要修改句点之前的名称。例如，你可以扩展`CodeFont.Red` 使字体变大：

```
<style name="CodeFont.Red.Big">
    <item name="android:textSize">30sp</item>
</style>
```

从`CodeFont` 和`CodeFont.Red` style 中同时继承，然后添加`android:textSize` 属性。

注意: 这种技巧仅适用于将你自己定义的资源链接起来。你不能用这种方式继承Android内建的`style`。要引用一个诸如`TextAppearance`的内建`style`，你必须使用`parent` 属性。

样式属性 - Style Properties

到目前，你已明白了一个`style`是如何定义的，你需要学习由`<item>` 元素定义的哪些属性是可用的。你很可能已经熟悉了某些，比如`layout_width`和`textColor`。当然，有更多的`style`属性供你使用。

找到适用于某个特定`View`的属性的最佳方法是相应的类的参考，其中列出了所有支持的`XML`属性。例如，在表格`TextView XML attributes`中列出的所有属性可以用在`TextView`元素（或它的一个子类）的`style`定义中。其中列出的一个属性是`android:inputType`，那么你通常可能将`android:inputType`属性放置在`<EditText>` 元素中，像这样：

```
<EditText
    android:inputType="number"
    ... />
```

你也可以为包含这个属性的[EditText](#)元素创建一个style：

```
<style name="Numbers">
    <item name="android:inputType">number</item>
    ...
</style>
```

所以你的布局XML现在可以这样实现这个style：

```
<EditText
    style="@style/Numbers"
    ... />
```

这个简单的例子看起来增加了工作量，但当你添加越来越多的style属性并考虑到此style在不同地方的可重用性时，你会发现获益是巨大的。

关于所有可用的style属性，请参见[R.attr](#)。记住所有的View对象并不接受相同的style属性，所以你通常应该参考特定的[View](#)类，查看其所支持的style属性。但是，如果你对一个View应用了style，而其并不支持此style中某些属性，那么此View将应用那些它支持的属性，并简单忽略那些不支持的。

然而一些style属性只能被当作一个theme来应用，而不支持任何View元素。这些style属性应用到整个窗口，而不是任何类型的View。例如那些用于隐藏应用标题、隐藏状态栏或改变窗口背景的style属性。这些style属性不属于任何 View对象。探究这些仅应用作theme的style属性，参见[R.attr](#)中那些以window 开头的属性。举个例子，[windowNoTitle](#) 和[windowBackground](#) 是仅当style作为theme应用于一个Activity或应用时才有效的style属性。参阅下一节，获得关于style应用作theme的信息。

注意:不要忘记对每个<item> 元素中的属性冠以android: 命名空间前缀。例如：<item name="android:inputType"> 。

样式和主题应用到UI - Applying Styles and Themes to the UI

有两种方式来设置style：

- 对一个独立的View，添加style 属性到你的布局XML中的View元素中。
- 或者，对一个Activity或应用添加android:theme 属性到Android manifest的<activity> 或<application> 元素中。

当你应用一个style到布局中一个单独的View上，由此style定义的属性会仅应用于那个View。如果一个style应用到一个ViewGroup上，那么子View元素并不会继承应用此style属性——只有你直接应用了style的元素才会应用其属性。然而，你可以通过将其作为theme来应用的方式应用一个style到所有View元素上。

将一个style作为一个theme来应用，你必须在Android manifest中将其应用到一个Activity或应用中。当你这样做，此Activity或应用中的每个View都将应用其所支持的属性。例如，如果你应用前面示例中的CodeFont style到一个Activity，那么支持此文本style属性的所有View元素都将应用它们。所有View所不支持的属性都会被忽略。如果一个View仅支持某些属性，那么它就只应用那些属性。

为视图套用样式 - Apply a style to a View

下面是如何在XML布局中为View设置style的方法：

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

现在这个TextView将应用名为CodeFont 的style所定义的属性。（参阅前面在[Defining Styles](#)中的示例）。

注意:style属性不能使用android: 命名空间前缀。

在Activity或应用程序中应用主题 - Apply a theme to an Activity or application

对你的应用程序中所有activity设置一个theme，打开AndroidManifest.xml 文件并编辑<application> 标签，使之包含android:theme 属性和style名称。例如：

```
<application android:theme="@style/CustomTheme">
```

如果你希望theme仅应用到你的应用程序中的某个Activity中，那么就将`android:theme` 属性添加到`<activity>` 标签里。

正如Android提供的其他内建资源一样，有许多你可以使用的预定义theme，而不用自己编写它们。例如，你可以使用Dialog theme使你的Activity看起来像一个对话框：

```
<activity android:theme="@android:style/Theme.Dialog">
```

或者你想让背景变成透明的，那就使用透明theme：

```
<activity android:theme="@android:style/Theme.Translucent">
```

如果你喜欢一个theme，但又想调整它，那么你可以将其作为你的自定义theme的parent。例如，你可以像这样修改传统的light theme来使用你自己定义的颜色：

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
    <item
        name="android:windowBackground">@color/custom_theme_color</item>
    <item
        name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

(注意，这里颜色需要作为单独的资源提供，因为`android:windowBackground` 属性只支持到另一个资源的引用，不像`android:colorBackground`，它不能得到一种文本颜色。) 然后在Android Manifest中使用CustomTheme 代替Theme.Light：

```
<activity android:theme="@style/CustomTheme">
```

选择基于平台版本的主题 - Select a theme based on platform version

新版本的Android应用程序提供额外的theme，你可能想使用它们在这些平台上运行，同时与旧版本兼容。你可以通过使用自定义theme资源选择不同的parent theme，根据平台版本之间切换完成。

例如，这里声明一个自定义theme，相当于是标准平台上默认的light theme。它将在XML文件的`res/values` 目录下（通常是`res/values/styles.xml`）：

```
<style name="LightThemeSelector" parent="android:Theme.Light">
    ...
```

</style>

当程序运行在Android3.0 (API等级11) 或更高的版本时使用新的 theme，你可以在`res/values-v11` 的XML文件中放置另一个声明theme，但 theme的parent theme像这样设置：

```
<style name="LightThemeSelector" parent="android:Theme.Holo.Light">
    ...
</style>
```

现在可以如其他的theme那样使用这个theme，如果你的应用程序运行在Android3.0或更高的版本时，将自动切换到 theme。

你可以在R.styleable.Theme中找到你能够使用的theme的标准属性列表。

获得更多关于如theme和layout提供替代资源，基础平台版本或其他设备配置的详细信息，请参阅Providing Resources文档。

使用平台的样式和主题 - **Using Platform Styles and Themes**

Android平台提供了大量的style和theme供你在应用程序中使用。你可以在R.style类中找到所有可用的style。要使用这些style，用句点替换style名称中的下划线。例如，你可以通过"`@android:style/Theme.NoTitleBar`"应用Theme_NoTitleBar theme。

然而，R.style没有好的文档，没有详细叙述这些style，所以查看这些style和theme的实际源代码将使你更好理解每个style属性提供了什么功能。为更好参考Android的style和theme，请参阅下列源代码：

- [Android Styles \(styles.xml\)](#)
- [Android Themes \(themes.xml\)](#)

这些文件将通过例子帮助你学习。举个例子，在Android theme源代码中，你将会找到一个`<style name="Theme.Dialog">`声明。在这个定义中，你将看到所有由Android框架使用的用于对话框的style属性。

为获得更多关于在XML中创建style的语法，参阅[Available Resource Types:Style and Themes](#)。

关于你可以用来定义style或theme的可用style属性（例如，“`windowBackground`”或“`textAppearance`”），参阅R.attr或者对应于你正在为其创建一个style的View类。

←返回 [用户界面 - User Interface](#)

来自“[index.php?title=Styles_and_Themes&oldid=8858](#)”

1个分类: [Android Dev Guide](#)



Custom Components

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： GloriousOnion

主任务原文链接：<http://docs.eoeandroid.com/guide/topics/ui/custom-components.html>

目录

[[隐藏](#)]

[1 定制组件](#)

- [1.1 基本方法](#)
- [1.2 完全定制的组件](#)
- [1.3 复合控件](#)
- [1.4 修改已有视图 \(View\)](#)

定制组件

Android平台提供了一套完备的、功能强大的组件化模型用于搭建用户界面，这套组件化模型以[View](#)和[ViewGroup](#)这两个基础布局类为基础。平台本身已预先实现了多种用于构建界面的View子类和ViewGroup子类，他们被分别称为部件 (widget) 和布局 (layout)。

部件 (widget) 包

括[Button](#)、[TextView](#)、[EditText](#)、[ListView](#)、[CheckBox](#)、[RadioButton](#)、[Gallery](#)、[Spinner](#)等这些常用部件以及有着专门用途的[AutoCompleteTextView](#)、[ImageSwitcher](#)和[TextSwitcher](#)。

布局 (layout) 包括：[LinearLayout](#)、[FrameLayout](#)、[RelativeLayout](#)等，如果需要查看更多实例，请查看[通用部件对象](#)。

如果这些已有部件和布局不能满足需求，您可以按需实现View子类。如果对已有的部件和布局进行小调整就能满足需求，可以通过继承部件或布局并重载特定方法的方式轻松实现。

开发者通过构造View子类可以对屏幕元素的样式及功能进行精确控制。为了使您对定制View的可控性有一个直观了解，下面给出可以对定制View进行操作的几个实例：

- 您可以将View定制成特别样式，比如一个使用2D图片渲染的音量调节器可以做成模拟电路控制的样子。
- 您可以将一组View组件合成为一个新的独立组件，比如制作一个下拉列表框（弹出列表和输入框的组合）、双区域选择控制器（有左、右两个选择区域，选择框中的元素可随意切换其左右位置）等等。
- 您可以重载EditText组件的屏幕绘制方式（[NotePad教程](#)很好地利用了这一点，使之产生了带有下划线的记事本页面）。
- 您可以监听多种事件（包括按下按键事件），并可以定制这些事件的处理方式。

以下小节会介绍如何定制View，并将其用于App中。若需要更详尽的参考信息，请查看[View类](#)。

基本方法

以下是定制View组件的步骤总览：

1. 使新类继承自[View](#)类或其子类；
2. 重载父类特定方法。被重载的父类方法通常以“on”开头，例如：[onDraw\(\)](#)、[int onMeasure\(\)](#)和[android.view.KeyEvent\) onKeyDown\(\)](#)，这与您重载[Activity](#)或[ListActivity](#)中的事件用以控制生命周期和钩子函数的做法类似；
3. 使用该扩展类。完成后，这个扩展类便可代替其父类使用，并体现出新的特性。

小提示：扩展类可以被定义为使用它的Activity的内部类，这样做可以控制扩展类的范文权限，但这并不是必须的（有可能您需要创建一个可以在App中多处用到的扩展类）。

完全定制的组件

不论您对于视图组件的期望多么夸张，完全定制的组件都可以实现，包括实现一个看起来像旧模拟仪表盘的图形化紫外线辐照计，或者实现一个有着进度标记的长文本使之就像一台卡拉ok机的字幕。总之，这些特殊功能不可以通过已有组件实现，而且组合使用已有组件也不能满足需求。

幸运的是，您可以轻松构建一个样式、外观完全满足您需要的新组件，包括控制新组件在屏幕上所占区域大小和在运行时的耗电量（需要牢记的是您的App将安装在相比工作站电量要有限得多的移动设备上）。

创建一个完全定制的组件需要以下几步：

1. 完全定制的组件通常继承自[View](#)类，所以搭建完全定制的组件的第一步通常是继承该类；
2. 可以向完全定制的组件加入一个接收XML属性和参数的构造函数，当然，您也可以为其指定属性和参数（比如紫外线辐照计的颜色和范围、辐照计指针的宽度或阻尼系数）；
3. 可以为完全定制组件创建新的事件监听器、属性访问修改器以及更为复杂的行为；
4. 基本上都会重载onMeasure()方法，在控制组件的显示时会重载onDraw()方法。这两个方法都有其默认行为，onDraw()方法默认情况下什么都不做，onMeasure()方法默认情况下会设置100*100的区域大小--而这并不一定是您所需的；
5. 对形如“on...”的方法按需进行重载。

扩展onDraw()与onMeasure()方法

onDraw()方法提供了[Canvas](#)对象的引用，您可以在其上实现各种需求，包括：绘制2D图片或特殊风格的文本、添加其他标准组件或定制组件等。

注意： Canvas不支持3D绘图，如果要绘制3D图片，您需要继承[SurfaceView](#)类而不是View类，并在一个单独的线程中绘制图像。具体细节请查看[GLSurfaceViewActivity](#)示例程序。

onMeasure()方法囊括的东西更多一些，onMeasure()方法是定制组件在其容器中正确呈现的关键。重载的onMeasure()方法应能够及时准确地反映外界对其的操作。如果定制组件的父组件对定制组件有某些限制（通过onMeasure()附加限制），或者在对组件大小经过计算后调用setMeasuredDimension()对定制组件的操作范围加以限制，在onMeasure()方法中的控制就会变得更复杂一些。如果您未能正确调用重载后的onMeasure()方法，程序就会在进行操作的时候抛出异常。

重载onMeasure()方法大致要达到以下几个要求：

1. 重载的后onMeasure()方法应该在指定操作区域被调用，而这个指定区域应该是由您加以限制（通过两个代表尺寸的整形参数：`widthMeasureSpec`和`heightMeasureSpec`）。如果需要查看对于指定区域尺寸的要求可以参考[View.onMeasure\(int, int\)](#)的文档（该参考文档也很好地解释了所有触屏操作）；
2. 您定制组件的onMeasure()方法应该计算出触控操作的区域大小，在绘制组件时

会用到该区域大小。操作应该都限于该指定区域之内，虽然在该指定区域外也可以进行操作（这种情况下，定制组件的父组件对操作进行处理，包括：切屏、滚动、抛出异常或是调用其他特殊操作区域的 `onMeasure()` 尝试处理）。

3. 在完成对操作区域长宽的计算之后，必须调用 `setMeasuredDimension(int width, int height)` 方法，调用该方法时的参数必不可少。

下面是Framework生成View时调用的标准方法汇总：

Category	Methods	Description
Creation	Constructors	组件有两种调用构造函数的形式： 1. 使用代码调用； 2. 组件在通过布局文件进行绘制时调用。其中第二种情况，需要对布局文件中定义的全部属性进行解析和应用。
	<u>onFinishInflate()</u>	该方法会在视图（View）及其全部子视图绘制完成后调用。
Layout	<u>onMeasure(int, int)</u>	用于确定视图及其所有子视图的尺寸。
	<u>onLayout(boolean, int, int, int, int)</u>	在根视图指定其所有子视图的尺寸和位置时调用。
	<u>onSizeChanged(int, int, int, int)</u>	在视图尺寸发生变

		化时调用。
Drawing	<u>onDraw(Canvas)</u>	在绘制视图内容时调用
Event processing	<u>onKeyDown(int, KeyEvent)</u>	在发生按下按钮事件时被调用。
	<u>onKeyUp(int, KeyEvent)</u>	在发生抬起按钮事件时被调用。
	<u>onTrackballEvent(MotionEvent)</u>	在发生轨迹球移动事件时调用。
	<u>onTouchEvent(MotionEvent)</u>	在发生屏幕触控事件时调用。
Focus	<u>onFocusChanged(boolean, int, Rect)</u>	在视图得到或失去焦点时调用。
	<u>onWindowFocusChanged(boolean)</u>	在包含该视图的窗口得到或失去焦点时调用。
Attaching	<u>onAttachedToWindow()</u>	在视图附加到窗口

		上时调用。
onDetachedFromWindow()		在视图从窗口上分离时调用。
onWindowVisibilityChanged(int)		当视图所在窗口可见性发生变化时调用。

定制View控件的示例

[Api Demos](#)提供了定制View控件的示例程序，被定义为[LabelView类](#)。从LabelView示例可以看出定制View控件与定制组件之间还是有很多不同的：

- 直接继承View类可以对定制组件有完全的控制；
- 含参的构造方法用于接收绘制属性（在XML中定义的参数）。这些参数中有一部分会传给其父类View，但更为重要的是，一部分参数会用于定义LabelView的属性；
- 定义了标准的外部方法，例如：setText(), setTextSize(), setTextColor()等等；
- 重载的onMeasure方法确定了组件的绘制区域大小（注意，在LabelView中通过私有的measurewidth()方法完成）；
- 重载的onDraw()方法会将组件在方法提供的Canvas对象上绘制。

您可以在[custom_view_1.xml](#)中看到如何使用LabelView定制组件。特别需要注意的是：你可以看到"android:"命名空间与"app:"定制组件命名空间的混合使用。

以"app:"开头的参数会被定制组件（本例中时LabelView）识别使用，并会在R资源文件中以内部类的形式定义。

复合控件

如果您并不需要创建一个完全定制的组件，仅通过将已有的组件组合到一起便可满足需求，那么就创建一个复合组件（控件）吧。通过将一些小的操作（或视图）按一定逻辑组合到一起使其对外表现为单一组件。例如，组合列表框可以看成是文本输入框、备选项按钮和弹出列表的组合。如果您按下备选项按钮并从列表中选择某一项，文本输入框将会显示您的选择，当然，用户也可以直接向文本输入框输入。实际上，Android平台已经提供了具有下拉列表框功能的组件：[Spinner](#)和[AutoCompleteTextView](#)，但请先假设他们并不存在，因为通过下拉列表框来讲解复合控件比较通俗易懂。构造复合控件需要以下步骤：

1. 通常首先要实现一个继承自**Layout**的类。对于下拉列表框，水平方向的**LinearLayout**是不错的选择。由于其他的布局（**Layout**）可以嵌入其中，所以复合组件可以在具体实现很复杂的同时保证其结构无误。您可以使用声明（通过**XML**文件）或编码两种方式使用你的复合组件，与在**Activity**的使用方法没有区别；
2. 新创建的复合组件类的构造方法需要接收其父类需要的参数，在接收到这些参数后，应首先将这些参数传给其父类构造方法用于初始化父类对象。接着您就可以向在构造函数中添加其他新视图，比如文本输入域和弹出列表。当然，您也可以在**XML**中引入自己的属性和参数，并在构造函数中取出并使用；
3. 您可也为复合组件中的视图（**view**）添加事件监听器，例如，用于选择列表项后更新文本输入框的事件监听器；
4. 可以为自己的属性创建访问控制器，比如，设置文本编辑框的初始值并可以在需要时取出它的内容；
5. 在继承**Layout**类的情况下，由于**Layout**默认行为可能刚好满足您的需求，所以修改**onDraw()**和**onMeasure()**方法并不是必须的，当然，您也可以按需修改这两个方法；
6. 可以重载其他“on...”方法，例如，可以通过重载**onKeyDown()**方法来决定按键与备选列表的对应项；

总而言之，以**Layout**为基础搭建定制组件有很多优点，包括：

- 可以像**Activity**那样使用声明式**XML**文件作为布局文件，或是通过代码创建视图部件（**view**）并将其嵌入布局之中；
- **onDraw()**与**onMeasure()**方法（以及其他绝对多数“on...”方法）的默认行为都很可能满足需要，这种情况下就不需要重载这些方法了；
- 您可以非常容易地构造复杂的组件视图并实现复用，就如同在操作单一组件。

复合组件控制示例

随SDK一起发布的[API Demos](#)示例中有两个列表示例 — **Views/Lists**下的例4和例6演示了如何构造继承自**LinearLayout**的**SpeechView**类，该类用于列举演讲名言。相关代码位于**List4.java**、**List6.java**。

修改已有视图（**View**）

还有一种更为简单地创建定制视图（**View**）的备选方式，这种方式在某些情况下很实用。如果已经存在一个与需求十分相似的组件，可以继承它并按需重载该组件的某些行为。完全定制的组件也可完成相同的工作，但如果能继承自一个更具体的**View**子类，那么很多功能使用默认即可而不用您亲自实现。例如，**SDK**中有[NotePad应用](#)的示例程序。从多个方面演示了如何使用**Android**平台，其中有介绍了如何扩展**EditText**视图类来绘制带线的记事本页面。这并不是一个绝佳示例，另外**API**也可能发生变化了，但是它确实证明了上述原则。您可以查看**NotePad**示例程序的源代码或是在**Eclipse**中

引入，尤其需要注意的是[NoteEditor.java](#)文件中LinedEditText的定义。有几点需要注意：

1. 类的定义

该类通过下述代码进行定义：

```
public static class LinedEditText extends EditText
```

- 它被定义为NoteEditor的内部类，由于被定义为公开的内部类，所以如有需要，可以在NoteEditor类外通过NoteEditor.MyEditText对其进行访问；
- 它是一个静态类，也就是说它不会产生“合成方法”来访问其所在类的数据，换言之，LinedEditText类具有很强的独立性且与NoteEditor类的耦合性很低。如果内部类并不需要访问其外部类，将其设成静态内部类是一种较为简洁的方式，这样做可以保证生成类体积较小，同时，也可被其他类轻松使用。
- 该内部类继承自EditText，在本例中我们以EditText来定制我们需要的组件。在完成之后，这个经过定制的新类就可以替换EditText视图使用。

2. 类的初始化

任何情况下，父类构造函数都会被最先调用。需要注意的是，这是一个含参构造函数而非无参构造函数。EditText视图组件会在绘制时从XML布局文件中取出必要参数，因此，定制组件的构造函数需要接收必要参数并将参数传给其基类构造函数。

3. 重载方法

在本例中，只有onDraw()方法被重载，您在创建自有的定制组件时，可以按需重载其他方法。

对于NotePad示例，重写onDraw()方法允许我们在EditText视图的Canvas（Canvas对象会通过onDraw()方法传入）上绘制蓝线。另外，super.onDraw()会在重载的onDraw()方法结束前被调用。在本例中，我们在绘制完蓝线后调用基类的方法。

4. 使用定制组件

现在我们有了定制组件，但是我们如何使用呢？在NotePad示例中，布局声明文件直接使用定制组件，所以请查看res/layout文件夹下的note_editor.xml文件。

```
<view class="com.android.notepad.NoteEditor$MyEditText"
    id="@+id/note"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:drawable/empty"
    android:padding="10dip"
```

```
    android:scrollbars="vertical"
    android:fadingEdge="vertical" />
```

- 定制组件在XML文件中以扩展视图(generic view)的形式创建，类定义通过完整包名进行指明。需要注意的是我们通过NoteEditor\$MyEditText句式来引用内部类，这也是Java语言引用内部类的标准方法。

如果定制视图组件不是以内部类的形式定义，那么，可以使用类的完整路径来声明视图组件，并去掉class属性。如下所示：

```
<com.android.notepad.MyEditText
    id="@+id/note"
    ... />
```

请注意MyEditText并定义在一个单独的类文件中，如果将该类嵌入NoteEditor类中，就不能使用这种方法了。

- 定制组件接收定义的其他属性和参数，并将其传给EditText的构造方法，所以定制组件接收的参数与EditText视图接收的参数是相同的。另外，您也可以添加自有参数，这会在下面的章节介绍。

到此本节就要结束了。本节的演示示例比较简单，您完全可以根据需要来构造不同复杂度的定制组件。复杂的定制组件会重载更多的"on..."方法并引入辅助方法来定制其属性和行为。实现定制组件正如那句“只有想不到没有做不到”所说，动手实现您需要的定制组件吧！

来自 "[index.php?title=Custom_Components&oldid=11905](#)"



App Resources

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址：<http://developer.android.com/guide/topics/resources/index.html>

翻译：croftwql

更新：2012.08.15



目录

- [1 应用资源](#)
 - [1.1 博客文章](#)
 - [1.1.1 屏幕尺寸管理的新工具](#)
 - [1.1.2 Holo无处不在](#)
 - [1.1.3 大屏幕上应用的新模式](#)
 - [1.2 训练](#)
 - [1.2.1 支持不同的设备](#)
 - [1.2.2 多个屏幕设计](#)

应用资源

构建一个出色应用，它比实际编码代码还要多。资源的其他文件和静态的内容，您的代码使用，如位图，布局定义，用户界面字符串，动画说明，

更多。

[概述](#)>

博客文章

屏幕尺寸管理的新工具

Android 3.2包含新的工具支持的屏幕尺寸范围广泛的设备。一个重要的结果是更好地支持一个新的屏幕大小，通常称为一个7英寸。此版本还提供了一些新的API，以简化开发吗？工作在适应不同的屏幕尺寸。

Holo无处不在

之前的Android 4.0系统主题器件的差异可能很难用一个单一的可预见的外观设计一个应用程序和感觉。我们的目标是在冰淇淋三明治和超越的开发者社区，以改善这种情况。

大屏幕上应用的新模式

Android平板电脑正在变得越来越受欢迎，我们很高兴地注意到，绝大多数的应用程序适应了较大的屏幕大小。要保持几个应用程序，不调整以及令人沮丧的用户与他们的牌位上的尴尬前瞻性应用的Android 3.2引入了一个屏幕兼容性模式，使得这些应用程序的可用。

训练

支持不同的设备

这堂课教你如何使用基础平台功能，利用替代资源和其他功能，使您的应用程序可以提供各种兼容Android的设备优化的用户体验，用一个单一的应用程序包（APK）。

多个屏幕设计

[这个类显示你如何实现这几个屏幕配置优化的用户界面。](#)

来自“[index.php?title=App_Resources&oldid=8512](#)”

Resources Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址：<http://developer.android.com/guide/topics/resources/overview.html>

翻译：croftwql

更新：2012.08.01

资源概况

你应该总是用外部资源，如图像和应用程序代码字符串，这样您就可以保持他们的独立。外部资源，您还可以提供替代资源，支持不同的语言或屏幕尺寸，这变得越来越重要，随着越来越多的android设备提供不同的配置，如特定的设备配置。为了提供不同配置的兼容性，您必须在项目的组织资源res/目录下，使用各种子目录组类型和配置资源。

对于任何类型的资源，你可以指定默认和多个 替代资源，为您的应用程序：

- 不管设备的配置，或当有不符合当前配置的替代资源，应使用默认资源。

主题

- [提供资源](#)
- [访问资源](#)
- [处理运行时更改](#)
- [本地化](#)

参考

- [资源类型](#)

- 替代资源，你使用一个特定的配置设计。

要指定一个特定的配置是一组资源，追加适当的配置目录名限定符。



图1。两个不同的设备，使用默认的布局（应用程序没有提供替代的布局）。

例如，当您的默认的UI布局保存在res/layout/ 目录，你可以 指定一个不同的布局时要使用的屏幕在横向，它保存在res/layout-land/ 目录。
android自动套用适当的资源，通过配套设备的当前配置资源目录名。

图1所示系统适用于两个不同的设备相同的布局时，没有可用的替代资源。
图2显示了相同的应用程序时，它增加了一个更大的屏幕替代布局资源。



图2。两个不同的设备，每次使用不同的布局提供不同的屏幕尺寸。

下列文件提供了一个完整的指南，你如何组织你的应用程序资源，指定 替代资源，在您的应用程序，访问它们：

[提供资源](#)

资源是哪些可以提供您的应用程序，保存到哪里，以及如何创建具体的设备配置的替代资源。

[访问资源](#)

如何使用您所提供的，或者引用从您的应用程序代码，或从其他XML资源的资源。

处理运行时更改

如何管理您的活动正在运行时发生的配置变更。

本地化

自底向上的本地化您的应用程序使用替代资源的指南。虽然这只是一个替代资源的具体使用，它是非常重要的，为了更多的用户。

资源类型

各种资源类型的参考，您可以提供，描述的XML元素，属性和语法。

例如，这个参考告诉你如何创建一个应用程序菜单，可绘制，动画，以及更多的资源。

来自“[index.php?title=Resources_Overview&oldid=8678](#)”

Providing Resources

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/intl/zh-CN/guide/topics/resources/providing-resources.html>

更新：Snowxwyo

2012年6月12日 (二) 22:54 (CST)

目录

- [1 资源提供-Providing Resources](#)
 - [1.1 分组资源类型-Grouping Resource Types](#)
 - [1.2 提供备选资源-Providing Alternative Resources](#)
 - [1.2.1 修饰语命名规则-Qualifier name rules](#)
 - [1.2.2 创建化名资源-Creating alias resources](#)
 - [1.2.2.1 绘图-Drawable](#)
 - [1.2.2.2 布局-Layout](#)
 - [1.2.2.3 字条串和其他基本值-Strings and other simple values](#)
 - [1.3 提供最佳的设备与资源的兼容性-Providing the Best Device Compatibility with Resources](#)
 - [1.4 Android怎么样寻找最佳匹配资源-How Android Finds the Best-matching Resource](#)

资源提供-Providing Resources

你应该经常外部化你应用程序代码中的资源，比如图片、字符串等，这样有利于你独立处理这些资源。你也应该根据特定的设备配置提供一些可替代的资源，并且把他们分组保存在指定的路径名下。运行时，Android可以根据当

快速浏览

- 不同类型的资源属于res/下不同的子路径。
- 可选资源提供了特定配置的资

前的配置使用适当的资源。比如，你也许会根据不同的屏幕尺寸提供不同的UI布局或是不同的语言设定提供不同的字符串。

一旦你外部化了应用程序中的资源，你就能通过项目中的R类<code>生成的ID来调用他们。怎么使用你的资源将在[资源访问-Accessing Resources](#)具体讨论。这篇文档将向你展示怎么样分类你Android项目中的资源，以及怎么样给特定的设备配置提供可替代的资源。

分组资源类型-Grouping Resource Types

你应该把每一种类型的资源分别放在你的项目中res中特定的子路径下。这是一个简单的项目中，文件分层的例子：

```
MyProject/
  src/
    MyActivity.java
  res/
    drawable/
      icon.png
    layout/
      main.xml
      info.xml
    values/
      strings.xml
```

源文件

- 总是包含默认资源，以至于你的应用不依赖于特定的设备配置

本文内容：

[分组资源类型-Grouping Resource Types](#)

[提供备选资源-Providing Alternative Resources](#)

[修饰语命名规则-Qualifier name rules](#)

[创建化名资源-Creating alias resources](#)

[提供最佳的设备与资源的兼容性-Providing the Best Device Compatibility with Resources](#)

[为Android1.5提供屏幕资源兼容性-Providing screen resource compatibility for Android 1.5](#)

[Android怎么样寻找最佳匹配资源-How Android Finds the Best-matching Resource](#)

[已知问题-Known Issues](#)

其他资源：

[资源访问-Accessing Resources](#)

[资源类型-Resource Types](#)

[Supporting Multiple Screens](#)

从这个例子中你可以看到，`res/`路径下包含了所有类型的资源（在每一个子路径中）：一个图片资源，两个布局资源和一个字符串资源文档。资源路径名非常重要，并在表1中做了具体描述。

表1 项目`res/`下的资源路径

路径	资源
<code>animator/</code>	XML文件，定义了 属性动画-property animations
<code>anim/</code>	XML文件，定义了 渐变动画-tween animations . (属性动画-property animations 也能保存在这个路径下，但 <code>animator/</code> 路径是专为 属性动画-property animations 准备的，用来区别这两种类型的动画)
<code>color/</code>	XML文件，定义了一个颜色状态列表. 详见 颜色状态列表资源-Color State List Resource
<code>drawable/</code>	位图(<code>.png</code> , <code>.9.png</code> , <code>.jpg</code> , <code>.gif</code>)或XML文件， 编译成以下绘图资源子类型： <ul style="list-style-type: none"> 位图文件 Nine-Patches (尺寸可变的位图) 状态列表 形状 动画画板 其他绘图 详见 绘图资源-Drawable Resources .
<code>layout/</code>	XML文件，定义了用户接口布局. 详见 布局资源-Layout Resource .
<code>menu/</code>	XML文件，定义了应用程序的菜单，如选择菜单<Options Menu>，快捷菜单<Context Menu>和子菜单. 详见 菜单资源-Menu Resource .
<code>raw/</code>	任意的原始格式文件。用 InputStream 来打开这些资源，通过资源ID，调用 Resources.openRawResource() 方法，即 <code>R.raw.filename</code> 。 <p>但是，如果你想调用原始的文件名和文件层级，你应该考虑把这些资源保存在<code>assets/</code>路径下(而不是<code>res/raw/</code>). 在<code>assets/</code>中的文件不会被赋予资源ID，所以你只</p>

能通过[AssetManager](#)类来读取它们。

values/

XML文件，包含了基本数值,如字符串，整型和颜色。

在res/路径下的其他XML资源文件中定义了单个基于XML文件名的资源，然而，在values/路径下描述了多个资源。在这个路径下的文件中，每一个子资源<resources>元素都定义了一个单独的资源。比如，一个字符串<string>元素创建了一个R.string资源，一个颜色<color>元素创建了一个R.color资源。

因为每一种资源都是由其XML元素所定义，所以你可以取任何你想要的文件名，并且添加不同类型的资源到同一个文件中。但是，为清楚起见，你应该把不同的资源放在不同的文件中。例如，以下是一些你可以使用的每一种资源对应的常用文件名：

- arrays.xml 数组资源 ([类型数组-typed arrays](#))
 - colors.xml 颜色-color values]
 - dimens.xml 维度-dimension values
 - strings.xml 字符串-string values
 - styles.xml 风格-styles
- 详见[字符串资源-String Resources](#), [风格资源Style Resource](#), 以及[更多资源类型-More Resource Types](#).

xml/

任意的XML文件，能在运行时被[Resources.getXML\(\)](#)调用。各种XML的配置文件必需存放在此，如：[可搜索配置-searchable Configuration](#)。

- 注意：请勿将任何资源文件直接保存在res/路径下——这将导致编译错误。

更多关于各类资源的信息，请查看[资源类型-Resource Types](#)文档。

所有保存在表1中提到的子路径下的资源都是你的“默认”资源。也就是说，这些资源为你的应用程序定义了默认的设计和内容。但是不同类型的Android设备，可能会要求不同类型的资源。例如，某个设备有一个比一般设备更大的屏幕，那你应该提供不同的布局资源来充分利用额外的屏幕空间。或是某个设备使用了一个不同的语言设置，那你应该提供

一个可替代资源来翻译你的用户接口中的字符串资源。为不同设备配置提供不同的资源，你需要在默认资源之外，提供一些可选资源。

提供备选资源-Providing Alternative Resources

几乎所有的应用都应该提供备选资源来支持特定的设备配置。比如，你应该为不同的屏幕密度提供备选的绘图资源，为不同的语言提供不备选的字符串资源。Android将检测当前的设备配置，为你的应用加载适当的资源。

指定一系列特殊配置备选资源：



1. 在`res/`路径下，以`<资源名><resources_name>-<配置修饰语><config_qualifier>`的形式创建一个新的路径。

- `<资源名><resources_name>`是对就默认资源的路径名（见表1）。
- `<修饰语><qualifier>`是一个名字，指向了一个独立的配置，表示这个资源的用途(见表2) 。

你可以添加多个`<修饰语><qualifier>`, 以'-'分隔。

图1 两个不同的设备，使用不同的布局资源

- 注意：当添加多个修饰语时，你必须把他们按表2中的名字顺序排列。如果修饰语的顺序不对，这个资源将被忽略。

2. 把资源保存各自的新路径下。资源的文件名必须与默认的资源文件名相当。

以下是默认和备选资源的例子：

`res/`

`drawable/`

`icon.png`
`background.png`

`drawable-hdpi/`

icon.png
background.png

hdpi这个修饰语指示了，在这个路径下的所有资源将被用于高屏幕密度的设备。在这两个绘图资源路径下的图片是根据不同的屏幕密度分组的，当是文件名是一模一样的。通过这种方法，两个icon.png图片和两个background.png图片的资源ID始终是一致的，不过Android可以通过比较设备的配置信息和资源路径中修饰语的名字，为当前设备选择最佳的资源。

Android支持几种配置修饰语，你可以添加多个修饰语到同一个路径名，通过'-'来分隔每一个修饰语。表2列举了可用的配置修饰语，考虑优先级，如果你为同一个资源路径使用了多个修饰语，你应该按照表中的列举的顺序添加。

表2 配置修饰名

配置	修饰值	描述
MCC and MNC	如： mcc310 mcc310- mnc004 mcc208- mnc00 等	<p>移动手机国家区号(MCC)，排在移动网络代码 (MNC) 之前。MNC来自设备的SIM卡。比如：mcc310表示美国，任何运营商；mcc310-mnc004表示美国Verizon；mcc208-mnc00表示法国Orange.</p> <p>如果设备使用无线电连接(GSM手机)，那么MCC和MNC的值来自于SIM卡。</p> <p>你也可以只使用MCC（例如：你的应用中包含了某个指定国家的合法资源）。如果你只是需要指定不同的语言，那么请使用语言和区域修饰语（见下文）。如果你决定使用MCC和MNC修饰语，你应该仔细的做一些测试以保证其正常工作。</p> <p>你也可以参与配置域mcc和mnc,他们分别指示了当前的移动手机国家区号和移动网络代码。</p>

<p>语言和区域<Language and region></p>	<p>如: en fr en-rUS fr-rFR fr-rCA 等</p>	<p>语言由两个ISO 639-1语言代码组成，后面可以跟着两个由ISO 3166-1-alpha-2定义的区域代码字母(前面加小写字母“r”)。</p> <p>这些代码不区分大小写，前缀“r”是用来区分区域代码的部分。你不前单独使用区域代码。</p> <p>在应用程序的运行生命周期中，如果用户更改了他的系统语言设定，这些设置可以被更改。详见运行时变化处理- Handling Runtime Changes，了解其在运行时将怎么样影响你的应用程序。</p> <p>查看定位-Localization完整向导，了解怎样根据其他言语定位你的应用程序。</p> <p>同样可见locale配置域，其指示了当前的定位。</p>
<p>最小宽度<smallestWidth></p>	<p>sw<N>dp 如: sw320dp sw600dp sw720dp 等.</p>	<p>屏幕的基本尺寸，由可用屏幕区域最短的尺寸决定。特别地，设备的最小宽度<smallestWidth>是指屏幕可用的高度和宽度中最短的那个（你也可以认为是一个屏幕“最短可能的宽度”）。你使用这个修饰语，可以保证，在不考虑当前屏幕方向的情况下，你的应用程序至少拥有<N> dps的宽度供其UI使用。</p> <p>比如，你的布局始终要求屏幕最小尺寸不少于600 dp，那么，你就可以使用这个修饰语创建一下布局资源，res/layout-sw600dp/。系统只会在最短的屏幕可能尺寸不少于600dp时使用这个资源，而不会去考虑600dp到底是高还是宽。最小宽度<smallestWidth>是一个确定的设备屏幕尺寸的特性；设备的最小宽度并不会因为屏</p>

幕的方向改变而改变。

设备的最小宽度`<smallestWidth>`需要考虑屏幕的装饰和系统的UI。例如，如果设备在屏幕上有一些固定的UI元素，需要战胜最小宽度`<smallestWidth>`轴上的空间，系统将申明最小宽度`<smallestWidth>`小于实际的屏幕尺寸，因为那些屏幕像素对于你的UI来说不可用。另外，你使用的数值应该是确切的你的布局所需要的最小尺寸（一般来说，这个值是你的布局所支持的“最小的宽度”，无需考虑屏幕当前的方向）

你可能使用到的一些通用的屏幕尺寸值：

- 320, 设备的屏幕配置如下：
 - 240x320 ldpi (QVGA 手机)
 - 320x480 mdpi (手机)
 - 480x800 hdpi (高密度手机)
- 480, 屏幕尺寸： 480x800 mdpi (平板电脑/手机)
- 600, 屏幕尺寸： 600x1024 mdpi (7寸平板电脑)
- 720, 屏幕尺寸： 720x1280 mdpi (10寸平板电脑)

当你的应用程序提供了不同的最小宽度`<smallestWidth>`修饰语，放在多资源路径下时，系统将使用最接近（但不超过）设备的最小宽度`<smallestWidth>`

在API level 13中加入。

也可见[android:requiresSmallestWidthDp](#)属性，申明了你的应用程序所兼容的最小的最小宽度`<smallestWidth>`值；及[smallestScreenWidthDp](#)配置域，设置了设

备的最小宽度<smallestWidth>值。

更多的关于不同屏幕设计和这个修饰语使用的信息，请见[Supporting Multiple Screens](#)开发者向导。

可用宽度<Available width>

w<N>dp

如: w720dp
w1024dp
等。

指定了一个最小的屏幕可用宽度，单位应该使用dp，能过<N>的值来定义。这个配置的值将会随着横屏和竖屏的转换而发生变化来匹配当前的实际宽度。

当你的应用程序为这个配置提供了不同的值并且放在多个资源路径下时，系统将会使用最接近（但不超过）设备当前的屏幕宽度。这个值需考虑屏幕的装饰，所以如果一个设备有一些固定的UI元素显示在左边缘或右边缘，它将使用一个比真实屏幕更小的宽度，考虑这些UI元素，并减小了应用的可用空间。

在API level 13中加入。

见[screenWidthDp](#)配置域，其设置了当前屏幕的宽度。

更多的关于不同屏幕设计和这个修饰语使用的信息，请见[Supporting Multiple Screens](#)开发者向导。

可用高度<Available height>

h<N>dp

如: h720dp
h1024dp 等。

指定了一个最小的屏幕可用高度。单位应该使用dp，能过<N>的值来定义。这个配置的值将会随着横屏和竖屏的转换而发生变化来匹配当前的实际高度。

当你的应用程序为这个配置提供了不同的值并且放在多个资源路径下时，系统将会使用

最接近（但不超过）设备当前的屏幕高度。这个值需考虑屏幕的装饰，所以如果一个设备有一些固定的UI元素显示在左边缘或右边缘，它将使用一个比真实屏幕更小的高度，考虑这些UI元素，并减小了应用的可用空间。不是固定不变的屏幕装饰（如屏幕状态条在全屏时可以被隐藏）不在这个考虑范围，窗口装饰如标题栏和工具栏等也不在考虑范围内，所以应用程序应该准备好处理一些比他们设定的更小的空间。

在API level 13中加入。

见[screenHeightDp](#)配置域，其设置了当前屏幕的宽度。

更多的关于不同屏幕设计和这个修饰语使用的信息，请见[Supporting Multiple Screens](#)开发者向导。

屏幕尺寸<Screen size>

small

normal **large**
xlarge

small:类似于低密度的QVGA屏幕的尺寸。小屏幕的最小布局尺寸约为320x426 dp。如QVGA低密度和VGA高密度。

normal:类似于中等密度的HVGA屏幕尺寸。一般屏幕的最小布局尺寸约340x470 dp。如WQVGA低密度，HVGA中等密度，WVGA高密度。

large:类似于中等密度的VGA屏幕尺寸。大屏幕的最小布局尺寸约480x640 dp。如中等密度的VGA和WVGA屏幕。

xlarge:那些比传统中等密度HVGA更大的屏幕。加大屏幕的最小布局尺寸约720x960 dp。大多数情况下，加大屏幕的设备因为屏幕过大而无法放入口袋，一般为平板电脑类的设备。在API level 9中加入。

- 注解：使用某一个尺寸的修饰语并不代表该设备必须是该尺寸的屏幕。

表资源只能被这种尺寸的屏幕使用。如果你提供的备选资源描述不能很好的与当前设备的配置匹配，而系统将选取其中最佳的那个资源。

- 注意：如果你的所有资源都使用了一个比当屏幕更大的尺寸修饰语，系统将不会使用任何一个资源，你的应用程序将在运行里崩溃（如：如果所有的资源都被标上了`xlarge`，但设备是一个普通尺寸的屏幕。）

在*API level 4*中加入。

更多信息，请见[Supporting Multiple Screens](#)开发者向导。

见[screenLayout](#)配置域，其中指示了屏幕是否为大，中或小。

屏幕<Screen aspect>

`long`

`notlong`

`long`: 长屏幕，如：`WQVGA`, `WVGA`, `FWVGA`

`notlong`: 非长屏幕，如：`QVGA`, `HVGA`, and `VGA`

在*API level 4*中加入。

这个主要是基于屏幕的比例（“长”屏幕更宽一些）。和屏幕的方向无关。

见[screenLayout](#)配置域，指示了屏幕是否为长屏幕。

屏幕方向<Screen orientation>

`port`

`land`

`port`: 竖屏 (垂直方向的)

`land`: 横屏 (水平方向的)

		<p>在应用程序的生命周期中这个配置会随着用户旋转屏幕而发生改变。详见运行时变化处理- Handling Runtime Changes，了解其在运行时将怎么样影响你的应用程序。</p> <p>见orientation配置域，指示了，设备当前的方向。</p>
停靠模式<Dock mode>	<p>car</p> <p>desk</p>	<p>car: 设备在车中</p> <p>desk: 设备在桌上</p> <p>在<i>API level 8</i>中加入。</p> <p>在应用程序生命周期中，这个配置会因为用户改变设备停放位置而发生改变。你可以通过UiModeManager启用或禁用这个模式。详见运行时变化处理- Handling Runtime Changes，了解其在运行时将怎么样影响你的应用程序。</p>
夜间模式<Night mode>	<p>night</p> <p>notnight</p>	<p>night: 夜间</p> <p>notnight: 白天</p> <p>在<i>API level 8</i>中加入。</p> <p>在应用程序生命周期中，如果使用自动（默认）夜间模式，这个模式将会发生改变，即，这个模式会依据时间而改变。你可以通过UiModeManager启用或禁用这个模式。详见运行时变化处理- Handling Runtime Changes，了解其在运行时将怎么样影响你的应用程序。</p>
屏幕像素密度<Screen pixel density> (dpi)	<p>ldpi</p> <p>mdpi</p> <p>hdpi</p>	<p>ldpi: 低密度屏幕；约120dpi。</p> <p>mdpi: 中等密度（传统的HVGA）屏幕；</p>

xhdpi nodpi
tvdpi

约160dpi。**hdpi**: 高密度屏幕；约240dpi。
xhdpi: 加高密度屏幕；约320dpi。在**API level 8**中加入。**nodpi**: 这个可以用做位图资源，你不需要通过拉伸来匹配屏幕密度。
tvdpi: 屏幕在中等与高密度屏幕之间；约213dpi。这个不在首选的密度分组范围之内。这个主要是为电视和大部分不需要这个配置的应用准备的。对于很多应用，提供**mdpi**和**hdpi**资源都不能有效的匹配，系统会将它们拉伸到适合的大小。这个修饰语是在**API level 13**中加入的。

在首选的密度中有一个3:4:6:8拉伸比例（不考虑**tvdpi**）。所以在**ldpi**中9x9的位图，在**mdpi**中为12x12，在**hdpi**中为18x18，在**xhdpi**中为24x24。

如果你认为你的图片在电视或其他一些设备中效果不会很好，并且想尝试**tvdpi**资源，那么拉伸系数为 $1.33 * \text{mdpi}$ 。例如：在**mdpi**中一个100px x 100px的图像，在**tvdpi**中将为133px x 133px.

- **注解：**使用一个密度修饰语并不表示这些资源只被用在指定的密度的屏幕中。如果你提供的可选资源和修饰语没有很好的匹配当前的设备配置，系统将会使用其中最好的那一个。

请见[Supporting Multiple Screens](#)，了解更多关于如何处理不同屏幕密度以及Android可能会拉伸你提供的位图来适应当前的密度。

触屏类型<Touchscreen type>

notouch
stylus finger

notouch: 设备不支持触屏。
stylus: 设备有一个电阻式触屏，则适合使
stylus

		用 。 finger : 设备有一个触摸屏。
键盘可用性<Keyboard availability>	keysexposed keyshidden keyssoft	<p>keysexposed: 设备有一个可用键盘。如果设备有一个软键盘可用，这个配置也可用，即使物理键盘没有暴露给用户，甚至设置没有物理键盘。如果没有软键盘如是被禁用，那个这配置只有在有物理键盘被暴露时可用。</p> <p>keyshidden: 设备有一个物理键盘可用，但被隐藏了，且没有软键盘。keyssoft: 设备有一个软键盘可以，不管其是否可见。</p> <p>如果你提供了一个keysexposed资源，但没有keyssoft资源，系统将使用keysexposed资源，而不考虑键盘是否可见，如果系统有一个软键盘可用。</p> <p>在应用程序的生命周期中这个配置会因用户打开一个物理键盘而发生改变。详见运行时变化处理- Handling Runtime Changes，了解其在运行时将怎么样影响你的应用程序。</p> <p>同样可 见hardKeyboardHidden和keyboardHidden配置域，分别指示了物理键盘的可见性和任何键盘（包括软键盘）的可见性。</p>
首选文本输入方法<Primary text input method>	nokeys qwerty 12key	<p>nokeys: 设备没有物理键用作文本输入。</p> <p>qwerty: 设备有一个物理的 qwerty 键盘，不管其是否对用户可见。12key: 设备有一个物理的 12-key 键盘，不管其是否对用户可见。</p>

		同样可见 keyboard 配置域，指示了可用的首选文本输入方法。
导航键可用性<Navigation key availability>	<code>navexposed</code> <code>navhidden</code>	<p><code>navexposed</code>: 导航键对用户可用。</p> <p><code>navhidden</code>: 导航键不可用（如：被盖子遮挡）。</p> <p>在应用程序生命周期中，这个配置会随着用户打开导航键而改变。见运行时变化处理-Handling Runtime Changes，了解其在运行时将怎么样影响你的应用程序。</p> <p>同样可见navigationHidden，指示了导航键是否被隐藏。</p>
首选非触摸导航方式<Primary non-touch navigation method>	<code>nonav</code> <code>dpad</code> <code>trackball</code> <code>wheel</code>	<p><code>nonav</code>: 设备除了触摸屏没有导航设备。</p> <p><code>dpad</code>: 设备有一个directional-pad (d-pad)用来导航。</p> <p><code>trackball</code>: 设备有一个trackball用来导航。</p> <p><code>wheel</code>: 设备有一个directional wheel(s)用来导航(不常见)。</p> <p>同样可见navigation配置域，用来指示可用的导航方法类型。</p>
平台版本<Platform Version> (API level)	如： <code>v3 v4 v7</code> 等。	<p>设备所支持的。例如, <code>v1</code> 指API level 1 (设备为Android 1.0或更高版本) and <code>v4</code> 指 API level 4 (设备为 Android 1.6 或更高版本). 头天这个值的更多信息，请见Android API levels文档</p> <ul style="list-style-type: none"> 注意：Android 1.5和Android 1.6只有在资源的修饰语和版本号完全匹配时才能

和这个资源匹配。更多信息，见后续章节[已知问题- Known Issues](#)。

- 注解：有些配置修饰语在Android 1.0时就被加入了，所以并不是所有的Android版本都支持全部的修饰语。使用一个新的修饰语，隐藏的添加了版本号修饰语，让旧版本的设备忽略这个修饰语。例如：使用w600dp修饰语，将会自动添加v13修饰语，因为可用宽度修饰语在API level 13中被加入。为了避免任何问题，始终包含一系列默认资源（一系列没有修饰语的默认资源）。更多信息，见章节[提供最好的设备兼容性与资源-Providing the Best Device Compatibility with Resources](#)。

修饰语命名规则-Qualifier name rules

以下是一些关于使用配置修饰语命名的规则：

- 你可以为单独的一系列资源指定多个修饰语，用“-”分隔。如：为美式英语、水平方向的设备指定drawable-en-rUS-land。
- 修饰语必须按照表2中所列的顺序排序。如：
 - 错误：drawable-hdpi-port/
 - 正确：drawable-port-hdpi/
- 可选资源的路径不能被嵌套。如，你不用这么使用res/drawable/drawable-en/
- 修饰语的值不区分大小写。为了避免在大小写敏感的文件系统中出错，资源编译器在处理前会先将路径名全部转化会小写。名字中的任何大写字母只是为了有利于阅读。
- 每一个修饰语类型中只有一个值被使用。例如，如果你想为西班牙和法国使用相同的绘图文件，你不能这样命名路径drawable-rES-rFR/。而是，你应该使用两个资源路径，如drawable-rES/和drawable-rFR/，其中包含了恰当的内容。然后，你并不需要在两个路径中使用相同的文件。你可以为一个资源创建一个化名。见[创建化名资源](#)。

当你把资源保存在了这些以修饰语命名的路径中之后，Android自动的根据当前设备的配置来为你的应用提供合适的资源。每一次需要调用一个资源时，Android将检查包含被要求资源的可选资源路径，然后找到最佳资源（在下文讨论）。如果没有与特定设备配置

所匹配的可选资源，那么，Android将使用默认资源（默认资源是指一系列特殊的没有设置配置修饰语的资源类型）。

创建化名资源-Creating alias resources

当你有一个资源，想为多个设备配置使用（但不想作为默认资源提供）时，你不需要把一个相同的资源放在多个可选资源路径下。而是，你可以（在某些情况下）创建一个可选资源作为某个资源的化名保存在你的默认资源路径下。

注解：不是所有资源都提供了这样的机制，能够为另一个资源创建一个化名。特别的：动画<animation>，菜单<menu>，raw，以及其他在xml/路径下的非特指的资源不提供这个功能。

例如，假设你有一个应用程序图标，icon.png需要为不同的地区指定唯一的版本。然后，两个地区，英国—加拿大人和法国—加拿大人，需要使用相同的版本。你也话会认为你需要拷贝同一个图片资源到英国—加拿大和法国—加拿大两个资源路径下，但这是不对的。而是，你可以把两个都使用到的图片资源保存为icon_ca.png（除了icon.png外的任何名字），并把他放在默认的res/drawable/路径下。然后在res/drawable-en-rCA和res/drawable-fr-rCA创建一个icon.xml文件，使用<bitmap>元素指向icon_ca.png资源。这允话你保存一个版本的PNG文件和两个小的XML文件来指向它。（下面是一个XML文件的例子。）

绘图-Drawable

使用<bitmap>元素为一个已存在的绘图文件创建一个化名。如：

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/icon_ca" />
```

如果把这个文件保存为icon.xml（在可选资源路径下，如res/drawable-en-rCA/），它将被编译成一个资源，你可以通过R.drawable.icon来引用，但它实际上也是R.drawable.icon_ca的化名（被保存在res/drawable/）。

布局-Layout

使用<include>元素为已存大的布局创建一个化名，包装在一个<merge>中 如：

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

```
<merge>
    <include layout="@layout/main_ltr" />
</merge>
```

如果你把这个文件保存为main.xml，那它将被编译成一个资源，你可以通过R.layout.main来引用，但实际上他是R.layout.main_ltr这个资源的化名。

字条串和其他基本值-**Strings and other simple values**

为一个已存在的字符串创建一个化名，简单的使用想要的字符串的资源ID作为一个新字符串的值。如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello</string>
    <string name="hi">@string/hello</string>
</resources>
```

现在R.string.hi资源是R.string.hello的一个化名。

[其他的基本值-Other simple values](#)使用方式相同。如，一个颜色值：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="yellow">#f00</color>
    <color name="highlight">@color/red</color>
</resources>
```

提供最佳的设备与资源的兼容性-**Providing the Best Device Compatibility with Resources**

为了让你的应用程序支持多个设备配置，始终为你的应用中每一种资源都提供默认的资源是非常重要的。

例如，如果你的应用程序支持多个语言，始终包括一个不含任何[语言和区域](#)修饰语的values/路径（其中保存了你用到的字符串）。如果你把所有的字符串都保存在带有语言和地区修饰语的路径下，那么如果一个设备设置了一个你的字符串不支持的语言和区域，你的应用程序将崩溃。但是，只要你提供了默认了的values/，那么你的应用将正常运行（即使是用户不懂这种语言，也好过于应用程序崩溃）。

同样地，如果你根据屏幕的方向提供了不同的布局资源，你应该选择一个作为你的默认资源。如，在提供布局资源时，不是为横屏提供一个`layout-land/`，为竖屏提供一个`layout-port/`，而是要留一个作为默认资源，比如，横屏用`layout/`，竖屏用`layout-port/`。

提供默认的资源很重要不仅仅因为你的应用程序可能会在一个你没有预想到的配置上运行，还因为新版本的Android有时候会添加一些老版本并不支持的新配置修饰语。如果你使用了一个新的资源修饰语，但你向老版本的Android兼容了你的代码，这样，当你的应用程序在一个老版本的Android设备上运行时，如果你没有提供默认的资源，其将崩溃，因为老版本的Android不能识别带有新修饰语的资源名。比如，你的[`minSdkVersion`]设置为4，而且你限制你所有的绘图资源使用`夜间模式` (`night`和`notnight`(在API level 8中被加入))，那么一个API level 4的设备就不能获取你的绘图资源，将会崩溃。在这种情况下，你大概想要`notnight`作为你的默认资源，所以你应该更改那个修饰语，将你的绘图资源放在`drawable/`或`drawable-night/`中。

所以，为了提供最好的设备兼容性，你应该总是为你的应用程序提供所需的默认资源，让其能正常运作。然后使用配置修饰语，为指定的设备配置创建可选资源。

这个规则有一个例外：如果你应用程序的[`minSdkVersion`]为4或更大，当你使用`屏幕密度`修饰语提供可选的绘图资源时，你不需要默认资源。即使没有默认资源，Android也能在你提供的可选的屏幕密度中找到最佳匹配的资源，并把其拉伸为需要的位图。然而，为了在所有设备上达到最好的体验效果，为三种不同的密度提供可选择的绘图资源。如果你的[`minSdkVersion`]小于4 (Android 1.5或更低版本)，请注意，屏幕尺寸，密度和这方面的修饰语不支持Android 1.5或更低版本。你可能要为这些版本执行其他的兼容性措施。

Android怎么样寻找最佳匹配资源-How Android Finds the Best-matching Resource

当你需要一个你提供的可选资源时，Android会根据当前的设备配置，在运行时来选择一个可选资源。为了展现Android怎么选择一个可选资源，假设下列绘图路径每一个都包含了不同版本的同一个图片：

`drawable/`
`drawable-en/`
`drawable-fr-rCA/`
`drawable-en-port/`



图2 Android寻找最佳匹配资源流程图

drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/

并且假设以下设备配置：

地区<Locale> = en-GB
屏幕方向<Screen orientation> = port
屏幕像素密度<Screen pixel density> = hdpi
触摸屏类型<Touchscreen type> = notouch
首选文本输入方法<Primary text input method> = 12key

通过对比设备配置和可能的备选资源，Android从drawable-en-port/选择了了绘图。

系统按照以下的逻辑来选择所要的资源：

1. 排除与设备配置矛盾的资源文件。

drawable-fr-rCA/这个路径被排除，因为他与en-GB这个地区矛盾。

drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/

例外：屏幕像素密度这个修饰语没有因矛盾而被排除。即使设备的屏幕密度是hdpi,但drawable-port-ldpi/没有被排除，因为这时，第一个屏幕密度都被认为是匹配的，更多我信息请见[Supporting Multiple Screens](#)文档。

2. 取（下一个）根据（表2）所列的优先级最高的修饰语。（从MCC开始，然后往下走。）

3. 是否有哪一个路径包含这个修饰语？

- 如果没有，返回第2步查找下一个修饰语。（在这个例子中，直到语言修饰语，答案者是“否”。）

如果是，继续第4步。

4. 排除不包含此修饰语的资源路径。在这个例子中，系统排除了所有不包含语言修饰语的路径：

```
drawable/
drawable-en/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

例外：如果这是关于屏幕像素密度的问题，Android将选择最接近与设备屏幕密度的选项。通常，Android倾向于缩小一个更大的原始图片而不是拉伸更小的原始图片。见[Supporting Multiple Screens](#)。

5. 返回，重复第2，3和4步，直到只有一个路径剩下。在这个例子中，屏幕方向是下一个匹配的修饰语。所以没的指定屏幕方向的路径将被排除：

```
drawable-en/
drawable-en-port/
drawable-en-notouch-12key/
```

剩下drawable-en-port/路径

虽然这个处理流程查找了每一个要求的资源，但系统在一些方面做出了进一步的优化。其中一个优化是，一旦这个设备配置已知，它将排除那些永远不能匹配的可选资源。例如，如果配置语言为英语（“en”），那么任意一个设置了除英语以外的语言修饰语的资源路径将不会再包括在被检测的资源池中（但是，没有设置语言修饰语的路径还被包括在内）。

当选择基于屏幕尺寸修饰语的资源时，如果没有更好的匹配资源，系统将会使用一个比当前屏幕更小的资源（如，在需要的情况下，一个大尺寸<large-size>屏幕将会使用一个普通尺寸<normal-size>的屏幕资源）。但是，如果只有一个比当前屏幕更大的可用资源，系统将不会使用他们，如果没有其他资源与设备配置匹配，你的应用程序将会崩溃（例如，所有的布局资源都标着xlarger这个修饰语，但设备是一个普通尺寸<normal-size>的屏幕）。

注解：修饰语的优先级（表2中所列）比与设备完全匹配的修饰语的数量更重要。

如，在第4步中，列表中剩下的最后一个选项包含了三个与设备完全匹配的修饰语（方向`<orientation>`，触屏类型`<touchscreen>`，和输入方式`<input method>`），但`drawable-en`只有一个参数（语言）匹配。然而，语言拥有一个更高的优先级，所以`drawable-por-notouch-12key`被淘汰。

学习更多关于如何在你的应用程序中使用资源，请继续阅读[资源访问-Accessing Resources](#)。

来自“[index.php?title=Providing_Resources&oldid=13867](#)”

个人工具

- [登录/创建账户](#)

名字空间

- [页面](#)
- [讨论](#)

变换

搜索

查看

- [阅读](#)
- [查看源代码](#)
- [查看历史](#)

操作

Accessing Resources

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/resources/accessing-resources.html>

更新：--[Snowxwyo](#) 2012年6月16日 (六) 00:13 (CST)

[应用程序资源-Application Resources:](#)

目录

[[隐藏](#)]

[1 资源访问-Accessing Resources](#)

- [1.1 在代码中访问资源-Accessing Resources in Code](#)
 - [1.1.1 语法](#)
 - [1.1.2 使用案例](#)
- [1.2 通过XML访问资源-Accessing Resources from XML](#)
 - [1.2.1 语法](#)
 - [1.2.2 使用案例](#)
 - [1.2.3 引用样式属性-Referencing style attributes](#)
- [1.3 访问平台资源-Accessing Platform Resources](#)

资源访问-Accessing Resources

一旦你在你的应用程序中提供了一个资源（在[资源提供-Providing Resources](#)中讨论过），你可以通过引用它的资源ID来调用它。所有在工程项目中[R class](#)中定义的资源ID都是能过[aapt](#)自动生成的。

快速浏览

- 资源可以通过[R.java](#)中的整型值在代码中被引用，如[R.drawable.myimage](#)

当你的应用程序被编译时，`aapt`生成了 `R class`，其中包含了 `res/` 路径下所有资源的资源ID。对于每天种类型的资源，都有的一 `R subclass` (子类) (如：`R.drawable` 对应绘图`<drawable>`资源) 并且对于每一种资源类型，都有一个静态整型常量 (如，`R.drawable.icon`)。这个整型常量就是可以被用来调用资源的资源ID。

虽然资源ID是在 `R class` 中指定的，但你并不需要去那儿找资源ID。一个资源ID总是由以下部分组成：

- 资源类型：每一个资源都是以某种“类型”分组的，如字符串`<string>`，绘图`<drawable>`，以及布局`<layout>`。更多的资源类型，请见[资源类型-Resource Types](#)。
- 资源名：即文件名，包括其括展名；或者，如果资源只是一个基本类型值（如一个字符串`<string>`），则是在XML中`android:name`属性的值。

访问一个资源有两种方法：

- 在代码中：使用一个 `R class` 子类中的一个静态整型值，如：

`R.string.hello`

- 资源可以在XML中，通过特殊的XML语法被引用，如`@drawable/myimage`
- 你也可以通过[Resources](#)中的方法来访问你的应用程序资源

关键类

[Resources](#)

此文档包括：

[在代码中访问资源-Accessing Resources in Code](#)

[通过XML访问资源-Accessing Resources from XML](#)

[引用样式属性-Referencing style attributes](#)

[访问平台资源-Accessing Platform Resources](#)

其他资源：

[资源提供-Providing Resources](#)

[资源类型-Resource Types](#)

`string`是资源类型，`hello`是资源名。当你提供一个这种格式的资源类型时，有很多的Android API可以访问你的资源。见[在代码中访问资源-Accessing Resources in Code](#)。

- 在XML中：使用一个特殊的XML语法，对应R class中定义的资源ID，如：

`@string.hello`

`string`是资源类型，`hello`是资源名。你可以在XML中任意一个需要的地方使用这个语法。见[通过XML访问资源-Accessing Resources from XML](#)。

在代码中访问资源-Accessing Resources in Code

你可以在代码中把资源ID传递给人个方法的参数来使用了一资源。例如，你可以通过[setImageResource\(\)](#)设置一个[ImageView](#)来使用`res/drawable/myimage.png`资源：

```
ImageView imageView = ( ImageView )  
findViewById(R.id.myimageview);
```

访问原始文件

偶尔，你可能需要访问原始文件和路径。如果你这么做了，那么你保存在`res/`中的文件将不会为你所用，因为唯一一个读取`res/`中资源的方法就是通过其资源ID。不过，你可以把你的资源保存在`assets/`路径下。

被保存在`assets/`路径下的文件将不会得到一个资源ID，所以你不可以通过R class或是XML来引用它们。不过，你可以像普通的文件系统一样，查询`assets/`路径下的文件，并且使用[AssetManager](#)来读取原始的未经处理的数据。

然而，如果你所要求的是读取原始

的未经处理的数据（如一个视频或音频文件），那么，把文件保存在`res/raw/`路径下，并通过`openRawResource()`来读取一个字节流。

```
imageView.setImageResource(R.drawable.myimage);
```

你也要以使用[Resources](#)中的方法来检索独立的资源，使用[getResources\(\)](#)来获得一个实例。

语法

这是在代码中引用一个资源的语法：

```
[<package_name>.]R.<resource_type>.<resource_name>
```

- `<package_name>`是资源所在的包名（如果被引用的资源在你自己的包中，不需要指明）。
- `<resource_type>`是R的子类中资源的类型。
- `<resource_name>`是不包括扩展名的资源文件名或XML元素中`android:name`属性的值（基本类型）。

使用案例

有很多种方法可以接受一个资源ID参数，你可以使用[Resources](#)中的方法来检索资源。你可以通过[Context.getResources\(\)](#)来获得[Resources](#)的一个实例。

以下是在代码中访问资源的一些实例：

```
// Load a background for the current screen from a drawable resource
getWindow().setBackgroundDrawableResource(R.drawable.my_background_image);

// Set the Activity title by getting a string from the Resources
// object, because
// this method requires a CharSequence rather than a resource ID
getWindow().setTitle(getResources().getText(R.string.main_title));
```

```
Accessing Resources - eoeAndroid wiki
// Load a custom layout for the current screen
setContentView(R.layout.main_screen);

// Set a slide in animation by getting an Animation from the Resources
// object
mFlipper.setInAnimation(AnimationUtils.loadAnimation(this,
    R.anim.hyperspace_in));

// Set the text on a TextView object using a resource ID
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

注意：你永远不要手动修改R.java文件——它是在项目被编译时由aapt生成的。任何的改变将在下一次编译时被重写。

通过XML访问资源-Accessing Resources from XML

你可以使用一个已存在的资源的引用来定义一些XML的属性和元素的值。当你创建布局文件时将会经常使用到这个方法为你的小部件提供字符串和图片。

例如，你为你的布局添加一个[Button](#)，你应该使用[字符串资源-String Resources](#)作为button的文字：

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/submit" />
```

语法

以下是XML中引用一个资源的语法：

`@[<package_name>:]<resource_type>/<resource_name>`

- `<package_name>`是资源所在的包名（如果被引用的资源在你自己的包中，不需要指明）。
- `<resource_type>`是R的子类中资源的类型。
- `<resource_name>`是不包括扩展名的资源文件名或XML元素中`android:name`属性的值（基本类型）。

更多关于怎么样引用每一种资源类型的信息，请见[资源类型-Resource Types](#)。

使用案例

在一些情况下，你必须使用一个资源作为XML中的一个值（如，为一个小部件提供一个图片），但是，你可以在XML中任意一个可以接受一个基本类型值的地方使用一个资源。例如，假如你有以下资源文件，包括了一个颜色启动和一个字符串资源：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <string name="hello">Hello!</string>
</resources>
</xm>
```

你可以在以下的布局文件中使用这些资源来设置文本颜色和文本字符串：`
`

```
<xml>
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

在这个案例中，在资源引用时，你不需要指名包名，因为这些资源来自于你自己的包。引用一个系统资源，你需要包括一个包名。如：

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@android:color/secondary_text_dark"
    android:text="@string/hello" />
```

注解：你应该始终都可以使用用字符串资源，这样你的应用就可以被其他语言定位。更多的关于创建可选资源（如可本地化的字符串），见[提供可选资源-Providing Alternative Resources](#)

你甚至可以在XML中创建化名。例如，你可以创建一个绘图资源，作为另一个绘图资源的化名：

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/other_drawable" />
```

这个听起来无用，但在使用可选资源时将会非常有用。更多请见[创建化名资源-Creating alias resources](#)

引用样式属性-Referencing style attributes

一个样式属性资源允许你引用一个当前使用主题中某个属性的值。引用一个样式属性允许你定制UI元素的外观，通过样式化他们来匹配当前主题提供的标准化的变化，而不是提供一个硬编码值。引用一个样式属性本质上说的是，“在当前主题下，使用一个被这个属性所定义的样式。”

引用一个样式属性，其命名的语法几乎是与一般资源的格式相同，但是，使用一个问号 (?) 来替代@符号，并且资源类型部分是可选的。如：

?[<package_name>:][<resource_type>/]<resource_name>

例如，这里是一个例子，展示了如何引用一个属性来设定文本颜色以匹配系统主题“主要的”文本颜色：

```
<EditText id="text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="?android:textColorSecondary"
    android:text="@string/hello_world" />
```

其中，`android:textColor`属性指定了当前主题中，一个样式属性的名字。在这个小部件中，Android将使用`android:textColorSecondary`样式属性的值作为`android:textColor`的值。因为系统资源工具知道，在这个环境下需要一个属性资源，你并不需要明确的指定其类型（需要可以这样定义`android:attr/textColorSecondary`）—你可以排除`attr`类型。

访问平台资源-Accessing Platform Resources

Android包含了很多标准化资源，如，样式、主题和布局。要访问这些资源，在你的资源引用中需要包括其android包名。例如，在[ListAdapter](#)中，Android提供了一个你可以用作列举项目的布局资源：

```
setListAdapter(new ArrayAdapter<String>(this,  
    android.R.layout.simple_list_item_1, myarray));
```

在这个范例中，[simple_list_item_1](#)是一个布局资源，在[ListView](#)中，为了项目由平台所定义。你可以使用这个资源而不用创建一个自己的布局来列举项目。
(更多关于如何使用[ListView](#)，请见[List View Tutorial](#)。)

[返回应用程序资源](#)

来自“[index.php?title=Accessing_Resources&oldid=8886](#)”

1个分类：[Android Dev Guide](#)



Handling Runtime Changes

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/guide/topics/resources/runtime-changes.html>

作者：542971618

完成时间：7月16日

Android之Handling Runtime Changes (处理运行时更改)

一些设备配置在运行过程中可能会发生改变（例如屏幕横向布局、键盘可用性和语言）。当这样的变化发生时，Android会重新启动这个正在运行的Activity (`onDestroy()`方法会被调用，然后调用`onCreate()`方法)。这个重启的动作是为了通过自动往你的应用程序中载入可替代资源，从而使你的应用适应新的配置。

为了正确执行一次重启，你的Activity在整个平凡的生命周期中重新保存它之前的状态是很重要的，Android是通过在销毁你的Activity之前调用`onSaveInstanceState()`方法来保存关于应用之前状态的数据。然后你就可以在`onCreate()`方法或者`onRestoreInstanceState()`方法中重新保存应用的状态了。为了测试你的应用可以通过应用的状态原封不动地重启自己，你应该给你的应用授权当程序在执行不同的任务时应用的配置可以改变（例如屏幕的方向变化）。

为了处理一些事件，例如当用户接听一个打入的电话然后返回到你的应用程序中，在没有丢失用户数据或者状态信息的情况下，你的应用应该具备在任何时候重启自己的能力（更多参见Activity lifecycle）。

然而，你可能要面对这样一个情景，重启你的应用程序并重新保存大量有价值的数据会导致很差的用户体验。在这样的情景面前，你有两种选择：

a 在配置改变期间维持一个对象

当配置发生改变时允许你的Activity重启，但让其携带一个有状态的对象到你的新Activity实例中。

b 你自己来处理配置的变化

当某些配置发生变化的时候阻止系统重启你的Activity，并且当配置改变时要接收一个回调，这样你就可以根据需要来手动更新你的Activity。

在配置改变期间维持一个对象

如果重启你的Activity，你需要恢复大量的数据，重新执行网络连接，或者其他深入的操作，这样由配置改变引起的一次完全启动就会引起不好的用户体验。而且，仅有Activity生命周期中为你保存的的Bundle对象，你是不可能完全维护你的Activity的状态的—不能传递很大的对象（如 bitmap对象），并且这些对象里面的数据必须序列化，然后解序列化，这些都需要消耗很多内存从而使配置改变得很慢。在这样的情境下，当你的Activity由于配置发生改变而重启时，你可以通过重新预置一个有状态的对象来减缓你程序的负担。

在运行期间配置改变时维护一个对象：

1. 扩展片段类，并且声明引用状态对象。
2. 创建片段时调用setRetainInstance(boolean)。
3. 向活动添加片段。
4. 活动重启时使用FragmentManager来检索片段。

例如用如下方式定义片段：

```
public class RetainedFragment extends Fragment {
```

```
// data object we want to retain
private MyDataObject data;

// this method is only called once for this fragment
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // retain this fragment
    setRetainInstance(true);
}

public void setData(MyDataObject data) {
    this.data = data;
}

public MyDataObject getData() {
    return data;
}
}
```

然后使用`FragmentManager`向活动添加片段。运行配置更改时活动重启，就可以从片段获取数据对象。例如按照如下方式定义活动：

```
public class MyActivity extends Activity {

    private RetainedFragment dataFragment;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // find the retained fragment on activity restarts
        FragmentManager fm = getFragmentManager();
        dataFragment = (DataFragment) fm.findFragmentByTag("data");

        // create the fragment and data the first time
        if (dataFragment == null) {
            // add the fragment
            dataFragment = new DataFragment();
            fm.beginTransaction().add(dataFragment,
"data").commit();
            // load the data from the web
            dataFragment.setData(loadMyData());
        }

        // the data is available in dataFragment.getData()
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // store the data in the fragment
        dataFragment.setData(collectMyLoadedData());
    }
}
```

此例中`onCreate()`添加片段或者恢复引用。`onCreate()`也在片段示例中存储了状态对象。`onDestroy()`更新了保留的片段示例中的状态对象。

你自己来处理配置的变化

如果在某个特殊的配置发生改变的期间你的应用程序不需要更新资源，而且你有个操作限制需要你避免**Activity**的重启，那么你可以声明使你自己的**Activity**来处理配置的变化，从而阻止系统重启你的**Activity**。

特别提醒：选择自己来处理配置的变化会使得可替代资源的使用变得更困难，因为系统不会为你来自动调用这些资源。这种技术应该被视为最后的手段，对于大多数应用程序不建议使用。

为了声明你的**Activity**来处理配置的变化，在清单文件中编辑正确的`<activity>`元素，包括赋好值的 `android:configChanges`属性，代表你要处理的配置。`android:configChanges`属性所有可能的值都要在文档中列出（最常用的值是：`orientation`来处理当屏幕的方向变化时，`keyboardHidden`来处理键盘可用性改变时）。你可以在属性中声明多个配置的值，通过“`|`”符号将它们分隔开。

例如，以下清单片段声明了**Activity**中将同时处理屏幕的方向变化和键盘的可用性变化：

```
<activity android:name=".MyActivity"
          android:configChanges="orientation|keyboardHidden"
          android:label="@string/app_name">
```

当这些配置中的一个发生改变时，`MyActivity`不会重新启动。相反，这个**Activity**会接收`onConfigurationChanged()`方法的调用。这个方法传递一个`Configuration`类的对象来标识新的设备配置。通过读取配置字段，你可以确定新的配置信息并通过更新你界面中使用的资源来正确应用这些改变。任何时候这个方法被调用，你的**Activity**的 `Resources`对象会被更新并返回一个基于新配置的`Resources`对象，因此你可以在不用系统重启你

Activity

UI

的 的情况下很容易地重置你的 元素。

Caution: Beginning with Android 3.2 (API level 13), the "screen size" also changes when the device switches between portrait and landscape orientation. Thus, if you want to prevent runtime restarts due to orientation change when developing for API level 13 or higher (as declared by the minSdkVersion and targetSdkVersion attributes), you must include the "screenSize" value in addition to the "orientation" value. That is, you must declare

android:configChanges="orientation|screenSize". However, if your application targets API level 12 or lower, then your activity always handles this configuration change itself (this configuration change does not restart your activity, even when running on an Android 3.2 or higher device).

例如，接下来的onConfigurationChanged()方法中实现了检查硬件键盘的可用性和当前设备的方向：

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Checks the orientation of the screen
    if (newConfig.orientation ==
        Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape",
        Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation ==
        Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait",
        Toast.LENGTH_SHORT).show();
    }
}
```

这个Configuration类的对象代表着当前所有的配置信息，不仅仅那些改变的配置信息。在很多时候，你不会确切地在乎这些配置是怎么改变的，并且可以简单地重新分配所有资源，提供您正在处理的配置的可替代资源。例如，因为这个Resources对象现在被更新，你可以通过setImageResource(int)方法重置任何ImageView，并重置恰当的资源给当前配置使用。（详见：Providing Resources）

请注意，配置字段的值是一些匹配**Configuration**类里特定的常量的整数。对于文档中的每个字段使用那个常量，请在**Configuration**类中参阅相应的字段。

记住：当你声明你的**Activity**来处理配置的变化时，你负责重置所有你提供可替代资源的元素。如果你声明你的**Activity**来处理屏幕方向的改变并具有在横向和纵向之间切换的图像，你必须在**onConfigurationChanged()**方法中给每个元素重新指定一个资源。

如果你不需要根据配置的变化来更新你的程序，你可以不实现**onConfigurationChanged()**方法。在这种情况下，所有在配置改变之前使用的资源仍然会被使用，并且你只需要避免你的**Activity**被重启。然而，您的应用程序应该始终能够关闭并从其之前的状态完好地重新启动。这不仅是因为存在有一些配置发生改变时你不能防止它重新启动您的应用程序，而且为了处理一些事件，例如当用户接收了电话然后返回到应用程序。

更多关于哪些配置变化时你可以在你的**Activity**中处理，参见**android:configChanges**文档和**Configuration**类。

来自“[index.php?title=Handling_Runtime_Changes&oldid=13869](#)”



Localization

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： CuGBabyBeaR

原文链

接：<http://docs.eoeandroid.com/guide/topics/resources/localization.html>

目录

[[隐藏](#)]

1 本地化

- [1.1 概览:Android的资源切换](#)
- [1.2 默认资源为什么很重要](#)
- [1.3 用资源文件来进行本地化](#)
 - [1.3.1 如何创建默认资源](#)
 - [1.3.2 如何创建替换用资源](#)
 - [1.3.3 哪组资源将获得优先权?](#)
 - [1.3.4 在Java代码中引用资源](#)
- [1.4 本地化策略](#)
 - [1.4.1 设计您的应用能够在任何语言设置下工作](#)
 - [1.4.2 设计一个灵活的布局](#)
 - [1.4.3 避免创建过多的资源文件](#)
 - [1.4.4 使用Android Context对象手动获取区域设置](#)
- [1.5 本地化应用的测试](#)
 - [1.5.1 在设备上测试](#)
 - [1.5.2 在模拟器上测试](#)
 - [1.5.3 创建和使用一个自定义区域设置](#)
 - [1.5.4 在ADB shell中改变模拟器的区域设置](#)
 - [1.5.5 默认资源的测试](#)

本地化

Android系统会在很多地区的不同设备上运行。为了能够将应用提供给更多的用户，您的应用应该以不同的方式处理文本、音频文件、数字、货币和图形，以适合于您应用的使用地点。

本文档描述了本地化Android应用最好的实例。原则上允许您用Eclipse、Ant-based tools、或者其他任何IED搭载ADT来开发Android应用。

您应该已经有了一些Java的工作经验并熟悉Android资源的加载，熟悉在XML中声明用户界面元素，了解一些开发的知识（如Activity的生命周期），了解国际化和本地化的基本原则。

这将是使用Android资源框架来尽可能的分离您应用本地化过的方面和Java功能性代码的一个好的实践：

- 您可以将您应用的用户界面大部分或所有目录放在您的资源文件夹下，正如本文和[Providing Resources](#)中描述的那样。
- 另一方面来说，用户界面的行为是由您的Java代码驱动的。例如如果用户输入的数据需要格式化或者基于本地选择，您就需要用Java以程序的形式储存这些数据。本文不会涉及如何本地化您的Java代码。

[L10N](#) 教程将会一步步教会你以本文描述的方法，使用本地化资源创建一个简单的本地化应用。

概览:Android的资源切换

资源包括文本的字符串、布局文件、声音、图片，以及其他任何您的Android应用所需的静态数据。一个应用可以包含多套资源，每套为不同的设备设置定义。当用户运行应用的时候，Android自动选择并加载最适合设备的资源。

（本文档着眼于本地化和区域设置。更多有关资源切换和您可以指定的所有配置类型信息，例如屏幕方向、触摸屏类型等，请查阅[Providing Alternative Resources](#)。）

您在写应用时： 创建一组默认资源，加上在其他不同区域设置下使用的可替换资源。

用户运行您的应用时： Android系统基于设备区域设置选择加载哪些资源。

当您在写应用时，您应创建应用使用到的，默认和替换用的资源。为了创建这些资源，您需要将文件放在项目的res/目录下特定的子文件夹中。

默认资源为什么很重要

当应用运行在您没有定义本地文本的某个区域设置时，Android会从res/values/strings.xml中加载默认字符串。如果默认文件不存在，或者没有您应用所需的某个字符串时，您的应用将不会运行，并显示一个error。下述的例子将说明当默认文本文件不完全时会发生什么。

例：一个应用的Java代码只引用了两个字符串，text_a和text_b。这个应用包含了一个以英文定义text_a和text_b的本地化资源文件(res/values-en/strings.xml)。另外还包含了一个定义了text_a但未定义text_b的默认资源文件(res/values/strings.xml)。

- 这个应用可能编译通过。如果资源不存在，Eclipse等IDE不会提醒此错误。
- 当这个应用在区域设置为英文的设备上运行时，或许也不会出现任何问题，因为res/values-en/strings.xml包含所有所需的字符串。
- 但是，当这个应用在语言设置为英文以外的设备上运行时，会看见一个错误信息并出现一个Force Close按钮。这个应用不会加载。

为了预防这种情况，请确保res/values/strings.xml文件存在，并定义了您所需的所有字符串。这种情况适用于所有类型的资源，而不仅限于字符串：您需要创建一组包含所有您应用调用到的默认资源文件，如布局，图片，动画等。有关调试的信息，请浏览[默认资源的测试](#)。

用资源文件来进行本地化

如何创建默认资源

将用于应用的默认文本放在这个文件中并置于如下路径：

`res/values/strings.xml` (必须拥有此目录)

`res/values/strings.xml`中的文本字符串必须使用默认语言，即为您预期您的用户最经常说的语言。

默认资源必须同样包括所有默认的图片、布局，并能包含其他类型的资源，例如动画。

`res/drawable/` (必须拥有此目录，必须包含至少一个用于Google Play上应用图标的图形文件。)

Tip: 检查您代码中对每个Android资源的引用。确信每一个引用都有默认资源定义它。同样保证默认字符串是完整的：一个本地化的字符串文件可以是所有字符串的子集，但是默认字符串文件必须包含所有的字符串。

如何创建替换用资源

应用本地化的很大一部分工作是为不同的语言提供替换用的文本。在某些情况下，您同样需要提供替换用的图形、声音、布局、以及其他本地化的资源。应用可以为不同的修饰符指定很多`res/<qualifiers>/`文件夹。为了向不同语言提供替换用的资源，您需要使用一个指定为一种语言或者语言-区域组合的修饰符。（资源文件夹的名字必须遵从[Providing Alternative Resources](#)中描述的命名方法，否则这些将不会被编译。）

例：假定您应用的默认语言是英语。同样假定您想将您应用中所有的文本翻译成法语，并将应用中的大部分文本（除开应用标题之外的作用东西）翻译为日语。这样的话，您应该建立3个可选`strings.xml`文件，每个都存储在本地化资源文件夹中：

1. `res/values/strings.xml` 包含应用用到的所有字符串的英语文本，包

括一个命名为title的字符串。

2. res/values-fr/strings.xml 包含所有字符串的法语文本，包括标题。
3. res/values-ja/strings.xml 包含除开标题之外，所有字符串的日语文本。

如果Java代码提到R.string.title，如下将是在运行时会发生的事情：

- 如果一个设备设置为法语以外的其他任何语言，Android会从res/values/strings.xml文件中加载标题。
- 如果设备设置为法语，Android会从res/values-fr/strings.xml文件中加载标题。

值得注意的是，如果设备设置为日语，Android会在res/values-ja/strings.xml文件中寻找标题。但是因为这个文件中没有此字符串，Android会回到默认情况，从res/values/strings.xml文件中加载英文标题。

哪组资源将获得优先权？

如果多个资源文件都匹配设备的配置，Android会遵循一组规则来决定使用哪个文件。通过资源目录名中定义的修饰符，区域设置通常总是会获得优先权。例：假设一个应用包含了一套默认的图像和另外两套图像，每一套都为不同的设备设置优化：

- res/drawable/

包含默认图像

- res/drawable-small-land-stylus/

包含为预期使用stylus输入，并有一个横向的低密度QVGA屏幕设备优化的图像。

- res/drawable-ja/

包含为日语优化的图像。

如果应用在配置为使用日语的设备上运行时，即使这个设备使用的是stylus输入并且有一个横向的低密度QVGA屏幕，Android也会从res/drawable-ja/中加载图像。

例外：优先级高于区域设置的唯二修饰符是MCC和MNC（手机国家码和手机网络码）。

例：假设您有如下情形：

- 应用代码调用R.string.text_a
- 两个相关资源文件可用：
 - res/values-mcc404/strings.xml 包含了使用应用默认语言的text_a，本例为英语。
 - res/values-hi/strings.xml 包含了使用印地语的text_a。
- 应用在使用如下配置的设备上运行：
 - SIM卡在印度连接移动电话网络(MCC 404)。
 - 语言设置为印地语(hi)。

即使设备配置为印地语，Android会从from res/values-mcc404/strings.xml (英文) 中加载text_a。这是因为资源选择过程中，Android会将MCC匹配的优先级提升超过语言匹配。

选择进程并不是始终和本例中显示的那么直接。请阅读Android Finds the Best-matching Resource一节以获得资源选择进程中更为细节的描述。所有的修饰符都在Table 2 of Providing Alternative Resources中以优先权顺序描述并列出了。

在Java代码中引用资源

在您应用的Java代码中，您可以以R.resource_type.resource_name 或者是android.R.resource_type.resource_name的语法来调用资源。获得于此相关的更多信息，请浏览[Accessing Resources](#)。

本地化策略

设计您的应用能够在任何语言设置下工作

您不能假设任何有关用户拿来运行您应用的设备的任何东西。这个设备可能拥有您无法预测的硬件，或有可能将区域设置为一个您没有预想过或者您无法测试的地区。请设计您的应用能够正常实现功能或者软性的失败，而与应用运行在什么设备无关。

重要：请保证您的应用包含了一套完整的默认资源。

请保证您的应用包含`res/drawable/`和`res/values/`文件夹（文件夹名不要加任何额外的修饰符），这些文件夹包括了您的应用所需的所有图片和文本。如果一个应用丢失了哪怕是一个默认资源，它都不会在设置为不支持区域的设备上运行。例如，`res/values/strings.xml`默认文件可能缺少一个应用需要的字符串：当应用在不支持区域上运行并试图加载`res/values/strings.xml`时，用户将会看见一个错误信息和一个**Force Close**按钮。像Eclipse那样的IDE不会提醒这种错误，并且当您在设置为受支持区域设置的设备或者仿真机上测试您的应用时不会发现这个问题。获取更多信息请浏览[Localization#默认资源的测试](#)。

设计一个灵活的布局

如果您需要为一个已存在的语言（例如带有长单词的德语）重新设计您的布局文件，您可以为此种语言创建一个替换用的布局文件（例如`res/layout-de/main.xml`）。但是这会使您的应用难于维护。更好的方法是只建立一个更加灵活布局文件。

另一个典型的情况是有的语言需要在布局文件中请求其他不同的资源。例如，您可能有一个表单，当应用在日语环境下运行时应该包括两个字段的名字空间，但是在其他语言环境下运行需要三个字段的名字空间。您可以用如下两种方法中的一种来实现：

- 创建一个布局文件，使之含有一个可以用程序控制显示或者不现实的输入空间。或者
- 让主要布局文件包含其他的具有可变输入空间的布局文件。被包含的布局文件根据不同的语言有不同的配置。

避免创建过多的资源文件

您或许并不需要在您的应用中为每一个资源都创建一个本地化的替换资源。例如，在`res/layout/main.xml`文件中定义的布局应该能够用在任何区域设置下，这样您就不必再创建一个替换用的布局文件。

同样的，您可能不必为所有的字符串都创建替换文本。例如，如下假设：

- 您的应用的默认语言是美国英语。应用用到的每一个字符串都用美国英语的拼写，定义在`res/values/strings.xml`文件中
- 您想为几个重要的短语提供英国英语的拼写。您想让您的应用在英国的设备上运行时使用这些字符串。

为了达到您的目的，您可以创建一个叫做`res/values-en-rGB/strings.xml`的小文件，仅包括应用在英国运行时需要的不同的字符串。而其他没有定义的字符串，应用将会回到默认语言，并使用定义在`res/values/strings.xml`中的字符串。

使用**Android Context**对象手动获取区域设置

您可以使用Android提供的Context对象获取区域设置：

```
String locale =
context.getResources().getConfiguration().locale.getDisplayName();
```

本地化应用的测试

在设备上测试

注意您用来测试的设备的可能与其他地区顾客使用的设备完全不同。您设备的可用区域设置可能与其他的设备不同。同样，设备屏幕的分辨率和密度也可能不同，这可能会影响您的UI上字符串和图片的显示。

请使用Settings应用来改变设备的区域设置(Home > Menu > Settings > Locale & text > Select locale)。

在模拟器上测试

有关使用模拟器的详细信息，请浏览[See Android Emulator](#)。

创建和使用一个自定义区域设置

一个自定义区域设置是一个**android**系统镜像并没有明确支持的“语言/区域”组合。（在SDK标签中的**Version Notes**获取Android平台支持的区域设置列表）。您可以通过在模拟器上创建一个自定义区域设置，来测试您的应用是如何在自定义区域设置下工作的。有两种方法来做到：

- 使用应用列表中提供的**Custom Locale**应用。（当您创建一个自定义区域设置后，长按名字以切换到此区域设置项。）
- 在**adb shell**中切换到一个自定义区域设置，方法在下一节

当您将模拟器设置为一个Android系统镜像中不存在的区域设置时，系统自身会显示为他自己的默认语言。但是，您的应用应该完全本地化了。

在**ADB shell**中改变模拟器的区域设置

通过使用**ADB shell**来改变模拟器的区域设置。

1. 选择您想测试的区域设置，并确定它的语言和地区码，例如**fr**是法语而**CA**是加拿大。
2. 启动模拟器。
3. 在主机命令行中运行如下命令：

adb shell

或者您有一个附属设备，请通过添加一个-e参数来指定：

adb -e shell

4. 当**adb shell**显示#的时候，运行如下命令：

setprop persist.sys.language [language code];setprop persist.sys.country [country code];stop;sleep 5;start
用第一步中决定的代码替换方括号。

例如，在加拿大法语环境下测试：

```
setprop persist.sys.language fr;setprop persist.sys.country
CA;stop;sleep 5;start
```

这会导致模拟器重启（这可能看起来像完全的冷重启，但它不是。）当桌面重新出现时，重新运行您的应用（例如，在**Eclipse**中点击**Run**图标），这样您的应用会在新的区域设置下运行。

默认资源的测试

这是测试应用是否包含了所有所需字符串资源的方法。

1. 设置模拟器或者设备为您的应用所不支持的语言。例如，如果应用在res/values-fr/文件夹中有法语字符串，但res /values-es/中没有任何西班牙语字符串，那就将模拟器的语言设置为西班牙语。（您可以使用Custom Locale应用来将模拟器设置为不支持的区域。）
2. 运行应用
3. 如果应用显示了一个错误信息和一个Force Close按钮，这可能表示不能加载某个字符串。请保证res/values/strings.xml文件拥有应用用到的所有字符串。

如果这个测试通过，在其他类型的设置下重复此测试。例如，如果应用拥有一个叫做res/layout-land/main.xml的布局文件，但不包含叫做res/layout-port/main.xml的文件，然后设置模拟器或者设备为portrait，试试这个应用是否能够运行。

来自“[index.php?title=Localization&oldid=13870](#)”



Resource Types

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://docs.eoeandroid.com/guide/topics/resources/available-resources.html>

作者：qishanmingfeng

时间：2012年9月14日

Resource Types(资源类型)

这一章节的每篇文档描述了一类应用资源的用法，格式和语法。你可以把这些资源放在你的资源目录下（即res/）。

下面是对每个资源类型的简介：

动画资源 [Animation Resources](#):

定义预设的动画。

补间动画被保存在res/anim目录下且可通过R.anim这个类访问它。

帧动画保存在res/drawable/目录下，可通过R.drawable这个类访问它。

颜色状态列表资源 [Color State List Resource](#)

定义了一个根据视图状态改变的颜色资源。

保存在res/color，通过R.color这个类访问。

可绘制的资源 [Drawable Resources](#)

用位图或XML定义各种各样的图像。

保存在res/drawable/目录下，通过R.drawable这个类访问。

布局资源 [Layout Resource](#)

定义你应用程序用户界面的布局。

保存在res/layout/目录下，通过R.layout这个类访问。

菜单资源 [Menu Resource](#)

定义你的应用程序的菜单内容。

保存在res/menu目录下，通过R.menu这个类访问。

字符串资源 [String Resources](#)

定义字符串，字符串数组和复数（字符串的格式和风格）。

保存在res/values目录下，通过R.string、R.array和R.plurals这几个类访问。

风格资源 [Style Resource](#)

定义用户界面的元素的格式和外观。

保存在res/values目录下，通过R.style这个类访问

更多的资源类型 [More Resource Types](#)

定义各种各样的值，如布尔值，整数值，数量值，颜色值，和其他数组。

保存在res/values下，且每个都用特有的R的子类访问
(如R.bool,R.integer,R.dimens等等)。

来自“[index.php?title=Resource_Types&oldid=11620](#)”



Animation

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

动画资源可以定义两类动画之一： 动画属性

在设定时间内用 **Animator**改变对象的属性值就可以创建动画。

动画视图

可以用视图动画框架实现两种类型的动画：

- **Tween animation**: 通过 **Animation**对单个图像执行一系列变换来创建动画。
- **Frame animation**: 用**AnimationDrawable**显示图像序列来创建动画。

目录

[[隐藏](#)]

[1 动画属性](#)

[2 视图动画](#)

- [2.1 Tween动画](#)
- [2.2 Interpolators](#)
- [2.3 自定义 interpolator](#)
- [2.4 帧动画](#)

动画属性

XML所定义的动画用于修改超过设定时间的目标对象的性质，比如背景颜色或者字母值。

文件位置: res/Animator/filename.xml

文件名将作为资源ID。

编译的资源数据类型: 指向ValueAnimator, ObjectAnimator, 或AnimatorSet的资源。

资源引用:

在Java中: R.animator.filename

在XML中: @[package:]animator/filename

SYNTAX:

```

<set
    android:ordering= [ "together" | "sequentially" ] >

    <objectAnimator
        android:propertyName= "string"
        android:duration= "int"
        android:valueFrom= "float | int | color"
        android:valueTo= "float | int | color"
        android:startOffset= "int"
        android:repeatCount= "int"
        android:repeatMode= [ "repeat" | "reverse" ]
        android:valueType= [ "intType" | "floatType" ] />

    <animator
        android:duration= "int"
        android:valueFrom= "float | int | color"
        android:valueTo= "float | int | color"
        android:startOffset= "int"
        android:repeatCount= "int"
        android:repeatMode= [ "repeat" | "reverse" ]
        android:valueType= [ "intType" | "floatType" ] />

    <set>
        ...
    </set>
</set>
```

文件必须有根元素: <set>, <objectAnimator>, 或<valueAnimator>。可以把动画元素包括其它<set>元素整合在<set>元素中。

元素:

<set>

存放其它动画元素的container (<objectAnimator>, <valueAnimator>, 或者其他 <set>元素)。代表AnimatorSet。
可以指定嵌套<set>标签进一步把动画组合在一起。每个<set>可以定义自己的排序属性。

属性：

android:ordering

关键字。指定这组动画的播放顺序。

<objectAnimator>

Animates a specific property of an object over a specific amount of time. 代表ObjectAnimator。

属性：

android:propertyName

字符串。必要。设置动画的对象属性，由名称来引用。例如可以为视图对象指定“alpha”或者“背景颜色”。 objectAnimator元素不会公开目标属性，因此不能在XML声明中设置动画处理的对象。可以调用loadAnimator()来扩展动画XML 资源，同时调用setTarget()来设置包含此属性的目标对象。

android:valueTo

浮点型，整型或颜色。必备。动画属性结束位置的值。颜色用6个十六进制数值来表示（比如#333333）。

android:valueFrom

浮点型，整型或者颜色。动画属性开始位置的值。如果没有指定，则动画会从属性get方法值处开始。颜色用6位十六进制数字来表示（比如 #333333）。

android:duration

整型。动画毫秒时间。默认值为300毫秒。

android:startOffset

整型。调用start () 之后动画延迟的毫秒数。

android:repeatCount

整型。重复动画的次数。设置 "-1" 来无限重复或者设置正整数。例如“1”代表动画初始运行后重复了一次，所以动画一共播放了两次。默认值是“0”，表示没有重复。

android:repeatMode

整型。动画到达结尾时如何表现。 android:repeatCount必须设置为正整数或者"-1"。设置“反向”可以让动画每次迭代时反方向，设置“重复”可以让动画每次开始时循环。

android:valueType

关键字。如果该值为颜色的话就不会指定这一属性。动画框架自动处理颜色值。

<animator>

在指定时间量内执行动画。代表ValueAnimator。

属性：

android:valueTo

浮点型，整型或者颜色。必备。动画结束处的值。颜色用六位十六进制数值来表示（比如 #333333）。

android:valueFrom

浮点型，整型或者颜色。必备。动画开始处的值。颜色用六位十六进制数值来表示（比如 #333333）。

android:duration

整型。动画毫秒时间。默认值为300毫秒。

android:startOffset

整型。调用start () 之后动画延迟的毫秒数。

android:repeatCount

整型。重复动画的次数。设置 "-1" 来无限重复或者设置正整数。例如 "1" 代表动画初始运行后重复了一次，所以动画一共播放了两次。默认值是 "0"，表示没有重复。

android:repeatMode

整型。动画到达结尾时如何表现。 android:repeatCount 必须设置为正整数或者 "-1"。设置 "反向" 可以让动画每次迭代时反方向，设置 "重复" 可以让动画每次开始时循环。

android:valueType

关键字。如果该值为颜色的话就不会指定这一属性。动画框架自动处理颜色值。

示例：

XML文件保存在res/animator/property_animator.xml:

```
<set android:ordering="sequentially">
<set>
    <objectAnimator
        android:propertyName="x"
        android:duration="500"
        android:valueTo="400"
        android:valueType="intType" />
```

```

<objectAnimator
    android:propertyName= "y"
    android:duration= "500"
    android:valueTo= "300"
    android:valueType= "intType" />
</set>
<objectAnimator
    android:propertyName= "alpha"
    android:duration= "500"
    android:valueTo= "1f" />
</set>

```

为了运行这个动画，必须扩展代码的XML资源，然后动画设置启动前要为所有动画设置目标对象。调用`setTarget()`为`AnimatorSet`所有子类设置单个目标对象。下列代码显示了具体做法：

```

AnimatorSet set = ( AnimatorSet )
AnimatorInflater.loadAnimator( myContext ,
    R.anim.property_animator );
set.setTarget( myObject );
set.start();

```

视图动画

视图动画框架支持tween和帧动画，这两种都可以在XML中声明。下列部分描述了如何使用这两种方法。

Tween动画

XML定义的动画执行过渡，比如图像旋转，褪色，移动和伸缩。

文件位置

`res/anim/filename.xml`

文件名会用作资源ID。

编译的资源数据类型：

资源指向`Animation`。

资源引用：

Java: `R.anim.filename`

XML: `@[package:]anim/filename`

语法：

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=[ "true" | "false" ] >
    <alpha>
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale>
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate>
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate>
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
    <set>
        ...
    </set>
</set>

```

文件必须有单个根目录：<alpha>, <scale>, <translate>, <rotate>, 或者<set>元素。

元素：

<set>

存有其他动画元素(<alpha>, <scale>, <translate>, <rotate>)的container或者其他<set>元素。代表AnimationSet。

属性：

android:interpolator

内插资源。适用于动画的分类。该值必须是引用到指定分类的资源。平台默认的可用分类资源或者创建自己的分类资源。请参阅下方更多信息。

android:shareInterpolator

布尔型。如果要分享所有字元素的相同分类，“正确”。

<alpha>

淡入或淡出动画。代表AlphaAnimation。

属性：

android:fromAlpha

浮点型。起点不透明度偏移，其中0.0是透明，1.0是不透明。

android:toAlpha

浮点型。终点不透明度偏移，其中0.0是透明，1.0是不透明。

<scale>

调整尺寸的动画。可以通过指定pivotX和pivotY从向外（内）增长位置指定图像中心点。例如，如果这些值为0.0（左上角），所有增长会向下及向右。代表ScaleAnimation。

属性：

android:fromXScale

浮点型。启动X尺寸偏移，其中1.0没有任何变化。

android:toXScale

浮点型。终止X尺寸偏移，其中1.0没有任何变化。

android:fromYScale

浮点型。启动Y尺寸偏移，其中1.0没有任何变化。

android:toYScale

浮点型。终止Y尺寸偏移，其中1.0没有任何变化。

android:pivotX

浮点型。对象标注时保持X坐标不变。

android:pivotY

浮点型。对象标注时保持Y坐标不变。

<translate>

垂直或水平运动。三种格式任何一种都支持下列属性：值从-100到100用“%”结尾，用来表示相对自身的百分比。值从-100到100用“%p”结尾，用来表示父类的相对百分比。不带后缀的浮点型，表示绝对值。代表TranslateAnimation。

属性：

android:fromXDelta

浮点或百分比。启动X偏移。用下列任一来表示：相对于正常位置（如“5”）的像素，元素宽度的相对百分比（比如“5%”）或者父类宽度的相对百分比（比如“5%p”）。

android:toXDelta

浮点或百分比。终止X偏移。用下列任一来表示：相对于正常位置（如“5”）的像素，元素宽度的相对百分比（比如“5%”）或者父类宽度的相对百分比（比如“5%p”）。

android:fromYDelta

浮点或百分比。启动Y偏移。用下列任一来表示：相对于正常位置（如“5”）的像素，元素宽度的相对百分比（比如“5%”）或者父类宽度的相对百分比（比如“5%p”）。

android:toYDelta

浮点或百分比。终止Y偏移。用下列任一来表示：相对于正常位置（如“5”）的像素，元素宽度的相对百分比（比如“5%”）或者父类宽度的相对百分比（比如“5%p”）。

<rotate>

旋转动画。代表RotateAnimation。

属性：

android:fromDegrees

浮点。初始角度位置，以度数为单位。

android:toDegrees

浮点。终止角度位置，以度数为单位。

android:pivotX

浮点或者百分比。旋转中心的X坐标。用下面任一来表示：对象左边界的相对像素（比如“5”），对象左侧的相对百分比（比如“5%”），或者父类container的左侧相对百分比（比如“5%p”）。

android:pivotY

浮点或者百分比。旋转中心的Y坐标。用下面任一来表示：对象左边界的相对像素（比如“5”），对象左侧的相对百分比（比如“5%”），或者父类container的左侧相对百分比（比如“5%p”）。

示例：

XML文件保存在res/anim/hyperspace_jump.xml:

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale
```

```
Animation - eoeAndroid wiki
android:interpolator="@android:anim/accelerate_decelerate_interpolator

    android:fromXScale="1.0"
    android:toXScale="1.4"
    android:fromYScale="1.0"
    android:toYScale="0.6"
    android:pivotX="50%"
    android:pivotY="50%"
    android:fillAfter="false"
    android:duration="700" />
<set
    android:interpolator="@android:anim/accelerate_interpolator"
    android:startOffset="700">
    <scale
        android:fromXScale="1.4"
        android:toXScale="0.0"
        android:fromYScale="0.6"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="400" />
    <rotate
        android:fromDegrees="0"
        android:toDegrees="-45"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="400" />
</set>
</set>
```

应用程序代码会把动画应用于 ImageView 并启动动画：

```
ImageView image = ( ImageView ) findViewById( R.id.image );
Animation hyperspaceJump = AnimationUtils.loadAnimation( this ,
R.anim.hyperspace_jump );
image.startAnimation( hyperspaceJump );
```

Interpolators

Interpolator 是 XML 定义的动画修改，会影响动画的变化速率。这样会加速，减速，重复，回升现有的动画效果。

Interpolator 应用于具有 android:interpolator 属性的动画元素，该值引用了 interpolator 资源。

安卓所有可用的 interpolator 都是 Interpolator 类的子类。对于每一个 interpolator 类，安卓包含了供参考的公共资源，可以把 interpolator 应用于具有 android:interpolator 属性的动画。下面显示了如何把这些应用于 android:interpolator 属性：

Animation - eoeAndroid wiki
<set android:interpolator="@android:anim/accelerate_interpolator">
...
</set>

自定义 interpolator

如果不满意平台提供的 interpolator，可以用修改属性来创建自定义 interpolator资源。例如可以调整 AnticipateInterpolator加速的速度，或者调整CycleInterpolator周期数。要做到这一点，需要在XML文件创建自己的interpolator资源。

文件位置：

res/anim/filename.xml

文件名会作为资源ID。

编译的资源数据类型：

资源指向对应的内插对象。

资源引用

XML: @+[package:]anim/filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<InterpolatorName
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:attribute_name="value"
/>
```

如果不应用任何属性，那么interpolator会和平台提供的那些进行相同操作。

元素：

请注意XML定义时每个 Interpolator的实现，名称会用小写字母表示。

<accelerateDecelerateInterpolator>

变化速率缓慢开始及结束，但是中间会加速。

没有属性。

<accelerateInterpolator>

变化速率缓慢开始，然后加速。

:

属性

android:factor

浮点型。加速速率（默认为1）。

<anticipateInterpolator>

变化起初后退，然后向前。

属性：

android:tension

浮点型。运用的tension量（默认为2）。

<anticipateOvershootInterpolator>

变化起初后退，甩向前方超过目标值，然后稳定在最终值。

属性：

android:tension

浮点。Tension使用量（默认为2）。

android:extraTension

浮点。与tension相乘的数量（默认为1.5）。

<bounceInterpolator>

变化在结尾处反弹。

没有属性。

<cycleInterpolator>

根据指定周期数来重复动画。变化率遵循正弦模式。

属性：

android:cycles

整型。周期数（默认为1）。

<decelerateInterpolator>

变化率起始快速，然后减慢。

属性：

android:factor

浮点型。减速速率（默认为1）。

<linearInterpolator>

变化率是恒定的。
没有属性。

<overshootInterpolator>

变化快速向前，然后超过终值，接着返回。

属性：

android:tension

浮点型。Tension申请量（默认为2）。

示例：

XML文件保存在res/anim/my_overshoot_interpolator.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<overshootInterpolator
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:tension="7.0"
/>
```

这一动画XML会应用于 interpolator:

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@anim/my_overshoot_interpolator"
    android:fromXScale="1.0"
    android:toXScale="3.0"
    android:fromYScale="1.0"
    android:toYScale="3.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="700" />
```

帧动画

XML定义的动画显示了图像序列（比如film）。

文件位置：

res/drawable/filename.xml

文件名用作资源ID。

编译的资源数据类型：

资源指向**AnimationDrawable**。

资源引用：

Java: R.drawable.filename

XML: @[package:]drawable.filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot=[ "true" | "false" ] >
    <item
        android:drawable="@[package:]drawable/drawable_resource_name"
        android:duration="integer" />
</animation-list>
```

元素：

<animation-list>

必备。必须是根元素。包含一个或多个<item>元素。

属性：

android:oneshot

布尔型。如果要执行一次动画，“真”；循环动画“假”。

<item>

动画的一帧。必须是<animation-list>元素的子元素。

属性：

android:drawable

可绘制资源。使用该框架的绘制。

android:duration

整型。显示框架的持续时间，以毫秒为单位。

示例：

XML文件保存在res/anim/rocket.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/rocket_thrust1"
        android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2"
        android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3"
        android:duration="200" />
</animation-list>
```

这一应用程序代码会把动画设置为视图的背景，然后播放动画：

```
ImageView rocketImage = ( ImageView ) findViewById( R.id.rocket_image );
rocketImage.setBackgroundResource( R.drawable.rocket_thrust );

rocketAnimation = ( AnimationDrawable ) rocketImage.getBackground();
rocketAnimation.start();
```

来自“[index.php?title=Animation&oldid=13871](#)”



Color State List

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/resources/color-list-resource.html>

翻译：Johnsun

Color State List Resource (颜色状态资源列表)

一个[ColorStateList](#)你可以在XML中定义并且作一个color来使用的对象，但实际上它会根据所使用它的[View](#)对象的状态改变颜色。例如，一个[Button](#)控件可以存在几种不同的状态（`press`压，`focused`获得焦点，或`neither`常态<原，两者都不>），此时使用ColorStateList，就可以根据每个状态提供不同的颜色值来显示。

您可以在一个XML文件中，把每种颜色被定义在一个单独的`<selector>`标签内的`<item>`标记里面。每个`<item>`可使用各种属性来描述其对应的状态。在每个状态变化时，状态列表将从上到下，找到第一个符合当前的状态将就会被用上。也就是说，并不是基于用“最佳匹配”的算法来选择的，而仅仅遇到第一个符合最低标准的就会被使用。

注：如果你想提供一个静态的色彩资源，就用一个简单的Color值。即color.xml中。

- 文件的位置：

res/color/filename.xml

- 文件名filename 会被用作资源ID。
- 编译的资源数据类型：
 - 资源指向到一个ColorStateList。
- 资源引用：
 - In Java: R.color.filename <即在java代码中>
 - In XML: @[package:]color/filename <在xml代码中>

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android"
>
    <item
        android:color="hex_color"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

标签:

- <selector>
 - 必需的。这个必须是根元素，包含一个或多个<item>。
- 属性:
 - xmlns:android
 - String类型。必需的。用于定义xml的命名空间。其值为:
 - "<http://schemas.android.com/apk/res/android>"。
- <item>
 - 定义颜色的，就是根据你想处理的状态使用不同的属性值来显示对应的颜色。必须是<selector>的子元素。
 - 属性:
 - android:color
 - 十六进制颜色值。必需的。Color用指定RGB值和可选的alpha通道。

- 其value以“#”号开头，后面跟Alpha-Red-Green-Blue 信息，如下所示：

1. RGB
2. ARGB
3. RRGGBB
4. AARRGGBB

- **android:state_pressed**

- Boolean控件被压(触摸touched/点击clicked)的时候返回true。默认为false。

- **android:state_focused**

- Boolean控件获得焦点的时候返回true。默认为false。这里解释一下使用场景，<模拟器容易看到>比如使用滑轮或键盘的方向导航键导航到某一个控件上，则此控件就得到了焦点。当然其它的点击，选择，光标等也都有焦点跟随的。

- **android:state_selected**

- Boolean控件被选中的时候返回true。默认为false。比如一个tab被打开。

- **android:state_checkable**

- Boolean控件处于可选状态的时候返回true。默认为false。说明，这个属性和下面的checked属性仅仅用于那些在可选和不可选两种状态当中过渡的控件。

- **android:state_checked**

- Boolean控件被选中的时候返回true。默认为false。

- **android:state_enabled**

- Boolean控件可用的时候返回true。默认为false。

- **android:state_window.Focused**

- Boolean当应用的窗口处于聚焦状态时返回true，当窗口失去焦点时返回false（就是说应用窗口处于前台时返回true，处于后台返回false。比如通知罩（即通知栏被拉下后的半透时界面）被打开时或者一个对话框（dialog）出现的时候。后面的窗口就处于失焦状态。

注意：请记住，在状态列表的第一项的会被直接应用到当前状态。因此

如果第一项没有包含上述状态中对应的状态，才会从上到下进行匹配。每次都是如此，这也就是为什么default默认状态要写在最末尾的原因。如下示例：

例如：

XML文件保存在res/color/button_text.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:color="#ffff0000"/> <!-- pressed -->
    <item android:state_focused="true"
          android:color="#ff0000ff"/> <!-- focused -->
    <item android:color="#ff000000"/> <!-- default -->
</selector>
```

作用于下面的这个button

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:textColor="@color/button_text" />
```

其他参考：

- [ColorColor \(simple value\)](#)
- [ColorStateList](#)
- [State List Drawable](#)

来自“[index.php?title=Color_State_List&oldid=8907](#)”



Drawable

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Xiangrufeifei3

原文链接：<http://developer.android.com/intl/zh-CN/guide/topics/resources/drawable-resource.html>

目录

- [1 Drawable Resources\(可绘制的资源\)](#)
 - [1.1 Bitmap](#)
 - [1.2 Bitmap File\(位图文件\)](#)
 - [1.2.1 XML Bitmap\(XML位图\)](#)
 - [1.3 Nine-Patch \(可拉伸图片\)](#)
 - [1.3.1 Nine-Patch File](#)
 - [1.3.2 XML Nine-Patch](#)
 - [1.4 Layer list](#)
 - [1.5 State List](#)
 - [1.6 Level List](#)
 - [1.7 Transition Drawable](#)
 - [1.8 Inset Drawable](#)
 - [1.9 Clip Drawable](#)
 - [1.10 Scale Drawable](#)
 - [1.11 Shape Drawable](#)

Drawable Resources(可绘制的资源)

一个**drawable**资源,就是能被画到屏幕的一个一般的图形概念<简单说一下,**drawable**就是对图片向上抽取之后的一个抽象名称,图片有多种表现形式,接着看下文.文中一些地方还是直译为**drawable**.它可以用[getDrawable\(int\)](#)这个方法api或者将其应用到xml文件资源中,使用**android:drawable** and **android:icon**.这两个属性.

有如下多种图形概念.

[Bitmap File](#)

一个位图文件(.png,.jpg,或.gif),生成一个BitmapDrawable对象.

[Nine-Patch File](#)

就是一张可以基于自动适应内容大小而伸缩区域的png图片(.9.png),生成一个NinePatchDrawable对象

[Layer List](#)

这个Drawable用来管理一个其它多个**drawable**的数组.既然是一个数组,所以就不难理解索引值最大的元素将画在最高部.生成一个LayerDrawable对象.

[State List](#)

这是一个xml文件用于不同的状态来引用不同的位图图形(比如,当一个Button控件按下状态要显示不同的图像).生成一个StateListDrawable对象.

[Level List](#)

一个xml文件,定义了一个**drawable**可用于管理几个可以替换的**drawable**.每一个都会分配一个最大的数值.生成一个LevelListDrawable.

[Transition Drawable](#)

一个xml文件,定义了一个**drawable**可用于两张图片形成一个渐变的过程

渡效果生成一个TransitionDrawable对象

Inset Drawable

一个xml文件,定义了一个drawable,跟据指定的距离插入到另一个drawable.当一个View<视图>对象需要一张比其实际边框要小的背景图时,就可以用到这个了.

Clip Drawable

一个xml文件,定义了一个drawable, 根据当前对准值作相应的拉伸处理,生成 ClipDrawable对象.

Scale Drawable

一个xml文件,定义了一个drawable, 根据当前对准值作相应的平铺处理,生成 ScaleDrawable对象.

Shape Drawable

就是通过一个xml文件来定义一个包含颜色和渐变的几何图形, 生成一个 ShapeDrawable对象

- 另见Animation Resource <动画资源>文档,学习如何创建一个AnimationDrawable对像.

注：在xml中一个color resource<颜色资源>也可以作为一个drawable.例如,创建一个state list drawable时,你可以为android:drawable属性引用一个颜色资源(android:drawable="@color/green").

Bitmap

Android 支持三种格式的位图文件.png(推荐), .jpg(可以接受), .gif(不建议使用). 可以直接用文件名来引用一个位图文件,或在xml中为其创建一个资源id的别名.

注意: 在编译过程中Bitmap文件可能使用aapt工具优化为无损压缩图像, 例如, 一个不需要256色的真彩PNG图片可能转化为附有调色板的8-bitPNG图片。这会产生同等质量的图片, 但只需要较少的内存。所以需要明白在此路径下的图片在编译中会改变。如果打算按bit流读取图片以转换成位图, 把图片放在res/raw/文件夹, 这不会被优化。

Bitmap File(位图文件)

一个位图文件就是一个.png, .jpg, 或 .gif文件. 当你把这些文件放在res/drawable/ 中, Android就会为其创建一个 Drawable (可绘制的)资源.

- 文件位置:

res/drawable/filename.png (.png, .jpg, or .gif)
文件名被用作资源id.

- 编译后的资源数据类型:

资源将生成一个BitmapDrawable对象.

- 资源引用:

用java: R.drawable.filename
用xml: @[package:]drawable/filename

- 例如:

当一个图片保存在res/drawable/myimage.png ,下面就是xml来使用此图到一个View上:

```
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

- 下面的应用代码是将一个图片恢复为一个Drawable对象：

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

另见：

- - 2D Graphics
 - BitmapDrawable

XML Bitmap(XML位图)

XML Bitmap是定义在xml文件中，指向一个位图的资源文件。这种作用对于原始的位图文件尤其有效。此XML可以设定抖动，拼接等位图的附加属性。

注意：可以使用<bitmap>成员作为<item>成员的子成员，例如：当创建一个statelist(状态列表)或者layerlist(图层列表)，可以从<item>元素中去除android:drawable属性，而在<item>中构建一个<bitmap>来定义此绘制项(drawable item)。

- 文件位置：

res/drawable/filename.xml 以此文件名作为标识资源的ID

- 编译过后的资源数据类型：

一个指向BitmapDrawable的资源指针

- 资源引用：
- Java: R.drawable.filename
- XML: @[package:]drawable/filename
- 语法：

```

<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:antialias=[ "true" | "false" ]
    android:dither=[ "true" | "false" ]
    android:filter=[ "true" | "false" ]
    android:gravity=[ "top" | "bottom" | "left" | "right" |
"center_vertical" | "fill_vertical" | "center_horizontal" |
"fill_horizontal" | "center" | "fill" | "clip_vertical" |
"clip_horizontal" ]
    android:tileMode=[ "disabled" | "clamp" | "repeat" | "mirror" ] />
```

- 元素：
- <bitmap>

定义位图资源和特性。 属性： `xmlns:android: String`. 定义XML的命名空间， 必须是：["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android), 只有当此<bitmap>是一个根元素时才要求， 当此<bitmap>是构筑在一个<item>中此要求是不必要的。

- `android:src`

可绘制资源， 必需项， 引用一个可绘制资源。

- `android:antialias`

布尔型， 是否允许平滑效果。

- `android:dither`

布尔型， 如果位图与屏幕的像素配置不同时， 是否允许抖动. (例如：一个位图的像素设置是 ARGB 8888， 但屏幕的设置是RGB 565)

- `android:filter`

布尔型。 是否允许对位图进行滤波。 对位图进行收缩或者延展使用滤波可以获得平滑的外观效果。

- `android:gravity`

关键字。定义位图的**gravity**，如果位图小于其容器，使用**gravity**指明在何处绘制。

取值必须是如下常量值中的一个或多个（以分隔‘|’）：

取值	描述
top	把对象放在容器的上方，不改变其大小。
bottom	把对象放在容器的底部，不改变其大小。
left	把对象放在容器的左边，不改变其大小。
right	把对象放在容器的右边，不改变其大小。
center_vertical	把对象放在容器的垂直居中位置，不改变其大小。
fill_vertical	对象垂直填充容器。
center	把对象放在容器的水平和垂直的居中位置，不改变其大小。
fill	同时水平和垂直拉伸对象以填充容器，此为默认项。
clip_vertical	额外选项，可以设置使得子对象的顶和/或底上剪切至容器的边界，这个切割基于垂

	直 gravity : 顶上 gravity 剪切底边, 底上 gravity 切割上面, 不同时剪切两边。
clip_horizontal	额外选项, 可以设置使得子对象的左边和/或右边剪切至容器的边界, 这个切割基于水平 gravity : 左边 gravity 剪切右边, 右边 gravity 切割左边, 不同时剪切两边。

- android:tileMode (平铺模式)

关键字。定义平铺模式, 当允许平铺模式时, 重复位图, 忽略**gravity**设置。取值必须是如下常量值中的一个:

取值	描述
desable	不给位图添加标题, 默认值。
clamp	复制边缘的颜色, 如果着色超出原来的界限。
repeat	垂直和水平重复着色的图片。
mirror	垂直和水平重复着色的图片, 交替镜像图片使得临近的一直衔接。

例如:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/icon"
    android:tileMode="repeat" />
```

另参见：

- BitmapDrawable
- Creating alias resources

Nine-Patch (可拉伸图片)

一个可拉伸图片是一个PNG图片，当View中的图片内容超过了其正常的图片边界时，可以在其中定义供Android缩放的拉伸区域。此类图片常被用于至少有一个维度设置为“wrap_content”的View的背景，View需要扩展自己来适应其内容，而Nine-Patch也会缩放以匹配View的大小。Nine-Patch的一个使用示例就是Android标准Button控件的背景，必须伸缩以适应Button的文字或图片。同正常的位图一样，你可以直接引用一个Nine-Patch文件也可以通过XML定义资源引用。对于关于如何创建一个可伸缩的地区Nine-Patch文件的一个完整讨论，请参见2D图形文档。

Nine-Patch File

文件位置：

res/drawable/filename.9.png

使用文件名来标示资源。

编译过的资源数据类型：

资源指针指向一个NinePatchDrawable文件。

资源引用：

Java: R.drawable.filename

XML: @[package:]drawable/filename

举例：有一个图片保存在res/drawable/myninepatch.9.png，此布局XML应用到一个View：

```
<Button  
    android:layout_height="wrap_content"
```

```
    android:layout_width="wrap_content"
    android:background="@drawable/myninepatch" />
```

另附参考：

- 2D Graphics
- NinePatchDrawable

XML Nine-Patch

一个**XML Nine-Patch** 是一个定义在**XML**中，指向一个**Nine-Patch**文件的资源，此**XML**能够设置图片的抖动。

文件位置：

res/drawable/filename.xml
使用文件名来标示资源。

编译过的资源数据类型：

资源指针指向一个**NinePatchDrawable**文件。

资源引用：

Java: R.drawable.filename
XML: @[package:]drawable/filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<nine-patch
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:dither=[ "true" | "false" ] />
```

元素：

<nine-patch>

定义**Nine-Patch**和其特性

属性:

`xmlns:android`

字符型，必要的。定义XML的命名空间，必须是“<http://schemas.android.com/apk/res/android>”。

`android:src`

可绘制资源，必要的，指向一个Nine-Patch文件。

`android:dither`

布尔型，如果位图与屏幕的像素配置不同时，是否允许抖动。（例如：一个位图的像素设置是 ARGB 8888，但屏幕的设置是RGB 565）。

举例：

```
<?xml version="1.0" encoding="utf-8"?>
<nine-patch
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/myninepatch"
    android:dither="false" />
```

Layer list

一个图层是一个管理一系列其他绘图的绘制对象。在列表中的每个可绘制物有序地绘制在列表内，最后一个绘图在最上面。

每个绘图作为一个`<item>`元素单独地呈现在`<layer-list>`元素内。

文件位置：

`res/drawable/filename.xml`

使用文件名来标示资源。

编译过的资源数据类型：

资源指针指向一个LayerDrawable文件。

资源引用:

Java: R.drawable.filename

XML: @[package:]drawable/filename

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@[+][package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</layer-list>
```

元素:

<layer-list>

必要的，这个必须是一个根元素，包含了一个或多个<item>元素。

属性:

xmlns:android

字符型，必要的，定义XML的命名空间，必须是["http://schemas.android.com/apk/res/android".](http://schemas.android.com/apk/res/android)

<item>

定义一个绘图放在图层里，位置由其属性确定。必须是<selector>的子元素。可接受<bitmap>最为子元素。

属性:

android:drawable

可绘制资源，必须的，引用一个可绘制资源。

android:id

资源ID，此绘图资源唯一的资源ID。为新项创建一个新的资源ID：“@+id/name”。“+”表示新增一个ID。可以使用ID号通过View.findViewById()或者Activity.findViewById()来检索和修改此绘图资源。

android:top

整型，顶上像素偏移。

android:right

整型，右方像素偏移。

android:bottom

整型，底部像素偏移。

android:left

整型，左方像素偏移。

默认情况下所有绘制项目缩放，以适应包含视图的大小。因此，放置图像在图层列表的不同位置，可能会增加View的大小和一些图像需要扩展到合适的大小。为了避免列表中的扩展项，在<item>元素内使用<bitmap>元素，指定可绘制性和定义gravity为不会扩展的属性，诸如“居中”。例如，下面的<item>定义了一个项，扩展到适应其容器View的大小：

```
<item android:drawable="@drawable/image" />
```

为避免扩展，如下的例子使用一个<bitmap>元素，gravity属性为居中：

```
<item>
  <bitmap android:src="@drawable/image"
          android:gravity="center" />
```

</item>

例如：

XML文件保存于res/drawable/layers.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<layer-list
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <bitmap android:src="@drawable/android_red"
            android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
        <bitmap android:src="@drawable/android_green"
            android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
        <bitmap android:src="@drawable/android_blue"
            android:gravity="center" />
    </item>
</layer-list>

```

注意这个例子使用**gravity**属性居中的内嵌**<bitmap>**元素，来定义每个列表项的绘图资源。由于，图片偏移量引起的尺寸调整，这确保了没有图片需要缩放来适应容器的大小。把这个XML布局应用到一个View:

```

<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/layers" />

```

结果是偏移量不断增加的一系列图片的堆叠:



另参考图层Drawable

State List

状态列表Drawable是一个定义在XML中的可绘制对象，使用几个不同的图像，根据对象的状态来呈现同一个图形。例如，一个Button控件，可以处于几种状态中的一种（按下，聚焦，或都不是）使用状态列表可绘制，可以

为每个状态提供不同的背景图像。

可以在XML文件中描述列表，每个图形由一个`<item>`元素内部有一个单一`<selector>`元素呈现。每个`<item>`拥有多个属性来描述应用到可绘制对象的图形的状态。

在每个状态改变期间，状态列表是横跨从顶到底，使用第一个项匹配当前状态，这种选择不是基于最佳匹配，而仅仅是第一个项符合状态的最小标准。

文件位置：

`res/drawable/filename.xml`
以文件名作为标示资源的ID。

编译后资源数据类型：

资源指针指向一个`StateListDrawable`。

资源引用：

Java: `R.drawable.filename`
XML: `@[package:]drawable/filename`

语法：

```

<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android"
    android:constantSize=[ "true" | "false" ]
    android:dither=[ "true" | "false" ]
    android:variablePadding=[ "true" | "false" ] >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:state_pressed=[ "true" | "false" ]
        android:state_focused=[ "true" | "false" ]
        android:state_hovered=[ "true" | "false" ]
        android:state_selected=[ "true" | "false" ]
        android:state_checkable=[ "true" | "false" ]
        android:state_checked=[ "true" | "false" ]
        android:state_enabled=[ "true" | "false" ]
        android:state_activated=[ "true" | "false" ]
        android:state_window_focused=[ "true" | "false" ] />
</selector>

```

元素：

<selector>

必要的，这个必须是根元素，包含了一个或多个<item>元素。

属性：

xmlns:android

字符串型，必要的，定义XML的命名空间，必须等于"<http://schemas.android.com/apk/res/android>"。

android:constantSize

布尔型。当状态变化时，绘制对象的内部大小保持不变（所有状态的最大值）为"true"；如果绘制对象的大小根据当前状态而改变为"false"，默认设置为"false"。

android:dither

布尔型，"true"，当位图的像素配置与屏幕的不一致时，允许位图抖动（例如，一个ARGB 8888 的位图和一个RGB 565 的屏幕）；"false"，不允许抖动，默认设置为"false"。

android:variablePadding

布尔型，为"true"时，根据所选择的当前状态，可绘制对象的填充需要改变；为"false"时，填充保持不变（所有状态的填充最大值）。使用此特性要求当状态改变时处理好布局，它会常常不支持。默认设置为"false"。

<item>

定义一个可应用在一定状态的绘制对象，属性由其描述所定，必须是<selector>的子元素。

属性：

android:drawable

可绘制资源，必要的，引用一个可绘制资源。

android:state_pressed

布尔型，为"true"时，当对象被按下（诸如触摸/单机一个按钮）使用此选项；为"false"时，默认设置应当使用此选项，处于没有按下状态。

android:state_focused

布尔型，为"true"时，当对象拥有输入焦点时应使用此选项（诸如当用户选择一个文本输入）；为"false"时，默认设置使用此选项，处于没有焦点状态。

android:state_hovered

布尔型，为"true"时，当对象处于游标徘徊在附近时，使用此选项；"false"时，默认使用此选项，处于没有游标徘徊状态。通常，这种绘制对象与聚焦状态的绘制对象相同。

API 14 中有介绍。

android:state_selected

布尔型，为"true"时，当对象是当前用户通过方向控制键导航选择选项（诸如通过一个十字键列表导航），使用此选项；为"false"时，当对象没有被选中时，使用此选项。

当聚焦不够用时使用此选择状态（诸如当列表视图拥有聚焦而使用十字键选择一个选项时）。

android:state_checkable

布尔型，为"true"时，对象是可核对的；为"false"时，对象不可核对的。（只当对象可以在可核对与不可核对控件间变换时使用）

android:state_enabled

布尔型，为 "true" 时，对象许可（能够接受到触摸或单击事件）；为 "false" 时，对象不许可。

android:state_activated

布尔型，为 "true" 时，由于持久的选择使对象处于激活状态，（诸如在一种持久的导航视图中高亮标示列表选项）；为 "false" 时，对象没有处于激活状态。

详细介绍在 API11

android:state_window_focused

布尔型，为 "true" 时，焦点在应用的窗口（此应用处于前台），此应用的窗口没有获得焦点（例如当有通知栏拉下或对话出现）。

注意：记住 Adroid 在状态列表中使用第一个选项以匹配对象的当前状态。所以，如果列表中第一个选项不包含任一个的状态属性，此时它就会每次都使用第一个选项，这也是为什么要把默认属性放在最后面的原因。（如下面的示例所示）

例子： XML 文件保存于 res/drawable/button.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!--
pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!--
focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!--
hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default
-->
</selector>
```

把此布局应用到一个按钮的可绘制状态列中：

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/button" />
```

另参见：状态列Drawable

Level List

一个可绘制对象管理许多可相互替换的可绘制对象，每个都标示了最大数值。使用setLevel()设置可绘制资源的级别，在级别列表中装载可绘制资源，此列表有个android:maxLevel 值，此值大于或等于方法传递给setLevel()方法的值。

文件位置：

res/drawable/filename.xml
文件名作为资源ID。

编译后的资源数据类型：

指向一个LevelListDrawable的资源。

资源引用：

Java: R.drawable.filename
XML: @[package:]drawable/filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<level-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/drawable_resource"
        android:maxLevel="integer"
        android:minLevel="integer" />
</level-list>
```

元素:

<level-list>

此必须是根元素，包含一个或多个<item>元素。

xmlns:android

字符型，必要的，定义XML的命名空间，必须是"<http://schemas.android.com/apk/res/android>".

<item>

在某级别上定义一个可绘制的对象。

属性:

android:drawable

可绘制资源，必要的，引用一个可绘制资源来插入。

android:maxLevel

整型，允许的级别最大值

android:minLevel

整型，允许的级别最小值

举例:

```
<?xml version="1.0" encoding="utf-8"?>
<level-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/status_off"
        android:maxLevel="0" />
    <item
        android:drawable="@drawable/status_on"
        android:maxLevel="1" />
</level-list>
```

一旦把此应用到一个视图上，此等级值可以通过setLevel() 或 setImageLevel()修改 另参见： LevelListDrawable

Transition Drawable

一个变换Drawable是一个可绘制对象，在两个可绘制资源间能够同时淡入淡出。每个可绘制对象由在单独的<transition>元素中的<item>元素表示。至多支持两个选项。前向过渡使用startTransition()，后向使用reverseTransition()。

文件位置：

res/drawable/filename.xml
文件名作为资源ID。

编译后的资源数据类型：

指向一个变换Drawable的资源。

资源引用：

Java: R.drawable.filename
XML: @[package:]drawable/filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<transition
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@[+][package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</transition>
```

元素：

<transition>

必要的，这个必须是根元素，包含一个或多个<item>元素。

属性:

`xmlns:android`

字符型，必要的，定义XML的命名空间，必须是"<http://schemas.android.com/apk/res/android>"。

`<item>`

定义一个可绘制对象作为此可绘制过度，必须是`<transition>`元素的子元素。接受`<bitmap>`子元素。

属性:

`android:drawable`

可绘制资源，必要的，引用一个可绘制资源。

`android:id`

必要的ID，此可绘制对象的唯一资源ID。若为此项创建一个新的资源ID，使用 "`@+id/name`"。加号表示应当创建一个新的ID。可以使用此符号通过`View.findViewById()`或者`Activity.findViewById()`来显示和修改此绘制对象。

`android:top`

整型，像素的上方偏移量。

`android:right`

整型，像素的右边偏移量。

`android:bottom`

整型，像素的下方偏移量。

`android:left`

整型，像素的左边偏移量。

举例：

XML文件保存于res/drawable/transition.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<transition
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

把此XML布局应用到试图：

```
<ImageButton
    android:id="@+id/button"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/transition" />
```

下面的代码实现了从第一项到第二项变换需要500ms。

```
ImageButton button = (ImageButton) findViewById(R.id.button);
TransitionDrawable drawable = (TransitionDrawable)
button.getDrawable();
drawable.startTransition(500);
```

另参见: 变换Drawable

Inset Drawable

一个定义在XML中的可绘制对象，在指定的距离内插入另外一个可绘制对象。当一个视图的背景小于视图的实际大小时，会用到此。

文件位置：

res/drawable/filename.xml
文件名作为资源ID。

编译后资源数据类型：

资源InsetDrawable指针

资源引用：

Java: R.drawable.filename

XML: @[package:]drawable/filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<inset
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:insetTop="dimension"
    android:insetRight="dimension"
    android:insetBottom="dimension"
    android:insetLeft="dimension" />
```

元素：

<inset>

定义一个插入drawable。必须是根元素。

属性：

xmlns:android

字符型，必要的，定义XML的命名空间，必须是["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android).

android:drawable

可绘制资源，必要的，引用一个可绘制资源以插入。

android:insetTop

尺寸，上端插入，作为一个尺寸值或尺寸资源。

android:insetRight

尺寸，右端插入，作为一个尺寸值或尺寸资源。

android:insetBottom

尺寸，底端插入，作为一个尺寸度值或尺寸资源。

android:insetLeft

尺寸，左边插入，作为一个尺寸值或尺寸资源。

举例：

```
<?xml version="1.0" encoding="utf-8"?>
<inset xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/background"
    android:insetTop="10dp"
    android:insetLeft="10dp" />
```

另参见： 插入Drawable

Clip Drawable

根据可绘制对象的当前级别，定义在XML中可绘制对象，可以剪贴另一个可绘制对象。

可以根据当前级别决定以什么样的宽高比裁剪子对象，还有gravity来控制在容器内部的摆放位置。

常用来实现例如进度条的东西。

文件位置：

res/drawable/filename.xml

文件名作为资源ID。

编译后资源数据类型。

资源引用：

Java: R.drawable.filename

XML: @[package:]drawable/filename

语法：

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

```

xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:clipOrientation=[ "horizontal" | "vertical" ]
    android:gravity=[ "top" | "bottom" | "left" | "right" |
"center_vertical" | "fill_vertical" | "center_horizontal" |
"fill_horizontal" | "center" | "fill" | "clip_vertical" |
"clip_horizontal" ] />

```

元素:

<clip> 定义剪贴绘制对象，必须是根元素。

属性:

xmlns:android

字符型，必要的，定义XML的命名空间，必须是["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android)。

android:drawable

可绘制资源，必要的，引用一个要剪贴的可绘制资源。

android:clipOrientation

关键字，对剪贴进行定位。

取值必须是如下常量值之一：

取值	描述
horizontal	水平剪切绘制对象。
vertical	垂直剪切绘制对象。

android:gravity

关键字，指定可绘制对象内的剪贴位置。

取值必须是如下常量之一或多个（以“|”分隔）：

取值	描述
top	把对象放在容器的上方，不改变其大小，当clipOrientation是“vertical”，从对象的底部开始分割。
clip_vertical	额外选项，可以设置使得子对象的顶和/或底上剪切至容器的边界，这个切割基于垂直gravity：顶上gravity剪切底边，底上gravity切割上面，不同时剪切两边。
clip_horizontal	额外选项，可以设置使得子对象的左边和/或右边剪切至容器的边界，这个切割基于水平gravity：左边gravity剪切右边，右边gravity切割左边，不同时剪切两边。
fill	同时水平和垂直拉伸对象以填充容器，此为默认项。
fill_vertical	对象垂直填充容器。
center	把对象放在容器的垂直与水平的居中位置，不改变其大小。
center_vertical	把对象放在容器的垂直居中位置，不改变其大小。

right	把对象放在容器的右边，不改变其大小。
left	把对象放在容器的左边，不改变其大小。
bottom	把对象放在容器的底部，不改变其大小。

举例: XML文件保存于 res/drawable/clip.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/android"
    android:clipOrientation="horizontal"
    android:gravity="left" />
```

下面的XML布局应用到一个试图的剪贴可绘制资源:

```
<ImageView
    android:id="@+id/image"
    android:background="@drawable/clip"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

如下的代码取得可绘制对象，为了渐进地显示图像增加了剪贴数:

```
ImageView imageview = ( ImageView ) findViewById( R.id.image );
ClipDrawable drawable = ( ClipDrawable ) imageview.getDrawable();
drawable.setLevel( drawable.getLevel() + 1000 );
```

增加级别减少了剪贴数，减慢了显示图像的速度，如下是级别数为7000:



注意: 默认级别为0，全部剪贴，图像不可见，当级别是10000，图片没有剪贴，全部可见。

另参见： 剪贴Drawable

Scale Drawable

一个定义在XML中的可绘制对象，能够依据当前级别改变另一个可绘制对象的大小。

文件位置：

res/drawable/filename.xml
文件名作为资源ID。

编译后数据类型：

指向尺寸Drawable的资源指针。

资源引用：

Java: R.drawable.filename
XML: @[package:]drawable/filename

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<scale
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:scaleGravity=[ "top" | "bottom" | "left" | "right" |
"center_vertical" | "fill_vertical" | "center_horizontal" |
"fill_horizontal" | "center" | "fill" | "clip_vertical" |
"clip_horizontal" ]
    android:scaleHeight="percentage"
    android:scaleWidth="percentage" />
```

元素：

<scale>

定义尺寸drawable.，必须是根元素。

属性:

`xmlns:android`

字符型，必要的，定义XML的命名空间，必须是"<http://schemas.android.com/apk/res/android>".

`android:drawable`

可绘制资源，必要的，引用一个可绘制资源。

`android:scaleGravity`

关键字，指定缩放后的gravity位置

取值必须是如下常量中一个或多个（以“|”分隔）：

取值	描述
<code>top</code>	把对象放在容器的上方，不改变其大小，当 <code>clipOrientation</code> 是"vertical"，从对象的底部开始分割。
<code>bottom</code>	把对象放在容器的底部，不改变其大小。
<code>left</code>	把对象放在容器的左边，不改变其大小。
<code>right</code>	把对象放在容器的右边，不改变其大小。
<code>center_vertical</code>	把对象放在容器的垂直居中位置，不改变其大小。

fill_vertical	对象垂直填充容器。
center	把对象放在容器的垂直与水平的居中位置，不改变其大小。
fill	同时水平和垂直拉伸对象以填充容器，此为默认项。
clip_vertical	额外选项，可以设置使得子对象的顶和/或底上剪切至容器的边界，这个切割基于垂直 gravity : 顶上 gravity 剪切底边，底上 gravity 切割上面，不同时剪切两边。
clip_horizontal	额外选项，可以设置使得子对象的左边和/或右边剪切至容器的边界，这个切割基于水平 gravity : 左边 gravity 剪切右边，右边 gravity 切割左边，不同时剪切两边。

android:scaleHeight

百分比，高度尺寸，表示可绘制对象的边界百分比，取值以百分比形式，例如：100%,12.5%，等。

android:scaleWidth

百分比，高度尺寸，表示可绘制对象的边界百分比，取值以百分比形式，例如：100%,12.5%，等。

举例：

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/logo"
    android:scaleGravity="center_vertical|center_horizontal"
    android:scaleHeight="80%"
    android:scaleWidth="80%" />
```

另参见: 尺寸Drawable

Shape Drawable

这是一个定义在XML中的通用形状。

文件位置:

res/drawable/filename.xml

文件名作为资源ID。

编译后的资源数据类型:

指向一个渐变Drawable的资源指针。

资源引用:

Java: R.drawable.filename

XML: @[package:]drawable/filename

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=[ "rectangle" | "oval" | "line" | "ring" ] >
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <gradient
        android:angle="integer"
        android:centerX="integer"
        android:centerY="integer"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
```

```

        android:startColor="color"
        android:type=[ "linear" | "radial" | "sweep" ]
        android:useLevel=[ "true" | "false" ] />
<padding>
    android:left="integer"
    android:top="integer"
    android:right="integer"
    android:bottom="integer" />
<size>
    android:width="integer"
    android:height="integer" />
<solid>
    android:color="color" />
<stroke>
    android:width="integer"
    android:color="color"
    android:dashWidth="integer"
    android:dashGap="integer" />
</shape>
```

元素:

<shape>

形状drawable，必须是根元素。

属性:

xmlns:android

字符型，必要的，定义XML的命名空间，必须
是["http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android)。

android:shape

关键字，定义形状的类型，取值是：

取值	描述
"rectangle"	填充包含的视图的矩形，是默认形状。
"oval"	椭圆形，适合包含的视图的尺寸。

"line"	划分包含的视图的水平线，这个形状要求<stroke>元素来定义线的宽度。
"ring"	环形。

如下属性只有当`android:shape="ring"`时才使用。

`android:innerRadius`

尺寸，内环半径（中间的孔），作为一个尺寸值或尺寸资源。

`android:thickness`

尺寸，环的厚度，作为一个尺寸值或尺寸资源。

`android:thicknessRatio`

浮点型，环的厚度比上环的宽度，例如，如果`android:thicknessRatio="2"`，厚度等于环的宽度的1/2，此值被`android:innerRadius`重写，默认为3.

`android:useLevel`

布尔型，为"true"时，用于`LevelListDrawable`，正常情况设为"false"，或者形状不出现。

<corners>

为形状创建圆角，只有当形状为矩形时才应用。

属性：

`android:radius`

尺寸数，所有的角的半径，作为一个尺寸值或尺寸资源，对于每个角会重写如下的属性：

android:topLeftRadius

尺寸数，左上角的半径，作为一个尺寸度值或尺寸资源。

android:topRightRadius

尺寸数，右上角的半径，作为一个尺寸度值或尺寸资源。

android:bottomLeftRadius

尺寸数，左下角的半径，作为一个尺寸值或尺寸度资源。

android:bottomRightRadius

尺寸数，右下角的半径，作为一个尺寸值或尺寸度资源。

注意：每个角的角半径必须大于1，不然没有圆角。如果想指定所有的角都不是圆角，使用`android: radius` 来设定默认的角半径大于1,然后重写每个角，并指定每个角的半径值为所需要的值，如果不需要圆角，值为0 (0dp) 。

<gradient>

为形状指定渐变颜色。

属性：

android:angle

整形，渐变的角度，度数，0度为从左到右，90度是从底到上，必须是45度的倍数，默认为0.

android:centerX

浮点型，距离渐变中心的X坐标的相对位置(0 - 1.0)。

android:centerY

浮点型，距离渐变中心的Y坐标的相对位置(0 - 1.0)。

android:centerColor

颜色，可选择开始到结束之间的颜色，作为一个十六进制值或颜色资源。

android:endColor

颜色，结束颜色，作为一个十六进制值或颜色资源。

android:gradientRadius

浮点型，渐变的半径，只有当android:type="radial"才使用

android:startColor

颜色，开始颜色，作为一个十六进制值或者颜色资源。

android:type

关键字，使用的渐变模式，有效值如下：

取值	描述
linear	线性渐变，默认选择
radial	辐射渐变，开始颜色也是结束颜色
sweep	卷曲线渐变

android:useLevel

布尔型，为"true"时，作为一个 LevelListDrawable。

<padding>

填充以适用于视图元素（填充视图内容的位置而不是形状）。

属性：

android:left

尺寸，左边填充，作为一个尺寸值或者尺寸资源。

android:top

尺寸，顶上填充，作为一个尺寸值或者尺寸资源。

android:right

尺寸，右边填充，作为一个尺寸值或者尺寸资源。

android:bottom

尺寸，底边填充，作为一个尺寸值或者尺寸资源。

<size>

形状的大小。

属性：

android:height

尺寸，形状的高，作为一个尺寸值或者尺寸资源。

android:width

尺寸，形状的宽，作为一个尺寸值或者尺寸资源。

注意：形状缩放大小以适应视图，与定义的尺寸相称，默认，当在一个图像视图使用形状时，可以限制缩放，通过设置 **android:scaleType to "center"**。

<solid>

固定颜色填充形状。

属性：

android:color

颜色，用到形状上的颜色，作为一个十六进制值或颜色资源。

<stroke>

形状的笔触。

属性：

android:width

尺寸，线的宽度，作为一个尺寸值或尺寸资源。

android:color

颜色，线的颜色，作为一个十六进制值或者颜色资源。

android:dashGap

尺寸，虚线间隔，作为一个尺寸值或尺寸资源，只有当设置`android:dashWidth`才有效。

android:dashWidth

尺寸，每个虚线的大小，作为一个尺寸值或尺寸资源，只有当设置`android:dashGap`才有效。

例子：

XML文件存于res/drawable/gradient_box.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
```

```
    android:endColor="#80FF00FF"
    android:angle="45" />
<padding android:left="7dp"
    android:top="7dp"
    android:right="7dp"
    android:bottom="7dp" />
<corners android:radius="8dp" />
</shape>
```

把此XML布局应用到一个视图的形状drawable：

```
<TextView
    android:background="@drawable/gradient_box"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

应用程序代码获得形状drawable，应用到视图：

```
Resources res = getResources();
Drawable shape = res.getDrawable(R.drawable.gradient_box);
TextView tv = (TextView) findViewById(R.id.textview);
tv.setBackground(shape);
```

另参见：

形状Drawable

来自“[index.php?title=Drawable&oldid=8950](#)”

Layout

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

布局资源定义了UI在活动或者UI组件中的体系结构。

文件位置：

`res/layout/filename.xml`

文件名用作资源ID。

编译的资源数据类型：

资源指向视图（或子类）资源。

资源引用：

Java: `R.layout.filename`

XML: `@[package:]layout/filename`

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
    android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
<View
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
    android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
    [View-specific attributes] >
    <requestFocus/>
</View>
<ViewGroup>
    <View />
</ViewGroup>
<include layout="@layout/layout_resource" />
</ViewGroup>
```

元素：

`<ViewGroup>`

其他视图元素的container。有许多不同类型的视图组对象，而每一个会用不同方式指定子元素的布局。不同类别的视图组对象包括 `LinearLayout`, `RelativeLayout`, 和`FrameLayout`。

You should not assume that any derivation of `ViewGroup` will accept nested Views. 某些`ViewGroup`实现了`AdapterView`类，这就决定仅`Adapter`才能实现子类。

属性：

`android:id`

资源ID。元素的独特资源名称，可以用来引用`ViewGroup`。

`android:layout_height`

尺寸或关键字。必备。把该组高度作为尺寸值（或者尺寸资源）或关键字("fill_parent"或"wrap_content")。

`android:layout_width`

尺寸或关键字。必备。把该组宽度作为尺寸值（或者尺寸资源）或关键字 ("fill_parent"或"wrap_content")。

ViewGroup基类支持了更多属性。如果想查看所有可用属性，请查阅视图组类对应的参考文档。

<View>

独立UI组件通常称为“窗口小部件”。不同种类的视图对象包括视图组，按钮和复选框。

属性：

android:id

资源ID。

资源ID。元素的独特资源名称，可以用来引用应用程序的View。

android:layout_height

尺寸或关键字。必备。把该组高度作为尺寸值（或者尺寸资源）或关键字("fill_parent"或"wrap_content")。

android:layout_width

尺寸或关键字。必备。把该组宽度作为尺寸值（或尺寸资源）或关键字 ("fill_parent"或"wrap_content")。

View基类支持了更多属性。查阅布局可以获得更多信息。如果想查看所有可用属性，请查阅对应的参考文档。

<requestFocus>

任何代表视图对象的元素都可以包含这一空元素，空元素会把父类初始重点显示在屏幕上。每个文件只能包含这些元素之一。

<include>

向该布局添加一个布局文件。

属性：

layout

布局资源。必备。引用到布局资源。

android:id

资源ID。覆盖根视图的ID。

android:layout_height

尺寸或关键字。在布局中覆盖根视图的给定高度。只有同时声明**android:layout_width**才有效果。

android:layout_width

尺寸或关键字。在布局中覆盖根视图的给定宽度。只有同时声明**android:layout_height**才有效果。

可以包含<include>中根元素支持的其他布局属性，它会覆盖根元素定义的属性。

另一种包含布局的方式就是使用**ViewStub**。**ViewStub**是在明确扩展前不占用布局空间的轻量级视图，它包括了**android:layout**定义的布局文件。

<merge>

在布局层次中不会绘制替代根元素。把该布局放入包含适当父视图的布局中以包含<merge>元素子类，此时替代根元素作为根元素非常有用。想要使用<include>把此布局包含在另一布局文件中，而此布局不需要不同视图组container时这一方法特别有用。

android:id值

对于ID值，通常使用“@+id/name”的语法形式。加号+表示这是一个新资源ID，并且如果新资源整型不存在，aapt工具会在R.java类中进行创建。例如：

```
<TextView android:id="@+id/nameTextbox" />
```

nameTextbox 名称是附加到此元素的资源ID。可以参考TextView 获取Java中相关ID：

```
findViewById(R.id.nameTextbox);
```

此代码返回到TextView对象。

然而如果定义了ID资源（尚未使用），就可以通过排除android:id值的加号把ID应用于视图元素。

android:layout_height和android:layout_width的值：

高度和宽度值可以用安卓支持的任意标注单位（px, dp, sp, pt, in, mm）或下列关键词来表示：

自定义视图元素

可以创建自己的自定义视图和视图组元素，并应用到与标准布局元素相同的布局中。同时可以指定XML元素支持的属性。

元素：

XML文件保存在 res/layout/main_activity.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" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

此应用程序代码会使用onCreate()方法来加载活动布局：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
}
```

来自“[index.php?title=Layout&oldid=13872](#)”



Menu Resource

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/resources/menu-resource.html>

翻译：croftwql

更新：2012.08.09

参见

[菜单](#)

菜单资源

一个菜单资源定义了一个应用程序菜单（选项菜单，上下文菜单或子菜单），可以与[MenuItemInflator](#)配合使用。

关于使用菜单指导，可以看[Menus](#)开发指南。

文件的位置：

res/menu/filename.xml
文件名将被用作资源ID。

编译的资源数据类型：

资源指针到菜单（或子类）的资源。

资源参考：

In Java: R.menu.filename In XML:@[package:]menu.filename

语法：

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@[+][package:]id/resource_name"
          android:title="string"
          android:titleCondensed="string"
          android:icon="@[package:]drawable/drawable_resource_name"
          android:onClick="method name"
          android:showAsAction=[ "ifRoom" | "never" | "withText" |
"always" | "collapseActionView" ]

        android:actionLayout="@[package:]layout/layout_resource_name"
        android:actionViewClass="class name"
        android:actionProviderClass="class name"
        android:alphabeticShortcut="string"
        android:numericShortcut="string"
        android:checkable=[ "true" | "false" ]
        android:visible=[ "true" | "false" ]
        android:enabled=[ "true" | "false" ]
        android:menuCategory=[ "container" | "system" | "secondary" |
"alternative" ]
        android:orderInCategory="integer" />
    <group android:id="@[+][package:]id/resource_name"
           android:checkableBehavior=[ "none" | "all" | "single" ]
           android:visible=[ "true" | "false" ]
           android:enabled=[ "true" | "false" ]
           android:menuCategory=[ "container" | "system" | "secondary" |
"alternative" ]
           android:orderInCategory="integer" >
        <item />
    </group>
    <item >
        <menu>
            <item />
        </menu>
    </item>
</menu>

```

内容：

<menu>

需要。这必须是根节点。包含的<item> /或 <group>的元素。
属性：

xmlns:android

XML命名空间。必需的。定义XML命名空间，它必须是"<http://schemas.android.com/apk/res/android>"。

<item>

一个菜单项。可能包含<menu>的元素（子菜单）。必须有一个<menu>或<group>元素。

属性：

android:id

资源ID。独特的资源ID。此项目创建一个新的资源ID，使用的形式：“@+id/ name ”。

加符号表示，这应该被视为一个新的ID创建的。

android:title

字符串资源。作为一个字符串资源或原始字符串的菜单标题。

android:titleCondensed

字符串资源。一个简明标题作为一个字符串资源或原始字符串。是用于正常的标题过长的情况下，这个称号。

android:icon

绘制的资源。菜单项图标的图像。

android:onClick

方法的名称。这个菜单项被点击时要调用的方法。该方法必须声明为公共活动和接受的[MenuItem](#)作为其唯一的参数，这表明该项目点击。这种方法比标准的回调到，优先[onOptionsItemSelected\(\)](#)。看到在底部的例子。

警告：如果您混淆您的代码使用ProGuard（或类似的工具），可以肯定排除你在这从命名属性指定的方法，因为它可以打破的功能。

介绍API等级11

android:showAsAction

关键字。何时以及如何一个动作项显示在动作栏。当一个活动界面包含[ActionBar](#)(API等级11介绍)，一个菜单项能作为动作项出现。有效值：

值	描述
ifRoom	如果有足够的空间，这个值会使菜单项显示在Action Bar上。
withText	分开
never	这个值使菜单项永远都不出现在Action Bar上
always	在操作栏中，始终显示这个项目。避免使用，除非该项目总是出现在操作栏中，它的关键。设置多个项目总是出现行动项目可能会导致他们与其他UI在操作栏重叠。
collapseActionBar	认为这个行动项目（如宣布相关的 <code>android:actionLayout</code> 或 <code>android:actionViewClass</code> ）的行动是可折叠的。 API等级14介绍 。

[ActionBar](#)更多信息，详见指南。介绍了API等级11。

android:actionLayout

布局资源。使用动作视图布局。

[ActionBar](#)更多信息，详见指南。介绍了API等级11。

android:actionViewClass

类的名称。为完全合格的类名作为行动视图查看使用。例如，“`android.widget.SearchView`” 使用搜索查看作为一个动作视图。

[ActionBar](#)更多信息，详见指南。

警告：如果您混淆您的代码使用ProGuard（或类似的工具），一定要排除你在这从命名属性指定的类，因为它可以打破功能。介绍了API等级11。

android:actionProviderClass

类的名称。为完全合格的类名[ActionProvider](#)行动项目的地方使用。例如，“`android.widget.ShareActionProvider`”使用`ShareActionProvider`

[ActionBar](#)更多信息，详见指南。

警告：如果您混淆您的代码使用ProGuard（或类似的工具），一定要排除你在这从命名属性指定的类，因为它可以打破功能。介绍了API等级14。

android:alphabeticShortcut

字符。一个字符为字母快捷键。

android:numericShortcut

整数。一些数字快捷键。

android:checkable

布尔。“真”，如果该项目是检验的。

android:checked

布尔。“真”，如果该项目被默认选中。

android:visible

布尔。“真”，如果该项目是默认可见。

android:enabled

布尔。“真”，如果该项目是默认启用的。

android:menuCategory

关键字。相应的菜单 CATEGORY_* 常量，它定义了该项目的优先级值。有效值：

值	描述
container	项目是容器的一部分
system	对于系统提供的项目
secondary	对于项目的用户提供二次选项（很少使用）
alternative	项目，是对当前显示的数据替代行动

android:orderInCategory

整数。该项目的“重要性”的顺序，一个组内。

<group>

菜单组（创建一个共享的特征，如他们是否是可见的，启用，或托运的物品的集合）。包含一个或多个的<item>元素。必须一个子<menu>元素。

属性：

android:id

资源ID。独特的资源ID。此项目创建一个新的资源ID，使用的形式：“@+id/name”。加符号表示，这应该被视为一个新的ID创建的。

android:checkableBehavior

关键字。组托运的行为类型。有效值：

值	描述
---	----

none	不可检查
all	可以检查所有项目 (使用复选框)
single	只有一个项目可以检查 (使用单选按钮)

android:visible

布尔。如果组是可见的“真实”。

android:enabled

布尔。“真”，如果该组已启用。

android:menuCategory

关键字。相应的菜单 CATEGORY_ * 常量，它定义组的优先级值。有效值：

值	描述
container	团体是一个容器的一部分
system	对于系统提供的组
secondary	对于群体的用户提供二次选项 (很少使用)
alternative	团体，是对当前显示的数据替代行动

android:orderInCategory

整数。默认类别内的项目顺序。

例如： XML文件保存在 res/menu/example_menu.xml:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:title="@string/item1"
          android:icon="@drawable/group_item1_icon"
          android:showAsAction="ifRoom|withText" />
    <group android:id="@+id/group">
        <item android:id="@+id/group_item1"
              android:onClick="onGroupItemClick"
              android:title="@string/group_item1"
              android:icon="@drawable/group_item1_icon" />
        <item android:id="@+id/group_item2"
              android:onClick="onGroupItemClick"
              android:title="@string/group_item2"
              android:icon="@drawable/group_item2_icon" />
    </group>
    <item android:id="@+id/submenu"
          android:title="@string/submenu_title"
          android:showAsAction="ifRoom|withText" >
        <menu>
            <item android:id="@+id_submenu_item1"
                  android:title="@string/submenu_item1" />
        </menu>
    </item>
</menu>
```

下面的应用程序代码膨胀从菜单onCreateOptionsMenu (菜单) 回调，也定义了按钮回调两个项目：

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.example_menu, menu);
    return true;
}

public void onGroupItemClick(MenuItem item) {
    // One of the group items (using the onClick attribute) was
    // clicked
    // The item parameter passed here indicates which item it is
    // All other menu item clicks are handled by
    onOptionsItemSelected()
}
```

注： android:showAsAction仅适用于的Android 3.0 (API等级11) 和以上。

来自“[index.php?title=Menu_Resource&oldid=8685](#)”



String Resources

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

<http://developer.android.com/guide/topics/resources/string-resource.html>

作者：smilysas

更新时间：2012年8月2日

目录

[[隐藏](#)]

[1 String Resources](#)

- [1.1 String](#)
- [1.2 String Array](#)
- [1.3 Quantity Strings \(Plurals\)](#)
- [1.4 Formatting and Styling](#)
 - [1.4.1 撇号和引号的转义](#)
 - [1.4.2 字符串的格式化](#)
 - [1.4.3 用HTML标记来样式化](#)

String Resources

字符串资源为应用程序提供可选样式和格式的文本字符串。应用程序可以使用三种类型的字符串资源：

[String](#)

提供了一个字符串的XML资源。

String Array

提供了一个字符串数组的资源。

Quantity Strings (Plurals)

提供同一个单词或者词组不固定重复组成的字符串资源

所有的字符串都可以匹配某种标记样式和格式化参数。要获取更多关于字符串样式和格式化的信息，请参见[Formatting and Styling](#)。

String

单个字符串可以被应用程序或者其它资源文件（如XML布局文件）引用。

注意：单个字符串资源是只能通过提供的name属性（而不是XML文件名）进行引用。所以，在一个XML文件中，可以在

<resource>元素下组合引用字符串和其他简单资源。

文件定位：

res/values/filename.xml

filename文件名是可以自定义的。<string>元素的name属性值会被用作资源的ID号。

编译资源数据类型：

指向字符串的资源指针。

资源引用：

在JAVA文件中：R.string.string_name

在XML文件中：@string/string_name

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string
        name="string_name"
        >text_string</string>
</resources>
```

元素：

<resources>

必须定义的。必须是根节点。没有属性值。

<string>

一个字符串，可以包含样式标签。

需要注意的是，必须转义撇号和引号。关于如何正确设定字符串样式和格式化的信息，请参见[Formatting and Styling](#)。

属性：

name

字符串。标识字符串的名字。这个名字将被用作资源ID.

例子：

XML文件保存为res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

XML布局文件把字符串应用到一个视图中：

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

应用程序通过以下代码返回一个字符串：

```
String string = getString(R.string.hello);
```

可以通过[getString\(int\)](#)或者[getText\(int\)](#)取回一个字符串。[getText\(int\)](#)将保留所有应用到字符串的文本样式。

String Array

由可以被应用程序引用的字符串组成的数组

注意：字符串数组是只能通过提供的name属性（而不是XML文件名）进行引用的简单资源。所以，在一个XML文件中，可以在<resource>元素下组合引用字符串和其他简单资源。

文件定位：

res/values/filename.xml

filename文件名是可以自定义的。<string-array>元素的name属性值会被用作资源的ID号。

编译资源数据类型：

指向字符串数组的资源指针。

资源引用：

在JAVA文件中：R.string.string_array_name

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="string\_array\_name">
        <item>text_string</item>
    </string-array>
</resources>
```

元素：

<resources>

必须定义的。必须是根节点。没有属性值。

<string-array>

定义一个字符串数组，可以包含一个或多个<item>标签。

属性：

name

字符串。标识字符串数组的名字。这个名字将被用作该数组的资源ID.

<item>

一个字符串，可以包含样式标签。值可以是对其它字符串资源的引用，必须是<string-array>元素的孩子。需要注意的是，必须转义撇号和引号。关于如何正确设定字符串样式和格式化的信息，请参见[Formatting and Styling](#)。没有属性。

例子：

XML文件保存为res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

应用程序通过代码返回字符串数组：：

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Quantity Strings (Plurals)

不同的语言对数量进行描述的语法规则也不同。比如在英语里，数量1是个特殊情况，我们写成“1 book”，但其他任何数量都要写成“n books”。这种单复数之间的区别是很普遍的，不过其他语言会有更好的区分方式。Android支持的全集包括zero、one、two、few、many和other。决定选择和使用某种语言和复数的规则是非常复杂的，所以Android提供了诸如getQuantityString()的方法来选择合适的资源。

注意，要按照语法规则来建立可选项。在英语里，即使数量为0，字符串零(zero)也不需要建立。因为0在英语语法表达上和2没有区别，和其他除1以外的任何数字都没有差别（"zero books", "one book", "two books"，等等）。不要被“two听起来似乎只能用于数量2”之类的事误导。某语言可能需要2、12、102和1一样形式表示，而又与其他数的形式都不同。

如果和应用程序的风格一致，常可以用诸如“Books: 1”的模糊数量形式来避免使用数量字符串。

注意：复数字字符串是只能通过提供的name属性（而不是XML文件名）进行引用的简单类型资源。所以，在一个XML文件中，可以在<resource>元素下组合引用字符串和其他简单资源。

文件定位：

res/values/filename.xml

filename文件名是可以自定义的。<plurals>元素的name属性值会被用作资源的ID号。

资源引用：

在JAVA文件中：R.plurals.plural_name

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals
        name="plural_name">
        <item
            quantity=[ "zero" | "one" | "two" | "few" | "many" |
"other" ]>text_string</item>
    </plurals>
</resources>
```

元素：

<resources>

必须定义的。必须是根节点。没有属性值。

<plurals>

一个字符串集，每个数量表示提供一个字符串。可以包含一个或多个<item>标签。

属性：

name

字符串。标识字符串集的名字。这个名字将被用作该数组的资源ID.

<item>

一个单数或复数形式的字符串。可以是对其他字符串资源的引用。必须是<plurals>元素的孩子。需要注意的是，必须转义撇号和引号。关于如何正确设定字符串样式和格式化的信息，请参见[Formatting and Styling](#)。

属性：

quantity

关键字。 表示要使用此字符串的数量值。以下是合法的值（括号内列出部分语言要求）：

Class/Interface	Description
zero	语言需要对数字0进行特殊处理。（比如阿拉伯语）
one	语言需要对类似1的数字进行特殊处理。 (比如英语和其它大多数语言里的1；在俄语里，任何以1结尾但不以11结尾的数也属于此类型。)
two	语言需要对类似2的数字进行特殊处理。 (比如威尔士语)
few	语言需要对较小数字进行特殊处理（比如捷克语里的2、3、4；或者波兰语里以2、3、4结尾但不是12、13、14的数。）
many	语言需要对较大数字进行特殊处理（比如马耳他语里以11-99结尾的数）
other	语言不需要对数字进行特殊处理。

例子：

XML文件保存为res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">One song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>
```

XML文件保存为res/values-pl/strings.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">Znaleziono jedną piosenkę.</item>
        <item quantity="few">Znaleziono %d piosenki.</item>
        <item quantity="other">Znaleziono %d piosenek.</item>
    </plurals>
</resources>
```

JAVA代码：

```
int count = getNumberOfSongsAvailable();
Resources res = getResources();
String songsFound =
res.getQuantityString(R.plurals.numberOfSongsAvailable, count,
count);
```

在使用[getQuantityString\(\)](#)方法时，如果字符串包含数字格式化串，则需要传递2个count参数。例如：对于字符串“%d songs found”，第一个count参数选择合适的复数字符串，第二个count参数插入占位符%d中。如果复数字符串资源不包含格式化信息，就不需要给[getQuantityString\(\)](#)传递第三个参数。

Formatting and Styling

下面是关于字符串资源格式化和样式应该了解的比较重要的地方。

撇号和引号的转义

如果字符串里包含撇号或引号，必须进行转义，或者把整个串封闭在与当前引号不同的成对的引号内。下面是一些有效或无效的字符串示例：

```
<string name="good_example">This'll work</string>
<string name="good_example_2">This\ 'll also work</string>
<string name="bad_example">This doesn't work</string>
<string name="bad_example_2">XML encodings don't work</string>
```

字符串的格式化

如果需要使用 [String.format\(String, Object...\)](#) 格式化字符串，可以把格式化参数放在字符串（string）资源里。比如存在以下资源：

```
<string name="welcome_messages">Hello, %1$s! You have %2$d new
messages.</string>
```

此例中存在两个参数：%1\$s 是个字符串，%2\$d 是个数字，在应用程序中可以用如下方式用参数来格式化字符串：

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages),
username, mailCount);
```

用HTML标记来样式化

可以用HTML 标记来为字符串加入样式。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

支持以下HTML元素：

- 文本加粗bold。
- <i>文本变斜体italic。
- <u>文本加下划线underline。

有时可能要创建一个样式的文本资源，并可作为格式化串使用。通常这不能生效，因为 [String.format\(String, Object...\)](#) 方法会去除字符串内的所有的样式信息。解决方法是写入一段转义后的HTML标记，然后在格式化后再用 [fromHtml\(String\)](#) 恢复出这些样式。例如：

1. 将样式化的文本资源存储为转义后的HTML字符串：

```
<resources>
  <string name="welcome_messages">Hello, %1$s! You have &lt;b>%2$d
new messages&lt;/b>.</string>
</resources>
```

在这个格式化字符串里，加入了一个元素。注意左尖括号是用标记**<**转义过的**HTML**。

2. 然后，按照通常方式格式化字符串，并调用**fromHtml(String)**把**HTML**文本转换成带样式的文本。

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages),
username, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

因为**fromHtml(String)**方法会格式化所有的**HTML**内容，所以要确保用**htmlEncode(String)**对带格式化文本的字符串内所有可能的**HTML**字符进行转义。比如，如果要把可能包含诸如“<”或“&”等字符的串作为参数传给**String.format()**，那么在格式化之前必须对这些字符进行转义。格式化之后再把字符串传入**fromHtml(String)**，这些特殊字符就能还原成本来意义了。例如：

```
String escapedUsername = TextUtil.htmlEncode(username);

Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages),
escapedUsername, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

来自“[index.php?title=String_Resources&oldid=7805](#)”



Style Resource

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

<http://developer.android.com/guide/topics/resources/style-resource.html>

作者：smilysas

更新时间：2012年8月2日

样式资源定义了用户界面的格式和外观。样式能被应用到单独的[View](#)（通过写入layout布局文件）或者整个[Activity](#)及应用程序（通过置入manifest清单文件）。

关于创建及应用样式的更多信息，请参阅[Styles and Themes](#)。

注意：样式是用名称属性（并非XML文件名）来直接引用的简单类型资源。因此，可以把样式资源和其他简单类型资源一起放入一个XML文件的一个<resources>元素下。

文件定位：

res/values/filename.xml

文件名可任意指定，元素的名称将被用作资源ID。

资源引用：

XML代码:@[package:]style/style_name

语法：

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style
        name="style_name"
        parent="@[package:]style/style_to_inherit">
        <item
            name="[package:]style_property_name"
            >style_value</item>
    </style>
</resources>

```

元素：

<resources>

必须定义的。必须是根节点。没有属性值。

<style>

定义单个样式。包含<item>元素。

属性：

name

String类型。必须定义的。样式的名称，作为资源ID应用到视图、活动或应用程序。

parent

样式资源。本样式的父资源，将继承其Style属性。

<item>

为样式定义单个属性。必须是<style> 元素的子元素。

属性：

name

属性资源。必须定义的。指定样式属性的名称，必要的话带上包前缀（例如 android:textColor）。

例子：

样式XML文件（存放在res/values/）：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText" parent="@style/Text">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

应用以上样式到TextView的XML文件（存放在res/layout/）：

```
<?xml version="1.0" encoding="utf-8"?>
<EditText
    style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

来自“[index.php?title=Style_Resource&oldid=9179](#)”



More Resource Types

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： jpengfly

原文链接：<http://developer.android.com/guide/topics/resources/more-resources.html>

目录

[[隐藏](#)]

[1 More Resource Types\(更多的资源类型\)](#)

- [1.1 Bool](#)
- [1.2 Color \(颜色\)](#)
- [1.3 Dimension\(尺码\)](#)
- [1.4 ID](#)
- [1.5 Integer\(整数\)](#)
- [1.6 Integer Array\(整型数组\)](#)
- [1.7 Typed Array](#)

More Resource Types(更多的资源类型)

这个页面定义了更多可以外部化的资源类型,包括:

Bool (布尔)

XML 资源, 承载布尔值

Color (颜色)

XML 资源 , 承载颜色值 (一个16进制的颜色)

Dimension (尺码)

XML 资源 , 承载尺码值 (带有一个度量单位).

ID

XML 资源 , 为程序资源和组件提供一个唯一的标识。

Integer

XML 资源 , 承载一个整形值。

Integer Array

XML 资源 , 承载一个整形数组。

Typed Array

XML 资源 , 提供一个TypedArray (你可以将这个TypedArray作为一个drawables数组).

Bool

在XML中定义一个布尔值。

注意:bool是一种简单的资源, 它用name属性提供的值来引用(而不是XML文件的名称)。这样,你可以将bool资源与其他简单的资源放在一个XML文件中的<resource>元素里。

: res/values/filename.xml

< bool >

文件位置

文件名是任意的。

元素的名

称name将用作资源ID。

资源引用: Java: R.bool.bool_name XML: @[package:]bool/bool_name

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="bool_name">[true | false]</bool>
</resources>
```

元素: <resources> 必须要有, 而且必须是根节点。 没有任何属性。

<bool> bool值:true或false。

属性:name 类型为String字符串。一个bool值的名称。该名称将用作资源ID。

例子: XML文件存在 res/values-small/bools.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="screen_small">true</bool>
    <bool name="adjust_view_bounds">true</bool>
</resources>
```

程序代码获取boolean: Resources res = getResources(); boolean screenIsSmall = res.getBoolean(R.bool.screen_small);

下面的XML布局文件将boolean作为属性:

```
<ImageView
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:src="@drawable/logo"
    android:adjustViewBounds="@bool/adjust_view_bounds" />
```

Color (颜色)

在XML中定义一个颜色值。颜色被指定为RGB值和alpha channel (阿尔法通

道)。您可以在任何地方使用color资源，只要它接受十六进制的颜色值表示。当XML文件需要drawable资源时，也可以使用color资源(例如,android:drawable="@color/green")。

值通常以#字符开头，接着Alpha-Red-Green-Blue (阿尔法-红-绿-蓝) 信息，以下面的某种格式：

1. RGB
2. ARGB
3. RRGGBB
4. AARRGGBB

注意:color是一种简单的资源，它用name属性提供的值来引用(而不是XML文件的名称)。这样,你可以将color资源与其他简单的资源放在一个XML文件中的<resource>元素里。

文件位置: res/values/colors.xml 文件名是任意的。<color>元素的名称将用作资源ID。

资源引用: Java: R.color.color_name XML: @[package:]color/color_name

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_name">hex_color</color>
</resources>
```

元素: <resources> 必须的且必须是根节点。没有任何属性。

<bool> 如上所诉， color用16进制表示。

属性:name String。一个color值的名称。该名称将用作资源ID。

例如: XML文件保存在res/values/colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <color name="translucent_red">#80ff0000</color>
</resources>
```

程序代码取回color资源: Resources res = getResources(); int color = res.getColor(R.color.opaque_red);

XML布局文件接受color作为属性值:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/translucent_red"
    android:text="Hello" />
```

Dimension(尺码)

在XML中定义dimension的值。dimension是指定一个数字,后跟一个计量单位。例如:10px,2in、5sp。以下的计量单位都被Android支持:

dp

Density-independent Pixels (分辨率无关的像素) - 基于屏幕上的物理密度的抽象单位。这些单位是相对于一个160 DPI (点每英寸) 屏幕, 其中1DP是大致相等为1px。在分辨率较高的屏幕上运行时, 绘制1dp的像素数量按比例扩大, 扩大基于屏幕的dpi。同样, 在分辨率较低的屏幕上时, 1DP使用的像素数缩减。dp-to-pixel的比例会随屏幕分辨率改变, 但不一定成正比。使用dp单位 (而不是像素单位) , 是一个简单的解决方案, 以使你的布局调整大小视图的尺寸适应不同的屏幕分辨率。换句话说, 它为不同设备上的UI元素提供了一致的真实大小。

sp

Scale-independent Pixels(大小无关的像素)- 这个和dp单位很像, 但是它基于用户的字体大小偏爱来调整。当指定字体大小时, 推荐你使用这种单位, 所以他们为屏幕分辨率和用户偏爱来调整。

pt

Points (点) 1/72英寸， 基于物理屏幕的大小。

px

Pixels(像素)对应于屏幕上的实际像素。这个测量单位以并不推荐,因为实际的显示可能在不同设备;每个设备可能有不同数量的每英寸个像素,可能有更多或者更少的像素点在屏幕上可见。

mm

Millimeters(毫米)-基于物理屏幕的大小。

in

Inches(英寸)- 基于物理屏幕的大小。

注意:dimension是一种简单的资源，它用name属性提供的值来引用(而不是XML文件的名称)。这样,你可以将dimension资源与其他简单的资源放在一个XML文件中的<resource>元素里。

文件位置: res/values/filename.xml 文件名是任意的。<dimen>元素的名称将用作资源ID。

资源引用: Java: R.dimen.dimension_name XML:
@[package:]dimen/dimension_name

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="dimension_name">dimension</dimen>
</resources>
```

元素: <resources>

必须的。必须是根节点。

没有任何属性。

<bool>

如上所述, dimension用float数表示, 后面跟计量单位(dp, sp, pt, px, mm, in)

属性:name

String。一个dimension值的名称。该名称将用作资源ID。

例如: XML文件保存在res/values/dimens.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="ball_radius">30dp</dimen>
    <dimen name="font_size">16sp</dimen>
</resources>
```

程序代码取回dimension:

```
Resources res = getResources();
float fontSize = res.getDimension(R.dimen.font_size);
```

XML布局文件接受dimensions作为属性

```
<TextView
    android:layout_height="@dimen/textview_height"
    android:layout_width="@dimen/textview_width"
    android:textSize="@dimen/font_size" />
```

ID

XML中定义的一个唯一的资源。使用你在<item>元素中提供的name, Android开发工具在项目中的R.java类里创建一个唯一的整数, 您可以使用它作为应用程序资源的标识符(例如, 在您的UI布局中的View)或一个唯一的整数, 用在你的程序代码中(例如, 作为一个对话框的ID或result代码)。

注意:ID是一种简单的资源，它用name属性提供的值来引用(而不是XML文件的名称)。这样,你可以将dimension资源与其他简单的资源 放在一个XML文件中的<resource>元素里。同样，记住， ID资源并不是引用一个真实的资源项；它仅仅是一个唯一的ID你可以附加到其他资源， 或者当作你程序中的唯一整数。

文件位置:

res/values/filename.xml

文件名是任意的。

资源引用:

Java: R.id.name

XML: @[package:]id/name

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item
        type="id"
        name="id_name" />
</resources>
```

元素:

<resources>

必须要有的且必须是根节点。

没有任何属性。

<item>

定义唯一的ID。不带值， 只有属性。

属性:type

必须是“id”

name

String。唯一的ID名称。

例如: XML文件保存在res/values/ids.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item type="id" name="button_ok" />
    <item type="id" name="dialog_exit" />
</resources>
```

然后, 这个layout片段用"button_ok"作为一个Button窗体的ID

```
<Button android:id="@+id/button_ok"
        style="@style/button_style" />
```

注意, android:id值在id引用中不包括加号, 因为id已经存在, 在上面的ids.xml例子中定义了。(当你为一个XML资源指定一个 ID, 使用加上加号——用格式 android:id="@+id/name"——意味着"name"ID是不存在的, 应该被创建)。

另一个例子, 下面的代码片段使用"dialog_exit"ID作为一个对话框的惟一标识符:

```
showDialog(R.id.dialog_exit);
```

在同一个应用程序中, 当创建一个对话框时"dialog_exit"ID是对照:

```
protected Dialog onCreateDialog(int) (int id) {
    Dialog dialog;
    switch(id) {
        case R.id.dialog_exit:
            ...
            break;
        default:
            dialog = null;
    }
    return dialog;
}
```

Integer(整数)

定义在XML中的整数。

注意:integer是一种简单的资源，它用name属性提供的值来引用(而不是XML文件的名称)。这样,你可以将integer资源与其他简单的资源放在一个XML文件中的<resource>元素里。

文件位置:

res/values/filename.xml

文件名是任意的。<integer>元素的name属性当作资源ID。

资源引用:

Java: R.integer.integer_name

XML: @[package:]integer/integer_name

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer
        name="integer_name">integer</integer>
</resources>
```

元素:

<resources>

必须的。必须是根节点。

没有任何属性。

<integer>

整数。

属性:

name

String。integer的名称。这将当作资源ID。

例如：XML文件存储在res/values/integers.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="max_speed">75</integer>
    <integer name="min_speed">5</integer>
</resources>
```

程序代码中取出integer：

```
Resources res = getResources();
int maxSpeed = res.getInteger(R.integer.max_speed);
```

Integer Array(整型数组)

XML中定义的整型数组

注意：integer array是一种简单的资源，它用name属性提供的值来引用（而不是XML文件的名称）。这样，你可以将integer array资源与其他简单的资源放在一个XML文件中的<resource>元素里。

文件位置：

res/values/filename.xml

文件名是任意的。<integer-array>元素的name属性当作资源ID。

资源引用：

Java: R.array.string_array_name

XML: @[package:]array.integer_array_name

语法：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="integer_array_name">
        <item>integer</item>
    </integer-array>
</resources>
```

元素：

<resources>

必须的。必须是根节点。

没有任何属性。

<string-array>

定义一个整型数组。包含一个或多个子元素<item>

属性： android:name

String。array的名称。这个名称被用作资源ID引用array

<item>

整数。这个值可以引用到另一个整型资源。必须是<integer-array>的子元素。

没有属性

例如： XML文件存储在res/values/integers.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="bits">
        <item>4</item>
        <item>8</item>
        <item>16</item>
        <item>32</item>
    </integer-array>
</resources>
```

程序代码中取出integer array:

```
Resources res = getResources();
int[] bits = res.getIntArray(R.array.bits);
```

Typed Array

TypedArray在XML中定义。您可以使用它来创建一个其他资源的数组,比如**drawables**。请注意数组不需要类型一致,所以你可以创建一个混合不同资源类型的数组,但你必须知道有什么数据类型,以及数据类型在数组的什么位置,这样就可以通过**TypedArray**的get...()方法正确的获得每一个item (项)。

注意:**integer array**是一种简单的资源,它用**name**属性提供的值来引用(而不是XML文件的名称)。这样,你可以将**integer array**资源与其他简单的资源放在一个XML文件中的<resource>元素里。

文件位置:

res/values/filename.xml

文件名是任意的。<array>元素的**name**属性当作资源ID。

编译资源数据类型:

TypedArray的资源指针。

资源引用:

Java: R.array.array_name

XML: @[package:]array.array_name

语法:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array
        name="integer_array_name">
        <item>resource</item>
    </array>
</resources>
```

元素：

<resources>

必须要有且必须是根节点。
没有任何属性。

<array>

定义一个整型数组。包含一个或多个子元素<item>

属性： android:name

String。 array的名称。这个名称被用作资源ID引用array

<item>

通用资源。这个值可以引用到某一资源或者简单的数据类型。必须是<array>的子元素。

没有属性

例如： XML文件存储在 res/values/arrays.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="icons">
        <item>@drawable/home</item>
        <item>@drawable/settings</item>
        <item>@drawable/logout</item>
    </array>
    <array name="colors">
        <item>#FFFF0000</item>
        <item>#FF00FF00</item>
        <item>#FF0000FF</item>
    </array>
</resources>
```

程序代码中取出每个array(数组)， 然后获取每个array(数组)中的第一个条目：

```
Resources res = getResources();
TypedArray icons = res.obtainTypedArray(R.array.icons);
file:///D:/guide/More_Resource_Types[2015/9/23 19:16:44]
```

```
Drawable drawable = icons.getDrawable(0);  
TypedArray colors = res.obtainTypedArray(R.array.colors);  
int color = colors.getColor(0, 0);
```

来自“[index.php?title=More_Resource_Types&oldid=9189](#)”



Animation and Graphics

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Handsomedylan

原文链接：<http://developer.android.com/intl/zh-CN/guide/topics/graphics/index.html>

动画与图画

用Android强大的图形特性如OpenGL，硬件加速，以及内置UI动画来使你的应用的外观和性能达到最棒。

[更多 >](#)

目录

[[隐藏](#)]

[1 博客文章](#)

- [1.1 Android 4.0 Graphics and Animations](#)
- [1.2 Introducing ViewPropertyAnimator](#)
- [1.3 Android 3.0 Hardware Acceleration](#)

[2 Training](#)

- [2.1 Displaying Bitmaps Efficiently](#)

博客文章

[Android 4.0 Graphics and Animations](#)

年初发布的Android 3.0 带有一个全新的二维渲染管道来支持平板上的硬件加速。这使所有UI工具上的画图都由GPU来执行。更令人惊喜的是，Android4.0也就是Ice Cream Sandwich带来了它的升级版。

[Introducing ViewPropertyAnimator](#)

这个全新的动画系统使任何对象的任意动画方式都变得很容易，而且还在3.0的View类添加了全新特性。在3.1的版本中，我们添加了一个小的通用类，来使这些特性的动画更加简单。

[Android 3.0 Hardware Acceleration](#)

硬件加速图形对于Android平台已不再是什么新鲜事，举例来说，它经常被用于合成窗口或是OpenGL游戏，但是有了这个新的渲染管道后，应用可以进一步提升性能。

Training

[Displaying Bitmaps Efficiently](#)

这节课涵盖了一些处理和下载位图对象的常用技术，这些技术将使你的UI对象保持响应，同时避免超出应用的内存限制。

来自“[index.php?title=Animation_and_Graphics&oldid=8955](#)”



Animation and Graphics Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

动画与绘图综述

Android提供了一系列强大的API来把动画加到UI元素中，以及绘制订制的2D和3D图像中去。下面的几节将综述这些可用的API以及系统的功能，同时帮你做出最优的选择。

动画

Android框架提供了两种动画系统：属性动画（在Android3.0中引进）以及视图动画。这两种动画系统都有变化的选择，但是总的来说，属性动画系统是更好的选择，因为它更加灵活，并提供了更多的特性。在这两种系统之外，你可以使用帧动画，即你可以加载画图资源，并一帧接一帧的显示它们。

[属性动画](#)

在Android3.0中引进 (API level 11)，属性动画系统可以使任何对象的属性成为动画，包括那些不会在屏幕上显示的对象。这个系统是可扩展的，而且也适用于自定义类型。

[视图动画](#)

视图动画则是比较旧的系统，只能用于View类。它相对更容易被设置，而且足够满足许多应用的需求。

[帧动画](#)

帧动画即一个接一个的显示[Drawable](#)资源，就像放一部电影。如果你

想要的动画可以简单的用资源来表示，比如一系列的位图，那就可以用这种方法。

2D 和 3D 图形

当你写一个应用时，确定你的图形需求十分的重要，不同的图形任务最好用不同的技术来实现。举例来说，一个稳定的应用和一个互动游戏的图形和动画是有很大不同的。在这里，我们要讨论你在Android上绘图的一些方式，以及它们分别适用于哪些任务。

画布与绘图对象

Android 提供了一系列的[View](#) 小部件来为一系列的用户界面提供广泛的支持。你也可以继承这些小部件来修改它们的外表或行为方式。除此以外，你可以用Canvas类里的各种绘图方法订制2D视图，或创建绘图对象。

硬件加速

从Android 3.0开始，你可以通过画图API来硬件加速大多数的绘画工作，从而进一步提高它们的性能。

OpenGL

Android 在框架API层面和NDK上都支持 OpenGL ES 1.0 和 2.0。当你想加入少数一些Canvas API不支持的图像增强时，或是你希望平台独立，而不需要高性能时，你可以使用框架API。但使用NDK时，性能会有明显的突破，所以对于许多关注图形的应用，比如游戏，使用NDK是更有利的选择。如果你有很多的本地代码想转到Android 平台上，也可以使用NDK上的OpenGL。想了解更多NDK，请阅读[NDK下载](#)中docs/directory下的文档。

来自“[index.php?title=Animation_and_Graphics_Overview&oldid=6674](#)”



Property Animation

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/graphics/prop-animation.html>

翻译：croftwql

更新：2012.06.11

[动画:](#)

目录

- [1 属性动画](#)
 - [1.1 属性动画工作机制](#)
 - [1.2 属性动画与视图动画的不同](#)
 - [1.3 API概述](#)
 - [1.4 ValueAnimator动画](#)
 - [1.5 Animating with ObjectAnimator](#)
 - [1.6 AnimatorSet创编动画](#)
 - [1.7 动画听众](#)
 - [1.8 动画布局的变化，以ViewGroups](#)
 - [1.9 使用一个TypeEvaluator](#)
 - [1.10 使用插值](#)
 - [1.11 指定关键帧](#)
 - [1.12 动画视图](#)
 - [1.12.1 ViewPropertyAnimator的动画](#)
 - [1.13 在XML定义动画](#)

属性动画

属性动画系统是一个强大的框架，它允许你动画几乎所有的东西。无论是否它绘制到屏幕上或没有，你可以定义一个动画改变任何对象的属性伴随时间的推移。属性动画改变了属性（对象的一个域）的值超过指定的时间长度。对应动画的东西，如指定你想要动画的对象属性，例如一个对象在屏幕中的位置，要动画多久，和动画之间的距值。

属性动画系统可让您定义动画以下特点：

- **Duration(时间):**您可以指定动画的持续时间。默认长度是300毫秒。
- **Time interpolation(时间插值):**定义了动画变化的频率。
- **Repeat count and behavior(重复计数和行为):**您可以指定是否有一个动画的重复，当它到达时间结束，如何多次重复的动画。您还可以指定是否要反向播放动画。把它设置为扭转起着动画向前然后向后反复，直到重复次数达到。
- **Animator sets(动画设置):**你能按照一定的逻辑设置来组织动画，一起播放或顺序或指定延迟。
- **Frame refresh delay(帧刷新延迟):**您可以指定如何经常刷新你的动画帧。默认设置每10毫秒刷新，但在您的应用程序可以指定刷新帧的速度，最终取决于系统整体的状态和提供多快服务的速度依据底层的定时器。

属性动画工作机制

首先，让我们去如何动画一个简单的例子。图1描绘了一个假想的动画对象的x属性，代表其在屏幕上的水平位置。动画的持续时间为40毫秒和旅行的距离是40像素。每隔10毫秒，这是默认的帧刷新速率，物体水平移动10个像素。在40ms的结束，动画停止，对象在水平位置40结束。这是一个线性插值动画的例子，这意味着对象在一个恒定的速度移动。



图1线性动画的例子。

您还可以指定动画有一个非直线插补。图2说明了一个假想的对象，加速在开头动画，在动画结束时减速。对象仍然在40毫秒移动40个像素，但非线性。在开始的时候，这个动画加速的中间点，然后从中间点减速，直到动画结束。如图2所示，动画的开始和结束移动距离小于中间。



图2非线性动画的例子。

让我们看在属性动画系统的重要组成部分，如何计算像上面显示的动画的详细介绍。图3描述了主要类是怎么工作的。



图3。动画是如何计算的

ValueAnimator对象保持动画的实时跟踪，如动画已经运行的时间，和当前动画的属性值。

在**ValueAnimator**封装**TimeInterpolator**，它定义动画插值，和**TypeEvaluator**，它定义了如何计算的动画属性的值。例如，如图2，使用的**TimeInterpolator**将是**AccelerateDecelerateInterpolator**和**TypeEvaluator**的将会是**IntEvaluator**的。

启动动画，创建一个**ValueAnimator**，给你想要的动画开始和结束值，定义动画的持续时间。当你调用的**start()**动画开始。在整个动画，**ValueAnimator**计算经过部分的分数介于0和1，基于动画的持续时间和多少时间已过。经过的部分代表，动画已完成的时间百分比，0表示0%和100%1的含义。例如，在图1中，在 $t = 10$ 毫秒时间的比例是0.25，因为总工期为 $T = 40$ 毫秒。计算经过部分**ValueAnimator**时，它调用**TimeInterpolator**当前设置，计算插值分数。一个插值分数经过部分映射到一个新的，考虑到设置的时间内插的分数。例如，在图2中，因为动画慢慢加速，插约0.15分数，是比过去0.25部分少，在 $t = 10$ 毫秒内。在图1中，插值分数始终是经过分数相同。

当插值分数计算，**ValueAnimator**的调用适当的**TypeEvaluator**，计算你的动画属性值，基于内插的分数，起始值，结束值和动画。例如，在图2中，插值部分是在 $t=0.15$ 在10毫秒内，所以当时时间属性值为 $0.15 \times (40 - 0)$ ，或6。

关于如何使用属性动画，系统的**com.example.android.apis.animation**包中的**API**演示示例项目提供了很多例子。

属性动画与视图动画的不同

视图动画系统仅对动画视图对象提供了性能，因此想要设置非视图对象的动画，必须用自己的代码来实现。视图动画系统受限，它仅公开视图对象进行动画处理的一些方面。比如视图的缩放和旋转。

视图动画系统的另一个缺点是它只能修改视图绘制地址而不是视图本身。例如要绘制在屏幕上移动的按钮，按钮绘制正确，但是点击按钮的实际位置并没有改变，因此要实现自己的逻辑来处理此问题。

属性动画系统会完全消除这些限制。可以为任何对象的属性设置对话，而对象本身实际已被修改。属性动画系统在执行动画方面更加强大。在高层次上，指定想要动态显示的属性，比如颜色，位置或大小，同时可以定义动画比如插值和多个动画同步等方面。

该视图动画系统可以快速设置，并需要较少的编写代码。如果视图动画完成所需一切，或者现有代码已经按照所需方式工作，就没有必要使用属性动画系统。

API 概述

您可以在`android.animation`里面找到属性动画系统大部分API。因为视图的动画系统已经定义了许多插值在`android.view.animation`，你可以使用属性动画系统的插值。下表描述的属性动画系统的主要组成部分。

`Animator`类提供了用于创建动画的基本结构。你通常不使用这个类，因为它直接提供最基本的功能，必须扩展到完全支持动画值。以下子类扩展`Animator`：

表1。 动画家(Animators)

类	描述
<code>ValueAnimator</code>	属性动画时序引擎也计算属性动画的值。它拥有所有的核心功能，计算动画值，并包含每个动画，有关时序的详细信息是否动画重复，听众接收更新事件，并设置自定义类型的能力评估。有两件，以生动活泼的属性：动画值计算和设置这些对象的属性动画值。 <code>ValueAnimator</code> 不进行第二件，所以你一定要更新计算值 <code>ValueAnimator</code> 和修改你想用自己的逻辑动画的对象。请参阅有关更多信息Animating with <code>ValueAnimator</code> 部分。
<code>ObjectAnimator</code>	<code>ValueAnimator</code> 的子类，允许你设置一个目标对

象和对象属性的动画。当计算出一个新的动画值，本类更新相应的属性。你大部分情况使用**ObjectAnimator**，因为它使得动画的目标对象的值更简单。然而，有时你直接使用**ValueAnimator**，因为**ObjectAnimator**有一些限制，如对目标对象目前要求的具体acessor方法。

AnimatorSet

提供机制，以组合动画一起，让他们关联性运行。你可以设置动画一起播放，顺序，或在指定的延迟之后。请参阅有关部分*Choreographing multiple animations with Animator Sets*更多信息。

评估人员告诉属性动画系统如何计算一个给定的属性值。他们采取的时机，是由一个数据**Animator**类，动画的开始和结束值，并计算基于此数据属性的动画值。属性动画系统提供了以下评价：

表2。 评价者(Evaluators)

类/接口	描述
IntEvaluator	默认的计算器来计算int属性值
FloatEvaluator	默认评估值来计算浮动属性。
ArgbEvaluator	默认的计算器计算值表示为十六进制值的色彩属性。
TypeEvaluator	一个接口，允许你创建自己的评估。如果你是动画对象的属性，不是一个整数，浮点数，或颜色，你必须实现的 TypeEvaluator 接口指定如何计算对象属性的动画值。您也可以指定自定

义TypeEvaluator整数，浮点数，颜色值以及，如果你想对比默认行为来处理这些类型的不同。请参阅有关部分[Using a TypeEvaluator](#)更多关于如何编写自定义评估信息。

一个时间插补定义如何在一个动画的特定值作为时间函数的计算。例如，你可以指定动画发生线性在整个动画，这意味着动画均匀地移动整个时间，或者可以指定使用非线性时间的动画，例如，在开始加速，并在最后减速动画。表3说明中所含的插值[android.view.animation](#)。如果没有提供插值适合您的需要，实施[TimeInterpolator](#)接口，并创建自己的。请参阅[Using interpolators](#)如何编写一个定制的插补的更多信息。

表3。 插值(Interpolators)

类/接口	描述
<code>AccelerateDecelerateInterpolator</code>	插补，其变化率慢慢开始和结束，但通过中间加速。
<code>AccelerateInterpolator</code>	插补，其变化率开始缓慢，然后加快。
<code>AnticipateInterpolator</code>	内插的变化开始落后，然后向前甩。
<code>AnticipateOvershootInterpolator</code>	内插的变化，开始落后，甩向前过冲目标值，然后终于可以追溯到最终值。
<code>BounceInterpolator</code>	插补，其变化在最后反弹。

CycleInterpolator	内插动画重复指定的周期数。
DecelerateInterpolator	插补，其变化的速度开始很快，然后减速。
LinearInterpolator	插补，其变化率是恒定的
OvershootInterpolator	内插的变化甩向前和过冲的最后一个值，然后回来。
TimeInterpolator	一个接口，使您实现自己的插补。

ValueAnimator动画

[ValueAnimator](#)类让你动画动画的持续时间由某种类型的值指定了一套整，浮，或颜色值动画。您获得通过[ValueAnimator](#)调用工厂方法之一：ofInt()，ofFloat()，or ofObject()。例如：

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);
animation.setDuration(1000);
animation.start();
```

在这段代码中，[ValueAnimator](#)开始动画的计算值，1000毫秒，当时 间为0和1之间，运行的start()方法。你也可以指定一个自定义类型的动画通过执行下列操作：

```
ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(),
startPropertyValue, endPropertyValue);
animation.setDuration(1000);
animation.start();
```

在这段代码中，[ValueAnimator](#)开始计算之间的动画值，使用所提供的逻辑MyTypeEvaluator 的start () 方法运行时间为1000毫秒，

当startPropertyValue和endPropertyValue。

然而，前面的代码片段，有没有对象的实际效果，因为在[ValueAnimator](#)不直接操作对象或属性。最有可能的事情，你想要做的是修改这些计算值要进行动画的对象。你定义在听众[ValueAnimator](#)妥善处理动画的寿命期间的重要事件，如帧更新。实施的听众时，你可以通过调用特定的帧刷新计算值[getAnimatedValue\(\)](#)。对听众的更多信息，请参阅有关部分[Animation Listeners](#)。

Animating with ObjectAnimator

The ObjectAnimator is a subclass of the ValueAnimator (discussed in the previous section) and combines the timing engine and value computation of ValueAnimator with the ability to animate a named property of a target object. This makes animating any object much easier, as you no longer need to implement the ValueAnimator.AnimatorUpdateListener, because the animated property updates automatically.

Instantiating an ObjectAnimator is similar to a ValueAnimator, but you also specify the object and the name of that object's property (as a String) along with the values to animate between:

```
ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);
anim.setDuration(1000);
anim.start();
```

To have the ObjectAnimator update properties correctly, you must do the following:

- The object property that you are animating must have a setter function (in camel case) in the form of set<propertyName>(). Because the ObjectAnimator automatically updates the property during animation, it must be able to access the property with this setter method. For example, if the property name is foo, you need to have a setFoo() method. If this setter method does not exist, you have three options:
 - Add the setter method to the class if you have the rights to do so.

Use a wrapper class that you have rights to change and have that wrapper receive the value with a valid setter method and forward it to

the original object.

Use ValueAnimator instead.

- If you specify only one value for the values... parameter in one of the ObjectAnimator factory methods, it is assumed to be the ending value of the animation. Therefore, the object property that you are animating must have a getter function that is used to obtain the starting value of the animation. The getter function must be in the form of get<propertyName>(). For example, if the property name is foo, you need to have a getFoo() method.
- The getter (if needed) and setter methods of the property that you are animating must operate on the same type as the starting and ending values that you specify to ObjectAnimator. For example, you must have targetObject.setPropName(float) and targetObject.getPropName(float) if you construct the following ObjectAnimator:

```
ObjectAnimator.ofFloat(targetObject, "propName", 1f)
```

- Depending on what property or object you are animating, you might need to call the invalidate() method on a View force the screen to redraw itself with the updated animated values. You do this in the onAnimationUpdate() callback. For example, animating the color property of a Drawable object only cause updates to the screen when that object redraws itself. All of the property setters on View, such as setAlpha() and setTranslationX() invalidate the View properly, so you do not need to invalidate the View when calling these methods with new values. For more information on listeners, see the section about Animation Listeners.

AnimatorSet创编动画

在许多情况下，你要播放的动画，取决于另一个动画开始或者结束时。Android系统，让你捆绑动画到[AnimatorSet](#)一起，使您可以指定是否要同时，按顺序，或在指定的延迟后开始动画。你可以在对方还[AnimatorSet](#)对象。

从下面的示例代码弹球样品（简单修改）扮演下面的动画 对象以下列方式：

- Plays bounceAnim.

- Plays squashAnim1, squashAnim2, stretchAnim1, and stretchAnim2 at the same time.
- Plays bounceBackAnim.
- Plays fadeAnim.

```

    AnimatorSet bouncer = new AnimatorSet();
    bouncer.play(bounceAnim).before(squashAnim1);
    bouncer.play(squashAnim1).with(squashAnim2);
    bouncer.play(squashAnim1).with(stretchAnim1);
    bouncer.play(squashAnim1).with(stretchAnim2);
    bouncer.play(bounceBackAnim).after(stretchAnim2);
    ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f,
    0f);
    fadeAnim.setDuration(250);
    AnimatorSet animatorSet = new AnimatorSet();
    animatorSet.play(bouncer).before(fadeAnim);
    animatorSet.start();
  
```

对于如何使用动画集更完整的例子，弹弹球在APIDemos样本。

动画听众

与下文所述的听众，你可以听动画的持续时间期间的重要事件。

- [Animator.AnimatorListener](#)
 - [onAnimationStart\(\)](#) - 所谓的动画开始时。
 - [onAnimationEnd\(\)](#) - 在动画结束时调用。
 - [onAnimationRepeat\(\)](#) - 调用时，动画的重演。
 - [onAnimationCancel\(\)](#) - 调用时，动画将被取消。取消动画还可以调用[onAnimationEnd\(\)](#)，不管他们是如何结束。
- [ValueAnimator.AnimatorUpdateListener](#)
 - [onAnimationUpdate\(\)](#) - 在动画的每一帧被调用。听此事件使用所产生的计算值[ValueAnimator](#)在一个动画。使用的值，查询[ValueAnimator](#)对象传递到事件与目前动画值[getAnimatedValue\(\)](#)方法。如果使用[ValueAnimator](#)实现这个监听器是必需的。

这取决于你是动画什么属性或对象，你可能需要浏览强制重绘新的动画值，到屏幕上面积上，调用[invalidate\(\)](#)。例如，动画的绘制对象的颜色属性，仅造成更新到屏幕上时，该对象重绘本身。在视图的全部属性set方法，例如[[http://developer.android.com/reference/android/view/View.html#setAlpha\(float\)](http://developer.android.com/reference/android/view/View.html#setAlpha(float))]和[setTranslationX\(\)](#)初始化视图属性，所以你不用使用新值调用这些方法初始化视图。

可以延长[AnimatorListenerAdapter](#)类，而不是实施的[Animator.AnimatorListener](#)接口，如果你不想执行的所有方法[Animator.AnimatorListener](#)接口。[AnimatorListenerAdapter](#)类提供的方法，你可以选择覆盖的空实现。

例如，[Bouncing Balls](#)(弹弹球)例子在API演示创建一个[AnimatorListenerAdapter](#)在[onAnimationEnd\(\)](#) 回调：

```
ValueAnimator animator = ObjectAnimator.ofFloat(newBall,
"alpha", 1f, 0f);
animator.setDuration(250);
animator.addListener(new AnimatorListenerAdapter() {
    public void onAnimationEnd(Animator animation) {
        balls.remove((ObjectAnimator) animation).getTarget();
    }
})
```

动画布局的变化，以ViewGroups

属性动画系统提供的能力，以动画变化ViewGroup对象以及自己的动画视图对象提供了一个简单的方法。

你可以在一个ViewGroup动画与布局的变化[LayoutTransition](#)类。一个ViewGroup内部的视图，可以通过动画出现和消失，当您添加或删除它们从一个ViewGroup或当你调用一个视图的[setVisibility\(\)](#)的方法[VISIBLE](#)(可见)，[android.view.View # INVISIBLE](#)(隐形)，或去除。在ViewGroup中存在的视图可以动画到你增加或者删除视图的新位置上。你可以通过调用定义在下面的动画[LayoutTransition](#)的对象[android.animation.Animator](#) [setAnimator\(\)](#)在通过动画对象有下列的[LayoutTransition](#)常量：

- 出现(APPEARING) -一个标志，指示在容器中的项目上出现运行的动画。
- CHANGE_APPEARING -一个标志，指示一个新的在容器中项目上出现运行的动画。
- 消失(DISAPPEARING) -一个标志，指示动画的运行项目，从容器中消失。
- CHANGE_DISAPPEARING -一个标志，指示一个从容器中的项目上运行的动画消失。

你可以定义自己的自定义动画这四种类型的事件定制的布局过渡，或只是告诉动画系统使用默认的动画。

在[Layout Animations](#)的API演示示例显示您如何定义布局过渡的动画，然后设置要动画视图对象的动画。

在[Layout Animations By Default](#)其的相应[layout_animations_by_default.xml](#)布局资源文件表明您如何启用在XML ViewGroups的默认布局转换。你唯一需要做的是设置的 Android : animateLayoutChanges 属性为 true 的ViewGroup的。例如：

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@+id/verticalContainer"
    android:animateLayoutChanges="true" />
```

这个属性设置为true，自动动画ViewGroup增加或移除存在于ViewGroup的视图。

使用一个TypeEvaluator

如果你想动画的类型是Android系统没有的，可以通过实施创建自己定义[TypeEvaluator](#)接口。被称为Android系统的类型是整数，浮点数，或一种颜色，这是由支持的[IntEvaluator](#), [FloatEvaluator](#), 和[ArgbEvaluator](#)的类型评估。

只有一个方法来实现在[TypeEvaluator](#)接口，[T, T\) evaluate\(\)](#)方法。这允许您使用您的动画属性的一个适当的值返回在当前点的动画。[FloatEvaluator](#)类演示了如何做到这一点：

```
public class FloatEvaluator implements TypeEvaluator {
    public Object evaluate(float fraction, Object startValue, Object endValue) {
        float startFloat = ((Number) startValue).floatValue();
        return startFloat + fraction * (((Number) endValue).floatValue() - startFloat);
    }
}
```

注：当ValueAnimator（或ObjectAnimator）运行时，它计算消耗动画（0和1之间的值）的一小部分，然后计算出一个插值取决于上什么插补您正在使用的版本。插值分数您的TypeEvaluator，通过接收部分参数，所以你没有考虑到插补计算动画值时。

使用插值

内插定义动画中的具体值作为时间函数的计算。例如，你可以指定动画发生线性在整个动画，这意味着动画均匀地移动整个时间，或者可以指定使用非线性时间的动画，例如，使用加速或减速的开始或结束动画。

动画集代表经过时间的动画内插动画系统收到的一小部分。插值修改这个分数与动画的类型相吻合，它旨在提供。Android系统提供了一个共同插值设置的[android.view.animation](#)包。如果没有这些适合您的需求，可以实现[TimeInterpolator](#)接口，并创建自己的。

作为一个例子，如何默认的插补[AccelerateDecelerateInterpolator](#)和[LinearInterpolator](#)计算的插值分数比较如下。在[LinearInterpolator](#)没有经过部分的影响。[AccelerateDecelerateInterpolator](#)加速到动画和它的减速。下列方法确定这些插值的逻辑：

AccelerateDecelerateInterpolator

```
public float getInterpolation(float input) {
    return (float)(Math.cos((input + 1) * Math.PI) / 2.0f) + 0.5f;
}
```

LinearInterpolator

```
public float getInterpolation(float input) {
    return input;
}
```

下表表示持续1000毫秒的动画，这些插值计算的近似值：

已过毫秒	经过分数/插值的一小部分（线性）	插值分数（加速/减速）
0	0	0

200	.2	.1
400	.4	.345
600	.6	.8
800	.8	.9
1000	1	1

作为该表显示，[LinearInterpolator](#)改变以同样的速度值，为每200ms传递.2。该[AccelerateDecelerateInterpolator](#)更改值比[LinearInterpolator](#)在200毫秒和600毫秒快和在600毫秒和1000毫秒之间慢。

指定关键帧

一个关键帧[Keyframe](#)的对象包括时间/值对，可以让你定义一个特定的状态，在特定时间的动画。每个关键帧也可以有自己的插补控制动画中的前一个关键帧之间的时间，这个关键帧的时间间隔的行为。

实例化一个关键帧[Keyframe](#)的对象，你必须使用工厂方法之一的，[ofInt\(\)](#)，[ofFloat\(\)](#)，或[ofObject\(\)](#)获得适当类型的关键帧。然后，你调用[android.animation.Keyframe...\)](#) [ofKeyframe\(\)](#)工厂方法获得[PropertyValuesHolder](#)对象。一旦你有了这个对象，你可以通过获得一个动画[PropertyValuesHolder](#)对象和对象的动画。下面的代码片段演示了如何做到这一点：

```
Keyframe kf0 = Keyframe.ofFloat( 0f, 0f );
Keyframe kf1 = Keyframe.ofFloat( .5f, 360f );
Keyframe kf2 = Keyframe.ofFloat( 1f, 0f );
PropertyValuesHolder pvhRotation =
PropertyValuesHolder.ofKeyframe("rotation", kf0, kf1, kf2);
ObjectAnimator rotationAnim =
```

```
ObjectAnimator.ofPropertyValuesHolder(target, pvhRotation)
    rotationAnim.setDuration(5000ms);
```

对于如何使用关键帧的更完整的范例，看到[MultiPropertyAnimation](#)在APIDemos的例子。

动画视图

属性动画系统允许精简视图对象的动画和offerse在视图动画系统有几个优点。视图动画系统改造，通过改变，他们绘制的方式查看对象。这是在每个视图的容器处理，因为视图本身没有属性来操作。这导致在动画的查看，但没有造成人员在视图对象本身的变化。这导致的行为，如在原来的位置上仍然存在的对象，即使是在屏幕上的不同位置上绘制。在Android 3.0中，增加了新的属性和相应的getter和setter方法 来消除这一缺点。

属性动画系统可以改变视图对象的实际属性，动画在屏幕上的视图。此外，视图还自动调用[invalidate\(\)](#)方法来刷新屏幕时改变其属性。在新的属性视图类促进属性动画的是：

- **translationX**和**translationY**：这些属性控制的地方查看位于从它的左侧和顶部设置其布局容器的坐标的增量。
- **rotation, rotationX, and rotationY**: 这些属性控制在2D旋转（旋转属性）和3D的支点周围。
- **scaleX** 和 **scaleY** : 这些属性控制周围的支点视角的2D缩放。
- **pivotX**和**pivotY**: 这些属性控制支点的位置，围绕着它旋转和缩放变换发生。默认情况下，支点位于中心的对象。
- **x** 和 **y** : 这些都是简单实用的特性来形容查看其容器的最终位置，左侧和顶部值和**translationX**和**translationY**的价值的总和。
- **alpha** : 代表alpha透明度。此值默认为1（不透明），0代表完全透明（不可见）。

动画视图对象的属性，如它的颜色或旋转值，所有你需要做的是创建一个属性的动画，并指定要动画视图属性。例如：

```
ObjectAnimator.ofFloat(myView, "rotation", 0f, 360f);
```

创建动画的详细信息, 请参阅与动画[ValueAnimator](#)和[ObjectAnimator](#)。

ViewPropertyAnimator的动画

在该[ViewPropertyAnimator](#)并行动画的几个属性查看, 使用单一的基础[Animator](#)对象提供了一个简单的方法。它的行为像一个[ObjectAnimator](#), 因为它修改视图属性的实际值, 但更有效的动画时许多属性一次。此外, 使用的的代码[ViewPropertyAnimator](#)是更简洁, 更易于阅读。下面的代码片段显示在使用多个的差异[ObjectAnimator](#)对象, 一个的单[ObjectAnimator](#), 以及时的[ViewPropertyAnimator](#)同时动画的 x 和 y 属性视图。

Multiple ObjectAnimator objects

```
ObjectAnimator animX = ObjectAnimator.ofFloat(myView, "x", 50f);
ObjectAnimator animY = ObjectAnimator.ofFloat(myView, "y", 100f);
AnimatorSet animSetXY = new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

One ObjectAnimator

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvhY).start();
```

ViewPropertyAnimator

```
myView.animate().x(50f).y(100f);
```

如需更详细的信息[ViewPropertyAnimator](#), 看到相应的Android开发者博客文章

在XML定义动画

属性动画系统, 可以让你声明与XML属性动画, 而不是做编程。在XML中定义的动画, 你可以很容易地在多个活动中重用的动画更容易编辑动画序列。来区分使用那些使用传统的新的属性的动画API 动画框架与Android 3.1开始, 动画文件, 你应该保存的XML文件中的属性的动画res/ animator/ 目录代替res/ anim/。使用

动画的目录名称是可选的，但必要的，如果你想使用的布局编辑工具，在Eclipse的ADT插件（ADT 11.0.0 +），因为ADT只搜索res/animator/目录属性的动画资源。

下列属性动画类有下列XML标记的XML声明支持：

- [ValueAnimator](#) - <animator>
- [ObjectAnimator](#) - <objectAnimator>
- [AnimatorSet](#) - <SET>

下面的示例播放两套对象的动画顺序，第一个是嵌套播放两个对象的动画：

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType" />
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType" />
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f" />
</set>
```

为了运行这个动画，你必须在你的代码XML资源到AnimatorSet对象，然后设置目标对象为所有的动画开始前的动画设置。调用[setTarget\(\)](#)设置一个所有儿童的单目标对象AnimatorSet的作为一种方便。下面的代码演示如何做到这一点：

```
AnimatorSet set = (AnimatorSet)
AnimatorInflater.loadAnimator(myContext,
    R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

为定义属性动画的XML语法的信息，请参阅动画资源[Animation Resources](#)。

[← Back to Animation](#)

View Animation

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/graphics/view-animation.html>

翻译：croftwql

更新：2012.06.14

动画：

视图动画

您可以使用视图的动画系统，在视图上执行的补间动画。补间动画的动画，如起点，终点，大小，旋转，动画的其他普通法方面的资料计算。

补间动画可以执行一系列简单的视图对象的内容转换（位置，大小，旋转，透明度）。所以，如果你有一个[TextView](#)的对象，你可以移动，旋转，成长，或缩小文本。如果它有一个背景图片，背景图片将随着文字转化。[动画包\(animation package\)](#)提供了所有用于补间动画类。

动画指定序列定义补间动画，[XML](#)或Android代码定义。定义一个布局，建议一个[XML](#)文件中，因为它更具可读性，可重复使用，相比编码的动画。在下面的例子中，我们使用[XML](#)。（定义动画，在您的应用程序代码，而不是[XML](#)，了解更多信息，请参阅的[AnimationSet](#)类和其他动画[Animation](#)的子类。）

动画指令定义的转换，你想要发生时，他们会发生，以及多久，他们应采取适用。转换可以连续或同时发生 - 例如，你可以有一个[TextView](#)从左至

右移动的内容，然后旋转180度，或者你可以有文本的举动，并同时旋转。每个转换需要一套这种转变（开始和结束的大小尺寸大小的变化，旋转的起始角度和结束角度，等等），也是一个共同的参数集（例如，开始时间和持续时间）的具体参数。为了让几个转变同时发生，给他们相同的开始时间，让他们连续计算起始时间加上前面的转型期。

动画的XML文件从属于android项目目录的res/anim/文件夹下。该文件必须有一个根元素：这将是一个<alpha>，<scale>，<translate>，<rotate>，插补元素，或<SET>元素，拥有这些元素组（其中可能包括另一个<SET>）。默认情况下，所有的动画指令同时适用。使它们发生的顺序，你必须指定的startOffset的属性，如下面的例子所示。

从下面的XML的ApiDemos之一是用于拉伸，然后同时旋转和旋转视图对象。

```

<set android:shareInterpolator="false">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"

        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />

    <set
        android:interpolator="@android:anim/decelerate_interpolator">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="700"
            android:duration="400"
            android:fillBefore="false" />

        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="700"
            android:duration="400" />
    </set>

```

</set>

屏幕坐标（在这个例子中不使用）(0,0)在左上角，并增加向下和右侧。

一些值，如pivotX，可以指定对象本身或相对于父相对。你想要的东西一定要使用正确的格式（“50”为50%，相对于父，或“50%”为50%，相对本身）。

你可以决定如何随着时间的推移应用分配一个插补[Interpolator](#)转换。Android包括几个插补子类指定不同的速度曲线：例如，[AccelerateInterpolator](#)的讲述一个开始减缓和加快转型。每个人都有一个属性值，可以在XML应用。

此XML保存在项目res/anim/目录下的hyperspace_jump.xml，下面的代码将引用它，并将其应用于从布局到[ImageView](#)的对象。

```
ImageView spaceshipImage = (ImageView)
findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation =
AnimationUtils.loadAnimation(this,R.anim.hyperspace_jump);
spaceshipImage.startAnimation(hyperspaceJumpAnimation);
```

作为一个的替代startAnimation()，你可以定义动画开始时间[Animation.setStartTime\(\)](#)，然后分配与查看[View.setAnimation\(\)](#)动画。

有关更多信息，XML语法，可用的标记和属性，看到动画资源[Animation Resources](#)。

- 注：不管如何动画可以移动或调整，视图边界动画不会自动调整，以适应它。即便如此，动画仍然会超出视图界限，并不会被截断。然而，裁剪会发生，如果动画超过父视图的边界。

[← Back to Animation](#)

来自“[index.php?title=View_Animation&oldid=9016](#)”



Drawable Animation

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/graphics/drawable-animation.html>

翻译：croftwql

更新：2012.06.15

帧动画

绘制动画可以让你加载一系列Drawable的资源陆续创建一个动画。这是一个传统的动画，这是为了发挥不同的图像，就像一卷胶卷，序列创建。[AnimationDrawable](#)类是绘制动画的基础。

虽然你可以在代码中定义动画帧，使用[AnimationDrawable](#)的类的API，它更简单地用一个单一的XML文件，该文件列出了帧组成的动画完成。这种动画的XML文件在你的Android项目的res/drawable/目录下。在这种情况下，指示动画的每一帧的顺序和时间。

XML文件包含一个<animation-list>作为根节点和一系列子元素的<item>节点，每个定义帧：帧帧的持续时间绘制资源。下面是一个例子为绘制动画的XML文件：

```
<animation-list  
    xmlns:android="http://schemas.android.com/apk/res/android"  
        android:oneshot="true">
```

```

<item android:drawable="@drawable/rocket_thrust1"
      android:duration="200" />
<item android:drawable="@drawable/rocket_thrust2"
      android:duration="200" />
<item android:drawable="@drawable/rocket_thrust3"
      android:duration="200" />
</animation-list>

```

此动画运行只有三帧。设置列表属性中的`android:oneshot`为`true`，它将只循环一次到最后一帧停止。如果设置为`false`，则该动画将循环。这个XML保存为`rocket_thrust.xml`在项目的`res/drawable/`目录下。它能增加一个背景图片在视图上，到时候被播放。这里有个界面例子，一个动画是增加在一个[ImageView](#)上和当屏幕被触摸时开始动画：

```

AnimationDrawable rocketAnimation;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    ImageView rocketImage
    = (ImageView) findViewById(R.id.rocket_image);
    rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
    rocketAnimation = (AnimationDrawable)
    rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        rocketAnimation.start();
        return true;
    }
    return super.onTouchEvent(event);
}

```

重要的注意的是对象`AnimationDrawable`的`start()`方法不能在你的`Activity`的`onCreate()`方法里面调用，因为`AnimationDrawable`是尚未完全连接的窗口。如果你想立即播放动画，无需互动，到那时候你可以调用它的[onWindowFocusChanged\(\)](#)方法在你的`Activity`里，它将在你的窗口视图被聚焦时候调用。

有关更多信息，XML语法，可用的标记和属性，看到动画资源[Animation Resources](#)。

[← Back to Animation](#)

来自“[index.php?title=Drawable_Animation&oldid=9018](#)”



Canvas and Drawables

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文：[Canvas and Drawables](#)

翻译：[D.Winter](#)

目录

- [1 Canvas and Drawables](#)
- [2 Canvas 绘图](#)
 - [2.1 View 绘图](#)
 - [2.1.1 SurfaceView 绘图](#)
- [3 Drawables](#)
 - [3.1 由资源图像创建](#)
 - [3.2 由资源中的XML文件创建](#)
 - [3.3 Shape Drawable](#)
 - [3.4 Nine-patch](#)
 - [3.4.1 Example XML](#)

Canvas and Drawables

Android框架提供一系列2D绘画API，它允许你在画布上渲染自定义的图像和定制已经存在的视图的外型与体验。当绘制2D图像时，你将会使用代表性的两种方法：

a.通过布局在视图对象里绘制你的图像或者动画。这种方法，你的图像句柄被系统标准视图层绘制进程控制。你简单定义将图像插入视图中。

b. 直接在画布上绘制图像。此方法，你要亲自调用相应类的[onDraw\(\)](#)方法(*passing it your Canvas*)，或者其中一个画布draw开头的方法(比如[drawPicture\(\)](#))。这样做，你还：在控制任意的动画。

在你绘制简单图像时，方法a是最好的选择。它不需要不断改变也不属于高性能游戏。比如，你在视图中想要绘制静态的或者预先确定的动画在另外的静态应用中。更多信息请查看[Drawables](#)。

当你的应用需要规律性的在画布上重绘时，方法b更为合适。就像电子游戏，你要自己重绘画布。然而，我们还有其他方法来实现：

- 在UI Activity的同一线程里创建的自定义视图组建中，调用[invalidate\(\)](#)再控制[onDraw\(\)](#)回调。
- 或者，在单独的线程中，管理一个[SurfaceView](#)并在画布中绘图(你不需要请求[invalidate\(\)](#))。

Canvas 绘图

当你开发一个应用专门来完成绘制和控制图像的动画，你应该使用画布。画布工作机制就像一个接口，你表面图像将会被绘制。它控制所有绘画调用。通过画布，隐藏的位图(Bitmap)v完成了绘图。它被放在窗体里。

在[onDraw\(\)](#)的回调方法事件中绘图，你只需要调用画布即可。当处理SurfaceView对象时，你也可以从SurfaceHolder.lock [Canvas\(\)](#)获取一个画布。(所有这些场景将在下面的章节中讨论。)然而，你需要创建一个新的画布，你必须定义一个实际执行的位图Bitmap。这个位图Bitmap是画布所必须的。你可以像这样设置一个新画布：

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

现在你的画布将绘制你定义的位图Bitmap。你还可以将这个位图Bitmap移到另外一个画布，使用[Canvas.drawBitmap\(Bitmap,...\)](#)方法。通过View.[onDraw\(\)](#)或SurfaceHolder.lockCanvas()获得画布绘制图像的方法是值得推荐的。(请看下面章节)。

Canvas类有自己的设置绘画方法。像`drawBitmap(...)`, `drawRect(...)`, `drawText(...)`等等。你会用到的其他类也会有`draw()`方法。比如，你有可能有一些[Drawable](#)对象需要放到画布中。Drawable有自己的`draw()`方法，并将Canvas作为参数。

View 绘图

如果你的应用不需要大量的处理或者帧速率(或许是一个棋类游戏，贪吃蛇，或者其他)，你应该考虑创建一个自定义的View组件并通过[View.onDraw\(\)](#)在画布中绘制。Android框架提供了预定义的[Canvas](#)来完成绘图调用，这么做是最方便的。

首先，继承[View](#)类(或者其子类)并且定义[onDraw\(\)](#)回调方法。这个方法在接到Android框架绘图请求时被调用。通过[onDraw\(\)](#)回调执行所有画布绘图方法。

Android框架必须只调用[onDraw\(\)](#)方法。每时每刻你的应用准备被绘制，你必须调用[invalidate\(\)](#)使View无效。这表面你将会看到你的View被绘制，Android将会随即调用你的[onDraw\(\)](#)方法(尽管不能保证实时的回调)。

在你的View组件[onDraw\(\)](#)里, 使用画布进行所有的绘制, 使用各种[Canvas.draw...\(\)](#)方法, 或者其他类的`draw()`方法他们以Canvas作为参数。当[onDraw\(\)](#)完成后, Android框架使用你的Canvas绘制位图Bitmap由系统控制.

- 注意: 为了从一个非主Activity的线程请求[invalidate](#)，你必须调用[postInvalidate\(\)](#) .

继承View类的信息, 请看[自定义组件](#)。

对于一个简单的应用程序, 请看贪吃蛇, 在SDK案例文件: <your-sdk-directory>/samples/Snake/.

SurfaceView 绘图

[SurfaceView](#)是View的一个特殊的子类，提供一个专用的绘图表面，在View层。应用程序的次级线材中加入绘制，这样应用就不需要等到View层绘制完才能被请求。而且，副线程引用的SurfaceView可以绘制和控制自己的画布。

首先，你要创建一个SurfaceView的子类。这个类同时要实现[SurfaceHolder.Callback](#)。这个接口将反馈你底层的信息，比如什么时候创建，改变或者销毁。知道这些事件非常重要，你可以知道什么时候开始绘制，你是否需要对新外观属性的进行调整，和当你停止绘制并销毁一些任务。在SurfaceView类是定义副线程的好地方，它执行画布的所有绘制过程。

你应该通过SurfaceHolder，而非直接控制Surface对象。这样，当你的SurfaceView初始化，通过getHolder()获得[SurfaceHolder](#)。通知SurfaceHolder你要接收SurfaceHolder回调(SurfaceHolder.Callback)通过addCallback()。在SurfaceView class里覆盖所有SurfaceHolder.Callback方法。

为了在副线程中绘制界面画布，你必须传递线程到SurfaceHandler并且通过[lockCanvas\(\)](#)获得画布。你现在可以通过SurfaceHolder获得画布并且在上面绘图了。当你在画布上绘制时，调用[unlockCanvasAndPost\(\)](#)，传递到你的Canvas对象中。界面现在将绘制这个画布只要你关闭它。执行一系列的加锁解锁在你每次想要重绘时。

注：在每次你从SurfaceHolder获得画布时，Canvas之前的状态将会被保留。为了绘制正确的动画效果，你必须重绘所有的界面。比如，你可以通过填充颜色[drawColor\(\)](#)来清除画布之前的状态，或者由[drawBitmap\(\)](#)设置背景图片。否则，你将会看到之前执行的痕迹。

关于程序例子，请看登月者游戏，在SDK例子目录下：`<your-sdk-directory>/samples/LunarLander/`。或者，在案例代码篇章浏览源代码。

Drawables

Android为图形图像提供自定义2D图像处理库。你可以

在[android.graphics.drawable](#)包里找到针对二维绘图的公共类。

本篇章讨论使用Drawable对象绘图的基础，和如何使用Drawable类的多个子类。使用Drawables来绘制帧动画，请看[Drawable Animation](#)。

Drawable的基本含义是内容是可绘的，你会发现Drawable类被扩展用于各种各样的可绘制图像，包括[BitmapDrawable](#), [ShapeDrawable](#), [PictureDrawable](#), [LayerDrawable](#)等等。当然，你也可以继承它们来定义自己的Drawable对象。

有三种方法定义和实例化一个Drawable：使用项目资源中的图像；使用XML文件定义Drawable属性；或者使用一般的类构造函数。下面，我们将讨论前面两种方法（使用构造函数并没有什么新颖的地方）。

由资源图像创建

一个简单的方法添加图像到你的应用中去，就是从你的项目资源里引用图片文件。支持PNG(首选), JPG(还行)和GIF(最好不要)文件类型。当你添加图标、logo时这个方法是首选的。

为了使用图像资源,你要添加文件到项目目录res/drawable/下。从那你可以通过代码或者XML布局引用它们。不管怎样，都要通过资源ID来引用它，ID即为不包含扩展名的文件名(比如，my_image.png就为my_image)。

- 注: 在编译过程中，res/drawable/里的图像资源会被aapt工具自动无失真图像压缩。

比如，一个真彩色PNG文件它不需要大于256色会被调色板修改为8位PNG。这样相同质量的图片所占的内存就会更小。所以要注意图片的二进制文件在编译时会改变。如果你计划读取图片数据流来转换为位图,将他们放到res/raw/文件夹里,这里他们不会被优化。

代码例子

下面的代码片段演示了怎样使用[ImageView](#)通过drawable资源来使用图片并将其添加到布局layout中。

```

LinearLayout mLinearLayout;

protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

// Create a LinearLayout in which to add the ImageView
mLinearLayout = new LinearLayout(this);

// Instantiate an ImageView and define its properties
ImageView i = new ImageView(this);
i.setImageResource(R.drawable.my_image);
i.setAdjustViewBounds(true); // set the ImageView bounds to match
the Drawable's dimensions
i.setLayoutParams(new
Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT));

// Add the ImageView to the layout and set the layout as the
content view
mLinearLayout.addView(i);
setContentView(mLinearLayout);
}

```

除此之外，你可能想控制图片就像Drawable对象一样。所以，通过资源创建Drawable像这样：

```

Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);

```

注：项目中所有唯一的资源只有一个维护状态，不管你用它实例化了多少对象。比如，你由同一个资源图片实例化了2个Drawable对象，然后改变其中一个的属性（比如透明度），然后它也会影响到另外一个。所以，在多个实例使用一个图片资源时，不要直接转换为Drawable，你应该执行tween animation。

====XML例子====

下面的XML片段展示怎么在XML layout中，向[ImageView](#)中添加图像资源。

```

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:tint="#55ff0000"
    android:src="@drawable/my_image" />

```

更多关于项目资源的信息，请看[Resources and Assets](#)。

由资源中的**XML**文件创建

现在，您应该熟悉了Android的用户界面开发应遵循的一些规则。因此，你知道由**XML**来定义对象是多么方便灵活了吧。这种方法不管**Views**还是**Drawables**都适用。如果你要创建一个**Drawable**对象，它不依赖于某个变量或者用户的交互，那么在**XML**里定义它是个好办法。即时你希望用户在体验时去改变它的属性，你应该考虑在**XML**里定义，你可以不断改变它的属性直到它被实例化。

你一旦在**XML**里定义了**Drawable**，在项目文件夹**res/drawable/**里保存。然后，通过调用[Resources.getDrawable\(\)](#)来获取并实例化它，传入**XML**文件的资源ID。(请看下面的例子)

所有**Drawable**子类支持**inflate()**方法，在**XML**里定义和实例化。利用特定的**XML**标签来定义对象属性。

例子：

在**XML**里定义**TransitionDrawable**:

```
<transition
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/image_expand" />
    <item android:drawable="@drawable/image_collapse" />
</transition>
```

XML文件保存为**res/drawable/expand_collapse.xml**，下面的代码演示怎么实例化**TransitionDrawable**和像**ImageView**一样设置其内容。

```
Resources res = mContext.getResources();
TransitionDrawable transition = (TransitionDrawable)
res.getDrawable(R.drawable.expandCollapse);
ImageView image = (ImageView) findViewById(R.id.toggleImage);
image.setImageDrawable(transition);
```

然后这个**transition**会持续向前移动一秒:

```
transition.startTransition(1000);
```

Shape Drawable

当你动态的绘制一些二维图像时，[ShapeDrawable](#)对象会给你很大的帮助。你能以编程的方式绘制基本形状和风格。

[ShapeDrawable](#)是[Drawable](#)的扩展，所以在用到[Drawable](#)的时候你同样可以使用[ShapeDrawable](#)。比如设置[View](#)的背景时，[setBackgroundDrawable\(\)](#)。当然，你也可以在自定义的[View](#)里绘制你的模型。因为[ShapeDrawable](#)有自己的[draw\(\)](#)方法，你可以在[View](#)子类[onDraw\(\)](#)方法里绘制[ShapeDrawable](#)。下面是最基本的例子，绘制[ShapeDrawable](#)对象：

```
public class CustomDrawableView extends View {
    private ShapeDrawable mDrawable;

    public CustomDrawableView(Context context) {
        super(context);

        int x = 10;
        int y = 10;
        int width = 300;
        int height = 50;

        mDrawable = new ShapeDrawable(new OvalShape());
        mDrawable.getPaint().setColor(0xff74AC23);
        mDrawable.setBounds(x, y, x + width, y + height);
    }

    protected void onDraw(Canvas canvas) {
        mDrawable.draw(canvas);
    }
}
```

在构造方法里，[ShapeDrawable](#)被定义为一个[OvalShape](#)(椭圆模型)。然后设置颜色和大小(界限)。如果你不设置大小，模型不会被绘制。如果你不设置颜色，它将默认为黑色。

在Activity里通过代码绘制自定义的View:

```
CustomDrawableView mCustomDrawableView;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mCustomDrawableView = new CustomDrawableView(this);

    setContentView(mCustomDrawableView);
}
```

如果你喜欢通过XML布局文件定义[drawable](#)来代替直接在Activity直接写代码，那它必须覆盖[View\(Context, AttributeSet\)](#)构造方法，它在[View](#)从XML里获取初始化时被调用。然后添加CustomDrawable元素

在XML里：

```
<com.example.shapedrawable.CustomButtonView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

ShapeDrawable类允许你通过**drawable**的**public**方法设置不同的属性。像调整透明度，颜色，抖动，滤色器等。

你也可以由XML定义**drawable shapes**。更多信息请看[Drawable Resources](#)

Nine-patch

[NinePatchDrawable](#)图像是可伸展的位图。当你将它作为背景时，Android会自动适应View的大小。**NinePatch**的一个案例是背景使用标准Android按钮。按钮必须能根据不同的长度自动伸展。**NinePatch drawable**是标准的PNG格式，在最外面一圈额外增加1px的边框。它必须以.**.9.png**为扩展名，并且保存在**res/drawable/**目录下。

这个1px的边框就是用来定义图片中可扩展的和静态不变的区域。在边框的左上角画一个1像素的黑线来表面伸缩区域。你想要多少伸缩区域都可以：他们的绝对尺寸保持相同，所以最大的区域永远是最大的。

你也可以定义一个可选的可移动部分在图片上，通过下面和右边的线。如果一个View对象设置**NinePatch**作为背景，然后改变View上的文字，它会自己伸展，所有的文字将会适应你通过右边和下面的线定义的区域。如果不包括填充线，Android使用左边和上面的线定义可绘区域。

为了阐明这两条线的不同，左和上的线定义图片像素允许被复制以便拉伸图片。下方和右边的线定义相对区域在图片的内容区域。

下面是简单的按钮使用例子：



NinePatch通过左上线来定义伸缩区域，并且通过下右线来定义可绘区域。第一幅图，灰色虚线确定的图片区域将会被拉伸。第二幅图的粉色区域确定一个View里的内容显示区域。如果内容不适合这个区域，图片将会被拉伸。

[Draw 9-patch](#)工具提供了简便的方法来创建我们的NinePatch图片，使用WYSIWYG图形工具。

Example XML

```
<Button id="@+id/tiny"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerInParent="true"
    android:text="Tiny"
    android:textSize="8sp"
    android:background="@drawable/my_button_background" />

<Button id="@+id/big"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerInParent="true"
    android:text="Bi iiiiiig text!"
    android:textSize="30sp"
    android:background="@drawable/my_button_background" />
```

注意宽和高设置为wrap_content使按钮内容自动调整。



来自“[index.php?title=Canvas_and_Drawables&oldid=9026](#)”

OpenGL

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：sumakira

状态：已完成已排版

原文链接：<http://developer.android.com/guide/topics/graphics/opengl.html>

目录

- [1 OpenGL](#)
 - [1.1 The Basics](#)基础知识
 - [1.1.1 OpenGL包](#)
 - [1.2 OpenGL需求声明](#)
 - [1.3 为绘制对象映射坐标](#)
- [2 OpenGL ES 1.0 中的投影和镜头视角](#)
- [3 OpenGL ES 2.0和3.0中的投影和镜头视角](#)
 - [3.1 形状的朝向与卷绕](#)
 - [3.2 OpenGL 版本与设备兼容性](#)
 - [3.2.1 纹理压缩支持](#)
- [4 判断OpenGL扩展名](#)
- [5 检测OpenGL ES版本](#)
- [6 选择OpenGL API 的版本](#)

OpenGL

Android通过使用开放图形库（OpenGL）对高性能的2D和3D图形处理提供支持，尤其是，OpenGL ES API.OpenGL是一个跨平台图形处理的API，并且定义了标准的软件接口用以调用处理3D图形的硬件。OpenGL ES 是专门为嵌入式

设备而开发的OpenGL。从Android 1.0 就已经开始支持OpenGL ES 1.0 和1.1 API，而从Android 2.2(API 8)开始，Android提供了对OpenGL ES 2.0 API 的支持。从Android 4.3(API 18)开始，Android提供了对OpenGL ES 3.0 API 的支持。

注意：Android框架所提供的API和J2ME JSR239 OpenGL ES API非常相似，但是并不完全相同。如果你对J2ME JSR239很熟悉，那么请注意他们之间的不同之处。

The Basics基础知识

Android的框架API和本地开发组件（NDK）都提供对OpenGL的支持。本文着重讲解Android的框架接口。若想获取更多关于NDK的信息，请访问[Android NDK](#). API: [GLSurfaceView](#) 和 [GLSurfaceView.Renderer](#)。如果你是想在你的Android应用中使用OpenGL，那么你首先要做就是要理解如何在activity中来实现这这两个类。

[GLSurfaceView](#)

这是一个继承了[View](#)的类，可以通过使用OpenGL API来绘制和操作管理，这个在功能上和[SurfaceView](#)非常相似。使用这个类的话，你需要先创建一个[GLSurfaceView](#)实例并且给它添加你的[Renderer](#)。然而，如果你想捕获触屏事件，那则需要继承[GLSurfaceView](#) 类并实现对触屏动作的监听，像OpenGL讲解篇里一样[Responding to Touch Events](#)

[GLSurfaceView.Renderer](#)

这个接口定义了在OpenGL的[GLSurfaceView](#)绘制图形的方法。你需要新建一个类来实现这个接口并且通过 [GLSurfaceView.setRenderer\(\)](#)方法来添加到GLSurfaceView实例里。

[GLSurfaceView.Renderer](#)接口需要实现以下方法：

- [onSurfaceCreated\(\)](#): 系统在创建GLSurfaceView时候会调用这个方

法，且只会调用一次。只需要执行一次的动作可以定义在这个方法里，比如说，设置OpenGL环境参数或者初始化OpenGL图形对象。

- 当系统每次需要重新绘制GLSurfaceView时候会调用此方法，同时，此方法也是绘制图形对象的主要执行点。
- [onSurfaceChanged\(\)](#):当GLSurfaceView几何图形改变时，系统调用此方法。这包括GLSurfaceView尺寸大小的改变，设备横竖屏幕切换时候的改变。系统调用此方法来响应GLSurfaceView 容器的变化。

OpenGL包

一旦你使用了GLSurfaceView和GLSurfaceView.Renderer建立起来容器，你可以开始通过以下类来调用OpenGL API：

- OpenGL ES 1.0/1.1 API 包
 - android.opengl – 这个包为OpenGL ES1.0/1.1 的类提供了静态接口，并且性能要优于javax.microedition.khronos包的接口。
 - GLES10Ext
 - GLES10Ext
 - GLES11
 - GLES10Ext
 - javax.microedition.khronos.opengles –这个包提供了OpenGL ES 1.0/1.1 的标准实现方法。
 - GL10Ext
 - GL10Ext
 - GL11
 - GL11Ext
 - GL11ExtensionPack
- OpenGL ES 2.0 API 类
 - android.opengl.GLES20 –这个包提供了对于OpenGL ES 2.0的支持，并且从android 2.2 (API Level 8) 开始使用。如果你想马上就开发支持OpenGL的应用，那么请看Displaying Graphics with OpenGL ES类。

OpenGL需求声明

如果你的应用使用了OpenGL，但是并不能被所有的设备所支持的话，那么你需要在你的 `AndroidManifest.xml` 配置文件中添加以下的需求生命。以下是一些

最常用的关于OpenGL的配置声明：

- OpenGL ES 版本需求 –如果你的应用只支持OpenGL ES 2.0， 那你需要在manifest.xml配置文件中添加以下声明需求


```
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```
- 添加此声明后， Google市场会限制不支持OpenGL ES 2.0的设备无法安装此应用。
- 纹理压缩需求–如果你的应用使用了纹理压缩格式， 你需要在manifest.xml用<supports-gl-texture>对所使用的格式进行声明。更多有关纹理压缩格式的信息请访问Texture compression support。

在manifest.xml配置文件中声明纹理压缩需求后，如果用户设备对其中任何一项压缩类型不支持的话，此用户将无法在Google商店中看到此应用。更多关于Google商店如何过滤纹理压缩的信息，请访问Google Play and texture compression filtering 中的<supports-gl-texture> documentation部分

为绘制对象映射坐标

在Android设备中显示图形有一个很基本的问题是不同的设备的屏幕尺寸形状可能会各不相同。OpenGL 假设出一个正方形，有统一的坐标系，并且在默认情况下，也愿意在把那些坐标像画在正方形屏幕上那样画在你的非正方形屏幕上。  图1.默认情况下OpenGL坐标系（左图），映射到本地的Android设备屏幕上（右图） 上图展示了统一的坐标系在系统假设的OpenGL框架下情况，如左图，和此坐标系实际映射到本地设备屏幕后横屏下的表现情况，如右图。若要解决这个问题，可以通过设置OpenGL的投影模式和镜头视角来转换坐标，这样你的图形对象在任何情况下显示都会有正确的坐标系。要想设置投影和镜头视角，可以创建投影矩阵和镜头视角矩阵，并且设置到OpenGL的渲染管线。投影矩阵重新计算图型的坐标，然后映射正确的坐标到Android设备屏幕上。镜头视角矩阵创建一个变换来从特定的视角位置渲染对象。

OpenGL ES 1.0 中的投影和镜头视角

1. 在ES 1.0 API中，你可以通过分别创建每个矩阵并且把他们添加到OpenGL环

境中来设置投影和镜头视角，投影矩阵-使用设备屏幕的几何体来创建投影矩阵是为了重新计算对象坐标，使他们可以用正确的坐标系来进行绘制。下边的代码范例说明了如何修改[GLSurfaceView.Renderer](#)中的[onSurfaceChanged\(\)](#)方法来创建基于屏幕比例的投影矩阵，并且应用到OpenGL渲染环境中。

```

2.     public void onSurfaceChanged(GL10 gl, int width, int height) {
3.         gl.glViewport(0, 0, width, height);
4.         // make adjustments for screen ratio
5.         // 修改调整屏幕比例
6.         float ratio = (float) width / height;
7.         gl.glMatrixMode(GL10.GL_PROJECTION); // set matrix to
projection mode
8.         // 设置投影模式
9.         gl.glLoadIdentity(); // reset the matrix to its default state
10.        // 重置默认矩阵
11.        gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7); // apply the
projection matrix
12.        // 应用投影矩阵
13.    }
14. }
```

2. 镜头变换矩阵-一旦你已经使用投影矩阵调整了坐标系，你需要应用镜头视角。以下的代码范例展示了如何修改[GLSurfaceView.Renderer](#)中的[onDrawFrame\(\)](#)实现方法来应用视角模型和使用[GLU.gluLookAt\(\)](#)来创建可以模拟视角位置的镜头变换。

```

18.     public void onDrawFrame(GL10 gl) {
19.         ...
20.         // Set GL_MODELVIEW transformation mode
21.         // 设置变换模式为GL_MODELVIEW
22.         gl.glMatrixMode(GL10.GL_MODELVIEW);
23.         gl.glLoadIdentity();
24.         // reset the matrix to its default state
25.         // 重置默认矩阵
26.         ...
27.         ...
28.         // When using GL_MODELVIEW, you must set the camera view
29.         // 使用GL_MODELVIEW时，需要设置镜头视角
30.         GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 1.0f, 0.0f);
31.         ...
32.     }
33. }
```

OpenGL ES 2.0和3.0中的投影和镜头视角

1. 在ES 2.0 API中，你首先需要给图形对象添加一个矩阵到定点着色器，然后才能应用投影和镜头视角。矩阵添加后，你可以给你的图形对象生成并且应用投影和镜头视角矩阵。 1. 添加矩阵到定点着色器-为投影矩阵创建一个变量并且将它视为着色器位置的乘积来包含。下边的定点着色器的例子中，被包含的MVPMatrix 成员允许你给使用这个着色器的对象坐标设置投影和镜头视角矩阵。

```

2.      private final String vertexShaderCode =
3.      // This matrix member variable provides a hook to manipulate
4.      // the coordinates of objects that use this vertex shader
5.      // 这个矩阵成员变量提供了钩子操作使用此顶点着色器的对象坐标
6.
7.      "uniform mat4 uMVPMatrix;    \n" +
8.      "attribute vec4 vPosition;   \n" +
9.      "void main(){               \n" +
10.     "    // the matrix must be included as part of gl_Position
11.     // 此矩阵必须要做为gl_Position的一部分被包含进来
12.     "    gl_Position = uMVPMatrix * vPosition; \n" +
13.     "}" \n;

```

2. 访问着色器矩阵-在你的顶点着色器中创建钩子来应用投影和镜头视角后，你可以有权访问此变量来应用投影及镜头视角矩阵。下边的代码演示了如何修改中的实现[GLSurfaceView.Renderer](#)中的[onSurfaceCreated\(\)](#) 方法来访问之前定义过的顶点着色器的矩阵变量

```

18.      public void onSurfaceCreated(GL10 unused, EGLConfig config) {
19.      ...
20.      muMVPMatrixHandle =
GLES20.glGetUniformLocation(mProgram, "uMVPMatrix");
21.      ...
}

```

3. 创建投影和镜头视角矩阵-生成投影和镜头视角矩阵以供图形对象应用。以下的代码范例展示了如何实现[\[1\]](#) 中的[onSurfaceCreated\(\)](#)和[onSurfaceChanged\(\)](#) 方法来创建基于设备屏幕比例的镜头视角矩阵和投影矩阵

```

23.      public void onSurfaceCreated(GL10 unused, EGLConfig config) {
24.      ...
25.      // Create a camera view matrix
26.      // 创建镜头视角矩阵
27.      Matrix.setLookAtM(mVMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
28.      }
29.

```

```

30.     public void onSurfaceChanged(GL10 unused, int width, int height) {
31.         GLES20.glViewport(0, 0, width, height);
32.         float ratio = (float) width / height;
33.         // create a projection matrix from device screen geometry
34.         // 根据设备几何体创建投影矩阵
35.         Matrix.frustumM(mProjMatrix, 0, -ratio, ratio, -1, 1, 3, 7);
36.     }
37.
38. 
```

4. 应用投影与镜头视角矩阵-要想应用投影与镜头视角变换，矩阵相乘然后设置到顶点着色器上。以下代码范例演示了如何实现[GLSurfaceView.Renderer](#) 中的[\[2\]](#)方法，通过OpenGL，结合上述的投影矩阵与镜头视角代码来应用到对图形对象的渲染中。

```

40.     public void onDrawFrame(GL10 unused) {
41.         ...
42.         // Combine the projection and camera view matrices
43.         // 结合投影与镜头视角矩阵。
44.         Matrix.multiplyMM(mMVPMatrix, 0, mProjMatrix, 0, mMMatrix, 0);
45.         ...
46.         // Apply the combined projection and camera view
47.         // transformations
48.         // 应用投影与镜头视角的结合。
49.         GLES20.glUniformMatrix4fv(muMVPMatrixHandle, 1, false, mMVPMatrix, 0);
50.         ...
51.         // Draw objects
52.         // 绘制对象
53.     }
54. 
```

有关如何在OpenGL2.0中应用投影与镜头视角完整的内容，请访问[Displaying Graphics with OpenGL ES](#)

形状的朝向与卷绕

在OpenGL中，形状的朝向是通过3个或者多个点在三维空间里决定的。一组三个或者多个三维点（在OpenGL里叫顶点）有正面和背面。如何得知哪面是正面哪面是背面呢？这个问题很好，答案需要通过卷绕这个概念来解答，或者，由定义形状点的方向来找到答案。 图1.演示了变换成为逆时针顺序绘制的坐标组 在上例中，三角形的顶点是按照逆时针方向来绘制的。这个绘制坐标的顺序定义了这个形状的卷绕方向。默认情况下，在OpenGL中，朝逆时针绘制图形的方向就是正面。图1中的三角形所展示的面就是形状的正面了（像OpenGL解释中的一样），另一面就是背面。为什么知道哪面是正面这个问题很重要呢？这关系到一个OpenGL中很常用的功能，叫做Culling(剔除)。

面剔除是OpenGL环境中确定是否要渲染管线还是忽略（不计算或绘制）形状体背面，节省时间，内存和处理周期。

```
// enable face culling feature
// 允许面剔除功能

gl.glEnable(GL10.GL_CULL_FACE);
// specify which faces to not draw
// 指定不需要绘制的面

gl.glCullFace(GL10.GL_BACK);
```

如果你在尚未确定哪面是你形状体的正面和背面的情况下尝试使用面剔除功能，你的OpenGL图形看起来会有点窄小，或者可能根本就显示不出来。所以，通常情况下，定义你的OpenGL形状按照逆时针方向来绘制坐标系。

注意：设置OpenGL环境来按顺时针方向绘制正面也是可行的，但是这么做的话，要加入更多代码，而且可能当你向会有OpenGL开发经验的人寻求帮助时候令他感觉混淆不清。所以不推荐这么做。

OpenGL 版本与设备兼容性

penGL ES 1.0和1.1 API规范从Android 1.0就开始被支持了。从Android2.2版本开始，也就是API level 8开始，框架开始支持OpenGL ES 2.0 API 规范。大部分设备都支持OpenGL ES 2.0，并且这也是开发OpenGL应用所推荐使用的。更多有关支持OpenGL ES的设备信息请访问[OpenGL ES Versions Dashboard](#)。

OpenGL ES 1.0/1.1 API图形编程与使用2.0及更高版本编程完全不同。API1.x版本有更多便利方法和固定图形管线，而OpenGL ES 2.0和3.0 API通过OpenGL着色器提供了更直接的控制管线。要谨慎考虑图形需求并选择最适合应用的API版本。欲了解更多信息，请参阅选择OpenGL API版本。

OpenGL ES 3.0 API提供了额外功能和更好的性能，并可向下兼容。这意味着您可以针对OpenGL ES 2.0编写应用程序，并可包含OpenGL ES 3.0图形功能。欲了解查看3.0 API可用性的更多信息，请参阅检查OpenGL ES版本。

纹理压缩支持

纹理压缩能够通过减少内存占用和提高对内容使用效率来很明显的提

高OpenGL应用的表现性能。Android框架采用ETC1压缩格式为标准格式，包括[ETC1Util](#) 工具类和etc1tool压缩工具（位于Android SDK< sdk >/tools/）。例如，一个Android应用使用纹理压缩，请见代码范例[CompressedTextureActivity](#)。大部分的Android设备都支持ETC格式，但是并不能保证一定能正常使用。检查是否设备可以支持ETC1格式，可以通过调用[ETC1Util.isETC1Supported\(\)](#)方法。

注意：ETC1纹理压缩格式不支持带有alpha通道的纹理。如果你的应用需要带有alpha通道的纹理，你可以采用其它你的设备所支持的压缩格式。

使用OpenGL ES 3.0 API时ETC2/EAC文本压缩格式保证可用。这种文本格式提供了视觉效果高且质量良好的压缩比，同时还支持透明度。

除ETC1格式之外，Android设备还支持很多其它基于GPU芯片和OpenGL实现有关的纹理压缩格式。你可以先查看目标设备上都支持哪些纹理压缩，然后在决定使用那种纹理压缩类型。要想监测出设备都支持哪些纹理压缩格式，你可以查询并且检查一下OpenGL的扩展名，扩展名中标识了设备所支持的纹理压缩格式（还有其它所支持的功能）。一些常用的纹理压缩格式有如下几种：

- ATITC (ATC) - ATI 纹理压缩 (ATITC或者ATC) 被大多设备所支持，支持固定频率的RGB纹理压缩，无论有无alpha通道。这个格式可能被好几种OpenGL扩展名所标识，例如：
 - GL_AMD_compressed_ATC_texture
 - GL_ATI_texture_compression_atitc
- PVRTC-PowerVR纹理压缩 (PVRTC) 也是一种被广泛支持的压缩格式，它支持2-bit和4-bit像素纹理压缩，有无alpha通道均可。这个格式在OpenGL扩展名中的标识为：
 - GL_IMG_texture_compression_pvrtc
- S3TC (DXTn/DXTc) - S3纹理压缩 (S3TC) 有很多种不同的格式 (DXT1到DXT5)，还尚未达到被广泛支持的程度。这种压缩格式支持RGB纹理和4-bit alpha 或者8-bit alpha通道。在OpenGL扩展名中的标识方法为：
 - GL_OES_texture_compression_S3TC
 - GL_EXT_texture_compression_s3tc
 - GL_EXT_texture_compression_dxt1
 - GL_EXT_texture_compression_dxt3

- GL_EXT_texture_compression_dxt5
- GL_EXT_texture_compression_dxt5
- 3DC -3DC-3DC文本压缩（3DC）格式使用范围较小，它支持带有alpha通道的RGB文本。这种格式用下列OpenGL扩展名来表示：
 - GL_AMD_compressed_3DC_texture

警告：以上这些压缩格式并不是被所有设备支持的。厂商与设备对以上格式的支持可能会有变化。更多有关如何判断特定设备支持哪些纹理压缩格式，请看下面介绍。

注意：一旦你决定了使用那一种纹理压缩格式，需要在**manifest.xml**配置文件中通过<supports-gl-texture>进行声明。这样声明后，Google商店会过滤掉不支持此格式的设备。

判断OpenGL扩展名

OpenGL的实现会根据OpenGL ES API支持的扩展来变化。这些扩展包含文本压缩，也包括其它OpenGL功能的扩展。

要确定特定设备支持哪些结构压缩格式和其他OpenGL扩展：

1. 在目标设备上运行以下代码来判断哪些纹理压缩格式会被支持：

```
String extensions
=javax.microedition.khronos.opengles.GL10.glGetString(GL10.GL_EXTENSIONS
;
```

警告：结果会根据设备不同而不同！你必须运行这个指令在不同的设备上来查看哪些纹理压缩类是普遍被支持的。

检测OpenGL ES版本

安卓设备拥有几种可用的OpenGL ES版本。可以指定应用程序所需API最低版本，但同时会想要运用较新API的功能。例如，OpenGL ES 3.0 API运用了2.0版本的向后兼容，因此可以编写自己的应用程序以便使用OpenGL ES 3.0功能，但如果3.0API不可用就会回落到2.0API。

在使用比最低版本高的OpenGL ES功能前，应用程序要检查设备的可用API版本。可以通过以下两种方式实现：

1. 尝试创建更高级别的OpenGL ES内容并检查结果。
2. 创建最小支持OpenGL ES的结构并检查版本值。

下列代码显示了如何通过创建anEGLContext及检查结果来检测可用OpenGL ES版本。此示例显示了OpenGL ES 3.0版本如何进行检测：

```
private static double glVersion = 3.0;

private static class ContextFactory implements
GLSurfaceView.EGLContextFactory {

    private static int EGL_CONTEXT_CLIENT_VERSION = 0x3098;

    public EGLContext createContext(
        EGL10 egl, EGLDisplay display, EGLConfig eglConfig) {
        Log.w(TAG, "creating OpenGL ES " + glVersion + " context");
        int[] attrib_list = {EGL_CONTEXT_CLIENT_VERSION, (int) glVersion,
            EGL10.EGL_NONE};
        // attempt to create a OpenGL ES 3.0 context
        EGLContext context = egl.eglCreateContext(
            display, eglConfig, EGL10.EGL_NO_CONTEXT, attrib_list);
        return context; // returns null if 3.0 is not supported;
    }
}
```

如果 createContext()方法显示返回null，那么代码要创建OpenGL ES 2.0结构并且仅使用这一API。

下列代码显示了通过创建最小支持结构并检查版本字符串如何来检查OpenGL ES版本：

```
// Create a minimum supported OpenGL ES context, then check:
String version = javax.microedition.khronos.opengles.GL10.glGetString(
    GL10.GL_VERSION);
Log.w(TAG, "Version: " + version );
// The version format is displayed as: "OpenGL ES <major>.<minor>"
```

通过这种方式，如果发现设备支持更高级别的**API**版本，就要销毁最小的OpenGL ES内容并创建更高可用**API**版本的新内容。

选择**OpenGL API** 的版本

OpenGL ES API 1.0版本（包括1.1扩展版本）和2.0版本都提供高性能图形接口来创建3D游戏，视觉效果和用户接口。OpenGL ES 1.0/1.1 API和ES2.0图形编程方面区别很大，所以开发者在开始开发前需要很小心谨慎的考虑以下因素：

- 性能—一般来说，OpenGL ES 2.0提供了比ES 1.0/1.1更快的图形性能。然而，性能很大程度上取决于应用运行在什么设备上，这取决与**OpenGL**图形管线的实现方法不同。
- 设备兼容性—开发者要考虑设备类型，Android版本和**OpenGL ES**版本适用于哪些用户。关于更多跨平台设备兼容的问题，请访问[OpenGL Versions and Device Compatibility](#)部分。
- 编程方便性—OpenGL ES 1.0/1.1 API 提供固定的功能管线和方便的功能，这是**OpenGL ES** 2.0 API所不支持的。对于新入门的**OpenGL**开发者来说，基于**OpenGL ES** 1.0/1.1会更加容易方便。
- 图形控制—OpenGL ES 2.0 API提供了通过高层次的控制，可全部程序化管线，对图形处理操作提供了更多直接操作方法。开发者可以创建出1.0/1.1 API很难做出的效果。

性能，兼容性，方便性，控制性和一些其它因素可能影响你的决定，你要基于如何提供给用户更好的体验来选择一个**OpenGL API**版本。

来自“[index.php?title=OpenGL&oldid=13881](#)”

Hardware Acceleration

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文：[Hardware Acceleration](#)

翻译： D.Winter

目录

[[隐藏](#)]

[1 Hardware Acceleration](#)

- [1.1 控制硬件加速](#)
 - [1.1.1 Application 层](#)
 - [1.1.2 Activity 层](#)
 - [1.1.3 Window 层](#)
 - [1.1.4 View 层](#)
- [1.2 判断View是否已经硬件加速](#)
- [1.3 Android绘画模型](#)
 - [1.3.1 基于软件的绘画模型](#)
 - [1.3.2 基于硬件的绘画模型](#)
- [1.4 不支持的绘画操作](#)
- [1.5 View 层](#)
 - [1.5.1 View层和动画](#)
- [1.6 要领和技巧](#)

Hardware Acceleration

从Android 3.0 (API level 11)开始，Android 2D渲染管线能更好的支持硬件加速。硬件加速通过GPU执行各种绘画操作。因为硬件加速需要消耗更多的

资源，所以你的App需要更多的RAM。

开启硬件加速最简单的方法是在整个应用全局设置。如果应用只使用标准的[View](#)和[Drawable](#)，全局设置不会产生不利的影响。然而，因为硬件加速不支持所有的2D绘制操作，开启会影响一些自定义View或者绘制调用。问题显示为不可见的元素、异常，或者错误渲染像素。为了补救这些，Android提供给你选项开启或者关闭硬件加速在以下几个层面：

- Application
- Activity
- Window
- View

如果你的应用执行自定义绘制，开启硬件加速，测试应用在实际的硬件设备上去查找问题。

控制硬件加速

你可以在以下层面控制硬件加速

- Application
- Activity
- Window
- View

Application 层

在你的Android manifest文件里，添加以下属性到[`<application>`](#)标签里，针对整个应用开启硬件加速：

```
<application android:hardwareAccelerated="true" ...>
```

Activity 层

如果整个应用开启硬件加速表现的不稳定，你也可以针对单个Activity进行控制。在Activity层开启或者关闭硬件加速，你可以使用[android:hardwareAccelerated](#)属性在[`<activity>`](#)标签内。下面是单个Activity中关闭硬件加速的例子：

```
<application android:hardwareAccelerated="true">
    <activity ... />
    <activity android:hardwareAccelerated="false" />
</application>
```

Window 层

如果你需要更细致的控制，可以在获得的**Window**里开启硬件加速：

```
getWindow().setFlags(
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
```

- 注：在**window**层不能取消硬件加速。

View 层

通过下面代码，你可以在运行时关闭硬件加速：

```
myView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

注：你不能在**view**层开启硬件加速。**View**面板有其他方法关闭硬件加速。更多详细内容请看[View面板](#)。

判断**View**是否已经硬件加速

有时候，知道一个应用当前是否硬件加速是非常有用的，尤其像自定义**View**。这非常有用，当你的应用做大量的自定义绘制并且不是所有操作都支持新的渲染管道。

有两个方法查看应用是否硬件加速：

- [View.isHardwareAccelerated\(\)](#) 如果**View**附属于硬件加速的窗体，将会返回**true**。
- [Canvas.isHardwareAccelerated\(\)](#) 如果**Canvas**已硬件加速，将会返

回true。

如果你一定要在绘制代码中进行这个查看，请尽可能用[Canvas.isHardwareAccelerated\(\)](#)代替[View.isHardwareAccelerated\(\)](#)。当一个View附属于硬件加速窗体，它仍可以通过非硬件加速的Canvas来绘制。这种情况，在实例化时，绘制一个View到bitmap中，为了缓存目的。

Android绘画模型

当开启硬件加速，新的绘画模型利用显示列表在界面上渲染你的应用。为了完全理解显示列表和对你的应用有怎样的影响，知道Android不通过硬件加速怎么样绘制View也很重要。下面的篇章阐述基于软件和基于硬件的绘画模型。

基于软件的绘画模型

在软件绘画模型中，View通过以下两个步骤被绘制：

- 使层次失效
- 绘制层次

每当应用需要更新部分UI时，调用[invalidate\(\)](#)(或者它的变形)在任意需要改变内容的view里。这些失效信息被传播始终在view层，计算需要重绘的界面区域。然后，Android系统绘制任意view在这些区域。不幸的是，这种模型有两个缺点：

- 第一，这种模型需要执行的大量的代码在每一次绘画。比如，如果你的应用在按钮里调用了[invalidate\(\)](#)，而且这个按钮在另外一个view之上，此时Android系统会重绘这个view，即时它没有发生改变。
- 第二个问题是绘画模型会隐藏你应用的错误。从Android系统开始重绘view，当他们与脏区域融合时，你改变的view的内容可能会被重绘即时[invalidate\(\)](#)没有被调用。此时，你依赖于另外一个view使之失效来达到合适的反应。这个反应会在任何时候改变在你修改你的应用时。因此，你需要一直调用[invalidate\(\)](#)在你的自定义view上，无论你修改数据或者影响你view绘画代码的声明。

注: Android view当它的属性改变时, 自动调用[invalidate\(\)](#), 比如背景颜色或者文本框内容。

基于硬件的绘画模型

Android仍使用[invalidate\(\)](#)和[draw\(\)](#)来响应界面更新和视图渲染, 不同的是控制当前绘制。Android系统将他们记录在显示列表来代替马上执行绘画命令, 它包含view层绘画代码的输出。另外一个优化是Android系统只需对记录和更新显示列表, 通过调用 [invalidate\(\)](#)将view标记为。还没失效的View通过重新运行之前的记录显示列表被重绘。这个新绘画模型包括三个阶段:

- 1、层失效
- 2、记录更新显示列表
- 3、绘画显示列表

用此模式, 你不能依赖与脏区域交叉的view来执行它的[draw\(\)](#)。为了确保Android系统记录一个View的显示列表, 你必须调用[invalidate\(\)](#)。忘记这么做会导致视图看起来一样, 甚至在改变它后。一旦发生, 这个BUG很容易被发现。

使用显示列表也利于动画增强, 因为设置特殊的属性, 像透明度和旋转, 不需要使目标视图失效(它自动完成)。这个优化也适用于显示列表的视图(你应用的任一视图都硬件加速)比如, 假设有个[LinearLayout](#)在[Button](#)上有个[ListView](#)。[LinearLayout](#)的显示列表看上去这样:

`DrawDisplayList(ListView)`

`DrawDisplayList(Button)`

假设你现在要改变ListView的不透明, 在调用`setAlpha(0.5f)`后, 显示列表变成:

`SaveLayerAlpha(0.5)`

`DrawDisplayList(ListView)`

`Restore`

`DrawDisplayList(Button)`

[ListView](#)的设置代码没有被执行。系统只更新显示列表中更简单的[LinearLayout](#)。

在未开启硬件加速的应用中,列表的绘制代码在其父亲中还会被执行一次。

不支持的绘画操作

当启动硬件加速, 2D渲染通道支持一般使用的画布绘画操作和一些较少使用的操作。所有的绘画操作被用于渲染程序, 默认为widget和layout, 还有些高级应用比如反光和纹理平铺也是被支持的。下面是不被硬件加速的操作清单:

- Canvas
 - [clipPath\(\)](#)
 - [clipRegion\(\)](#)
 - [drawPicture\(\)](#)
 - [\[int, int, android.graphics.Path, float, float, android.graphics.Paint\) drawTextOnPath\(\) \]](#)
 - [\[int, float\[\], int, float\[\], int, int\[\], int, short\[\], int, int, android.graphics.Paint\) drawVertices\(\) \]](#)
- Paint
 - [setLinearText\(\)](#)
 - [setMaskFilter\(\)](#)
 - [setRasterizer\(\)](#)
- Xfermodes
 - [AvoidXfermode](#)
 - [PixelXorXfermode](#)

另外, 有些操作在硬件加速开启后会发生变化:

- Canvas
 - [clipRect\(\)](#) : XOR, Difference and ReverseDifference clip modes are ignored. 3D transforms do not apply to the clip rectangle

[int, int, float\[\], int, int\[\], int, android.graphics.Paint\) drawBitmapMesh\(\) \] :](#) colors array is ignored

- Paint
 - [setDither\(\)](#) : ignored
 - [setFilterBitmap\(\)](#) : filtering is always on

- [float, float, int\) setShadowLayer\(\)](#) : works with text only

- PorterDuffXfermode

- [PorterDuff.Mode.DARKEN](#) will be equivalent to [SRC_OVER](#) when blending against the framebuffer.
- [PorterDuff.Mode.LIGHTEN](#) will be equivalent to [SRC_OVER](#) when blending against the framebuffer.
- [PorterDuff.Mode.OVERLAY](#) will be equivalent to [SRC_OVER](#) when blending against the framebuffer.

- ComposeShader

- [ComposeShader](#) can only contain shaders of different types (a [BitmapShader](#) and a [LinearGradient](#) for instance, but not two instances of [BitmapShader](#))
- [ComposeShader](#) cannot contain a [ComposeShader](#)

如果你的应用受缺失属性和限制的影响,你可以关闭硬件加速,在你受影响的部分调用[android.graphics.Paint](#))

[setLayerType\(View.LAYER_TYPE_SOFTWARE, null\)](#)。这个方法,你仍可以利用硬件加速其他任一地方。查看[Controlling Hardware Acceleration](#)得到关于硬件加速的更多信息。

View 层

在Android的不同版本, view已经有能力渲染进入屏幕缓存区内,无论是view的绘制缓存,还是使用[android.graphics.Paint, int\)](#)

[Canvas.saveLayer\(\)](#)。屏幕缓存,或层,有多种用途。你可以使用他们获得更好的性能,当动画组合视图或者需要应用复合效应。比如,你可以使用[Canvas.saveLayer\(\)](#)来实现消退效果来临时渲染一个view进入层,然后使用opacity factor合成到界面。

Android 3.0 (API level 11)开始,你有更多的控制通过[android.graphics.Paint](#)) [View.setLayerType\(\)](#)方法,怎样或者什么时候使用layers。这个API有2个参数:layer的类型和可选的[Paint](#)对象,它阐明layer怎么样合成。你可以使用[Paint](#)参数使用滤色镜,特殊的混合模式,或者设置为不透明。view可以使用三种layer类型的其中一种:

- [LAYER_TYPE_NONE](#):被一般渲染并且不会被进入屏幕缓存。这是默认行为。

- [LAYER_TYPE_HARDWARE](#): 由硬件渲染到硬件纹理，如果应用开启硬件加速。如果未开启，就同[LAYER_TYPE_SOFTWARE](#)。
- [LAYER_TYPE_SOFTWARE](#): 由软件渲染到位图。

The type of layer you use depends on your goal:

- **Performance**: 由硬件渲染到硬件纹理，一旦View被渲染到layer，它的绘图代码不会被执行直到调用[invalidate\(\)](#)。有些动画，像透明度动画，直接放入layer，由GPU完成非常有效率。
- **Visual effects**: 使用硬件或者软件layer类型和[Paint](#)，对view应用特殊的视觉处理。比如，你使用[ColorMatrixColorFilter](#)绘制一个view为黑色或者白色。
- **Compatibility**: 使用软件layer类型促使view软件渲染。如果硬件加速的，有着渲染问题，这是一个简单的方法来绕过限制的硬件渲染管道。

View层和动画

开启硬件加速，硬件层提供更快的和更平滑的动画效果。当有很多绘图操作时，动画每秒60帧不是一直能保持的。硬件层可以减轻这个，通过渲染为硬件纹理。硬件纹理可以优化View，不再需要视图不断重绘本身。当你调用[invalidate\(\)](#)或者改变view属性时，view才会重绘。如果动画显示的不够平滑，考虑开启硬件层在你使用的View。

当view进入后台硬件层，层被混合到界面，view的属性被控制。设置这些属性很有效果，因为他们不需要view失效或者重绘。以下属性作用于混合层。通过setter测试属性，获得最优效果：

- alpha: 改变层透明度
- x, y, translationX, translationY: 改变层位置
- scaleX, scaleY: 改变层大小
- rotation, rotationX, rotationY: 改变层的三维定位
- pivotX, pivotY: 改变层的转换源

当view被作为[ObjectAnimator](#)启用时，这些属性被使用其他名字。如果你想使用这些属性，请调用适当的setter或者getter。比如，为了改变alpha属性，调用[setAlpha \(\)](#)。下面的代码片段展示了最有效的方法旋

转viewiew在3D的Y轴上:

```
view.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator.ofFloat(view, "rotationY", 180).start();
```

因为硬件层消耗媒体资源,强烈介意你只在持续动画时开启, 并且在动画结束时关闭。你可以使用animation listeners完成:

```
View.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "rotationY",
180);
animator.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        view.setLayerType(View.LAYER_TYPE_NONE, null);
    }
});
animator.start();
```

更多信息, 请看[Property Animation](#)。

要领和技巧

开启硬件加速2D图像可以立即提高性能, 但你仍可以设计你的应用使用GPU性能, 以下推荐:

减少应用中视图的数量

视图越多, 速度越慢. 请求软件渲染通道也越慢。减少视图是优化UI的最简单的办法。

避免透支

不要在顶层绘制太多。去除那些被完全遮挡住的图层。如果你要绘制相互叠加的多个图层, 考虑将他们合并为一个图层。

不要在**draw**方法中创建**render**对象

一个通常的错误是创一个[Paint](#)对象或者[Path](#)对象, 在任何时候**rendering**方法被调用。这使得垃圾收集器运行的更频繁, 忽视缓存, 在硬件通道优化。

不要太频繁的改变图形

图形实例化时, 由**texture masks**被渲染。任何时候你改变路径, 硬件通道创

建一个新的mask,这是非常耗资源的。

不要太频繁的改变位图

任何时候你改变一个位图的内容,它作为GPU纹理重新被上传在下一次绘制时。

使用透明度要小心

当你通过[setAlpha\(\)](#)来改变透明度时, [AlphaAnimation](#), 或者[ObjectAnimator](#), 在off-screen buffer被渲染, 它两倍于使用fill-rate。当应用alpha在非常大的界面上时, 考虑设置视图属性LAYER_TYPE_HARDWARE.

来自 "[index.php?title=Hardware_Acceleration&oldid=9051](#)"



Computation

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： 夔娘

原文链

接：<http://docs.eoeandroid.com/guide/topics/connectivity/index.html>

计算

Renderscript提供了一个平台独立的计算引擎，在母语水平。用它来加速您的应用程序，需要大量的计算能力。

博客文章

在**Renderscript**级别

集成电路，Renderscript（RS）已更新了一些新的特点，以简化计算加速加入到您的应用程序。RS是有趣的计算加速，当你有大量的数据上，你需要做大量的处理缓冲区。在这个例子中，我们将着眼于应用位图的水平/饱和操作。

Renderscript部分2

我在介绍Renderscript的简要介绍了这项技术。在这篇文章中，我会更详细地看看“计算”。我们在Renderscript使用“计算”意味着数据处理的Dalvik代码Renderscript代码可能运行相同或不同的处理器（S）卸载。

来自“[index.php?title=Computation&oldid=9059](#)”



RenderScript

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

RenderScript 是高效运行计算密集型任务的框架。 RenderScript 主要用于数据并行计算，尽管串行计算密集型工作负载同样有效。 RenderScript 运行时会跨越设备上所有处理器并行处理工作，比如多核 CPU, GPU, 或 DSP，让您专注于算法而不是安排工作或负载平衡。 RenderScript 适用于图像处理，计算摄影或者计算机视觉。

研究RenderScript之前，有两个主要概念应该明白：

- C99衍生语言中写入了高性能计算内核。
- Java API 用于管理RenderScript资源的生命周期及控制内核执行。

目录

[\[隐藏\]](#)

[1 编写RenderScript内核](#)

- [1.1 设置浮点精度](#)

[2 访问RenderScript API](#)

[3 使用RenderScript支持库API](#)

[4 Using RenderScript from Java Code](#)

编写RenderScript内核

RenderScript内核通常位于 <project_root>/src/ 目录中的.rs文件；每个 .rs 文件称为一个脚本。每个脚本包含自己的一套内核，函数和变量。脚本会包含：

- Pragma声明 (`#pragma version(1)`)。这声明了RenderScript内核语

言的版本。目前1是唯一有效值。

- Pragma声明(#pragma rs java_package_name(com.example.app))。这声明了脚本反映的Java类包名。
- 某些可调用函数。可调用函数是单线程的RenderScript功能，您可以从Java代码中调用任意参数。这些适用于在较大处理系统中进行初始设置或串行计算。
- 某些脚本全局变量。脚本全局变量相当于C中全局变量。可以从Java代码中访问脚本全局变量，而这些通常用于RenderScript内核参数传递。
- 某些运算内核。内核是跨越每一个元素的并行功能。

简单内核如下所示：

```
uchar4 __attribute__((kernel)) invert(uchar4 in, uint32_t x,
uint32_t y) {
    uchar4 out = in;
    out.r = 255 - in.r;
    out.g = 255 - in.g;
    out.b = 255 - in.b;
    return out;
}
```

在许多方面这等同于标准C函数。第一个显著特征是__attribute__((kernel))应用到函数原型。这表明该函数是RenderScript内核而不是可调用函数。另一个特征是参数和类型。在RenderScript 内核中有特殊的参数，可以自动根据传递给内核启动的输入分配来填充。默认情况下内核在整个分配中运行。第三个显著特征是内核返回类型。从内核返回的值会自动写入输出分配的对应位置。RenderScript运行时进行检查，以确认输入的元素类型和输出分配与内核原型相匹配。如果不匹配则会引发异常。

内核可能会有一个输入配置，一个输出配置或者两者兼有。内核可能不会有超过一个输入或者输出分配。如果必须有一个以上的输入或输出，这些对象要绑定rs_allocation脚本全局变量，并且通过rsGetElementAt_type()和rsSetElementAt_type()调用函数。

内核可以使用x,y,z参数访问当前执行的坐标。这些参数是可选的，但是坐标参数类型必须是uint32_t。

- 可选的**init()**函数。**init()**函数是可调用函数的特殊类型，当脚本第一次实例化时会运行此函数。这样脚本创建时会自动进行一些计算。
- 某些静态全局脚本和函数。静态全局脚本相当于全局性脚本，只是它不能在Java代码里设置。静态函数是可以从脚本任何内核或可调用函数调用的标准C函数。

设置浮点精度

你可以控制浮点精度所需水平。这一做法适用于不需要完整的IEEE754-2008标准（默认情况下使用）的情况。下列编译可以设置不同级别的浮点精确度。

- 1. **pragma rs_fp_full** (默认情况下如果不指定) : 针对IEEE754-2008标准概述中需要浮点精度的应用程序。
- 2. **pragma rs_fp_relaxed**针对不需要严格遵守IEEE754-2008并承受较低精确度的应用程序。
- 3. **pragma rs_fp_imprecise** 针对没有严格精度要求的应用程序。This mode enables everything in `rs_fp_relaxed` along with the following:
 - Operations resulting in -0.0 can return +0.0 instead.
 - Operations on INF and NAN are undefined.

大多数应用程序可以安全使用**rs_fp_relaxed**。

访问RenderScript API

当使用RenderScript开发安卓应用程序时，可以用两种方式之一来访问API：

- **android.renderscript** 这个类包的API适用于运行安卓3.0 (API级别11) 及更高版本的设备。这些是RenderScript原始API，当前不能进行更新。
- **android.support.v8.renderscript**-包里的API通过支持库可用，它允许在安卓2.2 (API级别8) 及更高版本的设备上运行。

API	RenderScript	API
-----	--------------	-----

极力建议您使用支持库 来访问 , 因为 包含RenderScript计算框架最新改进，同时提供了更广泛的设备兼容性。

使用RenderScript支持库API

为了使用RenderScript支持库API，必须配置开发环境以便能够访问它们。使用API需要下列安卓SDK工具：

- 安卓SDK工具版本22.2或者更高
- 安卓SDK Build-tools版本18.1.0或者更高版本

您可以检查和更新安卓SDK管理内这些工具的安装版本。

要使用Eclipse中的支持库 RenderScript API：

- 1、请确保拥有所需的安卓SDK版本和编译工具安装版本。
- 2、打开应用程序项目根文件夹中的**project.properties** 文件。
- 3、将下列语句添加到文件：

```
renderscript.target=18
renderscript.support.mode=true
sdk.buildtools=18.1.0
```

- 4、在使用RenderScript的应用程序类中，添加一个支持库类的进口。

```
import android.support.v8.renderscript.*;
```

- **Renderscript.target**-指定要生成的字节码版本。建议您把此值设置成最高API级别，同时把 **renderscript.support.mode**设置为true。这一设置的有效值是从11到最新发布API级别的任何整数值。如果应用程序清单指定 的最低SDK版本设置为较高的值，会忽略此值而把目标值设定为最低SDK版本。
- **Renderscript.support.mode** 指定了如果运行设备不支持目标版本，那生成的字节码应该返回兼容版本。
- **Sdk.buildtools** 可用的安卓SDK生成工具版本。此值应该设置为18.1.0或更高。如果未指定此选项，可以使用最高安装编译工具版本。

Using RenderScript from Java Code

根据 android.renderscript或者android.support.v8.renderscript中的API类可以使用RenderScript。大多数应用程序都遵循相同的基本使用模式：

- 1、初始化RenderScript内容。create(Context)创建的RenderScript方面，要确保 RenderScript可以使用并提供控制所有后续RenderScript对象的生命周期。应该考虑到内容创建是潜在长期运行的操作，因为可能在不同 硬件创建资源。
- 2、至少创建一个传递给脚本的配置。配置是一个提供固定量数据存储的RenderScript对象。

来自“[index.php?title=Renderscript&oldid=13882](#)”



Advanced Renderscript

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：
址：

<http://developer.android.com/guide/topics/renderScript/advanced.html>

翻译者： jcccn

更新时间： 2012.7.16

目录

[[隐藏](#)]

[1 渲染脚本进阶](#)

[2 渲染脚本运行时层](#)

◦ [2.1 反射层](#)

▪ [2.1.1 函数](#)

▪ [2.1.2 变量](#)

▪ [2.1.3 结构体](#)

▪ [2.1.4 指针](#)

◦ [2.2 内存分配API](#)

[3 处理内存](#)

◦ [3.1 分配和绑定动态内存到渲染脚本](#)

◦ [3.2 读写内存](#)

▪ [3.2.1 读写全局变量](#)

▪ [3.2.2 读写全局指针](#)

渲染脚本进阶

使用**RenderScript(渲染脚本)**的程序仍然在安卓虚拟机内部运行，因此除了在适当的时候使用渲染脚本之外，还可以使用你熟悉的框架api 来编写程序。为了实现**android**框架和渲染脚本运行时之间的交互，设计了一个中间层代码来帮助这两层之间通信和内存管理。此文档将对这两层代码以及**Android**虚拟机与渲染脚本运行时之间内存共享方式的细节进行分解。

渲染脚本运行时层

渲染脚本代码的编译和执行都在一个紧凑和优化的运行时层。运行时层的api提供了对密集计算（轻便性、根据可用处理器单元的数量自动伸缩）提供支持。

注：NDK中的标准C函数必须在CPU上运行，而渲染脚本被设计成在不同种类的处理器上运行，因此渲染脚本不能访问这些NDK的库。

渲染脚本的代码保存在.rs或者.rsh文件中，并放在Android工程的src/目录下面。这些代码会被作为Android构建一部分的 llvm编译器编译成中间字节码。当程序在设备上运行的时候，这些字节码会被驻留在设备上的另一个llvm编译器实时编译成机器码。机器码会被优化并且缓存起来，以便后续使用这些渲染脚本的程序不需要再重复编译中间码。

下面是渲染脚本运行时库的一些功能：

- 请求内存分配的功能
- 一个包含许多重载了各种方式的标量和矢量数序函数的大集合。比如加法、乘法、点积、叉积、原子算数、比较等
- 原始数据类型和向量的转换程序、矩阵运算程序、日期和时间程序
- 支持渲染脚本系统的数据类型和结构，比如用来定义二维、三维或者四维向量的**Vector**类型
- 日志功能

关于渲染脚本运行时库更多函数，请参考渲染脚本运行时API手册。

反射层

反射层由一系列类组成，这些类由Android构建工具生成用来实现从Android框架访问渲染脚本运行时的能力。该层也提供方法和构造器来分配和使用脚本代码中定义的指针指向的内存。反射的主要组件如下：

- 每一个创建的.rs文件被自动生成到一个名为project_root/gen/package/name/ScriptC_renderscript_filename的ScriptC类中。这个文件是.rs文件的.java版本，它能够被Android框架层的代码调用。这个类包含下列从.rs文件生成的项：
 - 非静态的函数
 - 非静态、全局的渲染脚本变量。每个变量都生成了相关的存取方法，以便从Android框架层来读写这些变量。如果一个全局变量是在渲染脚本运行时层初始化的，那么这些变量值被用来初始化Android框架层中的相应值。如果一个全局变量被标记成const的，这个变量就不会自动生成对应的set方法了。
 - 全局指针
- 一个结构体被反射到名为project_root/gen/package/name/ScriptField_struct_name的扩展自Script.FieldBase的类中。这个类表示了一组结构体，它们允许为一个或多个struct的实例分配内存。

函数

函数被反射到脚本类自身，位置

在project_root/gen/package/name/ScriptC_renderscript_filename。比如，如果你在脚本代码中声明了下面的函数：

```
void touch(float x, float y, float pressure, int id) {
    if (id >= 10) {
        return;
    }
    touchPos[id].x = x;
    touchPos[id].y = y;
    touchPressure[id] = pressure;
}
```

那么会生成如下代码：

```

public void invoke_touch( float x, float y, float pressure, int id ) {
    FieldPacker touch_fp = new FieldPacker( 16 );
    touch_fp.addF32( x );
    touch_fp.addF32( y );
    touch_fp.addF32( pressure );
    touch_fp.addI32( id );
    invoke( mExportFuncIdx_touch, touch_fp );
}

```

这些函数没有返回值，因为渲染脚本系统被设计成异步执行的。当你的Android框架代码调用到渲染脚本时，这些调用会被加入队列尽快执行。这种机制允许渲染脚本系统不用立即中断来执行函数，进而提高效率。如果函数有返回值，函数调用就会被阻塞直到返回值返回。如果你希望渲染脚本的代码把值传回Android框架，请使用[rsSendToClient\(\)](#)函数。

变量

支持类型的变量被反射到脚本类，位置

在`project_root/gen/package/name/ScriptC_renderscript_filename`。每个变量都生成了相应的存取方法。比如，如果你在脚本代码声明了下面的变量：

```
uint32_t unsignedInteger = 1;
```

那么会生成如下代码：

```

private long mExportVar_unsignedInteger;
public void set_unsignedInteger( long v ){
    mExportVar_unsignedInteger = v;
    setVar( mExportVarIdx_unsignedInteger, v );
}

public long get_unsignedInteger(){
    return mExportVar_unsignedInteger;
}

```

结构体

结构体被反射到它们自己单独的类中，保存

在`<project_root>/gen/com/example/renderscript`

`/ScriptField_struct_name`。该类表示了一组**struct**，允许你为指定数量的结构体分配内存。比如，如果你在脚本代码中声明了下面的结构：

```
typedef struct Point {
```

```

    float2 position;
    float size;
} Point_t;

```

那么会在`ScriptField_Point.java`文件中生成如下代码：

```

package com.example.android.rs.hellocompute;

import android.renderscript.*;
import android.content.res.Resources;

/**
 * @hide
 */
public class ScriptField_Point extends
android.renderscript.Script.FieldBase {

    static public class Item {
        public static final int sizeof = 12;

        Float2 position;
        float size;

        Item() {
            position = new Float2();
        }
    }

    private Item mItemArray[];
    private FieldPacker mIOBuffer;
    public static Element createElement(RenderScript rs) {
        Element.Builder eb = new Element.Builder(rs);
        eb.add(Element.F32_2(rs), "position");
        eb.add(Element.F32(rs), "size");
        return eb.create();
    }

    public ScriptField_Point(RenderScript rs, int count) {
        mItemArray = null;
        mIOBuffer = null;
        mElement = createElement(rs);
        init(rs, count);
    }

    public ScriptField_Point(RenderScript rs, int count, int
usages) {
        mItemArray = null;
        mIOBuffer = null;
        mElement = createElement(rs);
        init(rs, count, usages);
    }

    private void copyToArray(Item i, int index) {
        if (mIOBuffer == null) mIOBuffer = new
FieldPacker(Item.sizeof * getType().getX()/* count
*/);
        mIOBuffer.reset(index * Item.sizeof);
    }
}

```

```

        mIOBuffer.addF32(i.position);
        mIOBuffer.addF32(i.size);
    }

    public void set(Item i, int index, boolean copyNow) {
        if (mItemArray == null) mItemArray = new
Item[getType().getX() /* count */];
        mItemArray[index] = i;
        if (copyNow) {
            copyToArray(i, index);
            mAllocation.setFromFieldPacker(index, mIOBuffer);
        }
    }

    public Item get(int index) {
        if (mItemArray == null) return null;
        return mItemArray[index];
    }

    public void set_position(int index, Float2 v, boolean copyNow) {
        if (mIOBuffer == null) mIOBuffer = new
FieldPacker(Item.sizeof * getType().getX() /* count */);
        if (mItemArray == null) mItemArray = new
Item[getType().getX() /* count */];
        if (mItemArray[index] == null) mItemArray[index] = new
Item();
        mItemArray[index].position = v;
        if (copyNow) {
            mIOBuffer.reset(index * Item.sizeof);
            mIOBuffer.addF32(v);
            FieldPacker fp = new FieldPacker(8);
            fp.addF32(v);
            mAllocation.setFromFieldPacker(index, 0, fp);
        }
    }

    public void set_size(int index, float v, boolean copyNow) {
        if (mIOBuffer == null) mIOBuffer = new
FieldPacker(Item.sizeof * getType().getX() /* count */);
        if (mItemArray == null) mItemArray = new
Item[getType().getX() /* count */];
        if (mItemArray[index] == null) mItemArray[index] = new
Item();
        mItemArray[index].size = v;
        if (copyNow) {
            mIOBuffer.reset(index * Item.sizeof + 8);
            mIOBuffer.addF32(v);
            FieldPacker fp = new FieldPacker(4);
            fp.addF32(v);
            mAllocation.setFromFieldPacker(index, 1, fp);
        }
    }

    public Float2 get_position(int index) {
        if (mItemArray == null) return null;
        return mItemArray[index].position;
    }

    public float get_size(int index) {
        if (mItemArray == null) return 0;
        return mItemArray[index].size;
    }
}

```

```

    }

    public void copyAll() {
        for (int ct = 0; ct < mItemArray.length; ct++)
copyToArray(mItemArray[ct], ct);
        mAllocation.setFromFieldPacker(0, mIOBuffer);
    }

    public void resize(int newSize) {
        if (mItemArray != null) {
            int oldSize = mItemArray.length;
            int copySize = Math.min(oldSize, newSize);
            if (newSize == oldSize) return;
            Item ni[] = new Item[newSize];
            System.arraycopy(mItemArray, 0, ni, 0, copySize);
            mItemArray = ni;
        }
        mAllocation.resize(newSize);
        if (mIOBuffer != null) mIOBuffer = new
FieldPacker(Item.sizeof * getType().getX() /* count */);
    }
}

```

自动生成的代码让使用者能够方便地为来自渲染脚本运行时的内存分配请求分配内存，并与内存中的结构体交互。每一个结构体的类都定义了下列方法和构造器：

- 允许你分配内存的重载了的构造器。`ScriptField_struct_name(RenderScript rs, int count)`这个构造器中，允许你用`count`参数定义希望分配内存的结构体的数量。`ScriptField_struct_name(RenderScript rs, int count, int usages)`这个构造器中，`usages`参数用来指定内存分配的内存空间。可以有下列四种内存空间：
 - **USAGE_SCRIPT**: 在脚本内存空间分配。如果不指定内存空间，该方式为默认。
 - **USAGE_GRAPHICS_TEXTURE**: 在GPU的纹理内存空间分配。
 - **USAGE_GRAPHICS_VERTEX**: 在GPU的顶点内存空间分配。
 - **USAGE_GRAPHICS_CONSTANTS**: 在GPU的被各种程序对象使用的常量区分配。

使用位操作符`OR`可以指定多种内存空间。这样做就告诉渲染脚本运行时你打算在指定的内存空间访问数据。下面的例子在脚本内存空间和GPU纹理内存空间两处同时分配内存：

```
ScriptField_Point touchPoints = new
ScriptField_Point(myRenderScript, 2,
Allocation.USAGE_SCRIPT | Allocation.USAGE_GRAPHICS_VERTEX);
```

- 允许你以对象的形式创建**struct**实例的嵌套类**Item**。如果你的**Android**代码中，使用**struct**更有意义，那么这个嵌套类就灰常有用啦。当你完成操作这个对象之后，你可以通过调用**set(Item i, int index, boolean copyNow)**方法并设置**Item**到数组中指定的位置，把该对象推到分配的内存中。渲染脚本运行时自动拥有对新写入的内存的访问能力。
- 存取方法，用来**get**和**set**结构体中的值。每个存取方法有一个**index**参数来指定你要读写的**struct**在数组中的位置。每个 **setter**方法有一个**copyNew**参数来指定是否立即同步内存到渲染脚本运行时。如果要同步所有尚未同步的内存，调用**copyAll()**方法。
- **createElement()**方法，创建结构体在内存中的描述符。描述符被用来分配一个或者多个元素构成的内存。
- **resize()**方法，和C语言中的**realloc()**方法类似，允许你扩展之前分配的内存，同时保持先前的创建的值。
- **copyAll()**方法，同步**Android**框架层设置的内存到渲染脚本运行时。当你对一个成员调用了**set**方法时，有一个可选的 **copyNew**布尔值参数可以指定，**true**表示你调用**set**方法的时候同步内存。如果指定为**false**，可以调用一次**copyAll()**，所有没有同步的属性都会进行内存同步。

指针

指针被反射到它自己的脚本类中，保存在**project_root/gen/package/name/ScriptC_renderscript_filename**。你可以声明指向**struct**或者任何支持的**Renderscript**类型的指针，但是 **struct**不能包含指针或者嵌套数组。比如，如果你声明了下面的指向**struct**和**int32_t**的指针：

```
typedef struct Point {
    float2 position;
    float size;
} Point_t;

Point_t *touchPoints;
int32_t *intPointer;
```

那么会生成如下代码：

```
private ScriptField_Point mExportVar_touchPoints;
```

```

public void bind_touchPoints( ScriptField_Point v ) {
    mExportVar_touchPoints = v;
    if ( v == null ) bindAllocation( null, mExportVarIdx_touchPoints );
    else bindAllocation( v.getAllocation() ,
mExportVarIdx_touchPoints );
}

public ScriptField_Point get_touchPoints() {
    return mExportVar_touchPoints;
}

private Allocation mExportVar_intPointer;
public void bind_intPointer( Allocation v ) {
    mExportVar_intPointer = v;
    if ( v == null ) bindAllocation( null, mExportVarIdx_intPointer );
    else bindAllocation( v, mExportVarIdx_intPointer );
}

public Allocation get_intPointer() {
    return mExportVar_intPointer;
}

```

一个get方法和特殊的名为bind_pointer_name(而不是set())的方法被生成。这个方法允许你绑定在Android虚拟机分配的内存到渲染脚本运行时(你可以在.rs文件中分配内存)。更多信息请参考“内存相关的工作”。

内存分配API

使用渲染脚本的程序仍然在Android虚拟机中运行。但是，实际的渲染脚本代码在本地运行，并且需要访问分配在虚拟机中的内存。为了实现这些，必须把分配在虚拟机中的内存附加到渲染脚本运行时。这个过程叫做绑定，允许渲染脚本运行时与它需要但是不能直接分配的内存之间无缝工作。结果等价于在C层调用malloc函数。带来的好处是Android虚拟机能够对渲染脚本运行时进行垃圾回收与内存共享。绑定仅仅对动态分配的内存是不要的。静态分配的内存 在编译的时候就为渲染脚本创建了。

为了支持这种内存分配系统，有一系列的API用来允许Android虚拟机分配内存和提供与malloc相似功能的函数调用。这些类描述了内存如何分配和实施内存分配。为了更好地理解这些类如何工作，有必要想想它们跟一个简单的malloc调用的关系，如下例：

```
array = ( int * ) malloc( sizeof( int ) * 10 );
```

这个malloc调用可以分解为两个部分：要分配的内存的大小(sizeof(int))、

与之关联的应该分配的内存单元的个数(10)。Android框架为表示malloc本身提供了一个类，还为这两个部分提供了各自的类。

Element类表示了malloc调用的(sizeof(int))部分，并封装了一个单元的内存分配，比如一个简单的float值或者一个结构体。**Type**类封装了**Element**类和要分配的元素的数量(本例中是10)。你可以认为一个**Type**是一个**Element**的数组。**Allocation**类基于给定的**Type**类执行实际的内存分配工作，体现了实际分配的内存。

大多数情况下，不需要直接调用这些内存分配API。反射层的类自动生成代码来使用这些API，你所需要做的跟内存分配相关的工作就是调用在反射层中声明的一个类的构造器，然后绑定分配了内存的Allocation到Renderscript。也有一些情况下需要使用这些类来直接分配内存，比如从资源中载入一个位图，比如为指向原始类型的指针分配内存。可以参考后文的“分配和绑定内存到渲染脚本”一节。下表更详细地描述了这三个内存管理类：

Android对象类型	描述
Element	<p>一个Element描述一个内存分配的单元，可以有两种形式：基础的、复杂的。</p> <p>一个基础的元素包好一个单一的数据组件，数据可以是任何有效的Renderscript数据类型。比如，一个单一的float数值，一个 float4向量，或者单个RGB-565颜色。一个复杂的的元素包含一系列基础元素，从在渲染脚本代码声明的struct创建。比如，一个内存分配可以包含在内存中顺序排列的多个struct。每个 struct被当成是一个元素Element，而不是struct中的每个数据类型分别是一个元素。</p>
Type	<p>一个Type是一个内存分配的模版，由一个元素和一个或多个维度组成。它描述了内存的布局(基</p>

本是一组元素)但是不为它描述的数据分配内存。

一个Type包含5个维度: X, Y, Z, LOD(level of detail), Faces(立方图的)。你可以将X,Y,Z赋值为任何内存允许的正整数。一维分配拥有一个大于0的X维度, 同时Y和Z维度都是0表示不存在。比如, 一个x=10,y=1的内存分配是二维的, x=10,y=0是一维的。LOD和Faces是布尔值, 表示存在或者不存在。

Allocation

一个Allocation根据Type表示的内存描述提供程序内存。分配的内存可以同时存在与多个内存空间。如果一个内存空间中的内存被更改了, 必须显示同步内存, 以便它所在的其他多个内存空间能够更新。

Allocation数据主要通过两种途径上传: 检查类型的、不检查类型的。对于简单的数组, 通过copyFrom()函数从 Android系统读取一个数组并复制到本地内存。不检查类型的方式允许Android系统复制结构体的数组, 因为Allocation不支持结构体。比如, 如果有一个n分浮点数的数组的内存分配, 数据被包含在一个float[n]数组中, 或者一个byte[n*4]数组中。

处理内存

渲染脚本中声明的非静态的全局变量是在编译期间就分配内存的。对于这些变量, 不用在Android框架级别为它们分配内存就可以在渲染脚本代码中直接使用。Android框架层也可以通过反射层自动生成的存取方法来访问这些变量。如果这些变量在渲染框架运行时层中初始化, 那它们就同时初始

化了 Android 框架层中相应的值。如果一个全局变量被声明为 `const` 的，系统就不会为这个变量生成 `set` 方法。

注：如果你使用包含指针的渲染脚本结构体，比如 `rs_program_fragment` 和 `rs_allocation`，你就必须先取出一个相应 Android 框架类的对象，然后调用该结构体的 `set` 方法来绑定内存到渲染脚本运行时，而不能在渲染脚本运行时层中直接操作这些结构。这些限制仅对系统 预定义的结构体适用，因为用户自定义的结构体不可以包含指针。声明一个包含指针的非静态全局结构体会导致编译错误。

渲染脚本支持指针，但是必须在 Android 框架代码中为其显示分配内存。如果你在 `.rs` 文件中声明了一个全局指针，你就需要通过合适的反射层类分配内存，并将内存绑定到本地渲染脚本层。你可以同时从 Android 框架层和渲染脚本层访问这些内存，这样就能够 在最合适的代码层灵活地修改相关变量。

分配和绑定动态内存到渲染脚本

为了分配动态内存，最常见的做法是调用一个 `Script.FieldBase` 类的构造器。当然，也可以手动地创建一个 `Allocation` 来实现，但是这需要原始类型指针等东西。为了简化，应该使用 `Script.FieldBase` 类构造器。在获取到内存分配之后，需调用反射出来的指针的 `bind` 方法来绑定内存到渲染脚本运行时。下面的例子展示了如何为原始类型指针 `intPointer` 和 结构类型指针 `touchPoints` 动态分配内存，然后绑定内存到渲染脚本：

```
private RenderScript myRenderScript;
private ScriptC_example script;
private Resources resources;

public void init(RenderScript rs, Resources res) {
    myRenderScript = rs;
    resources = res;

    // 调用构造器为结构指针分配内存
    ScriptField_Point touchPoints = new
    ScriptField_Point(myRenderScript, 2);

    // 创建一个 Element 为 intPointer 手动分配内存
    intPointer = Allocation.createSized(myRenderScript,
    Element.I32(myRenderScript), 2);
```

```

//创建一个Renderscript实例并指向字节码
mScript = new ScriptC_example(myRenderScript, resources,
R.raw.example);

//绑定指针到Renderscript
mScript.bind_touchPoints(touchPoints);
script.bind_intPointer(intPointer);

...
}

```

读写内存

在渲染脚本运行时和Android框架层都可以对静态和动态分配的内存进行读写操作。静态分配的内存 在渲染脚本运行时层级只能进行单向通信。当渲染脚本代码修改了一个变量的值，出于效率的考虑，它不会被传回Android框架层。调用 `get`方法时，返回的值总是最后一次从Android框架层（而不是渲染脚本）设定的值。但是，渲染脚本代码修改了一个变量的值，会自动修改到 Android框架层中的对应变量，或者稍后同步。如果你需要把一个数据从渲染脚本层发送到Android框架层，你就必须使用 `rsSendToClient()`函数来突破这些限制。当处理动态分配的内存时，如果使用关联的指针修改了内存分配，任何渲染脚本层的改动都会回传至Android框架层。在Android框架层对一个对象做的任何改动，都会立即回传至渲染脚本运行时。

读写全局变量

读写全局变量是一个很简单的过程。你可以在Android框架层使用访问器方法，或者直接在脚本运行时直接设置它们。记住，你在渲染脚本代码中做的改动，都不会回传到回传到Android框架层。例如，在一个`rsfile.rs`文件中给定了如下的结构：

```

typedef struct Point {
    int x;
    int y;
} Point_t;

Point_t point;

```

你可以直接在`rsfile.rs`文件中像下面一样对变量赋值。这些值不会被回传至Android框架层：

```
point.x = 1;
point.y = 1;
```

你也可以在Android框架层像下面一样对变量赋值。这些值会回传至渲染脚本运行时。

```
ScriptC_rsfile mScript;
...
Item i = new ScriptField_Point.Item();
i.x = 1;
i.y = 1;
mScript.set_point(i);
```

你可以在渲染脚本中像下面一样读取变量值：

```
rsDebug("Printing out a Point", point.x, point.y);
```

你可以在Android框架层中使用如下代码读取变量值。记住，这段代码只返回在Android框架层设定的值。如果你只在渲染脚本中而没有在Android框架层中为变量设值，你将得到一个空指针异常。

```
Log.i("TAGNAME", "Printing out a Point: " + mScript.get_point().x +
" " + mScript.get_point().y);
System.out.println(point.get_x() + " " + point.get_y());
```

读写全局指针

内存已经在Android框架层动态分配并绑定到了渲染脚本运行时之后，你就可以在Android框架层使用那些指针的set和get方法来读写内存了。在脚本运行时中，你可以照常使用指针读写内存，并且这些改动会被传回Android框架层，这一点与静态分配的内存不同。例如，在rsfile.rs文件中给定了如下指针：

```
typedef struct Point {
    int x;
    int y;
} Point_t;

Point_t *point;
```

因为你已经在Android框架层动态分配了内存，你就可以像平常一样访问**struct**中的变量。你通过指针变量对结构体做的改动也会在Android框架层自动生效。

```
point [ index ].x = 1;  
point [ index ].y = 1;
```

你也可以在Android框架层读写指针：

```
ScriptField_Point p = new ScriptField_Point (mRS, 1);  
Item i = new ScriptField_Point.Item();  
i.x=100;  
i.y = 100;  
p.set(i, 0, true);  
mScript.bind_point(p);  
  
points.get_x(0); //read x and y from index 0  
points.get_x(0);
```

一旦内存被绑定，以后就不必每次修改变量值都重新绑定内存了。

来自 "[index.php?title=Advanced_Renderscript&oldid=9090](#)"



Runtime API Reference

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

[OverView](#)

[Globals](#)

[Structs](#)

来自“[index.php?title=Runtime_API_Reference&oldid=7068](#)”



Media and Camera

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： 夔娘

原文链接：<http://docs.eoeandroid.com/guide/topics/media/index.html>

媒体和照相

在你的应用中增加视频，音频和照相功能，通过Android强大的APIs来播放音频和录制音频。

[媒体播放](#)

博客文章

使应用程序能够互相发挥的更好：处理远程控制按钮

如果你的媒体播放应用程序创建了一个媒体播放服务，就像音乐，媒体按钮事件响应，用户将如何知道这些事件运行在哪里？在音乐中，还是在你新的应用程序中？

使Android游戏运行地更完美

在Android上制作一款游戏很简单，但是制作一款大的游戏为一个手机，一个任务的，多内核的，多用途的系统像Android是需要技巧的。甚至那些出色的开发者在Android系统和其它的应用程序互相交互的时候也经常犯错误。

更多的Android游戏可以运行地更好

Android用户经常使用back键。我们期望这些按键在直观的方式上工作。我

们期望home键运行的方式如Android的导航模式相一致。

练习

拍摄照片

这个类可以让你利用现有的相机应用的一些超级简单的方法快速点击。在稍后的课程中，你会深入的学习到怎么直接地来控制摄像头硬件。

管理音频播放

这个类后，你将能自己开发一些能响应硬件音频按键的应用程序，要求音频设备播放音频时，系统或其他应用程序所造成的音频焦点的变化作出适当反应。

来自“[index.php?title=Media_and_Camera&oldid=9093](#)”



Media Playback

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：我喝咖啡

原文链

接：<http://docs.eoeandroid.com/guide/topics/media/mediaplayer.html>

[← 返回 Multimedia and Camera](#)

目录

[[隐藏](#)]

[1 媒体播放 - Media Playback](#)

- [1.1 基础知识 - The Basics](#)
- [1.2 清单声明 - Manifest Declarations](#)
- [1.3 使用MediaPlayer - Using MediaPlayer](#)
 - [1.3.1 异步准备 - Asynchronous Preparation](#)
 - [1.3.2 状态管理 - Managing State](#)
 - [1.3.3 释放MediaPlayer-Releasing the MediaPlayer](#)
- [1.4 使用服务控制MediaPlayer - Using a Service with MediaPlayer](#)
 - [1.4.1 异步运行 - Running asynchronously](#)
 - [1.4.2 处理异步错误 - Handling asynchronous errors](#)
 - [1.4.3 使用唤醒锁 - Using wake locks](#)
 - [1.4.4 作为前景服务运行 - Running as a foreground service](#)
- [1.5 处理音频焦点 - Handling audio focus](#)
 - [1.5.1 执行清理 - Performing cleanup](#)
- [1.6 处理AUDIO_BECOMING_NOISY意图 - Handling the AUDIO_BECOMING_NOISY Intent](#)
- [1.7 从内容解析器检索媒体 - Retrieving Media from a Content Resolver](#)

媒体播放 - Media Playback

Android的多媒体框架包括支持播放多种常见的媒体类型，使您可以轻松地把音频、视频和图像集成到你的应用。你可以播放音频或视频媒体文件，这些文件是存储在你的应用程序的资源文件中的。应用程序的资源文件可以是文件系统中独立的文件，或通过网络连接获取的一个数据流，所有使用[MediaPlayer API](#)的资源文件。

本文档给你说明如何编写一个能与用户进行交互并且性能良好、用户体验良好的media-playing应用程序。

注意：你只能在标准输出设备上播放音频数据。目前，标准输出设备是移动设备的扬声器或耳机。你不能在谈话音频调用期间播放声音文件。

基础知识 - The Basics

下面的类是Android框架中用于播放声音的类：

[媒体播放器](#)

这个类主要用于播放声音和视频。

[音频管理器](#)

这个类主要管理音频和音频输出设备。

清单声明 - Manifest Declarations

在开发你的应用程序中使用媒体播放器前，先确保你的Manifest有适当的声明来允许使用相关的功能。

- Internet许可 -如果你使用的是基于网络内容的流媒体播放器，你的应用程序必须请求访问网络。

```
<uses-permission android:name="android.permission.INTERNET" />
```

- 唤醒锁定许可 - 如果你的程序需要保持屏幕变暗或处理器睡眠，或使用[MediaPlayer.setScreenOnWhilePlaying\(\)](#)或[MediaPlayer.setWakeMode\(\)](#)方法的时候，你必须请求许可。

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

使用MediaPlayer - Using MediaPlayer

[MediaPlayer](#) 类是媒体框架最重要的组成部分之一。这个类的对象只要安装应用程序运行必需的最少数目的功能和文件，就可以读取，解码，并播放音频和视频。它支持多种不同的媒体来源，如：

- 本地资源
- 内部的URIs，比如你可能获得的内容解析器
- 外部的URIs（流）

Android支持的媒体格式的清单，请参照[Android Supported Media Formats](#)文档。

这是一个如何播放本地音频的例子（资源文件保存在你的应用程序res/raw/directory的资源文件目录下）：

```
MediaPlayer mediaPlayer = MediaPlayer.create(context,
R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does
that for you
```

在这种情况下，文件系统是不会以任何特定的方式去分析一个"raw"资源文件的。然而，这一资源内容不应该是原始音频。它应该是一种支持的格式的适当编码和格式化媒体文件

这是你如何通过URI从可用的本地系统中播放（这是你通过内容解析器获得的数据，例如）：

```
Uri myUri = ....; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

通过一个远程的URL经由HTTP流来播放，就像下面这样子：

```
String url = "http://....."; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
```

```
MediaPlayer.setDataSource(url);
MediaPlayer.prepare(); // might take long! (for buffering, etc)
MediaPlayer.start();
```

注意：如果你通过一个URL来获取一个在线媒体文件，该文件必须能够支持渐进式下载。

警告：当你使用setDataSource()方法时，必须捕捉或通过非法数据异常和输入输出异常，因为正在被你引用的文件可能不存在。

异步准备 - Asynchronous Preparation

使用[MediaPlayer](#) 的原则很简单。然而，重要的是要记住，有必要将更多的一些东西正确地集成到一个典型的Android应用程序。比如，调用[prepare\(\)](#)方法可能需要较长的时间来执行，因为它可能涉及获取和解码媒体数据。因此，如同任何方法，可能需要很长时间执行，你不应该从应用程序的UI线程调用它。这样做将导致UI挂到方法返回，这是一个非常糟糕的用户体验，也可能会引起应用程序没有响应的错误。即使你预期你的资源能快速加载，记住，任何超过十分之一秒的反应在界面上会造成明显的停顿，将导致给用户的印象是：你的应用程序是缓慢的。

为了避免你的UI线程挂起，产生另一个线程准备[MediaPlayer](#) 当完成时通知主线程。你可以写自己的线程的逻辑，框架提供[prepareAsync\(\)](#) 方法，方便的使用[MediaPlayer](#)。该方法在后台开始准备媒体并立即返回。当媒体准备好了，[MediaPlayer.OnPreparedListener](#)的[onPrepared](#) 方法，通过配置[setOnPreparedListener\(\)](#)方法来调用。

状态管理 - Managing State

关于[MediaPlayer](#)，你需要记住的另一点是它的状态。即，在你编写自己的代码的时候，必须时刻意识到[MediaPlayer](#) 有一个内部状态，因为只有当玩家在特定状态，某些特定的操作才会有效。如果您在错误的状态执行一个操作，系统可能会抛出一个异常或引起其它令人不快的行为。

[MediaPlayer](#) 类里有文件显示一个完整的状态转换图，阐明哪些方法可以把[MediaPlayer](#) 从一个状态改变到另一个状态。例如，当您创建一个新

的 [MediaPlayer](#) ,它就处于闲置状态。这时,你应该调用[android.net.Uri\)](#)
[setDataSource\(\)](#) 方法来初始化它, 把它设置为初始化状态。然后, 你必须
使用[prepare\(\)](#)方法或[prepareAsync\(\)](#)方法。当[MediaPlayer](#) 准备好了, 它将
进入准备状态,这就意味着你可以调用[start\(\)](#)方法来播放媒体。另外, 在状态
转换图上阐明了, 你可以调用[start\(\)](#),[pause\(\)](#)和 [seekTo\(\)](#)这些方法
在Started, Paused和PlaybackCompleted状态之间进行转换。如果您调
用[stop\(\)](#)方法,这时请注意,你需要再次准备[MediaPlayer](#) , 才可以再一次调
用[start\(\)](#)方法。

在编写代码与[MediaPlayer](#) 对象交互时, 心里要随时想着状态转换图, 因为
从错误的状态调用它的方法,是引起错误的常见原因。

释放MediaPlayer-Releasing the MediaPlayer

[MediaPlayer](#) 会消耗宝贵的系统资源。因此 , 你应该经常采取额外的预防措
施来确保及时把不需要的[MediaPlayer](#) 取消掉。您需要调用[release\(\)](#) 方法来
确保系统分配给它的资源正确释放。例如,您正在使用[MediaPlayer](#) , 同时,
你的活动调用[onStop\(\)](#)方法, 这时你必须释放[MediaPlayer](#),因为你的活动并
非与用户交互, 留着它没什么意义 (除非你是在后台播放多媒体,这是下一
节中将讨论的内容)。当你的活动恢复或者重新启动, 恢复播放之前,您需
要创建一个新的[MediaPlayer](#)并且重新准备。

下面是释放和取消你的[MediaPlayer](#)的方法:

```
mediaPlayer.release();
mediaPlayer = null;
```

作为思考题, 考虑一下如果当活动停止的时候你忘了释放[MediaPlayer](#) , 活
动重启后新建一个[MediaPlayer](#) , 可能会发生的问题。正如你可能知道的,
当用户更改屏幕的方向 (或以另一种方式更改设备配置) , 该系统通过重启
活动处理 (通过默认方式) , 所以当用户频繁在纵向和横向之间切换时, 你
可能会很快消耗掉所有的系统资源, 原因是你没有释放方向变化时各个方向
上创建的新[MediaPlayer](#)。(更多关于运行时重启的资料,请查看[Handling
Runtime Changes](#))。

你可能会想知道在用户离开活动时后台继续播放媒体是如何实现的, 采用同
样的方式实现的, 如内置的音乐应用程序的行为。在这种情况下,你需要通
过一个[Service](#)来控制[MediaPlayer](#) , 所以我们开始学习[Using a Service with
MediaPlayer](#)。

使用服务控制MediaPlayer - Using a Service with MediaPlayer

如果你希望后台播放媒体，你希望用户操作其他应用时继续播放，你必须开始一个[Service](#)并且从那里控制[MediaPlayer](#)实例。你必须慎重考虑这个设置，因为用户与系统期望应用程序运行的后台服务应该与系统的其余部分相互作用。如果应用程序不满足这些预期，就不能有良好的用户体验。本节介绍的主要内容是：告诉你相关知识，并提供建议如何接触它们。

异步运行 - Running asynchronously

首先，如一个[Activity](#)，服务里的所有任务默认在单一线程中完成。如果你从同一个应用程序里运行一个[Activity](#)和一个[Service](#)，它们默认使用相同的线程（“主线程”）。因此，[Service](#)需要迅速处理传入的意图并且响应它们的时候从不执行冗长的计算。如果预计调用一些复杂的任务或阻塞，你必须异步处理这些任务：由另一个线程自己实现自己，或使用框架处理异步。

例如，当你从主要线程使用一个[MediaPlayer](#)，你应该调用[prepareAsync\(\)](#)方法而不是[prepare\(\)](#)方法，实

现[MediaPlayer.OnPreparedListener](#)，以便当你准备工作完毕后，得到可以开始播放的通知。

代码如下：

```
public class MyService extends Service implements
MediaPlayer.OnPreparedListener {
    private static final ACTION_PLAY = "com.example.action.PLAY";
    MediaPlayer mMediaPlayer = null;

    public int onStartCommand(Intent intent, int flags, int startId)
    {
        ...
        if (intent.getAction().equals(ACTION_PLAY)) {
            mMediaPlayer = ... // initialize it here
            mMediaPlayer.setOnPreparedListener(this);
            mMediaPlayer.prepareAsync(); // prepare async to not
block main thread
        }
    }
    /** Called when MediaPlayer is ready */
    public void onPrepared(MediaPlayer player) {
        player.start();
    }
}
```

处理异步错误 - Handling asynchronous errors

在同步操作中，错误通常会出现异常或错误代码信息。但当你使用异步资源时，您需要确保您的应用程序有错误提示，在[MediaPlayer](#)中，要做到这一点，可以通过实现[MediaPlayer.OnErrorListener](#)，并且将它设置在你

的 [MediaPlayer](#) 实体中。

```
public class MyService extends Service implements
MediaPlayer.OnErrorListener {
    MediaPlayer mMediaPlayer;
    public void initMediaPlayer() {
        // ...initialize the MediaPlayer here...
        mMediaPlayer.setOnErrorListener(this);
    }
    @Override
    public boolean onError(MediaPlayer mp, int what, int extra) {
        // ... react appropriately ...
        // The MediaPlayer has moved to the Error state, must be
        reset!
    }
}
```

请牢记,当出现错误, 将这个[MediaPlayer](#) 设置为错误状态 (请参考[MediaPlayer](#) 类文档的完整的状态关系图) 您再次使用它之前, 必须重置这个状态。

使用唤醒锁 - Using wake locks

应用程序在后台播放媒体, 其服务在运行期间, 设备可能会进入休眠状态。因为Android系统希望在设备休眠时节省电池。系统试图关闭手机的应用程序, 是没有必要的, 包括CPU和WiFi硬件。但是, 如果你的服务正在运行或播放着音乐, 你希望防止系统干扰你的回放。

为了确保您的服务在这些条件下能继续运行, 你需要使用“wake locks”。唤醒锁是一种信号系统, 它发出信号, 显示: 应用程序正在使用或可用的功能, 或手机闲置。

注意:你应该尽量少用唤醒锁, 只有在必要时候才使用它们。它们会使设备的电池寿命大大降低。

你[MediaPlayer](#) 正在播放时, 需要确保CPU持续运行, 当初始化你的[MediaPlayer](#)时, 调用[setWakeMode\(\)](#)方法。一旦你这样做了, 当暂停或停止时候, [MediaPlayer](#) 持有指定的锁:

```
mMediaPlayer = new MediaPlayer();
// ... other initialization here ...
mMediaPlayer.setWakeMode(getApplicationContext(),
PowerManager.PARTIAL_WAKE_LOCK);
```

在这个例子中获得唤醒锁是指在保证CPU在唤醒状态。当你通过网络获取媒体和您正在使用WiFi时,你可能希望有个WifiLock, 可以手动获取并释放。当你开始通过远程URL准备MediaPlayer, 你应该创建并获得wi - fi锁。代码如下:

```
WifiLock wifiLock = ((WifiManager)
getSystemService(Context.WIFI_SERVICE))
.createWifiLock(WifiManager.WIFI_MODE_FULL, "mylock");
wifiLock.acquire();
```

当你暂停或停止你的媒体时,或当你不再需要这样的网络,你应该释放该锁:代码如下:

```
wifiLock.release();
```

作为前景服务运行 - **Running as a foreground service**

服务通常用于执行后台任务, 例如获取电子邮件, 同步数据, 下载内容, 或其他。在这些情况下, 用户不会意识到这个服务的执行, 甚至可能不会注意到这些 服务被打断,后来重新启动。毫无疑问, 后台播放音乐是一个服务, 用户能意识到, 任何中断都会严重影响到用户体验。此外, 用户可能会希望在这个服务执行期间 作用于它。这种情况, 服务应该运行一个“前景服务”。前台服务在系统中持有一个更高水平的重要性, 系统几乎从未将服务扼杀, 因为它对用户有着直接的重要 性。当应用在前台运行, 该服务还必须提供一个状态栏来通知用户意识有服务正在运行同时允许他们打开一个活动,可以与服务进行交互。

为了把你的服务变为前景服务, 您必须为状态栏创建一个[Notification](#), 并且从[Service](#)调用[startForeground\(\)](#)方法。

代码如下:

```
String songName;
// assign the song name to songNamePendingIntent pi =
PendingIntent.getActivity(getApplicationContext(), 0,
        new Intent(getApplicationContext(),
MainActivity.class),
        PendingIntent.FLAG_UPDATE_CURRENT);
Notification notification = new Notification();
notification.tickerText = text;
notification.icon = R.drawable.play0;
notification.flags |= Notification.FLAG_ONGOING_EVENT;
notification.setLatestEventInfo(getApplicationContext(),
"MusicPlayerSample",
        "Playing: " + songName, pi);
startForeground(NOTIFICATION_ID, notification);
```

通知区域可见的设备告诉你，服务在前台运行。如果用户选择了这个通知，系统将调用你提供的PendingIntent。在上面的例子中，它打开了一个Activity。（MainActivity）

图1显示了如何将通知呈现给用户：

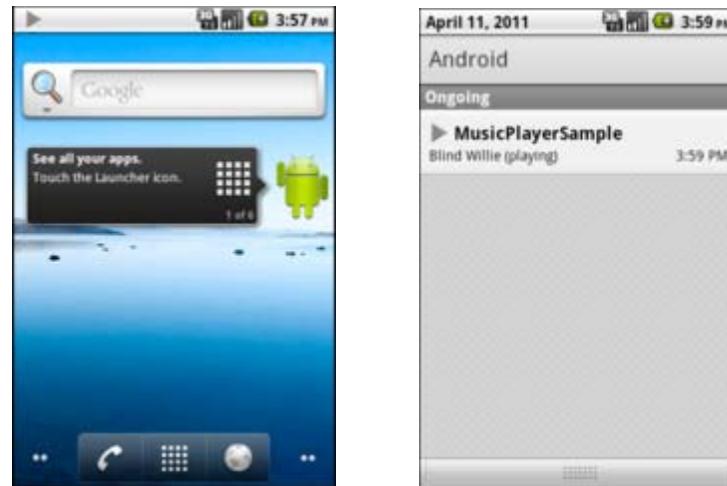


图1:界面的一个前景服务通知,如上图, 显示通知图标(在左)、扩展视图(在右)。

实际执行一些用户能够意识到的服务时，你应该保留“foreground service”的状态。相反情况下，你应该调用stopForeground()方法来释放它。代码如下：

```
stopForeground ( true ) ;
```

更多信息，请参考[Service](#)和[Status Bar Notifications](#)的相关文档。

处理音频焦点 - Handling audio focus=

在给定的时间尽管只有一个活动可以运行,但Android是一个多任务环境。这对应用程序使用音频造成了一个特别大的难度，由于只有一个音频输出,可能会有好几个媒体服务争夺使用它。Android 2.2之前,没有内置机制来解决这个问题，这可能在某些情况下导致糟糕的用户体验。例如，一个用户正在听音乐，同时，另一个应用程序有很重要的事需要通知用户，由于吵闹的音乐用户可能不会听到提示音。从Android 2.2开始,Android平台为应用程序提供了一个方式来协商设备的音频输出。这个机制被称为音频焦点。

当您的应用程序需要输出音频如音乐或一个通知,这时你就必须请求音频焦点。一旦得到焦点，它就可以自由的使用声音输出设备，同时它会不断监听焦点的更改。如果它被告知已经失去了音频焦点，它会要么立即杀死音频或立即降低到一个安静的水平（被称为“ducking”——有一个标记,指示哪

个是适当的) 当它再次接收焦点时, 继续不断播放。

音频焦点是自然的合作。应用程序都期望(强烈鼓励)遵守音频焦点指南, 但规则并不是系统强制执行的。如果应用程序失去音频焦点后想要播放嘈杂的音乐, 在系统中没有什么会阻止他。然而, 这样可能会让用户有更糟糕的体验, 并可能卸载这运行不当的应用程序。

请求音频焦点, 您必须从[AudioManager](#)调用[requestAudioFocus\(\)](#)方法, 下面展示一个例子:

```
AudioManager audioManager = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);
int result = audioManager.requestAudioFocus(this,
AudioManager.STREAM_MUSIC,
AudioManager.AUDIOFOCUS_GAIN);

if (result != AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    // could not get audio focus.}
```

[requestAudioFocus\(\)](#)的第一个参数

是[AudioManager.OnAudioFocusChangeListener](#), 每当音频焦点有变动的时候其[onAudioFocusChange\(\)](#)方法被调用。您还应该在你的服务和活动上实现这个接口。

代码如下:

```
class MyService extends Service
    implements AudioManager.OnAudioFocusChangeListener {
    ...
    public void onAudioFocusChange(int focusChange) {
        // Do something based on focus change...
    }
}
```

[focusChange](#)参数告诉你音频焦点是如何改变的, 并且可以使用以下的值之一(他们都是在[AudioManager](#)中定义常量的):

- [AUDIOFOCUS_GAIN](#): 你已经得到了音频焦点。
- [AUDIOFOCUS_LOSS](#): 你已经失去了音频焦点很长时间了。你必须停止所有的音频播放。因为你应该不希望长时间等待焦点返回, 这将是你尽可能清除你的资源的一个好地方。例如, 你应该释放[MediaPlayer](#)。
- [AUDIOFOCUS_LOSS_TRANSIENT](#): 你暂时失去了音频焦点, 但很快会重新得到焦点。你必须停止所有的音频播放, 但是你可以保持你的资源, 因为你可能很快会重新获得焦点。
- [AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK](#): 你暂时失去了音频焦点, 但你可以小声地继续播放音频(低音量)而不是完全扼杀音频。

下面是一个示例实现:

```

public void onAudioFocusChange(int focusChange) {
    switch (focusChange) {
        case AudioManager.AUDIOFOCUS_GAIN:
            // resume playback
            if (mMediaPlayer == null) initMediaPlayer();
            else if (!mMediaPlayer.isPlaying())
                mMediaPlayer.start();
            mMediaPlayer.setVolume(1.0f, 1.0f);
            break;
        case AudioManager.AUDIOFOCUS_LOSS:
            // Lost focus for an unbounded amount of time: stop
playback and release media player
            if (mMediaPlayer.isPlaying()) mMediaPlayer.stop();
            mMediaPlayer.release();
            mMediaPlayer = null;
            break;
        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
            // Lost focus for a short time, but we have to stop
// playback. We don't release the media player because
playback
            // is likely to resume
            if (mMediaPlayer.isPlaying()) mMediaPlayer.pause();
            break;
        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
            // Lost focus for a short time, but it's ok to keep
playing
            // at an attenuated level
            if (mMediaPlayer.isPlaying())
                mMediaPlayer.setVolume(0.1f, 0.1f);
            break;
    }
}

```

记住,音频焦点APIs在API级别8(Android 2.2)及以上才有效。所以如果你想要支持的以前版本的Android, (如果有的话)你应该采取一种向后兼容性策略,允许您使用该特性, (如果没有的话), 只能选择8以后的版本。通过反射调用音频焦点方法或通过在一个单独类中实现所有的音频焦点特性, 您可以实现向后兼容性 ([AudioFocusHelper](#) 中阐明)。下面是这样一个类的示例:

```

public class AudioFocusHelper implements
AudioManager.OnAudioFocusChangeListener {
    AudioManager mAudioManager;
    // other fields here, you'll probably hold a reference to an
interface
    // that you can use to communicate the focus changes to your
Service

    public AudioFocusHelper(Context ctx, /* other arguments here */)
{
        mAudioManager = (AudioManager)
mContext.getSystemService(Context.AUDIO_SERVICE);
        // ...
}

```

```

    }

    public boolean requestFocus() {
        return AudioManager.AUDIOFOCUS_REQUEST_GRANTED ==
            mAudioManager.requestAudioFocus(mContext,
        AudioManager.STREAM_MUSIC,
            AudioManager.AUDIOFOCUS_GAIN);
    }

    public boolean abandonFocus() {
        return AudioManager.AUDIOFOCUS_REQUEST_GRANTED ==
            mAudioManager.abandonAudioFocus(this);
    }

    @Override
    public void onAudioFocusChange(int focusChange) {
        // let your service know about the focus change
    }
}

```

当你发现系统运行时API级别在8级或以上时，您可以创建**AudioFocusHelper**类的一个实例，例如：

```

if (android.os.Build.VERSION.SDK_INT >= 8) {
    mAudioFocusHelper = new
    AudioFocusHelper(getApplicationContext(), this);
} else {
    mAudioFocusHelper = null;
}

```

执行清理 - Performing cleanup

正如前面所提到的，[MediaPlayer](#) 对象会消耗大量的系统资源，所以你可以在你需要用他的时候保留他，当你不需要他的时候调用[release\(\)](#) 方法。调用这个显式地清除方法是重要的，而不是依赖于系统的垃圾收集，因为它可能需要一些时间垃圾收集器才能收回[MediaPlayer](#)，因为这只是内存敏感的需求而不是其他媒体资源的短缺。既然如此，当你使用一个服务时，你应该覆盖[onDestroy\(\)](#)方法，来确定你的[MediaPlayer](#)释放了。

代码如下：

```

public class MyService extends Service {
    MediaPlayer mMediaPlayer;
    // ...

    @Override
    public void onDestroy() {
        if (mMediaPlayer != null) mMediaPlayer.release();
    }
}

```

你最好始终寻找其它机会来释放你的[MediaPlayer](#)，关闭的时候就释放掉。例如，如果你期望较长的一段时间不能够播放媒体（例如，失去音频焦点后），你肯定得先释放你现有的[MediaPlayer](#)，以后要用的时候再创建它。另一方面，如果你只希望停止播放很短的一段时间，你应该尽量保留你的[MediaPlayer](#)，以免花费时间重新创建和准备一遍。

处理AUDIO_BECOMING_NOISY意图 - Handling the AUDIO_BECOMING_NOISY Intent

许多编写良好的应用程序有以下特点，当一个事件导致音频变得聒噪时，自动停止音频播放。（通过外部扬声器输出）。例如，一个用户戴着耳机听音乐，可能会不小心切断耳机和设备的链接。虽然，这种行为不会自动发生。如果您没有实现这个特性，设备的外部扬声器会将音频播放出来，这可能是用户不希望发生的。

这些情况下通过处理[ACTION_AUDIO_BECOMING_NOISY](#)意图，可以让你的应用程序停止播放音乐，通过在你的manifest里添加以下代码，你可以注册一个接收器：

```
<receiver android:name=".MusicIntentReceiver">
    <intent-filter>
        <action android:name="android.media.AUDIO_BECOMING_NOISY" />
    </intent-filter></receiver>
```

注册[MusicIntentReceiver](#)类当作一个广播接收器的意图，然后您应该实现这类，代码如下：

```
public class MusicIntentReceiver implements
    android.content.BroadcastReceiver {
    @Override
    public void onReceive(Context ctx, Intent intent) {
        if (intent.getAction().equals(
            android.media.AudioManager.ACTION_AUDIO_BECOMING_NOISY)) {
            // signal your service to stop playback
            // (via an Intent, for instance)
        }
    }
}
```

从内容解析器检索媒体 - Retrieving Media from a Content Resolver

在媒体播放器应用程序中，另一个可能有用的特性是用户可以在设备上检索音乐。你可以通过为外部媒体查询[ContentResolver](#)，代码如下：

```
ContentResolver contentResolver = getContentResolver();
```

```

Uri uri =
    android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
Cursor cursor = contentResolver.query(uri, null, null, null, null);
if (cursor == null) {
    // query failed, handle error.
} else if (!cursor.moveToFirst()) {
    // no media on the device
} else {
    int titleColumn =
cursor.getColumnIndex(android.provider.MediaStore.Audio.Media.TITLE);
    int idColumn =
cursor.getColumnIndex(android.provider.MediaStore.Audio.Media._ID);
    do {
        long thisId = cursor.getLong(idColumn);
        String thisTitle = cursor.getString(titleColumn);
        // ...process entry...
    } while (cursor.moveToNext());
}

```

要和[MediaPlayer](#)一起使用,你可以这样做,代码如下:

```

long id = /* retrieve it from somewhere */;
Uri contentUri = ContentUris.withAppendedId(
    android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, id);
mMediaPlayer = new MediaPlayer();
mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mMediaPlayer.setDataSource(getApplicationContext(), contentUri);
// ...prepare and start...

```

[← 返回 Multimedia and Camera](#)

[↑ 回到顶部](#)

来自“[index.php?title=Media_Playback&oldid=9114](#)”



Supported Media Formats

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：我喝咖啡

原文链接：<http://docs.eoeandroid.com/guide/appendix/media-formats.html>

[← 返回 Multimedia and Camera](#)

目录

[[隐藏](#)]

[1 Android支持的媒体格式-Android Supported Media Formats](#)

- [1.1 网络协议-Network Protocols](#)
- [1.2 核心媒体格式-Core Media Formats](#)
- [1.3 视频编码建议-Video Encoding Recommendations](#)

Android支持的媒体格式-Android Supported Media Formats

本文档描述了媒体编解码器、容器和Android平台提供的网络协议支持。

作为应用程序开发人员，你可以在任何可用的android设备上自由地使用任何媒体编解码器，包括所提供的Android平台，以及那些设备特许。然而，最佳实践是使用媒体的编码介绍设备无关的属性。

网络协议-Network Protocols

下面的网络协议支持音频、视频回放：

- RTSP (RTP, SDP)
- HTTP/HTTPS progressive streaming
- HTTP/HTTPS live streaming [draft protocol](#)：
 - MPEG-2 TS media files only
 - Protocol version 3 (Android 4.0 and above)
 - Protocol version 2 (Android 3.x)
 - Not supported before Android 3.0

注意:Android 3.1之前不支持HTTPS。

核心媒体格式-**Core Media Formats**

下表描述了可以支持构建到Android平台的媒体格式。请注意,任何给定的移动设备,可以提供支持附加的格式或文件类型的,不在下表中列出。

注意:媒体解码器,不能保证在Android平台上的所有版本都可用,因此在括号中指出,例如“(Android 3.0 +)”。

表1、 核心的媒体格式和编码支持。

类型	格式/编解码器	编码器	解码器	详细说明	支持文件类型(s)/集装箱格式
Audio	AAC LC/LTP	•	•	Mono/Stereo 内容的任何组合标准高达160 kbps的比特率和采样率从8至48 khz	<ul style="list-style-type: none"> • 3GPP (.3gp) • MPEG-4 (.mp4, .m4a) • ADTS raw AAC (.aac, decode in Android 3.1+, encode in Android 4.0+, ADIF not supported) • MPEG-TS (.ts, not seekable, Android 3.0+)

HE-AACv1 (AAC+)		•		
HE-AACv2 (enhanced AAC+)		•		
AMR-NB	•	•	4.75到12.2 kbps采样@ 8 khz	3GPP (.3gp)
AMR-WB	•	•	9率从6.60到23.85 kbit kbit / s / s采样@ 16千赫	3GPP (.3gp)
FLAC		• (Android 3.1+)	Mono/Stereo (没有多路)。48千赫采样率(但高达44.1 kHz推荐使用44.1 kHz的设备输出,为48至44.1 kHzdownsampler并不包括一个低通滤波器)。16位推荐;直接申请24位。	FLAC (.flac) only
MP3		•	Mono/Stereo 8 - 320 kbps常数(CBR)或可变码率(VBR)	MP3 (.mp3)
MIDI		•	MIDI输入0和1。DLS版本1和2。XMF和移动。支持铃声格式RTTTL / RTX,在线旅行社,iMelody	<ul style="list-style-type: none"> • Type 0 and 1 (.mid, .xmf, .mxmf) • RTTTL/RTX (.rtttl, .rtx)

					<ul style="list-style-type: none"> • OTA (.ota) • iMelody (.imy)
	Vorbis		•		<ul style="list-style-type: none"> • Ogg (.ogg) • Matroska (.mkv, Android 4.0+)
	PCM/WAVE		•	8到16位线性PCM(利率限制的硬件)	WAVE (.wav)
Image	JPEG	•	•	Base+progressive	JPEG (.jpg)
	GIF		•		GIF (.gif)
	PNG	•	•		PNG (.png)
	BMP		•		BMP (.bmp)
	WEBP	• (Android 4.0+)	• (Android 4.0+)		WebP (.webp)
Video	H.263	•	•		<ul style="list-style-type: none"> • 3GPP (.3gp) • MPEG-4 (.mp4)
	H.264 AVC	•	•	Baseline Profile (BP)	• 3GPP (.3gp)

	(Android 3.0+)			• MPEG-4 (.mp4) • MPEG-TS (.ts, AAC audio only, not seekable, Android 3.0+)
MPEG-4 SP		•		3GPP (.3gp)
VP8		•(Android 2.3.3+)	Streamable只有 在Android 4.0及以 上	• WebM (.webm) • Matroska (.mkv, Android 4.0+)

视频编码建议-Video Encoding Recommendations

表2,下面,列出了示例视频编码的概要文件和参数,Android媒体框架支持回放。除了这些编码参数的建议,一个设备可用的录像概要文件可以被用于衡量媒体播放功能。从API8以上的版本,这些概要文件可以检查使用CamcorderProfile类。

表2、 支持视频编码参数的例子。

	SD (低品质)	SD (高品质)	HD (不是在所有设备 上可用)
Video codec	H.264 Baseline Profile	H.264 Baseline Profile	H.264 Baseline Profile
Video resolution	176 x 144 px	480 x 360 px	1280 x 720 px
Video	12 fps	30 fps	30 fps

frame rate			
Video bitrate	56 Kbps	500 Kbps	2 Mbps
Audio codec	AAC-LC	AAC-LC	AAC-LC
Audio channels	1 (mono)	2 (stereo)	2 (stereo)
Audio bitrate	24 Kbps	128 Kbps	192 Kbps

对于视频内容,是通过HTTP或RTSP,还有附加条件:

- 3GPP和MPEG-4容器,moov atom必须先于任何mdat atom,但必须成功的ftyp atom。
- 3GPP,MPEG-4和WebM容器, 音频和视频的例子,同一时间偏置相距可能不超过500 KB。要最小化这种音频/视频漂移,考虑交叉音频和视频在较小块的大小。

[← 返回 Multimedia and Camera](#)

[↑ 回到顶部](#)

来自“[index.php?title=Supported_Media_Formats&oldid=9119](#)”



Audio Capture

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

[← 返回 Multimedia and Camera](#)

目录

[[隐藏](#)]

[1 音频采集 - Audio Capture](#)

- [1.1 In this document](#)
- [1.2 Key classes](#)
- [1.3 See also](#)

[2 执行音频捕捉-Performing Audio Capture](#)

音频采集 - Audio Capture

Android多媒体框架包括支持捕获和编码各种常见的音频格式,可以方便地集成音频到应用程序里。如果设备的硬件支持, 您可以使用[MediaRecorder APIs](#)录音。

本文档介绍了如何编写一个可以从设备的麦克风采集音频, 保存音频和播放音频的应用程序。

注意:Android模拟器无法捕捉的音频,但真实的设备一般都提供这些功能。

In this document

[Performing Audio Capture](#)
[Code Example](#)

Key classes

[MediaRecorder](#)

See also

[Android Supported Media Formats](#)

执行音频捕捉- Performing Audio Capture

[Data Storage](#)

[MediaPlayer](#)

从设备捕捉音频比音视频播放要更复杂些,但还是相当简单的:

- 1、创建一个新实例[android.media.MediaRecorder](#)。
- 2、使用[MediaRecorder.setAudioSource\(\)](#)设置音频源。你可能会想要使用[MediaRecorder.AudioSource.MIC](#)。
- 3、使用[MediaRecorder.setOutputFormat\(\)](#)设置输出文件格式。
- 4、使用[MediaRecorder.setOutputFile\(\)](#)设置输出文件名。
- 5、使用[MediaRecorder.setAudioEncoder\(\)](#)设置音频编码器。
- 6、在MediaRecorder实例上调用[MediaRecorder.prepare\(\)](#)。
- 7、调用[MediaRecorder.start\(\)](#)开始音频捕捉。
- 8、调用[MediaRecorder.stop\(\)](#)停止音频捕捉。
- 9、当您完成了MediaRecorder实例，调用它的[MediaRecorder.release\(\)](#)调用[MediaRecorder.release\(\)](#)总是建议立即释放资源。

例如:录制音频和播放录制的音频

下面的示例类说明了如何设置,启动和停止音频捕捉,和播放录制的音频文件。

```
/*
 * The application needs to have the permission to write to external
storage
 * if the output file is written to the external storage, and also
the
 * permission to record audio. These permissions must be set in the
* application's AndroidManifest.xml file, with something like:
*/
```

Audio Capture - eoeAndroid wiki

```
* <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
* <uses-permission android:name="android.permission.RECORD_AUDIO"
/>
*
*/
package com.android.audiorecordtest;

import android.app.Activity;
import android.widget.LinearLayout;
import android.os.Bundle;
import android.os.Environment;
import android.view.ViewGroup;
import android.widget.Button;
import android.view.View;
import android.view.View.OnClickListener;
import android.content.Context;
import android.util.Log;
import android.media.MediaRecorder;
import android.media.MediaPlayer;

import java.io.IOException;

public class AudioRecordTest extends Activity
{
    private static final String LOG_TAG = "AudioRecordTest";
    private static String mFileName = null;

    private RecordButton mRecordButton = null;
    private MediaRecorder mRecorder = null;

    private PlayButton mPlayButton = null;
    private MediaPlayer mPlayer = null;

    private void onRecord(boolean start) {
        if (start) {
            startRecording();
        } else {
            stopRecording();
        }
    }

    private void onPlay(boolean start) {
        if (start) {
            startPlaying();
        } else {
            stopPlaying();
        }
    }

    private void startPlaying() {
        mPlayer = new MediaPlayer();
        try {
            mPlayer.setDataSource(mFileName);
            mPlayer.prepare();
            mPlayer.start();
        } catch (IOException e) {
            Log.e(LOG_TAG, "prepare() failed");
        }
    }

}
```

```

private void stopPlaying() {
    mPlayer.release();
    mPlayer = null;
}

private void startRecording() {
    mRecorder = new MediaRecorder();
    mRecorder.set AudioSource(MediaRecorder.AudioSource.MIC);

    mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mRecorder.setOutputFile(mFileName);

    mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

    try {
        mRecorder.prepare();
    } catch (IOException e) {
        Log.e(LOG_TAG, "prepare() failed");
    }

    mRecorder.start();
}

private void stopRecording() {
    mRecorder.stop();
    mRecorder.release();
    mRecorder = null;
}

class RecordButton extends Button {
    boolean mStartRecording = true;

    OnClickListener clicker = new OnClickListener() {
        public void onClick(View v) {
            onRecord(mStartRecording);
            if (mStartRecording) {
                setText("Stop recording");
            } else {
                setText("Start recording");
            }
            mStartRecording = !mStartRecording;
        }
    };

    public RecordButton(Context ctx) {
        super(ctx);
        setText("Start recording");
        setOnClickListener(clicker);
    }
}

class PlayButton extends Button {
    boolean mStartPlaying = true;

    OnClickListener clicker = new OnClickListener() {
        public void onClick(View v) {
            onPlay(mStartPlaying);
            if (mStartPlaying) {
                setText("Stop playing");
            } else {

```

```

        setText( "Start playing" );
    }
    mStartPlaying = !mStartPlaying;
}
}

public PlayButton( Context ctx ) {
super( ctx );
setText( "Start playing" );
setOnClickListener( clicker );
}

public AudioRecordTest() {
    mFileName =
Environment.getExternalStorageDirectory().getAbsolutePath();
    mFileName += "/audiorecordtest.3gp";
}

@Override
public void onCreate( Bundle icicle ) {
super.onCreate( icicle );

LinearLayout ll = new LinearLayout( this );
mRecordButton = new RecordButton( this );
ll.addView( mRecordButton,
new LinearLayout.LayoutParams(
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT,
0 ) );
mPlayButton = new PlayButton( this );
ll.addView( mPlayButton,
new LinearLayout.LayoutParams(
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT,
0 ) );
setContentView( ll );
}

@Override
public void onPause() {
super.onPause();
if ( mRecorder != null ) {
mRecorder.release();
mRecorder = null;
}
if ( mPlayer != null ) {
mPlayer.release();
mPlayer = null;
}
}
}

```

[← 返回 Multimedia and Camera](#)

[↑ 回到顶部](#)

来自“[index.php?title=Audio_Capture&oldid=3031](#)”



JetPlayer

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

[← 返回 Multimedia and Camera](#)

目录

[[隐藏](#)]

[1 JetPlayer](#)

- [1.1 In this document](#)
- [1.2 Key classes](#)
- [1.3 Related Samples](#)
- [1.4 See also](#)

[2 播放JET内容-Playing JET content](#)

JetPlayer

Android平台有一个JET引擎，允许您添加交互播放的JET音频内容到您的应用程序。你可以通过附带SDK的JETCreator编写应用程序来创建交互播放的JET内容。使用[JetPlayer](#)类来播放和管理应用程序的JET内容。

播放JET内容- Playing JET

In this document

[Playing JET content](#)

Key classes

[JetPlayer](#)

Related Samples

[JetBoy](#)

See also

content

本节将向您展示如何编写，创建和播放JET内容。描述JET内容和说明如何使用JetCreator的编辑工具的资料请参考[JetCreator User Manual](#)。在在Windows,OS X操作系统,Linux平台上都可以使用该编辑工具(Linux不支持试听导入资产,比如Windows和OS X操作系统版本)。下面演示如何建立从存储在SD卡上的a .jet文件播放JET。代码如下：

```
JetPlayer jetPlayer = JetPlayer.getJetPlayer();
jetPlayer.loadJetFile( "/sdcard/level1.jet" );
byte segmentId = 0;

// queue segment 5, repeat once, use General MIDI, transpose by -1
// octave
jetPlayer.queueJetSegment( 5, -1, 1, -1, 0, segmentId++ );
// queue segment 2
jetPlayer.queueJetSegment( 2, -1, 0, 0, 0, segmentId++ );

jetPlayer.play();
```

SDK有一个示例应用程序,JetBoy - 展示了如何使用[JetPlayer](#)在游戏中创建交互式音乐配乐。它还演示了如何使用JET事件来同步音乐和游戏逻辑。这个应用程序位于[JetBoy](#)中。

[← 返回 Multimedia and Camera](#)

[↑ 回到顶部](#)

来自“[index.php?title=JetPlayer&oldid=3033](#)”



[JetCreator User Manual](#)

[Android Supported Media Formats](#)

[Data Storage](#)

[MediaPlayer](#)

Camera

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

[← 返回 Multimedia and Camera](#)

目录

[[隐藏](#)]

[1 照相机 - Camera](#)

[2 注意事项-Considerations](#)

[3 基础知识-The Basics](#)

[4 清单声明-Manifest Declarations](#)

照相机 - Camera

Android框架可以有效支持设备上的各种照相机和相机功能。允许您在应用程序中捕获照片和视频。本文档讨论一个核心，拍照和视频捕获的简单方法，和一个为用户创建自定义相机体验的高级方法。

注意事项-Considerations

在Android设备上的应用程序使用相机之前，应该思考应用程序打算如何利用这个硬件功能的几个问题。

- 相机需求 - 在应用程序中使用相机功能是否很重要，你不希望在设备上安装的应用程序没有照相机功能？如果是这样的话，你应该[在清单中声明相机需求](#)。
- 快拍或自定义相机 - 您的应用程序将如何使用相机？你只是对快速抢拍照

片或者视频片段感兴趣，还是，您的应用程序将提供一种新方式使用相机吗？要快速抓拍或得到视频片段，考虑[使用现有的相机应用程序](#)。为开发一个定制的拍照功能，查看[Building a Camera App](#)部分。

- 存储 - 图像或视频应用程序只对应用程序或共享可见，所以其他应用程序如画廊或其他媒体和社交应用程序可以使用它们？你希望即使应用程序卸载了，照片和视频也是可用的？查看[Saving Media Files](#)部分来了解如何实现这些设置。

基础知识-The Basics

Android框架支持通过[Camera API](#)或[camera Intent](#)来捕获图像和视频。以下是相关的类：

[Camera](#)

这个类是用于控制设备的摄像头的主要的API。构建摄像机应用程序时，这个类可以用来拍摄照片或视频。

[SurfaceView](#)

这个类用来给用户提供照相机预览功能。

[MediaRecorder](#)

这个类用来从相机录制视频。

[Intent](#)

一个 `MediaStore.ACTION_IMAGE_CAPTURE` 或 `MediaStore.ACTION_VIDEO_CAPTURE` 意图的动作类型，通过使用它我们可以不直接使用[Camera](#)对象来捕获的图像或视频。

清单声明-Manifest Declarations

用Camera API开发应用程序开始前，你应该确保有适当的声明允许使用照相机硬件和其他相关的特性。

- Camera许可 - 您的应用程序必须请求使用相机设备许可。代码如下：

```
<uses-permission android:name="android.permission.CAMERA" />
```

注意:如果您是通过意图使用照相机,您的应用程序不需要请求这个许可。

- Camera功能 - 您的应用程序还必须声明使用照相机功能,例如:

```
<uses-feature android:name="android.hardware.camera" />
```

Camera功能的功能列表,请查看[Features Reference](#)清单。

添加Camera功能到你的manifest导致谷歌阻止应用程序被安装到设备,应用程序不能包括一个摄像头或不支持您指定的相机功能。需要了解更多关于通过Google Play使用feature-based filtering的信息,请查看[Google Play and Feature-Based Filtering](#)。

如果正确操作您的应用程序可以使用camera或是camera功能,但不需要它,你应该在manifest中指定android:required属性,并将它设置为false:

```
<uses-feature android:name="android.hardware.camera"
    android:required="false" />
```

- 存储许可 - 如果您的应用程序保存图像或视频到设备的外部存储器(SD卡),您还必须在manifest中指定如下:

```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- 录音许可 - 记录音频与视频捕获,必须请求音频捕捉的许可。

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- 位置许可 - 如果您的应用程序需要标记图像与GPS locationinformation,你必须请求位置许可:

```
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />
```

关于获取用户位置的更多资料,请进入[Location Strategies](#)

来自[index.php?title=Camera&oldid=13886](#)



Location and Sensors

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

定位和传感器

在设备上使用传感器能为你的应用添加丰富的定位和运动的能力，从GPS和网络定位到加速度计，陀螺仪，温度传感器，气压传感器，等等。

[位置和地图](#)

博客文章

一个屏幕得到另一种

然而，有一个新的缺点：最近，一些设备已经发布（看这里）运行在Android屏幕上的自然风景会根据他们的方向而改变。那就是，当握住在默认的位置，屏幕比他们高处时更加宽，这里介绍了一些相当精细的问题，一些我们已经注意到在一些应用中引起的问题。

深入查看位置

我已经写了一个开放源代码的引用应用程序，这个程序包含所有的提示、技巧和我所知道的，减少打开一个应用程序，并看到附近场地的最新列表，以及在一个合理的水平之间提供脱机支持的时间间隔的秘籍。

练习

使你的应用有定位能力

这个类能帮助你怎么来使是你的Android应用程序能结合位置服务。你将会学到一些方法来接受位置更新和相关的最佳做法。

来自“[index.php?title=Location_and_Sensors&oldid=7553](#)”



Location and Maps

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

快速查看

anroid提供了一个位置框架让你的应用能够检测设备的时时方位

google地图提供的外部类能够让你显示地图和管理地图数据

主题

[获取用户地址](#)

也可查看

[google地图外部类>>](#)

址：<http://developer.android.com/guide/topics/location/index.html>

翻译：承影

更新：2012.06.09

-

目录

[[隐藏](#)]

[1 地图和地理位置-Location and Maps](#)

[2 位置服务 Location Services](#)

[3 谷歌地图安卓API-Google Maps Android API](#)

-

地图和地理位置-Location and Maps

地理位置和基本的地图应用仅仅是使用于手机设备用户。你有能力根据地理位置包和google 扩展的地图类库用一些android的类来建立你的应用。下面的内容将描述一些细节。

位置服务 Location Services

通过`android.location`的类库， android让你的应用可以应用地理位置的相关服务。这个主要的位置框架组件是[LocationManager system service](#)， 他提供给你APIs去锚定你和你的设备所在的位置（如果可利用的话）。和其他的系统服务一样， 你不能直接实例化一个LocationManager对象， 当然啦， 你需要从系统调用`getSystemService(Context.LOCATION_SERVICE)`来实例化。该方法将返回一个句柄,一个新的LocationManager 实例。一旦您的应用程序有一个LocationManager, 您的应用程序就可以做三件事：

- 查询所有的LocationProviders的列表,得到最近已知位置坐标。
- 注册/注销定期更新用户当前的位置从一个位置提供者(通过标准或指定名称)。
- 如果设备是在一个给定的距离(指定半径米)给定的纬度/经度， 注册和注销的一个Intent。

更多信息,请阅读指南,以获取用户位置。

谷歌地图安卓API-Google Maps Android API

使用谷歌地图安卓API， 可以基于谷歌地图数据来添加地图。API自动处理谷歌地图服务器， 数据下载， 地图显示和触摸手势。同时可以使用API调用 来添加标记， 多边形和覆盖，并且更改特定地图区域的用户视图。

谷歌地图安卓API的关键类是MapView。MapView显示了从谷歌地图服务获取数据的地图。当MapView获得焦点时， 它将捕捉 按键和触摸手势来自动

平移和缩放地图，包括为额外地图tiles处理网络请求。它也提供所有控制地图的必需UI元素。应用程序可以使用 **MapView**类方法来编程控制地图，并绘制地图上方叠加层。

安卓平台不包含谷歌地图安卓**API**，但是可以在安卓2.2或者更高版本的设备上通过谷歌播放服务使用。

要把谷歌地图融入应用程序，您需要为安卓**SDK**安装谷歌播放服务库。欲寻求更多细节，请阅读谷歌播放服务。

来自“[index.php?title=Location_and_Maps&oldid=13833](#)”



Location Strategies

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

[[隐藏](#)]

- [1 定位策略](#)
- [2 确定用户位置的挑战](#)
- [3 位置更新请求](#)
- [4 获取用户权限](#)
- [5 定义最优化模型](#)
- [6 常见的应用案例](#)
- [7 提供模拟位置](#)

定位策略

知晓用户所在的位置可以使应用更加的智能并且能够给用户传递更好的信息。当开发一款基于位置服务的android应用时，可以利用GPS和android的“Network Location Provider”（网络位置内容提供者组件）。尽管GPS是十分精确的，但是它的缺点也十分的令人恼火，比如只在户外工作、易耗费电池的电量以及返回用户位置缓慢等。而android的“Network Location Provider”使用移动信号塔和Wi-Fi信号来定位用户的位置。这种方式可以在室内、户外提供位置信息并且反应迅速耗电更少。在应用中可以同时使用GPS和“Network Location Provider”来获取用户的位置信息，也可以仅仅使用其中的一个。

确定用户位置的挑战

从一个移动设备上获取用户的位置是十分复杂的。读取位置出错（读取无响应）或者不准确可能有很多原因，下面就是列举一些其中的原因：

- 多种位置来源

GPS, Cell_ID和Wi_Fi都可以提供用户位置的线索，决定使用和信任哪种位置来源需要在精确度、速度和电池效率上做谨慎的权衡。

- 用户经常移动位置

因为用户的位置经常变化，所以必须常常考虑因用户移动而重新计算用户的位置。

- 不同的精度

从不同的位置来源计算出的位置信息可能不符合他们的精确度。一种位置源10秒钟前获取的位置可能比从其他位置源获取的当前位置更加准确。也可能一种位置源10秒前的位置信息和当前的位置信息一样。

基于这些存在的问题，获取用户真实的位置信息就变得相对困难。本文档提供一些信息来帮助你面对“如何获取更真实可靠的位置信息”的挑战。除此之外，还提供了一些思路和经验，来帮助你在应用程序中给用户提供准确，反应灵敏的地理位置。

位置更新请求

在解决上述的有关位置错误之前，首先简单的介绍下怎么在Android设备上获取用户位置。

在Android中，获取用户的位置是靠一系列的回调函数来完成的。可以通过调用LocationManager（位置管理器）的 requestLocationUpdates()方法来更新用户的位置，但此方法必须要传递一个 LocationListener实例作为参数。 LocationListener必须实现一些回调函数，这些回调函数会在用户位置变化和服务状态发生变化时被位置管理器调用。

举例，下面的代码就演示了怎么定义一个 LocationListener来实现位置更新的监听。

```
// 获取系统位置服务的引用
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

// 定义对位置变化的监听函数
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
```

```

    // Called when a new location is found by the network location
    provider.
    makeUseOfNewLocation( location );
}

public void onStatusChanged( String provider, int status, Bundle
extras ) {}

public void onProviderEnabled( String provider ) {}

public void onProviderDisabled( String provider ) {}
};

// 向位置服务注册监听函数来接受位置更新
locationManager.requestLocationUpdates( LocationManager.NETWORK_PROVIDER
, 0 , 0 , locationListener );

```

`requestLocationUpdates()`的第一个参数是**Location Provider**（位置提供者）的使用类型。（本例中使用的是基于手机信号塔和Wi_FI基站的网络位置提供者）。可以通过第二个和第三个参数来控制监听 函数接受更新的频率。具体点讲即第二个参数是通知之间的最短时间间隔，第三个是通知之间最小的距离变化。将两个的值设置为零可以尽可能频繁的获取位置的通知。最后一个参数是 接受位置更新回调函数**LocationListener**。

为了使用**GPS**提供者来获取位置的更新，可以将**NETWORK_PROVIDER**.更改为**GPS_PROVIDER**。也可以通过两次调用 `requestLocationUpdates()`（一次 `NETWORK_PROVIDER`，一次**GPS_PROVIDER**。）来同时使用**GPS**和**Network Location** 内容提供者获取位置更新。

获取用户权限

为了能够接受来自 `NETWORK_PROVIDER`或者**GPS_PROVIDER**的位置更新，必须在**Android** 的**mainifest**的文件中声明**ACCESS_COARSE_LOCATION** 或者**ACCESS_FINE_LOCATION**的用户权限。比如：

```

<manifest ... >
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>

```

没有这些权限，应用会在请求位置更新时出错。

NETWORK_PROVIDER GPS_PROVIDER

注意：如果同时使用 和 ，那么只需要获取 `ACCESS_FINE_LOCATION` 的权限，因为`ACCESS_FINE_LOCATION` 包含了两种提供者所需要的权限。
`(ACCESS_COARSE_LOCATION)`的权限仅仅包括`NETWORK_PROVIDER`所需的权限。)

定义最优化模型

基于位置的应用可谓是数不胜数，但是由于不能提供最佳的准确度，用户位置的移动，多种方法获取用户位置和尽可能减少耗电量等原因使得获取用户位置变得较为复杂。在减少耗电量的同时去解决获取极佳用户位置的众多难题，必须定义一个长效模型来指定应用怎么获取用户的位置。当开始或者停止监听更新以及使用缓存的位置数据时，此模型会被使用。

- 获取用户位置的流程

以下就是获取用户位置的典型流程：

1. 启动应用
2. 一段时间后，开始监听location provider来获取位置信息
3. 通过去除不够准确的位置更新来保持以最佳状态去获取位置信息
4. 停止监听获取位置信息。
5. 采用最新最好的位置。

图像-1通过使用时间线展示了此模型。这个时间线体现了应用监听用户位置更新的各个时间段和各个时间段发生的事件。



图像-1获取用户位置更新的流程时间线

当向应用添加基于位置的服务时，决定哪一种位置更新被接收期间，需要做出的许多决定适用此窗口模型。

- 决定什么时候开始监听更新

也许你想要应用一启动就开始监听用户位置的更新，也许你想仅当用户触发特定的条件时才启动监听。但是要清楚的意识到两点，第一点是长时间的监听位置更新可能导致耗电量急剧上升，第二点是短时间的监听又可能使得用户位置获取的准确度不够。

如上所示，可以通过调用`requestLocationUpdates()`开始监听更新：

```
LocationProvider locationProvider =
LocationManager.NETWORK_PROVIDER;
//或者, 使用LocationManager.GPS_PROVIDER
// LocationProvider locationProvider = LocationManager.GPS_PROVIDER;

locationManager.requestLocationUpdates( locationProvider, 0, 0,
locationListener );
```

- 通过最后可知位置快速修正

位置监听函数接收第一次位置更新所花费的时间可能让用户难以忍受。直到位置监听函数接收到一个更精确的位置信息前，应该暂时使用用户缓存的位置，缓存位置可以通过调用`getLastKnownLocation(String)`函数来获取。

```
LocationProvider locationProvider =
LocationManager.NETWORK_PROVIDER;
//或者使用 LocationManager.GPS_PROVIDER
Location lastKnownLocation =
locationManager.getLastKnownLocation( locationProvider );
```

- 决定什么时候停止监听更新

决定什么时候最新的更新不在有用，这个逻辑因应用不同而可能非常简单，也可能十分复杂。在获取位置信息和使用位置信息之间加入一点时间的延迟可能提高位置获取的准确度。始终清楚的记得，持续监听会消耗大量的电量，所以只要获取了所需的信息，应该通过调用`removeUpdates(PendingIntent)`停止监听更新。

```
//移除先前添加的监听
locationManager.removeUpdates( locationListener )
```

- 保持最佳的位置获取状态

你可能认为最新获取的位置信息就是最精确的。但是，由于位置修正的精确度经常变化，最新获取到的位置信息并不一定都是最准确的。你应该添加基于不同标准的位置获取逻辑。这些标准也根据具体的应用和现场测试

实例不同而变化。

下面是验证位置修正可以采用的步骤：

- 检查是否最近得到的位置信息明显比以前的要新。
- 检查位置精度是好于还是差于之前的。
- 检查最新的位置信息是来自于哪一个Provider(提供者)，并且判断是否这个位置信息相比之前的更加准确可靠。

下面是一个符合上述逻辑的例子：

```

private static final int TWO_MINUTES = 1000 * 60 * 2;

/** 判断哪一种位置读取方式比当前的位置修复更加的准确
 * @param location 新位置
 * @param currentBestLocation 当前的位置，此位置需要和新位置进行比较
 */
protected boolean isBetterLocation(Location location, Location
currentBestLocation) {
    if (currentBestLocation == null) {
        // A new location is always better than no location
        return true;
    }

    //检查最新的位置是比较新还是比较旧
    long timeDelta = location.getTime() -
currentBestLocation.getTime();
    boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
    boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
    boolean isNewer = timeDelta > 0;

    //如果当前的位置信息来源于两分钟前，使用最新位置，
    //因为用户可能移动了
    if (isSignificantlyNewer) {
        return true;
    } else if (isSignificantlyOlder) {
        return false;
    }

    //检查最新的位置信息是更加的准确还是不准确
    int accuracyDelta = (int) (location.getAccuracy() -
currentBestLocation.getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    //检查旧的位置和新的位置是否来自同一个provider
    boolean isFromSameProvider =
isSameProvider(location.getProvider(),
    currentBestLocation.getProvider());

    //结合及时性和精确度，决定位置信息的质量
    if (isMoreAccurate) {

```

```

        return true;
    } else if (isNewer && !isLessAccurate) {
        return true;
    } else if (isNewer && !isSignificantlyLessAccurate &&
isFromSameProvider) {
        return true;
    }
    return false;
}

/** 检查两个提供者是否是同一个*/
private boolean isSameProvider(String provider1, String provider2) {
    if (provider1 == null) {
        return provider2 == null;
    }
    return provider1.equals(provider2);
}

```

- 调整模型来保存电量和数据交换

当测试应用程序的时候，你可能会在模型是要提供更佳的位置信息还是更佳的效率之间做出选择调整。下面就是在两者之间找到平衡点的一些建议。

- 减少窗口的大小

在一个较小的窗口下监听位置更新，意味着和GPS或者Network Location 服务进行更少的交互，这样就可以保存电池电量。但是这样会使得可选位置变少，从而导致获取最佳位置信息变得困难。

- 减少**location providers**(位置提供者)的更新频率

在窗口中减少更新出现的频率也可以提高电池使用效率，但是这样会牺牲精确度。两者之间的权衡要依赖于具体的的实际应用。可以通过增加requestLocationUpdates()函数的第二个和第三个参数的值来减少更新的频率。

- 仅支持一种提供者

根据应用程序的使用场景和对精度的要求，也许只需要在Network Location Provider 和 GPS之间选择一种提供者，而不是两者都需要。只和其中的一种服务进行交互可以大大的减少耗电的可能性。

常见的应用案例

在应用中使用用户位置的原因可能多种多样。下面就要介绍一些使用用户位置的应用场景，这些场景可以使应用变得丰富起来。为了同时达到精确度和电池使用效率的最佳状态，每一个应用场景在何时开始和停止监听用户更新等问题上做出了良好的描述。

- 使用位置标记用户创建的内容

你也许正在开发一个在用户创建的内容中嵌入位置信息的应用。这个应用可以让用户分享自己的本地经验，评价一个餐馆或者记录一些在当前位置发生的事情。在图像-2中就展示了通过位置服务进行交互的模型。



图像-2

这条时间线符合图像-1中如何获取用户位置的流程。为了获取最精确的用户位置，可以在用户开始创建内容时，或者在应用启动时就开始监听用户的位置的更新。当用户创建完成或者保存内容时停止用户位置更新的监听。你也许需要考虑一次典型的创建内容需要耗费多少时间并判断这个时间段是否能够有效的计算出用户位置。

- 帮助用户决定要去哪里

你可能正在开发一个引导用户去哪里的应用。例如你想提供附近位置的餐馆，商店和娱乐场所的列表，并且这个列表会随着用户位置的变化而变化。

若要实现此功能，你需要：

- 当收到更新更准确的位置信息时，要对已有的推荐列表重新排列。
- 如果推荐列表不在变化，则可以停止对位置更新的监听

这种情况的模型在图像-3中表现出来。



图像-3

提供模拟位置

在开发应用的过程中，当然需要对获取用户位置的模型进行效率测试。最简单的测试就是使用**android**真机设备。但是如果没有一个真正的物理设备，也可以使用**Android**虚拟机的虚拟位置进行基于用户位置的测试。向应用提供模拟位置数据的方法主要有三种：Eclipse，DDMS或者模拟器控制台的“geo”命令行。

注意：提供模拟位置数据是使用的**GPS**的数据类型，所以必须使用**GPS_PROVIDER**来获取位置更新，否则模拟数据无法工作。

- 使用**Eclipse**

选择 **Window > Show View > Other > Emulator Control**。

在模拟器控制面板上，进入位置控制（Location Controls）下输入**GPS**坐标，**GPX**文件中是路径回放，**KML**文件中是多个位置的记录。（请确定在设备面板下已经有个设备被选择，查看**Window > Show View > Other > Devices**）

- 使用**DDMS**

使用**DDMS**工具，可以使用多种方法模拟位置数据。

- 向设备手动发送独立的经纬度
- 使用**GPX**文件向设备发送的一系列路径。
- 使用**KML**文件向设备发送独立的一序列化的路径位置

查看[使用DDMS](#)，获取更多的信息。

- 使用模拟器控制台的“geo”命令行

使用命令行发送模拟位置数据：

1. 在**Android**模拟器上装载应用，并打开**sdk**下的**/tools**目录下打开设备终端的控制台。
2. 连接到模拟器控制台

```
telnet localhost <console-port>
```

3. 发送位置数据

o **geo fix** 发送固定的geo位置。这个命令接收十进制的经度和纬度，和一个可选的海拔（单位m）

```
geo fix -121.45356 46.51119 4392
```

o **geo nmea** 发送一个NMEA 0183句子。

```
geo nmea  
$GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
```

怎么连接到模拟器控制台，请查看[使用模拟器控制台](#)

来自“[index.php?title=Location_Strategies&oldid=9406](#)”



Sensors Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：

http://developer.android.com/guide/topics/sensors/sensors_overview.html

翻译者：[AlbertLi](#)

最后更新：2012-06-13

[sensor:](#)

目录

[\[隐藏\]](#)

[1 感应器概述](#)

[2 感应器介绍](#)

- [2.1 感应器框架](#)
- [2.2 感应器可用性](#)
- [2.3 识别感应器和感应器功能](#)
- [2.4 监测感应器事件](#)
- [2.5 处理不同的传感器配置](#)
 - [2.5.1 运行时检测传感器](#)
 - [2.5.2 使用谷歌播放器来针对特定传感器配置](#)
- [2.6 传感器坐标系](#)
- [2.7 访问和使用传感器的最佳实践](#)
 - [2.7.1 注销传感器监听器](#)
 - [2.7.2 不要在模拟器上测试代码](#)
 - [2.7.3 不要堵塞onSensorChanged\(\)方法](#)

感应器概述

大多数安卓设备都内置了传感器，用来测量移动，方向和各种环境条件。这些传感器能够提供精度高且准确的原始数据。如果要监视三维设备运动或定位，或者监视设备周围的环境变化，那么传感器会极其有用。例如游戏可以追踪设备重力感应器的读数来推断复杂的用户手势和动作，比如倾斜，摇晃，旋转或摆动。同样地，天气应用程序可以使用设备温度传感器和湿度传感器来计算并报告dewpoint，旅行应用程序可以使用地磁传感器和加速计来报告罗盘方位。

安卓平台支持以下三类传感器：

- 运动传感器

这些传感器测量三个轴的加速度和旋转力。这一类别包括加速度计，重力感应器，陀螺仪和旋转矢量传感器。

- 环境传感器

这些传感器测量各种环境参数，例如环境空气温度和压力，照明和湿度。此类别包括了气压计，光度计和温度计。

- 位置传感器

这些传感器测量了设备的物理位置。此类别包括方向传感器和磁力计。

感应器框架提供了用以帮助你执行各种与感应器相关任务的一些类和接口。例如，你可使用感应器框架做下

快速预览

- 了解Android支持的传感器和Android的传感器框架。
- 找出列出感应器的方法，确定感应器的功能，以及监测感应器数据。
- 了解访问和使用传感器的最佳方法。

本文件描述

[感应器简介](#)

[识别感应器和感应器功能](#)

[监测感应器事件](#)

[处理不同感应器配置](#)

[感应器坐标系](#)

[访问和使用感应器的最佳实践](#)

关键类和接口

[**Sensor**](#)

[**SensorEvent**](#)

[**SensorManager**](#)

[**SensorEventListener**](#)

相关例子

[加速度计演示](#)

[API 示范 \(OS - \(向量旋转\)RotationVectorDemo\)](#)

[API 示范 \(OS - 感应器\)](#)

参见

[感应器](#)

[动态感应器](#)

列事情：

[位置感应器](#)

[环境感应器](#)

- 确定设备的可用感应器。
- 确定独立感应器的兼容性，如它的最大范围，厂家，电源需求，解析度。
- 请求感应器的原始数据，定义你请求感应器数据的最小量。
- 注册和注销监视感应器变化的感应器监听事件。

本主题提供一个可用于Android平台的感应器预览，他还提供了感应器框架的介绍

感应器介绍

Android感应器框架允许您访问许多类型的感应器。其中的一些感应器是基于硬件的，有些是基于软件的。基于硬件的感应器是一个手机或平板电脑设备内建的物理组件。他们通过直接测量特定的环境属性，如加速、磁场强度、或角度变化获取数据。基于软件的感应器不是物理设备，尽管他们模仿基于硬件的感应器。基于软件的感应器从一个或多个基于硬件的感应器获取数据，有时被称为虚拟感应器或合成感应器。线性加速感应器和重力感应器是基于软件的感应器的例子。表1总结了Android平台所支持的感应器。

少数安卓设备带有所有类型的感应器。例如，大多数手机设备和平板带有电脑加速器和磁力计，但是部分设备带有温度计或晴雨表。同样，一个设备可以有不止一个给定类型的感应器。例如，一个设备可以有两个重力感应器，每一个有不同的范围。

表一. Android平台所支持的感应器类型.

感应器	类型	说明	一般用途
TYPE_ACCELEROMETER	硬	测量一个设备不包括重力作用的所有三轴(x,y,和z)m / s ² 加速度。	沿着一条单独的轴监控加速

			度。
<u>TYPE_AMBIENT_TEMPERATURE</u>	硬件	测试周围的室温(摄氏度(°C))。请参阅下面的注释。	监测空气温度。
<u>TYPE_GRAVITY</u>	软件或硬件	测量一个设备所有三轴(x,y,和z)m / s2重力加速度。	运动检测(晃动,倾斜,等等)。
<u>TYPE_GYROSCOPE</u>	硬件	测量一个设备三个物理轴的(x,y,和z)旋转速率。	旋转检测(旋转,转动)。
<u>TYPE_LIGHT</u>	硬件	测量光强 (照明) 单位 lx。	屏幕亮度控制。
<u>TYPE_LINEAR_ACCELERATION</u>	软件或硬件	测量一个设备不包括重力作用的所有三轴(x,y,和z)m / s2加速度。	沿着一条单独的轴监控加速度。
			指南

[TYPE_MAGNETIC_FIELD](#)

硬件 测量三个物理轴(x,y,和z)的磁强度
单位 uT。 针。

[TYPE_ORIENTATION](#)

软件 测量装置的三个物理轴(x,y,z)旋转度。在API级别3可同时测量重力感应器和磁场感应器的倾斜矩阵和旋转矩阵，并调用getRotationMatrix()方法获取。 确定设备位置。

[TYPE_PRESSURE](#)

硬件 测量环境空气压力 单位:hPa
mbar。 监测气压变化。

[TYPE_PROXIMITY](#)

硬件 测量一个物体靠近屏幕的距离(单位:cm)。通常用于检测手机离人耳朵的距离 通话时电话离人的距离

[TYPE_RELATIVE_HUMIDITY](#)

硬件 测量环境相对湿度。 监测结露点、绝对和相对湿度。

沿着一条

[TYPE_ROTATION_VECTOR](#)

软件或硬件	测量一个设备不包括重力作用的所有三轴(x,y,和z)m / s2加速度。	单独的轴监控加速度。
-------	--------------------------------------	------------

[TYPE_TEMPERATURE](#)

硬件	测量设备的摄氏温度(°C).在API 14中这个值可由TYPE_AMBIENT_TEMPERATURE替换。	监测温度。
----	--	-------

感应器框架

通过使用Android感应器框架可以访问这些感应器并获取原始感应器数据，感应器框架是 `android.hardware` 包的一部分，包括以下类和接口：

SensorManager 你可以使用这个类创建感应器服务的一个实例。这个类提供了各种用于访问、列出、登记、注销感应器的事件监听器方法和获得方向信息的方法。这个类还提供了一些报告感应器的精度，集数据采集速率，和校准感应器的常量，

Sensor 你可以使用这个类来创建一个特定的感应器实例。这个类提供了各种方法，让你确定感应器的能力。

SensorEvent 系统使用这个类创建一个带一个感应器事件信息的感应器事件对象。感应器事件对象包括以下信息：感应器原始数据，生成的事件，数据的准确性，事件的时间戳类型。

SensorEventListener 您可以使用此接口来创建两个接收（感应器事件）通知的

回调方法：感应器值的变化事件，感应器的精度变化事件。

在一个典型应用中，使用这些与感应器相关的API来完成两项基本任务：

- 识别感应器和他的能力

如果你的应用程序依赖特定类型或能力的感应器，在运行时识别感应器和感应器的能力是很有用的，比如，你可能想识别设备上所有的感应器，禁用那些不提供响应应用程序功能的感应器。同样，你可能要确定一个给定类型的感应器所有的能力，这样，你可以为您的应用程序选择最佳性能的感应器实现

- 监测感应器事件

监测感应器事件：如何获取感应器原始数据。当检测到一个感应器参数变化时，产生一个感应器事件。一个感情器事件带有四条信息：触发感应器的事件名称，事件时间戳，该事件的准确性，触发该事件的原始感应器数

感应器可用性

感应器可用性因设备而异，也可因版本而异。这是由于**ANDROID**感应器行为依赖于发布台实现的缘故。大部分感应器，在**Android 1.5 (API Level 3)** 带有，但有一部分的在**Android 2.3 (API level 9)** 中才有。同样，还有一些感应器在**Android 2.3 (API level 9)** 和**Android 4.0 (API level 14)** 中。已经有两个传感器已被弃用，取而代之的是新的，更好的感应器。

表2总结了不同平台基础上每个感应器的可用性。列表中只有四个平台，因为只有这些平台的感应器可用性有变化。列表中为废弃的传感器仍然可以在更高版本的平台（设备上要提供感应器），这符合**Android**的向前兼容性规则。

表2。感应器的可用性平台列表。

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a ¹	n/a ¹
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes ²	Yes ²	Yes ²	Yes
TYPE_PRESSURE	Yes	n/a ¹	n/a ¹	
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes ²	Yes	Yes	Yes

- 1 感应器在 Android 1.5 (API Level 3) 中加入, 但只有从Android 2.3 (API Level 9) 版本才开始能够使用.
- 2 感应器可用, 但已经废弃.

识别感应器和感应器功能

Android的感应器框架提供了一些方法, 使您可以轻松地在运行时确定设备带有的感应器。该API还提供了其他一些方法, 让你确定每个感应器的能力, 如它的最大值范围, 它的分辨率, 其功率需求。

要确定设备上的感应器的功能, 你首先需要得到一个感应器服务的引用. 要做

到这一点，你要通过调用`getSystemService()`方法并传递`SENSOR_SERVICE`参数给它来创建一个`SensorManager`实例，。例如：

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
```

接下来，你可以通过调用`getSensorList()`方法和使用的`TYPE_ALL`常量取得设备上的每个感应器。例如：

```
List<Sensor> deviceSensors =
mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

如果要列出一个给定类型的感应器，可以使用另一个常量代替`TYPE_ALL`如`TYPE_GYROSCOPE`, `TYPE_LINEAR_ACCELERATION`, 或`TYPE_GRAVITY`。

您也可以传递一个特定的感应器类型常量给`getDefaultSensor()`方法来确定设备上是否存在特定感应器。如果设备有多个给定类型的感应器，其中只一必须被指定为默认感应器。如果一个给定类型的默认感应器不存在，该方法调用返回`null`，这意味着设备没有这种类型的感应器。例如，下面的代码检查设备上是否有一个磁力感应器：

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
    // 成功 | 存在磁力感应器.
}
else {
    // 失败! 不存在.
}
```

- Note: Android 并不要求设备制造商内建任何特有的感应器，设备感应器配置可以多样化。

除了列出设备上的感应器，还可使用的感应器类的公共方法来确定各个感应器的功能和属性。对于你希望在基于可用感应器或感应器功能的设备上，应用能够有不同的表现，这将很有用。例如，您可以使用的`getResolution()`和`getMaximumRange()`方法取得一个感应器的分辨率和最大测量范围。你也可以使用的`getPower()`方法来取得一个感应器的电源要求。

对于不同制造商的感应器或不同版本的感应器 如果你想优化您的应用程序适应他们，有两个公共方法是特别有用。例如，如果您的应用程序需要监视，如倾斜和晃动用户的手势，您可以创建一个用于优化特定厂家重力感应器的数据过滤规则集，和另一组用于优化没有重力感应器（只有一个加速感应器）的数据过滤规则集，。下面的代码示例通过使用的 `getVendor()` 和 `getVersion()` 方法来做到这一点。在此示例中，我们正在寻找一个谷歌产的重力感应器，版本号为3。如果是在设备上不存在特定的感应器，我们尝试使用加速度感应器。

```

private SensorManager mSensorManager;
private Sensor mSensor;

...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
    List<Sensor> gravSensors =
mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
    for(int i=0; i<gravSensors.size(); i++) {
        if ((gravSensors.get(i).getVendor().contains("Google Inc."))
&& (gravSensors.get(i).getVersion() == 3)){
            // 使用 版本号为3 重力感应器
            mSensor = gravSensors.get(i);
        }
    }
} else{
    // 使用加速度感应
    if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
        mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    } else{
        // 对不起你的设备没有加速感应器
        // 你无法进行游戏
    }
}

```

另一个有用的方法是 `getMinDelay()` 方法，它返回一个传感器用来检测数据的最短时间间隔（微秒）。如果 `getMinDelay()` 方法返回一个非零值说明传感器是一个定期检测数据流的流传感器。在Android 2.3 (API等级9) 有介绍。如果一个传感器时调用 `getMinDelay()` 方法返回零，这意味着传感器是不是流的传感器，因为只有在它参考值刚好 被检测到发送变化时，才报告数据。

`getMinDelay()` 方法十分有用，因为他可让你决定请求传感器的最大速率，如果应用程序中的某些功能需要高数据采集率或使用流式传感器，使用此方法决定传感器是否符合要求，然后相应地启用或禁用应用程序中的相关功能。

注意：传感器的最大数据采集率不一定是传感器框架提供给应用的传感数据采集率。传感器框架通过传感器事件报告数据，并且一些因素影响应用接受到传感器事件的数据采集率，有关更多信息，请参阅监视传感器事件。

监测感应器事件

检测感应器需要实现由SensorEventListener 接口开放的两个回调方法：`onAccuracyChanged()` 和 `onSensorChanged()`，当下列事情发生时，Android系统调用这些方法。

- 传感器精度的变化。

在这种情况下，系统调用`onAccuracyChanged`方法，为您提供一个更新的传感器精度传感器对象的引用。精度如下四个状态常量之一
`SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`,
`SENSOR_STATUS_ACCURACY_HIGH`, or `SENSOR_STATUS_UNRELIABLE`.

- 传感器报告新数值

这种情况下系统会调用`onSensorChanged()`方法，为您提供SensorEvent 对象。SensorEvent 对象包含了新传感器的信息，包括：数据准确性，生成数据的传感器，生成数据的时间戳和传感器记录的新数据。

下列代码显示了如何使用`onSensorChanged()`方法来监视光传感器的数据。这个例子显示TextView原始传感器数据，main.xml文件定义了这一数据。

```
public class SensorActivity extends Activity implements
SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mLight;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
```

```

        mLight = mSensorManager.getDefaultSensor( Sensor.TYPE_LIGHT );
    }

@Override
public final void onAccuracyChanged( Sensor sensor, int accuracy ) {
    // Do something here if sensor accuracy changes.
}

@Override
public final void onSensorChanged( SensorEvent event ) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[ 0 ];
    // Do something with this sensor value.
}

@Override
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener( this, mLight,
SensorManager.SENSOR_DELAY_NORMAL );
}

@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener( this );
}
}

```

此示例中，当调用`registerListener()`时会指定默认数据延迟(`SENSOR_DELAY_NORMAL`)。数据延迟(或者采样率)控制传感器事件通过`onSensorChanged()`回调方法发送给应用程序的时间间隔。默认数据延迟适用于监视典型屏幕方位改变，并使用200,000微秒的延迟。您可以指定其他数据延迟，比如`SENSOR_DELAY_GAME`(20,000微秒延迟)，`SENSOR_DELAY_UI`(60,000微秒延迟)，或者`SENSOR_DELAY_FASTEST`(0微秒延迟)。对于安卓3.0(API级别11)，您也可以指定延迟作为绝对值(以微秒为单位)。

指定的延迟仅仅是建议延迟。安卓系统和其他应用程序可以进行更改。作为最佳实践，您可以指定最大延迟，因为系统通常使用比定义更小的延迟。使用较大的延迟会在处理器上施加较低负载，因此会消耗较少的电量。

没有公共方法可以确定传感器框架发送传感器事件的速度；然而你可以使用与各传感器事件相关的时间戳来计算采样率。一旦设置了采样率就不要进行修改。如果由于某种原因需要修改延迟，就必须注销并重新注册传感器监听器。

请注意此示例使用了`onResume()`和`onPause()`回调方法来注册和注销传感器事件监听器。如果不这样做可能会几小时内耗尽电池电量，因为某些传感器具有实质功率需求并且迅速耗尽电量。屏幕关闭时系统不会自动禁用传感器。

处理不同的传感器配置

安卓不会知道设备的标准传感器配置，这就意味着设备制造商的安卓供电设备可以容纳任何需要的传感器配置。因此装置可以包含广泛配置范围内的各种传感器。例如，Motorola Xoom拥有压力传感器，而Samsung Nexus S则没有。同样Xoom和Nexus S有陀螺仪，而HTC Nexus One则没有。如果应用程序依赖于传感器具体类型，就要确保传感器安装在应用程序可以正常运行的设备上。要确保给定的传感器存在设备上，您有两种选择：

- 运行时检查传感器，并根据需要启用或禁用应用程序功能。
- 使用谷歌播放器来定位特定传感器配置的设备。

各选项会分别在下面章节进行讨论。

运行时检测传感器

如果应用程序使用特定类型的传感器，您可以使用传感器框架来检测传感器，然后根据需要禁用或启用应用程序功能。例如导航应用程序可以使用温度传感器，压力传感器，GPS传感器和地磁传感器来显示温度，气压，位置和方位。如果设备没有压力传感器，可以使用传感器框架来检测压力传感器缺失，然后禁用应用程序UI界面显示压力的部分。例如下列代码检测设备上是否有压力传感器：

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null) {
    // Success! There's a pressure sensor.
}
else {
    // Failure! No pressure sensor.
}
```

使用谷歌播放器来针对特定传感器配置

如果要在谷歌播放发布新应用，可以使用清单文件中的<uses-feature>元素来筛选设备中的应用程序。<uses-feature>元素包含基于特定传感器来筛选应用程序的硬件描述。可以列出的传感器包括：加速计，气压计，指南针（地磁场），陀螺仪，光信号等 等。下列是示例清单条目：

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
```

```
        android:required="true" />
```

如果向应用程序清单添加此元素和描述符，只有设备安装减速计的用户才会在谷歌播放上看到应用。

传感器坐标系

一般情况传感器框架使用标准3轴坐标系来表示数值。对于大多数传感器，当设备在默认方向使用时坐标系定义为相对于设备屏幕。当设备在默认方位时，X轴水平并指向右侧，Y轴垂直并指向上方，Z轴指向屏幕外侧。系统屏幕背面为负向Z轴。以下传感器会使用这一坐标系：

- 加速度传感器
- 重力传感器
- 陀螺仪
- 线性加速度传感器
- 地磁传感器

这一坐标系最重要的一点是屏幕方向变化时坐标轴不会交换。也就是说，设备移动时传感器坐标系不会改变。这种行为与OpenGL坐标系行为相同。

如果应用程序与屏幕上显示的传感器数据一致，就需要使用getRotation()方法来确定屏幕旋转，然后使用remapCoordinateSystem()方法来映射传感器坐标到屏幕坐标。

访问和使用传感器的最佳实践

注销传感器监听器

当使用传感器或者传感器活动暂停时，请确保注销传感器监听。如果已经注册传感器侦听，并且活动暂停，传感器会继续采集数据并使用电池资源。下列代码显示了如何使用onPause()方法来注销监听器：

```
private SensorManager mSensorManager;  
...  
@Override  
protected void onPause() {  
    super.onPause();  
    mSensorManager.unregisterListener(this);  
}
```

不要在模拟器上测试代码

目前无法在模拟器测试传感器代码，因为模拟器不能模拟传感器。必须在物理设备上测试传感器代码。而传感器模拟器可以用来模拟传感器输出。

不要堵塞**onSensorChanged()**方法

传感器数据可以高速变化，这意味着系统会频繁调用**onSensorChanged(SensorEvent)** 方法。

来自 "[index.php?title=Sensors_Overview&oldid=13838](#)"



Motion Sensors

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

安卓平台提供了几种用来监控设备移动传感器。加速度计和陀螺仪传感器始终基于硬件，而重力，线性加速度和旋转矢量传感器则可以基于硬件或软件。例如某些设备上基于软件的传感器从加速度计和磁力计来获取数据，但是其他设备还可以使用陀螺仪来获取数据。大多数安卓供电设备带有加速度计，而且还会包含陀螺仪。基于软件的可用传感器更加可变，因为它们通常依赖于一个或多个硬件传感器来推导数据。

运动传感器用于监控设备运行，比如倾斜，摇晃，旋转或摇摆。运动通常反映了直接用户输入，同时也可以反映设备所处的物理环境。第一种情况下您在监控相对于设备框架参照或应用程序框架的运动；第二种情况您在监控相对全局框架引用的运动。运动传感器通常不用于监控设备位置，但是它们可以用于其他传感器，比如地磁传感器，来确定设备相对于全局框架的位置。

所有的运动传感器会返回各SensorEvent传感器数值的多维数组。例如单个传感器事件期间，加速计会返回三个坐标轴加速度的数据，陀螺仪返回三个坐标轴旋转速率。这些数据值以float数组与其他SensorEvent参数形式返回。表1总结了安卓平台上可用的运动传感器。

旋转矢量传感器和重力传感器是最经常使用的传感器，用于运动检测和监测。旋转矢量传感器功能多样，可以用来执行与运动相关的任务，例如检测手势，检测角度变化和监测相对定向改变。例如，如果正开发一个游戏，扩展现实应用程序，2维或3维罗盘，或者摄像头应用程序，那么旋转矢量传感器是理想的。大多数情况下，比起使用加速度计、地磁传感器或方向传感器，使用这些传感器是更好的选择。

目录

[\[隐藏\]](#)

- [1 安卓开源项目传感器](#)
- [2 使用加速度计](#)
- [3 使用重力感应](#)
- [4 使用陀螺仪](#)
- [5 使用未校准陀螺仪](#)
- [6 使用线性加速度计](#)

安卓开源项目传感器

安卓开源项目 (AOSP) 提供了三种基于软件的运动传感器：重力传感器，线性加速度传感器以及旋转矢量传感器。这些传感器在安卓4.0进行了更新，并使用设备的陀螺仪来改善稳定性和性能。如果想尝试这些传感器，您可以使用`getVendor()`和`getVersion()`方法来识别它们。通过供应商和版本号来识别这些传感器非常必要，因为安卓系统把这三种传感器定义为次要传感器。例如设备制造商提供了自己的重力传感器，那么AOSP重力感应器显示为次要重力感应器。这三种传感器都依靠陀螺仪：如果设备不具备陀螺仪，那么就不会显示和使用这些传感器。

使用加速度计

加速度传感器测量施加到设备的加速度，包括重力。下列代码显示了如果获取默认加速度传感器的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

从概念上说，使用下列关系可以通过测量施加到传感器本身的力量来确定施加到设备(Ad)的加速度：

$$Ad = - \sum F_s / \text{mass}$$

然而根据下列关系式，重力总会影响所测量的加速度：

$$Ad = -g - \sum F / \text{mass}$$

因此当设备位于桌子上时，加速计会读取 $g=9.81 \text{ m/s}^2$ 。类似地，当设备自由落体并因此以 9.81 m/s^2 加速向地面运动时，加速度计会读取 $g=0 \text{ m/s}^2$ 。相反，低通滤波器可以用来隔离重力作用。下面的例子显示了如何操作：

```
public void onSensorChanged(SensorEvent event) {
    // In this example, alpha is calculated as t / (t + dt),
    // where t is the low-pass filter's time-constant and
    // dt is the event delivery rate.

    final float alpha = 0.8;

    // Isolate the force of gravity with the low-pass filter.
    gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

    // Remove the gravity contribution with the high-pass filter.
    linear_acceleration[0] = event.values[0] - gravity[0];
    linear_acceleration[1] = event.values[1] - gravity[1];
    linear_acceleration[2] = event.values[2] - gravity[2];
}
```

加速度计使用标准的传感器坐标系。这就意味着当设备自然方向平放在桌子上时会应用下列条件：

- 如果在左侧推动设备（则它向右移动），X加速度为正值。
- 如果在底部推动设备（则它会向远处移动），Y加速度为正值。
- 如果以 $A \text{ m/s}^2$ 的加速度向天空加速，z加速度值为 $A+9.81$ ，这对应于设备加速度 $(+A \text{ m/s}^2)$ 减去重力 (-9.81 m/s^2) 。
- 固定设备会具有 $+9.81$ 的加速度，这对应于该设备的加速度 (0 m/s^2) 减去重力 (-9.81 m/s^2) 。

一般情况如果要监控设备运动，加速度计很有用处。几乎所有安卓供电的手机和平板都具有加速度计，且加速度计比其他运动传感器少消耗10倍功率。

使用重力感应

重力传感器提供了三维向量用来指示重力的方向和幅度。下列代码显示了如何获取默认重力感应器的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
```

使用陀螺仪

陀螺仪以弧度/秒来测量设备X,Y,Z轴的速率或旋转。下列代码显示了如何获取默认陀螺仪的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

传感器的坐标系与加速度传感器使用的坐标系相同。逆时针方向为旋转正方向；也就是说如果设备逆时针旋转的话，从设备X,Y,Z轴的正方向观测会判断为正方向旋转。

通常情况下，陀螺仪输出随时间积分来计算描述角度变化的旋转角。例如：

```
// Create a constant to convert nanoseconds to seconds.
private static final float NS2S = 1.0f / 1000000000.0f;
private final float[] deltaRotationVector = new float[4]();
private float timestamp;

public void onSensorChanged(SensorEvent event) {
    // This timestep's delta rotation to be multiplied by the current
    rotation
    // after computing it from the gyro sample data.
    if (timestamp != 0) {
        final float dT = (event.timestamp - timestamp) * NS2S;
        // Axis of the rotation sample, not normalized yet.
        float axisX = event.values[0];
        float axisY = event.values[1];
        float axisZ = event.values[2];
```

```

// Calculate the angular speed of the sample
float omegaMagnitude = sqrt(axisX*axisX + axisY*axisY +
axisZ*axisZ);

// Normalize the rotation vector if it's big enough to get the
axis
// (that is, EPSILON should represent your maximum allowable
margin of error)
if (omegaMagnitude > EPSILON) {
    axisX /= omegaMagnitude;
    axisY /= omegaMagnitude;
    axisZ /= omegaMagnitude;
}

// Integrate around this axis with the angular speed by the
timestep
// in order to get a delta rotation from this sample over the
timestep
// We will convert this axis-angle representation of the delta
rotation
// into a quaternion before turning it into the rotation matrix.
float thetaOverTwo = omegaMagnitude * dT / 2.0f;
float sinThetaOverTwo = sin(thetaOverTwo);
float cosThetaOverTwo = cos(thetaOverTwo);
deltaRotationVector[0] = sinThetaOverTwo * axisX;
deltaRotationVector[1] = sinThetaOverTwo * axisY;
deltaRotationVector[2] = sinThetaOverTwo * axisZ;
deltaRotationVector[3] = cosThetaOverTwo;
}

timestamp = event.timestamp;
float[] deltaRotationMatrix = new float[9];
SensorManager.getRotationMatrixFromVector(deltaRotationMatrix,
deltaRotationVector);
// User code should concatenate the delta rotation we computed
with the current rotation
// in order to get the updated rotation.
// rotationCurrent = rotationCurrent * deltaRotationMatrix;
}
}

```

标准陀螺仪提供了没有经过任何过滤或校正噪声及偏移的原始数据。实践中陀螺仪的噪声和偏移会引入误差。通过监控其他传感器，比如重力传感器或加速度计来确定偏移和噪声。

使用未校准陀螺仪

未校准陀螺仪类似于陀螺仪，除了前者的旋转速度没有采用陀螺偏移补偿。工厂校准和温度补偿也适用于旋转速度。未校准陀螺仪用于后期处理和融合定位数据。一般情况下，gyroscope_event.values[0]接近

`uncalibrated_gyroscope_event.values[0]-uncalibrated_gyroscope_event.values[3]`。即`calibrated_x ~= uncalibrated_x - bias_estimate_x`。除了旋转速度外，未校准陀螺仪还提供了各轴的估算偏移。下列代码显示了如何获取默认的未校准陀螺仪示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE_UNCALIBRATED);
```

使用线性加速度计

线性加速度计提供了三维矢量，用来表示各设备轴的加速度，不计入重力。下列代码显示了如何获取默认的线性加速度传感器的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
```

概念上，该传感器提供了基于以下关系的加速度数据：

linear acceleration = acceleration - acceleration due to gravity

当要获取不受重力影响的加速度数据时，通常会使用这种传感器。例如，可以使用此传感器来检测汽车可以开多快。线性加速度传感器总是会有偏移量，需要您来删除。最简单的方法是在应用程序中建立校准步骤。校准过程中您可以要求用户在界面上对设备进行设置，然后读取三个轴的偏移量。接着减去加速度传感器的直接读数偏差来获得实际的线性加速度。

来自 "[index.php?title=Motion_Sensors&oldid=13883](#)"



Position Sensors

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接: http://developer.android.com/guide/topics/sensors/sensors_position.html

作者: qishanmingfeng

日期: 2012年9月18日

目录

[[隐藏](#)]

[1 位置传感器](#)

- [1.1 使用游戏旋转矢量传感器](#)
- [1.2 使用地磁旋转矢量传感器](#)
- [1.3 使用方向传感器](#)
- [1.4 使用地磁场传感器](#)
- [1.5 使用未校准磁力计](#)
- [1.6 使用近距离传感器](#)

位置传感器

安卓平台提供了两个传感器：地磁场传感器和方向传感器。让你来确定设备的位置。同时安卓平台也提供了一个让你可以判断设备的表面与一个物体之间的距离（被称为近距离传感器）。地磁场传感器和近距离传感器是基于硬件的。大多数手机和平板制造商都会加进一个地磁场传感器。同样，手机厂商通常也会加进一个近距离传感器来判断手机离用户的脸有多近（例如，打电话的时候）。方向传感器是基于软件的，它的数据源于加速器和地磁场传感器。

注意：方向传感器在Android 2.2 (API Level 8)以上就不再使用。

位置传感器通常用来决定一个设备在世界参考坐标系的物理位置。例如，你可以使用地磁场传感器与加速器结合来确定物理设备与北极的相对位置。你也可以使用方向传感器（或者类似的方法）来确认物理设备在你应用的参考坐标的位置。位置传感器并不用来监视设备的运动。例如摇晃、倾斜，挤压（想了解更多信息，请看[Motion Sensors](#)）。

地磁场传感器和方向传感器返回每个 [SensorEvent](#) 的多维传感值。例如，方向传感器提供每次传感事件三坐标的磁场强度值。同样地，方向传感器提供每次传感事件的偏移，倾斜，滚动值。想了解更多关于传感器使用的坐标系统，请参见 [Sensor Coordinate Systems](#)。近距离传感器为每次传感事件提供单一的值。表1归纳了安卓系统支持的位置传感器。Table 1 .安卓平台支持的位置传感器。

Sensor	Sensor event data	Description	Units of measure
TYPE_MAGNETIC_FIELD	SensorEvent.values[0]	沿X轴的磁场强度	µT
	SensorEvent.values[1]	沿Y轴的磁场强度	
	SensorEvent.values[2]	沿Z轴的磁场强度	
TYPE_ORIENTATION (1)	SensorEvent.values[0]	方位 (与Z轴的夹角)	度
	SensorEvent.values[1]	倾斜 (与X轴的夹角)	
	SensorEvent.values[2]	旋转 (与Y轴的夹角)	
TYPE_PROXIMITY	SensorEvent.values[0]	物体间距(2)	cm

(1)在Android 2.2 (API Level 8)后，不再使用这个传感器。这个传感器框架提供了二选一的方法来获取设备方向，在 [Using the Orientation Sensor](#) 这段探讨。

(2)一些近距离传感器只提供二进制值来代表近和远。

使用游戏旋转矢量传感器

游戏旋转矢量传感器除了不使用地磁场之外，其他方面与旋转矢量传感器相同。因此Y轴不指向北，而是指向其他参考。That reference is allowed to drift by the same order of magnitude as the gyroscope drifts around the Z axis.

因为游戏旋转矢量传感器不使用磁场，相关旋转会更加准确，且不受磁场变化影响。如果不关注北方方位，且正常旋转矢量由于依赖磁场而不符合需求时，可以在游戏中使用这种传感器。

下列代码显示了如何获取默认游戏旋转矢量传感器的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GAME_ROTATION_VECTOR);
```

使用地磁旋转矢量传感器

地磁旋转矢量感应器类似于旋转矢量传感器，但是它采用磁力计而不是陀螺仪。此传感器的精度低于正常旋转矢量传感器精度，但同时功耗降低。如果想不消耗太多电量来收集背景的旋转信息的情况才会使用这种传感器。与batching配合使用时此传感器是最有用的。

下列代码显示了如何获取默认的地磁旋转矢量传感器的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_GEOGRAPHIC_ROTATION_VECTOR);
```

使用方向传感器

方向传感器让你监视设备相对于地球参考坐标(指北极)的位置。以下代码显示怎么得到一个默认方向传感器的实例。

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
```

方向传感器的源数据是通过设备的地磁场传感器结合加速器得到。使用这两个硬件传感器，方向传感器为下面三个方面提供数据：

- 方位：（绕Z轴旋转的角度）。这是北极与设备Y轴之间的角度。例如，如果设备的Y轴与北极一致的话，值为0.如果设备的Y轴指向北极的话，值为180.同样的，当Y轴指向东，值为90.指向西的话，值为270.
- 倾斜：（绕X轴旋转的角度）。当+Z轴旋转到+Y轴方向，这个值为正，当+Z轴旋转到-Y轴时，值为负。值的范围是+180--180度。
- 旋转：（绕Y轴旋转的角度）。当+Z轴旋转到+X轴方向，这个值为正，当+Z轴旋转到-X轴时，值为负。值的范围是90--90度。

与在航空方面的方位，倾斜，旋转定义不同，在这里X轴是指平面的长边（头到尾）。且由于历史原因，旋转的角度顺时针方向为正（数学上来讲，应该是逆时针方向才是正的）。

方向传感器的源数据是通过加工加速传感器和地磁场传感器的未加工传感数据。由于繁重的加工是复杂的，所以方向传感器的准确度和精度会被削弱（只有当旋转组件为0时才可靠）。所以，在Android 2.2 (API level 8)后，这个方向传感器就不建议使用了。取而代之的是使用方向传感器未加工的数据，我们建议你使getRotationMatrix()结合[getOrientation\(\)](#)方法来计算方向值。你也可以使用remapCoordinateSystem()方法把方向值转换成你应用程序的参考坐标值。

下面代码展示如何从方向传感器中直接获取方向数据。我们建议你，只有当设备有较小的旋转时才这样做。

```
public class SensorActivity extends Activity implements SensorEventListener {

    private SensorManager mSensorManager;
    private Sensor mOrientation;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mOrientation = mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
        // You must implement this callback in your code.
    }

    @Override
    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mOrientation,
SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        float azimuth_angle = event.values[0];
        float pitch_angle = event.values[1];
        float roll_angle = event.values[2];
        // Do something with these orientation angles.
    }
}
```

你通常不需要对你从方向传感器上获得的未加工的数据进行数据处理和滤波操作。除了把传感器的坐标系统转成应用的参考坐标。[Accelerometer Play](#)这个sample向你展示了

如何把加速传感器的数据转换成其他参考坐标，它的方法与方向传感器的一样。

使用地磁场传感器

地磁场传感器让你能监视地球磁场的改变。下面的代码展示了怎么去获取一个缺省的地磁场传感器实例。

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

这个传感器为三个坐标轴提供未加工的场强(uT)。通常，你不需要直接使用这个传感器。你可以使用旋转矢量传感器来确未加工的旋转运动或者你可以使用加速器和地磁场传感器结合getRotationMatrix()方法来获得旋转矩阵和倾向矩阵。你可以用这些矩阵加上getOrientation()和getInclination()方法来获得方位和磁场倾斜数据。

使用未校准磁力计

未校正磁力计类似于地磁场传感器，不同之处是前者没有在磁场施加hard iron校准。工厂校准和温度补偿仍会施加到磁场。未校正磁力计有助于处理不良hard iron评估。一般情况geomagneticsensor_event.values[0]接近于

uncalibrated_magnetometer_event.values[0]-
uncalibrated_magnetometer_event.values[3]. 即calibrated_x ~ = uncalibrated_x - bias_estimate_x。

除了在磁场中，未校正磁力计也会提供各轴的hard iron偏差。下列代码演示了如何获取默认的未校正磁力计的示例：

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED);
```

使用近距离传感器

近距离传感器让你确认你的设备离一个物体有多远。下面的代码展示了如何获取一个缺省的近距离传感器实例。

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
```

```
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

近距离传感器通常用来判断人的头距离手机有多远（例如，当用户打电话或接电话时）。大多数近距离传感器探返回一个绝对距离值。以CM为单位。但是，有一些只返回近或远这两种状态。下面的代码教你如何使用近距离传感器。

```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mProximity;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Get an instance of the sensor service, and use that to get an instance
        // of
        // a particular sensor.
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
        float distance = event.values[0];
        // Do something with this sensor data.
    }

    @Override
    protected void onResume() {
        // Register a listener for the sensor.
        super.onResume();
        mSensorManager.registerListener(this, mProximity,
SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        // Be sure to unregister the sensor when the activity pauses.
        super.onPause();
        mSensorManager.unregisterListener(this);
    }
}
```

注意：一些传感器返回二进制值来表示“近”和“远”。在这个例子中，传感器通常在远的状态时报告它的最大探测距离，在近的状态时报告最短距离。一般地，最远距离大于5厘米，但是不同的传感器可能不同。你可以通过 `getMaximumRange()`方法来获取传感器的最远距离。

来自 "[index.php?title=Position_Sensors&oldid=13885](#)"



Environment Sensors

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：http://developer.android.com/guide/topics/sensors/sensors_environment.html

作者： qishanmingfeng

日期： 2012年9月18日

目录

[\[隐藏\]](#)

1 环境传感器

- [1.1 使用光敏, 压力, 温度传感器](#)
- [1.2 使用湿度传感器](#)
 - [1.2.1 露点](#)
 - [1.2.2 绝对湿度](#)

环境传感器

安卓平台提供了四个传感器，让你来监视各种各样的环境属性。你可以使用这些传感器来监视安卓设备周围相关的环境湿度、光照、气压、和温度。这四个传感器都是基于硬件的，只有当设备商把它们置入设备中，才可以使用它们。除了光敏元件，大多数设备商用它来控制屏幕亮度外，其他环境传感器并不总是有效。因此，当你想要获取数据的时候，先检测一下环境传感器是否存在，这一点非常重要。

不像大多数运动传感器和位置传感器，每一个[SensorEvent](#)都会返回传感器数值的多维数组。环境传感器只是返回与每个数据事件有关的单传感能值。例如温度用°C或气压用hPa。运动传感器和位置传感器通常需要高通或低通滤波，但环境传感器并不需要数据滤波或数据处理。表1提供了一个安卓平台支持的环境传感器的一个总结。

Table 1 .安卓平台支持的环境传感器

Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	event.values[0]	°C	周围空气温度

TYPE_LIGHT	event.values[0]	lx	光照度
TYPE_PRESSURE	event.values[0]	hPa or mbar	周围大气压
TYPE_RELATIVE_HUMIDITY	event.values[0]	%	周围相对湿度
TYPE_TEMPERATURE	event.values[0]	°C	设备温度 (1)

(1) 与设备到设备的实现不同。这个传感器在Android 4.0(API Level 14)上就不用了。

使用光敏，压力，温度传感器

从光敏、压力、温度传感器获得的原始数据通常不需要经过校正，滤波，修改。这使它们很容易使用。想要获得传感器的数据，首先，你要创建一个[SensorManager](#)类的实例，这样你将会得到一个物理传感器的实例。然后在[onResume\(\)](#)方法中注册传感器的监听器，接着在[onSensorChanged\(\)](#)方法中开始操作传入的传感器数据。以下代码向你展示这个过程：

```

public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mPressure;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mPressure = mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
        float millibars_of_pressure = event.values[0];
        // Do something with this sensor data.
    }

    @Override
    protected void onResume() {
        // Register a listener for the sensor.
        super.onResume();
        mSensorManager.registerListener(this, mPressure,
            SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        // Be sure to unregister the sensor when the activity pauses.
        super.onPause();
        mSensorManager.unregisterListener(this);
    }
}

```

一定要执行[onAccuracyChanged\(\)](#) 和 [onSensorChanged\(\)](#)这两个回调方法。且当活动暂停时要确保注销传感器。这样做可以防止传感器不断地读出数据和消耗电池。

使用湿度传感器

你可以像使用光敏、压力、温度传感器一样地使用湿度传感器来获取与湿度相关的原始数据。但是，如果一个设备上有两个湿度传感器([TYPE_RELATIVE_HUMIDITY](#))和一个温度传感器([TYPE_AMBIENT_TEMPERATURE](#))，你就可以使用这两个数据流来计算露点和绝对湿度。

露点

露点是在恒压下冷却一定组成蒸气的过程中，凝结出第一个液滴时的温度。下面是露点的计算公式：

$$td(t, RH) = Tn \cdot \frac{\ln(RH/100\%) + m \cdot t / (Tn+t)}{m - [\ln(RH/100\%) + m \cdot t / (Tn+t)]}$$

在这里，

- td = 露点温度, 单位°C
- t = 真实温度, 单位°C
- RH = 真实的相对湿度, 单位%。
- $m = 17.62$
- $Tn = 243.12$

绝对湿度

绝对湿度是指每立方米空气中所含水汽的克数。单位为g/m^3.下面是绝对湿度的计算公式：

$$dv(t, RH) = 216.7 \cdot \frac{(RH/100\%) \cdot A \cdot \exp(m \cdot t / (Tn+t))}{273.15 + t}$$

在这里

- dv = 绝对湿度 , 单位为g/m^3.
- t = 实际温度, 单位°C
- RH = 实际相对湿度, 单位%
- $m = 17.62$
- $Tn = 243.12$ °C
- $A = 6.112$ hPa

来自 "[index.php?title=Environment_Sensors&oldid=11651](#)"



Connectivity

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

连接

在标准的网络连接之外， android在bluetooth、 NFC、 Wi-Fi管理、 USB和SIP等方面提供了更加丰富的接口，可以让你的应用连接到其他的设备上并和其交互。

[蓝牙技术 >](#)

目录

[\[隐藏\]](#)

[1 博客文章](#)

- [1.1 Android's HTTP Clients](#)

[2 Training](#)

- [2.1 Transferring Data Without Draining the Battery](#)
- [2.2 Syncing to the Cloud](#)

博客文章

[Android's HTTP Clients](#)

大部分拥有网络连接的android应用都使用了HTTP协议来请求和接受数据。 android系统包含两种HTTP客户端： HttpURLConnection和Apache HTTP Client。 蓝牙技术支持HTTPS协议，流方式上传和下载，请求超时配置， IPv6和连接池。

Training

[Transferring Data Without Draining the Battery](#)

这里演示了在调度和执行下载使用技术中，诸如高速缓存、轮询和预存取之类问题的最佳解决方法。您将学习在使用无线电时，该如何配置电源的使用方式。这将会影响到您选择在什么时候，什么程度并且如何最小影响电池使用寿命的情况下，去有序的传输数据。

[Syncing to the Cloud](#)

这个类包含了云功能应用的不同策略。它包括使用后端web应用同步你的数据到云端，并且备份数据到云端以便用户在安装有你的应用的新设备上恢复这些数据。

来自 "[index.php?title=Connectivity&oldid=7255](#)"



Bluetooth

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/wireless/bluetooth.html>

翻译： jykenan

更新： 2012.06.19

目录

- [1 蓝牙](#)
 - [1.1 基础知识](#)
 - [1.2 蓝牙权限](#)
 - [1.3 蓝牙设置](#)
 - [1.4 搜索设备](#)
 - [1.4.1 查找匹配设备](#)
 - [1.4.2 扫描设备](#)
 - [1.4.3 使能被发现](#)
 - [1.5 连接设备](#)
 - [1.5.1 作为服务器连接](#)
 - [1.5.2 作为客户端连接](#)
 - [1.6 管理连接](#)
 - [1.6.1 示例](#)
 - [1.7 使用配置文件](#)
 - [1.7.1 Vendor-specific AT 指令](#)
 - [1.7.2 医疗设备模式](#)
 - [1.7.3 创建一个 HDP 应用](#)

蓝牙

Android平台支持蓝牙网络协议栈，实

现蓝牙设备之间数据的无线传输。本文档描述了怎样利用android平台提供的蓝牙API去实现蓝压设备之间的通信。蓝牙具有**point-to-point** 和 **multipoint**两种连接功能。 使用蓝牙API，可以做到：

- 搜索蓝牙设备
- 从本地的Bluetooth adapter中查询已经配对的设备
- 建立RFCOMM通道
- 通过service discovery连接到其它设备
- 在设备之间传输数据
- 管理多个连接

本文档介绍了如何使用传统蓝牙。经典蓝牙适用于电池密集型操作，比如流媒体和安卓设备之间进行通信。对于低功耗蓝牙设备，安卓4.3（API级别18）引入了蓝牙低功耗的API支持。

基础知识

本文档介绍了如何使用Android的蓝牙API来完成的四个必要的主要任务，使用蓝牙进行设备通信，主要包含四个部分：蓝牙设置、搜索设备（配对的或可见的）、连接、传输数据。所有的蓝牙API在[android.bluetooth](#)包中。实现这些功能主要需要下面这几个类和接口：

[BluetoothAdapter](#) 代表本地蓝牙适配器（蓝牙发射器），是所有蓝牙交互的入口。通过它可以搜索其它蓝牙设备，查询已经配对的设备列表，通过

快速查询

- Android蓝牙API允许你的应用和其他设备进行无线数据通信。

[本文档中](#)

[基础知识](#)

[蓝牙权限](#)

[设置蓝牙](#)

[搜索蓝牙](#)

[查找匹配设备](#)
[扫描设备](#)

[连接设备](#)

[服务器的连接](#)
[客户端的连接](#)

[管理连接](#)

[使用配置文件](#)

关键字 [BluetoothAdapter](#)

[BluetoothDevice](#)

[BluetoothSocket](#)

[BluetoothServerSocket](#)

相关例子

[蓝牙聊天](#)

[Bluetooth HDP \(Health Device Profile\)](#)

已知的MAC地址创建**BluetoothDevice**，创

建**BluetoothServerSocket**监听来自其它设备的通信。

BluetoothDevice 代表了一个远端的蓝牙设备， 使用它请求远端蓝牙设备连接或者获取 远端蓝牙设备的名称、 地址、 种类和绑定状态。 （其信息是封装在 bluetoothsocket 中） 。

BluetoothSocket 代表了一个蓝牙套接字的接口（类似于 tcp 中的套接字） ， 他是应用程 序通过输入、 输出流与其他蓝牙设备通信的连接点。

BluetoothServerSocket 代表打开服务连接来监听可能到来的连接请求（属于 server 端） ， 为了连接两个蓝牙设备必须有一个设备作为服务器打开一个服务套接字。 当远端设备发起连 接连接请求的时候，并且已经连接到了的时候， **Blueboothserversocket** 类将会返回一个 bluetoothsocket。

BluetoothClass 描述了一个设备的特性（profile）或该设备上的蓝牙大致可以提供哪些服务(service)，但不可信。比如，设备是一个电话、计算机或手持设备；设备可以提供audio/telephony服务等。可以用它来进行一些UI上的提示。

BluetoothProfile

BluetoothHeadset 提供手机使用蓝牙耳机的支持。这既包括蓝牙耳机和免提（V1.5）模式。

BluetoothA2dp 定义高品质的音频，可以从一个设备传输到另一个蓝牙连接。“A2DP的”代表高级音频分配模式。

BluetoothHealth 代表了医疗设备配置代理控制的蓝牙服务

BluetoothHealthCallback 一个抽象类，使用实现**BluetoothHealth**回调。你必须扩展这个类并实现回调方法接收更新应用程序的注册状态和蓝牙通道状态的变化。

BluetoothHealthAppConfiguration 代表一个应用程序的配置，蓝牙医疗第三方应 用注册与远程蓝牙医疗设备交流。

BluetoothProfile.ServiceListener 当他们已经连接到或从服务断开时通 知**BluetoothProfile** IPX的客户时一个接口（即运行一个特定的配置文件，内部服 务）。

蓝牙权限

为了在你的应用中使用蓝牙功能，至少要在`AndroidManifest.xml`中声明两个权限：`BLUETOOTH`（任何蓝牙相关API都要使用这个权限） 和 `BLUETOOTH_ADMIN`（设备搜索、蓝牙设置等）。

为了执行蓝牙通信，例如连接请求，接收连接和传递数据都必须有`BLUETOOTH`权限。

必须要求`BLUETOOTH_ADMIN`的权限来启动设备发现或操纵蓝牙设置。大多数应用程序都需要这个权限能力，发现当地的蓝牙设备。此权限授予其他的能力不应该使用，除非应用程序是一个“电源管理”，将根据用户要求修改的蓝牙设置注释：要请求`BLUETOOTH_ADMIN`的话，必须要先有`BLUETOOTH`。

在你的应用`manifest` 文件中声明蓝牙权限。例如：

```
<manifest ... >
<uses-permission android:name="android.permission.BLUETOOTH" />
...
</manifest>
```

通过查看`<uses-permission>`资料来声明应用权限获取更多的信息。

蓝牙设置

在你的应用通过蓝牙进行通信之前，你需要确认设备是否支持蓝牙，如果支持，确信它被打开。如果不支持，则不能使用蓝牙功能。如果支持蓝牙，但不能够使用，你刚要在你的应用中请求使用蓝牙。这个要两步完成，使用`BluetoothAdapter`。

1. 获取`BluetoothAdapter` 所有的蓝牙活动请求`BluetoothAdapter`，为了获取`BluetoothAdapter`，呼叫静态方法`getdefaultAdapter()`。这个会返回一个`BluetoothAdapter`，代表设备自己的蓝牙适配器（蓝牙无线电）。这个蓝牙适配器应用于整个系统中，你的应用可以通过这个对象进行交互。如果`getdefaultAdapter()`返回`null`, 则这个设备不支持蓝牙。例如：

```
BluetoothAdapter mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
// Device does not support Bluetooth
}
```

2. 打开蓝牙 其次。你需要确定蓝牙能够使用。通过`isEnabled()`来检查蓝牙当前是否可用。如果这个方法返回`false`, 则蓝牙不能够使用。为了请求蓝牙使用, 呼叫`startActivityForResult()`与的`ACTION_REQUEST_ENABLE`动作意图。通过系统设置中启用蓝牙将发出一个请求 (不停止蓝牙应用)。例如:

```
if (!mBluetoothAdapter.isEnabled()) {
Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```



对话框中显示请求使用蓝牙权限。如果响应"Yes", 这个进程完成 (或失败) 后你的应用将能够使用蓝牙。`REQUEST_ENABLE_BT`常量作为一个整型传到`startActivityForResult()`中 (值必须大于0) , 该系统传回给你, 在你`onActivityResult()`作为实现的`requestCode`参数。

如果调用蓝牙成功, 你的`Activity`就会在`onActivityResult()`中收到`RESULT_OK`结果, 如果蓝牙不能使用由于错误 (或用户响应"No"那么结果返回`RESULT_CANCELED`。

除了通过`onActivityResult()`, 还可以通过监听`ACTION_STATE_CHANGED`这个`broadcast Intent`来知道蓝牙状态是否改变。这个`Intent`包含`EXTRA_STATE`, `EXTRA_PREVIOUS_STATE`两个字段, 分别代表新旧状态。可能的值是`STATE_TURNING_ON`, `STATE_ON`, `STATE_TURNING_OFF`, 还有`STATE_OFF`。

小贴: Enabling discoverability 将自动启用蓝牙。如果您计划执行蓝牙活动之前, 始终使设备可发现, 你可以跳过上面的步骤2。参阅enabling discoverability。

搜索设备

使用`BluetoothAdapter`可以通过设备搜索或查询配对设备找到远程Bluetooth设备。

Device discovery（设备搜索）是一个扫描搜索本地已使能Bluetooth设备并且从搜索到的设备请求一些信息的过程（有时候会收到类似“**discovering**”，“**inquiring**”或“**scanning**”）。但是，搜索到的本地Bluetooth设备只有在打开被发现功能后才会响应一个**discovery**请求，响应的信息包括设备名，类，唯一的**MAC地址**。发起搜寻的设备可以使用这些信息来初始化跟被发现的设备的连接。一旦与远程设备的第一次连接被建立，一个**pairing**请求就会自动提交给用户。如果设备已配对，配对设备的基本信息（名称，类，**MAC地址**）就被保存下来了，能够使用Bluetooth API来读取这些信息。使用已知的远程设备的**MAC地址**，连接可以在任何时候初始化而不必先完成搜索（当然这是假设远程设备是在可连接的空间范围内）。

需要记住，配对和连接是两个不同的概念：

配对意思是两个设备相互意识到对方的存在，共享一个用来鉴别身份的链路键（**link-key**），能够与对方建立一个加密的连接。

连接意思是两个设备现在共享一个**RFCOMM**信道，能够相互传输数据。

目前Android Bluetooth API's要求设备在建立**RFCOMM**信道前必须配对（配对是在使用Bluetooth API初始化一个加密连接时自动完成的）。

下面描述如何查询已配对设备，搜索新设备。

注意：Android的电源设备默认是不能被发现的。用户可以通过系统设置让它在有限的时间内可以被发现，或者可以在应用程序中要求用户使能被发现功能。

查找匹配设备

在搜索设备前，查询配对设备看需要的设备是否已经是已经存在是很值得的，可以调用[getBondedDevices\(\)](#)来做到，该函数会返回一个描述配对设备[BluetoothDevice](#)的结果集。例如，可以使用**ArrayAdapter**查询所有配对设备然后显示所有设备名给用户：

```
Set<BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
```

```
// Loop through paired devices
for (BluetoothDevice device : pairedDevices) {
    // Add the name and address to an array adapter to show in a
ListView
    mArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
}
};
```

[BluetoothDevice](#)对象中需要用来初始化一个连接唯一需要用到的信息就是MAC地址。

扫描设备

要开始搜索设备，只需简单的调用[startDiscovery\(\)](#)。该函数时异步的，调用后立即返回，返回值表示搜索是否成功开始。搜索处理通常包括一个12秒钟的查询扫描，然后跟随一个页面显示搜索到设备Bluetooth名称。

应用中可以注册一个带[ACTION_FOUND](#) Intent的BroadcastReceiver，搜索到每一个设备时都接收到消息。对于每一个设备，系统都会广播ACTION_FOUND Intent，该Intent携带着而外的字段信息[EXTRA_DEVICE](#)和[EXTRA_CLASS](#)，分别包含一个[BluetoothDevice](#)和一个BluetoothClass。

下面的示例显示如何注册和处理设备被发现后发出的广播：

代码如下：

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // Add the name and address to an array adapter to show in a
ListView
            mArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
        }
    }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister
during onDestroy
```

警告：完成设备搜索对于Bluetooth适配器来说是一个重量级的处理，要消耗大量它的资源。一旦你已经找到一个设备来连接，请确保你在尝试连接前使用了cancelDiscovery()来停止搜索。同样，如果已经保持了一个连接的时候，同时执行搜索设备将会显著的降低连接的带宽，所以在连接的时候不应该执行搜索发现。

使能被发现

如果想让本地设备被其他设备发现，可以带ACTION_REQUEST_DISCOVERABLE action Intent调用startActivityForResult(Intent, int) 方法。该方法会提交一个请求通过系统刚设置使设备出于可以被发现的模式（而不影响应用程序）。默认情况下，设备在120秒后变为可以被发现的。可以通过额外增加EXTRA_DISCOVERABLE_DURATION Intent自定义一个值，最大值是3600秒，0表示设备总是可以被发现的（小于0或者大于3600则会被自动设置为120秒）。下面示例设置时间为300：

```
Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```



询问用户是否允许打开设备可以被发现功能时会显示一个对话框。如果用户选择“**Yes**”，设备会在指定时间过后变为可以被发现的。Activity的onActivityResult()回调函数被调用，结果码等于设备变为可以被发现所需时长。如果用户选择“**No**”或者有错误发生，结果码会是Activity.RESULT_CANCELED。

提示：如果Bluetooth没有启用，启用Bluetooth可被发现功能能够自动开启Bluetooth。

在规定的时间内，设备会静静的保持可以被发现模式。如果想在可以被发现模式被更改时受到通知，可以用ACTION_SCAN_MODE_CHANGED Intent注册一个BroadcastReceiver，包含额外的字段信

息`EXTRA_SCAN_MODE`和`EXTRA_PREVIOUS_SCAN_MODE`分别表示新旧扫描模式，其可能的值为`SCAN_MODE_CONNECTABLE_DISCOVERABLE` (`discoverable mode`) , `SCAN_MODE_CONNECTABLE` (`not in discoverable mode but still able to receive connections`) , `SCAN_MODE_NONE` (`not in discoverable mode and unable to receive connections`) 。如果只需要连接远程设备就不需要打开设备的可以被发现功能。只在应用作为一个服务器socket的宿主用来接收进来的连接时才需要使能可以被发现功能，因为远程设备在初始化连接前必须先发现了你的设备。

连接设备

为了在两台设备上创建一个连接，你必须在软件上实现服务器端和客户端的机制，因为一个设备必须打开一个server socket，而另一个必须初始化这个连接(使用服务器端设备的MAC地址进行初始化)。当服务器端和客户端在同一个RFCOMM信道上都有一个BluetoothSocket时，就可以认为它们之间建立了一个连接。在这个时刻，每个设备能获得一个输出流和一个输入流，也能够开始数据传输。本节介绍如何在两个设备之间初始化一个连接。服务器端和客户端获得BluetoothSocket的方法是不同的，服务器端是当一个进入的连接被接受时才产生一个[BluetoothSocket](#)，客户端是在打开一个到服务器端的RFCOMM信道时获得[BluetoothSocket](#)的。



一种实现技术是，每一个设备都自动作为一个服务器，所以每个设备都有一个server socket并监听连接。然后每个设备都能作为客户端建立一个到另一台设备的连接。另外一种代替方法是，一个设备按需打开一个server socket，另外一个设备仅初始化一个到这个设备的连接。Note: 如果两个设备在建立连接之前并没有配对，那么在建立连接的过程中，Android框架将自动显示一个配对请求的notification或者一个对话框，如Figure 3所示。所以在尝试连接设备时，你的应用程序无需确保设备之间已经进行了配对。你的RFCOMM连接将会在用户确认配对之后继续进行，或者用户拒绝或者超时之后失败。

作为服务器连接

如果要连接两个设备，其中一个必须充当服务器，通过持有一个打开的[BluetoothServerSocket](#)对象。服务器socket的作用是侦听进来的连接，如果一个连接被接受，提供一个连接好的[BluetoothSocket](#)对象。
从[BluetoothServerSocket](#)获取到[BluetoothSocket](#)对象之后，[BluetoothServerSocket](#)就可以（也应该）丢弃了，除非你还要用它来接收更多的连接。

下面是建立服务器socket和接收一个连接的基本步骤：

1. 通过调用[listenUsingRfcommWithServiceRecord\(String, UUID\)](#)得到一个[BluetoothServerSocket](#)对象。

该字符串为服务的识别名称，系统将自动写入到一个新的服务发现协议（SDP）数据库接入口到设备上的（名字是任意的，可以简单地是应用程序的名称）项。UUID也包括在SDP接入口中，将是客户端设备连接协议的基础。也就是说，当客户端试图连接本设备，它将携带一个UUID用来唯一标识它要连接的服务，UUID必须匹配，连接才会被接受。

2. 通过调用[accept\(\)](#)来侦听连接请求。

这是一个阻塞的调用，知道有连接进来或者产生异常才会返回。只有远程设备发送一个连接请求，并且携带的UUID与侦听它socket注册的UUID匹配，连接请求才会被接受。如果成功，[accept\(\)](#)将返回一个连接好的[BluetoothSocket](#)对象。

3. 除非需要再接收另外的连接，否则的话调用[close\(\)](#)。

[close\(\)](#)释放server socket和它的资源，但不会关闭连接[accept\(\)](#)返回的连接好的[BluetoothSocket](#)对象。与TCP/IP不同，RFCOMM同一时刻一个信道只允许一个客户端连接，因此大多数情况下意味着在[BluetoothServerSocket](#)接受一个连接请求后应该立即调用[close\(\)](#)。

[accept\(\)](#)调用不应该在主Activity UI线程中进行，因为这是个阻塞的调用，会妨碍其他的交互。经常是在在一个新线程中做[BluetoothServerSocket](#)或[BluetoothSocket](#)的所有工作来避免UI线程阻塞。注意所有[BluetoothServerSocket](#)或[BluetoothSocket](#)的方法都是线程安全的。

====示例====： 下面是一个简单的接受连接的服务器组件代码示例：

示例

```

private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned to
        mmServerSocket,
        // because mmServerSocket is final
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the client
            code
            tmp =
mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) { }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is returned
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                // Do work to manage the connection (in a separate
thread)
                manageConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    /**
     * Will cancel the listening socket, and cause the thread to finish
     */
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) { }
    }
}

```

本例中，仅仅只接受一个进来的连接，一旦连接被接受获取到BluetoothSocket，就发送获取到的BluetoothSocket给一个单独的线程，然后关闭BluetoothServerSocket并跳出循环。注意：accept()返回BluetoothSocket后，socket已经连接了，所以在客户端不应该呼叫connect()。

manageConnectedSocket()是一个虚方法，用来初始化线程好传输数据。

通常应该在处理完侦听到的连接后立即关闭BluetoothServerSocket。在本例中，close()在得到BluetoothSocket后马上被调用。还需要在线程中提供一个公

共的方法来关闭私有的BluetoothSocket，停止服务端socket的侦听。

作为客户端连接

为了实现与远程设备的连接，你必须首先获得一个代表远程设备[BluetoothDevice](#)对象。然后使用BluetoothDevice对象来获取一个[BluetoothSocket](#)来实现来接。

下面是基本的步骤：

1. 用[BluetoothDevice](#)调用[createRfcommSocketToServiceRecord\(UUID\)](#)获取一个BluetoothSocket对象。这个初始化的BluetoothSocket会连接到BluetoothDevice。UUID必须匹配服务器设备在打开BluetoothServerSocket时用到的UUID(用[java.util.UUID](#)) [listenUsingRfcommWithServiceRecord\(String, UUID\)](#))。可以简单的生成一个UUID串然后在服务器和客户端都使用该UUID。
2. 调用[connect\(\)](#)完成连接 当调用这个方法的时候，系统会在远程设备上完成一个SDP查找来匹配UUID。如果查找成功并且远程设备接受连接，就共享RFCOMM信道，[connect\(\)](#)会返回。这也是一个阻塞的调用，不管连接失败还是超时（12秒）都会抛出异常。

注意：要确保在调用[connect\(\)](#)时没有同时做设备查找，如果在查找设备，该连接尝试会显著的变慢，慢得类似失败了。

实例：下面是一个完成Bluetooth连接的样例线程：

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        // Use a temporary object that is later assigned to mmSocket,
        // because mmSocket is final
        BluetoothSocket tmp = null;
        mmDevice = device;

        // Get a BluetoothSocket to connect with the given
        BluetoothDevice
        try {
            // MY_UUID is the app's UUID string, also used by the server
            code
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
        }
        mmSocket = tmp;
    }
}
```

```
}

public void run() {
    // Cancel discovery because it will slow down the connection
    mBluetoothAdapter.cancelDiscovery();

    try {
        // Connect the device through the socket. This will block
        // until it succeeds or throws an exception
        mmSocket.connect();
    } catch (IOException connectException) {
        // Unable to connect; close the socket and get out
        try {
            mmSocket.close();
        } catch (IOException closeException) { }
        return;
    }

    // Do work to manage the connection (in a separate thread)
    manageConnectedSocket(mmSocket);
}

/** Will cancel an in-progress connection, and close the socket */
public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) { }
}
```

注意到`cancelDiscovery()`在连接操作前被调用。在连接之前，不管搜索有没有进行，该调用都是安全的，不需要确认（当然如果有要确认的需求，可以调用`isDiscovering()`）。`manageConnectedSocket()`是一个虚方法，用来初始化线程好传输数据。在对`BluetoothSocket`的处理完成后，记得调用`close()`来关闭连接的socket和清理所有的内部资源。

管理连接

如果已经连接了两个设备，他们都已经拥有各自的连接好的[BluetoothSocket](#)对象。那就是一个有趣的开始，因为你在设备间共享数据了。使用[BluetoothSocket](#)，传输任何数据通常来说都很容易了：

1. 通过socket获取输入输出流来处理传输（分别使用getInputStream()和getOutputStream()）。
 2. 用read(byte[])和write(byte[])来实现读写。

仅此而已。

当然，还是有很多细节需要考虑的。首要的，需要用一个专门的线程来实现流的读写。只是很重要的，因为`read(byte[])`和`write(byte[])`都是阻塞的调用。`read(byte[])`会阻塞直到流中有数据可读。`write(byte[])`通常不会阻塞，但是如果远程设备调用`read(byte[])`不够快导致中间缓冲区满，它也可能阻塞。所以线程中的主循环应该用于读取[InputStream](#)。线程中也应该有单独的方法用来完成写[OutputStream](#)。

示例

下面是一个如上面描述那样的例子：

```

private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        mmInStream = null;
        mmOutStream = null;

        // Get the input and output streams, using temp objects because
        // member streams are final
        try {
            mmInStream = socket.getInputStream();
            mmOutStream = socket.getOutputStream();
        } catch (IOException e) {}

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024]; // buffer store for the stream
        int bytes; // bytes returned from read()

        // Keep listening to the InputStream until an exception occurs
        while (true) {
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);
                // Send the obtained bytes to the UI activity
                mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
                    .sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }

    /* Call this from the main activity to send data to the remote
device */
    public void write(byte[] bytes) {
        try {
            mmOutStream.write(bytes);
        } catch (IOException e) {}
    }
}

```

```

}
}

/* Call this from the main activity to shutdown the connection */
public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) { }
}
}
}

```

构造函数中得到需要的流，一旦执行，线程会等待从InputStream来的数据。当read(byte[])返回从流中读到的字节后，数据通过父类的成员Handler被送到主Activity，然后继续等待读取流中的数据。向外发送数据只需简单的调用线程的write()方法。线程的cancel()方法时很重要的，以便连接可以在任何时候通过关闭BluetoothSocket来终止。它应该总在处理完Bluetooth连接后被调用。

使用配置文件

从Android 3.0开始，Bluetooth API就包含了对Bluetooth profiles的支持。Bluetooth profile是基于蓝牙的设备之间通信的无线接口规范。例如Hands-Free profile（免提模式）。如果移动电话要连接一个无线耳机，他们都要支持Hands-Free profile。

你在你的类里可以完成[BluetoothProfile](#)接口来支持某一Bluetooth profiles。Android Bluetooth API完成了下面的Bluetooth profile：

- 耳机。Headset profile提供了移动电话上的Bluetooth耳机支持。Android提供了[BluetoothHeadset](#)类，它是一个协议，用来通过IPC（interprocess communication）控制Bluetooth Headset Service。[BluetoothHeadset](#)既包含Bluetooth Headset profile也包含Hands-Free profile，还包括对AT命令的支持。
- A2DP. Advanced Audio Distribution Profile (A2DP) profile，高级音频传输模式。Android提供了BluetoothA2dp类，这是一个通过IPC来控制Bluetooth A2DP的协议。
- Android4.0(API级别14)推出了支持蓝牙医疗设备模式(HDP)，这使您可以创建支持蓝牙的医疗设备，使用蓝牙通信的应用程序，例如心率监视器，血液，温度计和秤等等。支持的设备和相应的设备数据专业化代码，请参阅蓝牙分配在www.bluetooth.org数。请注意，这些值的ISO / IEEE11073-20601引用[7] MDC_DEV_SPEC_PROFILE_*命名代码附件的规范。对于更多的HDP讨论，查看[Health Device Profile](#).

下面是使用profile的基本步骤：

1. 获取默认的Bluetooth适配器。
2. 使用getProfileProxy()来建立一个与profile相关的profile协议对象的连接。在下面的例子中，profile协议对象是[BluetoothHeadset](#)的一个实例。
3. 设置[BluetoothProfile.ServiceListener](#)。该listener通知[BluetoothProfile](#) IPC客户端，当客户端连接或断连服务器的时候
4. 在[android.bluetooth.BluetoothProfile\) onServiceConnected\(\)](#)内，得到一个profile协议对象的句柄。
5. 一旦拥有了profile协议对象，就可以用它来监控连接的状态，完成于该profile相关的其他操作。

例如，下面的代码片段显示如何连接到一个[BluetoothHeadset](#)协议对象，用来控制Headset profile：

```
BluetoothHeadset mBluetoothHeadset;

// Get the default adapter
BluetoothAdapter mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();

// Establish connection to the proxy.
mBluetoothAdapter.getProfileProxy(context, mProfileListener,
BluetoothProfile.HEADSET);

private BluetoothProfile.ServiceListener mProfileListener = new
BluetoothProfile.ServiceListener() {
    public void onServiceConnected(int profile, BluetoothProfile proxy)
{
    if (profile == BluetoothProfile.HEADSET) {
        mBluetoothHeadset = (BluetoothHeadset) proxy;
    }
}
public void onServiceDisconnected(int profile) {
    if (profile == BluetoothProfile.HEADSET) {
        mBluetoothHeadset = null;
    }
}
};

// ... call functions on mBluetoothHeadset

// Close proxy connection after use.
mBluetoothAdapter.closeProfileProxy(mBluetoothHeadset)
```

Vendor-specific AT 指令

从Android 3.0开始，应用程序可以注册侦听预定义的Vendor-specific AT命令这样的系统广播（如Plantronics +XEVENT command）。例如，应用可以接收到一个广播，该广播表明连接的设备电量过低，然后通知用户做好其他需要的操作。创建一个带[ACTION_VENDOR_SPECIFIC_HEADSET_EVENT](#) intent的broadcast receiver来为耳机处理

医疗设备模式

Android4.0(API级别14)推出了支持蓝牙医疗设备模式 (HDP)，这使您可以创建支持蓝牙的医疗设备，使用蓝牙通信的应用程序，例如心率监视器，血液，温度计和秤。蓝牙卫生API包括基础类[BluetoothHealth](#), [BluetoothHealthCallback](#), [BluetoothHealthAppConfiguration](#)。在使用蓝牙卫生API，它有助于理解这些关键的HDP概念：

概念	描述
源	HDP定义的一个角色. 源是一个传送传输医疗数据的医疗设备（体重，血糖，体温等）到一个小型设备中，例如Android手机或平板电脑。
接收器	HDP定义的一个角色. 在HDP中，一个接收器是一种接收医疗数据的小型设备。在一个Android HDP 应用中，接收器代表了一个BluetoothHealthAppConfiguration 对象。
注册	指一个特定的医疗设备注册接收器。
连接	指打开医疗设备和智能设备之间的通道，如Android手机或平板。

创建一个 HDP 应用

创建一个Android HDP应用要下面几步：

1. 获取一个参考的[BluetoothHealth](#)代理对象。

类似普通的耳机和A2DP设备，你必须调用BluetoothProfile与[getProfileProxy \(\)](#)。[ServiceListener](#) 和医疗配置类型来建立一个配置代理对象的连接。

2. 创建一个[BluetoothHealthCallback](#)和注册的应用程序配置
([BluetoothHealthAppConfiguration](#)) 作为一个医疗sink。

3. 建立一个连接到医疗设备。一些设备将初始化连接。开展这一步对于这些设备，这是不必要的。

4. 当连接成功到一个医疗设备时，使用文件描述符读/写到医疗设备。

接收到的数据需要使用健康管理，实现了IEEE11073-XXXX规范进行解释。

5. 当完成后，关闭医疗通道和注销申请。通道也有延伸静止时关闭。

为了完善这个例子说明这些步骤。查看[Bluetooth HDP \(Health Device Profile\)](#)

。

来自“[index.php?title=Bluetooth&oldid=13830](#)”

NFC

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： River

原文链

接：

<http://developer.android.com/guide/topics/connectivity/nfc/index.html>

Near Field Communication

近场通信(NFC)是一套短距离无线技术，通常需要4厘米或者更短的距离来初始化一个连接。 NFC允许我们在NFC目标(其他支持NFC功能的设备等)和android供电设备之间，或者在两个不同的android供电设备之间分享小型有效荷载数据。 译者注：因为NFC的数据传输速率较低，仅为212Kbps，不适合诸如音视频流等需要较高带宽的应用。

标签范围的复杂性。简单标签仅仅提供读写语义，甚至有时候会使用一次性可编程领域使卡只读。更复杂的标签提供数学运算，并拥有加密硬件认证入口通道。最复杂的标签包含操作环境，允许与标签上执行的代码进行复杂交互。在标签中存储的数据同样也能被写入多种格式，不过很多Android框架API都是围绕基于NFC论坛标准，这个标准名为：NDEF(NFC数据交换格式 NFC Data Exchange Format)

NFC_Basics-NFC基础

本文档介绍了Android怎样处理发现的NFC标签以及如何通知与申请有关的数据应用。同时复习怎样在你的应用程序中使用NDEF数据，并给出一个支持基本的NFC功能的Android框架API的概述。

Advanced NFC

本文档复习启用各种android支持的标签技术，当你不是使用NDEF数据时，或者当你正在使用Android不能完全明白的NDEF 数据时，你必须使用你自定义的协议栈用原始字节的方式读取标签。这些情况下， android支持检测某些标签技术并使用你自定义的协议栈打开标签间的连接。

来自“[index.php?title=NFC&oldid=9542](#)”



NFC Basics

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

- [1 NFC基础](#)
- [2 标签调度系统](#)
- [3 NFC标签是如何调度应用程序](#)
- [4 要求NFC访问Android清单](#)
- [5 NFC的意图过滤](#)
- [6 创建NDEF记录的常见类型](#)
 - [6.1 TNF_ABSOLUTE_URI](#)
 - [6.2 TNF_MIME_MEDIA](#)
 - [6.3 TNF_WELL_KNOWN with RTD_TEXT](#)
 - [6.4 TNF_WELL_KNOWN with RTD_URI](#)
 - [6.5 TNF_EXTERNAL_TYPE](#)
 - [6.6 安卓应用程序记录](#)
- [7 其他NDEF设备消息](#)

NFC基础

本文档介绍了Android中执行的NFC的基本任务。它说明了如何发送和接收NDEF消息的形式NFC数据，并介绍了Android框架中的API对这些功能的支持。至于包括与非NDEF数据工作的讨论的更高级的主题，详见[NFC进阶](#)。

NDEF数据和Android工作时，一般主要用于以下两个用途：

- 阅读NFC标签NDEF数据

- 近距离分享功能NDEF信息从一个设备到另一个[Android Beam™](#)

阅读NFC标签NDEF数据处理与标签的调度系统，分析发现NFC标签，适当的数据进行分类，并开始在分类数据感兴趣的应用程序。要处理扫描NFC标签的应用程序可以声明的意图过滤和处理数据的要求。

在Android Beam™功能允许设备推到另一台设备的物理窃听设备NDEF消息。这种相互作用提供了一个简单的方法比其他无线技术如蓝牙发送数据，因为与NFC，没有手动装置发现或配对要求。范围来时，两个设备连接自动启动。Android的光束可以通过一组的NFC API的，所以任何应用都可以传输设备之间的信息。例如，联系人，浏览器和YouTube应用程序使用Android梁共享联系人，网页，并与其他设备的视频。

标签调度系统

Android的设备通常是寻找NFC标签，当屏幕被锁定，除非NFC是设备的设置菜单中禁用。当一个Android供电设备，发现了一个NFC标签，所期望的行为是最合适的活动，而不要求用户使用什么样的应用处理的意图。由于设备扫描NFC标签，在很短的范围内，很容易让用户手动选择的活动将迫使他们远离标签和移动设备断开连接。你要发展你的活动，只处理NFC标签，您的活动，关心，以防止出现活动选择。

为了帮助您与这个目标，Android提供了一个特殊的标记调度系统，分析扫描NFC标签，解析它们，并试图找到感兴趣的是在扫描数据的应用。它通过：

1、NFC标签的解析，并找出MIME类型或一个URI标识在标签中的数据有效载荷。

2、MIME类型或URI和负载封装到一个意图。前两个步骤的描述，在NFC标签如何被映射到MIME类型和URI。

3、启动基于意图的活动。这是在NFC标签如何被分派到应用程序的描述。

NFC标签如何被映射到MIME类型和URI

在你开始写你的NFC应用，重要的是要了解不同类型的NFC标签的，标签调度系统如何解析NFC标签，标签调度系统的专项整治工作时检测NDEF的消息。NFC标签来在各种各样的技术，还可以有许多不同的方式写在他们的数据。Android已经支持NFC论坛的NDEF标准，这是由定义。

NDEF数据封装内包含一个或多个记录（NdefRecord）的消息

（NdefMessage）。每个NDEF记录必须根据你要创建的记录类型，规格，形成。Android还支持其他类型android.nfc.tech包不包含NDEF的数据，你可以工作在使用类的标签。要了解有关这些技术的更多信息，请参阅先进的NFC话题。与其他类型的标记这些工作包括编写自己的协议栈与标签进行通信，所以我们建议NDEF在可能的情况下发展的易用性和最大支持Android的供电设备使用。

注：要下载完整NDEF的规格，请到[NFC Forum Specification Download](#)下载，到[Creating common types of NDEF records](#)查看如何构造NDEF记录的例子。

现在，你有NFC标签的背景下，以下各节描述更详细的Android如何处理NDEF格式化标签。当一个Android供电设备扫描NFC标签NDEF格式的数据，它解析的消息，并试图找出数据的MIME类型或标识的URI。要做到这一点，系统会读取第一的NdefMessage NdefRecord内，以确定如何解释的整个NDEF消息（NDEF消息可以有多个NDEF记录）。在一个格式良好的NDEF消息，第一NdefRecord 包含以下字段：

3位肿瘤坏死因子（类型名称格式）指示如何解释变量长度类型字段。有效的值在表1中所述。可变长度类型 描述记录类型。如果使用TNF_WELL_KNOWN，使用此字段指定的记录类型定义（RTD）。有效的电阻值在表2。可变长度的ID 一个记录的唯一标识符。这个领域是不是经常使用，但如果需要唯一标识标签，你可以为它创建一个ID。可变长度的有效载荷 要读或写的实际数据有效载荷。一个NDEF消息可以包含多个NDEF记录，所以不承担充分的有效载荷是在的NDEF消息的第一NDEF纪录。标签调度系统使用肿瘤坏死因子和类型字段对应的MIME类型，或URI的NDEF消息，。如果成功，它封装了，里面的信

息ACTION_NDEF_DISCOVERED沿与实际载荷的意图。不过，也有标签调度系统的情况下，当不能确定基于第一NDEF记录的数据类型。发生这种情况，当NDEF数据不能被映射到一个MIME类型或URI，或者时NFC标签不包含NDEF的数据，开始。在这种情况下，一个 Tag 对象有标签的技术信息和有效载荷的意图，而不是一个ACTION_TECH_DISCOVERED封装内。

表1介绍了如何标记调度系统地图肿瘤坏死因子和类型字段MIME类型或URIs。它还介绍了这的TNFs不能映射到MIME类型或URI。在这些情况下，标签调度系统回落到 ACTION_TECH_DISCOVERED的。

例如，如果标签调度系统遇到一个类型的记录TNF_ABSOLUTE_URI，它映射到一个URI，该记录的可变长度的类型字段。标签调度系统封装在一个数据字段，URI 以及与其他信息的标签，如有效载荷，ACTION_NDEF_DISCOVERED意图。另一方面，如果遇到记录类型TNF_UNKNOWN，它创建一个封装标签的技术，而不是意图。

NFC标签是如何调度应用程序

当标签调度系统做创建封装NFC标签和识别信息的意图，它的意图发送到感兴趣的应用程序过滤器的意图。如果多个应用程序可以处理的意图，活动选择，使用户可以选择的活动。标签调度系统定义了三个意图，这是最高到最低优先级的顺序列出：

1.ACTION_NDEF_DISCOVERED: 此意图是用来启动一个活动标签包含NDEF的载荷扫描时，是一个公认的类型。这是最高优先级的意图，和标签调度系统尝试启动活动之前，任何其他的意图，这个意图尽可能。

2.ACTION_TECH_DISCOVERED: 如果没有活动登记来处理的ACTION_NDEF_DISCOVERED的 意图，标签调度系统尝试启动应用程序与此意图。这意图也直接开始（没有开始ACTION_NDEF_DISCOVERED第一）如果扫描的标签包含NDEF数据不能被映射到MIME类型或URI，或者如果标签不包含NDEF数据，但已知的标签技术。

3.ACTION_TAG_DISCOVERED: 此意图开始活动，如果没有处理的ACTION_NDEF_DISCOVERED或ACTION_TECH_DISCOVERED 意图。

标签调度系统工作的基本方法如下： 尝试启动与 标签（或者解析时被标记调度系统创建的意图的活动 ACTION_NDEF_DISCOVERED或ACTION_TECH_DISCOVERED）。

2. 如果没有这一意图过滤器的活动，尝试开始下一个优先级最低的意图（无论是活动的意图或标签调度系统尝试所有可能的意图，直到应用程序过滤器ACTION_TECH_DISCOVERED的或ACTION_TAG_DISCOVERED直到）。
3. 如果没有应用任何意图过滤器，什么也不做。



图1。标签调度系统

只要可能，工作NDEF消息和ACTION_NDEF_DISCOVERED的意图，因为它是最具体的出三个。此意图，让您在最合适的时间比其他两个意图开始您的应用程序，为用户提供更好的体验。

要求NFC访问Android清单

才可以访问设备的NFC硬件和妥善处理NFC的意图之前，在你申报的这些项目的AndroidManifest.xml文件：

- 通过NFC的<uses-permission>属性访问NFC硬件：

```
<uses-permission android:name="android.permission.NFC" />
```

- 可以支持您的应用程序的最低的SDK版本。API级9仅支持通过有限的标签派遣ACTION_TAG_DISCOVERED，只有通过访问NDEF消息EXTRA_NDEF_MESSAGES额外。没有其他标记的属性或I / O操作都可以访问。API 10级，包括全面支持读/写，以及前景NDEF推，API级别14提供了一个更简单的方法，推动与Android束和额外的方便的方法NDEF消息到其他设备，以创建NDEF记录。

```
<uses-sdk android:minSdkVersion="10" />
```

- 只为有NFC硬件设备的使用功能的元素，使您的应用程序显示在谷歌播放：

```
<uses-feature android:name="android.hardware.nfc"  
    android:required="true" />
```

如果您的应用程序使用NFC功能，但该功能是不是你的应用程序的关键，你可以省略了使用功能的元素，并在运行时的NFC availability检查，通过检查，看看是否[getDefaultAdapter \(\)](#) 是空的。

NFC的意图过滤

启动您的应用程序，你要处理一个NFC标签进行扫描时，您的应用程序可以过滤为一，二，或所有三个在Android体现了NFC意图。然而，你通常要筛选的ACTION_NDEF_DISCOVERED大多数控制应用程序启动时的意图。ACTION_TECH_DISCOVERED意图是后备ACTION_NDEF_DISCOVERED当过滤器时的有效载荷是不是NDEF ACTION_NDEF_DISCOVERED或没有应用。过滤ACTION_TAG_DISCOVERED通常是一般类别筛选。许多应用程序将过滤ACTION_NDEF_DISCOVERED或ACTION_TECH_DISCOVERED ACTION_TAG_DISCOVERED，使您的应用程序有一个起点的概率很低。ACTION_TAG_DISCOVERED仅作为最后手段的应用，在没有其他应用程序被安装到处理的案件筛选 ACTION_NDEF_DISCOVERED或ACTION_TECH_DISCOVERED意图。

因为NFC标签部署各不相同，很多次是不是你的控制之下，这并不总是可能的，这是为什么你可以退回到其他两个意图在必要时。当你有控制标记和书面数据的类型，建议您使用NDEF格式化您的标签。以下各节描述如何筛选各类型的意图。

ACTION_NDEF_DISCOVERED 要筛选ACTION_NDEF_DISCOVERED意图，宣布要筛选的数据类型的意图过滤器。下面的例子过滤器ACTION_NDEF_DISCOVERED 意图与一个MIME类型text / plain的：

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
</intent-filter>
```

下面的例子为形式的URI过滤器

<http://developer.android.com/index.html。>

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http"
          android:host="developer.android.com"
          android:pathPrefix="/index.html" />
</intent-filter>
```

ACTION_TECH_DISCOVERED

如果您的活动过滤器ACTION_TECH_DISCOVERED意图，你必须创建一个XML资源文件，该文件指定在您的活动，支持高新技术列表集的技术。您的活动被认为是一场比赛，如果一个高科技列表的集合是一个技术支持的标签，您可以通过调用getTechList () 的子集。

例如，如果扫描标签支持MifareClassic的，NdefFormattable，NFCA，您的高科技名单组必须指定所有三个，两个，或者为您的活动中要匹配的技术（没有别的）。

下面的示例定义了所有的技术。你可以删除那些你不需要的。<project-root> / RES / XML文件夹保存在这个文件（可以将其命名为你想要的东西）。

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.IsoDep</tech>
        <tech>android.nfc.tech.NfcA</tech>
        <tech>android.nfc.tech.NfcB</tech>
        <tech>android.nfc.tech.NfcF</tech>
        <tech>android.nfc.tech.NfcV</tech>
        <tech>android.nfc.tech.Ndef</tech>
        <tech>android.nfc.tech.NdefFormattable</tech>
        <tech>android.nfc.tech.MifareClassic</tech>
        <tech>android.nfc.tech.MifareUltralight</tech>
    </tech-list>
</resources>
```

你也可以指定多个高科技列表套。每个技术清单 集被认为是独立的，和你的活动被认为是一场比赛，如果任何单一的一套技术清单getTechList () 是一种技术，通过返回的子集。这提供与 匹配技术的语义。下面的例子匹配的标签，可以支持NFC A和NDEF技术的或可以支持NfcB和NDEF技术：

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.NfcA</tech>
        <tech>android.nfc.tech.Ndef</tech>
    </tech-list>
</resources>

<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.NfcB</tech>
        <tech>android.nfc.tech.Ndef</tech>
    </tech-list>
</resources>
```

在你的AndroidManifest.xml文件，指定资源文件，您只需在创建<meta-data>里面的元素<activity> 元素，如在下面的例子：

```
<activity>
    ...
    <intent-filter>
        <action android:名称=“android.nfc.action.TECH_DISCOVERED” />
    </意图过滤器>

    <元数据 的android: 名称=的“android.nfc.action.TECH_DISCOVERED”
        机器人：资源= “@ XML / nfc_tech_filter的” />
    ...
</活动>
```

欲了解更多有关使用标签技术和工作信息ACTION_TECH_DISCOVERED意图，看到在高级NFC文档标签技术支持工作。

ACTION_TAG_DISCOVERED

要筛选ACTION_TAG_DISCOVERED使用下面的意图过滤器：

```
<intent-filter>
    <action android:名称=“android.nfc.action.TAG_DISCOVERED” />
</意图过滤器>
```

从意图获取信息

如果一项活动，因为NFC的意图开始，你可以得到约意图扫描NFC标签的信息。意图可以包含以下额外取决于扫描标签：

- **EXTRA_TAG** (必填) : 代表扫描标签一个标签对象。
- **EXTRA_NDEF_MESSAGES** (可选) : 一个数组的标签解析NDEF消息。这额外的强制性意图。
- {@链接android.nfc.NfcAdapter # EXTRA_ID} (可选) : 低级别的ID标签。

要获得这些额外的，请检查如果您的活动，推出NFC的意图之一，以确保标签被扫描，然后获得了额外的意图。下面的例子检查为ACTION_NDEF_DISCOVERED 意图和得到的意图额外NDEF消息。

```
public void onResume() {
    super.onResume();
    ...
    if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction()))
    {
        Parcelable[] rawMsgs =
        intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        if (rawMsgs != null) {
            msgs = new NdefMessage[rawMsgs.length];
            for (int i = 0; i < rawMsgs.length; i++) {
                msgs[i] = (NdefMessage) rawMsgs[i];
            }
        }
    }
    //process the msgs array
}
```

另外，您可以获取标签的意图对象，其中将包含有效载荷，并允许你列举标签的技术：

```
Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
```

创建NDEF记录的常见类型

本节将介绍写入NFC标签或者采用安卓Beam发送数据时，如何创建NDEF记录常见类型。采用安卓4.0 (API级别14)，createUri()方法可以自动创

建URI记录。采用安卓4.1 (API级别14) ,
createExternal()和createMime()可以用创建MIME和外置式NDEF记录。必要时使用这些辅助方法可以在手动创建NDEF记录时避免错误。

TNF_ABSOLUTE_URI

注意：建议您使用RTD_URI类型而不是TNF_ABSOLUTE_URI，因为前者会更有效果。

你可以按以下方式创建TNF_ABSOLUTE_URI NDEF 记录：

```
NdefRecord uriRecord = new NdefRecord(
    NdefRecord.TNF_ABSOLUTE_URI ,
    "http://developer.android.com/index.html".getBytes( Charset.forName (
    "US-ASCII" )),
    new byte[0] , new byte[0]);
```

以前NDEF记录的意图筛选器是这样的：

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http"
          android:host="developer.android.com"
          android:pathPrefix="/index.html" />
</intent-filter>
```

TNF_MIME_MEDIA

你可以通过以下方式创建 TNF_MIME_MEDIA记录。

使用 createMime()方法：

```
NdefRecord mimeRecord =
NdefRecord.createMime( "application/vnd.com.example.android.beam" ,
    "Beam me up, Android".getBytes( Charset.forName( "US-ASCII" )) );
```

手动创建 NdefRecord:

```
NdefRecord mimeRecord = new NdefRecord(
file:///D:/guide/NFC_Basics[2015/9/23 19:18:33]
```

```
NdefRecord.TNF_MIME_MEDIA ,  
  
"application/vnd.com.example.android.beam" .getBytes( Charset.forName( "US-ASCII" ) ),  
    new byte[0] , "Beam me up,  
Android!" .getBytes( Charset.forName( "US-ASCII" ) ) );
```

NDEF记录的意图筛选器是这样的：

```
<intent-filter>  
    <action android:name= "android.nfc.action.NDEF_DISCOVERED" />  
    <category android:name= "android.intent.category.DEFAULT" />  
    <data android:mimeType= "application/vnd.com.example.android.beam"  
/>  
</intent-filter>
```

TNF_WELL_KNOWN with RTD_TEXT

可以采用以下方式来创建TNF_WELL_KNOWN NDEF记录：

```
public NdefRecord createTextRecord( String payload, Locale locale,  
boolean encodeInUtf8 ) {  
    byte[] langBytes =  
locale.getLanguage() .getBytes( Charset.forName( "US-ASCII" ) );  
    Charset utfEncoding = encodeInUtf8 ? Charset.forName( "UTF-8" ) :  
Charset.forName( "UTF-16" );  
    byte[] textBytes = payload.getBytes( utfEncoding );  
    int utfBit = encodeInUtf8 ? 0 : (1 << 7);  
    char status = (char) (utfBit + langBytes.length);  
    byte[] data = new byte[1 + langBytes.length +  
textBytes.length];  
    data[0] = (byte) status;  
    System.arraycopy( langBytes, 0, data, 1, langBytes.length );  
    System.arraycopy( textBytes, 0, data, 1 + langBytes.length,  
textBytes.length );  
    NdefRecord record = new NdefRecord( NdefRecord.TNF_WELL_KNOWN,  
NdefRecord.RTD_TEXT, new byte[0] , data );  
    return record;  
}
```

意图筛选器是这样的：

```
<intent-filter>  
    <action android:name= "android.nfc.action.NDEF_DISCOVERED" />  
    <category android:name= "android.intent.category.DEFAULT" />  
    <data android:mimeType= "text/plain" />  
</intent-filter>
```

TNF_WELL_KNOWN with RTD_URI

可以采用以下方式创建TNF_WELL_KNOWN NDEF记录。

使用 createUri(String) 方式：

```
NdefRecord rtdUriRecord1 =
NdefRecord.createUri("http://example.com");
```

使用 createUri(Uri) 方式：

```
Uri uri = new Uri("http://example.com");
NdefRecord rtdUriRecord2 = NdefRecord.createUri(uri);
```

手动创建 NdefRecord：

```
byte[] uriField = "example.com".getBytes(Charset.forName("US-
ASCII"));
byte[] payload = new byte[uriField.length + 1]; //add
1 for the URI Prefix
byte payload[0] = 0x01;
//prefixes http://www. to the URI
System.arraycopy(uriField, 0, payload, 1, uriField.length);
//appends URI to payload
NdefRecord rtdUriRecord = new NdefRecord(
    NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_URI, new byte[0],
    payload);
```

NDEF记录的意图筛选器是这样的：

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http"
        android:host="example.com"
        android:pathPrefix="" />
</intent-filter>
```

TNF_EXTERNAL_TYPE

可以采用以下方式创建 TNF_EXTERNAL_TYPE NDEF记录：

使用createExternal()方法：

```
byte[] payload; //assign to your data
String domain = "com.example"; //usually your app's package name
String type = "externalType";
NdefRecord extRecord = NdefRecord.createExternal(domain, type,
payload);
```

手动创建NdefRecord:

```
byte[] payload;
...
NdefRecord extRecord = new NdefRecord(
    NdefRecord.TNF_EXTERNAL_TYPE, "com.example:externalType", new
byte[0], payload);
```

NDEF记录的意图筛选器是这样的：

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="vnd.android.nfc"
        android:host="ext"
        android:pathPrefix="/com.example:externalType" />
</intent-filter>
```

更通用的NFC标签要采用 TNF_EXTERNAL_TYPE，这样可以对安卓系统和非安卓系统的供电设备提供更好支持。

安卓应用程序记录

安卓4.0（API级别14）中，安卓应用程序记录（AAR）提供了强大支持，确保扫描NFC标签时应用程序已经启动。AAR包含了嵌入NDEF记录应用程序的 package名。可以把AAR添加到NDEF消息记录，因为安卓搜索AAR中的全部NDEF信息。如果找到AAR，就会基于AAR package名称开启应用程序。如果应用程序没有在设备上显示，谷歌播放会启动应用程序下载。

如果要防止其他应用程序筛选相同意图并潜在处理特定标签，使用AAR是很有效的。如果想处理活动级别的意图，请使用意图筛选器。

如果标签包含AAR，标签调度系统采用以下方式调度：

1. 尝试使用意图筛选器启动活动。如果与意向匹配的活动也匹配AAR，就启动这一活动。
2. 如果意图筛选器的活动不能匹配AAR，如果多个活动可以处理该意图，又或者没有活动能处理该意图，就启动AAR指定的应用程序。
3. 如果没有应用程序能开启AAR，就要去谷歌播放下载基于AAR的应用程序。

如果仍想筛选不包含AAR的扫描标签，您可以正常声明意图筛选器。如果应用程序倾向于不包含AAR的其他标签，这一做法非常有用。例如想要保证应用程序处理第三方部署的专有标签。请记住AAR针对安卓4.0或者更新版本，因此当部署标签时，可能要使用AAR和MIME类型组合以便支持最广泛设备。此外当部署NFC标签时，考虑要怎样写NFC标签以便支持大多数设备。可以通过定义相对独特的MIME类型或者URL，使得区分应用程序更容易。

安卓提供了简单API来创建AAR，`createApplicationRecord()`。所要做的只是在NdefMessage嵌入AAR。不想使用NdefMessage的第一条记录，除非AAR是在NdefMessage的唯一记录。这是因为安卓系统会检查NdefMessage的第一条记录以确定标签的MIME类型或URL。下面的代码显示了如何创建AAR：

```
NdefMessage msg = new NdefMessage(
    new NdefRecord[] {
        ...
    }
)
NdefRecord.createApplicationRecord("com.example.android.beam") }
```

其他NDEF设备消息

Android的光束可以简单的对等两款Android供电设备之间的数据交换。要束数据到另一个设备的应用程序必须在前台和设备接收的数据不能被锁定。当喜气洋洋的设备与接收设备不够紧密接触，喜气洋洋设备显示“梁”的UI触摸。然后，用户可以选择是否梁消息的接收设备。

你可以使你的应用程序Android的光束，通过调用这两种方法之一：

`·setNdefPushMessage()`：接受一个 `NdefMessage` 设置梁消息的。自动光束的消息时，两个设备足够接近接近。

`·setNdefPushMessageCallback()`：接受一个回调，其中包含 `createNdefMessage()` 束数据范围是当一个设备被称为。回调可以让你创建NDEF消息只在必要时。

活动只能推一次NDEF消息，所以`setNdefPushMessageCallback()` 优先超过`setNdefPushMessage()` 如果都设置了。使用Android的光束，必须满足以下一般原则：

- 必须在前台活动，喜气洋洋的数据。两台设备都必须有自己的屏幕解锁。

- 您必须封装的数据，在喜气洋洋的`NdefMessage` 对象。

·横梁数据接收的NFC设备必须支持 `com.android.npp` NDEF推协议或NFC论坛的SNEP（简单NDEF交换协议）。`com.android.npp`协议所需的设备API 9级的Android 2.3 API级别13（的Android 3.2）。`com.android.npp`和SNEP都需要API 14级的Android 4.0及更高版本。

要启用安卓Beam：

1. 创建一个包含要推入其他设备的`NdefRecord`的`NdefMessage`。
2. 用`NdefMessage`调用 `setNdefPushMessage()`或者调用`setNdefPushMessageCallback` 在活动`onCreate()`中传递`NfcAdapter.CreateNdefMessageCallback` 对象。这些方法需要至少一个关联安卓Beam的活动，并伴随着其他要激活的活动的可选列表。如果两个设备在范围内连接时，活动仅需要一直推动相同的NDEF消息，通常会使用`setNdefPushMessage()`。当应用程序关注当前内容并且想根据当前用户进程来推动NDEF消息时，就可以使用 `setNdefPushMessageCallback`。

下面的示例显示了简单的活动如何用`onCreate()`方法调用 `NfcAdapter.CreateNdefMessageCallback`。这一示例可以帮助你创建MIME记录：

```
package com.example.android.beam;

import android.app.Activity;
import android.content.Intent;
```

```

import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.nfc.NfcAdapter.CreateNdefMessageCallback;
import android.nfc.NfcEvent;
import android.os.Bundle;
import android.os.Parcelable;
import android.widget.TextView;
import android.widget.Toast;
import java.nio.charset.Charset;

public class Beam extends Activity implements
CreateNdefMessageCallback {
    NfcAdapter mNfcAdapter;
    TextView textView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView textView = (TextView) findViewById(R.id.textView);
        // Check for available NFC Adapter
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) {
            Toast.makeText(this, "NFC is not available",
Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        // Register callback
        mNfcAdapter.setNdefPushMessageCallback(this, this);
    }

    @Override
    public NdefMessage createNdefMessage(NfcEvent event) {
        String text = ("Beam me up, Android!\n\n" +
                      "Beam Time: " + System.currentTimeMillis());
        NdefMessage msg = new NdefMessage(
            new NdefRecord[] { createMime(
                "application/vnd.com.example.android.beam",
text.getBytes()) })
        /**
         * The Android Application Record (AAR) is commented out.
When a device
         * receives a push with an AAR in it, the application
specified in the AAR
         * is guaranteed to run. The AAR overrides the tag dispatch
system.
         * You can add it back in to guarantee that this
         * activity starts when receiving a beamed message. For
now, this code
         * uses the tag dispatch system.
        */
        //,NdefRecord.createApplicationRecord("com.example.android.beam")
        });
        return msg;
    }

    @Override

```

```

public void onResume() {
    super.onResume();
    // Check to see that the Activity started due to an Android
Beam
    if
    (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction()))
    {
        processIntent(getIntent());
    }
}

@Override
public void onNewIntent(Intent intent) {
    // onResume gets called after this to handle the intent
    getIntent(intent);
}

/**
 * Parses the NDEF Message from the intent and prints to the
TextView
 */
void processIntent(Intent intent) {
    textView = (TextView) findViewById(R.id.textView);
    Parcelable[] rawMsgs = intent.getParcelableArrayExtra(
        NfcAdapter.EXTRA_NDEF_MESSAGES);
    // only one message sent during the beam
    NdefMessage msg = (NdefMessage) rawMsgs[0];
    // record 0 contains the MIME type, record 1 is the AAR, if
present
    textView.setText(new
String(msg.getRecords()[0].getPayload()));
}
}

```

请注意此代码为AAR添加注释，您可以自行删除。如果启用了AAR，其指定的应用程序总会收到安卓Beam消息。如果应用程序不存在，谷歌播放会开始下载应用程序。因此下面的意图筛选器对于安卓4.0或者更新版本的设备在技术上并非必需：

```

<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="application/vnd.com.example.android.beam" />
</intent-filter>

```

采用这种意图筛选器，当扫描NFC标签或用 com.example.android.beam类型AAR接收安卓Beam，又或者NDEF格式化消息包含application/vnd.com.example.android.beam类型的MIME记录时，com.example.android.beam应用程序可以启动。

尽管**AAR**确保应用程序启动或者下载，仍然推荐意图过滤器，因为这样会启动你选择的活动，而不是启动**AAR**指定的包内主要活动。**AAR**没有活动级别层次。此外因为一些安卓设备不支持**AAR**，就应该嵌入**NDEF**消息的第一记录中的识别信息。请参阅[Creating Common Types of NDEF records](#)获得创建记录的更多信息。

来自“[index.php?title=NFC_Basics&oldid=13832](#)”

Advanced NFC

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： River

原文链

接：

<http://developer.android.com/guide/topics/connectivity/nfc/advanced-nfc.html>

目录

[[隐藏](#)]

[1 高级NFC](#)

- [1.1 使用支持的标签技术](#)
 - [1.1.1 使用标签技术以及ACTION_TECH_DISCOVERED intent](#)
 - [1.1.2 读写标签](#)
- [1.2 使用前台分发系统\(Foreground Dispatch System\)](#)

高级NFC

本文档介绍高级NFC技术，比如各种标签技术，写NFC标签，以及前景调度，它允许一个应用程序能够在优先控制intent(意图)即使其他的程序filter也指向同一个intent(意图)。

使用支持的标签技术

当与NFC标签和Android供电设备交互时，我们在标签上用以读和写的主要

格式是NDEF。当一个设备扫描到包含NDEF数据的标签，如果可能，Android会提供一个NdefMessage对象来解析信息并传输它。

然后，在一些情况下，当你扫描到的标签没有包含NDEF数据或者当NDEF数据无法映射到MIME类型或者URI，在这些情况下，你需要直接打开与标签的连接，并使用你自己的协议(使用原始字节)去读写。Android使用android.nfc.tech包为这些情况提供了通用的支持，详见Table1.你可以使用getTechList方法来确定标签支持的技术并使用android.nfc.tech提供的类来创建相应的标签技术对象。

Table1. 支持的标签技术

类	描述
TagTechnology	所有标签技术类必须实现的接口
NfcA	提供NFC-A(ISO 14443-3A)属性通道以及I/O操作
NfcB	提供NFC-B(ISO 14443-3B)属性通道以及I/O操作
NfcF	提供NFC-F(JIS 6319-4)属性通道以及I/O操作
NfcV	提供NFC-V(ISO 15693)属性通道以及I/O操作
IsoDep	提供ISO-DEP(ISO 14443-4)属性通道以及I/O操作

Ndef	提供NDEF数据通道以及在已经被转换为NDEF格式的NFC标签上的操作
NdefFormatable	为可能转换为NDEF格式的标签提供转换操作

以下标签技术并不需要被android供电设备支持。 Table2 可选支持标签技术

类	描述
MifareClassic	提供MIFARE Classic属性通道以及I/O操作，如果这个android设备支持MIFARE
MifareUltralight	提供MIFARE Ultralight属性通道以及I/O操作，如果这个android设备支持MIFARE

使用标签技术以及**ACTION_TECH_DISCOVERED intent**

当设备扫描到包含NDEF数据的标签，但不能被映射到MIME或者URI，标签分发系统会试着根据ACTION_TECH_DISCOVERED intent启动一个activity。 ACTION_TECH_DISCOVERED同时也用于当被扫描的标签包含的数据是非NDEF数据时。有了这个回调，如果标签分发系统无法解析标签数据，你还是可以直接使用标签上的数据，当使用标签信息时，基本步骤如下：

1. 为ACTION_TECH_DISCOVERED定义过滤器，指定你希望处理的标签技术，获取更多信息，详见 [Filtering for NFC intents](#). 通常情况下，当NDEF信息不能被映射成MIME类型或者URI，或者扫描到的标签不包

含NDEF数据时，标签分发系统会试着启动一个ACTION_TECH_DISCOVERED intent。更多关于这些如何定义，详见[Tag Dispatch System\(标签分发系统\)](#)

2. 当你的程序接收到intent，从intent中获取到标签对象：

```
Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
```

3. 通过调用android.nfc.tech中类的其中一个get工厂方法获取一个TagTechnology实例。在调用get工厂方法之前，你可以通过getTechList()方法获取标签支持的技术列表。比如，从标签获取一个MifareUltralight实例，如下：

```
MifareUltralight.get(intent.getParcelableExtra(NfcAdapter.EXTRA_TAG));
```

读写标签

读写标签涉及到从一个intent中获取标签以及打开与标签间的通信。你必须定义你自己的协议栈读写数据到标签，然而，请记住，当你直接使用标签时，你仍然可以读写NDEF数据。这个取决于你自己想怎样构建东西。以下例子演示了怎样使用MIFARE Ultralight标签：

```
package com.example.android.nfc;

import android.nfc.Tag;
import android.nfc.tech.MifareUltralight;
import android.util.Log;
import java.io.IOException;
import java.nio.charset.Charset;

public class MifareUltralightTagTester {

    private static final String TAG =
MifareUltralightTagTester.class.getSimpleName();

    public void writeTag(Tag tag, String tagText) {
        MifareUltralight ultralight = MifareUltralight.get(tag);
        try {
            ultralight.connect();
            ultralight.writePage(4,
"abcd".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(5,
"efgh".getBytes(Charset.forName("US-ASCII")));
            ultralight.writePage(6,
```

```

"ijkl".getBytes(Charset.forName("US-ASCII")));
    ultralight.writePage(7,
"mnop".getBytes(Charset.forName("US-ASCII")));
} catch (IOException e) {
    Log.e(TAG, "IOException while closing
MifareUltralight...", e);
} finally {
    try {
        ultralight.close();
    } catch (IOException e) {
        Log.e(TAG, "IOException while closing
MifareUltralight...", e);
    }
}

public String readTag(Tag tag) {
    MifareUltralight mifare = MifareUltralight.get(tag);
    try {
        mifare.connect();
        byte[] payload = mifare.readPages(4);
        return new String(payload, Charset.forName("US-ASCII"));
    } catch (IOException e) {
        Log.e(TAG, "IOException while writing MifareUltralight
message...", e);
    } finally {
        if (mifare != null) {
            try {
                mifare.close();
            } catch (IOException e) {
                Log.e(TAG, "Error closing tag...", e);
            }
        }
    }
    return null;
}
}
}

```

使用前台分发系统(Foreground Dispatch System)

前台分发系统允许一个activity拦截intent并且优先于其他处理同一个intent的activity。 使用这个系统涉及为Android系统构建一些数据结构以达到能被用来发送适合的intent到你的应用程序。 启动前台分发系统： 1.加入以下代码到你activity的onCreate()方法中：

1. 创建一个PendingIntent对象，如此Android系统扫描到标签时，则会填充找个对象。

```

PendingIntent pendingIntent = PendingIntent.getActivity(
    this, 0, new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);

```

2. 声明intent filter(意图过滤器)来处理你想拦截的intent。前台调度系统会检查指定的intent filter和当设备扫描到标签接收到的intent，如果匹配，你的应用程序则处理这个intent(意图)，如果不匹配，前台分发系统回退到意图分发系统(intent dispatch system).指定一个intent filter(意图过滤)和technology filter(技术过滤)的空数组,指定一个意图过滤器和技术的过滤器的null数组，并指定要过滤的回退到的TAG_DISCOVERED意图的所有标签。

以下代码片段处理了为NDEF_DISCOVERED的所有MIME类型，你应该只处理你需要得一个：

```
IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
try {
    ndef.addDataType("*/*");          /* Handles all MIME based
displays.                                         You should specify only the
ones that you need. */
}
catch (MalformedMimeTypeException e) {
    throw new RuntimeException("fail", e);
}
intentFiltersArray = new IntentFilter[] { ndef, };
```

3. 创建一个你程序想处理的标签技术数组，调用Object.class.getName()方法去获取你想支持的技术类。

2. 重写以下activity的声明周期回调函数以及添加启用和禁用前台分发系统的逻辑，当activity失去 onPause() 或者重获 onResume() 焦点。

enableForegroundDispatch() 方法必须在主线程并且当activity处于前置位置时被调用(调用onResume()来保证)。同时你也需要实现onNewIntent回调处理来自扫描到的标签数据。

```
public void onPause() {
    super.onPause();
    mAdapter.disableForegroundDispatch(this);
}

public void onResume() {
    super.onResume();
    mAdapter.enableForegroundDispatch(this, pendingIntent,
intentFiltersArray, techListsArray);
}
```

```
public void onNewIntent ( Intent intent ) {  
    Tag tagFromIntent =  
intent.getParcelableExtra ( NfcAdapter.EXTRA_TAG );  
    //do something with tagFromIntent  
}
```

详见API例子(Demos)中的完整例子[前台分发\(ForegroundDispatch\)](#)

来自“[index.php?title=Advanced_NFC&oldid=9549](#)”



Wi-Fi Direct

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Allan Zhang

完成时间： 8月8日

原文链

接：<http://developer.android.com/guide/topics/connectivity/wifip2p.html>

目录

[[隐藏](#)]

[1 Wi-Fi 直连](#)

- [1.1 API 概述](#)
- [1.2 创建一个Wi-Fi直连意图使用的广播接收器](#)
- [1.3 创建一个Wi-Fi直连的应用](#)
 - [1.3.1 初始化设置](#)
 - [1.3.2 发现对等设备](#)
 - [1.3.3 连接到设备](#)
 - [1.3.4 数据传输](#)

Wi-Fi 直连

Wi-Fi 直连技术允许已经配备了相应硬件并预装了Android 4.0(API level 14)或更后的操作系统的设备在不需要Wi-Fi中间热点的支持下通过Wi-Fi直接互联的技术。使用这些API,你可以发现和连接其他支持此技术的设备，然后以距离远超蓝牙连接技术且速度更快的方式进行通信。这项技术对于一些多用户共享资料，比如多用户联机游戏或者相片分享等应用非常有用。

Wi-Fi直连技术的API包含以下主要部分：

- 允许用户发现，请求然后连接对等设备的各种方法，定义在[WifiP2pManager](#)类中。
- 允许用户定义收到调用[WifiP2pManager](#)类中方法成功或失败的通知的监听器。当用户调用[WifiP2pManager](#)类中的方法时，每一个方法都可以收到一个以参数形式传过来的特定监听。
- 通知用户被Wi-Fi直连技术框架检测到的特定事件的意图，比如一个已丢掉的连接或者一个新的对等设备的发现等。

你经常会同时使用这三个主要组件的相关功能。例如，你可以为去调用[android.net.wifi.p2p.WifiP2pManager.ActionListener discoverPeers\(\)](#)方法而提供一个[WifiP2pManager.ActionListener](#)的监听器，这样以后你可以收到一个[ActionListener.onSuccess\(\)](#)或者[ActionListener.onFailure\(\)](#)方法的通知。一个[WIFI_P2P_PEERS_CHANGED_ACTION](#)意图同时也是当[android.net.wifi.p2p.WifiP2pManager.ActionListener discoverPeers\(\)](#)方法发现的对等设备列表发生改变时的一个广播。

API 概述

[WifiP2pManager](#)类提供了很多方法允许用户通过设备的Wi-Fi模块来进行交互，比如做一些如发现，连接其他对等设备的事情。下列的方法都是可以使用的：

表格1.Wi-Fi直连技术方法

方法名	详细描述
initialize()	通过Wi-Fi框架对应用来进行注册。这个方法必须在任何其他Wi-Fi直连方法使用之前调用。
connect()	开始一个拥有特定设置的设备的点对点连接。
cancelConnect()	取消任何一个正在进行的点对点组的连接。

requestConnectInfo()	获取一个设备的连接信息。
createGroup()	以当前设备为组拥有者来创建一个点对点连接组。
removeGroup()	移除当前的点对点连接组。
requestGroupInfo()	获取点对点连接组的信息。
discoverPeers()	初始化对等设备的发现。
requestPeers()	获取当前发现的对等设备列表。

[WifiP2pManager](#)的方法可以让你在一个监听器里传递参数，这样Wi-fi直连框架就可以通知给你的窗体这个方法调用的状态。可以被使用的监听器接口和使用监听器的相应的[WifiP2pManager](#)的方法的调用都将在下面这张表中有所描述：

表格 2. Wi-Fi直连监听器方法

监听器接口	相关联的方法
WifiP2pManager.ActionListener	connect(), cancelConnect(), createGroup(), removeGroup(), and discoverPeers()
WifiP2pManager.ChannelListener	initialize()
WifiP2pManager.ConnectionInfoListener	requestConnectInfo()

WifiP2pManager.GroupInfoListener	requestGroupInfo()
WifiP2pManager.PeerListListener	requestPeers()

Wi-Fi直连技术的API定义了一些当特定的Wi-Fi直连事件发生时作为广播的意图，比如说当一个新的对等设备被发现，或者一个设备的Wi-Fi状态的改变。你可以在你的应用里通过创建一个处理这些意图的广播接收器来注册去接收这些意图。

Table 3. Wi-Fi 直连意图

意图名称	详细描述
WIFI_P2P_CONNECTION_CHANGED_ACTION	当设备的Wi-Fi连接信息状态改变时候进行广播。
WIFI_P2P_PEERS_CHANGED_ACTION	当调用discoverPeers()方法的时候进行广播。在你的应用里处理此意图时，你通常会调用requestPeers()去获得对等设备列表的更新。
WIFI_P2P_STATE_CHANGED_ACTION	当设备的Wi-Fi 直连功能打开或关闭时进行广播。

WIFI_P2P_THIS_DEVICE_CHANGED_ACTION

当设备的详细信息改变的时候进行广播，比如设备的名称

创建一个**Wi-Fi**直连意图使用的广播接收器

一个广播接收器允许你接收由**android**系统发布的意图广播，这样你的应用就可以对那些你感兴趣的事件作出响应。创建一个基本的**Wi-Fi**直连意图使用的广播接收器的步骤如下：

1. 创建一个继承自**BroadcastReceiver**的类。对于类的构造，一般最常用的就是以**WifiP2pManager**, **WifiP2pManager.Channel**作为参数，同时这个广播接收器对应的窗体也将被注册进来。这个广播接收器可以像窗体发送更新或者在需要的时候可以访问**Wi-Fi**硬件或通信通道。
2. 在广播接收器里，处理**onReceive()**方法里你感兴趣的意图。执行接收到的意图的任何需要的动作。比如，广播接收器接收到一个**WIFI_P2P_PEERS_CHANGED_ACTION**的意图，你就要调用**requestPeers()**方法去获得当前发现的对等设备列表。

下面的代码展示了怎样去创建一个典型的广播接收器。广播接收器接收一个**WifiP2pManager**对象和一个窗体对象作为参数然后利用这两个类去处理接收到的意图的特定的动作需求。

```
/*
 * A BroadcastReceiver that notifies of important Wi-Fi p2p events.
 */
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

    private WifiP2pManager manager;
    private Channel channel;
    private MyWiFiActivity activity;

    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
                                       MyWiFiActivity activity) {
        super();
        this.manager = manager;
        this.channel = channel;
        this.activity = activity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
    }
}
```

```

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action))
{
    // Check to see if Wi-Fi is enabled and notify appropriate
activity
} else if
(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
    // Call WifiP2pManager.requestPeers() to get a list of
current peers
} else if
(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
    // Respond to new connection or disconnections
} else if
(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
    // Respond to this device's wifi state changing
}
}
}

```

创建一个**Wi-Fi**直连的应用

创建一个**Wi-Fi**直连的应用包括创建和注册一个广播接收器，发现其他设备，连接其他设备，然后传输数据等步骤。接下来的几个部分描述了怎么去做这些工作。

初始化设置

在使用**Wi-Fi**直连的API之前，你必须确保你的应用可以访问设备的硬件并且你的设备要支持**Wi-Fi**直连的通讯协议。如果**Wi-Fi**直连技术是 支持的，你可以获得一个**WifiP2pManager**的实例对象，然后创建并注册你的广播接收器，然后开始使用**Wi-Fi**直连的API方法。

1. 为设备的**Wi-Fi**硬件获取权限并在Android的清单文件中声明你的应用正确使用的最低SDK版本：

```

<uses-sdk android:minSdkVersion="14" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>

```

2. 检查设备是否支持**Wi-Fi**直连技术。一种好的解决办法是当你的广播接收器接收到一个**WIFI_P2P_STATE_CHANGED_ACTION**意图。通知你的窗体**Wi-Fi**直连的状态和相应的反应。

```

public void onReceive(Context context, Intent intent) {
    ...
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE,
-1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            // Wifi Direct is enabled
        } else {
            // Wi-Fi Direct is not enabled
        }
    }
}
...

```

3. 在你的窗体的onCreate()方法里，获得一个WifiP2pManager的实例并调用initialize()方法通过Wi-Fi直连框架去注册你的应用。这个方法返回一个WifiP2pManager.Channel对象，是被用来连接你的应用和Wi-Fi直连框架的。你应该再创建一个以WifiP2pManager和WifiP2pManager.Channel为参数且关联你的窗体的广播接收器的实例。这样你的广播接收器就可以接收到你感兴趣的事件去通知你的窗体并更新它。它还可以让你在需要的时候操纵设备的Wi-Fi状态。

```

WifiP2pManager mManager;
Channel mChannel;
BroadcastReceiver mReceiver;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mManager = (WifiP2pManager)
getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel = mManager.initialize(this, getMainLooper(), null);
    mReceiver = new WiFiDirectBroadcastReceiver(manager, channel, this);
}
...
```

4. 创建一个意图过滤器并把它添加在你的广播接收器需要处理的意图上。

```

IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
}
```

```
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION
;
...
}
```

5. 注册你的广播接收器在窗体的onResume()方法，解除注册在onPause()方法中。

```
/* register the broadcast receiver with the intent values to be matched */
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mReceiver, mIntentFilter);
}
/* unregister the broadcast receiver */
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mReceiver);
}
```

当你获取到一个WifiP2pManager.Channel对象并且设置好你的广播接收器时，你的应用就可以调用Wi-Fi直连的方法并且可以接收Wi-Fi直连的意图。

你可以现在就通过调用WifiP2pManager中的方法去实现你的应用体验Wi-Fi直连技术的特性了。下面的章节描述了怎样去实现一些常用的操作，比如说发现其他设备和连接它们。

发现对等设备

要发现可以使用并连接的对等设备，调用discoverPeers()方法去检测在范围内的可使用设备。这个方法的调用是异步的同时如果你创建了一个WifiP2pManager.ActionListener监听器的话你会通过onSuccess()或者onFailure()方法收到发现成功或失败的消息。onSuccess()方法只能通知你发现的过程是否成功而不能提供任何关于发现设备的信息：

```
manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        ...
    }
    @Override
    public void onFailure(int reasonCode) {
        ...
    }
});
```

```

} );
}
```

如果发现过程成功且检测到了对等设备，系统将会广播出一个WIFI_P2P_PEERS_CHANGED_ACTION意图，这样你就可以利用广播监听器监听并获得发现设备的列表。当你的应用接收到了WIFI_P2P_PEERS_CHANGED_ACTION意图时，你就可以调用requestPeers()方法来获取发现设备的列表，代码如下：

```

PeerListListener myPeerListListener;
...
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
    // request available peers from the wifi p2p manager. This is an
    // asynchronous call and the calling activity is notified with a
    // callback on PeerListListener.onPeersAvailable()
    if (manager != null) {
        manager.requestPeers(channel, myPeerListListener);
    }
}
```

连接到设备

当你已经找到你要连接的设备在获得发现设备列表之后，调用connect()方法去连接指定设备。这个方法的调用需要一个包含待连接设备信息的WifiP2pConfig对象。你可以通过WifiP2pManager.ActionListener接收到连接是否成功的通知。下面的代码展示了怎样去连接一个想得到的连接：

```

//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
manager.connect(channel, config, new ActionListener() {
    @Override
    public void onSuccess() {
        //success logic
    }
    @Override
    public void onFailure(int reason) {
        //failure logic
    }
});
```

数据传输

一旦连接已经建立，你可以通过套接字来进行数据的传输。基本的数据传输步骤如下：

1. 创建一个**ServerSocket**对象。这个服务端套接字对象等待一个来自指定地址和端口的客户端的连接且阻塞线程直到连接发生，所以把它建立在一个后台线程里。
2. 创建一个客户端**Socket**.这个客户端套接字对象使用指定ip地址和端口去连接服务端设备。
3. 从客户端给服务端发送数据。当客户端成功连接服务端设备后，你可以通过字节流从客户端给服务端发送数据。
4. 服务端等待客户端的连接(使用**accept()**方法)。这个调用阻塞服务端线程直到客户端连接上，所以叫这个过程一个新的线程。当连接建立时，服务端可以接受来自客户端的数据。执行关于数据的任何动作，比如保存数据或者展示给用户。

下来的例子，从Wi-Fi直连示例改编而来，展示了怎样去创建服务端和客户端的连接和通信，并且使用一个客户端到服务端的服务来传输了一张JPEG图像。如需要得到一个完整的商业例子，请移步Wi-Fi直连示例。

```

public static class FileServerAsyncTask extends AsyncTask {
    private Context context;
    private TextView statusText;

    public FileServerAsyncTask(Context context, View statusText) {
        this.context = context;
        this.statusText = (TextView) statusText;
    }

    @Override
    protected String doInBackground(Void... params) {
        try {
            /**
             * Create a server socket and wait for client connections.
             * This call blocks until a connection is accepted from a client
             */
            ServerSocket serverSocket = new ServerSocket(8888);
            Socket client = serverSocket.accept();

            /**
             * If this code is reached, a client has connected and
             transferred data
             * Save the input stream from the client as a JPEG file
             */
            final File f = new
File(Environment.getExternalStorageDirectory() + "/"
+ context.getPackageName() + "/wifip2pshared-" +
System.currentTimeMillis()
+ ".jpg");
        }
    }
}

```

```

        File dirs = new File(f.getParent());
        if (!dirs.exists())
            dirs.mkdirs();
        f.createNewFile();
        InputStream inputStream = client.getInputStream();
        copyFile(inputStream, new FileOutputStream(f));
        serverSocket.close();
        return f.getAbsolutePath();
    } catch (IOException e) {
        Log.e(WiFiDirectActivity.TAG, e.getMessage());
        return null;
    }
}

/**
 * Start activity that can handle the JPEG image
 */
@Override
protected void onPostExecute(String result) {
    if (result != null) {
        statusText.setText("File copied - " + result);
        Intent intent = new Intent();
        intent.setAction(android.content.Intent.ACTION_VIEW);
        intent.setDataAndType(Uri.parse("file://" + result),
"image/*");
        context.startActivity(intent);
    }
}
}

```

在客户端，使用客户端套接字连接服务端套接字并传输数据。这个例子从客户端的文件系统里传输了一张JPEG的图像到服务端。

```

Context context = this.getApplicationContext();
String host;
int port;
int len;
Socket socket = new Socket();
byte buf[] = new byte[1024];
...
try {
    /**
     * Create a client socket with the host,
     * port, and timeout information.
     */
    socket.bind(null);
    socket.connect((new InetSocketAddress(host, port)), 500);

    /**
     * Create a byte stream from a JPEG file and pipe it to the output
     * stream
     * of the socket. This data will be retrieved by the server device.
     */
    OutputStream outputStream = socket.getOutputStream();
    ContentResolver cr = context.getContentResolver();
    InputStream inputStream = null;
    inputStream = cr.openInputStream(Uri.parse("path/to/picture.jpg"));
    while ((len = inputStream.read(buf)) != -1) {
        outputStream.write(buf, 0, len);
    }
}

```

```
        }
        outputStream.close();
        inputStream.close();
    } catch (FileNotFoundException e) {
        //catch logic
    } catch (IOException e) {
        //catch logic
    }

    /**
     * Clean up any open sockets when done
     * transferring or if an exception occurred.
     */
finally {
    if (socket != null) {
        if (socket.isConnected()) {
            try {
                socket.close();
            } catch (IOException e) {
                //catch logic
            }
        }
    }
}
```

来自“[index.php?title=Wi-Fi_Direct&oldid=7586](#)”



USB

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/usb/index.html>

编辑者：流风而逝

更新时间：

USB主机和配件

Android通过两种模式来支持各种USB外设和Android USB配件（指那些符合Android附件协议的硬件）：USB配件和USB主机。在USB配件的模式之下，外部的USB配件就像USB主机那样。这种配件可以包括机器人控制器，基站连接器，医疗和音乐设备，电话亭以及读卡器这样很多的设备。这样就使得那些搭载Android系统的设备不需要具备主机的特性就可以和USB硬件进行交互。Android USB配件是指那些专门用来为搭载Android系统的设备工作以及符合[Android附件通信协议](#)的设备。在USB主机的模式之下，搭载Android的设备就像主机那样工作。这些设备包括数码相机，键盘，鼠标以及游戏控制器。Android USB设备被设计成具有广泛的应用领域，可以很好的完成人机互动应用的通信设备。



图1.USB主机和配件模式

图1就显示了这两种模式的区别。当搭载Android系统的设备处于主机的模式下，它就充当USB主机并且为总线提供能源。而当搭载Android系统的设备处于USB配件的模式下时，连接的USB硬件（这种情况下，指的是一个Android USB配件）作为主机一样并且为总线提供能源。

在Android3.1（API12级）或较新的平台直接支持USB配件和主机模

式。USB配件模式以一个附加的类库的方式支持范围更广的设备 被移植到Android 2.3.4(API10级)。设备生产商可以决定是否在系统镜像上附加这个类库。

注意:支持USB主机和配件模式主要取决于设备的硬件,而不是平台的等级。你可以通过一个[[<uses-feature>]]元素来为设备进行过滤以支持USB主机和配件。看这个[USB配件](#)和[主机](#)文档来了解更多的详情。

调试注意事项

当用USB主机或者配件调试应用程序时,你最好有连接到搭载Android程序的设备的USB硬件。这样可以避免你要通过USB来为搭载Android的设备建立一个adb的连接。你可以在一个网络连接中一直连着adb。确保adb在一个网络连接的方式:

1. 通过USB连接搭载Android系统的设备和你的电脑
2. 在命令提示符中找到你的SDK platform_tools/ 目录, 输入 adb tcpip 5555
3. 输入 adb connect <device-ip-address>: 5555 你应该已经连接到了搭载Android程序的设备并且能够发出像 adb logcat 这样一般的adb命令
4. 在USB上为你的设备设置一个监听, 输入 adb usb

来自 "[index.php?title=USB&oldid=2927](#)"

Accessory

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

[[隐藏](#)]

[1 USB配件](#)

- [1.1 选择正确的USB附件APIs](#)
 - [1.1.1 安装谷歌APIs的附加类库](#)
- [1.2 API 概述](#)
 - [1.2.1 关于平台APIs和附加类库之间的用法差异](#)
- [1.3 Android ManiFest 需求](#)
 - [1.3.1 清单和资源文件例子](#)
- [1.4 用配件工作](#)
 - [1.4.1 发现配件](#)
 - [1.4.1.1 使用一个意图过滤器](#)
 - [1.4.1.2 枚举所有配件](#)
 - [1.4.2 获得使用一个配件的权限](#)
 - [1.4.3 和配件之间的“交流”](#)
 - [1.4.4 中止和配件的“交流”](#)

USB配件

USB配件模式允许用户连接那些专门搭载Android设备的USB主机硬件。这些配件必须遵守[Android配件开发工具包](#)文档中所列出的Android附件协议。这使得搭载Android系统的设备在不充当USB主机的情况下,仍然可以和USB硬件进行交互。当一台搭载Android系统的设备处于USB配件模式时,所依附的Android USB配件作为主机为USB总线提供能源以及列举出相连的设备。Android3.1(API12级)提供了USB配件模式并且这一特点也继承了Android2.3.4(API10级)以此来支持更多设备。

选择正确的USB附件APIs

尽管USB附件API在Android3.1平台才开始介绍，但是也可以在Android2.3.4API中通过附加类库使用。因为这些APIs都是通过额外的类库来使用的，你可以导入两个包来支持USB配件模式。取决于你想支持什么样的搭载Android系统的设备，你也许不得不在一个的基础上使用另外一个：

`com.android.feature.usb`: 为了支持Android2.3.4的USB配件模式, [Google APIs附加类库](#)包括了USB外设APIs并且它们就是包含在这个命名空间的后面。Android3.1还支持导入和调用这个命名空间的类来支持附加类库编写的应用程序。这个附加的类库只是关于[android.hardware.usb](#)外设APIs的一个简单的封装并且它不支持USB主机模式。如果你希望更大范围支持USB配件模式的设备，使用附加类库并且导入改包就行。需要注意的是，并不是所有搭载Android2.3.4的设备都需要拥有USB外设这一特色。每个设备生产商在决定是否具有这个特色，这也就是为什么你必须要在 manifest文件中声明的原因了。

[android.hardware.usb](#): 这个命名空间包含在Android3.1版本中支持USB附件模式的类。因为这个包是框架APIs中的一部分，所以Android3.1版本可以在不用附加类库的前提下支持USB附件模式。使用这个包时，如果你只关心Android3.1或者更新的支持USB附件模式的硬件的设备，你可以在 manifest文件中进行声明。

安装谷歌APIs的附加类库

如果你想安装这个附加类库，你可以通过在SDK管理器上面安装谷歌APIs中的Android API10包的方式来做。更多关于安装附加类库的信息请参见[安装谷歌APIs附加元件](#)。

API 概述

因为附加类库是一个框架APIs的封装，和那些支持USB附件功能的类是相似的。即使你在用附加类库的时候，你也可以用[android.hardware.usb](#)参考文档作为参考。

注意：然而，你要注意在附加类库和框架APIs之间还是有一些细微的[使用差异](#)

别的。

下面的表格为您描述了那些支持USB外设APIs的类：

类	详细描述
<u>UsbManager</u>	允许您用已连接的USB配件直接进行枚举和交流
<u>UsbAccessory</u>	可以表示一个USB配件并且包含来连接识别信息的方法

关于平台**APIs**和附加类库之间的用法差异

在分别使用谷歌**APIs**附加类库和平台**APIs**的时候，通常会有两种差异。

如果您正在使用附加类库，则肯定会通过下列方式来创建[UsbManager](#)对象：

```
UsbManager manager = UsbManager.getInstance(this);
```

如果您不是用的附加类，则必须通过下列方式来创建[UsbManager](#)对象：

```
UsbManager manager = (UsbManager)
getSystemService(Context.USB_SERVICE);
```

当您通过一个意图过滤器来过滤一个已经连接的配件，那么这个[UsbAccessory](#)对象就必须包含在传给您应用的这个意图中。如果您正在使用附加类库，您就必须通过下列方式来声明[UsbAccessory](#)对象：

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

如果您不是用的附加类，则必须通过下列方式来声明[UsbAccessory](#)对象：

```
UsbAccessory accessory = (UsbAccessory)
intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

Android Manifest 需求

下面的列表向您描述了在用USB配件APIs工作前需要在您应用中的manifest文件里面添加什么。下面的[清单和资源文件例子](#)将教您如何声明这些项：

- 因为并不是所有搭载Android系统的设备都保证支持USB配件APIs，包括一个`<uses-feature>`元素来声明您的应用具有

`android.hardware.usb.accessory`这个特色。

- 如果你正在使用[附加类库](#)，添加`<uses-library>`这个元素用来为类库特定说明`com.android.future.usb.accessory`。
- 如果您使用附加类库，那么您所设置的最低SDK版本是10级；如果您使用[android.hardware.usb](#)这个类的话，您所设置最低版本的SDK就应该为12级。
- 如果您希望您的应用带有一个附加USB配件的通知，在您的主activity中为`android.hardware.usb.action.USB_ACCESSORY_ATTACHED`这个意图指定一对`<intent-filter>`和`<meta-data>`元素。这个`<meta-data>`元素指向一个额外的声明关于你希望探测到的配件的识别信息的XML资源文件。

在这个XML资源文件中，为您希望过滤的配件声明`<usb-accessory>`元素。每一个`<usb-accessory>`都有下面的属性：

- 制造商
- 模式
- 版本

在`res/xml/`这个目录下保存资源文件。这个资源文件的名字（没有`.xml`的拓展名）必须和你在`<meta-data>`元素指定的一样。这个XML资源文件的格式在下面的例子中给出。

清单和资源文件例子

下面的例子就为您展示了一个简单的manifest以及与之相关的资源文件：

```
<manifest ...>
    <uses-feature android:name="android.hardware.usb.accessory" />
    <uses-sdk android:minSdkVersion="<version>" />
```

```

<application>
    <uses-library android:name="com.android.future.usb.accessory" />
        <activity ...>
            ...
            <intent-filter>
                <action
                    android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
                <meta-data
                    android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
                    android:resource="@xml/accessory_filter" />
            </activity>
        </application>
</manifest>

```

在这种情况下，下面的资源文件保存在`res/xml/accessory_filter.xml`文件中，并且指定那些只有与其相关的模式，制造商和版本的配件能够被选择。这个配件把这些属性传递给搭载Android系统的设备：

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-accessory model="DemoKit" manufacturer="Google"
version="1.0" />
</resources>

```

用配件工作

当用户将USB配件连接到搭载Android系统的设备上面时，Android系统会判断您的应用是否适用于已连接的该配件。如果适用，您就可以根据您的喜好为该设备建立连接。要这么做，您的应用必须做下面这些动作：

1. 您需要通过一个可以过滤配件附加事件的意图过滤器或者枚举已连接的配件来发现连接的配件来找到合适的接口。
2. 尚未获得许可的用户在适用配件操作时需要验证权限。
3. 通过在接入的端点进行读写数据的操作达到和配件交互的目的。

发现配件

您的应用可以通过两种方式来发现配件，一种是用一个意图过滤器在用户连接一个配件时对其进行通知，另一种则是通过枚举您已经连接的所有配件。如果您希望您的应用能够自动的探测到你想要的配件，请使用一个意

图过滤器来做。但是，如果您希望得到一个已连接配件的列表或者您不希望过滤意图，枚举所有的配件会是一个更好的选择。

使用一个意图过滤器

为了让您的应用可以发现一个特定的USB配件，您可以为`android.hardware.usb.action.USB_ACCESSORY_ATTACHED`这个意图指定一个意图来进行过滤。伴随着这个意图过滤器，您需要指定一个资源文件来特别说明这个USB配件的属性，例如制造商，模式和版本。当用户连接到一个符合您配件过滤条件的配件时，

下面的例子告诉您该如何声明这个意图过滤器：

```
<activity ...>
    ...
    <intent-filter>
        <action
            android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
    </intent-filter>

    <meta-data
        android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_filter" />
</activity>
```

下面的例子告诉您怎么样声明指定您希望连接的USB配件的相关资源文件：

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

<resources>
    <usb-accessory manufacturer="Google, Inc." model="DemoKit"
version="1.0" />
</resources>
```

在您的activity文件中，您可以从像这样的意图（有附加类的）中获取[UsbAccessory](#)来代表这个相关的配件：

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

或者像这样（用平台APIs的）：

```
UsbAccessory accessory =
```

```
(UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

枚举所有配件

您可以使您的应用在运行时列举出所有能够被识别的配件。

通过[getAccessoryList\(\)](#)方法来获得一个包含所有已连接USB配件的数组：

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
UsbAccessory[] accessoryList = manager.getAccessoryList();
```

注意:现在，只能一次连接一个USB配件操作，但是在以后的API中会支持多配件的操作的。

获得使用一个配件的权限

在您使用一个配件前，您的应用必须从用户那里获得权限。

注意:如果您的应用在连接配件时通过一个意图过滤器来发现它们，如果用户允许您的应用来处理这个意图，它将自动接收这个权限。如果用户不允许，那么您就必须在连接配件之前详细在您的应用中写明需要请求的权限。

在某些情况下很有必要明确权限的许可要求，例如当您的应用枚举出所有已经连接的配件并且您希望和其中的一个进行“交流”。您必须在和该配件“交流”前检查是否有连接该配件的权限。如果不是这样，您的应用将在用户拒绝您连接该配件的权限之后收到个运行错误。

为了确切地获得权限，首先需要创建个广播接收器。这个接收器在您调用[requestPermission\(\)](#)这个方法时从您得到的广播中舰艇这个意图。通过调用[requestPermission\(\)](#)这个方法为用户跳出一个是否连接配件的对话框。下面的例子告诉您如何创建一个广播接收器：

```

private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new
BroadcastReceiver() {

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action)) {
            synchronized (this) {
                UsbAccessory accessory = (UsbAccessory)
intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);

                if
(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false))
{
                    if(accessory != null){
                        //call method to set up accessory
communication
                    }
                    else {
                        Log.d(TAG, "permission denied for accessory " +
accessory);
                    }
                }
            }
        }
    }
};

```

为了注册您的广播接收器，将其放在您activity中的`onCreate()`方法中去：

```

UsbManager mUsbManager = (UsbManager)
getSystemService(Context.USB_SERVICE);
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
...
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new
Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);

```

当您需要展示征求用户同意连接这个配件的权限的对话框时，调用`requestPermission()`这个方法：

```

UsbAccessory accessory;
...
mUsbManager.requestPermission(accessory, mPermissionIntent);

```

当用户回应这个对话框时，你的广播接收器就会收到一个包含用一个`boolean`值来表示结果的`EXTRA_PERMISSION_GRANTED`字段的意图。在

您连接配件之前检查这个字段的值是否为true。

和配件之间的“交流”

您可以通过使用[UsbManager](#)这个类和配件进行“交流”，通过这个类可以获得一个文件描述符，然后您可以利用这个描述符来设置输入和输出流来读取和写入数据。这些流用来代表输入和输出的批量端点。您最好另起一个线程来让您的设备和配件进行“交流”，因为这样您就可以不需要将主线程锁起来了。下面的例子告诉您该如何和一个配件进行“交流”：

```
UsbAccessory mAccessory;
ParcelFileDescriptor mFileDescriptor;
FileInputStream mInputStream;
FileOutputStream mOutputStream;

...
private void openAccessory() {
    Log.d(TAG, "openAccessory: " + accessory);
    mFileDescriptor = mUsbManager.openAccessory(mAccessory);
    if (mFileDescriptor != null) {
        FileDescriptor fd = mFileDescriptor.getFileDescriptor();
        mInputStream = new FileInputStream(fd);
        mOutputStream = new FileOutputStream(fd);
        Thread thread = new Thread(null, this, "AccessoryThread");
        thread.start();
    }
}
```

在这个线程的run()方法中，您可以通过[FileInputStream](#)或者[FileOutputStream](#)对象来对配件进行读取和写出数据操作。当您通过[FileInputStream](#)对象读取配件中的数据时，请确保您所使用的缓存能够存储下USB数据包数据。Android配件协议支持数据包支持高达16384字节的数据包缓存区，所以您可以选择一直让您的缓存区是这个简单的大小。

注意:在一个比较低的水平下，64字节的数据包是全速配件以及512字节的数据包是高速配件。Android配件协议将这两种速度捆绑成一个简单的逻辑数据包。

想要知道更多如何使用Android中多线程的信息，请参见[进程和线程](#)。

“ ”

中止和配件的 交流

当您在完成和配件的“交流”之后，又或者该配件被移除了，通过调用[close\(\)](#)方法来关闭你已经打开的文件描述符。为了监听分离这样的事件，您需要创建一个如下的广播接收器：

```
 BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {
            UsbAccessory accessory =
            (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
            if (accessory != null) {
                // call your method that cleans up and closes
                communication with the accessory
            }
        }
    }
};
```

在您的应用中创建这个广播接收器，不是在manifest文件中，允许您的应用只能在它运行的时候处理这样的配件分离事件。这样的话，配件分离这个事件就只向正在运行的应用广播，而不是向所有的应用进行广播。

来自“[index.php?title=Accessory&oldid=11388](#)”



Host

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

[\[隐藏\]](#)

1 USB主机

- [1.1 文档内容](#)
- [1.2 相关例子](#)
- [1.3 API概述](#)
- [1.4 Android中manifest文件的需求](#)
 - [1.4.1 Manifest文件和资源文件的例子](#)
- [1.5 用配件工作](#)
 - [1.5.1 发现设备](#)
 - [1.5.1.1 使用一个意图过滤器](#)
 - [1.5.1.2 枚举所有配件](#)
 - [1.5.2 获得使用一个配件的权限](#)
 - [1.5.3 和设备之间的“交流”](#)
 - [1.5.4 中止和设备的“交流”](#)

USB主机

当您搭载Android系统的设备处于USB主机模式时，它就像一个USB主机，为总线提供能源，并且列举出所有已经连接上的设备。在Android 3.1或者更高的版本中支持USB主机模式。

API概述

文档内容

- [API概述](#)
- [Android中manifest文件需求](#)
- [工作的设备](#)
 - [发现设备](#)
 - [获得和设备进行“交流”的权](#)

在您开始之前，有个很重要的一点就是您必须对将要用到的类有个了解。下面的表格就向您描述了在[android.hardware.usb](#)这个包下USB主机APIs的一些特点。

表1.USB主机APIs

- 限**
- 和设备进行“交流”
 - 中止和设备的“交流”

相关例子

- [Adb测试用例](#)
- [相关链接](#)

Class/Interface	Description
UsbManager	允许您枚举已连接的USB设备并且与其进行“交流”。
UsbDevice	代表了一个已连接的USB的设备并且包含具有该设备验证信息，接口和接入点的方法。
UsbInterface	代表了一个USB设备的一个接口，该接口定义了一系列关于设备的函数。一个设备在进行“交流”的时候可以有一个或者多个接口。
UsbEndpoint	代表一个接口的接入点，该接入点就是这个接口的通信信道。一个接口可以有一个或者多个这样的接入点，而且一般都是有输入和输出双向通信的接入点。
UsbDeviceConnection	代表该设备的一个连接，用来在接入点上传输数据。这个类允许您能用同步或者异步的方式发送和返回数据。

UsbRequest	在通过 UsbDeviceConnection 和设备进行“交流”的一个异步请求。
UsbConstants	关于在linux内核中linux/usb/ch9.h的相关定义的USB常量。

在大多数的情况之下，在和一个USB设备进行“交流”时，上面这些类都需要用到（[UsbRequest](#)这个类只有在您做异步通信的时候才会用到）。一般来说，您可以通过查询要操作的[UsbDevice](#)来获得一个[UsbManager](#)。当您有这个设备时，您需要找到正确的[UsbInterface](#)以及和这个接口所对应的[UsbEndpoint](#)来进行和设备的“交流”。一旦您获得了正确的接入点，打开[UsbDeviceConnection](#)来和该USB设备进行“交流”。

Android中manifest文件的需求

下面的列表就是描述您应该在用USB主机APIs之前应该在您的应用中的manifest文件中添加些什么：

- 因为不是所有的搭载Android系统的设备都能保证支持USB主机的APIs，不能包含那个声明您的应用使用`android.hardware.usb.host`这一特点的`android.hardware.usb.host`的这一元素。
- 设置您的应用的最低的SDK版本在12级或者更高。这个USB主机APIs不在更前面的版本之中。
- 如果您希望您的应用能够被连接的USB设备所提示，只要在您的主activity中在`<intent-filter>`和`<meta-data>`元素对中添加一个`android.hardware.usb.action.USB_DEVICE_ATTACHED`意图。`<meta-data>`元素指向一个额外的XML资源文件，该文件是用来声明验证您希望探测到的设备的验证信息。

在这个XML资源文件中，为您希望过滤的USB设备声明`<usb-device>`元素。下面的列表描述`<usb-device>`的属性。一般来说，如果您想为一个特定的设备过滤就使用该产品的供应商和产品ID，如果您希望为一组USB设备，例如大量存储设备或者是数码相机来进行过滤那么就应该用类，子类

和协议。您可以不指定这些属性，也可以指定所有的属性。不为每个设备指定属性，只有在您的应用需要它时才这么做（这句话翻译的一点问题^_^）：

- 供应商ID
- 产品ID
- 类
- 子类
- 协议（设备或者借口）

将您的资源文件保存到`res/xml/`目录下。资源文件名（不包含`.xml`的扩展名）必须和您在`<meta-data>`元素中指明的那个名字。在下面的例子中是这个XML资源文件的格式。

Manifest文件和资源文件的例子

下面的例子告诉您一个manifest文件以及与它相关资源文件的例子：

```

<manifest ...>
    <uses-feature android:name="android.hardware.usb.host" />
    <uses-sdk android:minSdkVersion="12" />
    ...
    <application>
        <activity ...>
            ...
            <intent-filter>
                <action
                    android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
                <meta-data
                    android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
                    android:resource="@xml/device_filter" />
            </activity>
        </application>
    </manifest>

```

在这种情况下，下面的资源文件应该被保存在`res/xml/device_filter.xml`来确保找到那些特定符合您要求属性的USB设备：

```

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

<resources>
    <usb-device vendor-id="1234" product-id="5678" class="255"
    subclass="66" protocol="1" />
</resources>

```

用配件工作

当用户将USB配件连接到搭载Android系统的设备上面时，Android系统会判断您的应用是否适用于已连接的该配件。如果适用，您就可以根据您的喜好为该设备建立连接。要这么做，您的应用必须做下面这些动作：

1. 您需要通过一个可以过滤USB设备附加事件的意图过滤器或者枚举已连接的USB设备来发现连接的配件来找到合适的接口。
2. 尚未获得许可的用户在适用USB设备操作时需要验证权限。
3. 通过在接入的端点进行读写数据的操作达到和USB设备交互的目的。

发现设备

您的应用可以通过两种方式来发现USB设备，一种是用一个意图过滤器在用户连接一个设备时对其进行通知，另一种则是通过枚举您已经连接的所有USB设备。如果您希望您的应用能够自动的探测到你想要的设备，请使用一个意图过滤器来做。但是，如果您希望得到一个已连接设备的列表或者您不希望过滤意图，枚举所有的设备会是一个更好的选择。

使用一个意图过滤器

为了让您的应用可以发现一个特定的USB设备，您可以为`android.hardware.usb.action.USB_DEVICE_ATTACHED`这个意图指定一个意图来进行过滤。伴随着这个意图过滤器，您需要指定一个资源文件来特别说明这个USB设备的属性，例如供应商和产品ID。当用户连接到一个符合您配件过滤条件的配件时，这个系统会弹出一个对话框询问他们是否希望开始您的应用。如果用户同意，那么您的应用在失去连接之前会自动获取和设备连接的权限。

下面的例子告诉您该如何声明这个意图过滤器：

```
<activity ...>
...
<intent-filter>
    <action
        android:name = "android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>

<meta-data
    android:name = "android.hardware.usb.action.USB_DEVICE_ATTACHED"
```

```
        android:resource="@xml/device_filter" />
</activity>
```

下面的例子告诉您怎么样声明指定您希望连接的USB设备的相关资源文件：

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

<resources>
    <usb-device vendor-id="1234" product-id="5678" />
</resources>
```

在您的activity文件中，您可以从像这样的意图（有附加类的）中获取[UsbDevice](#)来代表这个相关的配件：

```
UsbDevice device = (UsbDevice)
intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
```

枚举所有配件

您可以使您的应用在运行时列举出所有能够被识别的USB设备。通过[getDeviceList\(\)](#)方法来获得一个包含所有已连接USB配件的数组：

```
UsbManager manager = (UsbManager)
getSystemService(Context.USB_SERVICE);
...
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
UsbDevice device = deviceList.get("deviceName");
```

如果您喜欢，您也可以一个接一个的从每一个设备的哈希图和过程中获取一个迭代器：

```
UsbManager manager = (UsbManager)
getSystemService(Context.USB_SERVICE);
...
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
while(deviceIterator.hasNext()){
    UsbDevice device = deviceIterator.next()
    //your code
}
```

获得使用一个配件的权限

在您使用一个USB设备前，您的应用必须从用户那里获得权限。

注意:如果您的应用在连接USB设备时通过一个意图过滤器来发现它们，如果用户允许您的应用来处理这个意图，它将自动接收这个权限。如果用户不允许，那么您就必须在连接设备之前详细在您的应用中写明需要请求的权限。

在某些情况下很有必要明确权限的许可要求，例如当您的应用枚举出所有已经连接的USB设备并且您希望和其中的一个进行“交流”。您必须在和该设备“交流”前检查是否有连接该设备的权限。如果不是这样，您的应用将在用户拒绝您连接该设备的权限之后收到个运行错误。

为了确切地获得权限，首先需要创建个广播接收器。这个接收器在您调用[requestPermission\(\)](#)这个方法时从您得到的广播中监听这个意图。通过调用[requestPermission\(\)](#)这个方法为用户跳出一个是否连接该设备的对话框。下面的例子告诉您如何创建一个广播接收器：

```
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new
BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action)) {
            synchronized (this) {
                UsbDevice device =
(UsbDevice) intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);

                if
(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false))
{
                    if(device != null){
                        //call method to set up device communication
                    }
                    else {
                        Log.d(TAG, "permission denied for device " +
device);
                    }
                }
            }
        }
    }
};
```

为了注册您的广播接收器，将其放在您activity中的`onCreate()`方法中去：

```
UsbManager mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
private static final String ACTION_USB_PERMISSION =
    "com.android.example.USB_PERMISSION";
...
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);
```

当您需要展示征求用户同意连接这个设备的权限的对话框时，调用[requestPermission\(\)](#)这个方法：

```
UsbDevice device;
...
mUsbManager.requestPermission(device, mPermissionIntent);
```

当用户回应这个对话框时，你的广播接收器就会收到一个包含用一个boolean值来表示结果的[EXTRA_PERMISSION_GRANTED](#)字段的意图。在您连接设备之前检查这个字段的值是否为true。

和设备之间的“交流”

我们可以同步或者异步的和USB设备进行“交流”。在任意一种情况之下，您都应该创建一个新的线程来进行数据传输，这样就不会阻塞您的主线程了。要想正确的设置好和一个设备之间的连接，您需要获得该设备正确的[UsbInterface](#)和[UsbEndpoint](#)来和您进行“交流”以及通过[UsbDeviceConnection](#)在这个接入点上发送请求。一般来说，您的代码应该这样：

- 检查一个[UsbDevice](#)对象的属性，例如产品ID，供应商ID，或者是关于设备的类，以此来确认您是否希望和该设备进行“交流”。
- 当您确信您希望和该设备进行“交流”时，找到关于该设备正确的[UsbInterface](#)以及和该接口所对应的[UsbEndpoint](#)。接口可以有一个或者多个接入点，而且一般都会有一个双向通信的输入和输出接入点。
- 当您找到正确的接入点时，在该接入点时打开一个[UsbDeviceConnection](#)。
- 您可以通过`bulkTransfer()`和`controlTransfer()`这两个方法在接入点上传

输您所需要传递的数据。您最好在另起一个新的线程来进行这个步骤以避免阻塞主线程。想要详细地了解关于Android中使用线程的信息，详见[线程和进程](#)。

下面的代码段是做同步数据传输的一个简单方式。您的代码应该有更多的逻辑来准确地找到和设备“交流”的接口和接入点，而且应该能够在不同于主线程的线程中能够传输任何的数据传输。

```
private Byte[] bytes
private static int TIMEOUT = 0;
private boolean forceClaim = true;

...
UsbInterface intf = device.getInterface(0);
UsbEndpoint endpoint = intf.getEndpoint(0);
UsbDeviceConnection connection = mUsbManager.openDevice(device);
connection.claimInterface(intf, forceClaim);
connection.bulkTransfer(endpoint, bytes, bytes.length, TIMEOUT);
//do in another thread
```

为了能够异步传输数据，使用[UsbRequest](#)类来[初始](#)和[队列化](#)一个异步请求，然后等待[requestWait\(\)](#)方法的结果。

想要了解更多地信息，请您参考[Adb Test sample](#)，这个参考将会告诉您如何进行异步批量传输，还有[MissleLauncher sample](#)将会告诉您如何异步监听一个中断端点。

中止和设备的“交流”

当您在完成和设备的“交流”之后，又或者该设备被移除了，通过调用[releaseInterface\(\)](#)和[close\(\)](#)的方法来关闭[UsbInterface](#)和[UsbDeviceConnection](#)。为了监听分离这样的事件，您需要创建一个如下的广播接收器：

```
BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action)) {
            UsbDevice device =
                (UsbDevice) intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
            if (device != null) {
                // call your method that cleans up and closes
                communication with the device
        }
    }
}
```

```
        }  
    } ;  
}
```

在您的应用中创建这个广播接收器，不是在manifest文件中，允许您的应用只能在它运行的时候处理这样的设备分离事件。这样的话，设备分离这个事件就只向正在运行的应用广播，而不是向所有的应用进行广播。

来自“[index.php?title=Host&oldid=5385](#)”



SIP

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：
址：

<http://developer.android.com/intl/ja/guide/topics/connectivity/sip.html>

翻译：tyutNo4

更新：2012.06.06

目录

[[隐藏](#)]

[1 会话发起协议](#)

- [1.1 条件和限制](#)
- [1.2 SIP API类和接口](#)
- [1.3 创建Manifest文件](#)
- [1.4 创建一个SipManager对象](#)
- [1.5 在SIP服务器上进行注册](#)
- [1.6 拨打一个语音电话](#)
- [1.7 接收呼叫](#)
 - [1.7.1 实现BroadcastReceiver的子类](#)
 - [1.7.1.1 创建一个用来接收呼叫的intent过滤器](#)
- [1.8 测试SIP应用程序](#)

会话发起协议

Android提供了一个支持会话发起协议（SIP）的API，这可以让你添加基于SIP的网络电话功能到你的应用程序。Android包括一个完整的SIP协议栈和集成的呼叫管理服务，让应用轻松无需管理会话和传输层的沟通就可设

置传出和传入的语音通话，或直接音频记录或播放。

以下类型的应用程序可能使用**SIP API**:

- 视频会议。
- 即时消息。

条件和限制

以下是开发一个**SIP**应用程序的条件:

- 你必须有一个运行**Android2.3**或者更高版本的移动设备。
- **SIP**是通过无线数据连接来运行的，所以你的设备必须有一个数据连接（通过移动数据服务或者**Wi-Fi**）。这意味着你不能在模拟器（**AVD**）上进行测试，只能在一个物理设备上测试。详情请参见应用程序测试（**Testing SIP Applications**）。
- 每一个参与者在应用程序的通信会话过程中必须有一个**SIP**账户。有很多不同的**SIP**服务提供商提供**SIP**账户。

SIP API类和接口

以下是**Android SIP API**中包含的一些类和一个接口（**SipRegistrationListener**）的概述:

类/接口	描述
SipAudioCall	通过 SIP 处理网络音频电话
SipAudioCall.Listener	关于 SIP 电话的事件监听器，比如接收到一个电话（ on ringing ）或者呼出一个电话（ on calling ）的时候
SipErrorCode	定义在 SIP 活动中返回的错误代码
SipManager	为 SIP 任务提供 APIs ，比如初始化一个 SIP 连接。提供相关 SIP 服务的访问。
SipProfile	定义了 SIP 的相关属性，包含 SIP 账

	户、域名和服务器信息
SipProfile.Builder	创建SipProfile的帮助类
SipSession	代表一个SIP会话，跟SIP对话框或者一个没有对话框的独立事务相关联
SipSession.Listener	关于SIP会话的事件监听器，比如注册一个会话 (on registering) 或者呼出一个电话 (on calling) 的时候
SipSession.State	定义SIP会话的声明，比如“注册”、“呼出电话”、“打入电话”
SipRegistrationListener	一个关于SIP注册事件监听器的接口

创建Manifest文件

如果你开发一个用到SIP API的应用程序，记住它需要Android2.3 (API9) 或者更高版本的平台的支持。所以在你的设备上要运行Android2.3 (API9) 或者更高的版本，并不是所有的设备都提供SIP的支持。

为了使用SIP，需要添加以下权限到你的manifest文件：

- android.permission.USE_SIP
- android.permission.INTERNET

为了确保你的应用程序能够安装到支持SIP的设备上，你需要添加以下内容到你应用程序的manifest文件里：

- <uses-sdk android:minSdkVersion="9" />. 这个设置表明你的应用程序需要Android2.3或者更高版本的平台。详情请参考API Levels和<uses-sdk>元素相关的文档。

为了控制你的应用程序被那些不支持SIP的设备过滤掉（比如：在Google Play），你需要添加以下内容到你应用程序的manifest文件里：

- <uses-feature android:name="android.hardware.sip.voip" />. 这个设置声明了你的应用程序用到了SIP API。这个声明还应该包含一个android:required 属性来表明你是否想让你的应用程序被那些不提

供SIP支持的设备过滤掉。其他<uses-feature>声明你也可能需要，具体取决于你的实现，详情请参考<uses-feature>元素相关的文档。

如果你的应用程序设计用来接受呼叫，那么你还必须在应用程序的manifest文件里定义一个接收器（BroadcastReceiver的子类）：

- <receiver android:name=".IncomingCallReceiver" android:label="Call Receiver"/>

以下是从SipDemo项目manifest文件中摘录的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.sip">
    ...
    <receiver android:name=".IncomingCallReceiver"
        android:label="Call Receiver"/>
    ...
    <uses-sdk android:minSdkVersion="9" />
    <uses-permission android:name="android.permission.USE_SIP" />
    <uses-permission android:name="android.permission.INTERNET" />
    ...
    <uses-feature android:name="android.hardware.sip.voip"
        android:required="true" />
    <uses-feature android:name="android.hardware.wifi"
        android:required="true" />
    <uses-feature android:name="android.hardware.microphone"
        android:required="true" />
</manifest>
```

创建一个SipManager对象

要想使用SIP API，你的应用程序需要创建一个SipManager对象，这个SipManager对象在你的应用程序里负责以下内容：

- 发起SIP会话
- 发起和接受呼叫
- 在SIP provider里进行注册和注销
- 验证会话的连通性

你可以像下面一样实例化一个新的SipManager对象：

```
public SipManager mSipManager = null;
...
if(mSipManager == null) {
```

```
mSipManager = SipManager.newInstance(this);
```

} 在**SIP**服务器上进行注册

一个典型的Android SIP应用中包含一个或多个用户，他们中的每个人都有一个**SIP**账户。在Android SIP应用中，每一个**SIP**账户代表一个**SipProfile**对象。

一个**SipProfile**对象定义了一个**SIP**的概要文件，包括**SIP**账户、域名和服务器信息。跟正在这个设备上运行应用的**SIP**账户相关联的概要文件被称之为本地配置文件。与会话相连接的概要文件被称之为对应配置文件。当你的**SIP**应用通过本地**SipProfile**登录到**SIP**服务器的时候，这就有效的注册当前设备为基站来发送**SIP**呼叫到你想呼叫的**SIP**地址。

本节展示了如何创建一个**SipProfile**，以及如何把刚创建的**SipProfile**注册到**SIP**服务器上，并且跟踪注册事件。你可以像以下一样创建一个**SipProfile**对象：

```
public SipProfile mSipProfile = null;
...
SipProfile.Builder builder = new SipProfile.Builder(username,
domain);
builder.setPassword(password);
mSipProfile = builder.build();
```

接下来的代码摘录本地配置文件，用于呼出电话和/或接收通用的**SIP**电话。呼叫器可以通过**mSipManager.makeAudioCall**来呼出后续电话。这段摘录同样设置了一个**android.SipDemo.INCOMING_CALL**行动，这个行动会被一个**intent**过滤器来使用，当前设备接收到一个呼叫（见**Setting up an intent filter to receive calls**）。以下是注册步骤：

```
Intent intent = new Intent();
intent.setAction("android.SipDemo.INCOMING_CALL");
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
intent, Intent.FILL_IN_DATA);
mSipManager.open(mSipProfile, pendingIntent, null);
```

最后这段代码在**SipManager**上设置了一个**SipRegistrationListener**监听器，

这个监听器会跟踪SipProfile是否成功的注册到你的SIP服务提供者。

```
mSipManager.setRegistrationListener(mSipProfile.getUriString(), new
SipRegistrationListener() {
    public void onRegistering(String localProfileUri) {
        updateStatus("Registering with SIP Server...");
    }

    public void onRegistrationDone(String localProfileUri, long expiryTime) {
        updateStatus("Ready");
    }

    public void onRegistrationFailed(String localProfileUri, int errorCode,
        String errorMessage) {
        updateStatus("Registration failed. Please check settings.");
    }
})
```

当你的应用程序使用完一个profile的时候，你应该关闭它来释放相关联的对象到内存中以及从服务器上注销当前设备。例如：

```
public void closeLocalProfile() {
    if (mSipManager == null) {
        return;
    }
    try {
        if (mSipProfile != null) {
            mSipManager.close(mSipProfile.getUriString());
        }
    } catch (Exception ee) {
        Log.d("WalkieTalkieActivity/onDestroy", "Failed to close
local profile.", ee);
    }
}
```

拨打一个语音电话

要想拨打一个语音电话，你需要准备如下条件：

- 一个发起呼叫电话的SipProfile对象（本地配置文件）和一个用来接收呼叫的有效的SIP地址（对应配置文件）
- 一个SipManager对象

要想拨打一个语音电话，你应该建立一个SipAudioCall.Listener监听器。大部分客户与SIP堆栈的交互都是通过监听器来发生的。在这一小段你将会看

到SipAudioCall.Listener监听器是如何在呼叫制定之后建立事务的：

```
SipAudioCall.Listener listener = new SipAudioCall.Listener() {
    @Override
    public void onCallEstablished(SipAudioCall call) {
        call.startAudio();
        call.setSpeakerMode(true);
        call.toggleMute();
        ...
    }
    @Override
    public void onCallEnded(SipAudioCall call) {
        // Do something.
    }
};
```

一旦你创建了这个SipAudioCall.Listener监听器，你就可以拨打电话了，SipManager对象里的makeAudioCall方法接受以下参数：

- 一个本地SIP配置文件（呼叫方）
- 一个相对应的SIP配置文件（被呼叫方）
- 一个用来监听从SipAudioCall发出的呼叫事件的SipAudioCall.Listener，这个参数可以为null，但是如上所说，一旦呼叫电话制定，这个监听器将被用来创建事务
- 超时的值，以秒为单位

例如：

```
call = mSipManager.makeAudioCall(mSipProfile.getUriString(),
    sipAddress, listener, 30);
```

接收呼叫

为了接收呼叫，SIP应用程序必须包含一个BroadcastReceiver的子类，这个子类得有能力响应一个表明有来电的intent。因此你需要在你的应用程序里做如下事情：

- 在AndroidManifest.xml文件中声明一个<receiver>元素。在SipDemo项目中，<receiver>元素是这样的<receiver android:name=".IncomingCallReceiver" android:label="Call"

Receiver"/>

- 实现BroadcastReceiver的子类，在SipDemo中，这个子类是IncomingCallReceiver
- 通过挂起一个intent来初始化本地配置文件（SipProfile），当有人呼叫你的时候，这个挂起的intent会调用你的接收器。
- 创建一个intent过滤器，这个过滤器通过标志着来电的行动来进行过滤。在SipDemo中，这个action是android.SipDemo.INCOMING_CALL。

实现BroadcastReceiver的子类

为了接收呼叫，你的SIP应用必须实现BroadcastReceiver的子类。

当Android系统接收到一个呼叫的时候，他会处理这个SIP 呼叫，然后广播一个来电intent（这个intent由系统来定义），以下是SipDemo中实现BroadcastReceiver子类的代码。如果想查看完整的例子，你可以去SipDemo Sample项目，这个项目在SDK的samples文件夹中。关于下载和安装SDK samples，请参考 Getting the Samples。

```
/** Listens for incoming SIP calls, intercepts and hands them off
to WalkieTalkieActivity.
*/
public class IncomingCallReceiver extends BroadcastReceiver {
    /**
     * Processes the incoming call, answers it, and hands it over
     * to the
     * @param context The context under which the receiver is
     * running.
     * @param intent The intent being received.
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        SipAudioCall incomingCall = null;
        try {
            SipAudioCall.Listener listener = new
SipAudioCall.Listener() {
                @Override
                public void onRinging(SipAudioCall call, SipProfile
caller) {
                    try {
                        call.answerCall(30);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            };
            WalkieTalkieActivity wtActivity =
(WalkieTalkieActivity) context;
            incomingCall =
wtActivity.mSipManager.takeAudioCall(intent, listener);
            incomingCall.answerCall(30);
        }
    }
}
```

```
        incomingCall.startAudio();
        incomingCall.setSpeakerMode(true);
        if( incomingCall.isMuted() ) {
            incomingCall.toggleMute();
        }
        wtActivity.call = incomingCall;
        wtActivity.updateStatus(incomingCall);
    } catch (Exception e) {
        if (incomingCall != null) {
            incomingCall.close();
        }
    }
}
```

创建一个用来接收呼叫的**intent**过滤器

当SIP服务接收到一个新的呼叫的时候，他会发送一个intent，这个intent会附带一个由应用程序提供的action。在SipDemo项目中，这个action是`android.SipDemo.INCOMING_CALL`。

以下从SipDemo中摘录的代码展示了如何通过挂起一个基于`android.SipDemo.INCOMING_CALL` action的intent来创建SipProfile对象的。PendingIntent对象将执行一个广播当SipProfile接收到一个呼叫的时候：

```
public SipManager mSipManager = null;
public SipProfile mSipProfile = null;
...
Intent intent = new Intent();
intent.setAction("android.SipDemo.INCOMING_CALL");
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
intent, Intent.FILL_IN_DATA);
mSipManager.open(mSipProfile, pendingIntent, null);
```

上面被执行的广播如果被intent过滤器拦截的话，这个intent过滤器将会启动声明过的 Receiver（IncomingCallReceiver）。你可以在你的应用程序里的manifest文件中指定一个intent过滤器，或者通过代码来指定一个intent过滤器，就像SipDemo项目中Activity中的onCreate（）方法一样：

```
public class WalkieTalkieActivity extends Activity implements  
View.OnTouchListener {  
...  
    public IncomingCallReceiver callReceiver;  
    ...
```

```

@Override
public void onCreate( Bundle savedInstanceState ) {
    IntentFilter filter = new IntentFilter();
    filter.addAction( "android.SipDemo.INCOMING_CALL" );
    callReceiver = new IncomingCallReceiver();
    this.registerReceiver( callReceiver, filter );
    ...
}
...
}

```

测试SIP应用程序

要测试SIP应用程序的话，你需要以下条件：

- 一个运行Android2.3或者更高版本的移动设备。SIP通过无线来运行，所以你必须在一个真正的设备上测试，在AVD上是测试是行不通的
- 一个SIP账户，有很多不同的提供SIP账户的SIP服务提供商。
- 如果你要打电话，这个电话必须是有效的SIP账户。

测试一个SIP应用程序的步骤：

- 让你的设备连接到无线（设置>无线&网络>Wi-Fi>Wi-Fi设置）
- 设置你的移动设备进行测试，就像在Developing on a Device里描述的一样
- 在你的移动设备上运行程序，就像在Developing on a Device里描述的一样
- 如果你正在使用Eclipse，你可以在Eclipse中查看应用程序的日志输出（**Window > Show View > Other > Android > LogCat**）。

来自“[index.php?title=SIP&oldid=11397](#)”



Text and Input

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

文本与输入

使用文本服务来使你的应用更加的方便，比如支持复制粘贴以及拼写检查。你也可以开发你自己的文本服务来提供订制的**IME**，词典以及拼写检查器并将这些当成应用推送给用户。

[更多 >](#)

目录

[\[隐藏\]](#)

[1 博客文章](#)

- [1.1 为你的IME加入语音输入](#)
- [1.2 Android 中语音输入的API](#)
- [1.3 理解多点触摸](#)

博客文章

[为你的IME加入语音输入](#)

Android 4.0的一个新特性在于语音输入：对于用户来说，不同的是识别结果会实时出现在文框，即便用户还在说话，如果你是一个**IME**开发者，你可以很轻易的将语音输入集成到应用中。

[Android 中语音输入的API](#)

我们认为语音会从根本上改变移动设备的体验。我们建议每一个**Android**应用开发者都考虑通过**Android SDK**，将语音输入功能集成进他们的应用中去。

理解多点触摸

多点触摸已经是广为人知了，而且人们对于这项技术的认知也一向不明确，对于一些人来说，这是硬件功能，对于其他人来说，这是软件中支持的特定手势。不管你怎样去称呼它，今天你将看到当屏幕上多个手指时，如何去使你的应用和界面看上去更好。

来自“[index.php?title=Text_and_Input&oldid=7481](#)”



Copy and Paste

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址: <http://developer.android.com/guide/topics/clipboard/copy-paste.html>

翻译者: [AlbertLi](#)

最后更新:2012-06-06

目录

[[隐藏](#)]

- [1 复制和粘贴 - Copy and Paste](#)
- [2 剪贴板框架](#)
- [3 剪贴板类](#)
 - [3.1 ClipboardManager](#)
 - [3.2 ClipData, ClipData.Item, ClipDescription](#)
 - [3.3 ClipData 的方法](#)
 - [3.4 强制剪贴板为文本数据](#)
- [4 复制到剪贴板](#)
- [5 从剪贴板粘贴](#)
 - [5.1 粘贴纯文本](#)
 - [5.2 从content URI粘贴数据](#)
 - [5.3 粘贴Intent](#)
- [6 通过content provider 复制复杂的数据](#)
 - [6.1 URI标识符编码](#)
 - [6.2 复制数据结构](#)
 - [6.3 复制数据流](#)
- [7 设计有效的复制/粘贴功能](#)

复制和粘贴 - Copy and Paste

Android提供了强大的基于框架的剪贴板的用于复制和粘贴。它支持简单和复杂的数据类型，包括文本字符串，结构复杂的数据，文本和二进制数据流，甚至是应用数据。简单的文本数据直接存储在剪贴板中，而复杂的数据以引用的方式粘贴到应用中并由content provider解析。应用程序内部和应用程序之间的复制和粘贴工作都使用这个框架。

由于剪贴板框架一部分基于content provider，因此本主题假设对Android Content Provider组件有一定了解，此部分在主题 [Content Provider](#) 有详细描述。

剪贴板框架

当你使用的剪贴板框架，你将把数据放入剪贴对象中，然后把剪贴对象放入系统的全局剪贴板中。剪贴对象可以是以下三种形式之一：

- **文本**

一个文本字符串。你把要剪切的字符串直接放入剪贴对象，之后将对象存入剪贴板框架中。要粘贴字符串，从剪贴板中取出剪贴对象，并复制字符串到你的应用中。

- **URI**

[Uri对象-Uri](#)，任何形式的URI。这主要是为复制content provider中的复杂数据。要复制数据，你把一个[Uri对象 - Uri](#) 放入到剪帖对象，并将剪贴对象存入剪贴板中。要粘贴数据，先取得剪贴对象，并取得[Uri对象-Uri](#)，解析它的数据源，比如使用content provider，从源头复制数据到应用中。

- **Intent**

[Intent对象-Intent](#)，支持复制应用程序的快捷方式。要复制数据，先创建一个Intent对象，然后放入剪贴对象，并存入到剪贴板中。要粘贴数据，先到的剪贴对象，然后将 [Intent对象 - Intent](#) 复制到您的应用程序中。

剪贴板每次只保存一个剪贴对象。当另一个应用程序放入剪贴对象到剪贴板上，以前的剪贴对象消失。

如果你想允许用户粘贴数据到您的应用程序中，你不必处理所有数据类型。在用户粘贴数据之前，检查数据类型，然后选择。除了某些的数据类型，剪贴对象也包含元数据，告诉你MIME类型或其他类型可供选择。如果你的应用程序使用剪贴板数据工作，此元数据可以帮助你决定数据中的类型。例如，如果你有一个应用程序，主要是处理文本，你可能想忽略剪贴对象包含的URI对象或Intent对象。

您可能还希望让用户粘贴文本时不必关心剪贴板上的数据的形式。要做到这一点，可以强制剪贴板数据转换为文本表示形式，然后粘贴这个文本。详细内容请看[强制剪贴板为文本数据](#)。

剪贴板类

本节描述剪贴板框架中的类。

ClipboardManager

在Android系统，系统剪贴板统一由[ClipboardManager](#)类表示。你不用直接实例化这个类，而是通过调用[getSystemService \(CLIPBOARD_SERVICE\)](#)取得。

ClipData, ClipData.Item, ClipDescription

要把数据添加到剪贴板，您必须创建一个包含数据描述和数据本身的[ClipData](#)对象。剪贴板每次只保存一个[ClipData](#)。一个[ClipData](#)包含一个[ClipDescription](#)对象，以及一个或多个[ClipData.Item](#)对象。

一个[ClipDescription](#)对象包含有关剪贴对象的元数据。另外，它包含剪贴的数据的可用MIME类型的数组。当你把剪贴数据放到剪贴板上，这个阵列是可粘贴到应用程序，并可检查看看是否有可用的MIME类型用于处理。

一个[ClipData.Item](#)对象包含的文字，URI或Intent 数据：

- 文本

一个[CharSequence](#)。

- URI

一个[URI](#)。一般是content provider 提供的URI，但允许任何形式的URI。应用程

序将带有数据的URI放到剪贴板上。应用程序要粘贴数据的话，从剪贴板中的URI获得，并用它来访问的content provider（或其他数据源）来检索数据。

- Intent

[Uri对象-Uri](#)。该数据类型允许将应用程序的快捷方式复制到剪贴板。然后用户可以将其粘贴到他们的应用程序中，以备后用。您可以添加多个[ClipData.Item](#)对象到剪贴对象当中。这允许用户复制并粘贴多个选择作为一个剪贴对象。例如，如果你有一个列表控件，允许用户一次选择多个项目，您可以只要复制所有项目到剪贴板一次。要做到这一点，你为每个列表项创建一个单独的[ClipData.Item](#)，然后添加所有的[ClipData.Item](#)对象到一个[ClipData](#)对象。

ClipData 的方法

[ClipData](#)类提供创建带单一[ClipData.Item](#)对象和一个简单[ClipDescription](#)对象的[ClipData](#)对象的静态方法：[newPlainText \(label, text\)](#)

返回带有包含一个文本字符串的[ClipData.Item](#)对象的[ClipData](#)对象。[ClipDescription](#)对象的标签设置label。[ClipDescription](#)的MIME类型指定为[MIMETYPE_TEXT_PLAIN](#)。

使用[newPlainText\(\)](#) 创建一个文本字符串的剪贴对象。

- [newUri \(resolver, label, URI\)](#)

返回包含一个URI的单一[ClipData.Item](#)对象的[ClipData](#)对象。[ClipDescription](#)对象的标签设置label。如果URI是content URI ([Uri.getScheme\(\)](#) retrun content:)，该方法使用解析[ContentResolver](#)中提供的解析器解析对象,然后从content provider中检索到可用的MIME类型并将它存储在[ClipDescription](#)，而URI并不是一个content: URI对象，这个方法只是设置MIME类型为[MIMETYPE_TEXT_URI_LIST](#)。使用 [newUri\(\)](#)从URI对象创建剪贴对象，其是一个content:URI。

- [newIntent \(label, intent\)](#)

返回包含一个 Intent 的单一[ClipData.Item](#)对象的[ClipData](#)对象。[ClipDescription](#)对象的标签设置label。MIME类型设置到[MIMETYPE_TEXT_URI_LIST](#)。使用 [newIntent\(\)](#)从一个Intent对象创建剪贴对

象。

强制剪贴板为文本数据

即使您的应用程序只处理文本，你也可以通过使用[ClipData.Item.coerceToText\(\)](#)进行转换来实现从剪贴板中复制非文本数据。

这种方法转换成数据[ClipData.Item](#)成文本并返回[CharSequence](#)。[ClipData.Item.coerceToText\(\)](#)返回数据的值为[ClipData.Item](#)的数据格式

- 文本

如果[ClipData.Item](#)是文本 ([getText\(\)](#)不为空) , [coerceToText\(\)](#)返回的文本。

- **URI**

如果[ClipData.Item](#)是一个URI ([getUri\(\)](#)不为空) , [coerceToText\(\)](#)尝试使用它作为一个content URI: 如果URI是一个content URI并且provider可以返回一个文本流中, [coerceToText\(\)](#)返回一个文本流。

如果URI是一个content URI, 但provider不提供文本流中, [coerceToText\(\)](#)返回的URI表示。这种表示和[Uri.toString\(\)](#)效果一样。如果URI不是一个content URI, [coerceToText\(\)](#)返回的URI表示。这种表示和[Uri.toString\(\)](#)效果一样。

- **Intent**

如果[ClipData.Item](#)是一个Intent ([getIntent\(\)](#)不为空) , [coerceToText\(\)](#)将其转换为一个Intent URI, 并返回它。这和[Intent.toUri \(URI_INTENT_SCHEME\)](#)效果一样。

如图1剪贴板框架总结。要数据复制, 应用程序把一个[ClipData.Item](#)对象放到全局剪贴板ClipboardManager的对象。[ClipData.Item](#)包含一个或多个[ClipData.Item](#)对象和[ClipDescription](#)的对象。要粘贴数据, 应用程序得到的ClipData, 从[ClipDescription](#)得到它的MIME类型, 之后, 要不从[ClipData.Item](#)的取得数据要不从[ClipData.Item](#)中的content provider取得。



图1。 Android剪贴板框架的

复制到剪贴板

如上所述，复制到剪贴板，你得到一个句柄到全局数据 `ClipboardManager` 对象，创建 `ClipData.Item` 对象，添加 `ClipDescription` 和一个或多个 `ClipData.Item` 对象，并添加此 `ClipData` 对象到 `ClipboardManager` 对象。以下是详细描述：

1. 如果您使用 content URI 复制数据，建立一个 content provider。记事本示例应用程序是就是使用 content provider 复制和粘贴数据的一个例子。`NotePadProvider` 类实现 content provider。`NotePad` 类定义了与 provider 和其他应用的协议，包括支持的 MIME 类型的。
2. 获取系统剪贴板：

```
...
//如果用户选择复制
case R.id.menu_copy:
//获取一个剪贴板服务句柄
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
```

3. 复制数据到一个新的 `ClipData.Item` 对象：

- 纯文本

```
//创建一个新的文本剪贴对象，放入剪贴板
ClipData clip = ClipData.newPlainText("simple text", "Hello, World!");
```

- 一个 **URI**

这个片段，通过对记录编码在 content provider 上构造了一个 URI，用于 provider。更多细节查询 [URI 标识符编码](#)：

```
// 创建一个 Uri 基于基本的 URI 和联系人的记录 ID
// 声明基本 URI 字符串
private static final String CONTACTS =
"content://com.example.contacts";
// 声明一个复制数据的 URI 字符串路径
private static final String COPY_PATH = "/copy";

// 声明 URI 粘贴用于剪贴板
Uri copyUri = Uri.parse(CONTACTS + COPY_PATH + "/" + lastName);

..
```

```
// 创建一个新的URI剪贴对象。该系统采用的的匿名getContentResolver () 对象
// 从provider获得的MIME类型。剪贴对象的label是“URI”，其数据是
// 之前创建的Uri。
ClipData clip = ClipData.newUri(getContentResolver(), "URI", copyUri);
```

• Intent

这个片段，构建了一个应用程序的Intent，然后把它放在剪贴对象中：

```
// 创建 Intent
Intent appIntent = new Intent(this,
com.example.demo.myapplication.class);

.....
// 创建一个带Intent的剪贴对象。它的label是“Intent”和它的数据
// 之前创建的Intent
ClipData clip = ClipData.newIntent("Intent", appIntent);
```

4. 把新的剪贴对象放到剪贴板上：

```
// 设置剪贴板的主要剪贴对象。
clipboard.setPrimaryClip(clip);
```

从剪贴板粘贴

如前所述，从全局剪贴板对象中粘贴数据，取得剪贴对象，查看数据类型，如果可能的话将剪贴对象的数据复制到自己的存储。本节详细描述在如何粘贴三种形式的剪贴板数据。

粘贴纯文本

粘贴纯文本，首先获得全局剪贴板并检验它可以返回纯文本。然后使用的getText () 取得剪贴对象，并复制纯文本，如下所述：

1. 获取的的全局[ClipboardManager](#)对象，使用[getSystemService \(CLIPBOARD_SERVICE\)](#)。此外，声明一个全局变量包含要粘贴的文本：

```
ClipboardManager clipboard = (ClipboardManager)
getSystemService(Context.CLIPBOARD_SERVICE);
String pasteData = "";
```

2. 接下来，在当前的Activity选项中确定是否应该启用或禁用的“粘贴”。该验证剪贴板包含剪贴对象，这样你可以处理剪贴推向所代表的数据类型：

```
/ / 获取的“粘贴”菜单项的ID
MenuItem mPasteItem = menu.findItem(R.id.menu_paste);
// 如果剪贴板不包含数据，禁用粘贴“菜单项。
// 如果它包含数据，判断是否能处理
if (!clipboard.hasPrimaryClip()) {
    mPasteItem.setEnabled(false);
} else if
(!clipboard.getPrimaryClipDescription().hasMimeType(MIMETYPE_TEXT_PLAIN))
{
    // 这将禁用粘贴“菜单项，不是纯文本
    mPasteItem.setEnabled(false);
} else {
    // 这启用“粘贴”菜单项，因为剪贴板包含纯文本.
    mPasteItem.setEnabled(true);
}
```

3. 从剪贴板中复制数据。如果“粘贴”菜单项启用，程序才能取得数据，所以你可以假设剪贴板包含纯文本。你蹦不知道他是否包含一个指向纯文本的文本字符串或URI。下面的代码片断测试此功能，但它只显示处理纯文本的代码：

```
/ / 响应用户选择“粘贴”
case R.id.menu_paste:
    // 检查剪贴板上的项目。如果的getText() 没有返回null，剪辑的项目包含
    // 文本。假定该应用程序一次只能处理一个项目。
    ClipData.Item item = clipboard.getPrimaryClip().getItemAt(0);
    // 获得文本剪贴板
    pasteData = item.getText();
    // 如果字符串包含的数据，然后进行粘贴操作
    if (pasteData != null) {
        return;
    }
    // 剪贴板中不包含文本。如果它包含一个URI，试图从它那里得到的数据
    Uri pasteUri = item.getUri();
    // 如果URI包含的东西，尝试从它那里得到的文本
    if (pasteUri != null) {
        // 调用例程解析URI，并从它那里得到的数据。
        // 这里无此例程
        pasteData = resolveUri (URI) ;
```

```

        return;
    } else {
        // 某处有错。MIME类型为纯文本，但剪贴板不包含
        // 文本或一个URI。报告一个错误
        Log.e("Clipboard contains an invalid data type");
        return;
    }
}

```

从content URI粘贴数据

如果[ClipData.Item - ClipData.Item](#)对象包含content URI并且您已确定您可以处理其MIME类型，创建[ContentResolver](#)然后调用适当的content provider的方法检索数据。以下过程描述如何从 基于content provider URI内容的剪贴板上取得数据的。它检查provider可提供给应用程序可以使用的MIME类型：

1. 声明一个全局变量包含MIME类型：

```

// 声明一个MIME类型的常量，匹配由provider提供的MIME类型的
public static final String MIME_TYPE_CONTACT =
"vnd.android.cursor.item/vnd.example.contact"

```

2. 获取全局剪贴板。也得到了内容解析器，能够访问content provider：

```

// 获取一个句柄到 Clipboard Manager
ClipboardManager clipboard = (ClipboardManager)
getSystemService(Context.CLIPBOARD_SERVICE);

// 获取内容解析器实例
ContentResolver CR = getContentResolver();
<java>

```

3. 获取从剪贴板主要剪贴对象，并得到content作为一个URI：

```

<java>
// 获取剪贴板数据
ClipData clip = clipboard.getPrimaryClip();

if (clip != null) {
    // 从剪贴板中获取数据的第一项
    ClipData.Item item = clip.getItemAt(0);

    // 尝试获得该项目的content，作为一个URI
    Uri pasteUri = item.getUri();
}

```

4. 测试，通过调用

```

'[http://developer.android.com/reference/android/content/ContentResolver.html
getUri(URI)]' 判断是否为content URI。此方法返回null，那么URI不指向一个有效的content provider:

```

```
// 如果剪贴板中包含一个URI引用
if (pasteUri != null) {
    // 这是一个content URI
    String uriMimeType = cr.getType(pasteUri);
```

5. 测试，如果目前应用的能够明白content provider 支持的MIME类型。如果是的话，调用 [ContentResolver.query\(\)](#) 来获取数据。返回值是一个游标：

```
// 如果返回值不为null, Uri是一个content URI
if (uriMimeType != null) {
    // content provider 是否提供了一个当前的应用程序可以使用的MIME类型?
    if (uriMimeType.equals(MIME_TYPE_CONTACT)) {
        // 从content provider获取数据。
        Cursor pasteCursor = cr.query(uri, null, null, null,
null);
        // 如果包含数据, 移动到第一条记录
        if (pasteCursor != null) {
            if (pasteCursor.moveToFirst()) {
                // 这里游标的可以获得数据。根据于不同的数据模型格式该代码会有所不同
            }
        }
        // 关闭的光标
        pasteCursor.close();
    }
}
```

粘贴Intent

粘贴到一个Intent，首先获得全局剪贴板。检查[ClipData.Item - ClipData.Item](#)对象，看它是否包含一个Intent。然后调用[getIntent\(\)](#)复制到自己的存储空间的Intent。下面的代码片断演示此方法：

```
// 获得ClipboardManager句柄
ClipboardManager clipboard = (ClipboardManager)
getSystemService(Context.CLIPBOARD_SERVICE);

// 检查, 看看是否剪贴对象的项目包含一个Intent, 通过getIntent() 测试检查是否返回空
Intent pasteIntent =
clipboard.getPrimaryClip().getItemAt(0).getIntent();

if (pasteIntent != null) {
    // 处理的Intent
```

```

} else {
    // 如果您的应用程序在等一个剪贴板上的Intent，忽略剪贴板，
    // 或发出一个错误
}

```

通过**content provider** 复制复杂的数据

content provider支持复杂的数据，如数据库记录或文件流复制。要复制数据，你把**content URI**放到剪贴板上。粘贴到应用然后从剪贴板取出**URI**，并用它来检索数据库中的数据或文件流描述符。

由于应用程序只粘贴**content URI**数据，它需要知道要检索哪些数据块。您可以提供这些信息对**URI**本身的数据编码标识符，也可以提供一个独特的**URI**将返回您要复制的数据。选择哪种技术取决于您的数据组织。

以下部分描述如何设置uri，如何提供复杂的数据，以及如何提供文件流。下面描述假定您对**content provider**设计的一般原则比较熟悉。

URI标识符编码

将一个**URI**表示的数据复制到剪贴板的一个的方法是对**URI**自身进行编码转换成标识符。**content provider** 可以从**URI**标识符取得标识符，并用它来检索数据。粘贴应用程序不必知道标识符的存在，它需要做的就是让你从剪贴板“引用”**URI**和标识符，给您的**content provider**，并取回数据。

通常对**content URL**编码，是通过连接的**URI**。例如，假设你提供的**URI**定义为以下字符串：

```
"content://com.example.contacts"
```

如果你想将这个**URI**的编码加入名称，你可以使用下面的代码片断：

```

String uriString = "content://com.example.contacts" + "/" + "Smith"
// uriString中包含的content://com.example.contacts/Smith.
// 生成一个字符串表示的URI对象的
Uri copyUri = Uri.parse(uriString);

```

如果您已经使用了content provider，您可能需要添加一个新的URI路径，用于URI的复制。例如，假设你已经有以下URI路径：

```
"content://com.example.contacts"/people
"content://com.example.contacts"/people/detail
"content://com.example.contacts"/people/images
```

您可以添加另一个用于复制的特定URI路径：

```
"content://com.example.contacts/copying"
```

然后，您可以根据模式匹配检测带“copy”的URI并专门处理复制和粘贴的代码。

如果你已经使用了一个content provider，您通常使用的编码技术来组织您内部数据库，或内部表。在这种情况下，你有多个数据块要复制，很可能每个数据都有唯一标识符。通过响应应用程序粘贴的查询，你可以搜索到它的标识符数据，并返回它。

如果你没有多个数据块，那么你可能并不需要一个标识符编码。可以直接使用您的provider 程序特有的URI。响应查询后，它将返回它目前包含的数据。

[记事本](#)例程中获取单个记录的ID来打开笔记列表中的笔记。该示例使用SQL数据库的_id段，但你可以用任何你有的的数字或字符标识符。

复制数据结构

你设置content provider作为[ContentProvider](#)组件的一个子类进行复制和粘贴复杂数据。你还要对放在剪贴板上的URI编码，以便它精确的指向您希望提供的记录。此外，您必须考虑现有应用程序的状态：

如果你已经有了一个content provider，您可以添加它的功能。你可能只需要修改它的[query\(\)](#)方法来处理应用程序要粘贴数据的URI。你可能会想要修改的方法来处理URI模式“复制”。如果您的应用程序维护一个内部数据库，你可能想将他转移到content provider，以便从它的复制。如果您目前没有使用数据库，你可以实现一个简单的内容提供商，其唯一目的是提供应用程序从剪贴板粘贴数据。在content provider内部，您至少要覆盖以下方法：

[query\(\)](#) 应用程序粘贴时候将假定放在剪贴板上的URI的方法，可以得到数据。

为了支持复制，你应该有个方法检测的URI带有一个特殊的“copy”的路径。应用程序可以在剪贴板上创建一个“copy”的URI，包含你要复制准确的记录，和拷贝路径的指针。

[getUri\(\)](#)此方法应该返回您要复制的数据的MIME类型或类型。方法[newUri\(\)](#)调用[getUri\(\)](#)，用以将MIME类型放入新的[ClipData.Item](#) - [ClipData.Item](#)对象。

复杂数据的MIME类型见[Content Provider](#)主题。

请注意，你不需要任何其他content provider 提供的[insert\(\)](#)或[update\(\)](#)等方法。应用程序粘贴时候只需要取得支持的MIME类型和复制provider的数据。如果你已经拥有这些方法，他们不会影响复制操作。

以下的片段演示如何设置您的应用程序用于复制复杂的数据：

1.在您的应用程序的全局常量，声明一个基URI字符串和路径标识URI字符串，您正在使用的数据复制。也宣布为复制的数据的MIME类型：

```
// 声明基本的URI字符串
private static final String CONTACTS =
"content://com.example.contacts";

// 声明一个URI的路径字符串，用于复制数据
private static final String COPY_PATH = "/copy";

// 声明一个的MIME类型用于复制的数据
public static final String MIME_TYPE_CONTACT =
"vnd.android.cursor.item/vnd.example.contact"
```

2.在Activity中，用户复制数据，设置代码将数据复制到剪贴板。响应成复制请求，并把URI置于剪贴板上：

```
public class MyCopyActivity extends Activity {

    ...

    // 用户已经选择了一个名字，并请求复制。
    case R.id.menu_copy:
        // 添加名称到基本的URI字符串
        // 名称储存在lastName中
        uriString = CONTACTS + COPY_PATH + "/" + lastName;

        // 解析的字符串转换成一个URI
        Uri copyUri = Uri.parse(uriString);

        // 获取一个剪贴板句柄
```

```

ClipboardManager clipboard = (ClipboardManager)
    getSystemService(Context.CLIPBOARD_SERVICE);

ClipData clip = ClipData.newUri(getApplicationContext(), "URI",
copyUri);

//设置剪贴板的主要剪贴对象
clipboard.setPrimaryClip(clip);

```

3. 在全局范围内设置content provider，创建一个URI匹配，并添加一个URI模式，将匹配模式的URI放在剪贴板上：

```
public class MyCopyProvider extends ContentProvider {
```

```
...
```

```
// URI匹配的对象，简化匹配的content URI的模式。
private static final UriMatcher sUriMatcher = new
UriMatcher(UriMatcher.NO_MATCH);
```

```
//一个整数，用于选择基于传入的URI模式
private static final int GET_SINGLE_CONTACT = 0;
```

```
.....
```

```
//添加的内容URI匹配。它匹配
```

```
"/content://com.example.contacts/copy/*"
sUriMatcher.addURI(CONTACTS, "names/*", GET_SINGLE_CONTACT);
```

4. 设置query()方法。这种方法可以处理不同的URI模式，这取决于你如何编码，但仅显示剪贴板复制操作模式：

```

//设置你的content provider的query()方法
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs,
String sortOrder) {

...
//选择基于传入的内容URI
switch (sUriMatcher.match(uri)) {
    case GET_SINGLE_CONTACT:
        //请求的名称的查询，并返回联系人。在这里你会解码
        //传入的URI，查询基于姓氏的数据模型，并返回游标结果,
        ...
}

```

5. 设置getType()方法返回一个适当的复制数据的MIME类型：

/ / 设置你的供应商的get`getType()`方法

```
public String getType(Uri uri) {  
    ...  
    switch (sUriMatcher.match(uri)) {  
        case GET_SINGLE_CONTACT:  
            return (MIME_TYPE_CONTACT);  
    }  
}
```

从[content URI粘贴数据](#)介绍了如何从剪贴板取得content URI，并用它来获取和粘贴数据。

复制数据流

你可以以流的形式复制和粘贴大量的文本和二进制数据。数据可以有诸如下列形式：

- 在实际设备上存储的文件。
- Socket 数据流。
- 提供商的基础数据库系统中存储 的大量数据量。

一个数据流的content provider 通过文件描述符对象来访问其数据，，如对象[AssetFileDescriptor](#)而不是一个[Cursor](#)。应用程序粘贴时，使用文件描述符读取数据流。

设置您的应用程序带有复制数据流的provider，按照下列步骤：

1. 设置你剪贴板上要复制数据流的content URI。这个选项包括以下内容：

- 按照[URI标识符编码](#)一节中所述，对数据流进行URI标识符编码，然后维护你的content provider表，其中包含标识符和相应的流的名字。
- 直接对流名字进行URI编码。
- 使用一个唯一的URI总是返回provider提供的当前流。如果你使用这个选项，你必须要记得，只要你通过URI复制流到剪贴板，你就需要更新你的provider，让他指向一个不同的流。

2. 对你打算提供每个类型的数据流，提供MIME类型。应用程序粘贴需要此信息，以确定他们是否可以将其粘贴数据到剪贴板上。
3. 实现一个[ContentProvider](#)的方法返回一个文件描述符流。如果对URI标识符编码，使用此方法，以确定打开的流。
4. 复制数据流到剪贴板上，在剪贴板上构造content URI。

粘贴一个数据流，应用程序从剪贴板中取得剪贴对象，取得URI，并使用它们调用[ContentResolver](#)文件描述符的方法，打开流。[ContentResolver](#)方法调用相应的[ContentProvider](#)的方法，传递给它content URI。您的provide会传回文件描述符给ContentResolver的方法。然后应用程序粘贴能够从流中读取数据。

下面的列表显示content provider 最重要的文件描述符的方法。每个都有一个相应的带字符串“Descriptor”名称的[ContentResolver](#)方法，，例如，[ContentResolver](#)的[openAssetFile\(\)](#)的名称是[openAssetFileDescriptor\(\)](#)：

[openTypedAssetFile\(\)](#)只有当所提供的MIME类型是由provider支持时，此方法应该返回一个文件描述符资源。呼叫者（应用程序做的粘贴）提供了一个MIME类型模式。content provider（复制URI到剪贴板应用程序）返回一个[AssetFileDescriptor](#)文件句柄，如果它可以提供MIME类型，如果它不能，抛出一个异常。

用这种方法处理文件的细节。你可以用它来读取content provider已经复制到剪贴板的资源。

[openAssetFile\(\)](#)这个方法是[openTypedAssetFile\(\)](#)通用的形式。它不会对MIME类型过滤，但它可以读取文件细节。

使用[OpenFile\(\)](#)这是[openAssetFile\(\)](#)更普遍的形式。它不能读取文件的细节。

对于您的文件描述符方法，你可以选择使用的[openPipeHelper\(\)](#)方法。这允许应

用程序粘贴数据通过使用管道在后台线程读取数据流。使用这种方法，你需要实现的[ContentProvider.PipeDataWriter](#)接口。这样的一个例子是[Notepad](#)示例应用程序，在NotePadProvider.java文件中的openTypedAssetFile()方法中。

设计有效的复制/粘贴功能

为您的应用程序设计有效的复制和粘贴功能，请记住以下几点：

- 在任何时候，剪贴板只有一个剪贴对象。应用系统中的任何一个新的复制操作将覆盖以前的对象。在返回之前，由于用户可能会离开您的应用程序并操作复制功能，因此你不能假定剪贴板中包含该的用户以前在应用程序中复制的对象。
- 每剪贴对象的多个[ClipData.Item](#)的更好是用于支持复制和粘贴多个选择，而不是对不同形式的引用单一的选择。通常，你希望所有的在剪贴对象的[ClipData.Item](#)对象有相同的形式，那他们就都应该是简单的文本，content URI，或intent，而不是一种混合物。
- 当你提供的数据，可以提供不同的MIME表示。加入你要支持的MIME类型到[ClipDescription](#)，然后在content provider上实现您的MIME类型。
- 当你从剪贴板中取得数据，你的应用程序负责检查可用的MIME类型，然后决定使用哪一个，如果有的话。即使剪贴板上有一个剪贴对象并且用户请求粘贴，而您的应用程序不需要粘贴时，如果MIME类型是兼容的，你必须进行粘贴。如果你要选的话，你可以通过[coerceToText\(\)](#)强制剪贴板使用文本方式。如果您的应用程序支持多个可用的MIME类型，您可以允许用户选择使用哪一个。

来自“[index.php?title=Copy_and_Paste&oldid=2862](#)”



Creating an IME

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： sfshine

原文链接：<http://developer.android.com/guide/topics/text/creating-input-method.html>

目录

- [1 Creating an Input Method](#)
 - [1.1 The IME Lifecycle-IME的生命周期](#)
 - [1.2 Declaring IME Components in the Manifest-IME的组件清单中声明](#)
 - [1.3 The Input Method API-输入的API](#)
 - [1.4 Designing the Input Method UI-设置输入法的界面](#)
 - [1.4.1 Input view-输入法界面](#)
 - [1.4.2 Candidates view-候选字界面](#)
 - [1.4.3 UI design considerations-界面设计要点](#)
 - [1.4.4 Handling multiple screen sizes-多分辨率支持](#)
 - [1.4.5 Handling different input types-处理不同的输入类型](#)
 - [1.5 Sending Text to the Application-向应用发送文本](#)
 - [1.5.1 Editing the text around the cursor-在光标周围编辑文本](#)
 - [1.5.2 Composing text before committing-在提交文本之前的撰写](#)
 - [1.5.3 Intercepting hardware key events-拦截实体键盘按钮事件](#)

- [1.6 Creating an IME Subtype-创建一个输入法亚种](#)
 - [1.6.1 Choosing IME subtypes from the notification bar-通过通知栏选择输入法的亚种](#)
 - [1.6.2 Choosing IME subtypes from System Settings-通过系统设置选择输入法亚种](#)
- [1.7 General IME Considerations-创建输入法应用的注意事项](#)

Creating an Input Method

An input method editor (IME) is a user control that enables users to enter text. Android provides an extensible input method framework that allows applications to provide users alternative input methods, such as on-screen keyboards or even speech input. Once installed, users can select which IME they want to use from the system settings and use it across the entire system; only one IME may be enabled at a time.

输入法编辑器（IME，有的地方简称IME）是一个使用户能够输入文字的控件。Android提供了一个可扩展的输入法框架，它允许应用程序为用户提供可替代的输入方法，如屏幕上的键盘甚至是语音输入。一旦安装，用户就可以在系统设置里选择他喜欢的输入法，这个设置可以在Android系统的所有的地方使用。当然，使用过程中，一次只有一种输入法被激活（用户选择的那个）。

To add an IME to the Android system, you create an Android application containing a class that extends `InputMethodService`. In addition, you usually create a "settings" activity that passes options to the IME service. You can also define a settings UI that's displayed as part of the system settings.

想要给Android开发一个输入法应用，您的Android应用需要含一个继承自 `InputMethodService` 的类。此外，您通常需要创建一个“设置输入法”的Activity来设定应用的配置。您也可以定义一个作为系统设置的一部分来显示的设置界面。

This article covers the following:

- The IME lifecycle.
- Declaring IME components in the application manifest.
- The IME API.
- Designing an IME UI.
- Sending text from an IME to an application.
- Working with IME subtypes.

If you haven't worked with IMEs before, you should read the introductory article Onscreen Input Methods first. Also, the Soft Keyboard sample app included in the SDK contains sample code that you can modify to start building your own IME.

本文包括以下内容：

- IME的生命周期。
- 在应用程序清单中的声明IME组件。
- IME的API。
- 设计一个IME的界面。
- 从IME向应用程序发送文本。
- 使用IME的变种。

如果您以前没有开发过IME，建议先阅读《屏幕上的输入法》这一章的介绍。此外，您可以通过修改SDK中的软键盘（Soft Keyboard sample）实例代码来开始建立自己的输入法应用。

The IME Lifecycle-IME的生命周期

The following diagram describes the life cycle of an IME:

IME的生命周期如下图：



Figure 1. The life cycle of an IME.

The following sections describe how to implement the UI and code associated with an IME that follows this lifecycle.

下面介绍了怎么根据输入法的生命周期来实现输入法的界面和代码

Declaring IME Components in the Manifest- IME的组件清单中声明

In the Android system, an IME is an Android application that contains a special IME service. The application's manifest file must declare the service, request the necessary permissions, provide an intent filter that matches the action `action.view.InputMethod`, and provide metadata that defines characteristics of the IME. In addition, to provide a settings interface that allows the user to modify the behavior of the IME, you can define a "settings" activity that can be launched from System Settings.

在Android系统中，IME是一个Android应用程序包含一个特殊的输入法服务。应用程序的清单文件必须申报服务，索取必要的权限，提供相匹配的行动`action.view.InputMethod`意图过滤器，并提供元数据定义输入法的特点。此外，提供了一个设置界面，允许用户修改输入法的行为，你可以定义一个“设置”的活动，可以从系统设置推出。

The following snippet declares IME service. It requests the permission `BIND_INPUT_METHOD` to allow the service to connect the IME to the system, sets up an intent filter that matches the action `android.view.InputMethod`, and defines metadata for the IME:

下面的代码片段声明了IME服务，这个服务需要通过`BIND_INPUT_METHOD`权限来是系统允许它连接系统的IME（输入法服务）。还需要创建一个应答`android.view.InputMethod`的意图过滤器以及定义IME的元数据：

```
<!-- Declares the input method service -->
<service android:name="FastInputIME"  

    android:label="@string/fast_input_label"  

    android:permission="android.permission.BIND_INPUT_METHOD">  

    <intent-filter>  

        <action android:name="android.view.InputMethod" />  

    </intent-filter>  

    <meta-data android:name="android.view.im"  

    android:resource="@xml/method" />  

</service>
```

This next snippet declares the settings activity for the IME. It has an intent filter for ACTION_MAIN that indicates this activity is the main entry point for the IME application:

接下来的代码片段声明了设置IME的Activity。他有一个应答`android.intent.action.MAIN`的过滤器，这个过滤器表明这个Activity是IME应用的入口。

```
<!-- Optional: an activity for controlling the IME settings -->
<activity android:name="FastInputIMESettings" 
    android:label="@string/fast_input_settings">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
    </intent-filter>
</activity>
```

You can also provide access to the IME's settings directly from its UI.

您也可以把设置输入法的功能直接放到输入法应用的界面上。

The Input Method API - 输入的API

Classes specific to IMEs are found in the `android.inputmethodservice` and `android.view.inputmethod` packages. The `KeyEvent` class is important for handling keyboard characters.

在`android.inputmethodservice`和`android.view.inputmethod`包下面可以找到开发IME所需的特殊的类。按钮事件类对于处理键盘字符的输入非常重要。

The central part of an IME is a service component, a class that extends `InputMethodService`. In addition to implementing the normal service lifecycle, this class has callbacks for providing your IME's UI, handling user input, and delivering text to the field that currently has focus. By default, the `InputMethodService` class provides most of the implementation for managing the state and visibility of the IME and communicating with the current input field.

一个输入法应用的核心部分是一个服务组件，这是一个继承

自 **InputMethodService** 的类。除了继承通常的服务周期，这个类还有提供输入法界面、处理用户输入、发送文本到焦点区域的回调函数。默认的，**InputMethodService** 类提供了大部分管理输入法的状态、输入法的可见性以及与当前焦点区域交流的接口。

The following classes are also important:

BaseInputConnection

Defines the communication channel from an InputMethod back to the application that is receiving its input. You use it to read text around the cursor, commit text to the text box, and send raw key events to the application. Applications should extend this class rather than implementing the base interface **InputConnection**.

KeyboardView

An extension of View that renders a keyboard and responds to user input events. The keyboard layout is specified by an instance of **Keyboard**, which you can define in an XML file.

下面的类也很重要：

BaseInputConnection

定义了从输入法返回到应用后该应用接收输入法输入信息的渠道。您可以使用它来读取光标周围的文本，提交文本文本框内的信息，发送原生的按钮事件到应用。应用程序应该继承，而不是实现**InputConnection**类的接口。

KeyboardView

查看扩展，呈现一个键盘和响应用户的输入事件。键盘的一个实例，您可以定义在指定的键盘布局XML文件。

Designing the Input Method UI-设置输入法的

界面

There are two main visual elements for an IME: the input view and the candidates view. You only have to implement the elements that are relevant to the input method you're designing.

一个输入法应用有两个主要的界面元素：候选字界面和输入界面。您只需要实现和您设计的输入法有关的元素即可。

Input view-输入法界面

The input view is the UI where the user inputs text, in the form of keyclicks, handwriting or gestures. When the IME is displayed for the first time, the system calls the `onCreateInputView()` callback. In your implementation of this method, you create the layout you want to display in the IME window and return the layout to the system. This snippet is an example of implementing the `onCreateInputView()` method:

输入界面是用户输入通过按键，手写或者手势输入文本的界面。当一个自定义的输入法第一次显示的时候，系统将会执行`onCreateInputView()`方法。在您实现这方法的时候，您可以创建一个想要在输入法窗口显示的布局，然后返回给android系统。下面的代码片段是实现`onCreateInputView()`方法的一个例子。

```
@Override
public View onCreateInputView() {
    MyKeyboardView inputView =
        (MyKeyboardView) getLayoutInflater().inflate(
R.layout.input, null);

    inputView.setOnKeyboardActionListener(this);
    inputView.setKeyboard(mLatinKeyboard);

    return mInputView;
}
```

In this example, `MyKeyboardView` is an instance of a custom implementation of `KeyboardView` that renders a Keyboard. If you're building a traditional QWERTY keyboard, see the Soft Keyboard sample

app for an example of how to extend the KeyboardView class.

在这个例子中，MyKeyboardView是一个继承KeyboardView的自定义接口后实现的一个实例，它呈现了一个键盘。如果您在创建一个QWERTY键盘，您可以参照软键盘的简单实例（Soft Keyboard sample app）来学习怎么继承KeyboardView类。

Candidates view-候选字界面

The candidates view is the UI where the IME displays potential word corrections or suggestions for the user to select. In the IME lifecycle, the system calls onCreateCandidatesView() when it's ready to display the candidate view. In your implementation of this method, return a layout that shows word suggestions, or return null if you don't want to show anything (a null response is the default behavior, so you don't have to implement this if you don't provide suggestions).

候选字界面是输入法显示候选字或者是联想字的界面。在IME的生命周期里，当IME准备显示候选字界面的时候，系统或调用onCreateCandidatesView()方法。在您实现这个方法的时候，请返回一个显示候选字或者联想字的界面，或者，如果您不行显示什么东西，那返回空。（注意，这个方法默认就是返回空，所以如果您不想提供候选字，那你不用实现这个方法。）

For an example implementation that provides user suggestions, see the Soft Keyboard sample app.

实现候选字的实例请查看Soft Keyboard sample app工程。

UI design considerations-界面设计要点

This section describes some specific UI design considerations for IMEs.

这一节介绍输入法界面设计需要注意的要点：

Handling multiple screen sizes-多分辨率支持

The UI for your IME must be able to scale for different screen sizes, and it also must handle both landscape and portrait orientations. In non-fullscreen IME mode, leave sufficient space for the application to show the text field and any associated context, so that no more than half the screen is occupied by the IME. In fullscreen IME mode this is not an issue.

输入法界面必须可以适合不同的分辨率，它也必须适应横屏和竖屏。在非全屏模式，要为应用留出足够的空间来显示文本输入区域和其他相关情况，从而是输入法界面占用的屏幕空间少于屏幕的一半。在全屏输入法模式下就不用考虑这个问题了。

Handling different input types-处理不同的输入类型

Android text fields allow you to set a specific input type, such as free form text, numbers, URLs, email addresses, and search strings. When you implement a new IME, you need to detect the input type of each field and provide the appropriate interface for it. However, you don't have to set up your IME to check that the user entered text that's valid for the input type; that's the responsibility of the application that owns the text field.

android文本输入域允许用户输入特定的文本类型，比如任何文本输入数字输入，URL输入，email输入，搜索字符串输入等。创建一个新的输入法的时候，您需要检测每一个输入框的类型并为它提供正确的输入，然而，您不必设置您的输入法来检查用户输入的文本是否符合指定的类型；这是那个输入框所在的应用程序的工作。

For example, here are screenshots of the interfaces that the Latin IME provided with the Android platform provides for text and phone number inputs:

这里是拉丁输入法的提供文本输入和数字输入的截图：



Figure 2. Latin IME input types

When an input field receives focus and your IME starts, the system calls `onStartInputView()`, passing in an `EditorInfo` object that contains details about the input type and other attributes of the text field. In this object, the `inputType` field contains the text field's input type.

当一个输入域获得焦点后,您的输入法就会开始运行。系统会调用`onStartInputView()`, 传递一个含有输入类型和输入框其他属性的`EditorInfo`实体。在这个实体中, 输入类型域含有输入域的输入类型。

The `inputType` field is an `int` that contains bit patterns for various input type settings. To test it for the text field's input type, mask it with the constant `TYPE_MASK_CLASS`, like this:

输入类型域是一个`int`类型, 他含有一个标志各种输入类型的二进制数值。如果您想测试输入域类型, 您可以使用`TYPE_MASK_CLASS`常量, 如下所示

```
inputType & InputType.TYPE_MASK_CLASS
```

The input type bit pattern can have one of several values, including:

每一个二进制数值可以是下面几个值中的一个:

TYPE_CLASS_NUMBER

A text field for entering numbers. As illustrated in the previous screen shot, the Latin IME displays a number pad for fields of this type.

输入数字的输入域。就像上面介绍中的截屏那样,拉丁输入法展示这样一种字母输入模版。

TYPE_CLASS_DATETIME

A text field for entering a date and time.

一个输入时间和日期的输入域。

TYPE_CLASS_PHONE

A text field for entering telephone numbers.

一个输入电话号码的输入域。

TYPE_CLASS_TEXT

A text field for entering all supported characters.

一个输入字符的输入域。

These constants are described in more detail in the reference documentation for `InputType`.

在输入类型的文档里面有更多关于这些常量的描述。

The `inputType` field can contain other bits that indicate a variant of the text field type, such as:

输入类型标志也可以是从输入域变化而来的其他常量，比如：

TYPE_TEXT_VARIATION_PASSWORD

A variant of `TYPE_CLASS_TEXT` for entering passwords. The input method will display dingbats instead of the actual text.

一个`TYPE_CLASS_TEXT`的变种，用来输入密码。输入的内容将显示为点点而不是真正的文本。

TYPE_TEXT_VARIATION_URI

A variant of `TYPE_CLASS_TEXT` for entering web URLs and other Uniform Resource Identifiers (URIs).

一个`TYPE_CLASS_TEXT` 的变种，用来输入web URL和其他统一资源定位符（URI）。

TYPE_TEXT_FLAG_AUTO_COMPLETE

: A variant of `TYPE_CLASS_TEXT` for entering text that the application "auto-completes" from a dictionary, search, or other facility. : 一个`TYPE_CLASS_TEXT` 的变种，用来输入应用程序通过词典，搜索或者其他功能可以自动补全的文本。

Remember to mask `inputType` with the appropriate constant when you

test for these variants. The available mask constants are listed in the reference documentation for `InputType`.

当您测试这些变种标志的时候，记得使用正确的常量标志。这些常量标志在输入类型文档里面有详细说明。

Caution: In your own IME, make sure you handle text correctly when you send it to a password field. Hide the password in your UI both in the input view and in the candidates view. Also remember that you shouldn't store passwords on a device. To learn more, see the [Designing for Security guide](#). 注意:在您的输入法中,如果需要输入信息到密码框,请确保文字处理的正确性, 在您的输入界面和候选字界面都要隐藏密码文字。记住, 不要在一个设备上储存密码。更多信息参阅设计的安全性 (Designing for Security guide) 这一节。

Sending Text to the Application-向应用发送文本

As the user inputs text with your IME, you can send text to the application by sending individual key events or by editing the text around the cursor in the application's text field. In either case, you use an instance of `InputConnection` to deliver the text. To get this instance, call `InputMethodManager.getCurrentInputConnection()`.

当用户使用您的输入法输入文本的时候，您可以通过使用发送个人按键事件或者在应用输入域的光标周围编辑文本的方式来应用发送文本。或者，您可以使用`InputConnection` 的一个实例来发送文本。您可以通
过`InputMethodManager.getCurrentInputConnection()`来获取这个实例。

Editing the text around the cursor-在光标周围编辑文本

When you're handling the editing of existing text in a text field, some of the more useful methods in `BaseInputConnection` are:

当您处理文本域存在的文本的时候，`BaseInputConnection`里有很多非常有用的方法：

`getTextBeforeCursor()`

Returns a `CharSequence` containing the number of requested characters before the current cursor position.

返回光标所在位置前面的n个字符。

`getTextAfterCursor()`

Returns a `CharSequence` containing the number of requested characters following the current cursor position.

返回光标所在位置后面的n个字符。

`deleteSurroundingText()`

Deletes the specified number of characters before and following the current cursor position.

删除光标所在位置周围的n个字符

`commitText()`

Commit a `CharSequence` to the text field and set a new cursor position.

把一个文本提交到文本域并把光标设置到新的位置。

For example, the following snippet shows how to replace the text "Fell" to the left of the with the text "Hello!":

下面的代码片段展示了怎么使用“Hello”文本替换“Fell”的左边。

```
InputConnection ic = getCurrentInputConnection();
ic.deleteSurroundingText(4, 0);
ic.commitText("Hello", 1);
```

```
ic.commitText( "!" , 1 );
```

Composing text before committing-在提交文本之前的撰写

If your IME does text prediction or requires multiple steps to compose a glyph or word, you can show the progress in the text field until the user commits the word, and then you can replace the partial composition with the completed text. You may give special treatment to the text by adding a "span" to it when you pass it to `InputConnection#setComposingText()`.

如果您的输入法通过需要各种计算或者各种按键的组合来生成字符或单词(比如中文),您可以先在文本域显示这个输入进度直到用户提交了这些文字,这样您可以替换已经完成文本的一部分。当您需要把它传递到`InputConnection#setComposingText()`时,你可以通过使用“间隔”来区别对待。

The following snippet shows how to show progress in a text field:

下面的代码片段说明了怎么在输入域显示输入的进度。

```
InputConnection ic = getCurrentInputConnection();
ic.setComposingText( "Composi" , 1 );
...
ic.setComposingText( "Composin" , 1 );
...
ic.commitText( "Composing" , 1 );
```

The following screenshots show how this appears to the user:

下面的截屏显示这如何呈现给用户



Figure 3. Composing text before committing.

Intercepting hardware key events-拦截实体键盘按钮事件

Even though the input method window doesn't have explicit focus, it receives hardware key events first and can choose to consume them or forward them along to the application. For example, you may want to consume the directional keys to navigate within your UI for candidate selection during composition. You may also want to trap the back key to dismiss any popups originating from the input method window.

尽管输入法窗口不会争夺输入焦点，但输入法还是会先接受来自实体键盘的输入并且可以选择忽略他们或者把他们转发给应用

To intercept hardware keys, override `onKeyDown()` and `onKeyUp()`. See the Soft Keyboard sample app for an example.

如果您想拦截实体键盘的输入，请重写`onKeyDown()` 和`onKeyUp()`方法，详情参考键盘输入应用。

Remember to call the `super()` method for keys you don't want to handle yourself.

如果不想自己处理一些按键事件，记得调用按键的`super()`方法。

Creating an IME Subtype-创建一个输入法亚种

Subtypes allow the IME to expose multiple input modes and languages supported by an IME. A subtype can represent:

输入法亚种可以是输入法执行几种不同的输入模式也可以支持多种语言(比如搜狗输入法的中英文切换)。一个亚种可以是：

- A locale such as `en_US` or `fr_FR`
- 一种语言环境比如`en_US`或者`fr_FR`

- An input mode such as voice, keyboard, or handwriting
 - 一种输入模式比如声音键盘或者手写输入
-
- Other input styles, forms, or properties specific to the IME, such as 10-key or qwerty keyboard layouts.
 - 其他输入样式，输入形式或者输入法的特殊属性，比如10按键或者qwerty键盘布局。

Basically, the mode can be any text such as "keyboard", "voice", and so forth.

基本上这些模式可以是键盘输入，语音输入等等。

A subtype can also expose a combination of these.

一个亚种也可以上面几种的混合。

Subtype information is used for an IME switcher dialog that's available from the notification bar and also for IME settings. The information also allows the framework to bring up a specific subtype of an IME directly. When you build an IME, use the subtype facility, because it helps the user identify and switch between different IME languages and modes.

输入法切换窗口需要输入法亚种信息，这些信息可以从通知栏或者输入法设置里得到。这些信息也使得系统框架可以直接选择其中的一种特定的输入法亚种。当您创建输入法的时候，请使用亚种功能，因为他可以帮助用户区别和切换不同的语言和不同的模式。

You define subtypes in one of the input method's XML resource files, using the <subtype> element. The following snippet defines an IME with two subtypes: a keyboard subtype for the US English locale, and another keyboard subtype for the French language locale for France:

你可以使用<subtype>在一个xml资源文件里定义亚种。下面的代码片段定

义了两种输入法亚种：一种是美国英语语言环境，另一中是法国法语语言环境。

```

<input-method
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:settingsActivity="com.example.softkeyboard.Settings"
        android:icon="@drawable/ime_icon"
    <subtype android:name="@string/display_name_english_keyboard_ime"
        android:icon="@drawable/subtype_icon_english_keyboard_ime"
            android:imeSubtypeLanguage="en_US"
            android:imeSubtypeMode="keyboard"
            android:imeSubtypeExtraValue="somePrivateOption=true"
        />
    <subtype android:name="@string/display_name_french_keyboard_ime"
        android:icon="@drawable/subtype_icon_french_keyboard_ime"
            android:imeSubtypeLanguage="fr_FR"
            android:imeSubtypeMode="keyboard"
        android:imeSubtypeExtraValue="foobar=30,someInternalOption=false"
        />
    <subtype android:name="@string/display_name_german_keyboard_ime"
        ...
    />
/>
```

To ensure that your subtypes are labeled correctly in the UI, use %s to get a subtype label that is the same as the subtype's locale label. This is demonstrated in the next two snippets. The first snippet shows part of the input method's XML file:

为了确保您的输入法亚种在界面上的标签正确，请使用%s来获取一个和亚种本地标签相同的亚种标签。如下面的两个代码片段：第一个代码片段展示了输入法xml文件的一部分：

```
<subtype
    android:label="@string/label_subtype_generic"
    android:imeSubtypeLocale="en_US"
    android:icon="@drawable/icon_en_us"
    android:imeSubtypeMode="keyboard" />
```

The next snippet is part of the IME's strings.xml file. The string resource label_subtype_generic, which is used by the input method UI definition to set the subtype's label, is defined as:

下一个片段是输入法的strings.xml文件的一部分。这个字符串资源

“label_subtype_generic”被输入法界面用来定义亚种标签，他这样被定义：

```
<string name="label_subtype_generic">%s</string>
```

This sets the subtype's display name to “English (United States)” in any English language locale, or to the appropriate localization in other locales.

这个设置了在任何英文语言环境使用“美式英语”或者在其他语言环境使用恰当的语言。

Choosing IME subtypes from the notification bar-通过通知栏选择输入法的亚种

The Android system manages all subtypes exposed by all IMEs. IME subtypes are treated as modes of the IME they belong to. In the notification bar, a user can select an available subtype for the currently-set IME, as shown in the following screenshot:

android系统可以管理输入法暴露的所有输入法亚种。输入法亚种被看作是他们所属输入法的一种模式。在通知栏，一个用户可以当前输入法的选择一种输入模式，就下面的截图：



Figure 4. Choosing an IME subtype from the notification bar



Figure 5. Setting subtype preferences in System Settings

Choosing IME subtypes from System Settings-通过系统设置选择输入法亚种

A user can control how subtypes are used in the “Language & input” settings panel in the System Settings area. In the Soft Keyboard sample, the file `InputMethodSettingsFragment.java` contains an implementation that facilitates a subtype enabler in the IME settings. Please refer to the SoftKeyboard sample in the Android SDK for more information about how

to support Input Method Subtypes in your IME.

用户可以在设置的"语言和输入"设置栏里设置输入法亚种。在软键盘例子中，`InputMethodSettingsFragment.java` 文件实现了在输入法设置中显示输入法亚种这个功能。请更多信息请参阅android SDK的软键盘例子。



Figure 6. Choosing a language for the IME

General IME Considerations- 创建输入法应用的注意事项

Here are some other things to consider as you're implementing your IME:

这里是您创建输入法应用需要注意的其他事项：

- Provide a way for users to set options directly from the IME's UI.
• 为用户提供一个可以在输入法界面直接设置输入法的方法。
- Because multiple IMEs may be installed on the device, provide a way for the user to switch to a different IME directly from the input method UI.
• 由于输入法可能需要安装到设备上，请为用户提供一个方法来直接在输入界面切换不同的输入法
- Bring up the IME's UI quickly. Preload or load on demand any large resources so that users see the IME as soon as they tap on a text field. Cache resources and views for subsequent invocations of the input method.
• 输入法界面需要反应迅速。提前载入或者只载入需要的大的界面资

源，这样用户一点击文本框就可以看到。为了同样的目的，可以对输入法后续用到的资源和视图进行缓存。

- Conversely, you should release large memory allocations soon after the input method window is hidden, so that applications can have sufficient memory to run. Consider using a delayed message to release resources if the IME is in a hidden state for a few seconds.
- 相反的，当输入法窗口隐藏的时候，您应该释放为输入法分配的内存，这样应用可以有足够的内存来运行。如果输入法处于隐藏状态需要几秒钟，那您需要考虑使用一个延迟消息来释放资源。
- Make sure that users can enter as many characters as possible for the language or locale associated with the IME. Remember that users may use punctuation in passwords or user names, so your IME has to provide many different characters to allow users to enter a password and get access to the device.
- 请确保用户可以尽可能多的输入这种输入法需要输入的语言的字符。记住，用户可能需要输入用户名和密码，所以您的输入法必须提供各种不同的字符，使用户可以想设备输入密码。

来自“[index.php?title=Creating_an_IME&oldid=7985](#)”

Spelling Checker Framework

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

拼写检查器框架

Android平台提供了一个拼写检查器的框架，来方便你在你的应用中实现并使用拼写检查。这个框架是Android平台提供的文字服务API之一。

要在你的应用中使用该框架，你需要创建一个特殊的AndroidService来生成一个拼写检查器会话对象。这个会话对象会基于你提供的文字，来返回拼写检查器生成的拼写建议。

拼写检查器生命周期

以下的图表显示了拼写检查器服务的生命周期：



图1.拼写检查器服务的生命周期。

为了初始化拼写检查，你的应用必须实现它自己的拼写检查器服务。你应用中的客户端，比如activity,或是独立的UI元素，需要从服务中获取一个拼写检查器会话，然后使用该会话为文本获取拼写建议。当一个客户端停止它的活动时，它结束自己的会话。你的应用可以在必要时，随时关闭拼写检查器服务。

实现一个拼写检查器服务

为了在你的应用中使用拼写检查器框架，你需要添加一个包含会话对象实现的拼写检查器服务组件。你也可以为你的应用增加一个可选的activity来

控制设置。你也必须添加一个元数据xml文件来描述拼写检查器服务，并在manifest文件中增加适当的元素。

拼写检查类。

用以下的类来定义服务和会话对象：

一个[SpellCheckerService](#)的子类。

[SpellCheckerService](#)实现了[Service](#)类和拼写检查器框架接口，在你的子类中，你必须实现以下方法：

[createSession\(\)](#)

这是一个工厂方法，它为客户端返回一个[SpellCheckerService.Session](#)对象来实现拼写检查。

查看[Spell Checker Service](#)样例应用，来学习更多实现该类的知识。

一个[SpellCheckerService.Session](#)的实现。

这是一个拼写检查器服务给客户提供的对象，来让他们把文字传递给拼写检查器并接收提示。在这个类里，你必须实现以下方法：

[onCreate\(\)](#)

这是[createSession\(\)](#)的系统回调函数，在这个方法中，你可以基于本地化以及其它内容，初始化[SpellCheckerService.Session](#)对象。

[\[, int\) onGetSentenceSuggestionsMultiple\(\)](#)

该函数真正来进行拼写检查，它返回一个[SentenceSuggestionsInfo](#)数组，该数组包含了传入句子的提示。

你也可以实现一些可选的函数，比如[onCancel\(\)](#)是用来处理取消拼写检查的请求，[int\) onGetSuggestions\(\)](#)是用来处理一个词提示请求，或者[int\) onGetSuggestionsMultiple\(\)](#)是用来处理一组词的提示的请求。

查看[Spell Checker Service](#)例子应用来学习更多。

注意：你必须把拼写检查设为异步和线程安全的。一个拼写检查器可能被在不同核心上的不同的线程同时调用。[SpellCheckerService](#) 和 [SpellCheckerService.Session](#)自动实现了上述要求。

拼写检查器的声明和元数据

除了代码以外，你需要为拼写检查器提供合适的manifest文件和元数据文件。

manifest文件定义了应用，服务，以及用于设置的活动，如下示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.samplespellcheckerservice" >
    <application
        android:label="@string/app_name" >
        <service
            android:label="@string/app_name"
            android:name=".SampleSpellCheckerService"
            android:permission="android.permission.BIND_TEXT_SERVICE"
>
            <intent-filter >
                <action
                    android:name="android.service.textservice.SpellCheckerService" />
            </intent-filter>
            <meta-data
                android:name="android.view.textservice.scs"
                android:resource="@xml/spellchecker" />
        </service>
        <activity
            android:label="@string/sample_settings"
            android:name="SpellCheckerSettingsActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

注意想要使用该服务的组件需要使用BIND_TEXT_SERVICE的许可来确保系统绑定该服务。该服务的定义中还确定了spellchecker.xml元数据文件，这

将在下一节给出。

元数据文件spellchecker.xml的内容如下：

```
<spell-checker
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:label="@string/spellchecker_name"

    android:settingsActivity="com.example.SpellCheckerSettingsActivity">
        <subtype
            android:label="@string/subtype_generic"
            android:subLocale="en"
        />
        <subtype
            android:label="@string/subtype_generic"
            android:subLocale="fr"
        />
</spell-checker>
```

元数据里确定了拼写检查器用于控制设置的activity。它还定义了拼写检查器的子类;在这种情况下，子类定义了拼写检查器适用的地区。

从客户端接入拼写检查器服务

使用TextView视图应用自动获取拼写检查，因为TextView会自动使用一个拼写检查器。如下截图所示：



图2.TextView中的拼写检查

然而在其它情况下，你可能也想要直接与拼写检查器服务进行交流。如下图表是直接与拼写检查器交流的控制流程：



图3.与一个拼写检查器服务交互。

[Spell Checker Service](#) 例子应用，将教你如何与一个拼写检查服务进行交互。Android Open Source Project里的LatinIME输入法编辑器里也有一个拼写检查的例子。

Data Storage

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/guide/topics/data/index.html>

编辑者： eoe耗子

更新时间： 2012.07.25

在内存或可移动存储设备上存储应用程序数据，包括数据库、文件、参数。用户还可以增加一个数据备份服务，用于存储和恢复应用程序和系统数据。

练习

[同步到云](#)

这个分类中包含了云程序的不同策略。包括使用自己的后端web程序将数据同步到云，并使用云备份数据，以便让用户在新设备上安装程序时恢复数据。

来自“[index.php?title=Data_Storage&oldid=6155](#)”



Storage Options

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/guide/topics/data/data-storage.html>

编辑者：[贼寇在何方](#)

更新时间： 2012.06.08

Android为你提供了若干选项用于存储应用程序数据。选择何种方案时情况而定。比如，数据仅为你的应用所使用，或是可是为其他应用（以及用户）所共享。又比如，你的数据需要多少空间。

数据存储有以下几个选择：

[Shared Preferences | 公共配置-Shared Preferences](#)

通过键值对的方式存储私有的原始数据。

[Internal Storage | 内部存储-Internal Storage](#)

在存储器上存储私有的数据。

[External Storage | 外部存储 - External Storage](#)

在外部存储器上存储公开的数据。

[SQLite Database | SQLite数据库 - SQLite Database](#)

在私有的数据库中存储结构化的数据。

[Network Connection | 网络连接 - Network Connection](#)

使用你的网络服务器存储数据。

Android提供了暴露私有数据给其他应用的方法——使用一个[内容提供器\(content provider\)](#)。内容提供器是一个可选的组件，为你的应用程序数据提供读/写权限，并受制于你给定的限制。关于使用内容提供器的更多细节，请查看[Content Providers](#)的文档。

目录

[[隐藏](#)]

[1 使用共享的配置 - Using Shared Preferences](#)

[2 使用内部存储 - Using the Internal Storage](#)

- [2.1 保存缓存文件](#)
- [2.2 其他有用的方法](#)

[3 使用外部存储 - Using the External Storage](#)

- [3.1 获取外部存储访问](#)
- [3.2 检测媒体可用性](#)
- [3.3 保存与其他应用程序共享文件](#)
- [3.4 保存应用程序专用文件](#)
- [3.5 保存应当共享的文件](#)

[4 在媒体扫描器下隐藏你的文件](#)

- [4.1 保存缓存文件](#)
- [4.2 使用数据库 - Using Databases](#)
 - [4.2.1 数据库调试](#)
- [4.3 使用网络连接 - Using a Network Connection](#)

使用共享的配置 - Using Shared Preferences

[SharedPreferences](#)类提供了一个通用的框架，用于保存和检索以持久化的键值对形式存储的原始数据类型。你可以使用[SharedPreferences](#)保存任意类型的原始数据：布尔（boolean），浮点，（float），整型（int），长整型（long）和字符串（string）。这些数据将会存放在用户会话中（即使你的应用程序已经退出）。

在应用程序中取得`SharedPreferences`对象，使用以下两种方法之一：

- `int) getSharedPreferences()` - 当你需要多个由名字标识的配置文件——名称由第一个参数指定。
- `int) getPreferences()` - 当你仅需要一个配置文件。由于这是你的Activity的唯一一个配置文件，所以不必提供名称。

用户配置 共享的配置并非一定要保存“用户配置”，例如用户选择和哪个铃声。如果你有兴趣为你的应用创建用户配置，请参考[PreferenceActivity](#)。它提供了一个Activity的框架，可以用来创建持久化的用户配置（使用共享的配置）。

向SharedPreferences写入值的步骤：

- 1、调用`edit()`，取得一个`SharedPreferences.Editor`。
- 2、使用形如`boolean) putBoolean()`和`java.lang.String) putString()`这样的方法添加值。
- 3、使用`commit()`提交新值。

从`SharedPreferences`读取值，使用它的方法，例如`boolean) getBoolean()`和`java.lang.String) getString()`即可。这是一个例子，在一个计算器中使用无声的按键保存一个配置：

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
        super.onCreate(state);
        ...

        // Restore preferences
        SharedPreferences settings =
getSharedPreferences(PREFS_NAME, 0);
        boolean silent = settings.getBoolean("silentMode", false);
        setSilent(silent);
    }
}
```

}

@Override

protected void onStop(){

super.onStop();

// We need an Editor object to make preference changes.

// All objects are from android.context.Context

SharedPreferences settings =

getSharedPreferences(PREFS_NAME, 0);

SharedPreferences.Editor editor = settings.edit();

editor.putBoolean("silentMode", mSilentMode);

// Commit the edits!

editor.commit();

}

}

使用内部存储 - Using the Internal Storage

你可以直接将文件保存在设备上的内部存储中。缺省情况下，存放于内部存储的文件为你的应用程序所私有，其他应用程序不能够访问它们（其他用户亦然）。当用户卸载你的应用，这些文件也被删除。

在内部存储中创建并写入一个私有文件：

- 1、调用[openFileOutput\(\)](#)，传入文件名和操作模式。方法返回一个[FileOutputStream](#)对象。
- 2、使用[write\(\)](#)写文件。
- 3、使用[close\(\)](#)关闭文件流。

例如：

```

String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();

```

MODE_PRIVATE创建文件（或以同名文件替换），并为你的应用所私有。其他可用的模式有：MODE_APPEND, MODE_WORLD_READABLE, 和MODE_WORLD_WRITEABLE。

从内部存储中读一个文件：

- 1、调用[openFileInput\(\)](#)，并传递需要读取的文件的名称。这个方法返回一个[FileInputStream](#)。
- 2、使用[read\(\)](#)从文件中读取数据。
- 3、使用[close\(\)](#)关闭文件流。

提示：如果你想在编译时往你的应用中存入一个静态文件，就得把文件保存到项目的res/raw目录下。你可以调用[openRawResource\(\)](#)并传递资源的ID（R.raw.<filename>）来打开它。这个方法返回一个[InputStream](#)，你可以使用它读取文件，但不能够写入这个原始文件。

保存缓存文件

如果你想要缓存一些数据，而不是保存它们，你应该调用[getCacheDir\(\)](#)打开一个[File](#)对象，它表示你的应用应当保存临时缓存文件的内部目录。

当设备处于内部低存储空间时，安卓会删除这些缓存文件以便恢复空间。但是不应该依靠系统来清理这些文件，而应该始终保持缓存文件在合理消耗空间，比如1MB。当用户卸载应用程序时都会删除这些文件。

其他有用的方法

[getFilesDir\(\)](#)

取得内部文件在文件系统中保存位置的绝对路径。

[getDir\(\)](#)

创建（或者打开已存在的）内部存储空间所在的目录。

[deleteFile\(\)](#)

删除内部存储的一个文件。

[fileList\(\)](#)

返回当前由你的应用保存的文件的列表。

使用外部存储 - Using the External Storage

所有兼容Android的设备都支持一个可共享的“**外部存储(external storage)**”，可用来保存文件。这可以使一个可移动的存储设备（比如SD卡）或者一个内部的（不可移动的）存储。保存在外部存储的文件是可读的。并且当用于传输数据的**USB大容量存储**选项启用时，用户能够在计算机上修改它们。

注意：如果用户挂载外部存储到计算机上，或者移除媒体，外部文件将会消失不见。并且对于这些保存在外部存储的文件，没有强制的安全措施。所有的应用都可以读/写这些文件。用户也能够删除它们。

获取外部存储访问

为了在外部存储读取或者写入文件，应用程序必须获取**READ_EXTERNAL_STORAGE** 或**WRITE_EXTERNAL_STORAGE**系统权限。例如：

```
<manifest ...>
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

如果要同时读取和写入文件，那就仅需要 **WRITE_EXTERNAL_STORAGE** 权

限，因为它隐形要求了读写访问。

注意：如果在安卓4.4读写私有文件，则不再需要这些权限。欲了解更多信息，请查阅下方关于保存私有文件的章节。

检测媒体可用性

在你对外部存储做任何事情之前，你总是应当调用[getExternalStorageState\(\)](#)以检测媒体是否可用。媒体可能被计算机挂载，可能丢失，可能只读，或者处于某些其他状态。比如，这是示例代码：

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))
{
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states,
    // but all we need
    // to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

这个例子检测了外部存储是否可读或可写。[getExternalStorageState\(\)](#)方法返回你想要检测的其他状态。比如，媒体是否被共享（已连接到一台计算机），是否完全丢失，是否被移除等等。当你的应用需要访问媒体时，你可以依据这些以更详细的信息通知用户。

保存与其他应用程序共享文件

用户通过应用程序获取的新文件通常要保存到设备的“公共”位置，这样其它应用程序可以访问文件，同时方便用户复制文件。这样就应该用于公共共享目录，比如Music/, Pictures/，和 Ringtones/。

为了获取适当公共目录的文件，调用getExternalStoragePublicDirectory()，把它传递给想要的目录类型，比如 DIRECTORY_MUSIC, DIRECTORY_PICTURES, DIRECTORY_RINGTONES等等。通过在相应媒体-类型目录保存文件，系统媒体扫描可以将文件妥善分类。

例如下面显示了在公开图片目录内创建新相册目录的方法：

```
public File getAlbumStorageDir (String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new
File(Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs ()) {
        Log.e (LOG_TAG, "Directory not created");
    }
    return file;
}
```

保存应用程序专用文件

如果处理应用程序专用文件（比如仅用于您应用程序的图形文本或者声音效果），就应该调用getExternalFilesDir()来使用外部存储的专用存储目录。此方法还需要指定子目录类型的类型函数（比如DIRECTORY_MOVIES）。如果不需要特定媒体目录，可以传递null来接收私有目录的根目录。

从安卓4.4开始，在应用程序私有目录读取或写入文件不再需要READ_EXTERNAL_STORAGE 或WRITE_EXTERNAL_STORAGE权限。因此声明此权限仅在安卓较低版本使用。

```
<manifest ...>
    <uses-permission
    android:name= "android.permission.WRITE_EXTERNAL_STORAGE"
                android:maxSdkVersion= "18" />
    ...
</manifest>
```

外部存储可以提供SD卡插槽，因此设备有时会分配内部存储分区。这一设备运行安卓4.3及更低版本时，`getExternalFilesDir()`方法仅提供了内部分区访问，应用程序无法读取或者写入SD卡。从安卓4.4开始就可以调用`getExternalFilesDirs()`同时访问两个位置了，`getExternalFilesDirs()`会返回各位置文件数组。数组第一个条目是主要外部存储。如果想在安卓4.3及更低版本同时访问这两个位置，就要使用支持库的静态方式`ContextCompat.getExternalFilesDirs()`。这样会返回文件数组，但始终仅包含安卓4.3及更低版本的一个条目。

保存应当共享的文件

如果你想要保存与你的应用没有关联的文件，并且这些文件不应在应用卸载时被删除，把他们保存在外部存储的一个公共目录上即可。这些目录位于外部存储的根目录下，比如`Music/`, `Pictures/`, `Ringtones/`, 以及其它目录。

在API Level 8或者更高版本，调

用`getExternalStoragePublicDirectory()`，传递你需要的公共目录的类型，比如`DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, `DIRECTORY_RINGTONES`或者其它。如有必要，这个方法会创建新目录。

如果你正在使用API Level 7或者更低版本，调用`getExternalStorageDirectory()`打开一个`File`，它表示外部存储的根目录。接着保存你的共享文件在以下文件夹之一：

- `Music/` - 媒体扫描器把所有在此发现的媒体归类为用户的音乐。
- `Podcasts/` - 媒体扫描器把所有在此发现的媒体归类为播客。
- `Ringtones/` - 媒体扫描器把所有在此发现的媒体归类为铃声。
- `Alarms/` - 媒体扫描器把所有在此发现的媒体归类为闹铃。
- `Notifications/` - 媒体扫描器把所有在此发现的媒体归类为提示音。
- `Pictures/` - 所有照片（相机拍摄的除外）。

在媒体扫描器下隐藏你的文件

在你的外部文件目录中放置包括一个空的文件，命名为`.nomedia`(注意文件名前缀的点)。这会阻止Android的媒体扫描器读取你的媒体文件，并并在类似`Gallery`或者`Music`这样的应用中包括它们。

- [Movies/](#) - 所有影片（摄像机拍摄的除外）。
- [Download/](#) - 各类下载。

保存缓存文件

如果你正在使用API Level 8或者更高版本，调用[getExternalCacheDir\(\)](#)打开一个[File](#)，它表示你的应用应当保存文件所在的外部存储目录。如果用户卸载了你的应用，这些文件会被自动删除。然而，在你的应用的生命周期中，你应当管理这些缓存文件以及删除其中过期的部分以保证文件空间。

如果你正在使用API Level 7或者更低版本，使用[getExternalStorageDirectory\(\)](#)打开一个[File](#)，它表示外部存储的根目录。接下来，你应当写入你的数据在一下目录：

`/Android/data/<package_name>/files/`

`<package_name>`是Java风格的包名称，例如"com.example.android.app"。

使用数据库 - Using Databases

Android提供了对[SQLite](#)数据库的完整支持。你创建的任何数据库都能被应用程序中的任意类通过数据库名访问。但是不能够在应用程序以外访问。

推荐的创建新的SQLite数据库的方法是，在你执行SQLite命令在数据库中创建表的时候，创建一个[SQLiteOpenHelper](#)的子类，并重写[onCreate\(\)](#)方法。例如：

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME =
    "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +

```

```
KEY_WORD + " TEXT, " +
KEY_DEFINITION + " TEXT);";
```

```
DictionaryOpenHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(DICTIONARY_TABLE_CREATE);
}
```

接着，你可以使用构造方法取得你实现的[SQLiteOpenHelper](#)的一个实例。分别使用[getWritableDatabase\(\)](#)和[getReadableDatabase\(\)](#)读写数据库。两者都会返回一个[SQLiteDatabase](#)对象，代表那个数据库，并提供操作[SQLite](#)的方法。

你可以使
用[SQLiteDatabase.query\(\)](#)方法执行[SQLite](#)查询，这个方法接受各种查询参数，例如待查询的表、投影、选择、列、组，或者其他。对于复杂查询，比如一些需要用到列的别名的，你应当使
用[SQLiteQueryBuilder](#)，它能提供一些合适的方法以创建查询。

每一个[SQLite](#)查询都会返回一个[Cursor](#)对象，指向查询得到的所有行。[Cursor](#)对象总是你用来取得数据库查询或者读取行或列的结果的途径。

Android没有在标准[SQLite](#)概念之外加以任何限制。我们建议包含一个自增值的主键字段，能够作为一个唯一的ID，以便快速定位一条记录。这对私有数据来说不是必须的。但如果你实现了一个[content provider](#)，就必须包含一个唯一的ID，它使用[BaseColumns._ID](#)常量。

关于示例应用中演示如何在Android中使用[SQLite](#)数据库，见[Node Pad](#)和[Searchable Dictionary](#)这两个应用。

Android SDK包含了一个`sqlite3`的数据库工具，允许你浏览表内容，运行SQL命令，以及执行其他SQLite数据库中有用的功能。想要了解如何运行这个工具，请查看[Examining sqlite3 databases from a remote shell](#)这个例子。

使用网络连接 - Using a Network Connection

你可以使用网络连接(当可用时)通过基于web的服务存储或取得数据。需要执行网络操作时，使用以下包中的类：

[java.net.*](#)

[android.net.*](#)

来自“[index.php?title=Storage_Options&oldid=13829](#)”



Data Backup

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/intl/zh-CN/guide/topics/data/backup.html>

译者：贼寇在何方

完成时间：2012年7月19日

Android的[备份\(backup\)](#)服务允许你复制持久化的应用数据到远程的云储存，以此为应用程序数据和配置提供一个还原点。如果用户执行了一次重置为出厂状态或者转到了一台新的Android设备，当应用程序重新安装时，系统会自动恢复你的备份数据。通过这个方法，你的用户不必要复制它们之前的数据和应用程序配置。这个过程对用户是完全透明的，并且不会影响应用程序的功能和用户体验。

在备份操作中（你的应用程序能够请求的情况下），Android的备份管理器([BackupManager](#))会查询你的应用的备份数据，然后传递这些数据到一个备份传输系统，这个系统稍后将把数据传输到云储存。在还原操作中，备份管理器从备份传输系统中取回数据，并把数据交还给你的应用，以便于它能够把数据还原到设备上。你的应用是能够请求还原的，但是这并不是必要的——当你的应用安装并且已经存在关联到这个用户的备份数据时，Android会自动执行还原操作。在选择备份数据进行恢复的最好的时机是，用户重置他们的设备或者更新到一台新设备时，并且他们之前安装的应用程序重新被安装。

注意：备份服务并不是设计用于与其他客户端同步应用程序数据，或者用于在通常的应用程序生命周期中保存你想访问的数据。你不能够请求读/写备份数据，也不能够通过除备份管理器的API以外的方式访问备份数据。

备份传输系统是Android备份框架的客户端组件，它可以由设备制造商或者服务提供者定制。备份管理器的API将你的应用与给定设备上的备份系统隔离开来——你的应用通过固定的API与备份管理器通信，无视底层的传输系统。

数据备份并不保证在所有Android设备上可用。然后，即使设备不提供一个备份传输系统，您的应用也不会受到不利影响。如果你相信，在你的应用中，用户能够从数据备份中受益，那么你可以依照这篇文档的描述实现它。测试完成后，不必关心哪些设备运行了备份，直接发布你的应用即可。当你的应用运行在一个不提供备份传输系统的设备上，它能够正常执行，但不会从备份管理器发来的备份数据的回调。

尽管你不能够知道当前的备份传输是什么，但你总是能够确定你的备份数据不会被设备上的其他应用程序读取。只有备份管理器和备份传输系统能够访问备份操作产生的数据。

小心：因为云储存和传输服务可以区分不同的设备，所以在使用备份功能的时候，Android不保证数据的安全。你总是应当小心使用备份存放敏感数据，比如用户名和密码。

目录

[[隐藏](#)]

[1 基础 - The Basics](#)

[2 在清单中声明备份代理 - Declaring the Backup Agent in Your Manifest](#)

[3 注册Android备份服务 - Registering for Android Backup Service](#)

[4 扩展备份代理 - Extending BackupAgent](#)

- [4.1 需要的方法](#)
- [4.2 执行备份](#)
- [4.3 执行还原](#)

[5 扩展备份代理助手 - Extending BackupAgentHelper](#)

- [5.1 备份共享配置-Backing up SharedPreferences](#)
- [5.2 备份其他文件](#)

[6 检查还原数据的版本 - Checking the Restore Data Version](#)

[7 请求备份 - Requesting Backup](#)

8 请求还原 - Requesting Restore

9 测试你的备份代理 - Testing Your Backup Agent

基础 - The Basics

想要备份你的应用数据，你需要实现一个备份代理。你的备份代理会被备份管理器调用，以提供你想要还原需要的数据。当应用被重装时，备份代理也会被调用，用以还原备份数据。备份管理器管理着你所有在云储存（使用备份传输系统）上的数据事务。与此同时，你的备份代理管理着所有在你的设备上的数据事务。

为了实现一个备份代理，你必须这么做：

1. 在你的清单文件中声明你的备份代理，使用[android:backupAgent](#)属性。
2. 使用备份服务注册你的应用。谷歌提供了[Android Backup Service](#)用作大多数Android设备的备份服务。它需要你注册你的应用才能够工作。为了把你的数据存放在他们的服务器上，任意其他的可用备份服务也可能会要求你注册。
3. 通过以下的一种方法定义一个备份的代理：

1. 扩展BackupAgent

[BackupAgent](#)类提供了核心接口，用于应用程序和备份管理器之间的通信。如果你直接扩展这个类，你必须重写[onBackup\(\)](#)和[onRestore\(\)](#)两个方法，以控制备份和还原操作。

2. 扩展BackupAgentHelper

[BackupAgentHelper](#)类提供了[BackupAgent](#)类的一个合适的包装器。这个包装器最大程度减少了你需要编写代码的数量。在使用时，你必须用到一个或多个「helper」对象。它能够自动备份或恢复特定类型的数据，因此你不必实现[onBackup\(\)](#)和[onRestore\(\)](#)。

现在，Android提供了备份助手，能够备份和还原来自[共享配](#)

置和内部存储的完整文件。

在清单中声明备份代理 - Declaring the Backup Agent in Your Manifest

这是第一步，所以一旦你定下了备份代理的类名，就在你的清单中声明——在`<application>`标签中使用`android:backupAgent`属性。

例如：

```
<manifest ... >
    ...
    <application android:label="MyApplication"
                  android:backupAgent="MyBackupAgent" >
        <activity ... >
            ...
        </activity>
    </application>
</manifest>
```

另一个属性你也许会用到的是[android:restoreAnyVersion](#)。这个属性使用一个布尔值，用于指定你是否想要重置应用程序数据，并且不计较当前应用程序版本与生产这些数据的应用程序版本是否一致。（缺省值是「false」）。查看[Checking the Restore Data Version](#)以获得更多信息。

注意：备份服务和你必须使用的API只在运行API Level 8 (Android 2.2) 或者更高版本的设备上可用。因此，你应当设置[android:minSdkVersion](#)属性为「8」。然而，如果你要在应用中实现合适的[向后兼容](#)，你可以在运行API Level 8或者更高版本的设备上支持这个特性，同时为旧的设备保持兼容。

注册Android备份服务 - Registering for Android Backup Service

谷歌使用[Android备份服务 \(Android Backup Service\)](#)为大多数运行Android 2.2或更高版本的设备提供了一个备份传输服务。

为了让应用程序使用Android备份服务执行备份，你必须用这个服务注册应用程序，接收一个备份服务Key，然后在你的Android清单中声明这个Key。

为了取得备份服务Key，请[注册Android备份服务](#)。当你注册的时候，你会收到一个备份服务Key和对应的`<meta-data>`XML代码。你必须把这段XML代码放在清单文件的`<application>`元素下面作为它的子结点。例

如：

```
<application android:label="MyApplication"
    android:backupAgent="MyBackupAgent">
    ...
    <meta-data android:name="com.google.android.backup.api_key"
        android:value="AEdPqrEAAAAIDaYEVgU6DJnyJdBmU7KLH3kszDXLv_4DISeIyQ" />
</application>
```

android:name必须是「`com.google.android.backup.api_key`」，且android:value必须是从Android备份服务注册收到的备份服务Key。

如果你有多个应用程序，你必须用各自的包名称为它们每一个注册。

注意：由Android备份服务提供的备份传输系统不保证所有基于Android的设备都支持备份功能。一些设备可能使用不同的传输系统支持备份功能，另外一些可能根本不支持。对于你的应用程序，没有一种方法能够知道设备使用的是哪一种传输系统。然而，如果你的应用实现了备份功能，那么你总是应当包含一个备份服务Key。当设备使用Android备份服务传输系统时，这个Key用于Android备份服务，使你的应用程序能够执行备份。如果设备不适用Android备份服务，那么`<meta-data>`元素和备份服务Key会被忽略。

扩展备份代理 - Extending BackupAgent

绝大多数应用程序没必要直接扩展[BackupAgent](#)类，但应当[扩展BackupAgentHelper](#)以发挥内置辅助类的优势——自动备份和还原你的文件。当然，如有必要，你可以用下面的方法直接扩展[BackupAgent](#)：

- 为你的数据格式加上版本。比如，如果你预计到需要改变写入的应用程序数据格式，你可以建立一个备份代理，用于在执行恢复操作时再次确认你的应用程序版本，并且执行任何必要的兼容性工作（如果当前设备的版本与备份数据不一致）。详见[Checking the Restore Data Version](#)。
- 替代复制整个文件这种方式，你可以指定应当被复制的那一部分数据以及每一部分之后如何还原到设备上。（这也能帮助你管理不同的版本，因为你把数据当做特定的实体来读取，而不是完整的文件。）
- 复制数据库中的数据。如果你想在重装应用时恢复SQLite数据库，你应当创建一个自定义的[BackupAgent](#)（用于在执行还原操作时读取需要的

数据），然后在执行还原时创建你的表并插入数据。

如果你不需要执行以上任意一项任务，并且只是想完整地恢复[共享配置](#)或者[内部存储](#)中的文件，你应当跳过[扩展备份代理](#)这一节。

需要的方法

当你通过扩展[BackupAgent](#)创建一个备份代理，你必须实现以下回调方法：

[onBackup\(\)](#) 在你[请求备份](#)之后，备份管理器会调用这个方法。在这个方法中，你需要从设备中读取应用程序数据，并传递你要备份的数据到备份管理器。详细内容在之后的[执行备份](#)中提到。

[onRestore\(\)](#) 在还原操作中，备份管理器会调用这个方法（你可以[请求还原](#)，但系统会用户重装你的应用时自动执行还原）。当方法被调用时，备份管理器会传递你的备份数据，接着还原到设备上。详细内容在之后的[执行还原](#)中提到。

执行备份

当你的应用程序数据备份时，备份管理器调用你的[onBackup\(\)](#)方法。此处，你必须提供你的应用程序数据给备份管理器，以便于它能够保存数据到云储存上。

只有备份管理器能够调用你的备份代理的[onBackup\(\)](#)方法。每当你应用程序数据改变或者你想要执行一次备份时，你必须调用[dataChanged\(\)](#)方法请求备份（详见[请求备份](#)）。一次备份请求并不会在你的[onBackup\(\)](#)方法调用时立即返回结果。相反地，备份管理器会等待一段时间，然后为所有自上次备份执行以来请求备份的应用执行备份。

提示：当你开发你的应用时，你可以使用[bmgr工具](#)，从备份管理器发起一个立即执行的备份操作。

当备份管理器调用你的[onBackup\(\)](#)方法，它会传递以下三个参数：

oldState

一个公开的、只读的[ParcelFileDescriptor](#)对象，指向由你的应用提供的上一次备份状态。这不是来自云储存的备份数据，而是本地数据的一个标记，

是在上一次调用`onBackup()`时备份（由下面的`newState`定义，或者来自`onRestore()`——更多信息在下一章）。由于`onBackup()`不允许你读取云储存上已有的备份数据，你可以使用这个本地的标记来判断从上次备份以来数据是否发生改变。

data

一个`BackupDataOutput`对象，用于传递你的备份数据到备份管理器。

newState

一个公开的、可读可写的`ParcelFileDescriptor`对象，指向一个文件。在这个文件中，你必须写入一个你传递数据的标记（一个标记可以简单地使用你的文件最后一次修改的时间戳）。在下一次备份管理器调用`onBackup()`方法时，这个对象会被当做`oldState`被返回。如果你没有往`newState`写入备份数据，那么下一次备份管理器调用`onBackup()`时，`oldState`将会指向一个空文件。

使用这些参数，你应当事先你的`onBackup()`方法，做以下事情：

1. 通过比较`oldState`和你现在的数据，检查是否数据在上次备份后改变。你在`oldState`中读取数据的方式依赖于起初你写入`newState`的方式（见第3步）。记录文件状态最简单的方法是记录它的最后一次修改。这个例子是如何从`oldState`读取并比较一个时间戳的：

```
// Get the oldState input stream
FileInputStream instream = new
FileInputStream( oldState.getFileDescriptor() );
DataInputStream in = new DataInputStream( instream );

try {
    // Get the last modified timestamp from the state file and data
    // file
    long stateModified = in.readLong();
    long fileModified = mDataFile.lastModified();

    if ( stateModified != fileModified ) {
        // The file has been modified, so do a backup
        // Or the time on the device changed, so be safe and do a
        backup
    } else {
        // Don't back up because the file hasn't changed
        return;
    }
} catch ( IOException e ) {
    // Unable to read state file... be safe and do a backup
}
```

}

如果没有任何改变，并且你不需要备份，跳到第3步。

1. 如果你的数据对比`oldState`已发生改变，那么写入当前数据到`data`参数中，以备份到云储存的。

你必须把每一块数据当做一个"实体"写入`BackupDataOutput`对象。实体是一个扁平的二进制数据记录，由一个唯一的字符串键。那样，你备份的数据集就是一个键值对的集合。

为了向备份数据集添加实体，你必须：

调用`writeEntityHeader()`，传入一个唯一字符串键（指示准备写入的数据）和数据大小。

调用`[,%20int) writeEntityData()`，传入包含了数据的字节缓存和需要从缓存写入的字节数量（这与传入`writeEntityHeader()`的数据大小必须匹配）。

例如，下面的代码把一些数据放入字节流，并写入到单个实体：

```
// Create buffer stream and data output stream for our data
ByteArrayOutputStream bufStream = new ByteArrayOutputStream();
DataOutputStream outWriter = new DataOutputStream(bufStream);
// Write structured data
outWriter.writeUTF(mPlayerName);
outWriter.writeInt(mPlayerScore);
// Send the data to the Backup Manager via the BackupDataOutput
byte[] buffer = bufStream.toByteArray();
int len = buffer.length;
data.writeEntityHeader(TOPSCORE_BACKUP_KEY, len);
data.writeEntityData(buffer, len);
```

为每一部分需要备份的数据执行这一步操作。分割数据并写入实体的方式由你自己决定（并且你可以仅仅使用一个实体）。

不论是否执行一次备份（在第2步），向`newState ParcelFileDescriptor`对象写入一个当前数据的标记。备份管理器在本地保留这个对象，作为当前备份的数据的一个标记。当它下一次调用`onBackup()`，它会把这个标记作为`oldState`传递给你。这样你能够决定是否需要另一次备份（就如第1步操作的那样）。如果你把当前数据的状态写入这个文件，那么`oldState`在下一次回调时保持空值。

下面的例子保存了当前数据的一个标志（使用文件的最后一次修改的时间戳）到newState：

```
FileOutputStream outstream = new
FileOutputStream(newState.getFileDescriptor());
DataOutputStream out = new DataOutputStream(outstream);

long modified = mDataFile.lastModified();
out.writeLong(modified);
```

小心：如果你的应用程序数据保存在一个文件内，那么确认读取文件时使用同步状态。这样能确保当你的应用中的一个Activity在写入数据时，你的备份代理不在读取文件。

执行还原

当需要还原你的应用程序数据时，备份管理器会调用你的备份代理的onRestore()方法。此时，备份管理器传递你的备份数据，以便于你能够把数据还原到设备上。

只用备份管理器能够调用onRestore()。这发生在系统安装你的应用并发现已存在的备份数据的时候。然后，你可以通过调用[requestRestore\(\)](#)请求还原操作（详见[Requesting restore](#)）。

注意：当你开发应用是，你也可以使用bmgr工具请求还原操作。

当备份管理器调用你的onRestore()方法时，它传递三个参数：

data

一个[BackupDataInput](#)对象。它允许你读取备份数据。

appVersionCode

一个整数，标记你的应用程序的[android:versionCode](#)清单的属性值，这个值是当前数据备份的时间。你可以使用这个来再次确认当前应用程序的版本，并判断数据格式是否兼容。关于使用这个参数操作不同版本的恢复数据，详见下一章：[Checking the Restore Data Version](#)。

newState

一个公开的、可读可写的ParcelFileDescriptor对象，指向一个文件。在这个文件中，你必须写入数据提供的最终备份状态。当下一次onBackup()被调用，这个对象会作为oldState被返回。重复调用，你必须写入同样的newState在onBackup()的回调中。同时，这样可以确保传给onBackup()的oldState对象是合法的，即使是在设备恢复后第一次调用onBackup()。

在你的onRestore()实现中，你应当对数据调用readNextHeader()以便利所有数据集中的实体。对每一个找到的实体，执行以下操作：

1. 使用[getKey\(\)](#)取得实体键。
2. 将实体键与你已经在BackupAgent类中声明的一系列字符串（被static和final修饰）一一比较。当这个键与已知键匹配时，输入一个报表以提取实体数据，并保存到设备上：
 1. 调用[getDataSize\(\)](#)取得实体数据的大小，并且创建一个同样大的字节数组。
 2. 调用[\[,%20int,%20int\] readEntityData\(\)](#)并传入这个字节数组——数据将会放入其中——并指定起始位置和读取数据的大小。
 3. 现在，字节数组已经填满，你可以读取数据并随意写入到设备。
3. 当你读取并写回数据到设备上之后，写入数据状态到newState参数，就如你在调用onBackup()时一样。

例如，这是恢复之前章节备份的数据的方法：

```
@Override
public void onRestore(BackupDataInput data, int appVersionCode,
                      ParcelFileDescriptor newState) throws
IOException {
    // There should be only one entity, but the safest
    // way to consume it is using a while loop
    while (data.readNextHeader()) {
        String key = data.getKey();
        int dataSize = data.getDataSize();

        // If the key is ours (for saving top score). Note this key
        was used when
        // we wrote the backup entity header
        if (TOPSCORE_BACKUP_KEY.equals(key)) {
            // Create an input stream for the BackupDataInput
            byte[] dataBuf = new byte[dataSize];
            data.readEntityData(dataBuf, 0, dataSize);
            ByteArrayInputStream baStream = new
```

```

ByteArrayInputStream(dataBuf);
    DataInputStream in = new DataInputStream(baStream);

        // Read the player name and score from the backup data
        mPlayerName = in.readUTF();
        mPlayerScore = in.readInt();

        // Record the score on the device (to a file or
        something)
        recordScore(mPlayerName, mPlayerScore);
    } else {
        // We don't know this entity key. Skip it. (Shouldn't
        happen.)
        data.skipEntityData();
    }
}

// Finally, write to the state blob (newState) that describes
the restored data
FileOutputStream outstream = new
FileOutputStream(newState.getFileDescriptor());
DataOutputStream out = new DataOutputStream(outstream);
out.writeUTF(mPlayerName);
out.writeInt(mPlayerScore);
}

```

在这个例子里，传递给onRestore()的参数appVersionCode未被使用。然而，当用户的应用程序版本确实后退了，并且你执行过备份，你可能用得着它（例如，用户想要从版本1.5回到1.0）。详见[Checking the Restore Data Version](#)这一章。

如果需要[BackupAgent](#)实现的一个例子，详见[Backup and Restore](#)示例程序中的[ExampleAgent](#)。

扩展备份代理助手 - Extending BackupAgentHelper

如果你想要完整地备份文件（不论是来自共享配置或是内部存储），你应当使用[BackupAgentHelper](#)构建你的备份代理。使用这个方法所需的代码量远远少于扩展[BackupAgent](#)类，因为你不需要实现onBackup()和onRestore()。

你的[BackupAgentHelper](#)实现必须要使用至少一个备份助手。备份助手是一个[BackupAgentHelper](#)调用的专用组件，用于对特定类型的数据执行备份和还原操作。Android框架现在提供了两种不同的助手：

- [SharedPreferencesBackupHelper](#)用于备份共享配置（[SharedPreferences](#)）的文件。

- FileBackupHelper用于备份来自内部存储的文件。

你能够在BackupAgentHelper中包含多个助手，但是只有一个助手对每一个数据类型有用。因此，如果你有多个共享配置文件，那你只需要一个SharedPreferencesBackupHelper。

对于每一个想加入BackupAgentHelper的助手，你必须在onCreate()方法中做以下几件事：

1. 在你期望的助手类对象中实例化。在类的构造方法中，你必须指定你想要备份的文件。
2. 通过调用addHelper()添加助手到你的BackupAgentHelper。

下面几节描述了如何使用每一个可用的助手创建备份代理。

备份共享配置-Backing up SharedPreferences

当你实例化一个SharedPreferencesBackupHelper，你一定要包含一个或多个共享配置文件的名称。

例如，要备份一个名称为"user_preferences"的共享配置文件。一个使用BackupAgentHelper的完整的备份代理看起来像这样：

```
public class MyPrefsBackupAgent extends BackupAgentHelper {
    // The name of the SharedPreferences file
    static final String PREFS = "user_preferences";

    // A key to uniquely identify the set of backup data
    static final String PREFS_BACKUP_KEY = "prefs";

    // Allocate a helper and add it to the backup agent
    @Override
    public void onCreate() {
        SharedPreferencesBackupHelper helper = new
    SharedPreferencesBackupHelper(this, PREFS);
        addHelper(PREFS_BACKUP_KEY, helper);
    }
}
```

就是它！这是整个备份代理。SharedPreferencesBackupHelper类包含了所有用于备份和还原一个SharedPreferences文件的代码。

当备份管理器调用onBackup()和onRestore()，BackupAgentHelper调用你的

备份助手来执行指定文件的备份和还原。

注意：SharedPreferences是线程安全的，所以你可以从你的备份代理和其他Activity安全地读写共享配置文件。

备份其他文件

当你实例化一个[FileBackupHelper](#)对象，你必须包含至少一个文件，这些文件会被保存到你的应用程序的[内部存储](#)（由[getFilesDir\(\)](#)指定，与你调用[int openFileOutput\(\)](#)写入文件的位置一致）。

例如，要备份两个名字分别为"scores"和"stats"的文件，使用[BackupAgentHelper](#)的备份代理应该类似这样：

```
public class MyFileBackupAgent extends BackupAgentHelper {
    // The name of the SharedPreferences file
    static final String TOP_SCORES = "scores";
    static final String PLAYER_STATS = "stats";

    // A key to uniquely identify the set of backup data
    static final String FILES_BACKUP_KEY = "myfiles";

    // Allocate a helper and add it to the backup agent
    void onCreate() {
        FileBackupHelper helper = new FileBackupHelper(this,
TOP_SCORES, PLAYER_STATS);
        addHelper(FILES_BACKUP_KEY, helper);
    }
}
```

这个[FileBackupHelper](#)类包含所有必要的代码，以备份和还原保存到你的应用的内部存储的文件。

然而，在内部存储中读取和写入文件并不是线程安全的。为确保你的备份代理不在一个时间读取或写入你的文件，你必须在每一次执行读或者写操作时使用同步的状态。例如，在任何的Activity内读取和写入文件时，你需要使用一个对象为同步状态提供一个固有的锁：

```
// Object for intrinsic lock
static final Object sDataLock = new Object();
```

然后在每一次读取或写入文件时，使用这个锁创建一个同步的状态。例如，这是一个同步状态，用于写入游戏中的最后一次得分到一个文件：

```
try {
    synchronized (MyActivity.sDataLock) {
```

```

        File dataFile = new File(getFilesDir(), TOP_SCORES);
        RandomAccessFile raFile = new RandomAccessFile(dataFile,
"rw");
        raFile.writeInt(score);
    }
} catch (IOException e) {
    Log.e(TAG, "Unable to write to file");
}

```

你应当使用同一个锁同步你的读取状态。

接着，在你的[BackupAgentHelper](#)类中，你必须重写[onBackup\(\)](#)和[onRestore\(\)](#)，使用同一个固有的锁同步备份和还原操作。例如，上面[MyFileBackupAgent](#)这个例子需要以下方法：

```

@Override
public void onBackup(ParcelFileDescriptor oldState, BackupDataOutput
data,
        ParcelFileDescriptor newState) throws IOException {
    // Hold the lock while the FileBackupHelper performs backup
    synchronized (MyActivity.sDataLock) {
        super.onBackup(oldState, data, newState);
    }
}

@Override
public void onRestore(BackupDataInput data, int appVersionCode,
        ParcelFileDescriptor newState) throws IOException {
    // Hold the lock while the FileBackupHelper restores the file
    synchronized (MyActivity.sDataLock) {
        super.onRestore(data, appVersionCode, newState);
    }
}

```

就这些了。所有你需要做的仅仅是添加你的[FileBackupHelper](#)对象到[onCreate\(\)](#)方法，重写[onBackup\(\)](#)和[onRestore\(\)](#)以同步读写操作。

想要一个带[FileBackupHelper](#)的[BackupAgentHelper](#)实现的例子，在Backup and Restore的实例应用中见[FileHelperExampleAgent](#)类。

检查还原数据的版本 - Checking the Restore Data Version

当备份管理器保存你的数据到云储存，它会自动包含你的应用程序的版本——由你清单文件中的[android:versionCode](#)属性指定。在备份管理器调用你的备份代理还原数据之前，它会检查已安装应用的[android:versionCode](#)，并与还原数据集中的记录相比较。如果记录在还原数据集中的版本比设备上的应用程序版本更新时，说明用户给应用降级

了。在这种情况下，备份管理器将会终止还原操作，不再调用你的[onRestore\(\)](#)方法，因为还原数据到一个旧的版本是没有意义的。

你可以依据[android:restoreAnyVersion](#)属性重写这个行为。这个属性可以是"true"或者"false"，用于指定你是否想要还原应用程序并忽略还原数据集版本。缺省值是"false"。如果你定义它为"true"，那么备份管理器会忽略[android:versionCode](#)，并在任意情况下都会调用[onRestore\(\)](#)。在这种情况下，你可以在[onRestore\(\)](#)方法内手动检查版本差异，并采取任何必要的措施保证数据的兼容性（如果版本冲突的话）。

为了帮助你在还原操作中控制不同的版本，[onRestore\(\)](#)方法会传递你的版本代码（包含在还原数据集）作为appVersionCode参数。接着，你可以通过[PackageManager.versionCode](#)字段查询当前应用程序的版本代码。例如：

```
PackageManager info;
try {
    String name = getPackageName();
    info = getPackageManager().getPackageManager(name, 0);
} catch (NameNotFoundException nnfe) {
    info = null;
}

int version;
if (info != null) {
    version = info.versionCode;
}
```

Then simply compare the version acquired from PackageManager to the appVersionCode passed into [onRestore\(\)](#). 接下来，简单地比较从PackageManager取得的版本与传递给onRestore()的appVersionCode。

小心：确认你明白设置[android:restoreAnyVersion](#)为"true"对你的应用意味着什么。如果应用程序的每一个支持备份的版本不会在[onRestore\(\)](#)期间引发数据格式的改变，那么设备上的数据可以以一种并不兼容于当前安装版本的格式保存。

请求备份 - Requesting Backup

你可以在任意时间调用[dataChanged\(\)](#)请求备份操作。这个方法提醒备份管理器，你想要使用你的备份代理备份数据。接着，备份管理器会在未来某个合适的时刻回调备份代理的[onBackup\(\)](#)方法。一般来说，你应当在每一次数据改变时请求一次备份（例如当用户更改了应用程序配置，而这是你

想要备份的）。如果你在备份管理器从你的代理请求备份之前，连续调用`dataChanged()`，你的代理依旧只调用一次`onBackup()`。

注意：当开发应用时，你可以请求备份并立即开始备份操作，这需要使用`bmgr`工具。

请求还原 - Requesting Restore

在应用程序正常的生命周期中，你不需要请求还原。在应用程序安装时，系统会自动检查备份数据，并执行还原。然而，如果有必要，你可以调用`requestRestore()`手动请求还原。在这种情况下，备份管理器调用你的`onRestore()`实现，同时传递来自当前备份数据集的数据。

注意：开发应用时，你可以使用`bmgr`工具请求还原。

测试你的备份代理 - Testing Your Backup Agent

实现了你的备份代理之后，你可以使用`bmgr`，通过一下步骤，测试备份和还原功能：

1、在合适的Android系统映像上安装你的应用

- 如果使用模拟机，则创建并使用一个AVD，带Android 2.2(API Level 8)。
- 如果使用一台设备，则这台设备必须运行Android 2.2或者更高版本，并有内置Google Play。

2、确认备份功能已启用

- 如果使用模拟机，你可以使用以下命令启用备份——来自SDK中的tool/路径：

```
adb shell bmgr enable true
```

- 如果使用一台设备，则打开系统设置，选择隐私，然后启用备份我的数据和自动恢复。

3、打开你的应用并初始化一些数据

如果你正确地实现了备份功能，那么它应在每一次数据改变时请求一次备份。例如，每一次用户修改了一些数据，你的应调用**dataChanged()**。这个方法会添加一次备份请求到备份管理器的队列中。出于测试的目的，你也可以用下面的**bmgr**命令发起一次请求：

```
adb shell bmgr backup your.package.name
```

4、初始化一个备份操作：

```
adb shell bmgr run
```

这个命令强制备份管理器执行队列中所有备份请求。

5、卸载你的应用：

```
adb uninstall your.package.name
```

6、重装你的应用：

如果你的备份代理是正确的，所以的你在第4步中初始化的数据会被恢复。

来自“[index.php?title=Data_Backup&oldid=6239](#)”



App Install Location

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/topics/data/install-location.html>

编辑者： eoe耗子

更新时间： 2012.07.25

目录

[[隐藏](#)]

[1 概览](#)

- [1.1 快速预览](#)

[2 向后兼容](#)

[3 不应装在外部存储设备的程序](#)

[4 应该装在外部存储设备的程序](#)

概览

从API level 8开始，用户可以将应用程序安装到外部存储设备中

（如SD卡）。该属性可以通过[android:installLocation](#)属性在manifest中声明。如果不声明该属性，应用程序将只会安装在内存中，而且不能移动到外部存储设备。

要将应用程序安装到外部存储设备中，只需修改manifest文件，在[manifest](#)元素中添

快速预览

- 用户可以将应用程序安装到外部存储设备中
- 有些类型的应用程序不应装在外部存储设备中
- 与系统集成度较低的大型应用程序（大多是游戏）安装在外部存储设备中是理想之选

加`android:installLocation`属性，属性值为`preferExternal`或`auto`。例如：

参见

[manifest](#)

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:installLocation="preferExternal"
    ... >
```

如果

该属性值设为`"preferExternal"`，表明用户要求应用程序安装在外部存储设备中，但是系统并不保证应用程序真的安装在外部存储设备中。如果外部存储设备的空间已满，系统仍然会将应用程序安装在内存中。用户也可以在两个位置之间移动应用程序。

如果该属性值设为`"auto"`，表明应用程序可能安装在外部存储设备中，但是用户本身对安装位置没有特殊要求。系统会根据多种因素决定将应用程序安装在哪里。用户也可以在两个位置之间移动应用程序。

当应用程序安装在外部存储设备中时：

- 只要外部存储设备装载在设备上，应用程序的性能就不会受到影响。
- `.apk`文件是保存在外部存储设备中的，但是所有的私有用户数据、数据库、优化的`.dex`文件和提取的本地代码都是保存在设备的内存中的。
- 保存应用程序的唯一容器经过一个随机产生的密钥进行加密。该密钥只能由最初安装该程序的设备解密。因此，安装在SD卡上的应用程序只能在某一个固定的设备上运行。
- 用户可以通过系统设置将应用程序移动到内存上。

注意：当用户使用USB大容量存储器与计算机共享文件时，或通过系统设置卸载SD卡时，外部存储设备将从本设备卸载，并且所有在该外部存储设备中运行的应用程序将立刻被关闭。

向后兼容

应用程序能够安装在外部存储设备中是API level 8 (android 2.2) 以上的

设备的特点。在此之前的的应用程序只能安装在内存中，且不能移动到外部存储设备中（即使设备是API level 8的）。然而，如果应用程序设计的时候就支持低于API level 8的设备的，那么用户可以选择在API level 8及以上的设备上是否需要支持这一特性，并且在使用API level 8 以下的设备的时候仍然兼容。

要在外部存储设备中安装程序，并且与低于API level 8 的版本兼容，需要做到以下几点：

1. 在[manifest](#)元素中添加`android:installLocation`属性，设属性值为`"auto"`或`"preferExternal"`。
2. 保持`android:minSdkVersion`属性不变（低于8的某个值），确保应用程序的代码能够兼容该等级。
3. 为了能够编译程序，将生成目标设为API level 8。这一步必不可少，因为旧的`android`库不识别`android:installLocation`属性，当该属性存在时，应用程序将无法编译通过。

当应用程序安装在低于API level 8 的设备中

时，`android:installLocation`属性将被忽略，应用程序将安装在内存中。

注意: 尽管在老的平台上，类似这种的**XML**标记会被忽略，但是在`android:minSdkVersion`低于8时，编程时也一定要注意避免使用API level 8 推荐的写法，除非在代码中已经提供了向后兼容。更多关于应用程序代码中的向后兼容的信息，请参考[Backward Compatibility](#)条目。

不应装在外部存储设备的程序

当用户使用**USB**大容量存储器与计算机共享文件时（或相反地，卸载或移除外部存储设备时），任何安装在外部存储设备中、正在运行的应用程序将会被关闭。直到大容量存储设备不可用、随机又重新加载到设备中时，系统才会无法识别应用程序。除了关闭应用程序使其不可用之外，有些情况下会严重损坏某种类型的 应用程序。鉴于以上后果，如果应用程序有以下特点，为了让应用程序持续照常运行，用户应该允许应用程序安装在外部存储设备上。

服务

当外部存储设备重新加载时，原本运行的**服务**将被关闭，且不能重启。然而，用户可以注册[ACTION_EXTERNAL_APPLICATIONS_AVAILABLE](#)广播，

该广播会通知安装在外部存储设备上的应用程序系统再次可用。那时，用户就可以重启服务了。

警告服务

使用[AlarmManager](#)注册的所有警告将会取消。当外部存储设备重新加载时，用户必须重新手动注册。

输入法

用户的输入法将会由默认的代替。当外部存储设备重新加载时，用户可以通过系统设置重新启用自己的输入法。

动态壁纸

运行中的动态壁纸将会被默认的动态壁纸代替。当外部存储设备重新加载时，用户可以重新选择动态壁纸。

活动文件夹

活动文件夹将会从主页面移除。当外部存储设备重新加载时，用户可以重新添加活动文件夹到主页面。

窗口小部件

用户的[应用程序窗口小部件](#)将会从主页面移除。当外部存储设备重新加载时，窗口小部件不可用，除非用户重置了主页面应用程序（通常是系统重启）。

帐号管理

由[AccountManager](#)创建的帐户将会消失，直到外部存储设备重新加载为止。

同步适配器

[AbstractThreadedSyncAdapter](#)及其所有的同步功能将失效，直到外部存储设备重新加载为止。

设备管理器

[DeviceAdminReceiver](#)及其管理功能将会失效，这将会给设备功能带来无法预料的后果，这个后果将一直持续到外部存储设备重新加载为止。

监听“启动完成”的广播接收器

在外部存储设备加载到设备之前，系统负责[ACTION_BOOT_COMPLETED](#)广播。如果应用程序安装在外部存储设备中，将无法接收到这个广播。

复制保护

如果使用了android电子市场的复制保护，应用程序将无法安装在SD卡上。然而，如果使用了android电子市场的应用程序许可证，应用程序将可以安装在内存或外部存储设备上，包括SD卡。

如果应用程序包含以上特点，那么用户不应该将应用程序安装在外部存储

设备中。默认情况下，系统不允许将应用程序安装到外部存储设备中，所以用户不用担心已经存在的应用程序。然而，如果用户非常肯定应用程序永远都不会安装在外部存储设备中，那么用户可以显示地声明`android:installLocation`属性，并将属性值设为`"internalOnly"`。尽管这样不会改变系统的默认属性，但是这样非常明确地声明了应用程序只能安装在内存中，给了用户和其他开发者一个明确的提示。

应该装在外部存储设备的程序

简单地说，任何没有使用上述特点的应用程序，安装在外部存储设备时都是安全的。大型游戏通常就是应该装在外部存储设备的程序类型，因为游戏不使用时通常不需要提供额外的服务。当外部存储设备不可用时，游戏进程结束，当存储设备再次可用时，不应该有明显的影响，用户重启游戏即可（假设游戏在正常的活动周期内恰当地存储了游戏状态）。

如果apk文件需要好几兆空间，用户就需要考虑是否将应用程序安装在外部存储设备上，为内存保留更多的空间。

来自 "[index.php?title=App_Install_Location&oldid=6167](#)"



Administration

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：

[Administration](#)

翻译者： Cecho 翻译时间： 2012.8.1

管理

如果你是一个企业的管理员，你可以充分利用众多的API和系统的特性来管理Android设备并对他们的访问进行控制。

了解更多： [Device Policies](#)

博客文章：

[Unifying Key Store Access in ICS](#)

Android 4.0(ICS)附带了大量的改进,使人们能更容易地把他们的个人Android设备工作。在这篇文章中,我们将看看密钥存储的功能。

来自“[index.php?title=Administration&oldid=7663](#)”

1个分类: [Android Dev Guide](#)



Device Policies

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：

[Device Administration](#)

翻译者： Cecho 翻译时间： 2012.8.1

目录

- [1 设备管理](#)
- [2 设备管理API概述](#)
 - [2.1 它是如何工作的?](#)
 - [2.2 策略](#)
 - [2.3 其他特性](#)
- [3 事例程序](#)
- [4 开发一个设备管理程序](#)
 - [4.1 创建Manifest文件](#)
 - [4.2 完成编码](#)
 - [4.2.1 完成DeviceAdminReceiver的子类](#)
 - [4.2.2 启用程序](#)
 - [4.2.3 管理策略](#)

设备管理

Android 2.2 通过提供Android设备管理API来向企业级应用提供支持。设备管理API在操作系统级别提供设备管理特性。这些API使你可以创建那些在企业环境中对安全敏感的应用，在企业环境中IT专家要对员工的设备进行全面的控制（译者：信息安全很重要啊，私自泄漏要负法律责任的）。例如，内置的Android邮件应用增加了一些新的API来提高对Exchange的支持。通过邮件程序，Exchange管理员可以加强设备间的密码安全策略--包括字母数字密码或者数字PIN。管理员还可以远程擦除（也就是恢复出厂设置）

丢失或者被偷的手持终端。**Exchange**用户还可以同步他们的电邮和日历数据。

这份文档是为了那些要在**Android**设备上提供企业解决方案的开发者准备的。这里讨论了设备管理**API**为了加强员工们使用的**Android**设备的安全性而提供的众多特性。

设备管理**API**概述

可能会用到设备管理**API**的应用程序：

- 电子邮件客户端。
- 远程删除数据的安全应用程序。
- 设备管理服务和应用程序。

它是如何工作的？

你使用这些设备管理**API**来编写安装在用户设备上的设备管理程序。设备管理程序加强了期望的策略。下面是它如何起作用的：

- 一个系统管理员编写一个增强远端或者本地设备的安全策略。这些策略可以被硬编码到应用里面或者动态地从第三方服务器获取。
- 应用要被安装在用户设备上的。但是**Android**现在还没有提供自动化供应解决方案。系统管理员可以通过下面这些途径向用户发行应用：
 - Google Play
 - 其他应用商店
 - 其他方式，比如邮件或者网站
- 系统会提示用户开启设备管理程序。这一切，如何以及何时发生取决于应用的设计。
- 一旦用户启用了设备管理程序，就要受制于它的策略。遵守这些策略会带来好处，比如访问敏感的系统和数据。

如果用户不启用管理程序，它还在设备上，但是出于非激活状态。用户就不会受制于它的策略，当然也就不会得倒好处--例如，用户无法同步数据。

如果用户没有遵循安全策略(例如，设置了不符合要求的密码)，应用就会决定如何处理出现的情况。然而，一般也就是让用户同步数据。

如果设备尝试连接某个服务器，而这个服务器的安全策略不被设备管理**API**支持，连接就不会建立。设备管理**API**目前不允许部分匹配。换句话说，如果一个设备(比如，遗留下来的设备)没有支持全部的策略，这个设备就无法连接。

如果一个设备包含多个开启的管理程序，最强的策略就会被执行。应为没有办法确定一

个特定的管理程序。要卸载设备上的管理程序，首先需要用户以管理员的身份取消注册。

策略

在企业环境中，通常是雇员遵守一套严格的设备管理策略。Android的设备管理API支持Table 1 中列举的策略。记住，设备管理API目前只支持锁屏密码。

表格1.设备管理API支持的策略

策略	说明
启用密码	要求设备询问PIN或者密码
最小密码长度	设置密码的最小字符数。例如，你可以要求PIN或者密码最短要有6个字符
数字和字符混合模式	要求密码是由字符和数字组合而成的。密码可以包含符号字符。
启用复杂密码模式	要求密码必须包含至少一个字母、数字、特殊字符。这个特性是由Android 3.0 引入的。
最少字母数的限制	对所有管理员或者某个特定用户要求密码中最少要有多少个字母。这个特性是由Android 3.0 引入的。
最少小写字母数的限制	对所有管理员或者某个特定用户要求密码中最少要有多少个小写字母。这个特性是由Android 3.0 引入的。

最少非字母字符数的限制	对所有管理员或者某个特定用户要求密码中最少要有多少个非字母字符。这个特性是由Android 3.0 引入的。
最少数字字符数的限制	对所有管理员或者某个特定用户要求密码中最少要有多少个数字字符。这个特性是由Android 3.0 引入的。
最少符号字符数的限制	对所有管理员或者某个特定用户要求密码中最少要有多少个符号字符。这个特性是由Android 3.0 引入的。
最少大写字母字符数的限制	对所有管理员或者某个特定用户要求密码中最少要有多少个大写字母字符。这个特性是由Android 3.0 引入的。
密码过期时间设置	密码何时失效。当管理员设置过期时间后，会以毫秒递增直到设置的时间。这个特性是由Android 3.0 引入的。
密码历史记录	这项策略可以避免用户使用以前用过的n个密码。它通常还是和 <u>setPasswordExpirationTimeout()</u> 一起使用的。这个特性是由Android 3.0 引入的。
允许输入错误密码的次数	指明在设备擦除自身数据之前，用户可以输入错误密码的次数。设备管理API还允许管理员远程地使设备恢复出厂设置。这在设备丢失的情况下保护了设备的数据安全。
锁屏时	设置了用户没有触摸屏幕或者没有按键，设备锁屏要等待的时

间	间。用户要重新使用设备就要输入PIN或者密码。它的值可以设置为1-60分钟。
加密存储	在设备支持的情况下，指定要加密的存储位置。这个特性是由Android 3.0 引入的。
禁用摄像头	说明摄像头要被禁用。要注意，这不是永久的禁用。摄像头可以在任何情况、时间下启用/禁用。这个特性是由Android 4.0 引入的。

其他特性

除了支持上面列出来的策略外，设备管理API还允许你做下面的事情：

- 提示用户设置新密码
- 立即锁屏
- 擦除数据(也就是，恢复出厂设置)

事例程序

本篇wiki所使用的例子是基于SDK示例程序中的[设备管理API](#)示例程序创建的。想要获取和安装SDK示例，请看这里[获取SDK示例](#)。这还有[完整源代码](#)。

这个程序向你提供了展示设备管理特性的例子。它向用户展现了一个用户界面并允许用户开启设备管理程序。一旦用户启动程序，他们就可以用界面上的按钮来做下面的事情：

- 设置密码的质量
- 说明对用户密码的要求。比如，密码包含的最小长度、密码中最少要有几个数字字符等等。
- 设置密码。如果输入的密码不符合特定的策略，系统就会提示出错。
- 设置数据被擦除(恢复出厂设置)前，允许的输入错误的次数。
- 设置密码过期时间。
- 设置记录的密码数量。这可以避免用户使用旧的密码。
- 在设备支持的情况下，可以设定加密存储的区域。
- 设置设备上锁的时间。
- 使设备立即上锁。
- 擦除设备数据(恢复出厂设置)

- 禁用摄像头



图片1.示例程序截图

开发一个设备管理程序

系统管理员可以使用设备管理API创建一个设备管理程序本地\远程地对设备的安全策略进行加强。这一小节总结了必要的步奏。

创建Manifest文件

要使用设备管理API，程序的Manifest文件要包含下面的内容：

- [DeviceAdminReceiver](#) 的子类包含以下内容.
 - [BIND_DEVICE_ADMIN](#) 权限
 - 响应 [ACTION_DEVICE_ADMIN_ENABLED](#) intent 对象的能力，在Manifest文件中由 intent filter 完成。
- 在元数据中使用安全策略的声明。

这是一段设备管理程序Manifest文件的摘录：

```
<activity android:name=".app.DeviceAdminSample"
          android:label="@string/activity_sample_device_admin">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.SAMPLE_CODE" />
    </intent-filter>
</activity>
<receiver android:name=".app.DeviceAdminSample$DeviceAdminSampleReceiver"
          android:label="@string/sample_device_admin"
          android:description="@string/sample_device_admin_description"
          android:permission="android.permission.BIND_DEVICE_ADMIN">
    <meta-data android:name="android.app.device_admin"
              android:resource="@xml/device_admin_sample" />
    <intent-filter>
        <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
    </intent-filter>
</receiver>
```

注意：

- 下面属性所应用的字符串资源在项目的[ApiDemos/res/values/strings.xml](#)中。关于资源的更多信息，看这里 [Application Resources](#)
 - [android:label="@string/activity_sample_device_admin"](#) 为activity 引用了用户

可读标签。

- `android:label="@string/sample_device_admin"`为permission引用了用户可读标签。
- `android:description="@string/sample_device_admin_description"`为permission引用了用户可读标签。描述信息一般都比标签的内容长得多。
- `android:permission="android.permission.BIND_DEVICE_ADMIN"`是一个[DeviceAdminReceiver](#)的子类必须要获得的权限，来确保只有系统可以和接受者交互(没有程序可以保证这个权限)。这就可以阻止其他应用滥用你的系统管理程序。
- `android.app.action.DEVICE_ADMIN_ENABLED`是一个[DeviceAdminReceiver](#)的子类必须处理的动作，如果它要被允许管理设备。在程序启动时，它就已经被设置到接收者了。你的代码要重载[onEnabled\(\)](#)方法。同时还要具有[BIND_DEVICE_ADMIN](#)权限，以免被其他程序滥用。
- 一旦用户启动了程序，接受者也就获得了响应系统广播事件的权限。当合适的事件发生，程序就会应用对应的策略。例如，如果用户在设置信密码时，输入的内容不符合安全策略，程序会提示用户重新选择一个符合要求的密码。
- `android:resource="@xml/device_admin_sample"`声明了在元数据中使用的安全策略，元数据为设备管理原提供了特定的信息。元数据由[DeviceAdminInfo](#)类来解析。下面是device_admin_sample.xml的内容：

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-policies>
    <limit-password />
    <watch-login />
    <reset-password />
    <force-lock />
    <wipe-data />
    <expire-password />
    <encrypted-storage />
    <disable-camera />
  </uses-policies>
</device-admin>
```

在你设计你的设备管理程序时，并不需要包括所有的安全策略，按需定制即可。

关于Manifest文件的更多信息，请到[Android Developers Guide](#)。

完成编码

设备管理API包括下面这些类：

[DeviceAdminReceiver](#)

完成设备管理组件的基类。这个类提供了一个解释系统发送的原始Intent动作的方便途径。你的设备管理程序，必须包含一个它的子类。

[DevicePolicyManager](#)

这个类负责管理在设备上执行的安全策略。它的大多数客户端要发布一个已经被当前用户启用的[DeviceAdminReceiver](#)。[DevicePolicyManager](#)为最少一个[DeviceAdminReceiver](#)实例管理安全策略。

[DeviceAdminInfo](#)

这个类是用来为系统管理组件指定元数据的。

完成[DeviceAdminReceiver](#)的子类

要创建一个设备管理程序，就要实现一个继承[DeviceAdminReceiver](#)的类。[DeviceAdminReceiver](#)包含了一系列的回调函数，这些回调函数会在具体的事件发生时被调用。

在例子程序里，我们只是简单地显示了一个[Toast](#)，来做为对相应事件的应答。例如：

```
public class DeviceAdminSample extends DeviceAdminReceiver {
    void showToast(Context context, String msg) {
        String status = context.getString(R.string.admin_receiver_status, msg);
        Toast.makeText(context, status, Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onEnabled(Context context, Intent intent) {
        showToast(context,
            context.getString(R.string.admin_receiver_status_enabled));
    }
    @Override
    public CharSequence onDisableRequested(Context context, Intent intent) {
        return
            context.getString(R.string.admin_receiver_status_disable_warning);
    }
    @Override
    public void onDisabled(Context context, Intent intent) {
        showToast(context,
            context.getString(R.string.admin_receiver_status_disabled));
    }
    @Override
    public void onPasswordChanged(Context context, Intent intent) {
        showToast(context,
            context.getString(R.string.admin_receiver_status_pw_changed));
    }
}...
```

启用程序

用户启用程序，是设备管理程序要处理最重要的事件之一。用户必须明确启用设备管理程序才能使安全策略在设备上执行。如果用户选择不启用的话，那么安全策略就不会被

执行，用户也就无法利用我们的设备管理程序。

应用程序被启用的过程是，用户的动作出发了[ACTION_ADD_DEVICE_ADMIN Intent](#)。在这个程序里，它发生在用户点击了 **Enable Admin** 选择框。

当用户点击了**Enable Admin** 选择框，设备就会提示用户已经启用了设备管理程序，如图2，所示：



图2. 实例：启用应用程序

下面就是当用户点击了 **Enable Admin** 选择框要执行的代码。效果是触发了[onPreferenceChange\(\)](#)回调函数。当用户改变Preference的值时，就会调用这个回调函数。如果用户启用程序，界面就会提示用户正在启用程序，就像图2.否则就是禁止程序：

```

@Override
    public boolean onPreferenceChange(Preference preference, Object
newValue) {
        if (super.onPreferenceChange(preference, newValue)) {
            return true;
        }
        boolean value = (Boolean) newValue;
        if (preference == mEnableCheckbox) {
            if (value != mAdminActive) {
                if (value) {
                    // Launch the activity to have the user enable our
admin.
                    Intent intent = new
Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
                    intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN,
mDeviceAdminSample);

                    intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,
mActivity.getString(R.string.add_admin_extra_app_text));
                    startActivityForResult(intent,
REQUEST_CODE_ENABLE_ADMIN);
                    // return false - don't update checkbox until we're
really active
                    return false;
                } else {
                    mDPM.removeActiveAdmin(mDeviceAdminSample);
                    enableDeviceCapabilitiesArea(false);
                    mAdminActive = false;
                }
            }
        } else if (preference == mDisableCameraCheckbox) {
            mDPM.setCameraDisabled(mDeviceAdminSample, value);
            ...
        }
    }
    return true;
}

```

intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, mDeviceAdminSample)说明了mDeviceAdminSample是目标策略([DeviceAdminReceiver](#)是一个组件)。这些代码会调用图2的界面，让用户选择是否添加系统管理员。

当程序要执行一个操作，它要确定管理程序是否已经被启用了。实现的方法就是使用[DevicePolicyManager](#)的[isAdminActive\(\)](#)方法。要注意的是，[DevicePolicyManager](#)的[isAdminActive\(\)](#)方法需要一个[DeviceAdminReceiver](#)类型的参数。

```
DevicePolicyManager mDPM;
...
private boolean isActiveAdmin() {
    return mDPM.isAdminActive(mDeviceAdminSample);
}
```

管理策略

[DevicePolicyManager](#)是一个用来管理设备上所执行的策略的公共类。[DevicePolicyManager](#)为一个或多个[DeviceAdminReceiver](#)类的实例管理策略。

你可以这样获得[DevicePolicyManager](#)的一个实例：

```
DevicePolicyManager mDPM =
    (DevicePolicyManager) getSystemService(Context.DEVICE_POLICY_SERVICE);
```

这一节描述了怎样用[DevicePolicyManager](#)执行管理任务：

- 密码策略
- 设备锁定策略
- 指定数据擦除

设置密码策略

[DevicePolicyManager](#)包含了一些用来设置和执行设备密码策略的API。在设备管理API中，密码只是用来解锁屏幕。本小节描述了密码相关的普通任务。

为设备设置密码

这段代码显示一个界面提示用户设置密码：

```
Intent intent = new Intent(DevicePolicyManager.ACTION_SET_NEW_PASSWORD);
startActivity(intent);
```

设置密码的强度

密码的强度可以由[DevicePolicyManager](#) 中的常量来设置：

[PASSWORD_QUALITY_ALPHABETIC](#)

用户输入的密码必须要有字母（或者其他字符）。

[PASSWORD_QUALITY_ALPHANUMERIC](#)

用户输入的密码必须要有字母和数字。

[PASSWORD_QUALITY_NUMERIC](#)

用户输入的密码必须要有数字

[PASSWORD_QUALITY_COMPLEX](#)

用户输入的密码必须要有至少一个数字、字母、特殊字符。

[PASSWORD_QUALITY_SOMETHING](#)

由设计人员决定的。

[PASSWORD_QUALITY_UNSPECIFIED](#)

对密码没有要求。

例如，你应该这样设置密码策略来要求一个数字的密码：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
mDPM.setPasswordQuality(mDeviceAdminSample,
DevicePolicyManager.PASSWORD_QUALITY_ALPHANUMERIC);
```

设置对密码内容的要求

从 Android 3.0开始，[DevicePolicyManager](#)就提供了一些能很好调节密码内容的方法。例如，你可以要求密码必须有n个大写字母。下面这些就是提供功能的方法：

- [setPasswordMinimumLetters\(\)](#)
- [setPasswordMinimumLowerCase\(\)](#)
- [setPasswordMinimumUpperCase\(\)](#)
- [setPasswordMinimumNonLetter\(\)](#)
- [setPasswordMinimumNumeric\(\)](#)
- [setPasswordMinimumSymbols\(\)](#)

例如下面的片段就设置密码要有最少两个大写字母：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwMinUppercase = 2;
...
mDPM.setPasswordMinimumUpperCase(mDeviceAdminSample, pwMinUppercase);
```

设置密码最小长度

你可以指定密码的最小长度。例如：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwLength;
...
mDPM.setPasswordMinimumLength(mDeviceAdminSample, pwLength);
```

设置密码最多错误输入次数

你可以设置允许密码输入错误的最大次数，超过这个次数设备就要擦除数据（恢复出厂设置）。例如：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int maxFailedPw;
...
mDPM.setMaximumFailedPasswordsForWipe(mDeviceAdminSample, maxFailedPw);
```

设置密码过期时间

从 Android 3.0开始，你可以使用[setPasswordExpirationTimeout\(\)](#)方法设置密码何时失效，当设置完成系统会以毫秒为单位倒计时。例如：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
long pwExpiration;
...
mDPM.setPasswordExpirationTimeout(mDeviceAdminSample, pwExpiration);
```

对密码的历史记录进行限制

从 Android 3.0开始，你可以使用[setPasswordHistoryLength\(\)](#)限制用户使用的密码要多久不能重复，这个方法有一个`length`参数，用来设置要记录密码的个数。当这项策略激活了，用户就不能使用所设定范围内的旧密码当做新的密码来使用。这就防止了用户一直使用一个密码。这个策略通常与[setPasswordExpirationTimeout\(\)](#)一起使用，来使用户过一段时间就换一个新的密码。

例如，下面的代码可以防止用户使用最近使用的五个密码：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwHistoryLength = 5;
...
mDPM.setPasswordHistoryLength(mDeviceAdminSample, pwHistoryLength);
```

设置锁屏

你可以设置用户多长时间没有动作，就把设备锁住。例如：

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
long timeMs = 1000L*Long.parseLong(mTimeout.getText().toString());
mDPM.setMaximumTimeToLock(mDeviceAdminSample, timeMs);
```

你还可以使设备立即锁住：

```
DevicePolicyManager mDPM;
mDPM.lockNow();
```

数据擦除

你可以使用[DevicePolicyManager](#)的[wipeData\(\)](#)方法使设备恢复出厂设置。这在设备被偷或者丢失的情况下非常有用。通常，在做出要擦除设备的决定时，都是达到了某一条件。例如，你可以使用[setMaximumFailedPasswordsForWipe\(\)](#)来是设备在用户输入密码错误固定次数之后擦除设备数据。

你要这样做：

```
DevicePolicyManager mDPM;
mDPM.wipeData(0);
```

[wipeData\(\)](#)方法的参数是一个选项的位掩码，这里暂时必须为0；

禁用摄像头

从 Android 4.0开始，你可以禁用摄像头。注意这不是永久的禁用。摄像头可以动态的禁用/启用在不同的上下文、时间等待。

你使用[setCameraDisabled\(\)](#)来设置摄像头是否被禁用。例如，下面的代码就根据选择框的状态来决定摄像头是否被禁用：

```
private CheckBoxPreference mDisableCameraCheckbox;
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
mDPM.setCameraDisabled(mDeviceAdminSample, mDisableCameraCheckbox.isChecked());
```

加密存储

从 Android 3.0开始，你可以使用 `setStorageEncryption()` 方法，来设置加密存储，如果设备支持的话。

例如：

```
DevicePolicyManager mDPM;  
ComponentName mDeviceAdminSample;  
...  
mDPM.setStorageEncryption(mDeviceAdminSample, true);
```

请阅读[DeviceAdminSample](#)来获取完整的加密存储的例子

来自“[index.php?title=Device_Policies&oldid=7654](#)”

1个分类：

- [Android Dev Guide](#)

Web Apps

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： CuGBabyBeaR

原文链接：<http://developer.android.com/guide/webapps/index.html>

网络应用

Android一直致力于链接和提供优秀的的网络浏览体验，因此使用网络技术创建您的应用是一个很好的机会。您不仅可以在网络上建立一个应用，对您对于Android不同屏幕尺寸和密度的设计进行优化，也可以通过使用WebView，将基于网络的内容嵌入您的Android应用。

[概述 >](#)

来自“[index.php?title=Web_Apps&oldid=8673](#)”



Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： GloriousOnion

原文地址：<http://developer.android.com/guide/webapps/overview.html>

Web App概述

实质上，Android有两种不同的方法发布应用：一是通过客户端应用（通过Android SDK开发并以.apk文件安装到用户设备上），二是通过Web应用（其通过互联网标准进行开发并使用浏览器进行访问，其特点是不会向用户设备安装任何程序）。

图1说明了如何从web浏览器或者安卓应用程序来访问网页。但是不要仅仅作为查看网站的手段来开发安卓应用程序。相反，安卓应用程序嵌入的网页应该针对环境进行设计。甚至可以定义安卓应用程序和网页之间的接口，这一接口允许JavaScript调用安卓应用程序API——向基于web的应用程序提供安卓API。

要为安卓供电设备开发网页，请参阅下列文档：

- [Web应用的目标界面](#)

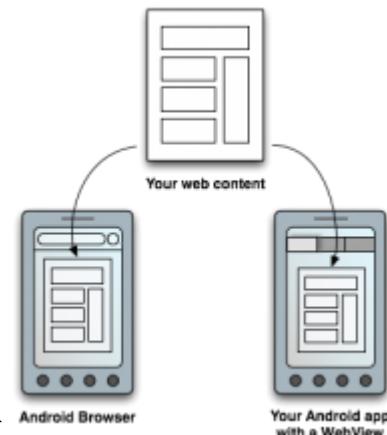


图1. 可以通过传统浏览器或是布局中有WebView的Android应用将你的网络内容提供用户使用。

介绍如何恰当地定义Android设备上Web应用尺寸并支持多种分辨率。如果你正在一个搭建至少对Android设备适用（应为发布到网上的所有东西做出假设）的Web应用，本文档会有不少帮助，而对以移动

设备为目标及正在使用[WebView](#)的读者，本文档则大有裨益。

- [**在WebView中构建Web App**](#)

介绍如何使用[WebView](#)将网页嵌入到Android应用中，并将JavaScript与Android API绑定。

- [**调试Web App**](#)

介绍如何通过JavaScript输出API调试Web App

- [**Web应用程序最佳实践**](#)

为了提供有效web应用程序，您应该遵循一系列做法。

来自“[index.php?title=Overview&oldid=13824](#)”



Targeting Screens from Web Apps

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

译： CuGBabyBeaR

完成时间： 8月15日

原文链接：<http://developer.android.com/guide/webapps/targeting.html>

目录

[[隐藏](#)]

[1 Web应用程序的屏幕适配](#)

- [1.1 快速预览](#)
- [1.2 在此篇文档中](#)

[2 使用Viewport Metadata](#)

- [2.1 预定义viewport尺寸](#)
- [2.2 自动调整尺寸](#)
- [2.3 预定义viewport缩放](#)
- [2.4 预定义viewport匹配密度](#)

[3 用CSS适配设备密度](#)

[4 用JavaScript适配设备密度](#)

Web应用程序的屏幕适配

如果您正在为Android或者移动设备开

发或者重新设计一个web应用，您应该仔细研究您的网页是如何在不同种类的屏幕上显示的。因为Android是可以存在于使用不同类型屏幕的设备上的，您应该考虑一些影响您的网页在Android设备上显示方式的因素。

注：这个本文档中描述的方法，在Android2.0或更高版本下，被Android浏览器(由缺省Android平台所提供的)和 WebView(用来显示网页的框架视图部件 (widget))所支持。运行在Android上的第三方浏览器可能不支持用这些方法来控制viewport的尺寸和屏幕的密度。

当您为Android设备指定您的网页时，有两个您应该注意的基本因素：

viewport的尺寸和网页的缩放

当Android浏览器加载一个网页

时，默认行为是以"概览模式"加载这个页面，"概览模式"是指提供缩小至这个页面的远景的视图。您可以~~以~~通过定义这个viewport的默认尺寸或者初始缩放值，为您的网页重写这个行为。如果可用的话，您同样可以控制用户能够将您的网页放大或者缩小多少。用户同样可以在浏览器的设置里关闭概览模式，所以您绝对不应该假设您的页面会在概览模式下加载。您应该为您的页面设置合适的自定义viewport的尺寸和/或网页的缩放。

但是，当您的页面在WebView中显示时，页面会在无缩放（而不是“概览模式”）下加载。这就是说，会以页面的默认大小显示它，而不缩小它。（这也是当用户关闭了概览模式时，页面的显示方式。）

设备的屏幕密度

快速预览

- 您可以通过viewport metadata, CSS, 和JavaScript为不同的屏幕匹配您的web页面。
- 本文所述的技术在Android 2.0或更高版本下，且显示在默认Android浏览器和WebView中的网页上工作。

在此篇文档中

[使用Viewport Metadata](#)

[预定义viewport尺寸](#)

[预定义viewport缩放](#)

[预定义viewport匹配密度](#)

[用CSS适配设备密度](#)

[用JavaScript适配设备密度](#)

Android设备的屏幕密度（每英寸像素数）影响着显示哪个网页及其显示尺寸的决定。（有三个屏幕密度类别：低、中和高。）Android浏览器和WebView会通过缩放网页补偿屏幕密度的变化，使得所有的设备在显示此网页时看上去和中等密度下显示的大小相同。如果图像是您页面设计中的一个重要元素，您需要特别注意在不同密度下的缩放，因为图片缩放可能产生出您不希望的效果（模糊和像素化）。

为了在所有的屏幕密度下提供最好的可见表示，您需要通过提供有关您页面的目标屏幕密度**viewport metadata**，以及提供不同密度下选择的图片，以控制如何缩放。您同样可以通过使用CSS或者JavaScript来控制它。

本文的其余部分介绍了如何使用这些效果并提供一个在多种类的屏幕上表现良好的设计。

使用**Viewport Metadata**

viewport是指您的页面上正在显示的区域。尽管**viewport**的可见区域和屏幕尺寸是相匹配的，但**viewport**有他自己的尺寸以决定网页上有多少像素是可见的。这就是说，网页上的可见像素数在他溢出屏幕之前就已经通过**viewport**的尺寸，而不是设备屏幕的尺寸，定义了。例如，即使一个设备可能有480像素宽，但**viewport**可以拥有一个800像素的宽，因此一个设计为宽度800像素的网页在这个屏幕上是完全可见的。

您可以使用HTML `<meta>`标签（`<meta>`标签必须放置在`<head>`中）的“**viewport**”属性为您的网页定义 **viewport**的属性。您可以在`<meta>`标签的**content**属性中定义多个**viewport**属性。例如，您可以定义**viewport**的宽和高，页面的初始缩放，以及指定屏幕密度。在**content**属性中的每个**viewport**属性必须要用逗号隔开。

例如，下面HTML文档的片段指定了**viewport**的必须和设备同宽，并且不可缩放

```
<head>
  <title>Example</title>
  <meta name="viewport" content="width=device-width, user-scalable=no" />
```

上文只是一个只有两个viewport属性例子，下文描述了所有支持的viewport属性及其允许的值类型。

```
<meta name="viewport"
content="
    height = [pixel_value | device-height] ,
    width = [pixel_value | device-width ] ,
    initial-scale = float_value ,
    minimum-scale = float_value ,
    maximum-scale = float_value ,
    user-scalable = [yes | no] ,
    target-densitydpi = [dpi_value | device-dpi |
        high-dpi | medium-dpi | low-dpi]
" />
```

下面的章节讨论如何使用这些viewport属性，以及他们可接受的值。

预定义viewport尺寸

Viewport的height和width属性允许你定义viewport的尺寸(在网页显示在屏幕上之前可用的像素数)。

上面的介绍中提到，Android浏览器默认以“概览模式”加载页面（除非用户禁用），其中设置最低的viewport的宽度为800像素。因此，如果您的网页指定其大小是宽320像素，那么你的网页看起来将小于可见屏幕（即便实际上屏幕只有320像素宽，因为viewport模拟了一个800像素宽的绘制区域），如下图1所示。为了避免这种效果，您应该明确定义viewport宽度与您设计的网页相匹配。

例如，如果您网页的设计恰好320像素宽，那么您可能要指定viewport宽度：



图1：一个有320像素宽图片的网页,没有viewport metadata设置(默认“概览模式”可用,viewport800像素宽),在Android浏览器中显示的情况。

```
<meta name="viewport" content="width=320" />
```

在本例中，因为网页和viewport宽度相同，您的网页和屏幕宽度完全相符。

注：大于10,000的宽度值会被忽略，小于等于320的宽度值等同于设备宽度（下面讨论）。大于10,000或者小于200的高度值同样被忽略。

为了说明这个属性如何影响您德尔网页大小，图2显示了一个网页，包含一个320像素宽的图像，但viewport的宽度设置为400。

注：如果您设置了viewport的宽度以匹配您的网页宽度，而设备的屏幕宽度并没有符合这些尺寸，此时无论设备是高或者低密度的，网页依然会适合屏幕，因为默认情况下，Android浏览器和WebView缩放一个网页与中等密度的屏幕相匹配（如您图2中所见，比较在高dpi和中dpi设备上的显示情况）。屏幕密度会在[预定义viewport匹配密度](#)中讨论更多。



图2：网页在viewport width=400并且概览模式可用的情况下。



图3：网页在viewport width=device-width或initial-scale=1.0的情况下。

自动调整尺寸

作为精确指定viewport像素尺寸的替代，通过定义viewport属

性**height**和**width**分别为**device-height**和**device-width**，您可以设置**viewport**的大小总是匹配设备的尺寸。当您开发一个具有可变宽度（而不是固定宽度）的web应用时，您想让他如同固定宽度时的显示一样（如同网页宽度为每个屏幕设置过一样，完美的契合每一个屏幕），这将是很好用的。例如：

```
<meta name="viewport" content="width=device-width" />
```

图3是**viewport width**匹配当前屏幕时的结果。重点是要注意到，图中的结果是，当当前设备不匹配目标密度（如果不特别指定将会是中等密度）时，为了匹配屏幕而缩放过。因此，图3中显示在高密度设备上的图片放大过，以便与中等密度的屏幕宽度相匹配。

注：如果您想使用**device-width**和**device-height**来匹配每一个设备的物理屏幕像素数，而不是缩放您的网页以匹配目标密度，您必须包括**device-dpi**值为**target-densitydpi**的属性。这将在[预定义viewport密度](#)一节详细讨论。另外，简单的使用**device-height**和**device-width**来定义**viewport**尺寸使得您的网页与每个设备屏幕相适合，但会缩放您的图像使之符合不同的密度。

预定义**viewport**缩放

viewport的缩放值定义了页面所允许的缩放范围。**viewport**属性允许您通过以下方式指定您的页面缩放：

initial-scale

页面的初始缩放。这个值是一个指示您的页面相对于屏幕大小倍数的**float**值。例如，如果您设置初始缩放为“1.0”那么页面将根据[目标密度](#)1比1的显示。如果设置为“2.0”那么页面将放大2倍。

为了将网页与**viewport**尺寸匹配，默认初始缩放是计算过的。因为默认**viewport**是800像素宽，如果设备屏幕判断为小于800像素宽，初始缩放将以小于1.0的某个值为默认值，以便在屏幕上匹配一个800像素宽的页面。

minimum-scale

允许的最小缩放。这个值是一个指示您的页面相对于屏幕大小最小倍数的**float**值。例如，如果您设置为1.0，那么因为最小大小和目标密度是1:1的，页面就不能缩小。

maximum-scale

允许的最大缩放。这个值是一个指示您的页面相对于屏幕大小最大倍数的**float**值。例如您设置此值为2.0，那么您就不能放大超过2倍。

user-scalable

是否允许用户缩放此页面。设置为yes允许缩放，no为不允许缩放。默认值是yes。如果您设置此值为no，那么minimum-scale和maximum-scale将会被忽略，因为缩放不可用。

所以的缩放值必须在0.01到10之间。

例如：

```
<meta name="viewport" content="initial-scale=1.0" />
```

这个metadata设置初始缩放为相对于viewport目标密度的原始大小。

预定义**viewport**匹配密度

设备屏幕的密度基于屏幕决定，由每英寸的像素数 (dpi) 定义。有三种Android支持的密度：高 (hdpi)、中 (mdpi) 和低 (ldpi)。低密度的屏幕每英寸有更少的可用像素，而高密度的屏幕每英寸有更多的可用像素（相对于中等密度的屏幕来说）。Android浏览器和WebView默认匹配中等密度屏幕。

因为默认密度是中，当用户使用一个有低或者高密度屏幕的设备时，Android浏览器和WebView会缩放网页，使网页的大小能够与中等密度大小的屏幕上显示的可视大小相一致。更具体的说，Android浏览器和WebView允许在高

密度屏幕上对网页进行大约1.5x的缩放
 (因为高密度的屏幕像素更小)，在低密度屏幕上对网页进行大约0.75x的缩放
 (因为低密度的屏幕像素更大)。

由于这个默认缩放的存在，
 图1、2和3在高和中等密度的设备上以相同的物理大小显示示例网页 (高密度设备用一个默认的缩放参数显示网页，使之为网页的实际像素分辨率的1.5倍大，以适合于目标密度)。这可能给您的图片引入一些不良的效果。例如，尽管一个图片在中和高密度设备上以相同的大小出现，但高密度设备上显示的图片会出现模糊，因为这个图片是以320像素宽设计，却以480像素宽显示的。

您可以通过使用viewport属性target-densitydpi来为您的网页改变目标密度。它允许以下的值：

- device-dpi - 使用设备本地的dpi作为目标dpi。默认缩放将不会起作用。
- high-dpi - 使用hdpi作为目标dpi。中和低密度的屏幕将会适当缩小。.
- medium-dpi - 使用mdpi作为目标dpi。高和低密度的屏幕将会分别放大和缩小。这是默认的密度值。
- low-dpi - 使用ldpi作为目标dpi。中和高密度的屏幕将会适当放大。
- <value> - 指定一个dpi值作为目标dpi。该值必须在70-400之间。

例如，您可以通过设置viewport的target-densitydpi属性，来阻止Android浏览器和WebView为不同的分辨率缩放网页。而这个页面会以当匹配前屏幕的密度的大小显示。在本例中，您同样应该定义viewport的宽以匹配设备宽度，这样您的页面才会自然适合屏幕大小。例如：



图4:网页在viewport width=device-width且target-densitydpi=device-dpi的情况下。

```
<meta name="viewport" content="target-densitydpi=device-dpi,
width=device-width" />
```

图4显示了使用了这些viewport设置的网页 - 显示的页面比较小，是因为高密度设备的像素比中密度设备的小，也就是说没有缩放而320像素宽的图片以320像素宽显示在每个屏幕上。（到此讲的是您应该怎样定义您的viewport，如果您想根据屏幕密度自定义页面并提供不同的图片资源，请使用CSS或 JavaScript。）

用CSS适配设备密度

Android浏览器和WebView支持CSS媒介类型，您可以使用-webkit-device-pixel-ratio的CSS媒介类型为特定的屏幕密度创建一个样式。您所允许使用的值包括"0.75", "1", 或 "1.5"，分别表示低、中和高密度屏幕。

For example, you can create separate stylesheets for each density: 例如，您可以分别为每个密度分别建立样式表。

```
<link rel="stylesheet" media="screen and (-webkit-device-pixel-ratio: 1.5)" href="hdpi.css" />
<link rel="stylesheet" media="screen and (-webkit-device-pixel-ratio: 1.0)" href="mdpi.css" />
<link rel="stylesheet" media="screen and (-webkit-device-pixel-ratio: 0.75)" href="ldpi.css" />
```

或者在一个样式表中指定不同的样式：

```
#header {
    background:url(medium-density-
image.png);
}

@media screen and (-webkit-device-
pixel-ratio: 1.5) {
    /* CSS for high-density screens */
    #header {
        background:url(high-density-
image.png);
    }
}

@media screen and (-webkit-device-
pixel-ratio: 0.75) {
```



```
/* CSS for low-density screens
*/
#header {
    background:url(low-density-
image.png);
}
```

图5:网页具有用-webkit-device-pixel-ratio媒介类型来匹配特定的屏幕密度时的CSS。注意css使hdpi设备应用了一个不同的图片。

注：默认样式#header适用于中密度设备设计的图片，是为了支持使用低于2.0版本安卓系统的设备。低于2.0的Android系统不支持 -webkit-device-pixel-ratio 媒介类型。

The types of styles you might want to adjust based on the screen density depend on how you've defined your viewport properties. To provide fully customized styles that tailor your web page for each of the supported densities, you should set your viewport properties so the viewport width and density match the device. That is: 您可能想基于你在viewport属性值中定义的屏幕密度，来调整样式类型。为了提供完全自定义的样式，使您的页面是为每个支持的密度定制的，您应该设置您的viewport属性，使viewport宽和密度将会和设备相匹配。如下：

```
<meta name="viewport" content="target-densitydpi=device-dpi,
width=device-width" />
```

通过这种方式，Android浏览器和WebView不对您的网页进行缩放并且viewport宽度完全匹配与屏幕宽度。这些viewport属性创建如图4所示的结果。但是通过添加一些使用-webkit-device-pixel-ratio媒介类型的自定义CSS，您可以使用不同的样式。例如，图5显示了一个网页，使用了如上viewport属性并添加了一些允许在高密度屏幕上使用高清晰度图片的CSS。

用JavaScript适配设备密度

Android浏览器和WebView支持允许您请求当前设备的屏幕密度的DOM属性

- DOM属性 `window.devicePixelRatio`。这个属性的值 指定当前设备使用的比例系数。例如，如果`window.devicePixelRatio`的值是"1.0"，那么这个设备被认为是中等密度并且默认情况下 不缩放；如果这个值是"1.5"，那么这个设备被认为是高密度并且默认情况下放大1.5x；如果这个值是"0.75"，那么这个设备被认为是低密度并且默认情况下缩小0.75x。当然，Android浏览器和WebView所允许的缩放是基于网页的目标密度-正如[预定义viewport匹配密度](#)一节所述， 默认匹配的是中等密度，但是您可以改变这个密度匹配来改变不同屏幕密度下您网页的缩放。

例如,这里是您如何使用JavaScript请求设备的密度:

```
if (window.devicePixelRatio == 1.5) {  
    alert("This is a high-density screen");  
} else if (window.devicePixelRatio == 0.75) {  
    alert("This is a low-density screen");  
}
```

来自 "[index.php?title=Targeting_Screens_from_Web_Apps&oldid=9527](#)"



Building Web Apps in WebView

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：卡马克

完成时间：8月12日

主任务原文链

接：<http://developer.android.com/guide/webapps/webview.html>

目录

[[隐藏](#)]

[1 在WebView上构建Web应用程序](#)

- [1.1 在应用中添加一个WebView](#)
- [1.2 在WebView中使用JavaScript](#)
 - [1.2.1 开启JavaScript](#)
 - [1.2.2 绑定JavaScript代码到Android源码](#)
- [1.3 处理页面导航](#)
 - [1.3.1 操作网页历史](#)

在WebView上构建Web应用程序

如果你想实现一个Web应用（或仅仅是一个网页）作为你应用中的一部分，你可以使用[WebView](#)来实现它。[WebView](#)是Android的[View](#)类的扩展，它允许你显示一个网页作为Activity布局的一部分。它不包含成熟的浏览器的一些功能，例如导航控制或输入栏。默认情况下，[WebView](#)显示一个网页。

一个使用[WebView](#)的很普遍的场合是当你想要在应用中提供需要时常更新的信息时，例如用户协议或用户手册。在Android应用当中，你能创建一个[Activity](#)它包含一个[WebView](#)，然后使用它你可以显示网上提供的文档。

另一个使用[WebView](#)的场合是：假如你的应用总是需要网络链接来获取数据并提供给用户，例如电子邮件。这时候，你能发现在应用中创建一个[WebView](#)并使用网页来显示用户数据，比执行网络请求，解析数据然后再渲染到Android布局当中更加方便。你只需要根据Android设备设计网页，然后在应用当中实现一个[WebView](#)然后载入刚刚设计的网页。

这篇文档会教你如何开始学习[WebView](#)并实现一些其他功能，例如页面导航和绑定JavaScript到网页当中。

在应用中添加一个**WebView**

想添加一个[WebView](#)到应用当中，你只需要包含WebView标签到你的Activity布局文件当中。例如，下面是一个[WebView](#)填充满屏幕的布局

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview" android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
```

在[WebView](#)中载入网页，可以使用[loadUrl\(\)](#)方法，例如

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.baidu.com");
```

这段代码运行之前，你需要接入到因特网中。你仅需要添加因特网权限到你的Manifest文件当中。

```
<manifest ... >
<uses-permission android:name="android.permission.INTERNET" /> ...
</manifest>
```

这些是显示一个网页的基本知识。

在**WebView**中使用**JavaScript**

如果你想要载入的网页实现了JavaScript，你必须在[WebView](#)中开启JavaScript功能。当JavaScript有用时，你还可以再你的应用代码和JavaScript代码之间创建接口。

开启JavaScript

在[WebView](#)中JavaScript默认是不开启的。你可以通过[WebSettings](#)添加到[WebView](#)当中开启它。你可以使用[getSettings\(\)](#)方法获得[WebSettings](#)，然后使用[setJavaScriptEnabled\(\)](#)来开启它。例如：

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings =
myWebView.getSettings(); webSettings.setJavaScriptEnabled(true);
```

在[WebSettings](#)中你可以发现很多有用的功能。例如，如果你使用[WebView](#)开发一个网络应用，你可以使用[setUserAgentString\(\)](#)方法定义一个自定义用户代理字符串，后面可以查询网页中的自定义用户代理，辨别当前请求网页的是不是你的Android应用。

绑定JavaScript代码到Android源码

当在应用当中使用[WebView](#)创建网络应用的时候，你可以在JavaScript和Android源码之间创建一个接口，例如，你的JavaScript代码可以调用Android中定义的方法来显示一个对话框，而不适用JavaScript的[alert\(\)](#)方法。

例如你可以在应用当中添加以下的类：

```
public class JavaScriptInterface {
    Context mContext;
    /** Instantiate the interface and set the context */
    JavaScriptInterface(Context c) {
        mContext = c;
    }
    /** Show a toast from the web page */
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

在这个例子中，`JavaScriptInterface`类使用`showToast()`方法创建`Toast`消息。

你可以绑定这个类到JavaScript当中并在WebView当中使用addJavaScriptInterface()方法运行并创建接口的名称为Android。

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new JavaScriptInterface(this),
"Android");
```

这段代码会创建一个名称为“Android”的JavaScript接口并运行在[WebView](#)当中。在这里，你的Web应用会接入到JavaScriptInterface类。例如，下面为HTML何JavaScript代码，它会在用户点击按钮的时候使用新的接口创建一个[Toast](#)消息。

```
<input type="button" value="Say hello"
onClick="showAndroidToast('Hello Android!')"/>
<script type="text/javascript">
function showAndroidToast(toast) {
Android.showToast(toast);
}
</script>
```

你不需要初始化从JavaScript当中初始化Android接口。[WebView](#)会自动的让他可以使用在网页当中。因此，当点击按钮的时候showAndroidToast()方法会使用Android接口调用JavaScriptInterface.showToast()方法。

备注：在JavaScript中绑定的对象会运行在另一个线程，它和创建它的线程是不同的线程。

注意点：使用[java.lang.String\) addJavaScriptInterface\(\)](#)允许JavaScript来控制Android应用。这是一个非常有用和危险的。当在[WebView](#)当中的HTML是不能信任的（例如，HTML中的一些内容是由不认识的人或线程来提供），攻击者可以随意控制执行在客户端的HTML代码。同样的，除了你需要写[的HTML或JavaScript在\[WebView\]\(#\)当中呈现出来，你不应该使用\[java.lang.String\\) addJavaScriptInterface\\(\\)\]\(#\)方法。你也不能允许用户在\[WebView\]\(#\)中操纵除了自己外的别的网页（相反，你应该让用户调用默](#)

认的浏览器来打开外部的连接--一般，用户的网页浏览器会打开所有的URL链接，因此只有当操纵的网页跟下部分描述的一样）。

处理页面导航

当用户在你的WebView中点击一个链接时，对于Android来说默认的动作时启动一个应用程序捕捉它。通常，默认的网页浏览器会打开并且载入目标URL。然而，你可以在WebView中重写该动作，把该链接在你的WebView中打开。你还可以让用户自己处理历史的回退和前进功能。

打开用户点击的链接，仅仅在[WebView](#)中提供[WebViewClient](#)即可，使用[setWebViewClient\(\)](#)方法。例如：

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());
```

完成，现在用户点击的所有链接都会在[WebView](#)中打开。如果你想要在点击链接并载入网页的时候做更多的操作，请创建自己的[WebViewClient](#)并重写[java.lang.String\) shouldOverrideUrlLoading\(\)](#)方法。例如：

```
private class MyWebViewClient extends WebViewClient {
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
if (Uri.parse(url).getHost().equals("www.example.com")) {
//这是我的网页，所以不要重写，让我的WebView载入它
return false;
}
//如果不是我的网页，则启动另外的Activity来处理该URL
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
startActivity(intent);
return true;
}
```

然后给[WebView](#)创建一个[WebViewClient](#)的实例。

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new MyWebViewClient());
```

现在当用户点击一个链接，系统会调用[java.lang.String\)](#)

[shouldOverrideUrlLoading\(\)](#)，它会辨别URL主机是否要匹配指定的域名（就像我们上面列举的一样）。如果它不能匹配，则该方法返回false，一个[Intent](#)会创建并启动默认的Activity来处理该URL（它一般解析为用户的默认浏览器）。

操作网页历史

当你的[WebView](#)重写URL的载入，它会自动累积访问过的网页历史。你可以使用[goBack\(\)](#)方法和[goForward\(\)](#)方法操纵回退和向前功能。例如，下面是[Activity](#)使用返回按钮操作“回退”功能

```
@Override
public boolean onKeyDown( int keyCode, KeyEvent event ) {
// Check if the key event was the Back button and if there's history
if ( (keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack() ) {
myWebView.goBack();
return true;
}
// If it wasn't the Back key or there's no web page history, bubble
up to the default
// system behavior (probably exit the activity)
return super.onKeyDown(keyCode, event);
}
```

如果有网页历史则[canGoBack\(\)](#)方法返回True，同样的，你可以使用[canGoForward\(\)](#)来检查是否有“前进”历史。如果你不执行这个检查，当用户到达历史的尽头的时候，[goBack\(\)](#)和[goForward\(\)](#)什么都不做。

来自“[index.php?title=Building_Web_Apps_in_WebView&oldid=9660](#)”



Debugging Web Apps

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：卡马克

原文链接：<http://developer.android.com/guide/webapps/debugging.html>

Debugging Web Apps

如果在安卓4.4或更高版本上测试web应用程序，就可以用Chrome Developer Tools在WebView远程调试网页。欲了解更多信息，请参阅安卓远程调试。

如果没有运行安卓4.4或更高版本的设备，您可以使用 console JavaScript API 调试JavaScript并且查看输出信息。如果熟悉用 Firebug 或者 Web Inspector 调试网页，那可能熟悉使用console（比如console.log()）。安卓WebKit 框架支持大多数相同API，因此当在安卓浏览器或WebView调试时可以从网页接收日志。本文档介绍了如何使用console API进行调试。

在Android浏览器使用Console API

当你调用console函数(在DOM的window.console对象)，输出会显示在logcat当中。例如，如果你的网页执行如下JavaScript代码：

```
console.log("Hello World");
```

logcat输出的信息为

```
        Console: Hello World  
http://www.example.com/hello.html :82
```

根据你是用的Android版本的不同，可能信息显示的格式也不一样。在Android2.1和更高级的版本中，Android浏览器中的控制台信息将会

以“browser”标签来命名。而在Android1.6和更低的版本中，将
以“WebCore”来命名。Android Webkit不会实现console API的所有功能。
你可以使用基本的文本记录方法：

- console.log(String)
- console.info(String)
- console.warn(String)
- console.error(String)

其他的控制台方法将不会报任何错误，但可能跟你在使用别的浏览器中显示的结果不大一样。

在WebView中使用console API

如果你在应用中实现了一个自定义[WebView](#)，当你使用WebView进行调试的时候所有的API都将支持。在Android 1.6和更低版本中，控制台信息将会自动发往到以“WebCore”命名的logcat中 在Android 2.1和更高版本中，你必须提供WebChromeClient，它拥有一个onConsoleMessage()回调方法，用此方法来显示信息到logcat中。另外，API等级7中的[onConsoleMessage\(String, int, String\)](#)方法在API Level 8中被[onConsoleMessage\(ConsoleMessage\)](#)替换掉。不管你是使用Android 2.1还是Android 2.2，你必须提供一个重写[onConsoleMessage\(\)](#)回调方法的[WebChromeClient](#)方法。然后使用[WebView](#)的[setWebChromeClient\(\)](#)方法应用该WebChromeClient。

在Android 2.1中，[int, java.lang.String\) onConsoleMessage\(String, int, String\)](#)代码显示如下：

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebChromeClient(new WebChromeClient() {
    public void onConsoleMessage(String message, int lineNumber,
String sourceID) {
        Log.d("MyApplication", message + " -- From line " + lineNumber
+ " of " + sourceID);
    }
});
```

而Android2.2中 [onConsoleMessage\(ConsoleMessage\)](#)方法代码为：

```

WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebChromeClient(new WebChromeClient() {
    public boolean onConsoleMessage(ConsoleMessage cm) {
        Log.d("MyApplication", cm.message() + " -- From line "
+ cm.lineNumber() + " of " + cm.sourceId());
        return true;
    }
});

```

[ConsoleMessage](#)它还包括[MessageLevel](#)来表明不同类型的控制台信息。你可以使用[messageLevel\(\)](#)方法来查询消息等级，并决定该消息的严重性，然后使用合适的[Log](#)方法或采取别的适当的措施。

不论你是使用[int, java.lang.String\) onConsoleMessage\(String, int, String\)](#)还是[onConsoleMessage\(ConsoleMessage\)](#)，当你在网页中执行一个控制台方法，Android将会调用合适的[int, java.lang.String\) onConsoleMessage\(\)](#)方法，这将有助于你发布错误信息。例如，你如果使用上面列举的代码，[Logcat](#)信息将会如下打印出来：

```
Hello World -- From line 82 of http://www.example.com/hello.html
```

来自“[index.php?title=Debugging_Web_Apps&oldid=13825](#)”



Best Practices for Web Apps

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：卡马克

原文链接：<http://developer.android.com/guide/webapps/best-practices.html>

Best Practices for Web Apps

对于给手机设备开发网页或网络应用跟传统的使用桌面浏览器开发网页相比有更多不同的挑战。为了帮助你有个良好的开始，下列实践你可以练习，可以开发最高效的网络应用。

1. 重定向手机设备到指定的网站版本

这里有很多方法可以重定向捏请求到指定的手机网站，使用服务端重定向。通常来说，这个一般通过网络浏览器提供的用户代理字符串“sniffing”完成。为确定是否服务于移动版本的网站，你应该在用户代理中查询“mobile”字符串，它会匹配很多手机设备。如果有必要，你还可以鉴别具体的操作系统（例如“Android 2.1”）。

备注：大屏幕Android设备可以适配全尺寸网站（例如平板电脑），不要在用户代理中包含“mobile”字符串，然而剩下的用户代理字符串大部分都一样。同样的，根据在用户代理中的“mobile”字符串，实现移动版本的网站是很重要的。

2. 使用适合手机设备的有效地**DOCTYPE**标记

在手机网站中最普遍使用的标记语言是[XHTML Basic](#).这个标准保证在你网站中的具体标记可以运行的最好。举例来说，它不允许使用HTML frame和嵌套表格，他们在手机设备中运行的很糟糕。根据**DOCTYPE**，在文档中需要声明合适的编码。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
 "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

并且你还要确定网页中的标记是有效的，并且没有违反已声明的**DOCTYPE**。使用一个验证程序，例如你可以在<http://validator.w3.org>中寻找一个可用的。

3. 使用一个试图元标签重新定义网页的大小

在<head>标签中，你可以提供元标签信息用来指定你在浏览器中网页的渲染方式。例如，在视窗元数据中你可以指定高度和宽度，开始初始化的网页的比例和目标屏幕的密度。例如

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0, user-scalable=no">
```

更多的在Android设备中如何使用视窗元数据，请阅读[Targeting Screens from Web Apps](#)。

4. 避免多个文件请求

因为Android设备一般比桌面电脑网络请求速度慢很多，你应该尽快载入你要的网页。其中加速载入速度的一个方法是在<head>标签中避免载入额外的文件例如样式表和脚本文件。相反，你应该直接在<head>标签中提供样式表和脚本文件（或在<body>的末端，当载入完网页后再载入脚本文件）。作为一种选择，你应该通过压缩的方式优化文件的大小及速度。

5. 使用一个垂直现行布局

避免用户在操纵网页的时候左右拖动网页。上下拖动对于用户来说更加的简单并且网页也更简单。

想要了解更多的关于创建移动网络应用，请查看W3C的[Mobile Web Best Practices](#)。另外的一些提高网页站点速度方面的文章，请查看Yahoo的[Exceptional Performance](#) 和Google的[Let's make the web faster](#)。

来自“[index.php?title=Best_Practices_for_Web_Apps&oldid=9663](#)”



Best Practices

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： yinjiabin

完成时间： 8月3日

原文链接：<http://developer.android.com/guide/practices/index.html>

目录

[[隐藏](#)]

[1 Best Practices](#)

- [1.1 提高应用程序质量](#)
- [1.2 告别菜单按钮](#)
- [1.3 新工具来管理屏幕尺寸](#)
- [1.4 识别应用程序安装](#)
- [1.5 完美的Android游戏](#)

Best Practices

本节主要讲设计和构建应用程序的正确方法，学习如何创建看起来很好的应用程序并且尽可能在更多的设备上很好的运行，从手机到平板电脑甚至更多的设备上

[Android兼容性](#)

[博客文章](#)

[提高应用程序质量](#)

一个提高你应用程序能见度的生态体系是通过部署目标明确的移动广告和促销活动的深度。然而，还有另一个久经考验的方法的提高应用程序被安装的排名周期:提高产品!

[告别菜单按钮](#)

随着冰淇淋三明治推出更多的设备,对于你来说，最重要的是开始迁移您的设计工具栏以促进一个一致的Android的用户体验。

[新工具来管理屏幕尺寸](#)

Android 3.2包括新工具支持设备与各种各样的屏幕大小。一个重要的结果是更好地支持一个新的屏幕的大小,通常称为“7英寸的”平板电脑。这一版本还提供了几个新API文档来简化开发者的工作适应不同的屏幕大小。

[识别应用程序安装](#)

这是很常见的,而且完全合理的,因为开发人员想要追踪特定的安装自己的应用程序。这听起来貌似很有道理的只需要打电话给电话的管理者获取程序的ID并且使用这个值来确定安装。这是有问题的。

[完美的Android游戏](#)

制作一个游戏在Android上是很容易的。做一个很大的游戏在Android上是棘手的因为移动,多任务、通常多核、多用途系统。即使是最好的开发人员经常犯的错误互相作用的方式与Android系统和与其他应用程序

来自“[index.php?title=Best_Practices&oldid=7511](#)”



Supporting Multiple Screens

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：冰凝

原文链接：http://developer.android.com/guide/practices/screens_support.html

目录

- [1 支持多屏](#)
 - [1.1 多屏支持概述](#)
 - [1.1.1 术语和概念](#)
 - [1.1.2 支持的屏幕范围](#)
 - [1.1.3 密度无关性](#)
 - [1.2 如何支持多屏](#)
 - [1.2.1 使用配置限定符](#)
 - [1.2.2 设计可替代的布局和绘图](#)
 - [1.2.2.1 可替代的布局](#)
 - [1.2.2.2 可替代的绘图](#)
 - [1.3 Android3.2平板布局的声明](#)
 - [1.3.1 新的尺寸限定符的使用](#)
 - [1.3.2 配置实例](#)
 - [1.3.3 支持屏幕尺寸的声明](#)
 - [1.4 最佳实践](#)
 - [1.5 附加密度的注意事项](#)
 - [1.5.1 调整运行时创建的位图对象](#)
- [2 -= 转换dp单位为像素单位 =-](#)
 - [2.1 使用预先调整的配置值](#)
 - [2.2 如何在多屏上测试你的应用程序](#)

支持多屏

Android涉及各种各样的支持不同屏幕尺寸和密度的设备。对于应用程序，Android系统通过设备和句柄提供了统一的开发环境，大部分工作是校正每一个应用程序的用户界面到它显示的屏上。与此同时，系统提供APIs允许你控制应用界面为特定的屏幕尺寸和密度，为不同屏幕的配置提供最优化的用户界面设计。例如，你可能会要一个平板电脑的用户界面，这不同于手机的用户界面。

虽然系统能缩放，调整其尺寸，以使应用软件工作在不同屏上，但是应该尽量优化应用软件适应不同的屏幕尺寸和密度。为此，对所有设备的用户体验应最大化且应让用户们相信应用软件是真正为他们的设备设计的，而不是简单的拉伸使屏适合他们的设备。

按照文中描述的做法，通过使用一个apk文件，可以创建一个应用软件能恰当显示并在所有的支持屏配置中提供最优用户体验。

注：在此文中假设应用程序是为android 1.6 (API level为4) 或者更高的Android系统设计的。如果应用程序支持android 1.5或者更低的安卓系统，请首先阅读[Strategies for Android 1.5](#)章节。

Also, be aware that Android 3.2 has introduced new APIs that allow you to more precisely control the layout resources your application uses for different screen sizes. These new features are especially important if you're developing an application that's optimized for tablets. For details, see the section about Declaring Tablet Layouts for Android 3.2.

多屏支持概述

本节提供了Android支持多屏的概述，包括：介绍了本文中API用到的术语和概念，总结了系统支持的屏设置，概述了API和下面的屏幕兼容特性。

术语和概念

屏幕尺寸

实际的物理尺寸，是按照屏幕的对角线计量的。

为简单起见，Android把所有的屏幕尺寸划分为四种广义的尺寸：小、标准、大，特大号。

屏幕密度

屏幕占据的物理区域所含像素的个数；通常被称为**dpi**（每英寸点数）。

例如在给定的物理区域中，与“标准的”或“高”密度屏幕相比，低密度屏幕具有较少的像素。

方向

屏幕的方向来自于用户的角度。这是横向或纵向，分别指屏幕各个角度的比例，而不是宽或高。需要注意的是，不仅不同的设备在不同方向运行，而且当用户旋转设备时，方向也同时在改变。

分辨率

屏幕上物理像素的总数。支持多屏时，应用程序不直接与分辨率有关，应用程序应该只关心屏幕的尺寸和密度，用指定的广义的尺寸和密度组。

dp

一种有效的在定义UI布局时你应当使用的像素单位，以一种密度无关的方式表示布局的尺寸或者位置。

dp相当于160**dpi**屏幕，它是系统为“中等的”密度屏设定的基准密度。同时，系统透明地处理任何一种**dp**单位，必要时，基于使用中的屏的实际密度。**dp**单位根据公式 $px = dp * (dpi / 160)$ 简单地转化为屏像素。例如，一个240**dpi**的屏幕，1**dp**等于1.5个物理像素。定义应用程序的UI时，你应该总是使用**dp**单位，以确保在不同密度的屏幕上正确地显示你的UI。

支持的屏幕范围

从Android1.6（API等级为4）开始，Android提供了支持多个屏幕的尺寸和密度，表明一种设备拥有许多不同的屏幕配置。你应该利用Android系统的这些特性去为每一个屏幕配置优化你的应用程序界面，并且应确保你的应用程序不仅能正常运行，而且应尽可能地在每一个屏幕上提供最好的用户体验。

为了简化为多个屏的用户界面设计方式，Android系统将实际的屏幕尺寸和密度范围划分为：

- 一组广义的尺寸：小，标准，大，特大号。

注：从Android3.2（API等级为13）起，这些尺寸分组已被弃用，大家支持的是一种新的基于可用屏幕宽度的管理屏幕尺寸的技术。如果你正在开发Android3.2或者更高版本，参见Android3.2平板电脑布局章节获取更多信息。

- 一组四个广义的密度：ldpi（低），mdpi（中等），hdpi（高），和xhdpi（超高）。

这种广义的尺寸和密度是围绕一种基准的配置进行划分的，这种基准的配置是指一个标准尺寸和mdpi（中等）密度。这个基线是基于第一个Android上电设备，T-Mobile G1的屏幕配置，它具有HVGA屏幕（直到Android1.6，这是Android支持的唯一的屏幕配置）。

每个广义的尺寸和密度跨越一套实际屏幕尺寸和密度。例如，当用手测量时，两种标准的屏幕尺寸的设备可能具有实际的稍微不同的屏幕尺寸和纵横比。同样，两种hdpi屏幕密度的设备可能包含稍微不同的实际像素密度。Android制造这些差异使应用程序抽象化，所以，你可以提供设计的UI给广义的尺寸和密度，必要时让系统处理任何最后的调整。图1阐明了不同的尺寸和密度被如何大致归类到不同的尺寸和密度组。



图1 阐明了不同的尺寸和密度被如何大致归类到不同的尺寸和密度组

当为不同的屏幕尺寸设计UI时，会发现每个设计需要最低限度的空间。因此，上面提到的每一个广义的屏幕尺寸都有系统定义的相关联的最小分辨率。这些最小尺寸在“dp”单位内，当设计布局时，应当使用相同的单位。

- 超大屏幕至少960dp x720dp
- 大屏幕至少640dp x480dp
- 标准屏幕至少470dp x320dp
- 小屏幕至少426dp x320dp

注：在Android3.0之前，这些最小屏幕尺寸没有很好的定义，所以可能会遇到一些在标准和大之间被错误归类的设备。这些也是基于屏幕的物理分辨率，因此各种不相同的设备也会遇到前面的问题。例如，一个有系统栏的1024x720平板会留少一点可用的空间给应用程序。

为了优化应用程序的UI适应不同的屏幕尺寸和密度，可以提供任何广义的尺寸和密度替代资源。一般来说，应当提供替代布局给不同屏幕尺寸和替代的位图图像给不同的屏幕密度。在运行时，基于当前设备屏幕的广义的尺寸或密度，系统会为你的应用程序使用适当的资源。

没有必要提供替代资源给每个屏幕尺寸和密度的组合。系统提供了强大的兼容特性，这些特性会处理大部分工作使你的应用程序呈现在任何设备的屏幕上，前提是已经通

过使用允许它适当地调整尺寸的技术实现了UI（正如下面最佳实践中描述的）。

注：定义一种设备的广义的屏幕尺寸和密度特性是彼此相互独立的。例如，一种WVGA高密度屏幕被认为是标准尺寸的屏幕，是因为它的物理大小与T-Mobile G1（Android的第一个设备和基准屏幕设置）大约相同。另一方面，一种WVGA中等密度的屏幕被认为是大尺寸的屏幕。尽管它有相同的分辨率（相同数量的像素），这种WVGA中等密度的屏幕有更低的屏幕密度，意思是每个像素比较大，因此，整个屏幕比基准（标准尺寸）屏幕要更大。

密度无关性

当应用程序保留了用户界面元素的物理尺寸以不同的密度显示在屏幕上（从用户的角度来看）时，它实现了“密度无关性”。维护密度无关系性很重要，因为，如果没有它，一个UI元素（如按钮）在一个低密度屏幕上看起来较大而在一个高密度屏幕上看起来很小。这样的密度相关的尺寸的改变影响应用程序的布局和使用。图2和图3分别展示了当它不提供密度无关性和提供了密度无关性时，应用程序之间的差异。



图2 当不提供密度无关性，应用程序在低、中等、高密度屏幕上显示实例



图3 当提供密度无关性，应用程序在低、中等、高密度屏幕上显示实例

Android系统通过以下两种方式帮助应用程序实现密度无关性：

- 系统为当前屏幕密度调整dp单位到适当的值
- 如有必要，系统会根据当前屏幕密度调整绘图资源到适当的尺寸

在图2中，文本视图和位图绘图有规定的尺寸的像素（像素单位），因此这些视图在低密度屏幕上看起来较大，在高密度屏幕上看起来较小。这是因为尽管实际的屏幕尺寸是一样的，但是高密度屏每英尺有较多的像素（相同数量的像素适合于较小区域）。

在图3中，布局的尺寸被指定为密度无关性像素（dp单位）。因为密度无关性像素的基线是中等密度的屏幕，一个有中等密度屏幕的设备看上去与图2中的一样。对于低密度和高密度屏幕而言，不管怎样，系统能分别降低、调高密度无关性像素的值去恰当地适应屏幕。在大多数情况下，你可以简单地在密度无关性像素里（dp单位）指定所有的布局尺寸或者恰当地“包装内容”来确保应用程序的密度无关性。然后系统会根据恰当的缩放因子为当前屏幕密度调整位图视图以适当的尺寸显示出来。但是，位图缩放会导致图片模糊，如上面的截图。为了避免这些问题，应该为不同的密度提供替

代位图资源。例如，应该给高密度屏幕提供更高分辨率的位图，系统会使用它们，而不是使用为中等密度屏幕设计的缩放位图。以下段落将介绍更多关于如何提供不同替代资源给不同的屏幕配置。

如何支持多屏

Android支持多屏的基础是它能够以适当的方式为当前屏幕设置管理应用程序的布局和位图绘图的渲染。根据实际情况，系统通过缩放布局去适应屏幕的尺寸/密度和为屏幕密度缩放位图绘图，处理大部分工作去适当地渲染应用程序到每一个屏幕配置。然而，为了更好地处理不同屏幕配置，应该： *在清单文件中明确申明应用程序支持哪种屏幕大小

通过申明应用程序支持哪种屏幕尺寸，可以确保只有支持的屏幕尺寸的设备才能下载应用程序。声明支持不同屏幕尺寸也会影响系统如何在较大屏幕上运行应用程序，尤其是，不论应用程序是否运行在屏幕兼容模式。为了申明应用程序支持的屏幕大小，应该在manifest文件中包含<supports-screens>的元素。

* 为不同的屏幕尺寸提供不同的布局

默认情况下，Android会重新调整应用布局去适合当前设备屏幕。在大多数情况下，这样做很好。在其它情况下，UI可能看上去不太好且可能不同屏幕尺寸需要调整。例如，在较大屏幕上，可能会调整某些元素的位置和尺寸去充分利用额外的屏幕空间，或者在一个较小屏幕上，会调整尺寸使得一切都可以在屏幕上显示。可以提供指定大小资源的配置限定符，有小、标准、大、超大。例如，一个超大屏幕的布局应该选Xlarge。从Android3.2 (API等级为13) 起，上面的尺寸分组已被弃用，你应该使用sw<N>dp配置限定符去定义布局资源需要的最小的可用的宽度。例如，如果多窗格平板布局需要至少600dp的屏幕宽度，应该选sw600dp。使用新技术申明布局资源将会在Declaring Tablet Layouts for Android 3.2章节讨论。

* 为不同的屏幕密度提供不同的位图绘图

默认情况下，Android调整你的位图绘图(.png, .jpg, and .gif 文件)和9补丁绘图(.9.png 文件)，让他们在每个设备上以适当的物理尺寸呈现。例如，你用程序只为基线、中等屏幕密度 (mdpi) 提供了位图绘图，系统会调高高密度屏幕，降低低密度屏幕。这些调整会导致图片不真实。为了确保图片看起来最好，应当在不同分辨率下包含替代版本去适应不同的屏幕密度。可以用来指定密度资源的配置限定符有ldpi (低)、mdpi (中等), hdpi (高), and xhdpi (超高)。例如，高密度屏幕的位图应该选hdpi。尺寸和密度配置限定符与上面的支持的屏幕范围中描述的广义尺寸和密度一致。

注释：如果你不太熟悉配置限定符且不知道系统如何使用他们应用替代资源的话，请阅读[Providing Alternative Resources](#)章节获取更多信息。在运行时，对于任何给定的资源，系统通过以下步骤实现在当前屏幕上获取最佳的显示：

1. 系统使用适当的替代资源

基于当前屏幕的尺寸和密度，系统会使用应用程序里的任何指定尺寸和密度的资源。例如，如果设备有一个高密度屏幕且应用程序请求绘图资源时，，系统会在绘图资源目录寻找最匹配的设备配置。依赖于其他可用的替代资源，一个有hdpi限定符的资源目录（如 drawable-hdpi）可能是最匹配的，因此系统使用这个目录中的绘图资源。

2. 如果没有匹配的资源可用，系统会使用默认资源且会调高或降低资源去匹配当前屏幕的尺寸和密度这些“默认”资源是指那些没有配置限定符标记的资源。例如在drawable/中的资源就是默认的绘图资源。系统假定默认资源是为基准屏幕尺寸和密度设计的，即标准屏幕尺寸和中等密度。例如，系统会恰当地为高密度屏幕调高默认密度资源，为低密度屏幕降低默认资源。

然而，当系统在寻找一个指定密度的资源且在指定密度目录没找到它时，它不会总是使用默认资源。为了获取更好的效果，系统会使用一个其他指定密度资源。例如，当系统在寻找一个低密度资源且这个资源是不可用时，系统更喜欢降低高密度版本的资源，因为系统可以简单地乘以0.5系数将高密度资源降低为低密度，与调整中等密度资源乘以0.75系数相比，这样用到很少的工件。

关于Android是如何通过匹配配置限定符到设备配置来选择替代资源的更多信息，请阅读[How Android Finds the Best-matching Resource](#)章节。

使用配置限定符

Android支持多种配置限定符，让你控制系统如何基于当前设备屏幕的特征选择替代资源。一个配置限定符是一个字符串，你可以把它附加到你的Android工程的资源目录中并指定里面的资源是为此配置设计的。

为了使用配置限定符：

1. 在你的工程的res目录创建一个新目录，并使用格式命名：

<resources_name>-<qualifier>

<resources_name>是标准的资源名称（如drawable or layout）。

<qualifier>是下面表1中的配置限定符，指定这些资源将要被用的屏幕配置（如hdpi or xlarge）。

2. 保存这些适当的指定配置的资源到这个新目录。这些资源文件的命名必须严格与默

认的资源文件名一样。

例如，`xlarge`是用于超大屏幕的配置限定符。当你添加这些字符串到一个资源目录名（如`layout-xlarge`）时，它告诉系统这些资源将被用到有超大屏幕的设备上。

表1 允许你提供指定资源给不同屏幕配置的配置限定符

Screen characteristic	Qualifier	Description
Size	small	Resources for small size screens
	normal	Resources for normal size screens. (This is the baseline size.)
	large	Resources for large size screens.
	xlarge	Resources for extra large size screens
Density	ldpi	Resources for low-density (ldpi) screens (~120dpi).
	mdpi	Resources for medium-density (mdpi) screens (~160dpi). (This is the baseline density.)
	hdpi	Resources for high-density (hdpi) screens (~240dpi).
	xhdpi	Resources for extra high-density (xhdpi) screens (~320dpi).
	nodpi	Resources for all densities. These are density-independent resources. The

		system does not scale resources tagged with this qualifier, regardless of the current screen's density.
	tvdpi	Resources for screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. If you find it necessary to provide tvdpi resources, you should size them at a factor of 1.33*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi.
Orientation	land	Resources for screens in the landscape orientation (wide aspect ratio).
	port	Resources for screens in the portrait orientation (tall aspect ratio).
Aspect ratio	long	Resources for screens that have a significantly taller or wider aspect ratio (when in portrait or landscape orientation, respectively) than the baseline screen configuration.
	notlong	Resources for use screens that have an aspect ratio that is similar to the baseline screen configuration.

注：如果在Android3.2或者更高版本上开发应用程序，请参阅Declaring Tablet Layouts for Android 3.2章节获取关于新的配置限定符的信息，当申明了指定屏幕尺寸（而不是使用表1中的尺寸限定符的布局资源时，你应当使用这些限定符。

获取更多关于这些限定符如何大致对应于真实的屏幕尺寸和密度的信息，请参阅本文中前面提到的支持的屏幕范围章节。

例如，下面是应用程序中的资源目录列表，这个程序为中等、高及超高密度屏幕提供了不同的为不同屏幕尺寸和位图绘图设计的布局。res/layout/my_layout.xml // 标准屏幕尺寸的布局("默认") res/layout-small/my_layout.xml // 小屏幕尺寸的布局
res/layout-large/my_layout.xml // 大屏幕尺寸的布局 res/layout-xlarge/my_layout.xml // 超大屏幕尺寸的布局 res/layout-xlarge-land/my_layout.xml // 横向超大的屏幕布局

res/drawable-mdpi/my_icon.png // 中等密度的位图 res/drawable-hdpi/my_icon.png // 高密度的位图 res/drawable-xhdpi/my_icon.png // 超高密度的位图

如需要更多关于如何使用替代资源和完整的配置限定符列表（不仅仅是屏幕配置）的信息，请参阅Providing Alternative Resources。

请注意，当Android系统挑选资源时，它采用一定的逻辑来判定“最匹配”资源。也就是说，使用的限定符没必要在所有情况下，为了系统能用到它而严格匹配当前屏幕配置。具体来说，当基于尺寸的限定符选择资源时，如果没有更匹配的资源，系统会使用比当前屏幕更小的屏幕资源（例如，必要时，大尺寸屏幕将会使用标准尺寸屏幕资源）。但是，如果唯一可用的屏幕资源比当前屏幕大，系统将不会使用它们，而且如果没有其他匹配设备的配置的话，应用程序将会崩溃（例如，如果所有布局资源都被标记为xlarger限定符，但是设备是一个标准尺寸的屏幕）。更多关于系统如何选择资源的信息，请阅读How Android Finds the Best-matching Resource章节。

小提示：如果你有一些系统从未调整过的绘图资源（或许因为在运行时对其进行调整），应当把他们放置在nodpi配置个限定符的目录。有这些限定符的资源被认为是密度不可知的资源，系统将不会调整它们。

设计可替代的布局和绘图

应该创建的可替代资源的类型取决于应用程序的需要。通常，应该使用尺寸和方向限定符提供可替代的布局资源，使用密度限定符去提供替代的位图绘图资源。下面的段落分别总结了应该如何使用尺寸和密度限定符来提供替代的布局和绘图。

可替代的布局

一般情况下，一旦在不同屏幕配置上测试应用程序，应该知道是否需要为不同屏幕尺寸创建可替代的布局。例如：

- 当在小屏幕上测试时，可能会发现，布局不是很适合这个屏幕。例如，一排按钮可能不适合在小屏幕的设备的屏幕宽度内。这种情况下，应该为小屏幕提供一种可替代的布局，即通过调整这些按钮的大小或位置。
- 当在超大屏幕上测试时，可能会意识到，布局并没有有效地利用大屏幕，而是通过拉伸来填充它。在这种情况下，应该为超大屏幕提供一种可替代的布局，即可通过提供一种重新设计的最合适于较大屏幕如平板的UI。

虽然应用程序应当可以在没有可替代布局的大屏幕上工作正常，但是，对用户来说，程序看起来好像是专门为他们的设备设计的这一点非常重要。如果是很明显被拉伸的UI，用户对应用程序体验会更加不满意。

- 而且，当在横向测试时可能会注意到，与纵向相比较，放置在纵向屏幕底部的UI元素应该是在横向屏幕的右侧。

简而言之，应该确保应用程序的布局：

- 适合在小屏幕上（这样用户真正使用应用程序）
- 优化大屏幕，充分利用额外的屏幕空间
- 优化横向和纵向两个方向

如果在系统调整了布局后，UI要使用合适大小的位图（如按钮的背景图片），应当使用九补丁的位图文件。九补丁文件基本是一个指定的可拉伸的二维PNG文件。当系统需要调整正在使用的位图的视图时，系统会拉伸九补丁位图，但仅延伸指定区域。同样地，没有必要提供不同的绘图给不同的屏幕尺寸，因为九补丁位图能调整任何大小。然而，应当提供可替代的九补丁文件的版本给不同屏幕密度。

可替代的绘图

几乎每个应用程序应该有对应于不同屏幕密度的可替代的绘图资源，因为几乎每个应用程序都有一个启动图标，而且图标应该在所有屏幕密度上看起来都很好。同样，如果在应用程序中包含了其他位图绘图（如菜单图标或应用程序的其他图像），应当提供可替代的版本或者每一个版本给不同的密度。

注：只需要给位图文件（.png, .jpg, or .gif）和九补丁文件(.9.png)提供指定密度的绘

图。如果你使用XML去定义形状，颜色或者其他绘图资源，应该在默认的绘图目录(`drawable/`)做一个备份。



图4支持每个密度的位图绘图的相对尺寸

为了给不同密度创建可替代的位图绘图，应该遵循基于四种广义密度的3:4:6:8的缩放比例。例如，你有一个48x48像素的中等密度屏幕的位图绘图（一个启动图标的尺寸），所有不同的尺寸应该是：

- 36x36适合于低密度
- 48x48适合于中等密度
- 72x72适合于高密度
- 96x96适合于超高密度

获取更多关于设计图标的信息，请参阅[the Icon Design Guidelines](#)，文中包含了各种位图绘图的尺寸信息，如启动图标，菜单图标，状态栏图标，选项卡图标等等。

Android3.2平板布局的声明

对于第一代运行在Android3.0上的平板，正确声明平板布局的方法是把他们放到一个有`xlarge`配置限定符的目录里（例如，`res/layout-xlarge/`）。为了适应其他类型的平板和屏幕尺寸-尤其是7寸平板-Android3.2为更多离散的屏幕尺寸引进了一种新的指定资源的方式。这项新技术是基于你的布局需要的空间（如`600dp`的宽度），而不是试图让你的布局去适合广义的尺寸组（如`large or xlarge`）。

设计7寸平板的原因是非常复杂的，当时使用广义的尺寸组是指一个7寸的平板在技术上与一个5寸的手机在同一个组（大组）。虽然这两个设备在尺寸上看上去很接近，但是应用程序的UI的空间是显著不同的，用户交互的风格也是如此。因此，一个7寸和5寸的屏不应该总是使用同一个布局。为了把提供两种不同屏的布局变成可能，Android现在允许你基于宽度与/或者高度指定布局资源，在`dp`单位中指定，这对于应用程序布局很有效。

例如，在已经设计好了要用于平板类型的设备的布局后，当屏幕少于`600dp`宽时，可能会决定让布局停止工作。这个阈值因此会成为平板布局需要的最小尺寸。照此，现在会指定这些布局资源应当能被使用，仅仅在应用程序的UI有至少`600dp`宽度可用时。

应该要么选择宽度，并把它设计为最小尺寸，要么测试布局一旦完成时，它支持的最小宽度是多少。

注：请记住，所有的这些新尺寸的APPs使用的数字都是密度无关性像素（dp）值，且布局的维度应该总是被定义使用dp单位，因为关心的是，系统占用屏幕密度后可用屏幕空间的数量。更多关于密度无关性像素的信息，请阅读文中前面提到的Terms and concepts。

新的尺寸限定符的使用

不同能够指定的基于布局的可用空间的资源配置的总结见表2。与传统的屏幕尺寸组（小、标准、大和超大）相比，这些新的限定符提供了更多的控制权在指定的应用程序支持的屏幕尺寸方面。

注：指定的使用这些限定符的尺寸不是实际的屏幕尺寸。相反，在dp单位中的宽度或高度的尺寸对你的activity的窗口是可用的。Android系统可能会使用一些屏幕做系统UI（如屏幕底部的系统栏或顶部的状态栏），所以一些屏幕有可能对于你的布局是不可用的。因此，你申明的尺寸应该是，特别是你的activity需要的尺寸-当申明了系统能提供给你的布局多少空间时，系统会占用任何被系统UI使用的空间。还需要注意的是，工具栏被认为是应用程序窗口空间的一部分，尽管你的布局没有申明，因此，系统会给布局缩减可用空间，在设计时必须考虑到这点。

表2 屏幕尺寸的新配置限定符(Android 3.2中介绍的)

Screen configuration	Qualifier values	Description
smallestWidth	sw<N>dp Examples: sw600dp sw720dp	The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width (you may also think of it as the "smallest possible width" for the screen). You can use this qualifier to ensure that, regardless of the screen's current orientation, your application's has at least <N> dps of width available for its UI.

For example, if your layout requires that its smallest dimension of screen area be at least 600 dp at all times, then you can use this qualifier to create the layout resources, res/layout-sw600dp/. The system will use these resources only when the smallest dimension of available screen is at least 600dp, regardless of whether the 600dp side is the user-perceived height or width. The smallestWidth is a fixed screen size characteristic of the device; the device's smallestWidth does not change when the screen's orientation changes.

The smallestWidth of a device takes into account screen decorations and system UI. For example, if the device has some persistent UI elements on the screen that account for space along the axis of the smallestWidth, the system declares the smallestWidth to be smaller than the actual screen size, because those are screen pixels not available for your UI.

This is an alternative to the generalized screen size qualifiers (small, normal, large, xlarge) that allows you to define a discrete number for the effective size available for your UI. Using smallestWidth to determine the general screen size is useful because width is often the driving factor in designing a layout. A UI will often scroll vertically, but have fairly hard constraints on the minimum space it needs horizontally. The available

		width is also the key factor in determining whether to use a one-pane layout for handsets or multi-pane layout for tablets. Thus, you likely care most about what the smallest possible width will be on each device.
Available screen width	w<N>dp Examples: w720dp w1024dp	<p>Specifies a minimum available width in dp units at which the resources should be used—defined by the <N> value. The system's corresponding value for the width changes when the screen's orientation switches between landscape and portrait to reflect the current actual width that's available for your UI.</p> <p>This is often useful to determine whether to use a multi-pane layout, because even on a tablet device, you often won't want the same multi-pane layout for portrait orientation as you do for landscape. Thus, you can use this to specify the minimum width required for the layout, instead of using both the screen size and orientation qualifiers together.</p>
Available screen height	h<N>dp Examples: h720dp h1024dp etc.	<p>Specifies a minimum screen height in dp units at which the resources should be used—defined by the <N> value. The system's corresponding value for the height changes when the screen's orientation switches between landscape and portrait to reflect the current actual height that's available for your UI.</p> <p>Using this to define the height</p>

required by your layout is useful in the same way as w<N>dp is for defining the required width, instead of using both the screen size and orientation qualifiers. However, most apps won't need this qualifier, considering that UIs often scroll vertically and are thus more flexible with how much height is available, whereas the width is more rigid.

虽然使用这些限定符看上去比使用屏幕尺寸组更加复杂，但是当你一旦确定UI的需求后，这实际上应当更简单。当设计UI时，可能关心的主要事情是，应用程序在手机类型的UI和多窗格的平板类型的UI之间切换时的实际尺寸。切换的关键在于特定的设计，也许需要720dp的宽度给平板布局，也许600dp已足够，或者480dp，或者其他位于这之间的数值。使用表2中的限定符，需要掌握布局改变时的精确尺寸。

更多关于这些尺寸配置限定符的讨论，请参阅[Providing Resources](#)文档。

配置实例

为了帮助实现一些为不同类型的设备设计的目标，下面是一些典型的屏幕宽度的数值：

- 320dp: 一种典型的手机屏幕 (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, 等等).
- 480dp: 一种有点像条纹的中间态的平板 (480x800 mdpi).
- 600dp: 7寸平板 (600x1024 mdpi).
- 720dp: 10寸平板 (720x1280 mdpi, 800x1280 mdpi, etc).

使用表2中的这些尺寸限定符，应用程序可以在使用任何想要的宽度或高度的平板和手机的不同布局资源间切换。例如，如果平板布局支持的最小可用宽度是600dp，应该提高两套布局：

`res/layout/main_activity.xml # 手机适用`

`res/layout-sw600dp/main_activity.xml # 平板适用`

在这种情况下，为了使平板布局能用，可用的屏幕空间的最小宽度必须是600dp。

对于其他情况，你要进一步自定义你的UI以区分如7寸和10寸平板的尺寸，可以定义额外的最小宽度布局：

res/layout/main_activity.xml # 适用于手机 (小于600dp的可用宽度)

res/layout-sw600dp/main_activity.xml #适用于7寸平板(大于等于600dp的可用宽度)

res/layout-sw720dp/main_activity.xml # 适用于10寸平板(大于或等于720dp的可用宽度)

请注意，第二套采用的“可用的宽度”限定符w<N>dp。通过这种方式，一个设备可能实际上使用两种布局，这取决于屏幕的方向（如果在一个方向上可用的宽度是至少600dp，在另一个方向上少于600dp）。

如果关心的是可用高度，可以同样使用h<N>dp限定符。或者，如果情况比较特殊，甚至可以结合w<N>dp 和 h<N>dp一起使用。

支持屏幕尺寸的声明

一旦已经实现了不同屏幕尺寸的布局，在manifest文件中声明应用程序支持哪种屏幕也同样重要。

随同新的屏幕尺寸的配置限定符一起，Android3.2引进了新的<supports-screens> manifest元素属性。

android:requiresSmallestWidthDp

指定需要的最小smallestWidth。smallestWidth是必须能被应用程序的UI利用的屏幕空间的（在dp单位里）最小尺寸。也就是，最短的可用的屏幕的二维尺寸。因此，为了让设备与应用程序兼容，设备的smallestWidth必须大于等于这个值。

（通常，不论屏幕当前的方向是什么，你提供的值是你的布局支持的“最小宽度”。

例如，如果应用程序是为600dp的最小可用宽度的平板类型设备设计的：

```
<manifest ... >
    <supports-screens android:requiresSmallestWidthDp="600" />
    ...
</manifest>
```

然而，如果应用程序支持所有Android支持的屏幕尺寸（如426dp x 320dp一样

小），那么没有必要申明这个属性，因为需要的最小宽度可能在任何设备上都是最小的。

注意：Android并不关心这个属性，因此它不会影响应用程序在运行时的行为。相反，它常常会为应用程序在服务如谷歌播放上进行过滤。然而，谷歌播放目前不支持过滤属性（在Android3.2上），因此，如果应用程序不支持小屏幕，应该继续使用其它尺寸的属性。

android:compatibleWidthLimitDp

这个属性允许通过指定应用程序支持的最大“最小宽度”将屏幕的兼容模式作为一个用户可选特征。如果设备的可用屏幕最小边大于这个值，用户仍然可以安装应用程序，但是不能在屏幕的兼容模式上运行。默认情况下，屏幕的兼容模式是禁用的，和平常一样，布局会被调整到适合屏幕，但是按钮在系统栏是可用的，它允许用户开启和关闭屏幕的兼容模式。

注：如果应用程序的布局正好适合大屏幕，就没必要使用这个属性。我们建议避免使用这个属性，而不是按照本文档中的建议确保布局适合更大屏幕。

android:largestWidthLimitDp

这个属性通过指定你的应用程序支持的最大“最小宽度”强制开启屏幕的兼容模式，如果设备的可用屏幕最小边大于这个值，应用程序会运行在屏幕兼容模式上，且用户没有办法去禁用它。

注：如果应用程序的布局正好适合大屏幕，就没必要使用这个属性。我们建议避免使用这个属性，而不是按照本文档中的建议确保你的布局适合更大屏幕。

注意：当在Android3.2或者更高版本上开发时，不应该使用较旧的屏幕尺寸属性并结合上面列出的属性。同时使用新属性和较旧的尺寸属性会导致不可预料的事情发生。

更多关于这些属性的信息，请查阅以上相应的链接。

最佳实践

支持多个屏幕的目的是为了创建一个能正常运行，且在任何Android支持的广义的屏幕配置上看起来都很舒服的应用程序。本文的前面章节提供了关于Android如何使应用程序适应屏幕配置和如何在不同屏幕配置上自定义应用程序的外观的信息。这节提供了一些额外的技巧来确保应用程序适用于不同屏幕配置的技术。

下面是关于如何确保你的应用程序能够恰当地显示在不同屏幕上的快速检查清单：

1. 当在XML布局文件中指定尺寸时，使用**wrap_content**, **fill_parent**, 或者**dp**单位
2. 在应用程序代码中不要使用硬编码的像素值
3. 不要使用绝对布局（已被弃用）
4. 对不同的屏幕密度采用可替代的位图绘图

下面章节将讲述更多细节。

1. 布局尺寸使用**wrap_content**, **fill_parent**, 或者 **dp**单位

当在XML布局文件中定义视图的**android:layout_width**和 **android:layout_height**时，使用**wrap_content**, **fill_parent**, 或者 **dp**单位以保证在当前设备屏幕上能给视图分配一个适当的尺寸。

例如，一个**layout_width**为100dp的视图在中等密度的屏幕上是100像素，在高密度屏幕上系统将把它调整到150dp，于是视图在屏幕上占用了大致相同的物理空间。

同样地，应该更喜欢用**sp**（与比例无关的像素）来定义文本的尺寸。**Sp**比例因子取决于用户的设置和系统调整的尺寸与它为**dp**调整的相同。

2. 在应用程序代码中不要使用硬编码的像素值

出于性能方面的原因及为了保持代码更简单，Android系统采用像素作为尺寸或坐标值的标准单位。意思是，在代码中，视图的尺寸总是用像素表达，但总基于当前屏幕密度。例如，如果**myView.getWidth()**函数的返回值是10，在当前屏幕上视图有10个像素宽度，但是在更高密度屏幕的设备上，返回值可能是15.如果在你的应用程序代码中，使用像素值为位图的单位，且该位图不是为当前屏幕密度预先调整的，可能会调整这些在代码中你用以匹配未调整的位图源的像素值。

假设应用程序在运行时巧妙地处理位图或像素值，请参阅下面的章节**Additional Density Considerations**。

3. 不要使用绝对布局

不像其他的布局部件，绝对布局强制使用固定位置给子视图布局，这很容易导致用户界面不能很好地工作在不同的屏幕上。正因为如此，在Android1.5（API等级为3）中已经弃用了绝对布局。

相反，应该使用相对布局，它会使用相对位置为它的子视图布局。例如，可以指定按钮部件应该在文本部件的右侧。

4. 使用尺寸和指定密度资源

虽然系统会基于当前屏幕配置调整你的布局和绘图资源，但是可以在不同屏幕尺寸上调整UI，且提供最优化的位图绘图给不同密度。这在本文的前面已经反复强调过。

如果需要严格控制应用程序在各种屏幕配置上的显示情况，那么在指定配置资源目录中调整布局和位图绘图。例如，假设希望图标显示在中等和高密度屏幕上。简单地创建两个不同尺寸的图标（例如100x100用于中等密度，150x150用于高密度），把这两个变体放在适当的目录，使用适当的限定符：

`res/drawable-mdpi/icon.png //适合于中等密度屏幕`

`res/drawable-hdpi/icon.png //适合于高密度屏幕`

注：如果密度限定符没有定义在目录名中，系统会假定在那个目录里的资源是为基线中等密度设计的，将会适当地为其他密度做调整。

更多关于有效的配置限定符的信息，请参阅本文前面结束的Using configuration qualifiers。

附加密度的注意事项

本节描述了更多关于系统如何在不同屏幕密度上调整位图绘图、以及如何更好地控制位图在不同密度上的显示信息。本节中的信息对大多数应用程序应该不是很重要，除非应用程序在不同屏幕密度上运行时或者应用程序篡改了图像时，遇到了问题。为了更好地了解在运行过程中改变了图像时如何做到支持多密度，应该了解，系统通过以下几种方式确保合适的位图尺寸：

1. 预先调整的资源（如位图绘图）

基于当前屏幕的密度，系统使用应用程序中任何指定尺寸和密度的资源，并显示

出来且不需要任何调整。如果资源在当前密度不可用，系统将会下载默认资源并调整他们使之适合于当前屏幕密度。系统认为默认资源（无配置限定符目录的资源）是为基准屏幕密度（**mdpi**）设计的，除非它们是从指定密度资源目录下载的。预先调整是指调整位图到当前屏幕密度适合的尺寸时系统所做的事情。

如果你请求预先调整好的资源的尺寸，系统会返回调整后代表该尺寸的值。例如，一个50x50 像素的**mdpi**屏幕的位图要在**hdpi**屏幕上扩大为75x75像素（如果此时没有可替代资源给**hdpi**），系统会这样返回此值。

有一些情况下，可能不需要Android的预先调整的资源。避免预先调整的最简单方式是将资源放置到**nodpi**配置限定符的目录中。例如：

`res/drawable-nodpi/icon.png`

当系统使用这个文件夹中的icon.png位图时，它不会基于当前屏幕密度去调整该位图。

2. 像素尺寸和坐标值的自动调整

应用程序可以通过在清单文件中设置**android:anyDensity**的属性为“假”或在程序中设置位图的**inScaled**值为“假”禁止预先调整资源。在这种情况下，系统在绘图时会自动调整绝对像素的坐标值和像素尺寸。这样做的目的是，为了确保已定义像素的屏幕元素仍然能以接近他们在基线屏幕密度（**hdpi**）上的物理尺寸显示出来。系统透明地处理这种调整并把调整后的像素尺寸，而不是物理像素尺寸告诉应用程序。

例如，假设一个设备有WVGA高密度屏幕，即480x800，这与传统的HVGA屏幕的尺寸大约相同，但是它运行着已禁止预先调整资源功能的应用程序。在这种情况下，当系统在查找屏幕尺寸时，它会“欺骗”应用程序，给它返回值320x533（转化成屏幕密度接近**mdpi**）。然后，当应用程序开始绘图操作时，如使矩形从(10,10) 扩大到 (100, 100) 变成无效，系统通过缩放接近数量的值调整坐标，且把区域(15,15) 扩大到 (150, 150) 变成无效。如果你的应用程序直接使用这个调整后的位图，这个误差会导致不可预期的行为，但是这被认为是合理的折中的尽可能保持应用程序性能的方法。如果遇到这种情况，请阅读下面关于Converting dp units to pixel units章节。

通常情况下，不应该禁止预先调整资源。支持多屏的最好方式是按照上面的How to Support Multiple Screens提到的基本技术操作。

如果应用程序在屏幕上以其它某种方式操作位图或与像素直接交互，你可能需要采取额外的步骤来支持不同屏幕密度。例如，如果你通过数手指划过时的像素值的方式响应触摸，你需要使用适当的密度无关性像素值，而不是实际的像素值。

调整运行时创建的位图对象

如果应用程序创建一个内存中的位图（位图对象），系统认为这个位图是为基线中等密度屏幕设计的，默认情况下，在绘制时自动调整位图。当位图没有指定密度特性时，系统采用“自动调整”技术。如果没有正确地考虑到当前屏幕密度，也没有指定位图的密度特性，自动调整会导致人为缩放，这与没有提供可替代资源时一样。

为了控制在运行时创建的位图是否需要调整，你可以通过`setDensity()`指定位图的密度，从`DisplayMetrics`传递一个密度常量，比如`DENSITY_HIGH` 或 `DENSITY_LOW`。

如果正在创建一个使用`BitmapFactory`（如从文件或者流）的位图，可以使`BitmapFactory`。选择定义一个已经存在的位图的特性，这决定系统是否或如何调整位图。例如，可以使用`Density`来定义位图是为哪种密度设计的，用`Scaled`去指定位图是否应该调整到匹配当前设备的屏幕密度。

如果设置`Scaled`为假，禁用了任何系统会用于位图的预先调整功能，系统在运行时将会自动调整它。使用自动调整而不是预先调整会耗费更多CPU，但是占用更少的内存。



图5 预先调整和自动调整位图的演示比较结果

图5 演示了当在高密度屏幕上下载低 (120)，中等(160) 和高 (240) 密度位图时，预先调整和自动调整机制的结果。区别是微妙的，因为所有的位图都被调整以匹配当前屏幕密度，然而调整过的位图的外观稍微不同，这取决于在绘制时采用的是预先调整还是自动调整。你会找到此示例应用程序的源代码，在`ApiDemos`里，演示了如何使用预调整和自动调整位图。

注：在Android3.0或者以上版本，由于图形框架的改进，预先调整和自动调整位图之间应该没有明显的差异。

-= 转换dp单位为像素单位 =-

在某些情况下，需要用`dp`来表示尺寸，然后把他们转换为像素。想象一下，在应用程序中滚动或者扔的手势在用户的手指划过至少16像素后才能被识别出。在基准屏幕上，在手势被识别出来之前，用户必须划过16像素除160dpi，这等于十分之一英寸（或者2.5毫米）。在一个高密度屏幕（240dpi）设备上，用户必须划过16像素除240dpi，即十五分之一英寸（或者1.7毫米）。这个距离相当短，因此对用户来说，

程序看上去更加敏感。

要解决这个问题，手势阈值必须用dp表示，然后转换成实际的像素。例如：

```
// The gesture threshold expressed in dp
private static final float GESTURE_THRESHOLD_DP = 16.0f;

// Get the screen's density scale
final float scale = getResources().getDisplayMetrics().density;
// Convert the dps to pixels, based on density scale
mGestureThreshold = (int) (GESTURE_THRESHOLD_DP * scale + 0.5f);

// Use mGestureThreshold as a distance in pixels...
```

根据当前屏幕密度，`DisplayMetrics.density`指定了你必须使用的将dp单位转为像素单位的比例因子。在一个中等密度屏幕上，`DisplayMetrics.density`等于1.0；在高密度屏幕上，`DisplayMetrics.density`等于1.5；在一个超高密度屏幕上，`DisplayMetrics.density`等于2.0；在低密度屏幕上，`DisplayMetrics.density`等于0.75.这个比例因子乘以dp单位得到的值就是当前屏幕的实际像素值（然后添加0.5f做四舍五入，转化的整数）。更多信息，请参看`DisplayMetrics`类。

然而，不是为这里事件定义任意阈值，而是你应该使用预先调整的可从`ViewConfiguration`获取的配置值。

使用预先调整的配置值

可以使用`ViewConfiguration`类去访问Android使用的共同的距离、速度和时间。例如，框架使用的作为滚动阈值的距离可通过`getScaledTouchSlop()`得到：

```
private static final int GESTURE_THRESHOLD_DP
=ViewConfiguration.get(myContext).getScaledTouchSlop();
```

以前缀`getScaled`开头的`ViewConfiguration`的方法保证返回值以像素为单位，无论当前屏幕的密度是什么，它都能正确显示。

如何在多屏上测试你的应用程序

在发布应用程序之前，应该在所有支持的屏幕尺寸和密度上彻底地测试应用程序。`Android`的SDK包含了你可以使用的模拟器，它复制了应用程序可以运行的通用的屏幕配置的尺寸和密度。可以修改模拟器默认的尺寸，密度和分辨率以复制任何指定屏幕的特征。使用模拟器和额外的自定义配置让你可以测试任何可能的屏幕配置，因此不必买各种设备来测试应用程序支持的屏幕。

为了建立测试应用程序支持的屏幕环境，通过使用模拟器和模仿应用程序支持的屏幕的尺寸和密度的屏幕配置，应当创建一组AVDs（Android虚拟设备）。要做到这一

点，可以使用AVD管理器去创建AVDs，然后以图形界面方式启动它们。

为了启动Android SDK管理器，在Android SDK目录（在windows上）执行SDK Manager.exe或者在<sdk>/tools/目录执行android。图6展示了用来测试各种屏幕配置的选择了AVDs的AVD管理器。



图6展示了用来测试各种屏幕配置的选择了AVDs的AVD管理器。

更多有关创建和使用AVDs测试应用程序的信息，请参阅Managing AVDs with AVD Manager。

表3 Android SDK中从模拟器获取的各种屏幕配置和其它典型的分辨率

	Low density (120), ldpi	Medium density (160), mdpi	High density (240), hdpi	Extra high density (320), xhdpi
Small screen	QVGA (240x320)		480x640	
Normal screen	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x960
Large screen	WVGA800** (480x800) WVGA854** (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024		
Extra Large screen	1024x600	WXGA(1280x800)† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

* 为了模仿此配置，在创建一个使用WVGA800或者WVGA854外观的AVD时指定：自定义的密度为160。
** 为了模仿此配置，在创建一个使用WVGA800或者WVGA854外观的AVD时指定：自定义的密度为120。
† 这个外观是Android3.0平台可用的。

想要知道支持任何给定屏幕配置的有源设备的相对数量，请参看屏幕尺寸和密度仪表板。

我们也建议在物理尺寸接近匹配的实际设备的模拟器上测试应用程序。这使得比较各种尺寸和密度上的测试结果变得非常容易。做到这点，需要知道电脑显示器（例如，30寸的Dell显示器的密度大约为960dpi）的以dpi为单位的近似密度。当从AVD管理器启动AVD时，可以在启动选项中，如图7所示，指定模拟器的屏幕尺寸和显示器的dpi。



图7 当从AVD管理器启动AVD时，你能设置的尺寸和密度

如果想在内置外观不支持的分辨率或密度的屏幕上测试应用程序，可以创建一个使用自定义分辨率或密度的AVD。在创建AVD时，应指定分辨率，而不是选择内置的外观。

如果正通过命令行启动AVD，可以指定通过选项参数-scale来指定比例。例如：

```
emulator -avd <avd_name> -scale 96dpi
```

为了改变模拟器的大小，可以通过参数选项选择想要的比例因子0.1-3实现。

更多关于从命令行创建AVDs的信息，请参阅Managing AVDs from the Command Line。

来自 "[index.php?title=Supporting_Multiple_Screens&oldid=11095](#)"

Distributing to Specific Screens

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文：<http://developer.android.com/guide/practices/screens-distribution.html>

翻译：Fiestina菲

目录

[[隐藏](#)]

1 适配指定屏幕

- [1.1 声明应用只适于手机](#)
- [1.2 声明应用只适于平板电脑](#)
- [1.3 为不同屏幕发布多版APK](#)

适配指定屏幕

尽管我们推荐，设计出的应用应该在多种屏幕尺寸和密度的配置上正常运行，但你也可以选择使其仅适配几种指定屏幕。比如仅适用于平板电脑或其他较大设备，或仅适用于手机或尺寸较小的设备。这样一来，你就可以通过外部服务（如安卓市场）——在清单文件（manifest file）中添加应用支持的屏幕配置——实现筛选。然而，在决定应用对指定屏幕的适配之前，你应当了解支持多种屏幕（[supporting multiple screens](#)）的技巧，并且尽量将其实现。只有支持多种屏幕，你的应用才有可能用一个APK，获取多种设备的最大用户量。

声明应用只适于手机

因为通常系统在大屏手机上适配的更好，所以不必将大屏设备筛掉。只要

按照屏幕独立最佳范例 ([Best Practices for Screen Independence](#)) 操作，你的应用会在大屏设备上运行更好，比如在平板电脑上。然而，你也许会发现应用并不能完美放大比例，或也许你已经决定为不同的屏幕配置发布两个版本的应用。在这种情况下，你可以使用<[compatible-screens](#)>元素，在结合屏幕尺寸和密度的基础上来处理应用的适配。

外部服务，如安卓市场则使用这一信息来为应用筛选——只有屏幕配置和应用声明的兼容性一致时，才能下载应用。<[compatible-screens](#)> 元素必须包含一个以上的<[screen](#)>元素。通过使用[android:screenSize](#)和[android:screenDensity](#) 属性，每个<[screen](#)>元素可以指定一个和应用匹配的屏幕配置。每个<[screen](#)>元素必须包含以上两个属性，来制定一个 独立屏幕配置；若任一属性丢失，则元素无效（外部服务，如安卓市场则会无视）。

例如，如果你的应用仅适配于小屏和普通屏手机，先不论屏幕密度，你必须指定八个不同的<[screen](#)>元素——因为每个手机尺寸有四种密度配置，你必须一一将其声明。任何未指定的尺寸和密度的组合，都将被视为应用不匹配的屏幕配置。这里提供一个清单，说明在应用仅适配于小屏和普通屏手机的情况：

```

<manifest ... >
    <compatible-screens>
        <!-- all small size screens -->
        <screen android:screenSize="small"
android:screenDensity="ldpi" />
        <screen android:screenSize="small"
android:screenDensity="mdpi" />
        <screen android:screenSize="small"
android:screenDensity="hdpi" />
        <screen android:screenSize="small"
android:screenDensity="xhdpi" />
        <!-- all normal size screens -->
        <screen android:screenSize="normal"
android:screenDensity="ldpi" />
        <screen android:screenSize="normal"
android:screenDensity="mdpi" />
        <screen android:screenSize="normal"
android:screenDensity="hdpi" />
        <screen android:screenSize="normal"
android:screenDensity="xhdpi" />
    </compatible-screens>
    ...
    <application ... >
        ...
    </application>
</manifest>
```

注：虽然你也可以用`<compatible-screens>`元素来处理反面情况（即你的应用不匹配小屏手机），但是使用下章中的`<supports-screens>`会更简单，因为它不需要你指定应用所支持的每种屏幕密度。

声明应用只适于平板电脑

如果你不想手机用户安装你的应用（也许你的应用在大屏幕上才有使用价值时），或者你需要时间来为小屏手机进行优化，你可以通过`<supports-screens>`清单元素来防止小屏设备下载应用，如下：

```
<manifest ... >
    <supports-screens android:smallScreens="false"
                      android:normalScreens="false"
                      android:largeScreens="true"
                      android:xlargeScreens="true"
                      android:requiresSmallestWidthDp="600" />
    ...
    <application ... >
        ...
    </application>
</manifest>
```

这在两个不同的方面描述应用支持的手机尺寸：它声明，该应用不支持“小”和“普通”屏幕尺寸，通常意义上不含平板电脑。它声明，该应用最小可用区域为至少600dp宽。第一种技巧涉及运行3.1和其之前版本的设备，因为这些设备声明其尺寸基于广义的屏幕尺寸。`requiresSmallestWidthDp`属性涉及3.2和其之后版本的设备，包括在最小可用密度独立像素（dip）数的基础上，该应用指定尺寸条件的能力。在本例中，该应用声明最小宽度条件为600dp，这通常意味着7"或更大的屏幕。

你的选择也许会不尽相同。当然，这基于你如何在不同屏幕尺寸上设计；例如，如果你为9"或更大屏设计，你就要设定最小宽度为720dp。重点在于，要使用`requiresSmallestWidthDp`属性，你必须为3.2编译该应用。较早版本不识别该属性，并且会报错：compile-time error。最安全的做法是，先为`minSdkVersion`设定好API等级，再在与此相匹配的平台上开发应用；在为构建发布候选版本做最后准备时，将生成目标改变为安卓3.2，并加

上`requiresSmallestWidthDp`属性。比3.2更老的版本将会无视此XML属性，所以就可以避免运行失败。

关于为何“最小宽度”的屏幕尺寸对支持不同屏幕尺寸有重要作用，请阅读管理屏幕尺寸的新工具[\[1\]](#) (New Tools for Managing Screen Sizes) 以了解更多信息。

注意：如果使用`<supports-screens>`元素来处理反面情况（即你的应用不匹配大屏手机），并设置大屏尺寸属性为“false”，那么外部服务（如安卓市场）则不会执行该筛选。你的应用仍然可以安装在大屏手机上，但在运行的时候将不会根据屏幕重新调整尺寸。相反地，系统将会模拟出手机屏幕尺寸（大约320dp x 480dp；更多信息请参阅屏幕适配模式[Screen Compatibility Mode](#)）。如果你想防止应用被大屏手机下载，请使用`<compatible-screens>`，方法在之前的章节声明应用只适用于手机中已讨论过。

请记住，你应当运用支持多种屏幕中的一切技巧，将应用尽量适配更多的设备。当你不能在所有屏幕配置上进行适配，或决定为不同屏幕配置提供不同版本进行适配时，才应该用`<compatible-screens>`或`<supports-screens>`。

为不同屏幕发布多版APK

虽然我们推荐一个应用仅做一版APK，但是安卓市场允许为同款应用发布不同版本的APK，来满足屏幕适配（如清单文件中所声明）。例如，如果你想同时发布一个应用的手机版和平板电脑版，但你又不能让同一个APK满足这二者的屏幕尺寸，那么你可以为一个应用列表发布两个APK。根据每款设备的屏幕配置，安卓市场将会呈递相应的、声明过支持该配置的APK。

请注意，然而为同一应用发布多版APK是一种高级特征，且大多数应用应当仅发布一个能支持大范围设备配置的APK。只要根据支持多屏幕尺寸的指导，用一个APK来适配多种屏幕尺寸还是合情合理的。

关于如何在安卓市场上发布多版APK，如果你想了解更多，请阅读多版APK支持（[Multiple APK Support](#)）。

来自“[index.php?title=Distributing_to_Specific_Screens&oldid=7478](#)”



Screen Compatibility Mode

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Fiestina菲

完成时间： 8月9日

原文地址：<http://developer.android.com/guide/practices/screen-compat-mode.html>

屏幕兼任模式

注意：如果你在低于安卓3.0的版本上进行应用开发，但其在更大屏幕的设备（比如平板电脑）上显示正常时，你就需要禁用屏幕兼任模式来保持最佳用户体验。要学习如何快速禁用用户选项，请跳转至禁用屏幕兼任模式。

屏幕兼任模式是一种改善方法，用于不能正常适配大屏设备（如平板电脑）的情况。从安卓1.6开始，系统就能支持多种屏幕尺寸，并且调整应用布局的显示，来适应每一款屏幕。然而，如果在支持多种屏幕[Supporting Multiple Screens](#)的指引下，你的应用仍不能成功适配屏幕，其问题就很有可能就出在适配更大屏幕上。对于存在这种问题的应用，屏幕兼任模式可以让其在更大屏设备上得到改善。目前有两个版本的屏幕兼任模式，它们稍有不同：

版本一（安卓1.6-3.1）

系统将应用UI显示为“邮票”式窗口。即，系统将应用的布局认定为适应普通尺寸手机（模拟320dp x 480dp的屏幕），窗口以外的屏幕部分显示为黑色背景。该版本随安卓1.6的产生而出现，安卓1.6的设计就是仅适用于320dp x 480dp的原始尺寸。因为现在安卓1.5的设备很少使用了，几乎所有应用都在安卓1.6或更高的平台上开发，所以应该不会有版本一的屏幕兼任模式来适配更大屏幕。该版本差不多已经过时了。



图1. 在安卓3.2平板电脑上，应用以屏幕兼任模式运行的情况。



图2. 图1中的应用，在禁用屏幕兼任模式下的运行情况。

要禁用该版本的屏幕适配模式，你只需要设定[android:minSdkVersion](#)或[android:targetSdkVersion](#)至“4”或更高，或设置[android:resizeable](#)至“true”。

版本二（安卓3.2或更高）

系统按照在普通屏幕手机上的显示方式（大致模拟320dp x 480dp的屏幕）显示应用，然后进行放大，填满屏幕。实际上就是“拉大”应用布局，这通常会引起UI的伪迹或像素丢失。

该版本随着安卓3.2的产生而出现，在应用还没有实施支持多种屏幕[Supporting Multiple Screens](#)中所提及的技术时，该版本起到了进一步支持应用在最新设备上的正常显示的作用。

这样屏幕兼任模式就能一直启用，用户不可禁用。（下面的章节将会讨论如何声明支持大屏幕）。



图3. 开关屏幕适配模式的弹出菜单（当前为禁用状态，仅正常调整）。

作为开发者，你应该掌握在何时该使用屏幕兼任模式。以下章节将告诉你，在安卓3.2或更高版本下，如何选择禁用或启用屏幕兼任模式来适配更

大屏幕。

禁用屏幕兼任模式

如果你的应用最初是为低于3.0的版本开发的，而它能在更大屏幕上正常显示，那么你就应当禁用屏幕兼任模式，从而保证最佳用户体验。否则，用户也许会启用屏幕兼任模式，这样就会影响最佳的应用体验效果。

在默认下，当一下可选特性之一为true时，3.2或更高版本设备的屏幕兼任模式将可用：

- 将应用中`android:minSdkVersion`和`android:targetSdkVersion`同时设为"10"或更低，并且使用`<supports-screens>`元素，不要声明支持大屏。
- 将应用中`android:minSdkVersion`或`android:targetSdkVersion`二者之一设定为"11"或更高，并且使用`<supports-screens>`元素，声明其不支持大屏。

要想完全禁用屏幕兼任模式的用户选项并且移除系统栏上的图标，你可以：

- 最简单：

在清单文件中，添加`<supports-screens>`元素，指定`android:xlargeScreens`属性为"true"；

```
<supports-screens android:xlargeScreens="true" />
```

就这么简单。这样就声明了你的应用支持所有大屏幕尺寸，系统也会根据屏幕调整布局。不论你在`<uses-sdk>`属性中设定了什么值，它都会起作用。

- 简单，但有其他效果：

在清单的`<uses-sdk>`元素中，设定`android:targetSdkVersion`为"11"或更高：

```
<uses-sdk android:minSdkVersion="4"
```

```
android:targetSdkVersion="11" />
```

这种方法可声明应用支持安卓3.0，并且可以在更大屏幕（平板电脑等）上起作用。

谨慎：3.0或更高版本上，该方法也可启用UI的全息主题，启用时将在Activity上添加[Action Bar](#)，且移除系统栏上的选项菜单按钮。

如果在你改变此处以后，屏幕兼任模式仍然启用，请检查清单中的[`<supports-screens>`](#)，确认其中没有设定为“false”的属性。最佳做法是使用[`<supports-screens>`](#)元素，声明其支持不同屏幕尺寸，这样你就可以一直使用该元素了。

关于针对安卓3.0设备升级应用的更多信息，请见[Optimizing Apps for Android 3.0.](#)

启用屏幕兼任模式

当应用针对安卓3.2（API等级13）或更高时，你可以使用[`<supports-screens>`](#)元素，针对某些屏幕启用或禁用屏幕兼任模式。

注意：屏幕兼任模式并不是一种最佳的选择——他会因缩放而导致UI的像素丢失和虚化。让应用在大屏手机上正常显示的最佳方法，请参见支持多种屏幕，并且为不同屏幕尺寸提供替换性选择。

在默认下，当设定`android:minSdkVersion`或`android:targetSdkVersion`二者之一为“11”或更高时，屏幕兼任模式将对用户不可用。若二者之一为true，且应用不能针对大屏进行适当调整时，你选择以下一种方式，来启用屏幕兼任模式：
· 在清单文件中，添加 `<supports-screens>`元素并且指定`android:compatibleWidthLimitDp`属性为“320”：

```
<supports-screens android:compatibleWidthLimitDp="320" />
```

这表明，该应用“最小屏幕宽度”的最大值为320dp。通过该方法，任何大于此值的最小屏幕值都可将屏幕兼任模式设为用户的可选特性。

注意：现今，屏幕兼任模式进模拟屏幕为320dp宽，所以，如果[android:compatibleWidthLimitDp](#)的值大于320，屏幕兼任模式将不会在任何设备上启用。

- 如果应用在针对大屏调整时出现功能性崩溃，所以你想强迫用户使用屏幕兼任模式（而不仅仅是提供这个选项），你可以使用[android:largestWidthLimitDp](#)属性。

```
<supports-screens android:largestWidthLimitDp="320" />
```

这个属性和[android:compatibleWidthLimitDp](#)效果一样，除了强制启用屏幕兼任模式而不允许用户禁用。

来自“[index.php?title=Screen_Compatibility_Mode&oldid=7964](#)”

Supporting Tablets and Handsets

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：<http://developer.android.com/guide/practices/tablets-and-handsets.html#Guidelines>

作者：长剑耿介 & &Monica & ~ ~ELite~ ~

目录

- [1 支持平板电脑和手机](#)
 - [1.1 基本方针](#)
 - [1.2 创建单窗格和多窗格布局](#)
 - [1.3 Action bar\(工具栏\)的应用](#)
 - [1.3.1 应用分离Action Bar](#)
 - [1.3.2 应用“UP”导航](#)
 - [1.4 其他的设计建议](#)

支持平板电脑和手机

Android平台运行在各种屏幕尺寸的设备，系统会调整应用程序UI的大小，以适合每个设备。通常情况下，你所需要做的是设计灵活的UI，通过提供替代资源([alternative resources](#))优化一些元素适用于不同尺寸的设备。（如调整一些视图或改变视图的尺寸值等替代的布局）。但是，有时您可能希望进一步地优化不同的屏幕尺寸上整体的用户体验。例如，平板电脑提供了更多的空间，您的应用程序可以一次提出多套信息，而手机设备通常需

要分割拆开集合，分别显示。因此，即使一个为手机设计的UI可以适当调整以适应平板电脑，但是它没有充分利用平板电脑的屏幕来提高用户体验的潜力。

到了Android 3.0 (API 11级)，Android的推出了一套新的框架的API，使您能够更有效地利用大屏幕设计活动：[Fragment APIs](#)。片段

(Fragments) 允许把你的UI中不同的行为的组件分成独立的部分。当运行在平板电脑上时，可以将这些组件创建成多窗格布局，或在手机上运行时，放置在分开的活动中运行。Android 3.0还介绍了[ActionBar](#)，在屏幕上提供了一个用来确定应用程序专用的用户界面，并提供用户操作和导航。

通过片段 (Fragments) 和工具栏 (action bar) 可以帮助你创建一个独特的、最佳的用户体验应用程序，无论是在手机还是在平板电脑上，本文档会给你提供指导。

在阅读本指南之前，建议您先阅读 [Supporting Multiple Screens](#)的指导。该文档分别介绍了开发支持不同的屏幕尺寸、灵活布局和替代位图的密度的用户界面的基本设计原则。

基本方针

这里有一些指导将会帮助您创建一个提供优化的用户体验应用程序，无论是在在平板电脑还是在手机上：

- 在片段 (**fragments**) 基础上创建您的活动 (**activity**) 可以对不同组合重复使用——在多窗格布局的平板电脑或是单窗口布局的手机上。一个[Fragment](#)代表了一种行为，或是活动中的用户界面的一部分。你可以把一个片段作为一个活动的模块化部分 (一个活动的“片段”)，它有自己的生命周期，在活动运行时，你还可以添加或移除它。如果你还没有使用过的片段，从阅读的[Fragment](#)开发人员指南开始吧。
- 使用工具栏 (**action bar**)，但得按照最佳做法，并确保您的设计非常灵活使得系统能根据屏幕的大小调整操工具栏布局。[ActionBar](#)是用来取代活动 (**activity**) 中屏幕顶部传统标题栏的一个UI组件。默认情况下，工具栏

的左侧是应用程序标识（**logo**），其次是活动标题，右侧是“选项”菜单上的项目。

您可以把选项菜单中的项目直接显示在工具栏中的“工具项目”中。您还可以添加导航功能到工具栏，如标签或下拉列表中，并且使用应用程序的图标来补充系统返回按钮的行为，以导航到您的应用程序的“主界面

（**home**）”或到达应用程序的结构层次的上层。本指南提供了同时支持平板电脑和手机的工具栏使用方式的一些技巧。欲知工具栏（**action bar**）API的详细讨论，请参阅的[ActionBar](#)开发人员指南。

- 实现灵活的布局，正如在最佳实践（[Best Practices](#)）中对支持多屏幕的讨论，要像网页设计师一样思考（[Thinking Like a Web Designer](#)）。

一个灵活的布局设计让您的应用程序能够适应屏幕尺寸的变化。正如平板电脑的尺寸不尽相同，手机也有着不同尺寸。尽管你可能会为“平板电脑”和“手机”提供不同的片段（**fragment**）组合，但使得每个设计都能够灵活调整其大小尺寸和高宽比仍然是必要的。以下各节将会详细讨论前两个建议。欲了解更多有关创建灵活的布局信息，请参阅上面提供的链接。

注：除了在工具栏的一个特点（译者按：指上面的最佳实践（[Best Practices](#)）和要像网页设计师一样思考（[Thinking Like a Web Designer](#)）），所有的API需要完成本文档中的建议都可以在Android 3.0查询。此外，您甚至可以实现片段的设计模式，并通过使用支持库来保持与Android 1.6的向后兼容，这些在下面的侧栏可以查询。

创建单窗格和多窗格布局

最有效的方法是要结合不同的方式来创建“单窗格中的”布局手机和“多窗格中的”布局片的片段，来优化您的手机和平板电脑的应用程序。例如，在平板电脑上的新闻应用程序可能会在左侧显示文章列表的，在右侧显示列表中所选择的文章内容。然而，在手机上，这两个部分应出现在单独的屏幕

上，从列表中选择一篇文章标题后进入到全屏显示该文章。

有两种技术来实现这种设计的片段：

- 对于任何屏幕，其中显示您的平板电脑版本的多个片段，为手机使用相同的活动，但一次只显示一个片段 - 内活动，在必要时交换的片段。
- 在手机上使用单独的活动，主办每个片段。例如，当片剂UI使用一个活动的两个片段，适用于手持设备中使用的相同的活性，但提供一个替代的布局，包括只有一个片段。当你需要切换片段（例如，当用户选择一个项目），启动另一个活动举办的其他片段。

您选择的方法取决于您的应用程序的设计和个人喜好。在第一个选项（单个活动）需要你动态添加的每个片段的活性，在运行时---而不是宣布在您的活动的布局文件的片段 - 因为你可以不删除活动中的一个片段，如果它的已运行在XML布局中。您可能还需要更新操作栏中的每个时间片段的变化，这取决于什么样的行动或导航模式提供的片段。在某些情况下，这些因素可能不会影响到您的应用程序，因此，使用一个互动的碎片会使你的应用程序很好地工作。然而，在其他情况下，只使用一个互动交换片段也可以使你的代码更复杂，因为你必须管理所有活动的代码，而不是利用替代布局文件的片段组合。

本文谈的第二个选项，更详细。这可能是一个多一点的前期工作，必须很好地操作，因为每个片段分布在不同的活动，但它通常能使你的应用程序变的更好。这意味着你可以使用其他的布局文件定义不同的片段组合，保持片段代码的模块化，简化操作栏中的管理，让系统来处理所有的后退堆栈工作。

下图演示了如何可以安排使用单独的活动时，为手机设计的手机和平板电脑应用程序的两个片段：



在这个程序，活动，一个是 主要活动 的时间，这取决于大小的屏幕上显示一个或两个片段，并使用不同的布局。一个手机大小的屏幕上时，布局只包含片段A（列表视图）的一个药片大小的屏幕上时，布局包含片段A和片段B。

注：活动B从来没有在平板电脑上使用。这是一个简单的容器呈现片段B，所以只用在手持设备的两个片段时，必须分别显示。

根据不同的屏幕尺寸，该系统适用于不同的main.xml的布局文件：
res/layout/main.xml 手机布局

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <!-- "Fragment A" -->
    <fragment class="com.example.android.TitlesFragment"
        android:id="@+id/list_frag"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</FrameLayout>
```

res/layout-large/main.xml 平板布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/frags">
    <!-- "Fragment A" -->
    <fragment class="com.example.android.TitlesFragment"
        android:id="@+id/list_frag"
        android:layout_width="@dimen/titles_size"
        android:layout_height="match_parent" />
    <!-- "Fragment B" -->
    <fragment class="com.example.android.DetailsFragment"
        android:id="@+id/details_frag"
        android:layout_width="match_parent" />
```

```
        android:layout_height="match_parent" />
</LinearLayout>
```

Note: Although the above sample layout for tablets is based on the "large" screen configuration qualifier, you should also use the new "minimum width" size qualifiers in order to more precisely control the screen size at which the system applies your handset or tablet layout. See the sidebar for more information.

应用程序的响应，当用户从列表中选择一个项目，取决于是否是片段B在布局。如果片段B是存在的，活动通知片段B进行自我更新。如果片段B是A开始在布局，片段A主持片段B。

您的应用程序要实现这种模式，需要将你的片段高度条块分割，这一点很重要。具体而言，你应该遵循两个通用的准则：

- 请勿直接操作其他的某个片段。
- 保留所有代码，涉及的内容有子片段，而不是在主机活动的代码片段。

为了避免从一个片段中直接调用的另一个片段，在每个片段中，类声明一个回调接口，它可以使事件传送给主机的活动，实现回调接口。当活动由于一个事件（例如用户选择的列表项）接回调，适当地根据行为上为当前片段进行配置。

例如，从上面处理这样的活动选择项：

```
public class MainActivity extends Activity implements
TitlesFragment.OnItemSelectedListener {
    ...
    /**
     * This is a callback that the list fragment (Fragment A)
     * calls when a list item is selected */
    public void onItemSelected(int position) {
        DisplayFragment displayFrag = (DisplayFragment)
getFragmentManager()
            .findFragmentById(R.id.display_frag);
        if (displayFrag == null) {
```

```
// DisplayFragment (Fragment B) is not in the layout
(handset layout),
    // so start DisplayActivity (Activity B)
    // and pass it the info about the selected item
Intent intent = new Intent(this, DisplayActivity.class);
intent.putExtra("position", position);
startActivity(intent);
} else {
    // DisplayFragment (Fragment B) is in the layout (tablet
layout),
    // so tell the fragment to update
    displayFrag.updateContent(position);
}
}
```

当DisplayActivity（活动）启动时，通过它的DisplayFragment（片段B）读取数据传递的意图。

如果片段B需要提供一个结果传回片段A，则该过程的工作方式类似于片段B和活动B的一个回调接口，活动B实现了一个回调接口，片段B获得的回调，并完成自身活动。活动接收结果，并提供片段A.

对于一个完整的示范这项技术用于创建不同的片段组合为不同的平板电脑和手机，看看代码，此更新版本的 [Honeycomb Gallery](#) (ZIP文件)

Action bar(工具栏)的应用

在平板和手机上，[Action bar](#)对于Android应用来说是个重要的组件。为了确保[Action Bar](#)可以适应所有的屏幕分辨率，应用[ActionBar](#)的API而没有添加的定制。通过应用标准的ActionBar API设计你的工具栏，为了[ActionBar](#)更好适应不同分辨率，Android系统做了很多工作。当你创建ActionBar时，这里有一些可供遵循重要的建议：

- 当设置菜单项作为工具栏的子项时，避免用`always`值。在你的菜单资源里面如果你想 在你的菜单项出现在你的工具栏中，使用`"android:showAsAction"`的`"ifRoom"`属性.然而 你可能需要`"always"`属性，当一个动作视图并不为溢出菜单(必须出现在动作视图中的) 提供一个默认的动作。但是你不能用`always`一次或两次以上。几乎在所有其他情况下， 当你想该条目作为一个动作项出现，使

用"android:showAsAction"值为 "ifRoom"。迫使太多的动作项到ActionBar中会导致混乱的UI，并且动作项会与其他的ActionBar部分重叠 像标题栏和导航栏。

- 当用文本标题向ActionBar添加动作项时，提供了一个适当的并且声明了showAsAction="ifRoom|withText"的图标，这样的话，如果标题没有足够的空间，但只有在图标应用的情况下，图标会有足够的空间。
- 通常为你的动作项添加一个标题，即使你不启用"withText"，但是用户可以通过长按动作项查看标题，标题文本会立刻出现在Toast信息中。提供标题对可访问性来说是必要的，因为屏幕阅读器会大声朗读项标题即使看不见
- 如果可能的话尽量不用传统的导航模式。如果可能的话用**built-in tab**和**drop-down**导航模式，他们是为了系统可以适应他们对于不同屏幕分辨率的描述。例如，当宽度太窄了，这标签和其他的动作项（例如手机的竖屏），选项卡出现在ActionBar（这被称为堆叠动作栏）下面。如果你一定要创建一个传统的导航模式或者其他视图模式，完全在小的屏幕上测试做任何必要的调整以支持一个狭窄的工具栏。

例如，下面的实物模型演示系统如何适应一个工具栏，可以基于可用的屏幕空间。在手机上只有两个动作项适用，剩余的菜单项出现在溢出菜单中（因为android:showAsAction设置为ifRoom）以及选项卡独立出现一行（溢出工具栏）在平板电脑上，更多的动作项可以适合在动作栏和标签



图形2：Mock-up 展示系统再配置的工具栏组件基于可用的屏幕空间

应用分离Action Bar

当你的应用程序运行在Android4.0 (API level 14) 或者更高的版本，还有一个额外的模式可供操作栏称为“分隔工具栏”。当你用分割工具栏时，一个分隔条会出现在屏幕的底部来展示所有的动作项，当Activity运行在窄的屏幕上（例如竖屏手机）。分隔工具栏确保合理数量的空间可用来显示动作项目在一个狭窄的屏幕，在顶部还留出空间为导航和标题元素。使用分割条，简单的添加uiOptions="splitActionBarWhenNarrow"在你

的<[activity](#)>和<[application](#)>在manifest中



图形3：左边是带有导航条的 Split action bar，右边是没有应用程序图标和标题

如果你想隐藏上面的主ActionBar，因为你正在应用内置的导航条以及分隔条，用[setDisplayShowHomeEnabled\(false\)](#)方法来隐藏应用图标和ActionBar。这样，在主的ActionBar里面就没有东西了，因此所以它就消失了，剩下的是导航标签顶部和底部的动作项，如图所示的第二个设备在图3中。

注释：虽然，`uiOptions`属性在Android 4.0(API level 14)添加上了，你可以安全地将其包括在您的应用程序，即使你[minSdkVersion](#)设置为一个值低于“14”保持兼容旧的Android。当运行旧的版本时，该系统简单地忽略了属性，因为它并不识别他。唯一的条件添加到Manifest中，必须编译您的应用程序对平台的版本，它支持API级别14或更高。只是要确保在您的应用程序中不公开使用其他api不支持您声明的最小版本的代码。

应用“UP”导航

开发者指南中说道，您可以使用应用程序图标，方便用户操作栏导航当一个方法跳转到主界面（类似于在网站上点击logo），一个提高应用的机构层次。在其他情况下，她可能看起来像标准的黑色导航条，`up navigation option` 提供了up导航选项提供了一个更可预测的导航方法的情况下，用户可能已经进入了从外部位置像通知，桌面小组件或者一个其他的应用当使用不同的组合碎片为不同的设备，重要的是要给予额外的考虑如何在每种配置了导航的行为。例如，当在手机上并且你的显示一次并且有一个片段它可能是合适的启用了导航到父屏幕，而这是没有必要的，在显示同样的片段在多窗格的配置。

更多关于up导航请参考[ActionBar](#)开发指南

其他的设计建议

- 当使用[Listview](#)时，考虑你可能提供或多或少的信息在每个基于可用空间的选项里。也就是说你可以在你的列表适配器中，你可以创建每一个列表项使用复杂的布局 这样一个大的屏幕就可以为每个列表项展示更多的细节
- 创建替代资源文件值,比如整数、尺寸、甚至是布尔值.使用这些资源的大小限定符,你可以很容易地应用不同的布局大小,字体大小,或启用/禁用特性根据当前的屏幕大小.

来自“[index.php?title=Supporting_Tablets_and_Handsets&oldid=10821](#)”

Designing for Performance

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Yuxlong2010

原文链接：<http://developer.android.com/intl/zh-CN/guide/practices/performance.html>

目录

[[隐藏](#)]

[1 Designing for Performance](#)

- [1.1 简介](#)
- [1.2 明智的优化](#)
- [1.3 避免创建不必要的对象](#)
- [1.4 性能之谜](#)
- [1.5 用静态代替虚拟](#)
- [1.6 避免内部的Getters/Setters](#)
- [1.7 对常量使用Static Final修饰符](#)
- [1.8 使用改进的For循环语法](#)
- [1.9 在私有内部类中，考虑用包访问权限替代私有访问权限](#)
- [1.10 合理利用浮点数](#)
- [1.11 了解并使用类库](#)
- [1.12 合理利用本地方法](#)

Designing for Performance

一款安卓应用将会运行在受限于有限的计算能力、存储能力和电池续航的移动装置上。因此，它必须高效。即使你的应用看上去已经运行的“足够快”

了”，但电池续航可能是你想继续优化的原因。对于用户来说，电池续航很重要，当电池消耗的速度很快时，用户会想知道是否是你的应用耗干了电池。

虽然这篇文档只涵盖了那些微小的优化，但这些绝不应该是你软件成败的关键。选择正确的算法和数据结构应该始终是你优先考虑的。

简介

要想写出高效的代码，应该遵循以下两个最基本的原则：

- 不做没有必要做的工作。
- 避免不必要的内存浪费。

明智的优化

这篇文档是关于安卓中特别的细微优化，所以假定你已经通过性能工具分析出哪些代码需要优化，并且对于你做出任何的改动都有办法测试出效果（好或坏）的情况下。你唯一需要投入的就是工作时间，所以最重要的就是合理的规划时间。

（查看最后的总结去了解和撰写更多的高效标准。）

这篇文档还假定你已经选择了最优的算法和数据结构，并且也考虑到**API**的选择所带来的潜在影响。使用正确的算法和数据结构将会比任何建议要起作用，并且考虑到**API**选择所带来的潜在影响将会使以后选择更好的实现变得更加容易（相比应用代码，在这一点上类库代码显得更重要）。

（如果你需要那样的建议，可查看Josh Bloch's Effective Java, item 47.）

当对安卓应用进行细微优化的时候，最棘手的问题是你要保证你的应用在多个配置不同的硬件上都能运行起来。不同版本的虚拟机会在不同的处理器上以不同的速度运行。它并不像通常所说的设备X比设备Y快或慢，再把你的结果类比到其它的设备上。特别是在模拟器上只能测试出很有限的结果相比于在真是设备上。设备上是否有JIT将会带来很大的差别。不见得带有JIT的设备上的代码就一定比不带有的强。

如果你想知道你的应用在一个给定的设备上的表现，你就需要在该设备上

测试。

避免创建不必要的对象

对象的创建永远都是有代价的。每条线程的分代GC都会给临时对象分配一个地址池以节省开支，但是不分配内存永远都比分配内存更奢侈。

如果在用户界面周期中分配对象，你会强制进行一个周期的垃圾回收，这将会给用户带来一丝停顿的用户体验。**Gingerbread** 中介绍的并发回收也许有用，但是没有必要做的工作应该避免。

因此，你应该避免创建没有必要的对象实例，下面的一些例子也许对你有帮助。

- 如果有一个返回**String**的方法，并且他的返回值常常附加在一个**StringBuffer**上，改变声明和实现，让函数直接在其后面附加，而非创建一个短暂存在的零时变量。
- 当从输入的数据集合中读取数据时，考虑返回原始数据的子串，而非新建一个拷贝。这样你虽然创建一个新的对象，但是他们共享该数据的**char**数组。（结果是即使仅仅使用原始输入的一部分，你也需要保证它的整体一直存在于内存中。）

一个更彻底的方案是将多维数组切割成平行一维数组：

- **Int**类型的数组常有余**Integer**类型的。推而广之，两个平行的**int**数组要比一个(**int,int**)型的对象数组高效。这对于其他任何基本数据类型的组合都通用。
- 如果需要实现一个容器来存放元组 (**Foo,Bar**)，两个平行数组**Foo[],Bar[]**会优于一个 (**Foo,Bar**) 对象的数组。（例外情况是：当你设计API给其他代码调用时，应用好的API设计来换取小的速度提升。但在自己的内部代码中，尽量尝试高效的实现。）

通常来讲，尽量避免创建短时零时对象。少的对象创建意味着低频的垃圾回收。而对于用户体验产生直接的影响。

性能之谜

前一个版本的文档给出了好多误导人的主张，这里做一些澄清：

在没有JIT的设备上，调用方法所传递的对象采用具体的类型而非接口类型会更高效（比如，传递HashMap map比Map map调用一个方法的开销小，尽管两个map都是HashMap）。但这并不是两倍慢的情形，事实上，他们只相差6%，而有JIT时这两种调用的效率不相上下。

在没有JIT的设备上，缓存后的字段访问比直接访问快大概20%。而在有JIT的情况下，字段访问的代价等同于局部访问，因此这里不值得优化，除非你觉得他会让你的代码更易读（对于final ,static，及static final 变量同样适用）

用静态代替虚拟

如果不需要访问某对象的字段，将方法设置为静态，调用会加速15%到20%。这也是一种好的做法，因为你可以从方法声明中看出调用该方法不需要更新此对象的状态。

避免内部的**Getters/****Setters**

在源生语言像C++中，通常做法是用Getters (`i=getCount()`) 代替直接字段访问 (`i=mCount`)。这是C++中一个好的习惯，因为编译器会内联这些访问，并且如果需要约束或者调试这些域的访问，你可以在任何时间添加代码。

而在Android中，这不是一个好的做法。虚方法调用的代价比直接字段访问高昂许多。通常根据面向对象语言的实践，在公共接口中使用Getters和Setters是有道理的，但在一个字段经常被访问的类中宜采用直接访问。

无JIT时，直接字段访问大约比调用getter访问快3倍。有JIT时（直接访问字段开销等同于局部变量访问），要快7倍。在Froyo版本中确实如此，但以后版本可能会在JIT中改进Getter方法的内联。

对常量使用**Static Final**修饰符

考虑下面类首的声明：

```
static int intValue = 42;
static String strValue = "Hello, world!";
```

编译器会生成一个类初始化方法<clinit>,当该类初次被使用时执行，这个方法将42存入intValue中，并得到类文件字符串常量strValue的一个引用。当这些值在后面被引用时，他们通过字段查找进行访问。

我们改进实现，采用 final关键字：

```
static final int intValue = 42;
static final String strValue = "Hello, world!";
```

类不再需要<clinit>方法，因为常量通过静态字段初始化器进入dex文件中。引用intValue的代码，将直接调用整形值42；而访问strValue，也会采用相对开销较小的“字符串常量”（原文：“string constant”）指令替代字段查找。

（这种优化仅仅是针对基本数据类型和String类型常量的，而非任意的引用类型。但尽可能的将常量声明为 static final是一种好的做法。

使用改进的For循环语法

改进for循环（有时被称为“for-each”循环）能够用于实现了iterable接口的集合类及数组中。在集合类中，迭代器让接口调用 hasNext() 和 next() 方法。在ArrayList中，手写的计数循环迭代要快3倍（无论有没有JIT），但其他集合类中，改进的for循环语法和迭代器具有相同的效率。

这里有一些迭代数组的实现：

```
static class Foo {
    int mSplat;
}
Foo[] mArray = ...

public void zero() {
    int sum = 0;
    for (int i = 0; i < mArray.length; ++i) {
        sum += mArray[i].mSplat;
    }
}

public void one() {
```

```

int sum = 0;
Foo[] localArray = mArray;
int len = localArray.length;

for (int i = 0; i < len; ++i) {
    sum += localArray[i].mSplat;
}

public void two() {
    int sum = 0;
    for (Foo a : mArray) {
        sum += a.mSplat;
    }
}

```

zero()是当中最慢的，因为对于这个遍历中的历次迭代，JIT并不能优化获取数组长度的开销。

One()稍快，将所有东西都放进局部变量中，避免了查找。但仅只有声明数组长度对性能改善有益。

Two()是在无JIT的设备上运行最快的，对于有JIT的设备则和one()不分上下。他采用了JDK1.5中的改进for循环语法。

结论：优先采用改进for循环，但在性能要求苛刻的ArrayList迭代中，考虑采用手写计数循环。

(参见 Effective Java item 46.)

在私有内部类中，考虑用包访问权限替代私有访问权限

考虑下面的定义：

```

public class Foo {
private class Inner {
    void stuff() {
        Foo.this.doStuff(Foo.this.mValue);
    }
}

private int mValue;

public void run() {
    Inner in = new Inner();
    mValue = 27;
    in.stuff();
}

```

```

private void doStuff( int value ) {
    System.out.println( "Value is " + value );
}
}

```

需要注意的关键是：我们定义的一个私有内部类（`Foo$Inner`），直接访问外部类中的一个私有方法和私有变量。这是合法的，代码也会打印出预期的“`Value is 27`”。

但问题是，虚拟机认为从`Foo$Inner`中直接访问`Foo`的私有成员是非法的，因为他们是两个不同的类，尽管Java语言允许内部类访问外部类的私有成员，但是通过编译器生成几个综合方法来桥接这些间隙的。

```

/*package*/ static int Foo.access$100( Foo foo ) {
    return foo.mValue;
}

/*package*/ static void Foo.access$200( Foo foo, int value ) {
    foo.doStuff( value );
}

```

内部类会在外部类中任何需要访问`mValue`字段或调用`doStuff`方法的地方调用这些静态方法。这意味着这些代码将直接存取成员变量表现为通过存取器方法访问。之前提到过存取器访问如何比直接访问慢，这例子说明，某些语言约会定导致不可见的性能问题。

如果你在高性能的Hotspot中使用这些代码，可以通过声明被内部类访问的字段和成员为包访问权限，而非私有。但这也意味着这些字段会被其他处于同一个包中的类访问，因此在公共API中不宜采用。

合理利用浮点数

通常的经验是，在Android设备中，浮点数会比整型慢两倍，在缺少FPU和JIT的G1上对比有FPU和JIT的Nexus One中确实如此（两种设备间算术运算的绝对速度差大约是10倍）

从速度方面说，在现代硬件上，`float`和`double`之间没有任何不同。更广泛的讲，`double`大2倍。在台式机上，由于不存在空间问题，`double`的优先级

高于float。

但即使是整型，有的芯片拥有硬件乘法，却缺少除法。这种情况下，整型除法和求模运算是通过软件实现的，就像当你设计Hash表，或是做大量的算术那样。

了解并使用类库

选择Library中的代码而非自己重写，除了通常的那些原因外，考虑到系统空闲时会用汇编代码调用来替代library方法，这可能比JIT中生成的等价的最好的Java代码还要好。典型的例子就是String.indexOf，Dalvik用内部内联来替代。同样的，System.arraycopy方法在有JIT的Nexus One上，自行编码的循环快9倍。

(参见 Effective Java item 47.)

合理利用本地方法

本地方法并不是一定比Java高效。最起码，Java和native之间过渡的关联是有消耗的，而JIT并不能对此进行优化。当你分配本地资源时（本地堆上的内存，文件说明符等），往往很难实时的回收这些资源。同时你也需要在各种结构中编译你的代码（而非依赖JIT）。甚至可能需要针对相同的架构来编译出不同的版本：针对ARM处理器的G1编译的本地代码，并不能充分利用Nexus One上的ARM，而针对Nexus One上ARM编译的本地代码不能在G1的ARM上运行。

当你想部署程序到存在本地代码库的Android平台上时，本地代码才显得尤为有用，而并非为了Java应用程序的提速。

(参见 Effective Java item 54.)

来自 "[index.php?title=Designing_for_Performance&oldid=10300](#)"



JNI Tips

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：朔月&futurexiong

分任务链接地址：<http://developer.android.com/guide/practices/jni.html>

目录

[[隐藏](#)]

[1 JNI Tips-JNI技巧](#)

[2 JavaVM and JNIEnv-JavaVM和JNIEnv](#)

[3 Threads-线程](#)

[4 jclass, jmethodID, and jfieldID](#)

[5 Exceptions-异常处理](#)

[6 Native Libraries-本地库](#)

[7 64-bit Considerations-64-bit 环境注意事项](#)

[8 Unsupported Features/Backwards Compatibility-不支持的特性/向前兼容性](#)

[9 FAQ: Why do I get UnsatisfiedLinkError?-FAQ: 为什么我得到了UnsatisfiedLinkError 错误?](#)

[10 FAQ: Why didn't FindClass find my class?-FAQ: 为什么我的类文件中找不到FindClass?](#)

[11 FAQ: How do I share raw data with native code?-FAQ: 我怎样才能与本地代码共享原始数据?](#)

JNI Tips-JNI技巧

JNI是Java本地接口(Java Native Interface)的简称。它定义了托管代码(用Java编程语言写的)与本地代码(用C/C++写的)交互的一种方式(译者注：这里的托管代码应该理解成受Java运行时环境监管的代码)。它与厂商

无关，支持从动态共享库中加载代码，虽然繁琐但有时是合理有效的。

你应该通读[JNI spec for J2SE 6](#)来获取对JNI是如何工作的以及它有什么可用的功能的一个认知。在你读第一遍的时候可能对JNI的某些方面的理解不会立刻就清晰，所以你会发现一下的章节可能会对你有所帮助。[JNI Programmer's Guide and Specification](#)里有更为详细的资料。

JavaVM and JNIEnv-JavaVM和JNIEnv

JNI定义了2种关键的数据结构，“JavaVM”和“JNIEnv”。它们本质上都是指向函数表的指针的指针。(在C++的版本中，它们被定义成类(译者注:准确来说是C++中的结构体)，类里面包含一个指向函数表的指针，以及与JNI函数一一对应的用来间接访问函数表的成员函数。)JavaVM 提供了“调用接口”的函数，允许你创建和销毁一个JavaVM。理论上每个进程你可以有多个JavaVM，但Android只允许有一个。

JNIEnv提供了大多数的JNI函数。你的本地方法都会接收**JNIEnv**作为第一个参数。

JNIEnv用于本地线程存储。因此，你不能在线程间共享同一个**JNIEnv**。如果一个代码段没有其他方式获取它自身线程的**JNIEnv**，你可以共享**JavaVM**，用**GetEnv**来获取线程的**JNIEnv**。(假设这个线程有一个**JavaVM**;参见下面的 **AttachCurrentThread**。)

C版本的**JNIEnv**和**JavaVM**的声明是异于C++版本的。**"jni.h"**头文件根据被包含在C或是C++文件中来提供不同类型的 **typedefs**。因此在被两种语言包含的头文件中包含**JNIEnv**参数不是明智的选择。(换句话说:如果你的头文件中需要用到**#ifdef __cplusplus**,那么在有涉及到**JNIEnv**的内容的时候你需要做一些额外的工作。)

Threads-线程

所有的线程都是Linux的线程，由内核调度。它们通常由托管代码启动(使用**Thread.start**)，但是也可以在别的地方创建它们，并把它们连接到**JavaVM**上。比如，一个由**pthread_create**方法启动的线程可以用JNI的**AttachCurrentThread**或者 **AttachCurrentThreadAsDaemon**函数来连接。在线程没有被连接到**JavaVM**之前是不会**JNIEnv**的，也无法发

起JNI的调用。

连接一个本地创建的线程会构造一个`java.lang.Thread`的对象，并把这个对象添加到“主”线程组里面，使之对调试者可见。对一个已被连接的线程调用`AttachCurrentThread`不做任何操作。

`Android`不会暂停正在执行本地代码的线程。如果垃圾收集器正在运行，或者是调试者发出了暂停的请求，`Android`会在它发起下一个JNI调用的时候暂停线程。

通过JNI连接的线程在它们退出之前必须调用`DetachCurrentThread`方法。如果直接编写这样的调用代码会显得很笨拙，在`Android 2.0 (Eclair)`及更高的版本，你可以使用`pthread_key_create`来定义一个析构函数，并在析构函数里调用`DetachCurrentThread`方法，析构函数会在线程退出之前被调用。

jclass, jmethodID, and jfieldID

如果你想在本地代码中访问一个对象的字段，你将会做以下事情：

- 取得`FindClass`得到的类的类对象引用
- 取得`GetFieldID`得到的字段的字段ID
- 用合适的方法取得字段的内容，比如`GetIntField`

同样的，要调用一个方法，你必须先得到一个类对象的引用和方法ID。这些ID通常来说只是指向内部的运行时数据结构。找到它们可能需要一些字符串的比较，但一旦你得到它们之后实际的字段取值或者方法调用将会非常的快。

Exceptions-异常处理

You must not call most JNI functions while an exception is pending. Your code is expected to notice the exception (via the function's return value, `ExceptionCheck`, or `ExceptionOccurred`) and return, or clear the exception and handle it.

大多数情况下，当程序发生异常的时候你是无法调用JNI模块的。因为你的

代码不但需要标记出异常出现的位置（可以通过**ExceptionCheck**或者**ExceptionOccurred**的返回值），而且需要对这些所抛出的异常进行处理。

The only JNI functions that you are allowed to call while an exception is pending are:

仅在以下异常被抛出时，你可以调用**JNI**函数：

- DeleteGlobalRef
- DeleteLocalRef
- DeleteWeakGlobalRef
- ExceptionCheck
- ExceptionClear
- ExceptionDescribe
- ExceptionOccurred
- MonitorExit
- PopLocalFrame
- PushLocalFrame
- Release<PrimitiveType>ArrayElements
- ReleasePrimitiveArrayCritical
- ReleaseStringChars
- ReleaseStringCritical
- ReleaseStringUTFChars

Many JNI calls can throw an exception, but often provide a simpler way of checking for failure. For example, if **NewString** returns a non-NULL value, you don't need to check for an exception. However, if you call a method (using a function like **CallObjectMethod**), you must always check for an exception, because the return value is not going to be valid if an exception was thrown.

多数**JNI**在被调用时能够抛出一个异常，但通常还有一种更简单的方式来检验该调用是否出现失效。比如说，如果调用**NewString**方法返回一个非**null**值，那么你就不需要再对其进行异常检查。但如果你调用的是一个类似**CallObjectMethod**这样的函数时，你还是必须做异常的检查处理，这是因为当出现异常时，调用该模块并不会返回一个有效值。

Note that exceptions thrown by interpreted code do not unwind native stack frames, and Android does not yet support C++ exceptions. The JNI Throw and ThrowNew instructions just set an exception pointer in the current thread. Upon returning to managed from native code, the exception will be noted and handled appropriately.

需要注意的是，由翻译码抛出的异常并不会释放堆栈帧空间，且目前的Android系统也不支持C++异常处理。所以JNI的Throw和 ThrowNew指令仅仅只是在当前的线程中设置了一个异常处理指针，而当从本地代码返回托管代码时，该异常才能被标注并得到有效的处理。

Native code can "catch" an exception by calling ExceptionCheck or ExceptionOccurred, and clear it with ExceptionClear. As usual, discarding exceptions without handling them can lead to problems.

当然，本地代码可以通过调用ExceptionCheck或者 ExceptionOccurred函数来捕获相应的异常，也能通过调用ExceptionClear函数来清理该异常。同样，如果对抛出的异常不作处理也会造成问题。

There are no built-in functions for manipulating the Throwable object itself, so if you want to (say) get the exception string you will need to find the Throwable class, look up the method ID for getMessage "()Ljava/lang/String;", invoke it, and if the result is non-NULL use GetStringUTFChars to get something you can hand to printf(3) or equivalent.

由于对Throwable对象的操作不支持内置函数，所以如果你需要获取相应的异常字符串，你首先需要找到Throwable类，然后找到 getMessage "()Ljava/lang/String;"方法的ID，调用它，假如调用后返回的结果是非null值，则需要通过GetStringUTFChars 方法来把获取的结果传给printf(3)或者是采用其他类似的方法完成这个工作。

JNI does very little error checking. Errors usually result in a crash. Android also offers a mode called CheckJNI, where the JavaVM and JNIEnv function table pointers are switched to tables of functions that perform an extended series of checks before calling the standard implementation.

JNI很少对错误进行检查。而错误常会导致程序崩溃。Android为此提供了一种CheckJNI处理模式，当JAVA虚拟机以及JNIEnv函数表指针被切换到函数列表时，该模式会在调用标准实现之前做出一系列的附加检查措施。

The additional checks include:

附加的检查包括：

- Arrays: attempting to allocate a negative-sized array.
• 数组：尝试分配一个长度大小为负数的数组。
- Bad pointers: passing a bad jarray/jclass/jobject/jstring to a JNI call, or passing a NULL pointer to a JNI call with a non-nullable argument.
• 坏指针：将一个错误的jarray/jclass/jobject/jstring传递给JNI调用，或者是带一个非空参数将空指针传递给JNI调用。
- Class names: passing anything but the "java/lang/String" style of class name to a JNI call.
• 类名称：JNI调用时未采用类似于"java/lang/String"这种格式的类名。
- Critical calls: making a JNI call between a "critical" get and its corresponding release.
• 临界调用：在“临界区”获取或者释放的时候进行JNI调用操作。
- Direct ByteBuffers: passing bad arguments to NewDirectByteBuffer.
• 直接ByteBuffers：将错误的参数传递给NewDirectByteBuffer函数。
- Exceptions: making a JNI call while there's an exception pending.
• 异常：当发生异常时进行JNI调用。
- JNIEnv*s: using a JNIEnv* from the wrong thread.
• JNIEnv*s: 在错误的线程中使用JNIEnv指针。
- jfieldIDs: using a NULL jfieldID, or using a jfieldID to set a field to a value of the wrong type (trying to assign a StringBuilder to a String field, say), or using a jfieldID for a static field to set an instance field or vice versa, or using a jfieldID from one class with instances of another class.
• jfieldIDs：使用空的jfieldID，或是通过jfieldID设置变量值时指定了错误的类型（比如说尝试将一个 String型变量设置为StringBuilder类型），又或者是通过jfieldID将一个静态变量设置为实例变量，反之亦然，还有可能是，通过 jfieldID将一个类的对象指定为另外一个类的对象。

- **jmethodIDs:** using the wrong kind of jmethodID when making a Call*Method JNI call: incorrect return type, static/non-static mismatch, wrong type for 'this' (for non-static calls) or wrong class (for static calls).
- **jmethodIDs:** 运用Call*Method的JNI调用时，jmethodID方法运用错误：返回错误的类型值，静态/非静态匹配错误，'this'类型错误（当使用非静态调用时）以及错误的类定义（使用静态调用时）。
- **References:** using DeleteGlobalRef/DeleteLocalRef on the wrong kind of reference.
- **引用：**调用DeleteGlobalRef/DeleteLocalRef函数时使用了错误的引用。
- **Release modes:** passing a bad release mode to a release call (something other than 0, JNI_ABORT, or JNI_COMMIT).
- **Release模式：**进行release调用时使用了错误的release模式（即选择0, JNI_ABORT 或者JNI_COMMIT以外的值）
- **Type safety:** returning an incompatible type from your native method (returning a StringBuilder from a method declared to return a String, say).
- **类型安全：**从你的本地方法中返回了一个不匹配的类型值（比如从一个声明为String作为返回值的方法中返回一个StringBuilder类型的返回值）
- **UTF-8:** passing an invalid Modified UTF-8 byte sequence to a JNI call.
- **UTF-8编码：**将一个无效的Modified UTF-8字节串传递给JNI调用。

(Accessibility of methods and fields is still not checked: access restrictions don't apply to native code.) (可访问的方法和变量在此并没有被检查：这是因为访问限制并不适用于本地代码)

There are several ways to enable CheckJNI. 下列方法可以用来使能CheckJNI模式。

If you're using the emulator, CheckJNI is on by default. 如果你用的是模拟器，那么CheckJNI模式默认已经被开启。

If you have a rooted device, you can use the following sequence of commands to restart the runtime with CheckJNI enabled: 如果你的设备已经获取root权限，可以通过以下的命令行来重新启动运行设备，使能CheckJNI模式：

```
adb shell stop
adb shell setprop dalvik.vm.checkjni true
adb shell start
```

In either of these cases, you'll see something like this in your logcat output when the runtime starts: 然后，你可以通过查看设备启动运行阶段时打印的logcat信息来确认是否开启该模式。

```
D AndroidRuntime: CheckJNI is ON
```

If you have a regular device, you can use the following command: 如果你手里的机器只是普通发行版本，你需要使用以下的命令行来开启模式：

```
adb shell setprop debug.checkjni 1
```

This won't affect already-running apps, but any app launched from that point on will have CheckJNI enabled. (Change the property to any other value or simply rebooting will disable CheckJNI again.) In this case, you'll see something like this in your logcat output the next time an app starts:

该方法不会对已经运行的应用软件起作用，只能使之后启动的应用软件带有CheckJNI模式。（如果将property参数值作任意修改或者重启系统都就能重新禁用CheckJNI模式）。如果你采用这个方法，那么当任何一个应用软件启动时，你都会在logcat的输出窗口看到如下的信息：

```
D Late-enabling CheckJNI
```

Native Libraries-本地库

You can load native code from shared libraries with the standard System.loadLibrary call. The preferred way to get at your native code is: 你可以通过标准的System.loadLibrary调用从共享库中载入本地代码。但你还有更好的渠道来获取本地代码，方法如下：

- Call System.loadLibrary from a static class initializer. (See the earlier example, where one is used to call nativeClassInit.) The argument is the "undecorated" library name, so to load "libfubar.so" you would pass in "fubar".
- 在静态类的初始化时调用System.loadLibrary。（可以参考之前的例子，比如调用nativeClassInit函数时）。将“原始”的库名作为参数，你将其传给“fubar”来载入“libfubar.so”动态链接库。

- Provide a native function: `jint JNI_OnLoad(JavaVM* vm, void* reserved)`
- 提供一个本地函数: `int JNI_OnLoad(JavaVM* vm, void* reserved)` 来实现。
- In `JNI_OnLoad`, register all of your native methods. You should declare the methods "static" so the names don't take up space in the symbol table on the device.
- 在`JNI_OnLoad`函数中注册你所有的本地方法。你可以将所有的方法都声明为“static”，这样就不会占用设备中符号列表上的空间。

The `JNI_OnLoad` function should look something like this if written in C++: 在C++中你可以参考如下方式来构造`JNI_OnLoad`函数：

```

jint JNI_OnLoad (JavaVM* vm, void* reserved)
{
    JNIEnv* env;
    if (vm->GetEnv (reinterpret_cast<void**> (&env),  

JNICALL _VERSION_1_6) != JNI_OK) {
        return -1;
    }

    // Get jclass with env->FindClass.
    // Register methods with env->RegisterNatives.

    return JNICALL _VERSION_1_6;
}

```

You can also call `System.load` with the full path name of the shared library. For Android apps, you may find it useful to get the full path to the application's private data storage area from the context object.

你还可以通过共享库的完整路径来调用`System.load`。在Android的应用软件中，你会发现，在对象的上下文中，通过获取完整路径来共享应用的私有数据存储空间是非常有用的方法。

This is the recommended approach, but not the only approach. Explicit registration is not required, nor is it necessary that you provide a `JNI_OnLoad` function. You can instead use "discovery" of native methods that are named in a specific way (see [the JNI spec](#) for details), though this is less desirable because if a method signature is wrong you won't know about it until the first time the method is actually used.

以上是推荐的方法，但并不是唯一方法。例如，明文注册并不是必要的环节同样，`JNI_OnLoad`函数的设置也不是。你可以通过对本地代码做特殊命名（参考the JNI spec这一章的细节）来手动“寻找”它们，但是这个方法并不是那么让人满意，因为如果你在某个方法的标记上如果出了错，除非第一时间这个方法就被调用，否则你并将不会意识到它有问题。

One other note about `JNI_OnLoad`: any `FindClass` calls you make from there will happen in the context of the class loader that was used to load the shared library. Normally `FindClass` uses the loader associated with the method at the top of the interpreted stack, or if there isn't one (because the thread was just attached) it uses the "system" class loader. This makes `JNI_OnLoad` a convenient place to look up and cache class object references.

关于`JNI_OnLoad`的其他备注：你在上文中所调用的`FindClass`函数都可以在共享库的类装载器中得到。通常来说，`FindClass`会使用相关方法的装载器来分析栈顶数据，但如果 没有这样的装载器（有可能是已经附着到线程上了），那么它也会使用“系统”的类装载器。这样做使得`JNI_OnLoad`很容易被找到，而且也能缓存类对象的引用。

64-bit Considerations-64-bit 环境注意事项

Android is currently expected to run on 32-bit platforms. In theory it could be built for a 64-bit system, but that is not a goal at this time. For the most part this isn't something that you will need to worry about when interacting with native code, but it becomes significant if you plan to store pointers to native structures in integer fields in an object. To support architectures that use 64-bit pointers, you need to stash your native pointers in a long field rather than an int.

Android当前多运行于32位平台环境中。理论上来说，它同样也能编译成64位版本，但这并不是最终的解决方案。一般来说，当你使用本地代码的时候，你并不需要担心64位的版本问题。但是，假如你打算在某个对象中的integer变量的本地结构体中存储指针，那么64位版本就会变成为很大的麻烦。因此，为了能够在64位架构下使用指针，你必须将你的本地指针存储为long形而非int型。

Unsupported Features/Backwards Compatibility-不支持的特性/向前兼容性

All JNI 1.6 features are supported, with the following exception: 除了以下情况外，所有的1.6版本的JNI的特性都被支持：

- **DefineClass** is not implemented. Android does not use Java bytecodes or class files, so passing in binary class data doesn't work.
- **DefineClass**方法还没有被实现。Android当前没有使用JAVA比特编码或者类，所以是无法传递给它们二进制类文件的。

For backward compatibility with older Android releases, you may need to be aware of: 对早期Android发布版本的向前兼容性，你需要注意以下几点：

- Dynamic lookup of native functions
- 动态查找本地函数

Until Android 2.0 (Eclair), the '\$' character was not properly converted to "_00024" during searches for method names. Working around this requires using explicit registration or moving the native methods out of inner classes.

到Android2.0 (Eclair) 版本为止，在搜索方法名称时，'\$'字符还不能够被准确转换为"_00024"。为了解决这个问题，你可以使用明文注册或者是将本地方法从内部类中移出。

- Detaching threads
- 分离线程

Until Android 2.0 (Eclair), it was not possible to use a `pthread_key_create` destructor function to avoid the "thread must be detached before exit" check. (The runtime also uses a `pthread key destructor` function, so it'd be a race to see which gets called first.) 到Android2.0 (Éclair) 版本为止，利用`pthread_key_create`析构函数来免除“线程在退出时先断开”的检查是无法做到的（运行时同样会用到一个`pthread`关键析构函数，所以看谁能更快的取得这个函数，这是一个比赛）。

- Weak global references
- 弱全局引用

Until Android 2.2 (Froyo), weak global references were not implemented. Older versions will vigorously reject attempts to use them. You can use the Android platform version constants to test for support.

到Android2.2 (Froyo) 版本为止，弱全局引用还不能实现。旧版本不遗余力的拒绝使用该功能。你可以用android的平台版本常数来测试是否支持这个功能。

Until Android 4.0 (Ice Cream Sandwich), weak global references could only be passed to NewLocalRef, NewGlobalRef, and DeleteWeakGlobalRef. (The spec strongly encourages programmers to create hard references to weak globals before doing anything with them, so this should not be at all limiting.)

而到Android4.0(Ice Cream Sandwich)版本为止，弱全局引用也仅能在NewLocalRef, NewGlobalRef, 以及DeleteWeakGlobalRef几个函数中实现（规范中鼓励程序员在引用操作之前创造强（hard）引用，从而代替弱全局引用，因而弱全局引用并未被完全限制使用）

From Android 4.0 (Ice Cream Sandwich) on, weak global references can be used like any other JNI references.

但从Android4.0(Ice Cream Sandwich)版本起，弱全局引用可以在除了JNI引用之外的所有地方使用了。

- Local references
- 局部引用

Until Android 4.0 (Ice Cream Sandwich), local references were actually direct pointers. Ice Cream Sandwich added the indirection necessary to support better garbage collectors, but this means that lots of JNI bugs are undetectable on older releases. See JNI Local Reference Changes in ICS for more details.

到Android 4.0 (Ice Cream Sandwich)版本为止，所谓的本地引用都是指直接指针。Ice Cream Sandwich版本为了更好进行垃圾回收，添加间接指针机制，但是这样一来也就意味着大量的JNI 的BUG在旧版本中无法被发现了。具体的细节可以参考JNI Local Reference Changes in ICS一节。

- Determining reference type with GetObjectRefType
- 通过GetObjectRefType确认引用类型

Until Android 4.0 (Ice Cream Sandwich), as a consequence of the use of direct pointers (see above), it was impossible to implement GetObjectRefType correctly. Instead we used a heuristic that looked through the weak globals table, the arguments, the locals table, and the globals table in that order. The first time it found your direct pointer, it would report that your reference was of the type it happened to be examining. This meant, for example, that if you called GetObjectRefType on a global jclass that happened to be the same as the jclass passed as an implicit argument to your static native method, you'd get `JNILocalRefType` rather than `JNIGlobalRefType`.

到Android 4.0 (Ice Cream Sandwich)版本为止，由于一直使用直接指针的缘故（见上），导致系统无法实现GetObjectRefType方法。我们只能探索性的使用弱全局列表，参数，本地列表以及全局列表。首先它会找到你的直接指针，然后报告你的引用刚好是可以被验证的类型。这也就是说，如果你在一个全局jclass中调用GetObjectRefType函数，那么与jclass将一个隐形参数传给静态本地方法的效果是一样的，此时你获得的其实是 `JNILocalRefType`的值，而非 `JNIGlobalRefType`的。

FAQ: Why do I get UnsatisfiedLinkError?-FAQ: 为什么我得到了UnsatisfiedLinkError 错误？

When working on native code it's not uncommon to see a failure like this: 在本地代码环境下，出现如下的错误是很常见的：

```
java.lang.UnsatisfiedLinkError: Library foo not found
```

In some cases it means what it says — the library wasn't found. In other cases the library exists but couldn't be opened by `dlopen(3)`, and the details of the failure can be found in the exception's detail message. 有时候，这个问题的原因就像它的字面意思一样——这个库找不到了。而还有些情况是这个库文件存在，但是你无法通过`dlopen(3)`函数打开它，关于该错误问题的细节你可以在异常的详细消息中进行查询。

Common reasons why you might encounter "library not found" exceptions: 对于你遇到的异常"library not found"常见的原因是：

- The library doesn't exist or isn't accessible to the app. Use adb shell `ls -l <path>` to check its presence and permissions.
该库文件不存在，或者是无法被应用所访问。使用adb shell `ls -l <path>`命令来检查它的状态以及访问权限。
- The library wasn't built with the NDK. This can result in dependencies on functions or libraries that don't exist on the device.
该库文件还没有被NDK编译出来。这会导致依赖函数或者库文件在设备上不存在。

Another class of `UnsatisfiedLinkError` failures looks like: 另外一类错误`UnsatisfiedLinkError`表示如下：

```
java.lang.UnsatisfiedLinkError: myfunc
  at Foo.myfunc(Native Method)
  at Foo.main(Foo.java:10)
```

In logcat, you'll see: 在logcat中，你可以看到：

```
W/dalvikvm( 880): No implementation found for native LFoo;.myfunc()
()V
```

This means that the runtime tried to find a matching method but was unsuccessful. Some common reasons for this are: 这是指运行时尝试去寻找一个匹配的方法但是失败了。这个问题的原因有可能是：

- The library isn't getting loaded. Check the logcat output for messages about library loading.
该库文件无法被载入。请检查库文件载入过程中的logcat输出消息。

- The method isn't being found due to a name or signature mismatch.
This is commonly caused by:
• 由于方法的名称或者是方法的签名问题，导致该方法无法被找到。这种情况经常是以下问题导致的：
 - For lazy method lookup, failing to declare C++ functions with `extern "C"` and appropriate visibility (`JNIEXPORT`). Note that prior to Ice Cream Sandwich, the `JNIEXPORT` macro was incorrect, so using a new GCC with an old `jni.h` won't work. You can use `arm-eabi-nm` to see the symbols as they appear in the library; if they look mangled (something like `_Z15Java_Foo_myfuncP7_JNIEnvP7_jclass` rather than `Java_Foo_myfunc`), or if the symbol type is a lowercase 't' rather than an uppercase 'T', then you need to adjust the declaration.
 - 对于惰性寻找方法，未能在C++函数前加入"C"的声明以及设置合适的访问权限会导致这个问题（`JNIEXPORT`宏）。注意在早于Ice Cream Sandwich的版本中，宏`JNIEXPORT`是不存在的，因此如果在新版本的GCC编译器中编译老版本的`jni.h`文件会失败。你可以通过`arm-eabi-nm`来查看库文件中是否存在上述文件符；如果这些文件符看起来混乱不堪（比如说类似`_Z15Java_Foo_myfuncP7_JNIEnvP7_jclass`，而不是标准的`Java_Foo_myfunc`），或者，如果这些文件符的类型（type）都用小写字母't'来替代大写字母'T'，那么你需要调整声明部分来解决这个问题。
 - For explicit registration, minor errors when entering the method signature. Make sure that what you're passing to the registration call matches the signature in the log file. Remember that 'B' is byte and 'Z' is boolean. Class name components in signatures start with 'L', end with ';', use '/' to separate package/class names, and use '\$' to separate inner-class names (`Ljava/util/Map$Entry;`, say).
 - 在明文注册中，在调用方法签名环节出现级别为minor的错误。请确认你传递给注册函数的签名与log文件中的需要一致。切记'B'指的是比特，而'Z'指的是布尔值。签名中的类文件名需要以'L'开头，以';'结尾，用'/'来分格包和类名，使用'\$'来声明一个内部类名（比如说`Ljava/util/Map$Entry`）。

Using `javah` to automatically generate JNI headers may help avoid some problems. 如果使用javah工具来自动生成JNI头文件，能避免很多麻烦。

FAQ: Why didn't `FindClass` find my class? -FAQ: 为什么我的类文件中找不到`FindClass`?

Make sure that the class name string has the correct format. JNI class names start with the package name and are separated with slashes, such as `java/lang/String`. If you're looking up an array class, you need to start with the appropriate number of square brackets and must also wrap the class with 'L' and ';', so a one-dimensional array of String would be `[Ljava/lang/String;`.

请确认你的类文件名称格式是否正确。JNI的类名的格式是从所在包的包名开始，以分隔符区分，比如`java/lang/String`。如果你需要查找一个数组类，你得以适当数目的方括号（[）作为开始，然后以'L'开头，';'结尾作为文件名称，以一个一位数组为例，它的名称应该是 `[Ljava/lang/String;`。

If the class name looks right, you could be running into a class loader issue. `FindClass` wants to start the class search in the class loader associated with your code. It examines the call stack, which will look something like:

如果类的名称看起来没有设置错，可能问题出在类装载器上。`FindClass`方法可以在你的代码的类装载器中找到该类。它以如下方式来检查调用堆栈：

```
Foo.myfunc(Native Method)
Foo.main(Foo.java:10)
dalvik.system.NativeStart.main(Native Method)
```

The topmost method is `Foo.myfunc`. `FindClass` finds the `ClassLoader` object associated with the `Foo` class and uses that.

排在首位的方法是`Foo.myfunc`。`FindClass`方法找到`Foo`类的类装载器并使用它。

This usually does what you want. You can get into trouble if you create a

thread yourself (perhaps by calling `pthread_create` and then attaching it with `AttachCurrentThread`). Now the stack trace looks like this:

这样做可以达到你的目的。但如果创建一个线程你就会陷入困境之中（可能是通过调用`pthread_create`方法，然后将其引入`AttachCurrentThread`方法）。此时，堆栈跟踪表显示如下：

```
dalvik.system.NativeStart.run(Native Method)
```

The topmost method is `NativeStart.run`, which isn't part of your application. If you call `FindClass` from this thread, the JavaVM will start in the "system" class loader instead of the one associated with your application, so attempts to find app-specific classes will fail.

排在首位的方法是`NativeStart.run`，但该方法并不在你的应用中。如果你在本线程中调用`FindClass`方法，JAVA虚拟机将会在"system"类装载器里，而不是从你的应用中启动，所以此时你去从应用中去寻找特定类是找不到的。

There are a few ways to work around this:

以下的方法可以解决这个问题：

- Do your `FindClass` lookups once, in `JNI_OnLoad`, and cache the class references for later use. Any `FindClass` calls made as part of executing `JNI_OnLoad` will use the class loader associated with the function that called `System.loadLibrary` (this is a special rule, provided to make library initialization more convenient). If your app code is loading the library, `FindClass` will use the correct class loader.
- 你是否在`JNI_OnLoad`函数中仅调用了一次`FindClass`，并且为了后续的使用将类引用缓存。任何一个`FindClass`调用都可以被视为执行`JNI_OnLoad`函数的一部分，这样做会使用`System.loadLibrary`库的类装载器的功能（这是一个特别的功能，专门用于初始化库文件，使其更加方便）。如果你的应用调用了库文件，那么`FindClass`会使用正确的类装载器。
- Pass an instance of the class into the functions that need it, by declaring your native method to take a `Class` argument and then passing `Foo.class` in.

- 通过声明你的本地方法，将一个类的实例传递给需要它的函数中，然后可以将类的参数传递给Foo.class。
- Cache a reference to the ClassLoader object somewhere handy, and issue loadClass calls directly. This requires some effort.
- 将引用缓存至ClassLoader的对象是很方便的，而且可以直接调用loadClass方法。当然，这需要一定的工作量。

FAQ: How do I share raw data with native code?-FAQ: 我怎样才能与本地代码共享原始数据？

You may find yourself in a situation where you need to access a large buffer of raw data from both managed and native code. Common examples include manipulation of bitmaps or sound samples. There are two basic approaches.

当需要在本地代码和托管代码间直接传递大量的原始数据时，你会陷入麻烦中。常见的情况包括对位图或者是声音样本的操作。这有两种基本方法来处理上述问题：

You can store the data in a byte[]. This allows very fast access from managed code. On the native side, however, you're not guaranteed to be able to access the data without having to copy it. In some implementations, GetByteArrayElements and GetPrimitiveArrayCritical will return actual pointers to the raw data in the managed heap, but in others it will allocate a buffer on the native heap and copy the data over.

你可以通过byte[]来存储数据。这样做可以从托管代码中快速访问数据段。而在本地代码侧，你不能保证如果没有复制代码还能够成功的访问它。在某些实现中，可以通过GetByteArrayElements以及GetPrimitiveArrayCritical函数返回原始数据在托管堆中的指针，另一些情况下，也可以在本地堆中分配一段缓冲区用来复制数据。

The alternative is to store the data in a direct byte buffer. These can be created with java.nio.ByteBuffer.allocateDirect, or the JNI NewDirectByteBuffer function. Unlike regular byte buffers, the storage is not allocated on the managed heap, and can always be accessed directly from native code (get the address with GetDirectBufferAddress).

Depending on how direct byte buffer access is implemented, accessing the data from managed code can be very slow.

还有一个方法，将数据存储至直接比特缓冲区中。缓冲区可以通过`java.nio.ByteBuffer.allocateDirect`或者是JNI `NewDirectByteBuffer`函数来创建。不同于一般的比特缓冲区，该缓冲区并不会由托管堆来分配地址，且可以由本地代码直接访问（通过`GetDirectBufferAddress`函数就可以直接获得访问地址）。但考虑直接比特缓存访问的实现方式，从托管代码获取数据的过程可能会非常缓慢。

The choice of which to use depends on two factors:

下面两个因素决定你怎么选择合适的方法：

1. Will most of the data accesses happen from code written in Java or in C/C++?

1. 数据访问部分的代码是由JAVA还是C/C++写的？

2. If the data is eventually being passed to a system API, what form must it be in? (For example, if the data is eventually passed to a function that takes a `byte[]`, doing processing in a direct ByteBuffer might be unwise.)

2. 如果数据最终需要传递给系统的API，它需要被包装为什么格式（举个例子，如果数据最终需要以`byte[]`的格式传给某个函数，将它包装为`ByteBuffer`就不合适了）？

来自 "[index.php?title=JNI_Tips&oldid=11936](#)"



Designing for Responsiveness

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

- [1 Designing for Responsiveness-设计响应](#)
 - [1.1 What Triggers ANR?-是什么原因引发的ANR?](#)
 - [1.2 How to Avoid ANR-如何避免ANR](#)
 - [1.3 Reinforcing Responsiveness-加强响应](#)

Designing for Responsiveness-设计响应

It's possible to write code that wins every performance test in the world, but still sends users in a fiery rage when they try to use the application. These are the applications that aren't responsive enough — the ones that feel sluggish, hang or freeze for significant periods, or take too long to process input.

就算编写的代码可以通过所有的性能测试。但是当用户不断使用软件时，还是仍然可能给用户带来困惑和不满。这些就是应用响应不能做到足够快的结果，如反应迟钝、系统长时间挂起或冻结或是很久不能响应输入。

In Android, the system guards against applications that are insufficiently responsive for a period of time by displaying a dialog to the user, called the Application Not Responding (ANR) dialog, shown at right in Figure 1. The user can choose to let the application continue, but the user won't appreciate having to act on this dialog every time he or she uses your application. It's critical to design responsiveness into your application, so

that the system never has cause to display an ANR dialog to the user.
用Android的话来说，不能够及时响应的应用经常使系统弹出令人恐惧的“应用没有响应”(ANR)的消息，如右图1。用户可以选择让程序继续运行，但是用户每次运行你的程序都需要如此操作，这使得你的设计产品将不再得到用户的关注，所以用户从来不会选择会报ANR错误的程序。

Generally, the system displays an ANR if an application cannot respond to user input. For example, if an application blocks on some I/O operation (frequently a network access), then the main application thread won't be able to process incoming user input events. After a time, the system concludes that the application is frozen, and displays the ANR to give the user the option to kill it.

通常，这种情况出现在应用不能及时响应用户输入上。例如，如果你的应用阻塞在I/O操作（常常是网络访问的情况）上时，主应用线程就无法处理已经到达的用户输入事件。一定的时间过后，系统可能认为你的程序已经挂起，于是就发送消息允许用户终止该程序。同时，如果你在内存中创建一个复杂的数据结构花费太多时间，或是在游戏中可能正在计算下一副场景时，同样系统也可能认为你的程序已经挂起。这就是为什么运用上述技术进行有效率的运算总是如此重要，即使是最高效的代码仍然需要时间去执行。

Similarly, if your application spends too much time building an elaborate in-memory structure, or perhaps computing the next move in a game, the system will conclude that your application has hung. It's always important to make sure these computations are efficient using the techniques above, but even the most efficient code still takes time to run.

同时，如果你为内存中创建一个复杂的数据结构花费太多的时间，或是在游戏中可能正在计算下一副场景花费大量的时间，系统同样可以认为你的程序已经挂起，这就是为什么运用上述技术进行有效的运算总是如此重要，即使最高效的代码也仍然需要时间去执行。

In both of these cases, the recommended approach is to create a child thread and do most of your work there. This keeps the main thread (which drives the user interface event loop) running and prevents the system from concluding that your code has frozen. Since such threading usually is accomplished at the class level, you can think of responsiveness

as a class problem. (Compare this with basic performance, which was described above as a method-level concern.)

针对上述两类情况，解决方案通常就是创建一个子线程去执行这里的大多数工作。这样就可使主线程（响应用户接口的事件循环）能正常运行并避免系统误认为程序已经冻结。由于这些线程通常由类级别的代码实现，你可以想到程序响应不太好与类问题有关（在基本性能与前述方法级相对比而言。）。

This document describes how the Android system determines whether an application is not responding and provides guidelines for ensuring that your application stays responsive.

本文档描述了应用程序受Android系统的响应的原因，并提供指导方针来确保你的应用在此情况下，仍然可以做出响应。



What Triggers ANR?-是什么原因引发的ANR?

In Android, application responsiveness is monitored by the Activity Manager and Window Manager system services. Android will display the ANR dialog for a particular application when it detects one of the following conditions:

在Android应用程序的响应监测活动管理和窗口管理系統服务中，Android将会显示一个特定的应用程序ANR对话框，它的检测条件如下：

No response to an input event (e.g. key press, screen touch) within 5 seconds

A [BroadcastReceiver](#) hasn't finished executing within 10 seconds

--5秒内没有响应输入事件（如按键，触摸屏幕等）

--[BroadcastReceiver](#)的最长执行时间超过10秒

How to Avoid ANR-如何避免ANR

Given the above definition for ANR, let's examine why this can occur in

Android applications and how best to structure your application to avoid ANR.

考虑上面的ANR定义，让我们来研究一下为什么它会在Android应用程序里发生和如何最佳构建应用程序来避免ANR。

Android applications normally run entirely on a single (i.e. main) thread. This means that anything your application is doing in the main thread that takes a long time to complete can trigger the ANR dialog because your application is not giving itself a chance to handle the input event or Intent broadcast.

Android应用程序通常是运行在一个单独的线程（例如，main）里。这意味着你的应用程序所做的事情如果在主线程里占用了太长的时间的话，就会引发ANR对话框，因为你的应用程序并没有给自己机会来处理输入事件或者Intent广播。

Therefore any method that runs in the main thread should do as little work as possible. In particular, Activities should do as little as possible to set up in key life-cycle methods such as onCreate() and onResume(). Potentially long running operations such as network or database operations, or computationally expensive calculations such as resizing bitmaps should be done in a child thread (or in the case of databases operations, via an asynchronous request). However, this does not mean that your main thread should block while waiting for the child thread to complete — nor should you call Thread.wait() or Thread.sleep(). Instead of blocking while waiting for a child thread to complete, your main thread should provide a [Handler](#) for child threads to post back to upon completion. Designing your application in this way will allow your main thread to remain responsive to input and thus avoid ANR dialogs caused by the 5 second input event timeout. These same practices should be followed for any other threads that display UI, as they are also subject to the same timeouts.

因此，运行在主线程里的任何方法都尽可能少做事情。特别是，Activity应该在它的关键生命周期方法（如onCreate()和onResume()）里尽可能少的去做创建操作。潜在的耗时操作，例如网络或数据库操作，或者高耗时的计算如改变位图尺寸，应该在子线程里（或者以数据库操作为例，通过异步请求的方式）来完成。然而，不是说你的主线程阻塞在那里等待子线程

的完成——也不是调用Thread.wait()或是Thread.sleep()。替代的方法是，主线程应该为子线程提供一个Handler，以便完成时能够提交给主线程。以这种方式设计你的应用程序，将能保证你的主线程保持对输入的响应性并能避免由于5秒输入事件的超时引发的ANR对话框。这种做法应该在其它显示UI的线程里效仿，因为它们都受相同的超时影响。

You can use [StrictMode](#) to help find potentially long running operations such as network or database operations that you might accidentally be doing your main thread.

您可以使用StrictMode来帮助找到潜在的长时间运行的操作,比如网络或数据库操作,但是,你可能不小心做掉你的主线程。

The specific constraint on IntentReceiver execution time emphasizes what they were meant to do: small, discrete amounts of work in the background such as saving a setting or registering a Notification. So as with other methods called in the main thread, applications should avoid potentially long-running operations or calculations in BroadcastReceivers. But instead of doing intensive tasks via child threads (as the life of a BroadcastReceiver is short), your application should start a [Service](#) if a potentially long running action needs to be taken in response to an Intent broadcast. As a side note, you should also avoid starting an Activity from an Intent Receiver, as it will spawn a new screen that will steal focus from whatever application the user is currently has running. If your application has something to show the user in response to an Intent broadcast, it should do so using the [Notification Manager](#).

IntentReceiver执行时间的特殊限制意味着它应该做：在后台里做小的、琐碎的工作如保存设定或者注册一个Notification。和在主线程里调用的其它方法一样，应用程序应该避免在BroadcastReceiver里做耗时的操作或计算。但不再是在子线程里做这些任务（因为BroadcastReceiver的生命周期短），替代的是，如果响应Intent广播需要执行一个耗时的动作的话，应用程序应该启动一个Service。顺便提及一句，你也应该避免在Intent Receiver里启动一个Activity，因为它会创建一个新的画面，并从当前用户正在运行的程序上抢夺焦点。如果你的应用程序在响应Intent广播时需要向用户展示什么，你应该使用[Notification Manager](#)来实现。

Reinforcing Responsiveness-加强响应

Generally, 100 to 200ms is the threshold beyond which users will perceive lag (or lack of "snappiness," if you will) in an application. As such, here are some additional tips beyond what you should do to avoid ANR that will help make your application seem responsive to users.

一般来说，在应用程序里，100到200ms是用户能感知阻滞的时间阈值。因此，这里有一些额外的技巧来避免ANR，并有助于让你的应用程序看起来有响应性。

If your application is doing work in the background in response to user input, show that progress is being made (ProgressBar and ProgressDialog are useful for this).

如果你的应用程序为响应用户输入正在后台工作的话，可以显示工作的进度（[ProgressBar](#)和[ProgressDialog](#)对这种情况来说很有用）。

For games specifically, do calculations for moves in a child thread.
特别是游戏，在子线程里做移动的计算。

If your application has a time-consuming initial setup phase, consider showing a splash screen or rendering the main view as quickly as possible and filling in the information asynchronously. In either case, you should indicate somehow that progress is being made, lest the user perceive that the application is frozen.

如果你的应用程序有一个耗时的初始化过程的话，考虑可以显示一个Splash Screen或者快速显示主画面并异步来填充这些信息。在这两种情况下，你都应该显示正在进行的进度，以免用户认为应用程序被冻结了。

来自“[index.php?title=Designing_for_Responsiveness&oldid=6918](#)”

Designing for Seamlessness

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

目录

[[隐藏](#)]

[1 Designing for Seamlessness-无缝设计](#)

- [1.1 Don't Drop Data-别丢弃数据](#)
- [1.2 Don't Expose Raw Data-不要暴露原始数据](#)
- [1.3 Don't Interrupt the User-不要打断用户](#)
- [1.4 Got a Lot to Do? Do it in a Thread-有太多事情要做? 在线程里做](#)
- [1.5 Don't Overload a Single Activity Screen-不要让一个Activity超负荷](#)
- [1.6 Extend System Themes-扩展系统主题](#)
- [1.7 Design Your UI to Work with Multiple Screen Resolutions-设计你的UI可以应对多屏幕分辨率](#)
- [1.8 Assume the Network is Slow-假设网络很慢](#)
- [1.9 Don't Assume Touchscreen or Keyboard-不要假定触摸屏或键盘](#)
- [1.10 Do Conserve the Device Battery-节省设备电池](#)

Designing for Seamlessness-无缝设计

Even if your application is fast and responsive, certain design decisions can still cause problems for users — because of unplanned interactions with other applications or dialogs, inadvertent loss of data, unintended blocking, and so on. To avoid these problems, it helps to understand the context in which your applications run and the system interactions that can affect your application. In short, you should strive to develop an

application that interacts seamlessly with the system and with other applications.

即使你的应用程序是快速且响应灵敏的，但一些设计仍然会给用户造成问题——与其它应用程序或对话框未事先计划的交互，意外的数据丢失，意料之外的阻塞等等。避免这些问题，有助于理解应用程序运行的上下文和系统的交互过程，而这些又正影响着你的应用程序。简而言之，你应该竭尽全力去开发一个与系统和其它应用程序流畅交互的应用程序。

A common seamlessness problem is when an application's background process — for example, a service or broadcast receiver — pops up a dialog in response to some event. This may seem like harmless behavior, especially when you are building and testing your application in isolation, on the emulator. However, when your application is run on an actual device, your application may not have user focus at the time your background process displays the dialog. So it could end up that your application would display its dialog behind the active application, or it could take focus from the current application and display the dialog in front of whatever the user was doing (such as dialing a phone call, for example). That behavior would not work for your application or for the user.

一个常见的流畅问题是，一个应用程序的后台处理——例如，一个Service或者BroadcastReceiver——弹出一个对话框来响应一些事件。这可能看起来没啥大碍，尤其是你在模拟器上单独地构建和测试你的应用程序的时候。然而，当你的应用程序运行在真机上时，有可能你的应用程序在没有获得用户焦点时后台处理显示了一个对话框。因此，可能会出现在活跃的应用程序后方显示了你的应用程序的对话框，或者从当前应用程序夺取焦点显示了一个对话框，而不管当前用户正在做什么（例如，正在打电话）。那种行为，对应用程序或用户来说，就不应该出现。

To avoid these problems, your application should use the proper system facility for notifying the user — theNotification classes. Using [notifications](#), your application can signal the user that an event has taken place, by displaying an icon in the status bar rather than taking focus and interrupting the user.

为了避免这些问题，你的应用程序应该使用合适的系统资源来通知用户——Notification类。使用Notification，你的应用程序可以在状态栏显示一

个icon来通知用户已经发生的事情，而不是夺取焦点和打断用户。

Another example of a seamlessness problem is when an activity inadvertently loses state or user data because it doesn't correctly implement the onPause() and other lifecycle methods. Or, if your application exposes data intended to be used by other applications, you should expose it via a ContentProvider, rather than (for example) doing so through a world-readable raw file or database.

另一个流畅问题的例子是未能正确实现Activity的onPause()和其它生命周期方法而造成意外丢失了状态或用户数据。又或者，如果你的应用程序想暴露数据给其它应用程序使用，你应该通过ContentProvider来暴露，而不是（举例）通过一个可读的原始文件或数据库来实现。

What those examples have in common is that they involve cooperating nicely with the system and other applications. The Android system is designed to treat applications as a sort of federation of loosely-coupled components, rather than chunks of black-box code. This allows you as the developer to view the entire system as just an even-larger federation of these components. This benefits you by allowing you to integrate cleanly and seamlessly with other applications, and so you should design your own code to return the favor. 这些例子的共同点是它们都应该与系统和其它应用程序协作好。Android系统设计时，就把应用程序看作是一堆松散耦合的组件，而不是一堆黑盒代码。作为开发者来说，允许我们把整个系统看作是更大的组件集合。这有益于我们可以与其它应用程序进行清晰无缝的集成，因此，作为回报，我们应该更好的设计我们的代码。

This document discusses common seamlessness problems and how to avoid them.

这篇文章将讨论常见的流畅问题以及如何避免它们。它将包括这些主题：

Don't Drop Data-别丢弃数据

Always keep in mind that Android is a mobile platform. It may seem obvious to say it, but it's important to remember that another Activity (such as the "Incoming Phone Call" app) can pop up over your own

Activity at any moment. This will fire the `onSaveInstanceState()` and `onPause()` methods, and will likely result in your application being killed.

一定要记住Android是一个移动平台。可以显而易见地说，其它Activity（例如，“Incoming Phone Call”应用程序）可能会在任何时候弹出来遮盖你的Activity，记住这个事实很重要。因为这个过程将触发`onSaveInstanceState()`和`onPause()`方法，并可能导致你的应用程序被杀死。

If the user was editing data in your application when the other Activity appeared, your application will likely lose that data when your application is killed. Unless, of course, you save the work in progress first. The “Android Way” is to do just that: Android applications that accept or edit input should override the `onSaveInstanceState()` method and save their state in some appropriate fashion. When the user revisits the application, she should be able to retrieve her data.

如果用户在你的应用程序中正在编辑数据时，其它Activity出现了，这时，你的应用程序被杀死时可能丢失那些数据。当然了，除非你事先保存了正在进行的工作。“Android方式”是这样做的：能接收和编辑用户输入的Android应用程序应该重写`onSaveInstanceState()`方法，并以恰当的方式保存它们的状态。当用户重新访问应用程序时，她能得到她的数据。进行这种处理方式最经典的例子是mail应用程序。

A classic example of a good use of this behavior is a mail application. If the user was composing an email when another Activity started up, the application should save the in-process email as a draft.

如果用户正在输入email，这时其它Activity启动了，mail应用程序应该把正在编辑的email以草稿的方式保存起来。

Don't Expose Raw Data-不要暴露原始数据

If you wouldn't walk down the street in your underwear, neither should your data. While it's possible to expose certain kinds of application to the world to read, this is usually not the best idea. Exposing raw data requires other applications to understand your data format; if you change that format, you'll break any other applications that aren't similarly

updated.

如果你不想穿着内衣在大街上溜达的话，你的数据也不应该这样。尽管可能存在暴露应用程序的某种形式给其它应用程序，但这通常不是最好的主意。暴露原始数据，要求其它应用程序能够理解你的数据的格式；如果你变更了格式，那么，你将破坏那些没有进行同步更新的应用程序。

The "Android Way" is to create a ContentProvider to expose your data to other applications via a clean, well-thought-out, and maintainable API. Using a ContentProvider is much like inserting a Java language interface to split up and componentize two tightly-coupled pieces of code. This means you'll be able to modify the internal format of your data without changing the interface exposed by the ContentProvider, and this without affecting other applications.

"Android方式"是创建一个ContentProvider，以一种清晰的、深思熟虑的和可维护的API方式暴露你的数据给其它应用程序。使用 ContentProvider，就好像是插入Java接口来分离和组装两片高耦合的代码。这意味着你可以修改数据的内部格式，而不用修改由 ContentProvider暴露的接口，这样，也不会影响其它应用程序。

Don't Interrupt the User-不要打断用户

If the user is running an application (such as the Phone application during a call) it's a pretty safe bet he did it on purpose. That's why you should avoid spawning activities except in direct response to user input from the current Activity.

如果用户正在运行一个应用程序（例如，Phone程序），断定对用户操作的目的才是安全的。这也就是为什么必须避免创建Activity，而是直接在当前的Activity中响应用户的输入。

That is, don't call `startActivity()` from BroadcastReceivers or Services running in the background. Doing so will interrupt whatever application is currently running, and result in an annoyed user. Perhaps even worse, your Activity may become a "keystroke bandit" and receive some of the input the user was in the middle of providing to the previous Activity. Depending on what your application does, this could be bad news.

那就是说，不要在BroadcastReceiver或在后台运行的Service中调用callActivity()。这么做会中断当前运行的应用程序，并导致用户恼怒。也许更糟糕的是，你的Activity可能成为“按键强盗”，窃取了用户要提供给前一个Activity的输入。视乎你的应用程序所做的事情，这可能是个坏消息。

Instead of spawning Activity UIs directly from the background, you should instead use the NotificationManager to set Notifications. These will appear in the status bar, and the user can then click on them at his leisure, to see what your application has to show him.

不选择在后台直接创建Activity UI，取而代之的是，应该使用NotificationManager来设置Notification。它们会出现在状态栏，并且用户可以在他空闲的时候点击它们，来查看你的应用程序向他显示了什么。

(Note that all this doesn't apply to cases where your own Activity is already in the foreground: in that case, the user expects to see your next Activity in response to input.)

(注意，如果你的Activity已经在前台了，以上将不适用：这时，对于用户的输入，用户期望的是看到下一个Activity来响应。)

Got a Lot to Do? Do it in a Thread-有太多事情要做？在线程里做

If your application needs to perform some expensive or long-running computation, you should probably move it to a thread. This will prevent the dreaded "Application Not Responding" dialog from being displayed to the user, with the ultimate result being the fiery demise of your application.

如果你的应用程序需要执行一些昂贵或耗时的计算的话，你应该尽可能地将它挪到线程里。这将阻止向用户显示可怕的“Application Not Responding”对话框，如果不这样做，最终的结果会导致你的应用程序完全终止。

By default, all code in an Activity as well as all its Views run in the same thread. This is the same thread that also handles UI events. For example, when the user presses a key, a key-down event is added to the Activity's

main thread's queue. The event handler system needs to dequeue and handle that event quickly; if it doesn't, the system concludes after a few seconds that the application is hung and offers to kill it for the user.

一般情况下，Activity中的所有代码，包括它的View，都运行在相同的线程里。在这个线程里，还需要处理UI事件。例如，当用户按下一个按键，一个key-down事件就会添加到Activity的主线程队列里。事件处理系统需要很快让这个事件出列并得到处理；如果没有，系统数秒后会认为应用程序已经挂起并为用户提供杀死应用程序的机会。

If you have long-running code, running it inline in your Activity will run it on the event handler thread, effectively blocking the event handler. This will delay input processing, and result in the ANR dialogs. To avoid this, move your computations to a thread. This Design for Responsiveness document discusses how to do that.

如果有耗时的代码，内联在Activity上运行也就是运行在事件处理线程里，这在很大程度上阻塞了事件处理。这会延迟输入处理，并导致ANR对话框。为了避免这个，把你的计算移到线程里。在响应灵敏性设计的文章里已经讨论了如何做。

Don't Overload a Single Activity Screen-不要让一个Activity超负荷

Any application worth using will probably have several different screens. When designing the screens of your UI, be sure to make use of multiple Activity object instances.

任何值得使用的应用程序都可能有几个不同的屏幕。当设计UI屏幕时，请一定要使用多个Activity对象实例。

Depending on your development background, you may interpret an Activity as similar to something like a Java Applet, in that it is the entry point for your application. However, that's not quite accurate: where an Applet subclass is the single entry point for a Java Applet, an Activity should be thought of as one of potentially several entry points to your application. The only difference between your "main" Activity and any others you might have is that the "main" one just happens to be the only one that expressed an interest in the "android.intent.action.MAIN" action

in your `AndroidManifest.xml` file.

依赖于你的开发背景，你可能理解Activity类似于Java Applet，它是你应用程序的入口点。然而，那并不精确：Applet子类是一个Java Applet的单一入口点，而一个Activity应该看作是你的应用程序多个潜在入口点之一。你的“main”Activity和其它之间的唯一不同点是“main”Activity正巧是在`AndroidManifest.xml`文件中唯一对“`android.intent.action.MAIN`”动作感兴趣的Activity。

So, when designing your application, think of your application as a federation of Activity objects. This will make your code a lot more maintainable in the long run, and as a nice side effect also plays nicely with Android's application history and "backstack" model.

因此，当设计你的应用程序的时候，把你的应用程序看作是Activity对象的集合。从长远来看，这会使得你的代码更加方便维护。

Extend System Themes-扩展系统主题

When it comes to the look-and-feel of the user interface, it's important to blend in nicely. Users are jarred by applications which contrast with the user interface they've come to expect. When designing your UIs, you should try and avoid rolling your own as much as possible. Instead, use a Theme. You can override or extend those parts of the theme that you need to, but at least you're starting from the same UI base as all the other applications. For all the details, read [Styles and Themes](#).

当谈到UI观感时，巧妙地交融非常重要。用户在使用与自己期望相反的UI的应用程序时，会产生不愉快的感觉。当设计你的UI时，你应该尽量避免太多自己的主题。相反的，使用同一个主题。你可以重写或扩展你需要的主题部分，但至少在与其它应用程序相同的UI基础上开始。详细请参照“应用风格和主题”部分。

Design Your UI to Work with Multiple Screen Resolutions-设计你的UI可以应对多屏幕分辨率

Different Android-powered devices will support different screen

resolutions. Some will even be able to change resolutions on the fly, such as by switching to landscape mode. It's important to make sure your layouts and drawables are flexible enough to display properly on a variety of device screens.

不同的Android设备可能支持不同的屏幕分辨率。甚至一些可以自己变更分辨率，例如，切换到风景模式。确保你的布局和图片能足够灵活地在不同的设备屏幕上正常显示。

Fortunately, this is very easy to do. In brief, what you must do is provide different versions of your artwork (if you use any) for the key resolutions, and then design your layout to accommodate various dimensions. (For example, avoid using hard-coded positions and instead use relative layouts.) If you do that much, the system handles the rest, and your application looks great on any device.

幸运的是，这很容易做到。简而言之，你需要做的是为主要分辨率提供不同版本的作品，然后为不同的尺寸设计你的布局。（例如，避免使用硬编码位置而使用相对布局。）如果那样做的话，系统会处理剩下的部分，而且你的应用程序在任何设备上都看起来很棒。

Assume the Network is Slow-假设网络很慢

Android devices will come with a variety of network-connectivity options. All will have some data-access provision, though some will be faster than others. The lowest common denominator, however, is GPRS, the non-3G data service for GSM networks. Even 3G-capable devices will spend lots of time on non-3G networks, so slow networks will remain a reality for quite a long time to come.

Android设备会有多种网络连接选项。所有的都提供数据访问，但之间肯定有更快的。其中，速度最慢的是GPRS，GSM网络的非3G数据服务。即使具备3G能力的设备在非3G的网络上也会花费很多的时间，所以，网络很慢仍然是一个长期存在的事实。

That's why you should always code your applications to minimize network accesses and bandwidth. You can't assume the network is fast, so you should always plan for it to be slow. If your users happen to be on faster

networks, then that's great — their experience will only improve. You want to avoid the inverse case though: applications that are usable some of the time, but frustratingly slow the rest based on where the user is at any given moment are likely to be unpopular.

这就是为什么你应该按照最小化的网络访问和带宽来编写你的代码。你不能假设网络是快速的，所以，你应该总是计划它是慢的。如果你的用户碰巧在一个快速的网络上，那很好——他们的用户体验会提升。你要避免相反的情形：在不同的地点和不同时间，应用程序有时可用，有时慢得令人抓狂，这样的程序可能不会受欢迎。

One potential gotcha here is that it's very easy to fall into this trap if you're using the emulator, since the emulator uses your desktop computer's network connection. That's almost guaranteed to be much faster than a cell network, so you'll want to change the settings on the emulator that simulate slower network speeds. You can do this in Eclipse, in the "Emulator Settings" tab of your launch configuration or via a [command-line option](#) when starting the emulator.

还有一个潜在的地方是，如果你正在使用模拟器，那么你很容易受它迷惑，因为模拟器使用电脑的网络连接。这比手机网络快很多，所以，你需要修改模拟器设定来 模拟较低的网络速度。你可以在Eclipse中做到这点，在启动选项的模拟器设置页里设置或者在启动模拟器时通过命令行选项设置。

Don't Assume Touchscreen or Keyboard-不要假定触摸屏或键盘

Android will support a variety of handset form-factors. That's a fancy way of saying that some Android devices will have full "QWERTY" keyboards, while others will have 40-key, 12-key, or even other key configurations. Similarly, some devices will have touch-screens, but many won't. Android可以支持多种外观形状。也就是说，一些Android设备拥有全“QWERTY”键盘，而其它可能会有40键、12键或其它键盘设置。同样的，一些设备可能有触摸屏，但一些也会没有。

When building your applications, keep that in mind. Don't make assumptions about specific keyboard layouts -- unless, of course, you're

really interested in restricting your application so that it can only be used on those devices.

当创建你的应用程序的时候，记住这一点。不要假定特定的键盘布局——除非你真的想限定你的应用程序只运行在某些设备上。

Do Conserve the Device Battery- 节省设备电池

A mobile device isn't very mobile if it's constantly plugged into the wall. Mobile devices are battery-powered, and the longer we can make that battery last on a charge, the happier everyone is — especially the user. Two of the biggest consumers of battery power are the processor, and the radio; that's why it's important to write your applications to do as little work as possible, and use the network as infrequently as possible.

如果移动设备经常插在墙上，那么，它也就不是很“移动”。移动设备是电池供电的，如果我们能让每次充电的电池使用得更持久一些，那么每个人都会更加开心——尤其是用户。其中两大耗电硬件是处理器和无线；这也就是我们为什么要写尽可能少做工作、尽可能少去使用网络的应用程序的重要原因。

Minimizing the amount of processor time your application uses really comes down to writing efficient code. To minimize the power drain from using the radio, be sure to handle error conditions gracefully, and only fetch what you need. For example, don't constantly retry a network operation if one failed. If it failed once, it's likely because the user has no reception, so it's probably going to fail again if you try right away; all you'll do is waste battery power.

如何让你的应用程序最小化的占用处理器，归根结底还是要写高效代码。

为了减少无线的电量消耗，确保对错误条件进行正确的处理，并只获取你要的东西。例如，如果某一个网络操作失败了，不要不断地进行重试。如果失败了一次，有可能是用户不受欢迎，因此，如果你再以正确的方式操作，有可能还会失败；所有你做的都是在浪费电池。

Users are pretty smart: if your program is power-hungry, you can count on them noticing. The only thing you can be sure of at that point is that your program won't stay installed very long.

用户是相当聪明的：如果你的程序高耗电，他们是一定会发现的。到那个时点，你唯一可以确定的是，你的程序将很快被卸载掉。

来自“[index.php?title=Designing_for_Seamlessness&oldid=6931](#)”



Designing for Security

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： sfshine

原文链接：<http://docs.eoeandroid.com/guide/practices/security.html>

目录

[[隐藏](#)]

[1 Designing for Security](#)

[2 使用Dalvik编码](#)

[3 使用本地编码](#)

- [3.1 存储数据](#)

- [3.1.1 使用内部文件](#)

- [3.1.2 使用外部存储](#)

- [3.1.3 使用内容提供者](#)

- [3.2 使用进程间通信](#)

- [3.2.1 使用意图](#)

- [3.2.2 使用binder和AIDL接口](#)

- [3.2.3 使用广播接收者](#)

- [3.2.4 使用服务](#)

- [3.2.5 使用Activities](#)

- [3.3 使用权限](#)

- [3.3.1 请求权限](#)

- [3.3.2 创建权限](#)

- [3.4 使用网络](#)

- [3.4.1 使用IP 网络](#)

- [3.4.2 使用拨号网络](#)

- [3.5 载入动态代码](#)

- [3.5.1 使用WebView](#)

- [3.6 进行输入校验](#)

- [3.7 处理用户数据](#)
 - [3.7.1 处理证书](#)
- [3.8 使用密码学](#)
- [3.9 总结](#)

Designing for Security

Android 是这样设计的，对于大多数开发者而言，使用默认设置就可以开发应用，而不需要因为安全的问题而苦恼。Android有很多安全的特性集成到了操作系统里，显著的降低了频率和安全应用问题的影响。

下边一些安全特性有助于开发出安全的应用：

- 在每一个基础应用里，Android应用沙盒能够隔离数据和代码的执行
- 对于常见的安全功能，Android应用层框架做了很好的实现，比如密码学，权限问题，以及安全的IPC机制。
- 像ASLR，NX，ProPolice，safe_iop，OpenBSD dlmalloc，OpenBSD calloc和Linux mmap_min_addr对于减少常见内存管理错误起了很大的作用
- 加密系统能够很好的保护数据不被盗取。

因此对于开发者来说，熟悉android的一系列安全机制，能够很好的利用android的特性，并且能够减少无意中引入的安全问题。

这篇文档主要是围绕常见的API和开发技术而编写，能够对开发者和应用的用户有个很好的启示。随着这些最佳实践的不断发展，我们建议在您开发应用的过程中偶尔核对一下。

使用Dalvik编码

写安全代码运行在虚拟机上，是非常容易学习的，许多问题并非是针对android。我们推荐你熟悉已经存在的文档，而不是企图重新处理这些东西。下面是两个比较流行的资源：

<http://www.securingjava.com/toc.html>

https://www.owasp.org/index.php/Java_Security_Resources

这个文档主要是集中在android的一些特殊领域或者是不同于其他的环境。对于要

在其他的环境中体验VM编程的开发者来说，有两个扩展的事情，这些东西是和写android应用是有很大不同的。一些虚拟机，比如JVM或者.net runtime，实际上是作为一个安全边界，从底层操作系统隔离了代码。在android上，DalvikVM并不是一个安全边界，而是一个在操作系统等级上实现的应用沙盒，因此Dalvik在一些应用中能够和本地代码进行交互而没有任何的安全限制。考虑到在移动手机有限的存储，对于开发者来说模块化应用和动态加载类是非常常见的。这个时候在做的时候需要考虑两个方面的问题，一个是重用你的应用逻辑，另一个资源本地化。不要从没有被核实的资源动态类加载，比如不安全的网络资源，或者外部存储资源，由于代码能够被恶意行为所修改。

使用本地编码

通常对于大部分应用的开发，我们鼓励开发者使用androidSDK，而不是使用本地代码。带有本地代码的应用一般都是比较复杂，不那么便携。比较可能包含一些内存泄露的错误，比如缓存溢出。Android是搭建在linux内核之上的，如果你要使用本地代码，熟悉linux开发的安全实践是特别有用的。这篇文档太短而不能讨论到所有的这些安全实践，但是下边有个很好的资源：对于linux和unix HOWTO编程通过<http://www.dwheeler.com/secure-programs>. 可以获得Android和大部分linux环境最大的不同在于应用沙盒。在android上所有的应用都是运行在应用沙盒上。包括使用本地代码写的。在最基本的层面上，对于熟悉linux的开发者思考这些东西，一个很好的方法就是知道每一个应用被分配一个独特的UID，这个UID带有很多的权限。在Android Security Overview里有很详细的讨论，你应该熟悉一些应用权限即使你在使用本地编码。

存储数据

使用内部文件

默认情况下，被创建在内存里的文件只能被创建这个文件的应用访问。这样的保护被android所使用，对于大部分应用是很有效的。对于IPC的文件，使用全局写或者全局读是不被鼓励的，因为它没有能力提供一些限制数据获取的特殊应用，也没有提供任何的数据格式的控制。作为一个可选择的方法，你可以使用ContentProvider提供的读写权限，能够授予动态权限在个例上。对于敏感数据，你可以提供一些额外的保护，一些应用可以使用带有密钥的加密文件。（例如，密钥可以放在KeyStore，使用一些不存在设备上的用户密码进行保护）。虽然这不能很好的保护来自监视用户输入密码数据，但是可以提供对于没有加密的文件系统的保护。

使用外部存储

对于创建在外部的文件，不如在SD卡中，属于可读写的文件。因为外部存储的文件可以被用户移除，以及能够被应用程序所修改。应用程序不应该把一些敏感的数据放在外部存储里。对于一些不可信的数据，应用程序从外部操作数据的时候，应该对其进行输入审核（查看输入审核片段）。我们强烈推荐应用程序使用动态加载的方式，而不要存储一些可执行文件和一些类文件在外部存储里。如果一个应用程序从外部存储获取可执行文件，在动态加载之前他们应该签署并使用密码验证。

使用内容提供者

内容提供者提供一种结构存储机制，能够对你的应用程序或者被其他应用程序导出时构成一种限制的效果。默认情况下，内容提供者是为用户被其他应用导出的。如果你不想让其他应用程序使用你的内容提供者，你可以在应用manifest里设置`android:exported=false`。当创建一个内容提供者被其他应用程序导出，你可以在manifest里指定一个单个的权限用于读和写，或者特有的权限用于读和写。我们推荐你对于一些请求完成任务的操作你应该限制一些权限。记住，对于取消一些权限或者打破已经存在的使用者，你可能会更容易增加一些权限暴漏了新的功能。如果你使用内容提供者在应用之间共享数据，应该优先使用署名等级的权限。署名权限并不需要用户请求验证，因此可以提供一种很好的用户体验以及更好的控制使用内容提供者。内容提供者也提供一些常规的过去方式，靠声明`grantUriPermissions`元素和在意图类里使用

`FLAG_GRANT_READ_URI_PERMISSION`和`FLAG_GRANT_WRITE_URI_PERMISSION`标志激活应用组件。这些权限的范围可以由`grant-uri-permission`元素进行限制。当获得了内容提供者，用户使用参数化查询方式比如`query()`,`update()`和`delete()`，避免来自于不可靠数据潜在的SQL注入。值得注意的是如果选择是建立连接用户数据提交之前的方法，使用参数化查询方式是不够的。对于写权限不要有虚假的安全感。考虑到写权限允许使用SQL语句，可以让一些数据被证实使用创造性语句`WHERE`语句和解析结果。例如，只要这个电话号码已经存在，一个攻击者可能在调用日志里修改一行就可以探测一个特定的电话号码。如果内容提供者使用可以预测的结构，那么写权限等同于读写权限。

使用进程间通信

一些安卓应用企图实现进程间通信使用传统的linux技术比如网络sockets以及共享文件。对于安卓系统的进程间通信，我们强烈推荐使用Intents、binders、services、receivers。安卓IPC机制允许你核实连接到你的IPC的应用程序的统一性，设置安全策略对于每一个IPC机制。许多的安保要素通过IPC机制共享。Broadcast Receivers Activities和Services都要在manifest文件里进行声明。如果你的IPC机制不打算被别的应用使用，你可以设置`android:exported`为False。考虑到有些应用程序带有相同UID，这样设置是很有用

的。或者如果后来你在开发的时候，你不想要暴漏 IPC但是你又不想重写代码，你也可以进行这样的设置。如果你的IPC打算被其他的应用获得，你可以使用权限标志进行一些安全策略的申请。如果应用程序之间的IPC通过相同的开发者建造，那么没有比署名级权限 更适合的了。署名权限并不需要用户验证，因此可以提供一个更好的用户体验，对于IPC机制可以更好的控制访问。对于一些引入混乱的区域，可以使用意图过滤来解决。注意意图过滤不用改考虑安全的特征，组件被直接调用，可能没有数据依照意图过滤。在你接受意图的时候，你应该执行输入验证以确保receiver、service、或者activity是否是合适的格式。

使用意图

在安卓中异步进程间通信首先使用的应该是意图机制。你可以使用**sendBroadcast()**,**sendOrderedBroadcast()**，这些依赖于你应用的请求方式。或者直接使用意图到一个指定的应用组件。注意命令广播能够被接受，因此他们并不能被传达到所有的应用。如果你正在发送一个意图发送给一个指定接受者是必须的，意图必须被直接分发给值接受者。发送者的意图可以核实收件人有一个指定的权限，允许在发送时指定一个非空权限。只有有权限的应用能够接受这个意图。如果一个带有广播的数据意图可能是比较的敏感，你应该申请一个权限确保没有权限的恶意应用不能够接收到信息。在这种情况下，你一二考虑直接调用接收者，而不是使用广播。

使用binder和AIDL接口

Binders是首选的机制对于RPC格式的IPC。他们提供了一个良好定义的接口使能相互认证的端点，如果需要的话。我们强烈推荐以这种方式设计接口，不需要特殊的权限的审核。**Binders**不需要在**manifest**中进行声明，一次不需要申请说明权限对于**binder**。在应用**manifest**里，**Binders**通常继承一些权限声明。如果你创建的一个接口需要认证，或者在指定的**binder**接口中获取控制权，这些控制权必须在接口里被明确的用代码添加。如果提供一个接口不需要获取控制权，使用**checkCallingPermission()**去验证一个有必须权限的**binder**在哪里被调用。在访问一个作为调用者的服务之前，确定你的应用被转到其他的接口是非常重要的。如果调用的接口是一个服务提供的，**bindService()**调用可能会失败，如果没有获得指定的服务的权限。如果调用的接口是你本地的应用提供的，使用**clearCallingIdentity()**进行内部安全检查可能非常的有用。

使用广播接收者

广播接收者用于处理异步请求初始化通过意图来实现。默认情况下，接收者是对外开放的，可以被其他的应用调用。如果你的**BroadcastReceivers**被其他的应用使用，你可以想要申请安全权限对于接收者可以在应用**manifest**使用**<receiver>**元素。如果没有了**BroadcastReceivers**发送的权限你的应用就不能够访问了。

使用服务

服务经常被用于提供一些功能共其他的应用使用，每一个服务类必须在他包里的**AndroidManifest.xml**里进行相应的声明。默认情况下，服务是对外开放的，可以被别的应用程序调用。服务可以使用权限的属性进行相应的保护。如果设置了相应的属性，那么别的应用需要在自己的**manifest**里申请相应的使用权限才能够启动、停止、或者绑定服务。在启动调用之前调用**checkCallingPermission()**，在权限里服务可以保护个人IPC调用。我们通常推荐在**manifest**里使用权限，这样更加安全。

使用**Activities**

Activities是最常用的，对于用户交互功能的应用程序。默认情况**Activities**是对外开放的，只要他们有相应的意图过滤或者绑定声明 就可以被别的应用调用的。通常，我们推荐你应该明确的指定一个接受或者服务区处理IPC，因为这种模式方法可以减少易被其他应用访问的功能块被暴漏的风险。如果你确实想显示对于IPC的**Activity**，在**manifest**里的一些权限属性可以设置，这些权限用于你指定的用户访问。

使用权限

请求权限

我们推荐减少被应用请求权限的数量。没有获取一些敏感的权限能够降低无意中滥用这些权限，这样能够更容易让用户接受，也可以让应用免遭一些黑客的攻击。如果以一种不请求权限的方式设计你的应用，这是更可取的一种方式。例如，应用中使用**GUID**，而不是靠创建标示符来请求获得设备信息。（这个特殊的例子在 操作用户数据的时候被讨论），或者在你应用目录里存储数据，而不是使用外部存储。如果一个权限不被请求那么就不要请求它。这听起来简单，但是有相当多的研究对于过度请求权限的频率。如果你对这个课题感兴趣你可以使用被U.CBerkeley公开的研究页面开始你的研究。

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-48.pdf> 除了请求权限之外，你的应用也可以使用权限保护一些敏感的用以暴漏给其他应用程序的IPC，比如**ContentProvide**。通常情况下，我们锐减使用 获取控制而不是让用户确认一些令人迷惑的权限。例如，考虑使用签名级的保护权限对于IPC在两个应用之间的通讯。这个应用是被单个开发者提供的。不要引起权限重复授权。这种情况发生在当一个应用通过能被获得的IPC暴漏数据，因为它有一个特殊的权限，但是如果他是IPC的借口，那么就不要求任何 客户端的权限。对于潜在影响的详细内容，以及这种问题类型的频率可以参考由USENIX提供的：http://www.cs.berkeley.edu/~afelt/felt_usenixsec2011.pdf

创建权限

通常，你应该尽量的少创建权限来满足你的安全需求。对于大部分应用创建新的权限是相当的不常见的，因为系统创建的权限涵盖了很多情况。再适合的地方 使用外部权限执行获取检查。如果你必须创建一个新的权限，考虑你是否使用签名权限完成你的任务。签名权限对于用户是透明的，在应用执行权限审核的时候，只有被应用签名的相同的开发者 才能够获取权限。如果你创建了一个危险的权限，这个时候用户就要考虑是否安装你的应用了。这可以扰乱其他的开发者，也扰乱的用户。如果你创建了危险的权限，下边有许多复杂的东西你需要考虑的。这个权限必须有一个字符串描述给用户，让用户做出安全决定。这个权限描述必须使用不同语言。用户可以拒绝安装一个让人困惑或者有风险的应用。当权限没有创建的时候应用不能够安装。对于一个应用开发者，这些都是非技术的挑战，我们不鼓励使用危险的权限。

使用网络

使用**IP** 网络

网络对于Android来说意义不同于Linux环境下那么值得注意。关键需要考虑的是确保对于一些敏感数据使用适当的协议。比如网页通信的 **HTTPS**，我们倾向于使用**HTTPS**，因此移动设备经常连接网络并不是很安全，比如公共热点**WiFi**。经过确认，加密的套接字级别的通信很容易被实现通过**SSLSocket**类。考虑到Android设备经常连接一些不安全的无线网络使用**WiFi**，对于所有的应用强烈推荐使用安全的网络。对于处理一些敏感的**IPC**我们有一些应用使用本地网络端口。我们不推荐这种方法，因为在设备上这些接口不能被其他的应用获得。替代方法是，我们使用**Android IPC**机制，在可能被认证的地方，比如**Service**和**Binder**。（甚至比使用回调更糟糕的是绑定**INADDR_ANY**，当你的应用可以被任何地方请求到的时候。我们也已经看到了。）一种常见的问题，认证重复是确保你不要信任一些从**HTTP**或者其他不安全协议下载的数据。这包括在**WebView**里输入认证，以及任何对立**HTTP**事物意图的相应。

使用拨号网络

SMS是一项被Android设备经常使用的拨号网络。开发者应该记住这项协议开始设计师用于用户到用户的通信，而不是很适合于一些应用目的。由于 **SMS**的一些限制，对于发送数据到设备我们强烈推荐使用**C2DM**和**IP**网络。许多开发者并没有意识到在网络上或者在设备上**SMS**并没有被加密和认证的。特别是，任何**SMS**接收者应该期待一个恶意的用户可能已经发送**SMS**到你的应用了，不要依赖未授权的**SMS**数据执行一些敏感的命令。你也应该意识到**SMS**可能被电子诈骗所支配，或者被一些网络拦截所使用。对于Android电源设备本身，**SMS**信息是被传递给广播意图的，因此他们可能被那些授予了**READ_SMS**权限的应用读取或者捕获。

载入动态代码

我们强烈不建议从应用apk外面载入代码.这样可以显著增大因为代码注入或者代码篡改而危机应用安全性的可能.它也增加了版本管理和应用测试的复杂性.最后,它是的这个应用的行为可以被监视,所以在有些环境中,是禁止使用这个方法的.如果你的应用确实需要动态加载代码,最重要的一件事情是时刻注意动态加载的代码和应用apk的权限拥有相同的权限.用户要有决定是否安装基于你的身份的应用的权利,这些应用包括应用内部的代码和动态加载的代码. 动态加载代码的主要安全风险是这些代码必须来自一个经过认证的地方.如果这个模块是包含在你apk中的,那么他可以被其他应用修改,无论这些代码是本地库还是通过DexClassLoader 类. 我们见到过很多应用试图从不安全的地方,比如从没有经过加密协议的网络或者可以任意读写的地方(外置存储等地方)加载代码的例子.这些地方可以允许网络中的一些人修改传输的内容,并且,用户设备上的其他应用也可能修改这些内容.

使用WebView

由于WebView可以解析包含HTML和JavaScript的web内容,不恰当的使用可能引起普通的web安全问题,比如跨站脚本攻击 ((JavaScript脚本注入)).Android包含了许多减少这些潜在问题的的机制.这些机制是通过限制你应用所请求的,WebView的一些功能 来实现的. 如果你的应用不需要在一个WebView中直接使用JavaScript,不要使用setJavaScriptEnabled()方法.我们看到过一些在 软件开发中使用这个方法做其他事情的实例代码--所以如果需要的话,把他移除掉.默认的,WebView是不能运行JavaScript 的所以跨站脚本攻击是不可能的. 使用addJavaScriptInterface()方法需要特别小心,因为他允许JavaScript调用本应该是Android应用程序才可以执行 的一些操作.请只暴露addJavaScriptInterface()接口给所有输入都是值得信赖的地方.如果不信任的地方的输入也被允许使用这个接口了,不被信任的JavaScript可能会调用Android的系统的方法.一般的我们建议只暴露addJavaScriptInterface()接口给你apk中的JavaScript. 不要信任从HTTP上下载的信息,应该使用Https.即使你只连接一个网站,并且这个网站是值得信赖的,或者你有这个网站的控制权,HTTP也经常受到 HiTM(中间人攻击)和截获攻击.使用addJavaScriptInterface()的敏感功能不应该暴露给没有经过认证的来自HTTP的脚本.注 意即使使用HTTPS, addJavaScriptInterface()接口也会增加你应用被攻击面,这包括服务器基础设施,和被Android设备信任的CA证书. 如果你的应用需要在WebView中访问敏感信息,你应该使用clearCache()方法来删除存储于本地的所有信息.服务端应该使用无缓存模式来表明 应该不应该缓存这些信息.

进行输入校验

无论是什么平台,不充分的输入校验是一个影响应用安全性的最普通的问题.当然,Android会通过减少应用暴露的平台级策略来进行输入校验,这些 功能是你应该

尽可能的使用的.也要注意选择类型安全语言来减少输入校验问题出现的可能.我们强烈建议你使用AndroidSDK来创建应用. 如果你在使用本地代码,那么任何从文件读取的数据,从网络接受的数据,或者从IPC接受的数据都有引起安全问题的可能.最普通的问题就是缓冲区溢出,use after free(当应用使用已释放的指针时,就会产生Use after free错误)和off-by-one errors(差一错误,一般发生在临界的时候).AndroidSDK提供了一系列的技术(比如ASLR和DEP)来减少这类错误的发生,但是不能从根本上解决问题.这些问题可以通过认真处理指针和管理缓冲来避免. 动态的,基于字符串的语言,比如 JavaScript和SQL也需要输入校验,因为有可能在语句中漏掉字母或者遇到代码注入的情况. 如果你在使用从SQL数据库查询到的数据或者是在使用一个Content Provider,可能会出现SQL注入的问题.最好的防御方法是使用参数化序列.这在ContentProviders一节中讲述过.有限的权限(只读或者只写)也可以减少和SQL注入有关的安全性问题. 如果你在使用WebView,那么你必须考虑XSS(跨站脚本攻击)问题.如果你的应用不直接使用WebView中的JavaScript,不要调用setJavaScriptEnabled(),那么XSS将永远不能发生.如果你必须使用JavaScript,那请参考WebView一章,那里提供了其他安全性说明. 如果你不能使用上面的安全功能,我们强烈建议你采用良好的数据结构格式,并校验这个数据是否符合格式.虽然把一些字符列入黑名单和字符替换可能是一个很好的策略,但这些技术在实践中是易出错的,最好尽可能的避免使用这些技术.

处理用户数据

一般的方法还是减少访问敏你感信息和个人数据的Api的使用.如果访问数据的时候可以避免储存或者传送这些信息,那就不要储存和传送这些信息了.最后,思考一下有没有一种可以使你应用通过使用混淆或者不可逆的加密的方式来实现业务逻辑的方式.比如,你的应用可能需要使用混淆后的一个 email地址作为主键来避免传送或者储存真正的邮件地址.这减少了不小心暴露数据的概率,也可以减少攻击者试图获取你数据的可能. 如果你的应用访问类似于用户密码和用户名之类的个人数据信息,记住有些地区可能要求你提供一个隐私政策来解释你为什么使用和存储这些信息.所以减少对用户信息的使用也会简化遵守法规的步骤. 你也应该考虑一下你的应用是否故意的暴露个人信息给其他部分了,比如暴露给了第三方组件或者被你应用使用的第三方组件的服务.如果你不知道是否这个组件或者这个服务需要访问个人信息,不要提这些信息.一般的,在应用中减少对个人信息的访问将会减少潜在的隐私问题. 如果真的需要访问敏感信息,评估一下这个信息是否必须发送到服务器还是这个操作可以由客户端进行.尽量在客户端操作敏感数据而不是传送用户数据给服务器. 同时,确保你没有故意把用户数据通过过于开放的IPC,可以任意读写的文件或者网络Socket暴露给设备上的其他应用.真只是重新授权的一个小情况,具体参阅Requesting Permissions一节. 如果需要一个GUID,请创建一个大的,独立的数字,然后存储它.不要使用电话识别码(比如IMEI)之类可能和用户个人信息有关的信息作为识别码.这个课题在[Android Developer Blog]有介绍. 用开发者在写设备目

志文件时应该注意一下.在Android上,日志文件是一个共享的资源,可以被任何应有 [READ_LOGS](#) 权限的应用访问.即使设备日志数据是临时的,会在下次启动的时候被清除.不合理的记录用户信息可能会不小心丢失用户的数据给其他应用.

处理证书

一般的,我们建议尽量少的请求用户证书.这样可以使钓鱼攻击行为更加明显却不容易成功.如果可能,用户名和密码不应该被存储在设备上.而是应该使用 用户提供的用户名和密码进行最初的授权,之后使用短期的,有服务器特性的认证口令.需要和许多应用通讯的服务应该通过

[\[docs.eoeandroid.com/reference/android/accounts/AccountManager.html\]](http://docs.eoeandroid.com/reference/android/accounts/AccountManager.html)

[AccountManager](#)]进行访问.如果可以,使

用[\[docs.eoeandroid.com/reference/android/accounts/AccountManager.html\]](http://docs.eoeandroid.com/reference/android/accounts/AccountManager.html)

[AccountManager](#)]类来调用基于云的服务而不要保存用户名和密码.用

了[AccountManager](#)重新获得一个帐号后,在授予他任何证书之前,请检查他的创建者,这样可以避免无意中把证书授予给错误的应用.用

了[\[docs.eoeandroid.com/reference/android/accounts/AccountManager.html\]](http://docs.eoeandroid.com/reference/android/accounts/AccountManager.html)

[AccountManager](#)]重新获得一个帐号后,在授予他任何证书之前,请检查他的创建者,这样可以避免无意中把证书授予给错误的应用.

使用密码学

为了进行数据隔离,对全文件系统进行加密,并提供安全通讯渠道,Android提供了一系列的算法来通过密码学保护数据.一般的,请尝试使用你的情况可以支持的,已有的,最高级的框架来实现.如果你需要从一个未知的地方安全地恢复一个文件,一个简单的[HttpsURI](#)可能就是足够的,而不需要你这边 使用密码学知识.如果你需要一个安全的通道,尝试使用[HttpsURLConnection](#)或者[SSLSocket](#),而不是自己写协议. 如果你发现你自己需要实现你自己的协议,我们建议你不要实现你自己的加密算法.请使用现存的密码算法,比如Cipher类中的AES或者是RSA。使用一个安全的随机数生成器([SecureRandom](#))来初始化任何密钥.使用一个非安全随机数生成器生成的数会降低算法的健壮性,可能会引起离线攻击. 如果你需要存储一个需要重复使用的密钥,请使用一个类似[\[docs.eoeandroid.com/reference/java/security/KeyStore.html\]](http://docs.eoeandroid.com/reference/java/security/KeyStore.html)的可以提供长期存储又可以检索得到的机制.

总结

Android使得开发者可以设计含有广泛安全性的应用.这些最好的实践将会帮助你确信你的应用利用了这个平台提供给你的这些安全福利. 你可以在 [<https://groups.google.com/forum/?fromgroups#!forum/android-security-discuss> Android Security Discuss Google Group]中接受根据这类主题,也可以和其他开发者交流这些安全实践.

来自“[index.php?title=Designing_for_Security&oldid=11618](#)”



Google Services

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：玄月冰灵

主任务原文链接：<http://developer.android.com/guide/google/index.html>

目录

[[隐藏](#)]

[1 谷歌服务](#)

- [1.1 使应用盈利](#)
 - [1.1.1 谷歌AdMob广告](#)
 - [1.1.2 应用内支付](#)
 - [1.1.3 应用许可](#)
- [1.2 提高你的应用性能](#)
 - [1.2.1 谷歌Play服务](#)
 - [1.2.2 谷歌云推送](#)
 - [1.2.3 谷歌地图](#)
- [1.3 管理应用发布](#)
 - [1.3.1 Google Play过滤](#)
 - [1.3.2 多样的应用支持](#)
 - [1.3.3 APK扩展文件](#)
- [1.4 跟踪性能和分析](#)

谷歌服务

谷歌提供大量服务帮助你建立新的收入来源，提高你的应用性能，管理分

发和负载，跟踪使用和安装。接下来一些高亮的部分由谷歌提供，链接到更多关于如何在你的安卓应用中使用它们的信息。

使应用盈利

有很多方法使你的安卓应用盈利，例如添加广告和应用内支付，如果你选择下载你的应用时收费，安卓也提供了验证应用许可能力来保护你的收入。因为不同的应用要求不同的策略，你可以选择一种最适合你的应用。

谷歌**AdMob**广告

以在你的应用中显示多样的广告网络的方式实现收入。

应用内支付

直接在你的应用中提供新内容或虚拟物品等特色来吸引用户。

应用许可

在你的应用中保护你的收入来源和整合政策的使用。

提高你的应用性能

安卓和谷歌技术一起工作来提供你引人注目的技术，例如地图和Google+。

谷歌**Play**服务

在你的应用中利用谷歌产品为你的用户提供一个简单使用的身份验证流。



谷歌云推送

以轻量级和省电的方式在有重要事件时用信息通知你的应用。



安卓的谷歌地图库给你的应用带来强大的地图性能。



管理应用发布

Google Play允许你以特征来管理你的应用发布，使你可以控制哪些用户可以下载你的应用，例如发布单独的基于你应用某一特征的平台版本。

Google Play过滤

通过过滤一个大范围的特征，例如平台版本和硬件特征来确认你的应用得到了正确的用户。

多样的应用支持

分发基于各种属性的不同APK，例如平台版本、屏幕尺寸和GLES贴图压缩支持。

APK扩展文件

轻击进入谷歌内容发送服务，提供高达4GB的免费资产服务，提供用户在你的应用中所需要的高品质图形、媒体文件或其他大型资产。

跟踪性能和分析

谷歌分析使你找出你的用户如何找到你的应用和如何使用它们。开始整合分析来估量你的应用是否成功。

来自“[index.php?title=Google_Services&oldid=9001](#)”



In-app Billing

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：玄月冰灵

原文链接：<http://developer.android.com/guide/google/play/billing/index.html>

应用内支付

应用内支付是一个使你从你的应用内部销售数字内容的Google Play服务，你可以使用此服务销售各种内容。包括下载内容，例如媒体文件或图片；虚拟内容，例如游戏等级和药水，高级服务和特色等。你可以使用应用内支付销售产品，例如

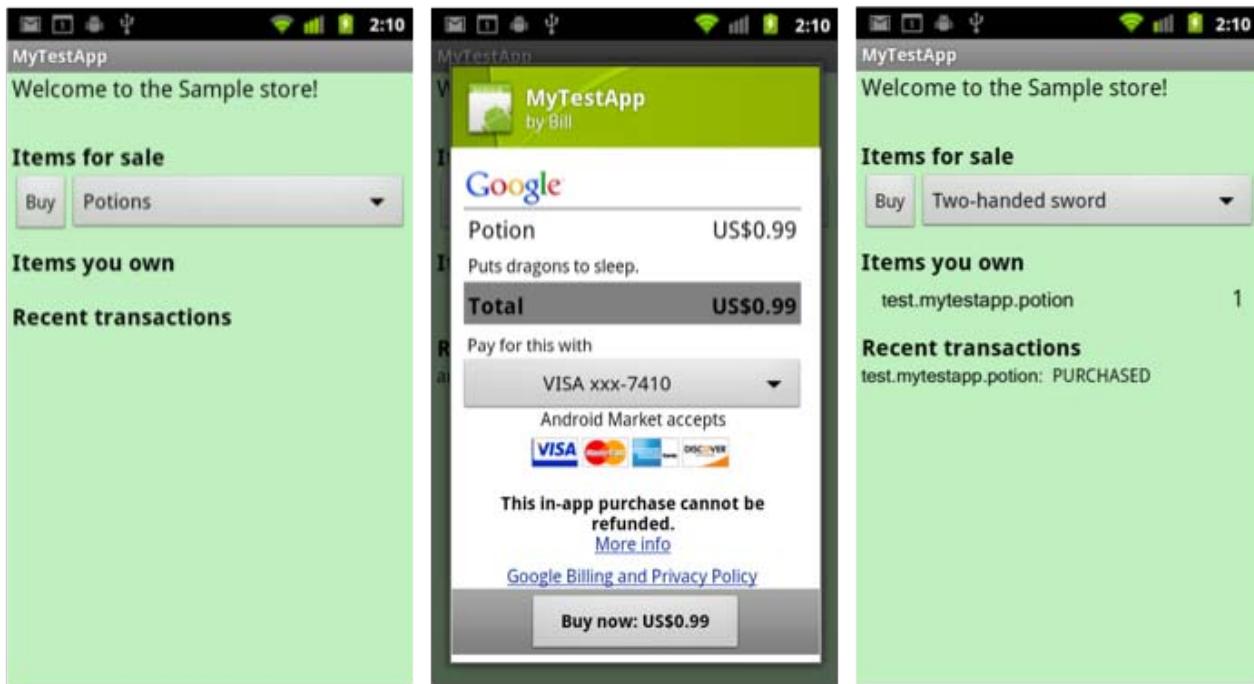
- 标准应用内产品（一次性支付），或
- 订阅（多次的自动订单）

当你使用应用内支付销售一件商品，无论是应用内商品还是一个订阅，Google Play处理所有的支付细节，你的应用无需处理任何财产交易。Google Play使用相同的支付后端服务，例如购买应用，因此你的用户经历了一致且相似的购买流程（查看数字1）。同时，应用内购买和购买应用的交易费一样（30%）。

任何你通过Google Play发布的应用都可以使用应用内支付。没有特别的账号，只需注册一个安卓市场发布者账号和谷歌钱包商务账号。同时，因为服务没有使用特定的框架API，你可以使用最小编制的API 4或更高来添加应用内支付到任何应用。

帮助你整合应用内支付到你的应用，安卓SDK提供了一个样例应用来展示如何从应用内销售标准的应用内产品和订阅。样例包含了在你的应用中可以实现应用内支付的订单相关类的例子。同时包含了实现应用内支付可能用到的数据库、用户界面和业务逻辑。

重要：虽然样例应用是教你如何实现应用内支付的可使用例子，我们强烈推荐在你把它用在产品应用中时修改和打乱代码。查看安全和设计。



数字1：应用初始化应用内支付通过它们自己的用户界面来请求（第一屏）。Google Play响应请求提供支付用户界面（第二屏）。当支付完成，应用继续。

学习更多关于Google Play应用内支付服务以及开始整合它到你的应用，阅读以下文档：

应用内支付预览

学习服务如何工作以及实现典型的应用内支付。

实现应用内支付

使用这个按步骤的向导开始把应用内支付合并入你的应用，说明同时应用于一次性和订阅购买。

订阅

学习订阅如何工作和如何在你的应用中实现对它们的支持。

安全和设计

回顾这些最好的练习帮助确保你的应用内支付安全实现且设计良好。

测试应用内支付

理解应用内支付测试工作如何工作和学习如何测试你的应用内支付是否实现。

管理应用内支付

学习如何建立你的产品列表，注册测试账号和处理退款。

应用内支付参考

获取关于Google Play响应代码和应用内支付用户界面的细节信息。

来自“[index.php?title=In-app_Billing&oldid=9005](#)”



In-app Billing Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：为海不为彼岸

原文链

接：

http://developer.android.com/guide/google/play/billing/billing_overview.html

目录

[[隐藏](#)]

[1 应用程序内部付费机制概述](#)

- [1.1 产品类型与购买方式](#)
 - [1.1.1 产品类型](#)
 - [1.1.2 购买方式](#)
- [1.2 应用内支付架构](#)
- [1.3 应用内支付消息宏定义](#)
 - [1.3.1 请求宏定义](#)
 - [1.3.2 响应宏定义](#)
 - [1.3.3 消息传递流程](#)
 - [1.3.4 处理 IN_APP_NOTIFY 消息](#)
 - [1.3.5 处理多次 IN_APP_NOTIFY 消息](#)
 - [1.3.6 处理退款和其他未请求便发送的 IN_APP_NOTIFY 消息](#)
- [1.4 安全控制](#)
- [1.5 应用内支付的要求和限制](#)

应用程序内部付费机制概述

应用程序内部付费机制（以下简称应用内支付）是Google Play的一项服务，这种服务为应用内购买提供支付流程。要使用这项服务，你的应用会对一个特定的应用内产品发送一个结账请求。然后该服务会处理这笔交易的所有细节，包括请求和确认支付形式和处理金融方面的交易。支付流程完成后，该服务会发送购买细节到你的App，比如订单号、订单时间、价格。你的App无需理会金融方面的交易；这都由Google Play的应用内支付服务来提供。

产品类型与购买方式

应用内支付支持不同种类的产品类型和购买类型，为你的App提供灵活的赚钱方式。在所有情况下，你都要使用Google Play开发者控制台定义你的产品，包括产品类型、购买类型、库存单位、价格、描述等等。想了解更多信息，请看 [管理应用内支付](#)。

产品类型

使用本服务你能卖两种产品：应用内产品和订阅。两种产品的账单特性完全不同，但是我们的API能让你使用同样的通信模型、数据结构、用户交互来处理他们，后面会讲到。

- 应用内产品——用户一次购买一个的东西。例如，典型是让用户购买数字内容，解锁App功能，一次性充值，或者添加任何东西到用户体验。不像购买Apps，一旦用户买了就没有退款的窗口了。如果用户想退款只有联系开发者。

该产品可以用两种方式出售：“限定账号”和“不限账号”。该产品总是跟唯一的App耦合。也就是说，一个App不能购买另一个App里面发布的产品，即使由一个开发者开发。该类产品被所有的应用内支付服务支持。

- 订阅（购买后有一定有效期）——这种物品使用开发者指定的、每隔一段时间就循环的账单。当用户购买一个 subscription，Google Play和它的支付处理器自动为用户生成一个账单，包含指定的时限和价格，装载这个数目到原始的支付方法。一旦用户购买一个订阅后，Google Play继续无限期地为这个产品标价，不会请求用户的确认。用户能在任何时候取消这个

订阅。

只能使用“限定账号”的方式。因为如果是买第一种产品，一旦用户买了就不会退款。若想退款只有直接联系开发者。想看更多信息以及如何销售，请看[订阅](#) 文档。

购买方式

我们的应用内支付服务提供两种购买方式：“限定账号”和“不限账号”。购买方式决定了Google Play如何来处理和跟踪购买。

- **限定账号**——物品只能每次被一个Google Play账号购买，当用户选择这种类型，Google Play 会为每个账号的每次购买永久保存交易信息。这使你能够查询到Google Play里面的购买信息。如果用户尝试购买一个已经买过的产品，那么系统会返回一个已经购买的错误报告。

如果你出售游戏等级或者应用特性，这种方式就很有用。这些东西往往不是一个临时行为，需要被存储以便用处重新安装你的App，删除他设备里的数据，或者把你的App安装到另一台设备。

- **不限账号**——物品的交易信息是不保存在Google Play中的。这意味着你无法从Google Play中查询你的交易信息，你必须自己负责管理交易信息。同样的，如果用这种方式，Google Play不会阻止用户多次购买。货物能购买多少次就交给你自己控制了。

这种方式很有用，如果你想出售一些类似消费品的东西，如燃料和魅力值。这些东西常常在你的应用里面被消费掉，而且能多次购买。

应用内支付架构

你的App使用设备中Google Play App提供的API 来访问应用内支付服务。Google Play App 使用异步消息循环来传达账单请求，并且在你的App和Google Play服务器间执行响应。在实践中，你的App绝不会直接与Google Play服务器交互（见图1）。相反，你的App使用IPC发送结账请求到Google Play App，然后取回购买响应，方式是异步广播。你的App自己不会去管与Google Play服务器的网络连接或者使用其他特殊的API。



图1：你的App通过Google Play App发送和取回结账消息，后者负责与Google Play服务器通信。

有些应用内支付实现会使用私有的服务器来交付内容或确认交易，但是远程服务器不必实现应用内支付。如果你出售需要下载到用户设备的数字内容（如说媒体文件），这种情况下私有服务器会有用。你也可能使用远程服务器来存储用户交易历史或执行各种确保支付安全的任务，比如签名验证。虽然你能够在App里面处理所有安全相关任务，但还是建议你放在远程服务器里，因为这样有助于你的App减少被攻击的风险。

典型的应用内支付实现包含3个组件：

- 一个Service (在示例中被命名为BillingService)，它处理从你的App发送账单请求到Google Play 应用内支付服务的购物消息。
- 一个BroadcastReceiver (在示例中被命名为 BillingReceiver)，他接收来自Google Play App的所有的账单异步响应。
- 一个安全组件 (在示例中被命名为Security)，它执行安全相关的任务，比如签名验证和随机数生成。想了解应用内支付安全的更多信息，请看本文中后续的Security controls。

你可能同样希望使用另外两个支持支付的组件。

- 一个响应Handler (在示例中被命名为 ResponseHandler)，他提供App指定的购物通知、报错和其他状态消息的处理。
- 一个观察者 (在示例中被命名为PurchaseObserver)，它负责发送回调到你的App，以便你能使用购买信息和状态更新你的GUI。

除了这些组件，你的App必须实现保存用户信息的方法，还有用户的购买、选择货物的接口。你不必提供结账的接口，当用户初始化一个应用内购买，Google Play App会展示出结账接口的。当用户完成结账流程，你的App会继续运行。

应用内支付消息宏定义

basic request-response messaging that takes place between your application and the Google Play application. 当用户开始购买，你的App使

用IPC函数调用发送购物消息到Google Play的应用内支付服务（MarketBillingService）。Google Play App同步响应所有支付请求，为你的App提供状态通知等等信息。Google Play App也异步响应一些账单请求，为你的App提供出错消息和交易细节。西门的章节描述了你的App和Google Play App之间基本的请求/响应消息。

请求宏定义

你的App发出应用内支付请求，需要调用IPC函数 (`sendBillingRequest()`)，这个函数由MarketBillingService接口提供。该接口定义于 Android Interface Definition Language 文件(`IMarketBillingService.aidl`)。你能够下载该AIDL文件和应用内支付示例程序。

`sendBillingRequest()`函数只有一个Bundle参数。你发送的Bundle必须包含一系列键值对来制定各种请求参数，比如账单请求的类型、被购买的物品和它的类型，还有发送该请求的App。想了解更多的关于请求中的Bundle键的信息，请见应用内支付Service接口。

Bundle中最重要的键之一是 `BILLING_REQUEST` 键，它让你指定账单请求的类型。Google Play应用内支付服务支持如下5种账单请求：

- **CHECK_BILLING_SUPPORTED**

这个请求用来验证Google Play App是否支持应用内支付。你常常得在App首次运行时候发送这个请求。这个请求非常有用，因为你可以根据是否支持应用内支付来安排你下一步的UI。

- **REQUEST_PURCHASE**

发送购买请求到Google Play，它也是应用内支付的基础。 You send this request when a user indicates that he or she wants to purchase an item in your application.当用户表示他想在你的App里面购买一些东西的时候，你就发送这个请求。 Google Play通过显示结账GUI来响应这个请求。

- **GET_PURCHASE_INFORMATION**

取回购买状态改变的信息。用户成功或失败购物都会使购买状态改变。退款

也会触发状态改变。一旦购买状态改变，Google Play会主动通知你，所以你有你想自己取回信息的时候才发送该请求。

- CONFIRM_NOTIFICATIONS

确认你的App收到了购买状态改变的通知。Google Play会一直发送状态改变通知到你的App，直到你发送这个确认。

- RESTORE_TRANSACTIONS

取回用户的交易状态，只针对限定账号的购买 和 订阅。仅仅在你想取回用户交易状态的时候才发送该请求，这种情况往往发生于你的App被重新安装或者首次安装时。

响应宏定义

Google Play App可以响应同步的或异步的应用内支付请求. 同步响应的Bundle 包含如下的3个键：

- RESPONSE_CODE

提供请求的状态、出错信息。

- PURCHASE_INTENT

提供 PendingIntent, 你使用它来生成一个结账界面。

- REQUEST_ID

提供请求的身份识别，你用它来匹配请求与异步响应。

这些键不是跟每个请求都相关的。想了解更多，请看本文后续的消息传递流程。

异步响应消息被以个别广播的形式来发送，包括下面**3**个宏：

- com.android.vending.billing.RESPONSE_CODE

该响应包括一个**Google Play**服务器响应码，它在你做出应用内支付请求后发送。服务器响应码能显示你的账单请求成功发送到**Google Play**，或者是请求出错。该响应不会用来报告购买状态的变更（比如退款或购买信息）。想了解更多的关于该响应的码字信息，请看 [应用内支付的服务器响应码](#)。

- `com.android.vending.billing.IN_APP_NOTIFY`

该响应表示购买状态变更，也就是说购买成功、取消、或退款。该响应包含一个或多个通知ID。每个通知ID跟一个指定的服务器端消息绑定，每个消息又包含了一个或多个交易。在你的App收到`IN_APP_NOTIFY`广播后，你发送一个 `GET_PURCHASE_INFORMATION` 请求，连同通知ID，去检索消息细节。

- `com.android.vending.billing.PURCHASE_STATE_CHANGED`

该响应包含一个或多个交易的细节信息。交易信息在一个**JSON**串中。该**JSON**串是已签名的，而且签名连同那个**JSON**串（未加密）发送给你的App。为帮助确保你的应用内支付消息的安全，你的App可以校验**JSON**串的签名。

该**JSON**串由叫做**PURCHASE_STATE_CHANGED** 的intent返回，它为你的App提供一个或多个账单交易的细节。一个针对订阅的**JSON**串示例如下：

```
{
  "nonce" : 1836535032137741465,
  "orders" :
    [ {
      "notificationId" : "android.test.purchased",
      "orderId" : "transactionId.android.test.purchased",
      "packageName" : "com.example.dungeons",
      "productId" : "android.test.purchased",
      "developerPayload" :
        "bGoa+V7g/yqDXvKRqq+JTFn4uQZbPiQJo4pf9RzJ",
        "purchaseTime" : 1290114783411,
        "purchaseState" : 0,
        "purchaseToken" : "rojeslcdyyiapnqcynkjyyjh" } ]
}
```

想了解更多关于**JSON**串的域的信息，请看应用内支付广播。

消息传递流程

典型的消息传递流程在图2中显示。每次sendBillingRequest()传递的请求类型都用粗体标示，广播intents用斜体标示。为了讲得明白，图2没有显示为每次请求发送的RESPONSE_CODE 广播intents。

基本的流程是如下的9步：

1. 你的App发送一个购买请求(**REQUEST_PURCHASE**)，指定一个产品ID和其他参数。
2. PURCHASE_INTENT 键提供一个 PendingIntent，你的App利用它来为给定的产品ID生成一个结账的UI。 Google Play App给你的App发送一个Bundle，其中包含3个键：RESPONSE_CODE, PURCHASE_INTENT, and REQUEST_ID。
3. 你的App发起一个挂起intent，同时弹出一个埋单的UI。

注意：你必须从一个Activity context发起挂起intent，而不能是整个程序的context。

4. 当结账流程结束（用户成功购买了货物或者取消了购买），Google Play会給你的App发送一个通知消息（**IN_APP_NOTIFY** 广播）。这个通知消息包括了指向该交易的通知ID。
5. 你的App通过发送一个**GET_PURCHASE_STATE_CHANGED**来请求交易信息，该请求指定了交易的通知ID。
6. Google Play App发送一个 Bundle，携带两个键：RESPONSE_CODE、REQUEST_ID。
7. Google Play 给你的App发送交易信息，该信息保存在PURCHASE_STATE_CHANGED 广播 intent中。
8. 你的App通过发送一个确认消息(**CONFIRM_NOTIFICATIONS**)来确认你接收到给定通知ID的交易信息，该消息指定了你接收到交易信息对应的通知ID

9. Google Play App给你的App发送一个Bundle，里面包含RESPONSE_CODE键和 REQUEST_ID键。



图2. 购物请求的消息传递流程

记住，当你从Google Play 接收到交易信息(图2第8步)你必须发送一个确认。否则，Google Play会继续发送IN_APP_NOTIFY消息因为你没有确认。作为最佳实践，对一个货物，直到你交付客户之前你都不应发送CONFIRM_NOTIFICATIONS请求。这样可以保证：一旦你的应用崩溃或者其他原因无法发货，你的App仍然会接收来自Google Play的IN_APP_NOTIFY广播，表示你需要发货。同样的，作为最佳实践，你的App必须能够处理包含多重订单的IN_APP_NOTIFY 消息。

图3显示的是，修复交易请求的消息流程。对每个 sendBillingRequest()方法的请求类型都用粗体标示，广播用斜体. 为了简明，图3没有显示发给每个请求的RESPONSE_CODE 广播。



图3. 修复交易请求的消息流程

该请求触发了三个响应。第一个响应是一个Bundle，它携带一个RESPONSE_CODE键和一个REQUEST_ID 键。第二个响应是Google Play App发送一个 RESPONSE_CODE 广播，它提供请求的状态信息和错误信息。RESPONSE_CODE消息总是指向一个特定的请求ID，所以你能决定RESPONSE_CODE消息适合哪个请求。

RESTORE_TRANSACTIONS请求也会触发一个PURCHASE_STATE_CHANGED广播，此广播包含在一个购买请求内发送的同类交易信息。然而，不像购买请求一样，该交易不会同通知ID一起交给你。

所以你不必使用CONFIRM_NOTIFICATIONS消息响应这个intent。

注意：只有当你的App首次安装或者卸载后再次安装，你才应当使用RESTORE_TRANSACTIONS 请求类型。

图4显示了检查系统是否支持应用内支付的消息流

程。 `sendBillingRequest()` 函数的请求类型用粗体显示。



图4. 检查系统是否支持应用内支付的消息流程

对 `CHECK_BILLING_SUPPORTED` 请求的同步响应提供了一个携带服务器响应码的 `Bundle`。`RESULT_OK` 响应码表明应用内支付被支持；而 `RESULT_BILLING_UNAVAILABLE` 响应码表明 应用内支付不被支持，因为你指定的 API 版本不可识别，或者用户无法合法地进行应用内购买(比如说，用户位于一个无法使用应用内支付的国家). `SERVER_ERROR`也可能被返回，表明 Google Play 服务器有问题。

处理 `IN_APP_NOTIFY` 消息

通常，你的App接到一个来自Google Play的 `IN_APP_NOTIFY`广播，作为 `REQUEST_PURCHASE`消息的响应 (请看图2). `IN_APP_NOTIFY`广播通知你的App请求购买的状态改变了。要检索购买细节的话，你的App要发送 `GET_PURCHASE_INFORMATION` 请求。Google Play用 `PURCHASE_STATE_CHANGED`广播来响应，该广播包含了购买状态变更的信息。然后你的App发送给一个 `CONFIRM_NOTIFICATIONS`消息，通知 Google Play 你收到了购买状态变更的信息。

在一些特殊情况下，你会收到多条 `IN_APP_NOTIFY`消息，即使你确认接收到购买信息；或者你会收到 `IN_APP_NOTIFY`信息说购买状态改变了，但你从来没有发起过购买。这两种特殊情况你的App都必须能够处理。

处理多次 `IN_APP_NOTIFY` 消息

当 Google Play 接到对应于 `PURCHASE_STATE_CHANGED` 的 `CONFIRM_NOTIFICATIONS` 消息，他通常会停止发送针对该 `PURCHASE_STATE_CHANGED` 消息的 `IN_APP_NOTIFY intents`。然而，有时候 Google Play 会发送重复的 `IN_APP_NOTIFY intents` 虽然你的App已经发送 `CONFIRM_NOTIFICATIONS`。当你发送 `CONFIRM_NOTIFICATIONS` 时设备丢失网络连接，这就可能发生。此时 Google Play 可能不会接到 `CONFIRM_NOTIFICATIONS` 所以他会发送多个 `IN_APP_NOTIFY` 知道它受到你的确认。所以你的App必须能够识别后来的 `IN_APP_NOTIFY` 消息是对应以

往处理的那个交易。你能够通过检查JSON串中的**orderID**来做到这一点，因为每个交易有唯一的一个**orderId**。

处理退款和其他未请求便发送的 **IN_APP_NOTIFY** 消息

两种情况下你的App会收到**IN_APP_NOTIFY**广播，即使你的App没有发送**REQUEST_PURCHASE**。图5 显示了这两种情况的消息传递流程。每个**sendBillingRequest()**函数的请求类型用粗体显示，广播用斜体显示。为了清晰，图5 没有画出针对每次请求的**RESPONSE_CODE**广播。



图5. 处理退款和其他未请求便发送**IN_APP_NOTIFY**消息的流程

第一种情况，你的App可能收到**IN_APP_NOTIFY**，当用户把你的App安装到多台设备中，然后用户从其中一台发起应用内购买。此时Google Play发送一个**IN_APP_NOTIFY** 消息到第二台设备，通知App购买状态发生了改变。你的App要能处理这条信息，就像它处理来自应用初始化的**REQUEST_PURCHASE**消息响应一样，以便你的App最终接收到**PURCHASE_STATE_CHANGED** 广播intent消息，该消息包括了被购买商品的信息。这只适用于[购买信息](#)被设置为“限定账号”的商品。

第二种情况，你的App会收到**IN_APP_NOTIFY**广播，当Google Play接到一个来自Google Wallet的退款通知。此时Google Play 发送一个**IN_APP_NOTIFY**到你的App。你的App就像它处理来自应用初始化的**REQUEST_PURCHASE**消息响应一样处理这个消息，最终使得你的App能收到**PURCHASE_STATE_CHANGED**消息，包含被退款的商品信息。

退款信息在JSON串中，该串与**PURCHASE_STATE_CHANGED**广播是一起的。同样的 JSON串中的**purchaseState** 域 被置为2.

重要提醒：你不能使用Google Wallet API来发出退款或者取消 应用内支付交易。你必须通过你的Google Wallet商业账号手工操作。但你可以使用Google Wallet API取回订单信息。

安全控制

为帮助确保发送给你的交易信息的完整性，Google Play 对JSON字符串进行了签名，它位于PURCHASE_STATE_CHANGED广播intent中。Google Play 使用私钥来关联你的发布账号来创建这个签名。发布者站点生成一个RSA key来匹配每个发布账号。在你的账号概览页面，你可以找到这个密钥对的公钥部分。他跟Google Play许可证使用的公钥一样。

当Google Play对一个账单响应做签名，它包括未加密的JSON串和一个签名。当你的App接到这个签名过的响应后，你可使用你的RSA key的公钥部分来校验该签名。通过执行签名验证你能够检测到被篡改的或被欺骗的响应。你能在App里执行这个签名校验步骤。然而，如果你的App连接到一个安全的远程服务器，我们建议你在服务器上完成校验步骤。

应用内支付也使用nonce（一次性随机数）来帮助验证Google Play返回的购买信息的完整性。你的App必须生成一个随机数然后用GET_PURCHASE_INFORMATION和RESTORE_TRANSACTIONS请求跟它一起发送。当Google Play接收到请求，它把随机数加入包含交易信息的JSON串，然后对这个JSON串签名并返回给你的App。当你的App收到此JSON串后你必须校验它的随机数和签名。

如果想了解更多最佳安全设计的实践，请看[安全与设计](#)。

应用内支付的要求和限制

在你开始应用内支付之前，确保你知道如下要求和限制：

- 应用内支付只能在Google Play发布的App中使用。
- 想使用Google Play应用内支付，你必须拥有一个Google钱包商业版账号。
- 应用内支付需要2.3.4或更高版本的Android Market App. 想用“订阅”的话，需要3.5或更高版本的Google Play App。在Android 3.0平板上，需要安装5.0.12或更高版本的MyApps。
- 运行Android 1.6 (API level 4)或更高版本的设备才能使用应用内支付。
- 使用应用内支付可以卖数字内容。用应用内支付不可出售实物、个人服务或者其他任何需要实物交付的东西。

- Google Play 不提供任何形式的内容交付，这由你自己负责。
- 在一个不联网的设备里无法使用应用内支付。为完成购买请求，用户必须能够连接上Google Play 服务器。

要了解更多应用内支付的要求，请看[应用内支付可用性与政策](#)。

来自 "[index.php?title=In-app_Billing_Overview&oldid=9138](#)"



Implementing In-app Billing

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：eversleeping

原文链接：http://developer.android.com/guide/google/play/billing/billing_integrate.html

目录

[[隐藏](#)]

[1 Implementing In-app Billing](#)

- [1.1 Downloading the Sample Application](#)
 - [1.1.1 Configuring and building the sample application-配置和构建这个示例应用程序](#)
 - [1.1.2 Uploading the sample application-上传示例应用程序](#)
 - [1.1.3 Running the sample application-运行样例应用程序](#)
- [1.2 Adding the AIDL file to your project-添加了AIDL文件到您的项目](#)

Implementing In-app Billing

In-app Billing on Google Play provides a straightforward, simple interface for sending in-app billing requests and managing in-app billing transactions using Google Play. This document helps you implement in-app billing by stepping through the primary implementation tasks, using the in-app billing sample application as an example.

在谷歌应用收费播放提供了一个简单的,简单的接口来发送请求应用收费和管理应用收费事务使用谷歌play。这个文档可以帮助你实现应用收费,通过加强通过初步实现任务,使用应用收费示例应用程序作为一个例子。

Before you implement in-app billing in your own application, be sure that you read Overview of In-app Billing and Security and Design. These documents provide background information that will make it easier for you to implement in-app billing.

在你实现应用收费在您自己的应用程序中,要确保你的阅读应用收费的概述以及安全设计。这些文档提供背景信息,将使您可以更轻松地实现应用收费。

To implement in-app billing in your application, you need to do the following:

- 1.Download the in-app billing sample application.
- 2.Add the IMarketBillingService.aidl file to your project.
- 3.Update your AndroidManifest.xml file.
- 4.Create a Service and bind it to the MarketBillingService so your application can send billing requests and receive billing responses from Google Play.
- 5.Create a BroadcastReceiver to handle broadcast intents from Google Play.
- 6.Create a security processing component to verify the integrity of the transaction messages that are sent by Google Play.
- 7.Modify your application code to support in-app billing.

Downloading the Sample Application

实现应用收费在您的应用程序,您需要做到以下几点:

1. 下载示例应用程序的应用收费。
2. 添加IMarketBillingService。aidl文件到您的项目。
3. 更新你的AndroidManifest。xml文件。
4. 创建一个服务并将它绑定到MarketBillingService所以应用程序可以发送请求和接回应计费账单从谷歌玩。
5. 创建一个BroadcastReceiver从谷歌处理广播意图玩。
6. 创建一个安全处理组件来验证事务的完整性发送的消息由谷歌play。
7. 修改应用程序代码来支持应用收费。

下载示例应用程序

Downloading the Sample Application

The in-app billing sample application shows you how to perform several tasks that are common to all in-app billing implementations, including:

应用收费的示例应用程序向你展示了如何完成几个任务,都很普遍应用收费的实现,包括:

- Sending in-app billing requests to Google Play.
- Handling synchronous responses from Google Play.
- Handling broadcast intents (asynchronous responses) from Google Play.
- Using in-app billing security mechanisms to verify the integrity of billing responses.
- Creating a user interface that lets users select items for purchase.

The sample application includes an application file (`Dungeons.java`), the AIDL file for the MarketBillingService (`IMarketBillingService.aidl`), and several classes that demonstrate in-app billing messaging. It also includes a class that demonstrates basic security tasks, such as signature verification.

- 发送请求到谷歌play。
- 处理同步响应至google Play。
- 处理广播意图(异步响应)从谷歌 play。
- 使用安全机制应用收费来验证完整性计费的反应。
- 创建一个用户界面,用户可以选择购买的物品。

样例应用程序包含一个应用程序文件(地牢java),MarketBillingService AIDL文件(IMarketBillingService.aidl),还有几类,演示应用收费的消息传递。它还包括了一个类,演示了基本的安全的任务,如签名验证。

Table 1 lists the source files that are included with the sample application.

表1列出了源文件中包含的示例应用程序。

Table 1. In-app billing sample application source files.

表1应用收费示例应用程序源文件。

File	Description
<code>IMarketBillingService.aidl</code>	Android Interface Definition Library (AIDL) file that defines the IPC interface to Google Play's in-app billing service (MarketBillingService).
<code>PurchaseDatabase.java</code>	A local database for storing purchase information.
<code>BillingReceiver.java</code>	A BroadcastReceiver that receives asynchronous response messages (broadcast intents) from Google Play.

	Forwards all messages to the BillingService.
BillingService.java	A Service that sends messages to Google Play on behalf of the application by connecting (binding) to the MarketBillingService
ResponseHandler.java	A Handler that contains methods for updating the purchases database and the UI.
PurchaseObserver.java	An abstract class for observing changes related to purchases.
Security.java	Provides various security-related methods.
Consts.java	Defines various Google Play constants and sample application constants. All constants that are defined by Google Play must be defined the same way in your application
Base64.java and Base64DecoderException.java	Provides conversion services from binary to Base64 encoding. The Security class relies on these utility classes.
Dungeons.java	Sample application file that provides a UI for making purchases and displaying purchase history.

The in-app billing sample application is available as a downloadable component of the Android SDK. To download the sample application component, launch the Android SDK Manager and then select the Google Market Billing package component (see figure 1), and click Install Selected to begin the download.

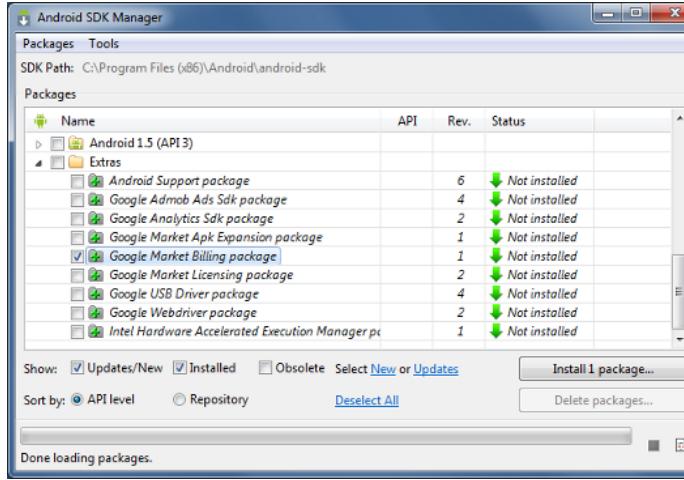


Figure 1. The Google Market Billing package contains the sample application and the AIDL file.

图1.谷歌的市场记帐包包含示例应用程序和AIDL文件。

When the download is complete, the Android SDK Manager saves the component into the following directory:

<sdk>/extras/google/market_billing/

If you want to see an end-to-end demonstration of in-app billing before you integrate in-app billing into your own application, you can build and run the sample application. Building and running the sample application involves three tasks:

- Configuring and building the sample application.
- Uploading the sample application to Google Play.
- Setting up test accounts and running the sample application.

下载完成后,Android SDK经理节省组件到以下目录:

< sdk > /配件/谷歌/市场账单/

如果你想看到一个端到端应用收费的示范应用收费之前你整合到您自己的应用程序,您可以构建和运行示例应用程序。构建和运行示例应用程序涉及到三个任务:

- 配置和构建这个示例应用程序。
- 上传示例应用程序谷歌玩。
- 设置测试帐户并运行样例应用程序。

Note: Building and running the sample application is necessary only if you want to see a demonstration of in-app billing. If you do not want to run the sample application, you can skip to the next section, Adding the AIDL file to your project.

注意:构建和运行示例应用程序是必须的,如果你想查看演示应用收费的。如果你不想运行样例应用程序,您可以跳过这一节,添加了AIDL文件到您的项目。

Configuring and building the sample application-配置和构建这个示例应用程序

Before you can run the sample application, you need to configure it and build it by doing the following:

在运行这个示例应用程序之前,您需要配置它,把它通过执行以下步骤:

1.Add your Google Play public key to the sample application code.

This enables the application to verify the signature of the transaction information that is returned from Google Play. To add your public key to the sample application code, do the following:

1.添加你的谷歌play公钥示例应用程序代码。

这使应用程序验证签名的事务信息返回的谷歌玩。添加您的公钥示例应用程序代码,做到以下几点:

1.Log in to your Google Play publisher account.

2.On the upper left part of the page, under your name, click Edit Profile.

3.On the Edit Profile page, scroll down to the Licensing & In-app Billing panel.

4.Copy your public key.

5.Open src/com/example/dungeons/Security.java in the editor of your choice.

You can find this file in the sample application's project folder.

6.Add your public key to the following line of code:

String base64EncodedPublicKey = "your public key here";

7.Save the file.

1.登录你的谷歌play发布者帐户。

2.左上角页面的一部分,在你的名字,点击编辑按钮。

3.在编辑页面,向下滚动到许可&应用收费面板。

4.复制你的公钥。

5. src / com/example/dungeons/security 打开。java 编辑器中的你的选择。

你可以找到这个文件在样例应用程序的项目文件夹。

6. 添加您的公钥以下代码:

字符串base64EncodedPublicKey = "您的公钥这里";

7. 保存文件。

2. Change the package name of the sample application. The current package name is com.example.dungeons. Google Play does not let you upload applications with package names that contain com.example, so you must change the package name to something else.

2. 改变包的名称的示例应用程序。

当前的包名是com例子地牢。谷歌play不让你上传应用程序和包名称中包含com。示例,因此您必须更改软件包名称到其他事情上。

3. Build the sample application in release mode and sign it. To learn how to build and sign applications, see Building and Running.

3. 构建样例应用程序在发布模式下并签字。

学习如何构建和签署应用程序,请参阅构建和运行。

Uploading the sample application-上传示例应用程序

After you build a release version of the sample application and sign it, you need to upload it as a draft to the Google Play publisher site. You also need to create a product list for the in-app items that are available for purchase in the sample application. The following instructions show you how to do this.

1. Upload the release version of the sample application to Google Play.

Do not publish the sample application; leave it as an unpublished draft application. The sample application is for demonstration purposes only and should not be made publicly available on Google Play. To learn how to upload an application to Google Play, see Uploading applications.

在您构建一个发布版本的示例应用程序并签字,你需要上传它作为草案到谷歌玩出版商网站。您还需要创建一个产品列表的应用内购买物品,在样例应用程序。下面的说明显示你如何做到这一点。

2. Create a product list for the sample application.

The sample application lets you purchase two items: a two-handed sword (sword_001) and a potion (potion_001). We recommend that you set up your product list so that sword_001 has a purchase type of "Managed per user account" and potion_001 has a purchase type of "Unmanaged" so you can see how these two purchase types behave. To learn how to set up a product list, see [Creating a Product List](#).

这个示例应用程序允许你购买的两个项目:一个双手剑(sword_001)和一种药剂(potion_001)。我们建议您设置您的产品列表,以便 sword_001有购买类型的"有管理的每个用户账户"和potion_001有购买类型的"托管",所以你可以看到这两个采购类型的行为。学习如何建立一个产品列表,请参阅创建一个产品列表.

Note: You must publish the items in your product list (sword_001 and potion_001) even though you are not publishing the sample application. Also, you must have a Google Wallet Merchant account to add items to the sample application's product list.

注意:你必须发布项目的产品列表(sword_001和potion_001)即使你不是出版样例应用程序。同时,你必须有一个谷歌钱包商人帐户将商品添加到样例应用程序的产品列表。

Running the sample application-运行样例应用程序

You cannot run the sample application in the emulator. You must install the sample application onto a device to run it. To run the sample application, do the following:

您不能运行模拟器中的样例应用程序。您必须安装示例应用程序到一个设备来运行它。要运行样例应用程序,请执行以下操作:

1. Make sure you have at least one test account registered under your Google Play publisher account. You cannot purchase items from yourself (Google Wallet prohibits this), so you need to create at least one test account that you can use to purchase items in the sample application. To learn how to set up a test account, see [Setting up Test Accounts](#).

1. 确保你有至少一个测试注册的账户在你的谷歌玩发布者帐户。你不能从自己购买一些物品(谷歌钱包禁止这个),所以您需要创建至少一个测试帐户,您可以使用它们来购买物品在样例应用程序。学习如何建立一个测试账号,请参阅设置测试账户。

2. Verify that your device is running a supported version of the Google Play application or the MyApps application. If your device is running Android 3.0, in-app billing requires version 5.0.12 (or higher) of the MyApps application. If your device is running any other version of Android, in-app billing requires version 2.3.4 (or higher) of the Google Play application. To learn how to check the version of the Google Play application, see [Updating Google Play](#).

2.确认你的设备正在运行一个受支持的版本的谷歌应用程序或应用程序MyApps玩。如果你的设备运行的是Android 3.0,应用收费5.0.12需要版本(或更高)的MyApps应用程序。如果你的设备运行的任何其他版本的Android,应用收费需要版本 2.3.4(或更高)的谷歌玩的应用程序。学习如何检查版本的谷歌玩应用程序,见更新谷歌玩。

3. Install the application onto your device. Even though you uploaded the application to Google Play, the application is not published, so you cannot download it from Google Play to a device. Instead, you must install the application onto your device. To learn how to install an application onto a device, see [Running on a](#)

device.

3. 将应用程序安装到您的设备。即使你上传应用程序以谷歌玩,应用程序就不会发布,所以你不能下载它从谷歌发挥一个设备。相反,您必须将应用程序安装到您的设备。学习如何安装一个应用程序到一个设备,请参阅在设备上运行。

4. Make one of your test accounts the primary account on your device. 让你的一个测试帐户主账户在你的设备上

The primary account on your device must be one of the test accounts that you registered on the Google Play publisher site. If the primary account on your device is not a test account, you must do a factory reset of the device and then sign in with one of your test accounts. To perform a factory reset, do the following:

在你的设备的主要帐户必须是一个测试你的账户,注册在谷歌玩出版商网站。如果主账户在你的设备不是一个测试账号,你必须做一个工厂复位的设备,然后登录你的某个测试账户。执行一个工厂复位,做到以下几点:

1. Open Settings on your device.
2. Touch Privacy.
3. Touch Factory data reset.
4. Touch Reset phone.
5. After the phone resets, be sure to sign in with one of your test accounts during the device setup process.

1. 打开设置你的设备上。

2. 触摸隐私。

3. 触摸工厂数据重置。

3. 触摸复位的电话。

4. 电话后重置,一定要签上名在与您的测试账户在设备安装过程。

5. 运行应用程序并购买刀和药水。

5. Run the application and purchase the sword or the potion. When you use a test account to purchase items, the test account is billed through Google Wallet and your Google Wallet Merchant account receives a payout for the purchase. Therefore, you may want to refund purchases that are made with test accounts, otherwise the purchases will show up as actual payouts to your merchant account.

5. 当您使用一个测试账号购买项目,测试帐户被谷歌钱包,你通过谷歌钱包商人帐户收到付款购买。因此,您可能想要退款采购与测试账户,否则购买将会显示为实际支付给你的商家帐户。

Note: Debug log messages are turned off by default in the sample application. You can turn them on by setting the variable DEBUG to true in the `Consts.java` file. 注意: 调试日志消息是默认关闭了在样本应用程序。你可以把它们在通过设置变量为true在Consts.java文件。

Adding the AIDL file to your project-添加了AIDL文件到您的项目

The sample application contains an Android Interface Definition Language (AIDL) file, which defines the interface to Google Play's in-app billing service (`MarketBillingService`). When you add this file to your project, the Android build environment creates an interface file (`IMarketBillingService.java`). You can then use this interface to make billing requests by invoking IPC method calls.

If you are using the ADT plug-in with Eclipse, you can just add this file to your `/src` directory. Eclipse will automatically generate the interface file when you build your project (which should happen immediately). If you are not using the ADT plug-in, you can put the AIDL file into your project and use the Ant tool to build your project so that the `IMarketBillingService.java` file gets generated.

To add the `IMarketBillingService.aidl` file to your project, do the following:

Create the following directory in your application's `/src` directory: `com/android/vending/billing/`

Copy the `IMarketBillingService.aidl` file into the `sample/src/com/android/vending/billing/` directory. Build your application. You should now find a generated interface file named `IMarketBillingService.java` in the `gen` folder of your project.

Updating Your Application's Manifest 样例应用程序包含了一个Android接口定义语言(AIDL)文件, 它定义了接口来谷歌Play的应用收费服务 (`MarketBillingService`)。当你添加这个文件到您的项目, Android构建环境创建一个接口文件 (`IMarketBillingService.java`)。然后, 您可以使用这个接口使计费请求通过调用IPC方法调用。

如果您正在使用Eclipse ADT插件, 你可以添加这个文件到你的`/ src`目录。Eclipse将自动生成的接口文件当您构建您的项目(应该立即发生)。如果你没有使用ADT插件, 你可以把AIDL文件到您的项目中, 使用Ant工具来构建您的项目, 以便`IMarketBillingService. java`文件中生成的。

添加`IMarketBillingService. aidl`文件到你的项目中, 做到以下几点:

创建下列目录在您的应用程序`/ src`目录:

`com/android/vending/billing/`

`IMarketBillingService`复制。`aidl`文件到样本`/ src / com/android/vending/billing/`目录。

构建自己的应用程序。

你现在应该找一个生成的接口文件命名为`IMarketBillingService. java`在创文件夹的项目。

更新您的应用程序的清单

In-app billing relies on the Google Play application, which handles all communication between your application and the Google Play server. To use the Google Play application, your application must request the proper permission. You can do this by adding the com.android.vending.BILLING permission to your AndroidManifest.xml file. If your application does not declare the in-app billing permission, but attempts to send billing requests, Google Play will refuse the requests and respond with a RESULT_DEVELOPER_ERROR response code.

In addition to the billing permission, you need to declare the BroadcastReceiver that you will use to receive asynchronous response messages (broadcast intents) from Google Play, and you need to declare the Service that you will use to bind with the IMarketBillingService and send messages to Google Play. You must also declare intent filters for the BroadcastReceiver so that the Android system knows how to handle the broadcast intents that are sent from the Google Play application.

For example, here is how the in-app billing sample application declares the billing permission, the BroadcastReceiver, the Service, and the intent filters. In the sample application, BillingReceiver is the BroadcastReceiver that handles broadcast intents from the Google Play application and BillingService is the Service that sends requests to the Google Play application.

应用收费依赖于谷歌玩应用程序,该应用程序处理所有通信和谷歌之间扮演服务器。使用谷歌玩的应用程序,您的应用程序必须请求适当的许可。为此,您可以添加com android自动售货。允许你的AndroidManifest计费。xml文件。如果你的应用程序没有声明应用收费许可,但是试图发送帐单请求,谷歌 玩会拒绝请求和响应,结果开发人员错误响应代码。

除了计费许可,您需要声明BroadcastReceiver,您将用于接收异步响应消息(广播意图)从谷歌玩耍,您需要声明服务,您将使用绑定与 IMarketBillingService和发送信息到谷歌玩。你还必须声明意图过滤器BroadcastReceiver这样Android系统知道 如何处理广播意图所发出的谷歌玩的应用程序。

例如,下面是示例应用程序如何应用收费的声明了计费许可,BroadcastReceiver,服务,和意图过滤器。在示例应用程序 中,BillingReceiver是BroadcastReceiver处理从谷歌广播意图玩应用和BillingService是服务发送请求到谷歌 玩的应用程序。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.dungeons"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-permission android:name="com.android.vending.BILLING" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Dungeons" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

```
<service android:name="BillingService" />
<receiver android:name="BillingReceiver">
    <intent-filter>
        <action android:name="com.android.vending.billing.IN_APP_NOTIFY" />
        <action android:name="com.android.vending.billing.RESPONSE_CODE" />
        <action android:name="com.android.vending.billing.PURCHASE_STATE_CHANGED" />
    </intent-filter>
</receiver>
</application>
</manifest>
```

Creating a Local Service

Your application must have a local Service to facilitate messaging between your application and Google Play. At a minimum, this service must do the following:

Bind to the MarketBillingService. Send billing requests (as IPC method calls) to the Google Play application. The five types of billing requests include:
CHECK_BILLING_SUPPORTED requests
REQUEST_PURCHASE requests
GET_PURCHASE_INFORMATION requests
CONFIRM_NOTIFICATIONS requests
RESTORE_TRANSACTIONS requests
Handle the synchronous response messages that are returned with each billing request. Binding to the MarketBillingService
Binding to the MarketBillingService is relatively easy if you've already added the IMarketBillingService.aidl file to your project. The following code sample shows how to use the bindService() method to bind a service to the MarketBillingService. You could put this code in your service's onCreate() method.

创建一个本地服务

您的应用程序必须有一个本地服务来促进应用程序间的消息传递和谷歌玩。至少,这个服务必须做到以下几点:

MarketBillingService绑定。

发送帐单请求(如IPC方法调用)到谷歌玩的应用程序。这五个类型的计费请求包括:

检查账单支持请求

请求购买请求

获得购买信息请求

确认通知请求

恢复交易请求

处理同步响应消息返回与每个结算请求。

绑定到MarketBillingService

绑定到MarketBillingService是比较容易的,如果您已经添加了IMarketBillingService。aidl文件到您的项目。下面的代码示例显示了如何用bindService()方法来绑定一个服务MarketBillingService。你可以把这段代码在您的服务的onCreate()方法。

```

try {
    boolean bindResult = mContext.bindService(
        new Intent("com.android.vending.billing.MarketBillingService.BIND"), this,
        Context.BIND_AUTO_CREATE);
    if (bindResult) {
        Log.i(TAG, "Service bind successful.");
    } else {
        Log.e(TAG, "Could not bind to the MarketBillingService.");
    }
} catch (SecurityException e) {
    Log.e(TAG, "Security exception: " + e);
}

/**
 * The Android system calls this when we are connected to the MarketBillingService.
 */
public void onServiceConnected(ComponentName name, IBinder service) {
    Log.i(TAG, "MarketBillingService connected.");
    mService = IMarketBillingService.Stub.asInterface(service);
}

```

You can now use the mService reference to invoke the sendBillingRequest() method.

For a complete implementation of a service that binds to the MarketBillingService, see the BillingService class in the sample application.

Sending billing requests to the MarketBillingService Now that your Service has a reference to the IMarketBillingService interface, you can use that reference to send billing requests (via IPC method calls) to the MarketBillingService. The MarketBillingService IPC interface exposes a single public method (sendBillingRequest()), which takes a single Bundle parameter. The Bundle that you deliver with this method specifies the type of request you want to perform, using various key-value pairs. For instance, one key indicates the type of request you are making, another indicates the item being purchased, and another identifies your application. The sendBillingRequest() method immediately returns a Bundle containing an initial response code. However, this is not the complete purchase response; the complete response is delivered with an asynchronous broadcast intent. For more information about the various Bundle keys that are supported by the MarketBillingService, see In-app Billing Service Interface.

You can use the sendBillingRequest() method to send five types of billing requests. The five request types are specified using the BILLING_REQUEST Bundle key. This Bundle key can have the following five values:

CHECK_BILLING_SUPPORTED—verifies that the Google Play application supports in-app billing and the version of the In-app Billing API available.

REQUEST_PURCHASE—sends a purchase request for an in-app item. GET_PURCHASE_INFORMATION—retrieves transaction information for a purchase or refund.

CONFIRM_NOTIFICATIONS—acknowledges that you received the transaction information for a purchase or refund. RESTORE_TRANSACTIONS—retrieves a user's

transaction history for managed purchases. To make any of these billing requests, you first need to build an initial Bundle that contains the three keys that are required for all requests: BILLING_REQUEST, API_VERSION, and PACKAGE_NAME. The following code sample shows you how to create a helper method named `makeRequestBundle()` that does this.

您现在可以使用`mService`引用调用`sendBillingRequest()`方法。

对于一个完整的实现一个服务,绑定的MarketBillingService,看到BillingService类的示例应用程序。

发送请求到MarketBillingService计费

现在,你的服务有一个IMarketBillingService接口的引用,您可以使用这个引用发送帐单请求(通过IPC方法调用)到 MarketBillingService。MarketBillingService IPC接口公开的一个公共方法(`sendBillingRequest()`),它接受一个Bundle参数。这个包交付使用此方法指定类型的请求,您希望执行,使用不同的键值对。例如,一个关键的指示请求类型的你正在制作,另一个显示项目被收购,和另一个标识您的应用程序。立即的 `sendBillingRequest()`方法返回一个包包含一个初始响应代码。然而,这并不是完整的购买响应;完整的反应是交付与异步广播的意图。了解 更多关于美国各种包键所支持MarketBillingService,看到应用收费服务接口。

您可以使用`sendBillingRequest()`方法发送请求的五种类型的账单。五个请求类型指定为使用计费请求绑定包的关键。这束键可以有以下5个值:

检查账单支持验证的应用程序支持谷歌玩应用收费和版本的应用收费API可用。

委托收购发送一个购买请求的应用程序项目。

获得购买信息检索事务信息一个购买或退款。

确认通知承认您收到的事务信息一个购买或退款。

恢复交易检索用户的交易历史购买管理。

要做出任一这些账单请求,您首先需要构建初始的包,包含了三个键所需要的所有请求:账单请求,api版本,和PACKAGE_NAME。下面的代码示例显示了如何创建一个辅助方法命名`makeRequestBundle()`,这是否。

```
protected Bundle makeRequestBundle(String method) {
    Bundle request = new Bundle();
    request.putString(BILLING_REQUEST, method);
    request.putInt(API_VERSION, 1);
    request.putString(PACKAGE_NAME, getPackageName());
    return request;
}
```

To use this helper method, you pass in a String that corresponds to one of the five types of billing requests. The method returns a Bundle that has the three

required keys defined. The following sections show you how to use this helper method when you send a billing request.

Important: You must make all in-app billing requests from your application's main thread.

Verifying that in-app billing is supported (`CHECK BILLING_SUPPORTED`) The following code sample shows how to verify whether the Google Play application supports in-app billing and confirm what version of the API it supports. In the sample, `mService` is an instance of the `MarketBillingService` interface. 使用这个 `helper` 方法, 您传递一个字符串相对应的五种计费请求。该方法返回一个 `Bundle`, 有三个定义所需的密钥。下面几节将向您展示如何使用这个 `helper` 方法当您发送一个账单请求。

重要: 您必须使所有应用收费请求从应用程序的主线程。

验证应用收费是支持(`CHECK_BILLING_SUPPORTED`)

以下代码示例展示了如何验证应用程序支持谷歌是否发挥应用收费和确认什么版本的API, 它支持。在此示例中, `mService`是`MarketBillingService`接口的一个实例。

```
/***
 * Request type is CHECK_BILLING_SUPPORTED
 */
Bundle request = makeRequestBundle("CHECK_BILLING_SUPPORTED");
Bundle response = mService.sendBillingRequest(request);
// Do something with this response.
}
```

The `makeRequestBundle()` method constructs an initial `Bundle`, which contains the three keys that are required for all requests: `BILLING_REQUEST`, `API_VERSION`, and `PACKAGE_NAME`. If you are offering subscriptions in your app, set the `API_VERSION` key to a value of "2", to confirm that In-app Billing v2 is available. For an example, see Subscriptions.

The `CHECK_BILLING_SUPPORTED` request returns a synchronous `Bundle` response, which contains only a single key: `RESPONSE_CODE`. The `RESPONSE_CODE` key can have the following values:

`RESULT_OK`—the specified version of in-app billing is supported. `RESULT_BILLING_UNAVAILABLE`—in-app billing is not available because the API version you specified is not recognized or the user is not eligible to make in-app purchases (for example, the user resides in a country that prohibits in-app purchases). `RESULT_ERROR`—there was an error connecting with the Google Play application. `RESULT_DEVELOPER_ERROR`—the application is trying to make an in-app billing request but the application has not declared the `com.android.vending.BILLING` permission in its manifest. Can also indicate that an application is not properly signed, or that you sent a malformed request. The `CHECK_BILLING_SUPPORTED` request does not trigger any asynchronous responses (broadcast intents).

We recommend that you invoke the `CHECK_BILLING_SUPPORTED` request within a `RemoteException` block. When your code throws a `RemoteException` it indicates that the remote method call failed, which means that the Google Play application is out of date and needs to be updated. In this case, you can provide users with an error message that contains a link to the Updating Google Play Help topic.

The sample application demonstrates how you can handle this error condition (see DIALOG_CANNOT_CONNECT_ID in Dungeons.java).

Making a purchase request (REQUEST_PURCHASE) To make a purchase request you must do the following:

Send the REQUEST_PURCHASE request. Launch the PendingIntent that is returned from the Google Play application. Handle the broadcast intents that are sent by the Google Play application. Making the request You must specify four keys in the request Bundle. The following code sample shows how to set these keys and make a purchase request for a single in-app item. In the sample, mProductId is the Google Play product ID of an in-app item (which is listed in the application's product list), and mService is an instance of the MarketBillingService interface.

这个makeRequestBundle()方法构造一个初始包,其中包含了三个键所需要的所有请求:账单请求,api版本,和 PACKAGE_NAME。如果你在自己的应用程序提供订阅,设置api版本的关键,值为“2”,证实应用收费v2是可用的。对于一个examnple,看到订阅。

检查账单支持请求返回一个同步包响应,其中只包含一个键:响应代码。响应码键可以有以下值:

结果可以specified版本的应用收费是支持的。

结果计费不可用,在app账单不使用,因为您指定API版本并不承认或用户没有资格使应用内购买(例如,用户驻留在一个国家禁止应用内购买)。

结果误差有一个错误,结合谷歌玩的应用程序。

结果开发人员错误应用程序试图使一个应用收费应用程序请求,但尚未宣布com android自动售货。在manifest权限计费。也可以显示一个应用程序签署的是不恰当的,或者你送来了一个畸形的请求。

检查账单支持请求不会触发任何异步响应(广播意图)。

我们建议您调用检查账单支持要求在一个RemoteException块。当你的代码抛出RemoteException它表明远程方法调用失败,这意味着谷歌玩应用程序已经过时,需要更新。在这种情况下,您可以为用户提供一个错误消息,包含一个链接到更新谷歌play帮助主题。

示例应用程序演示了如何处理这个错误条件(看到对话框不能连接id在地牢java)。

购物的请求(请求购买)

购买请求你必须做到以下几点:

购买请求发送请求。

启动PendingIntent返回谷歌玩的应用程序。

处理广播意图所发送的谷歌玩的应用程序。

发出请求

您必须指定4个键在请求包。下面的代码示例显示了如何设置这些键,使购买请求为单一应用程序项目。在此示例中,mProductId是谷歌 的一个应用内玩产品ID的项目(这是列在应用程序的产品列表),mService是MarketBillingService接口的一个实例。

```
/***
 * Request type is REQUEST_PURCHASE
 */
Bundle request = makeRequestBundle("REQUEST_PURCHASE");
request.putString(ITEM_ID, mProductId);
// Request is for a standard in-app product
request.putString(ITEM_TYPE, "inapp");
// Note that the developer payload is optional.
if (mDeveloperPayload != null) {
    request.putString(DEVELOPER_PAYLOAD, mDeveloperPayload);
}
Bundle response = mService.sendBillingRequest(request);
// Do something with this response.
```

The makeRequestBundle() method constructs an initial Bundle, which contains the three keys that are required for all requests: BILLING_REQUEST, API_VERSION, and PACKAGE_NAME. The ITEM_ID key is then added to the Bundle prior to invoking the sendBillingRequest() method.

The request returns a synchronous Bundle response, which contains three keys: RESPONSE_CODE, PURCHASE_INTENT, and REQUEST_ID. The RESPONSE_CODE key provides you with the status of the request and the REQUEST_ID key provides you with a unique request identifier for the request. The PURCHASE_INTENT key provides you with a PendingIntent, which you can use to launch the checkout UI.

Using the pending intent How you use the pending intent depends on which version of Android a device is running. On Android 1.6, you must use the pending intent to launch the checkout UI in its own separate task instead of your application's activity stack. On Android 2.0 and higher, you can use the pending intent to launch the checkout UI on your application's activity stack. The following code shows you how to do this. You can find this code in the PurchaseObserver.java file in the sample application. 这个makeRequestBundle()方法构造一个初始包,其中包含了三个键所需要的所有请求:账单请求,api版本,和 PACKAGE_NAME。然后,添加的ITEM_ID关键Bundle的调用sendBillingRequest之前()方法。

请求返回一个同步包响应,其中包含三个键:响应代码,购买意向,并请求id。响应码键为您提供请求状态和请求id为你提供了一个独特的请求标识符的请求。购买意图键为你提供了一个PendingIntent,你可以用它启动checkout UI。

使用悬而未决的意图

你如何使用这个悬而未决的意图取决于哪个版本的Android设备的运行。Android 1.6系统,您必须使用悬而未决的意图推出checkout UI在其自己的单独的任务而不是应用程序的活动堆栈。在Android 2.0和更高版本,您可以使用悬而未决的意图推出checkout UI应用程序的活动堆栈。下面的代码显示了你如何做到这一点。你可以找到

这段代码在PurchaseObserver。在示例应用程序的java文件。

```

void startBuyPageActivity(PendingIntent pendingIntent, Intent intent) {
    if (mStartIntentSender != null) {
        // This is on Android 2.0 and beyond. The in-app checkout page activity
        // will be on the activity stack of the application.
        try {
            // This implements the method call:
            // mActivity.startIntentSender(pendingIntent.getIntentSender(),
            //                             intent, 0, 0, 0);
            mStartIntentSenderArgs[0] = pendingIntent.getIntentSender();
            mStartIntentSenderArgs[1] = intent;
            mStartIntentSenderArgs[2] = Integer.valueOf(0);
            mStartIntentSenderArgs[3] = Integer.valueOf(0);
            mStartIntentSenderArgs[4] = Integer.valueOf(0);
            mStartIntentSender.invoke(mActivity, mStartIntentSenderArgs);
        } catch (Exception e) {
            Log.e(TAG, "error starting activity", e);
        }
    } else {
        // This is on Android 1.6. The in-app checkout page activity will be on its
        // own separate activity stack instead of on the activity stack of
        // the application.
        try {
            pendingIntent.send(mActivity, 0 /* code */, intent);
        } catch (CanceledException e) {
            Log.e(TAG, "error starting activity", e);
        }
    }
}

```

Important: You must launch the pending intent from an activity context and not an application context. Also, you cannot use the singleTop launch mode to launch the pending intent. If you do either of these, the Android system will not attach the pending intent to your application process. Instead, it will bring Google Play to the foreground, disrupting your application.

Handling broadcast intents A REQUEST_PURCHASE request also triggers two asynchronous responses (broadcast intents). First, the Google Play application sends a RESPONSE_CODE broadcast intent, which provides error information about the request. If the request does not generate an error, the RESPONSE_CODE broadcast intent returns RESULT_OK, which indicates that the request was successfully sent. (To be clear, a RESULT_OK response does not indicate that the requested purchase was successful; it indicates that the request was sent successfully to Google Play.)

Next, when the requested transaction changes state (for example, the purchase is successfully charged to a credit card or the user cancels the purchase), the Google Play application sends an IN_APP_NOTIFY broadcast intent. This message contains a notification ID, which you can use to retrieve the transaction details for the REQUEST_PURCHASE request.

Note: The Google Play application also sends an IN_APP_NOTIFY for refunds. For more information, see [Handling IN_APP_NOTIFY messages](#).

Because the purchase process is not instantaneous and can take several seconds (or more), you must assume that a purchase request is pending from the time you receive a RESULT_OK message until you receive an IN_APP_NOTIFY message for the transaction. While the transaction is pending, the Google Play checkout UI displays an "Authorizing purchase..." notification; however, this notification is dismissed after 60 seconds and you should not rely on this notification as your primary means of conveying transaction status to users. Instead, we recommend that you do the following:

Add an Activity to your application that shows users the status of pending and completed in-app purchases. Use a status bar notification to keep users informed about the progress of a purchase. To use these two UI elements, you could invoke a status bar notification with a ticker-text message that says "Purchase pending" when your application receives a RESULT_OK message. Then, when your application receives an IN_APP_NOTIFY message, you could update the notification with a new message that says "Purchase succeeded" or "Purchase failed." When a user touches the expanded status bar notification, you could launch the activity that shows the status of pending and completed in-app purchases.

If you use some other UI technique to inform users about the state of a pending transaction, be sure that your pending status UI does not block your application. For example, you should avoid using a hovering progress wheel to convey the status of a pending transaction because a pending transaction could last a long time, particularly if a device loses network connectivity and cannot receive transaction updates from Google Play.

Important: If a user purchases a managed item, you must prevent the user from purchasing the item again while the original transaction is pending. If a user attempts to purchase a managed item twice, and the first transaction is still pending, Google Play will display an error to the user; however, Google Play will not send an error to your application notifying you that the second purchase request was canceled. This might cause your application to get stuck in a pending state while it waits for an IN_APP_NOTIFY message for the second purchase request.

Retrieving transaction information for a purchase or refund (GET_PURCHASE_INFORMATION) You retrieve transaction information in response to an IN_APP_NOTIFY broadcast intent. The IN_APP_NOTIFY message contains a notification ID, which you can use to retrieve transaction information.

To retrieve transaction information for a purchase or refund you must specify five keys in the request Bundle. The following code sample shows how to set these keys and make the request. In the sample, mService is an instance of the MarketBillingService interface.

重要:您必须启动悬而未决的意图与一个活动上下文,而不是一个应用程序上下文。同样,你不能使用singleTop启动模式启动悬而未决的意图。如果你做的不是这些,Android系统将不把你的申请程序未决的意图。相反,它将使谷歌玩到前台,扰乱了你的应用程序。

处理广播意图

购买请求的请求也触发了两个异步响应(广播意图)。首先,谷歌玩应用程序发送一个响应代码广播的意图,它提供了错误信息的请求。如果请求并 不会生成一个错误,响应代码返回的结果可以播放的意图,这表明,他的请求被成功发送。(需要明确的是,一个ok响应结果并不表明请求的收购是成功的,它表 明请求发送成功,谷歌play。)

接下来,当请求的事务状态更改(例如,购买被成功地用于信用卡或用户取消购买),谷歌应用程序发送一个玩在应用程序通知广播的意图。此消息包含一个通知ID,您可以 使用它来检索事务细节为请求购买请求。

注意:谷歌玩应用程序还在应用程序发送一个通知的退款。有关更多信息,请参见**notify**消息处理程序。

由于收购过程不是瞬时和花几秒(或更多),你必须假定一个购买请求等待时间从你获得结果好的消息,直到你收到一个通知消息在应用程序的事务。尽管交易正在 等待,谷歌玩**checkout UI**显示一个“授权购买……”通知;然而,这种通知是驳回了60秒后,你不应该依赖于这种通知作为你主要的交易状态的方法向用户传达。相反,我们建议您做 到以下几点:

对您的应用程序添加一项活动,显示了用户等待完成的状态和应用内购买。

使用一个状态栏通知来让用户了解进程的购买。

使用这两个UI元素,您可以调用一个状态栏通知与一个醒目的短信,说“购买等待”当你的应用程序接收结果好信息。然后,当应用程序收到一个通知消息的应用程序,您可以更新通知的新消息,说“购买成功”或者“购买失败了。”当用户触摸扩展状态栏通知,您可以启动活动显示状态的未决,完成了应用内购买。

如果你使用一些其他的UI技术来通知用户等待事务的状态,确保你的暂挂状态UI不会阻塞你的应用程序。例如,您应该避免使用一个盘旋的进步车轮来传达一个悬而未决的事务的状态,因为一个悬而未决的事务可能会持续一段时间,特别是如果一个设备丧失网络连通性和不能接收事务更新从谷歌玩。

重要:如果一个用户购买一个托管的项,您必须防止用户购买物品时又原有的交易正在等待。如果用户试图购买一个托管项目两次,第一次交易仍悬而未决,谷歌玩将显示一条错误用户;然而,谷歌游戏都不会对您的应用程序发送一个错误通知你第二次购买请求已被取消。这可能导致应用程序会被困在一个未决状态,它会等待一个在应用程序通知消息的第二次购买请求。

检索事务信息一个购买或退款(获得购买信息)

您检索事务信息以响应一个在应用程序通知广播的意图。在应用程序的通知消息包含一个通知ID,您可以使用它来检索事务信息。

检索事务信息一个购买或退款,你必须在请求中指定的5个关键Bundle。下面的代码示例显示了如何设置这些键和请求。在此示例中,mService是MarketBillingService接口的一个实例。

```
/**  
 * Request type is GET_PURCHASE_INFORMATION  
 */  
Bundle request = makeRequestBundle( "GET_PURCHASE_INFORMATION" );  
request.putLong( REQUEST_NONCE, mNonce );  
request.putStringArray( NOTIFY_IDS, mNotifyIds );  
Bundle response = mService.sendBillingRequest( request );  
// Do something with this response.  
}
```

The `makeRequestBundle()` method constructs an initial Bundle, which contains the three keys that are required for all requests: BILLING_REQUEST, API_VERSION, and PACKAGE_NAME. The additional keys are then added to the bundle prior to invoking the `sendBillingRequest()` method. The REQUEST_NONCE key contains a cryptographically secure nonce (number used once) that you must generate. The Google Play application returns this nonce with the PURCHASE_STATE_CHANGED broadcast intent so you can verify the integrity of the transaction information. The NOTIFY_IDS key contains an array of notification IDs, which you received in the IN_APP_NOTIFY broadcast intent.

The request returns a synchronous Bundle response, which contains two keys: RESPONSE_CODE and REQUEST_ID. The RESPONSE_CODE key provides you with

the status of the request and the REQUEST_ID key provides you with a unique request identifier for the request.

A GET_PURCHASE_INFORMATION request also triggers two asynchronous responses (broadcast intents). First, the Google Play application sends a RESPONSE_CODE broadcast intent, which provides status and error information about the request. Next, if the request was successful, the Google Play application sends a PURCHASE_STATE_CHANGED broadcast intent. This message contains detailed transaction information. The transaction information is contained in a signed JSON string (unencrypted). The message includes the signature so you can verify the integrity of the signed string.

Acknowledging transaction information (CONFIRM_NOTIFICATIONS) To acknowledge that you received transaction information you send a CONFIRM_NOTIFICATIONS request. You must specify four keys in the request Bundle. The following code sample shows how to set these keys and make the request. In the sample, mService is an instance of the MarketBillingService interface. 这个makeRequestBundle()方法构造一个初始包,其中包含了三个键所需要的的所有请求:账单请求,api版本,和 PACKAGE_NAME。额外的键然后添加到包调用sendBillingRequest之前()方法。请求nonce键包含密码安全nonce(号码 使用一次),您必须生成。谷歌的玩的应用程序返回这个nonce与采购状态改变了广播的意图,这样你就可以验证事务信息的完整性。此通知IDs键包含一组 通知IDs,你收到了在应用程序通知广播的意图。

请求返回一个同步包响应,其中包含两个键:响应代码和请求id。响应码键为您提供请求状态和请求id为你提供了一个独特的请求标识符的请求。

一个get请求购买信息也触发了两个异步响应(广播意图)。首先,谷歌玩应用程序发送一个响应代码广播的意图,它提供了状态和错误信息的请求。接下来,如果请求是成功的,谷歌玩应用程序发送一个购买状态改变了广播的意图。这个信息包含详细的事务信息。事务信息包含在一个签署了JSON字符串(加密)。消息 包含签名,这样你就可以验证完整性的签名字串。

承认事务信息(确认通知)

承认你收到事务信息您发送一个确认通知请求。您必须指定4个键在请求包。下面的代码示例显示了如何设置这些键和请求。在此示例中,mService是MarketBillingService接口的一个实例。

```
/*
 * Request type is CONFIRM_NOTIFICATIONS
 */
Bundle request = makeRequestBundle("CONFIRM_NOTIFICATIONS");
request.putStringArray(NOTIFY_IDS, mNotifyIds);
Bundle response = mService.sendBillingRequest(request);
// Do something with this response.
}
```

The makeRequestBundle() method constructs an initial Bundle, which contains the three keys that are required for all requests: BILLING_REQUEST, API_VERSION, and PACKAGE_NAME. The additional NOTIFY_IDS key is then added to the bundle prior to invoking the sendBillingRequest() method. The NOTIFY_IDS key contains an array of notification IDs, which you received in an IN_APP_NOTIFY broadcast intent and also used in a GET_PURCHASE_INFORMATION request.

The request returns a synchronous Bundle response, which contains two keys: RESPONSE_CODE and REQUEST_ID. The RESPONSE_CODE key provides you with

the status of the request and the REQUEST_ID key provides you with a unique request identifier for the request.

A CONFIRM_NOTIFICATIONS request triggers a single asynchronous response—a RESPONSE_CODE broadcast intent. This broadcast intent provides status and error information about the request.

You must send a confirmation when you receive transaction information from Google Play. If you don't send a confirmation message, Google Play will continue sending IN_APP_NOTIFY messages for the transactions you have not confirmed. Also, your application must be able to handle IN_APP_NOTIFY messages that contain multiple orders.

In addition, as a best practice, you should not send a CONFIRM_NOTIFICATIONS request for a purchased item until you have delivered the item to the user. This way, if your application crashes or something else prevents your application from delivering the product, your application will still receive an IN_APP_NOTIFY broadcast intent from Google Play indicating that you need to deliver the product.

Restoring transaction information (RESTORE_TRANSACTIONS) To restore a user's transaction information, you send a RESTORE_TRANSACTIONS request. You must specify four keys in the request Bundle. The following code sample shows how to set these keys and make the request. In the sample, mService is an instance of the MarketBillingService interface.

这个makeRequestBundle()方法构造一个初始包,其中包含了三个键所需要的所有请求:账单请求,api版本,和 PACKAGE_NAME。额外的通知ids键然后添加到包调用sendBillingRequest之前()方法。此通知IDs键包含一组通知IDs中 接收到一个在应用程序通知广播的意图,也可用于一个get请求购买信息。

请求返回一个同步包响应,其中包含两个键:响应代码和请求id。响应码键为您提供请求状态和请求id为你提供了一个独特的请求标识符的请求。

确认通知请求触发一个异步响应的响应代码广播的意图。这个广播意图提供了状态和错误信息的请求。

你必须发送一条确认当你接收事务信息从谷歌玩。如果你不发送一条确认消息,谷歌比赛将继续进行,在app发送通知消息的事务你还没有证实。同样,您的应用程序必须能够处理notify消息在应用程序包含多个命令。

此外,作为一个最佳实践,您不应该发送一个确认通知要求购买的商品,直到你们救项目给用户。这样,如果你的应用程序崩溃或者别的什么东西阻止你的应用程序交付产品,您的应用程序仍然可以接收到一个在应用程序通知广播意图与谷歌play表明你需要交付产品。

恢复交易信息(恢复事务)

恢复一个用户的事务的信息,您发送一个恢复交易请求。您必须指定4个键在请求包。下面的代码示例显示了如何设置这些键和请求。在此示例中,mService是MarketBillingService接口的一个实例。

```
* Request type is RESTORE_TRANSACTIONS
*/
Bundle request = makeRequestBundle("RESTORE_TRANSACTIONS");
request.putLong(REQUEST_NONCE, mNonce);
Bundle response = mService.sendBillingRequest(request);
// Do something with this response.
}
```

The `makeRequestBundle()` method constructs an initial Bundle, which contains the three keys that are required for all requests: BILLING_REQUEST, API_VERSION, and PACKAGE_NAME. The additional REQUEST_NONCE key is then added to the bundle prior to invoking the `sendBillingRequest()` method. The REQUEST_NONCE key contains a cryptographically secure nonce (number used once) that you must generate. The Google Play application returns this nonce with the transactions information contained in the PURCHASE_STATE_CHANGED broadcast intent so you can verify the integrity of the transaction information.

The request returns a synchronous Bundle response, which contains two keys: RESPONSE_CODE and REQUEST_ID. The RESPONSE_CODE key provides you with the status of the request and the REQUEST_ID key provides you with a unique request identifier for the request.

A RESTORE_TRANSACTIONS request also triggers two asynchronous responses (broadcast intents). First, the Google Play application sends a RESPONSE_CODE broadcast intent, which provides status and error information about the request. Next, if the request was successful, the Google Play application sends a PURCHASE_STATE_CHANGED broadcast intent. This message contains the detailed transaction information. The transaction information is contained in a signed JSON string (unencrypted). The message includes the signature so you can verify the integrity of the signed string.

Note: You should use the RESTORE_TRANSACTIONS request type only when your application is installed for the first time on a device or when your application has been removed from a device and reinstalled.

Other service tasks You may also want your Service to receive intent messages from your BroadcastReceiver. You can use these intent messages to convey the information that was sent asynchronously from the Google Play application to your BroadcastReceiver. To see an example of how you can send and receive these intent messages, see the `BillingReceiver.java` and `BillingService.java` files in the sample application. You can use these samples as a basis for your own implementation. However, if you use any of the code from the sample application, be sure you follow the guidelines in Security and Design.

Creating a BroadcastReceiver

The Google Play application uses broadcast intents to send asynchronous billing responses to your application. To receive these intent messages, you need to create a BroadcastReceiver that can handle the following intents:

`com.android.vending.billing.RESPONSE_CODE` This broadcast intent contains a Google Play response code, and is sent after you make an in-app billing request. For more information about the response codes that are sent with this response, see Google Play Response Codes for In-app Billing.

`com.android.vending.billing.IN_APP_NOTIFY` This response indicates that a purchase has changed state, which means a purchase succeeded, was canceled, or was refunded. For more information about notification messages, see In-app Billing Broadcast Intents

`com.android.vending.billing.PURCHASE_STATE_CHANGED` This broadcast intent contains detailed information about one or more transactions. For more information about purchase state messages, see In-app Billing Broadcast Intents

Each of these broadcast intents provide intent extras, which your BroadcastReceiver must handle. The intent extras are listed in the following table (see table 1).

Table 1. Description of broadcast intent extras that are sent in response to billing requests. 这个`makeRequestBundle()`方法构造一个初始包,其中包含了三个键所需要的所有请求:账单请求,api版本,和`PACKAGE_NAME`。额外的请求`nonce`关键即添加到包调用`sendBillingRequest之前()`方法。请求`nonce`键包含密码安全`nonce`(号码使用一次),您必须生成。谷歌的玩的应用程序返回这个`nonce`与交易信息包含在采购状态改变了广播的意图,这样你就可以验证事务信息的完整性。

请求返回一个同步包响应,其中包含两个键:响应代码和请求id。响应码键为您提供请求状态和请求id为你提供了一个独特的请求标识符的请求。

一个恢复交易请求也触发了两个异步响应(广播意图)。首先,谷歌玩应用程序发送一个响应代码广播的意图,它提供了状态和错误信息的请求。接下来,如果请求是成功的,谷歌玩应用程序发送一个购买状态改变了广播的意图。这个信息包含详细的事务信息。事务信息包含在一个签署了JSON字符串(加密)。消息包含签名,这样你就可以验证完整性的签名字串。

注意:你应该使用恢复交易请求类型只有当你的应用程序安装在第一次在一个设备或当你的应用程序中已删除并重新安装一个装置。

其他服务任务

您可能还想要你的服务来接收消息从你BroadcastReceiver意图。您可以使用这些意图消息转达信息从谷歌异步发送应用程序 BroadcastReceiver玩。看一个例子说明你是如何可以发送和接收这些意图的信息,请参见**BillingReceiver.java**和**BillingService**。在示例应用程序的java文件。您可以使用这些示例作为一个基础,在您自己的实现。然而,如果你使用任何的代码样例应用程序,确保您遵循的指导方针的安全设计。

BroadcastReceiver创建一个

谷歌的应用程序使用意图发挥广播发送异步计费反应到您的应用程序。接受这些意图的消息,您需要创建一个BroadcastReceiver可以处理以下的意图:

com android 自动售货计费响应代码

这个广播意图包含一个谷歌玩响应代码,并且送了你做一个应用收费的请求。更多信息发送响应代码对这个响应,看到谷歌玩响应代码为应用收费。

com android 自动售货中的付费应用程序通知

此响应表明购买已经改变状态,这意味着一个购买成功,被取消了,或者被退还。通知消息的更多信息,请参阅应用收费广播意图

com android自动售货计费采购状态更改

这个广播意图包含详细的信息的一个或多个事务。为更多的信息关于采购状态消息,查看应用收费广播意图

每个这些广播意图提供额外的意图,而你们BroadcastReceiver必须处理。意图附加列在下面的表(参见表1)。

表1。描述广播意图额外物品,这些都是响应中发送请求到收费。

The following code sample shows how to handle these broadcast intents and intent extras within a BroadcastReceiver. The BroadcastReceiver in this case is named BillingReceiver, just as it is in the sample application.

```
public class BillingReceiver extends BroadcastReceiver {
    private static final String TAG = "BillingReceiver";

    // Intent actions that we receive in the BillingReceiver from Google Play.
    // These are defined by Google Play and cannot be changed.
    // The sample application defines these in the Consts.java file.
    public static final String ACTION_NOTIFY = "com.android.vending.billing.IN_APP_NOTIFY";
    public static final String ACTION_RESPONSE_CODE = "com.android.vending.billing.RESPONSE_CODE";
    public static final String ACTION_PURCHASE_STATE_CHANGED =
        "com.android.vending.billing.PURCHASE_STATE_CHANGED";

    // The intent extras that are passed in an intent from Google Play.
    // These are defined by Google Play and cannot be changed.
    // The sample application defines these in the Consts.java file.
    public static final String NOTIFICATION_ID = "notification_id";
    public static final String INAPP_SIGNED_DATA = "inapp_signed_data";
    public static final String INAPP_SIGNATURE = "inapp_signature";
    public static final String INAPP_REQUEST_ID = "request_id";
    public static final String INAPP_RESPONSE_CODE = "response_code";

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_PURCHASE_STATE_CHANGED.equals(action)) {
            String signedData = intent.getStringExtra(INAPP_SIGNED_DATA);
            String signature = intent.getStringExtra(INAPP_SIGNATURE);
            // Do something with the signedData and the signature.
        } else if (ACTION_NOTIFY.equals(action)) {
            String notifyId = intent.getStringExtra(NOTIFICATION_ID);
            // Do something with the notifyId.
        } else if (ACTION_RESPONSE_CODE.equals(action)) {
            long requestId = intent.getLongExtra(INAPP_REQUEST_ID, -1);
            int responseCodeIndex = intent.getIntExtra(INAPP_RESPONSE_CODE,
                ResponseCode.RESULT_ERROR.ordinal());
            // Do something with the requestId and the responseCodeIndex.
        } else {
            Log.w(TAG, "unexpected action: " + action);
        }
    }
}
```

```

        }
    }
}
// Perform other processing here, such as forwarding intent messages to your local service.
}

```

In addition to receiving broadcast intents from the Google Play application, your BroadcastReceiver must handle the information it received in the broadcast intents. Usually, your BroadcastReceiver does this by sending the information to a local service (discussed in the next section). The `BillingReceiver.java` file in the sample application shows you how to do this. You can use this sample as a basis for your own BroadcastReceiver. However, if you use any of the code from the sample application, be sure you follow the guidelines that are discussed in Security and Design .

Verifying Signatures and Nonces

Google Play's in-app billing service uses two mechanisms to help verify the integrity of the transaction information you receive from Google Play: nonces and signatures. A nonce (number used once) is a cryptographically secure number that your application generates and sends with every `GET_PURCHASE_INFORMATION` and `RESTORE_TRANSACTIONS` request. The nonce is returned with the `PURCHASE_STATE_CHANGED` broadcast intent, enabling you to verify that any given `PURCHASE_STATE_CHANGED` response corresponds to an actual request that you made. Every `PURCHASE_STATE_CHANGED` broadcast intent also includes a signed JSON string and a signature, which you can use to verify the integrity of the response.

Your application must provide a way to generate, manage, and verify nonces. The following sample code shows some simple methods you can use to do this. 除了接受广播意图从谷歌玩的应用程序,你必须处理的信息BroadcastReceiver中收到广播意图。通常,你可以通过 BroadcastReceiver把信息发送给本地服务(在下一节讨论)。BillingReceiver的.java文件在样例应用程序向你展示了如何做到这一点。您可以使用这个示例作为一个基础,在你自己的BroadcastReceiver。然而,如果你使用任何的代码样例应用程序,确保您遵循的指导方针,是讨论在安全性和设计。

验证签名和目前

谷歌玩的应用收费服务使用两种机制来帮助验证事务信息的完整性,您收到谷歌玩耍:目前和签名。一个nonce(号码使用一次)是一个密码安全号码,您的应用程序生成和发送给每个获得购买信息和恢复交易请求。返回此nonce与采购状态改变了广播的意图,使您能够验证任何给定的采购状态改变响应对应一个实际的请求您。每个采购状态改变了广播的意图还包括一个签署了JSON字符串和一个签名,您可以使用它来验证的完整性的响应。

您的应用程序必须提供一种方法来生成、管理和验证过的nonce。下面的示例代码展示了一些简单的方法你可以使用的。

```

private static final SecureRandom RANDOM = new SecureRandom();
private static HashSet<Long> sKnownNonces = new HashSet<Long>();

public static long generateNonce() {
    long nonce = RANDOM.nextLong();
    sKnownNonces.add(nonce);
}

```

```

        return nonce;
    }

    public static void removeNonce( long nonce ) {
        sKnownNonces.remove( nonce );
    }

    public static boolean isNonceKnown( long nonce ) {
        return sKnownNonces.contains( nonce );
    }
}

```

Your application must also provide a way to verify the signatures that accompany every PURCHASE_STATE_CHANGED broadcast intent. The Security.java file in the sample application shows you how to do this. If you use this file as a basis for your own security implementation, be sure to follow the guidelines in Security and Design and obfuscate your code.

You will need to use your Google Play public key to perform the signature verification. The following procedure shows you how to retrieve Base64-encoded public key from the Google Play publisher site.

Log in to your publisher account. On the upper left part of the page, under your name, click Edit profile. On the Edit Profile page, scroll down to the Licensing & In-app Billing panel (see figure 2). Copy your public key. Important: To keep your public key safe from malicious users and hackers, do not embed your public key as an entire literal string. Instead, construct the string at runtime from pieces or use bit manipulation (for example, XOR with some other string) to hide the actual key. The key itself is not secret information, but you do not want to make it easy for a hacker or malicious user to replace the public key with another key. 您的应用程序还必须提供一种方法来验证签名,伴随每次采购状态改变了广播的意图。安全。java文件在样例应用程序向你展示了如何做到这一点。如果你使用这个文件为基础,实现您自己的安全,一定要按照指南在安全、设计和混淆你的代码。

您将需要使用你的谷歌玩公钥来执行签名验证。下面的过程显示了如何检索base64编码的公共密匙的谷歌玩出版商网站。

登录到你的发布者帐户。

左上角页面的一部分,在你的名字,点击编辑按钮。

在编辑页面,向下滚动到许可&应用收费面板(见图2)。

复制你的公钥。

重要:为了保持你的公钥免受恶意用户和黑客,不要嵌入您的公钥作为一个整体文字字符串。相反,在运行时构建字符串从碎片或使用位操作(例如,XOR与其他字符串)来隐藏实际的关键。密匙本身不是秘密信息,但是你不想容易让一个黑客或者恶意的用户来取代公共密钥与另一个关键。

Licensing & In-app Billing

Your license key allows you to prevent unauthorized distribution of your apps. For more information about licensing, please refer to the [licensing documentation](#).

Your license key can also be used to verify in-app billing purchases. For more information about in-app billing, please refer to the [in-app billing documentation](#).

Below is your Base64-encoded RSA public key (remove spaces).

Public Key	<pre>MIIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIBCgKCAQEAmkzlgp278 HqD+gSyEtFEGEUH7MiedFamIFET5NjDZuh18OVI5PJXjvBMxH50Q</pre>
Test Accounts	<input type="text"/> Comma-separated list of additional Gmail addresses (aside from your address) that will get the License Test Response from applications you manage. These users will also be able to make in-app-billing purchases from uploaded, unpublished applications you manage.
License Test Response	<input type="button" value="Respond normally"/> This License Test Response will be sent to devices using your address or the Test Accounts listed above for applications you have uploaded to Market. Additionally, this account (but not the Test Accounts) will receive this response for applications that have not yet been uploaded to Market.

Figure 2. The Licensing and In-app Billing panel of your account's Edit Profile page lets you see your public key.

Modifying Your Application Code

After you finish adding in-app billing components to your project, you are ready to modify your application's code. For a typical implementation, like the one that is demonstrated in the sample application, this means you need to write code to do the following:

Create a storage mechanism for storing users' purchase information. Create a user interface that lets users select items for purchase. The sample code in Dungeons.java shows you how to do both of these tasks.

Creating a storage mechanism for storing purchase information You must set up a database or some other mechanism for storing users' purchase information. The sample application provides an example database (PurchaseDatabase.java); however, the example database has been simplified for clarity and does not exhibit the security best practices that we recommend. If you have a remote server, we recommend that you store purchase information on your server instead of in a local database on a device. For more information about security best practices, see Security and Design.

Note: If you store any purchase information on a device, be sure to encrypt the data and use a device-specific encryption key. Also, if the purchase type for any

of your items is "unmanaged," we recommend that you back up the purchase information for these items to a remote server or use Android's data backup framework to back up the purchase information. Backing up purchase information for unmanaged items is important because unmanaged items cannot be restored by using the RESTORE_TRANSACTIONS request type.

Creating a user interface for selecting items You must provide users with a means for selecting items that they want to purchase. Google Play provides the checkout user interface (which is where the user provides a form of payment and approves the purchase), but your application must provide a control (widget) that invokes the sendBillingRequest() method when a user selects an item for purchase.

You can render the control and trigger the sendBillingRequest() method any way you want. The sample application uses a spinner widget and a button to present items to a user and trigger a billing request (see Dungeons.java). The user interface also shows a list of recently purchased items.

图2。许可和应用收费面板您的帐户的编辑页面让你看到你的公钥。

修改您的应用程序代码

在你完成添加组件到你的项目中应用收费,你准备好修改应用程序的代码。对于一个典型的实现,就象一个演示了在示例应用程序中,这意味着您需要编写代码来做到以下几点:

创建一个存储机制,用于存储用户的购买信息。

创建一个用户界面,用户可以选择购买的物品。

在地下城的示例代码。java显示你如何做这两个任务。

创建一个存储机制,用于存储购买信息

你必须建立一个数据库或一些其他的机制,用于存储用户的购买信息。样例应用程序提供了一个示例数据库 (`PurchaseDatabase.java`);然而,这个示例数据库已经被简化为清晰起见,不存在安全最佳实践,我们建议。如果你有一个远程服务器上,我们建议你商店的购买信息在你的服务器,而不是在一个设备上的本地数据库。更多信息的安全最佳实践,见安全性和设计。

注意:如果您购买信息存储任何设备上,一定要对数据进行加密,使用一个特定于设备的加密密钥。同样,如果购买你的产品类型为任何"托管",我们建议您已经备份了这些商品的购买信息到一个远程服务器或者使用Android的数据备份框架来支持购买信息。备份购买信息非托管项目很重要,因为非托管项目不能恢复 使用恢复交易请求类型。

创建一个用户界面来选择项目

你必须提供给用户一个手段,他们想要选择物品购买。谷歌的**checkout**播放提供了用户界面(这是用户提供了一种形式的支付和确认购买的),但您的应用程序必须提供一个控制(小部件),调用**sendBillingRequest()**方法当用户选择一个项目,为购买。

你可以呈现控制和触发**sendBillingRequest()**方法任何你想要的方式。该示例应用程序使用一个转子部件和一个按钮呈现项目对用户和触发账单请求(参见地牢java)。用户界面还列出了最近购买的物品。

来自 "[index.php?title=Implementing_In-app_Billing&oldid=8539](#)"



Subscriptions

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： eversleeping

原文链
接：

http://developer.android.com/guide/google/play/billing/billing_subscriptions.html

目录

[[隐藏](#)]

[1 Subscriptions \(订阅\)](#)

- [1.1 Overview of Subscriptions](#)
 - [1.1.1 Subscription publishing and unpublishing](#)
 - [1.1.2 Subscription pricing](#)
 - [1.1.3 User billing](#)
 - [1.1.4 Subscription cancellation](#)
 - [1.1.5 App uninstallation](#)
 - [1.1.6 Refunds](#)
 - [1.1.7 Payment processing and policies](#)
 - [1.1.8 System requirements for subscriptions](#)
 - [1.1.9 Compatibility considerations](#)
- [1.2 Implementing Subscriptions](#)
 - [1.2.1 Sample application](#)
 - [1.2.2 Application model](#)
 - [1.2.3 Purchase token](#)
 - [1.2.4 Requesting a subscription purchase](#)
 - [1.2.5 Restoring transactions](#)
- [1.3 Checking subscription validity](#)
 - [1.3.1 Launching your product page to let the user cancel or view subscriptions](#)

- [1.3.2 Recurring billing, cancellation, and changes in purchase state](#)
- [1.3.3 Modifying your app for subscriptions](#)
- [1.4 Administering Subscriptions](#)
- [1.5 Google Play Android Developer API](#)
 - [1.5.1 Using the API](#)
 - [1.5.2 Quota](#)
 - [1.5.3 Authorization](#)
 - [1.5.4 Using the API efficiently](#)

Subscriptions (订阅)

Subscriptions let you sell content, services, or features in your app with automated, recurring billing. Adding support for subscriptions is straightforward and you can easily adapt an existing In-app Billing implementation to sell subscriptions.

订阅让你在自己的应用程序中卖内容、服务或功能提供自动化、重复计费。添加支持订阅非常简单,您可以轻松地调整现有应用收费实现销售订阅服务。

If you have already implemented In-app Billing for one-time purchase products, you will find that you can add support for subscriptions with minimal impact on your code. If you are new to In-app Billing, you can implement subscriptions using the standard communication model, data structures, and user interactions as for other in-app products.subscriptions. Because the implementation of subscriptions follows the same path as for other in-app products, details are provided outside of this document, starting with the In-app Billing Overview.

如果你已经实现了应用收费一次性购买产品时,你就会发现你可以添加支持订阅与最小的影响你的代码。如果你是新的应用收费,您可以实现订阅使用标准的通信模型、数据结构,以及用户交互应用内订阅等其他产品。因为实施订阅遵循相同的路径等其他应用内的产品,提供了详细的文档外,从应用收费的概述。

This document is focused on highlighting implementation details that are specific to subscriptions, along with some strategies for the associated billing and business models.

本文档重点关注强调实现细节是特定于订阅,连同一些战略相关的计费和商业模式。

Overview of Subscriptions

A subscription is a new product type offered in In-app Billing that lets you sell content, services, or features to users from inside your app with recurring monthly or annual billing. You can sell subscriptions to almost any type of digital content, from any type of app or game.

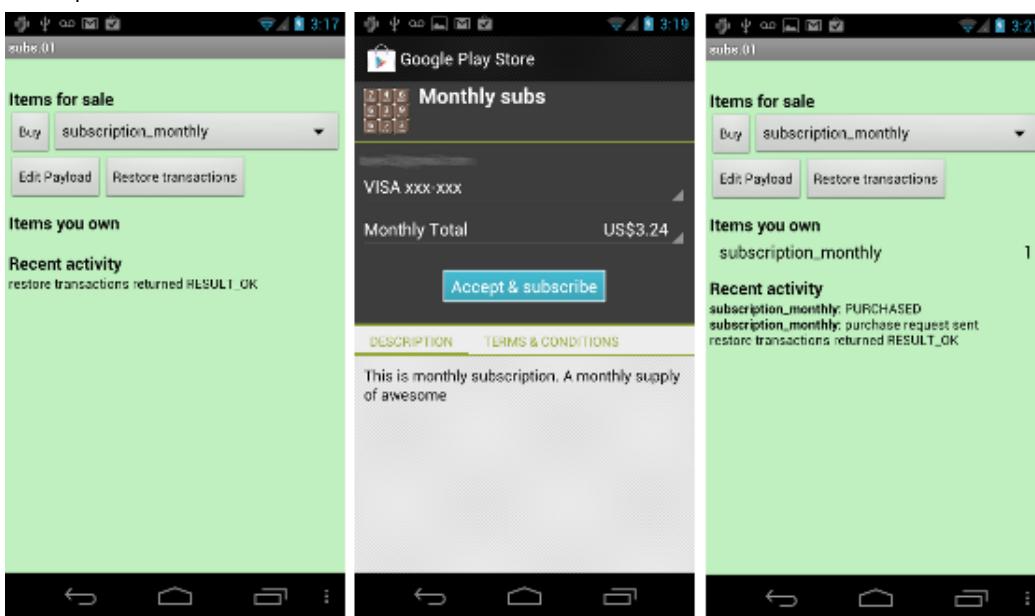
订阅是一个新的产品类型提供应用收费,让你卖内容、服务或功能来从您的应用程序的用户在用逐月或年度计费。你可以订阅卖给几乎任何类型的数字内容,从任何类型的应用程序或游戏。

As with other in-app products, you configure and publish subscriptions using the Developer Console and then sell them from inside apps installed on an Android-powered devices. In the Developer console, you create subscription products and add them to a product list, setting a price for each, choosing a billing interval of monthly or annually, and then publishing. In your apps, it's straightforward to add support for subscription purchases. The implementation extends the standard In-app Billing API to support a new product type but uses the same communication model, data structures, and user interactions as for other in-app products.

与其他应用程序的产品,您可以配置和发布订阅使用开发人员控制台,然后卖给他们从内应用程序安装在一个安卓设备。在开发人员控制台,您创建 订阅的产品并将它们添加到一个产品列表,设置一个价格对每个人来说,选择每月或每年的计费时间间隔,然后出版。在你的应用程序,可以直接添加支持订阅购 买。实现扩展了应用收费标准API支持一个新的产品类型,使用相同的通信模型、数据结构,以及用户交互等其他应用内的产品。

When users purchase subscriptions in your apps, Google Play handles all checkout details so your apps never have to directly process any financial transactions. Google Play processes all payments for subscriptions through Google Wallet, just as it does for standard in-app products and app purchases. This ensures a consistent and familiar purchase flow for your users.

当用户购买订阅在你的应用软件,谷歌玩处理所有检出细节让你的应用程序永远不需要直接处理任何金融交易。谷歌玩处理所有通过谷歌钱包支付订阅,就像它对标准应用内购买的产品和应用程序。这确保一致的和熟悉的采购流程为你的用户。



After users have purchased subscriptions, they can view the subscriptions and cancel them, if necessary, from the My Apps screen in the Play Store app or from the app's product details page in the Play Store app.

付费用户之后,他们可以查看订阅和取消他们,如果有必要,从我的应用程序屏幕播放存储应用程序或应用的产品细节页面在剧中商店应用程序。

Once users have purchased a subscription through In-app Billing, you can easily give them extended access to additional content on your web site (or other service) through the use of a server-side API provided for In-app Billing. The server-side API lets you validate the status of a subscription when users sign into your other services. For more information about the API, see Google Play Android Developer API, below.

一旦用户购买了一套通过应用收费订阅,你可以很容易地给他们扩展访问额外的内容在你的网站上(或其他服务)通过使用一个服务器端API提供了应用收费。服务器端API允许您验证状态的订阅当用户登录到您的其他服务。该API的更多信息,请参阅谷歌Android开发者API。

You can also build on your existing external subscriber base from inside your Android apps. If you sell subscriptions on a web site, for example, you can add your own business logic to your Android app to determine whether the user has already purchased a subscription elsewhere, then allow access to your content if so or offer a subscription purchase from Google Play if not.

您还可以构建在你现有的外部用户基数从内你的Android应用程序。如果你在网站上销售订阅服务,例如,您可以添加您自己的业务逻辑来你的Android应用程序来确定该用户是否已经购买了一个订阅其他地方,然后允许访问你的内容如果所以或提供订阅谷歌购买玩如果不是。

With the flexibility of In-app Billing, you can even implement your own solution for sharing subscriptions across as many different apps or products as you want. For example, you could sell a subscription that gives a subscriber access to an entire collection of apps, games, or other content for a monthly or annual fee. To implement this solution, you could add your own business logic to your app to determine whether the user has already purchased a given subscription and if so, allow access to your content.

灵活性应用收费,你甚至可以实现您自己的解决方案为共享订阅到尽可能多的不同的应用程序或产品像你希望的那样。例如,你可以卖一个订阅,给出了一个用户访问整个系列的应用、游戏或其他内容,按月或年费。实现这种解决方案,您可以添加您自己的业务逻辑来你的应用程序来确定该用户是否已经购买了一个给定订阅,如果这样的话,允许访问你的内容。

In general the same basic policies and terms apply to subscriptions as to standard in-app products, however there are some differences. For complete information about the current policies and terms, please read the policies document.

在一般的相同的基本政策和条款适用于订阅以标准应用的产品,但是有一些差异。关于当前的完整信息政策和条款,请阅读策略文档。

Subscription publishing and unpublishing

To sell a subscription in an app, you use the tools in the Developer Console to set up a product list for the app and then create and configure a new subscription. In the subscription, you set the price and billing interval and define a subscription ID, title, and description. When you are ready, you can then publish the subscription in the app product list.

出售订阅应用程序里,你使用工具的开发人员控制台来建立一个产品列表的应用程序,然后创建并配置一个新的订阅。在订阅,你可以设定价格和计费间隔和定义一个订阅ID、标题和描述。当你准备好了,然后您可以发布中的订阅应用程序产品列表。

In the product list, you can add subscriptions, in-app products, or both. You

can add multiple subscriptions that give access to different content or services, or you can add multiple subscriptions that give access to the same content but for different intervals or different prices, such as for a promotion. For example, a news outlet might decide to offer both monthly and annual subscriptions to the same content, with annual having a discount. You can also offer in-app purchase equivalents for subscription products, to ensure that your content is available to users of older devices that do not support subscriptions.

在产品列表中,您可以添加订阅,应用内的产品,或两者兼而有之。您可以添加多个订阅,获得不同的内容或服务,或者您可以添加多个订阅,给访问相同的内容,但对于不同的时间间隔或不同的价格,比如晋升。例如,一个新闻的出口可能决定同时提供月度和年度订阅相同的内容,年度有折扣。你也可以提供应用内购买等价的订阅的产品,以确保你的内容可供用户的老设备不支持订阅。

After you add a subscription or in-app product to the product list, you must publish the product before Google Play can make it available for purchase. Note that you must also publish the app itself before Google Play will make the products available for purchase inside the app.

在你添加订阅或应用内产品的列表,您必须发布产品之前谷歌玩耍,能让它可以购买。请注意,您还必须发布应用程序本身会使谷歌之前玩可供购买的产品进入应用程序。

Important: At this time, the capability to unpublish a subscription is not available. Support for unpublishing a subscription is coming to the Developer Console in the weeks ahead, so this is a temporary limitation. In the short term, instead of unpublishing, you can remove the subscription product from the product list offered in your app to prevent users from seeing or purchasing it.

重要:在这个时间,能够取消发布订阅是不可用的。支持unpublishing订阅来开发人员控制台在未来几周内,所以这是一个暂时的限制。在短期内,而不是unpublishing,您可以删除订阅的产品从产品列表提供了在自己的应用程序,以防止用户看到或购买它。

Subscription pricing

When you create a subscription in the Developer Console, you can set a price

for it in any available currencies. Each subscription must have a non-zero price. You can price multiple subscriptions for the same content differently — for example you could offer a discount on an annual subscription relative to the monthly equivalent.

当你创建一个订阅在开发人员控制台,您可以设置一个价格在任何可用的货币。每个订阅必须有一个非零的价格。您可以为多个订阅价格相同内容的不同——例如你可以提供一个优惠订阅一年相对于每月的等效。

Important: At this time, once you publish a subscription product, you cannot change its price in any currency. Support for changing the price of published subscriptions is coming to the Developer Console in the weeks ahead. In the short term, you can work around this limitation by publishing a new subscription product ID at a new price, then offer it in your app instead of the original product. Users who have already purchased will continue to be charged at the original price, but new users will be charged at the new price.

重要:在这个时间,一旦您发布一个订阅产品,你不能改变它的价格在任何货币。支持更改发布订阅的价格来开发人员控制台在未来几周内。在短期内,您可以解决这个限制通过发布一个新的订阅产品ID在一个新的价格,然后提供它在自己的应用程序而不是原始的产品。用户已经购买将继续被指控在原来的价格,但是新用户将被收取的新价格。

User billing

You can sell subscription products with automated recurring billing at either of two intervals:

你可以卖订阅产品与自动重复计费,要么两个间隔:

- Monthly — Google Play bills the customer's Google Wallet account at the time of purchase and monthly subsequent to the purchase date (exact billing intervals can vary slightly over time)
- 每月——谷歌玩账单客户的谷歌钱包账户在购买和每月的后续购买日期(确切的计费时间间隔随着时间会略有不同)
- Annually — Google Play bills the customer's Google Wallet account at the time of purchase and again on the same date in subsequent years.

- 每年一次——谷歌玩账单客户的谷歌钱包账户的时候再购买并在相同的日期在接下来的几年。

Billing continues indefinitely at the interval and price specified for the subscription. At each subscription renewal, Google Play charges the user account automatically, then notifies the user of the charges afterward by email. Billing cycles will always match subscription cycles, based on the purchase date.

计费无限延伸的间隔和指定订阅的价格。在每个续订谷歌玩指控用户帐号自动,然后通知用户通过电子邮件的指控之后。账单周期将总是会匹配订阅周期,根据购买日期。

Over the life of a subscription, the form of payment billed remains the same — Google Play always bills the same form of payment (such as credit card, Direct Carrier Billing) that was originally used to purchase the subscription.

一生的订阅,付款的形式宣传是一样的——谷歌对手总是账单相同的付款方式(比如信用卡,直接运营商账单),原本用来购买订阅。

When the subscription payment is approved by Google Wallet, Google Play provides a purchase token back to the purchasing app through the In-app Billing API. For details, see Purchase token, below. Your apps can store the token locally or pass it to your backend servers, which can then use it to validate or cancel the subscription remotely using the Google Play Android Developer API.

当订阅付款批准谷歌钱包,谷歌播放提供了一个购买记号传回采购应用程序到应用收费的API。有关详细信息,请参见购买令牌,下面。你的应用可以在本地存储令牌或将它传递给你的后端服务器,然后使用它来验证或取消订阅远程使用谷歌Android开发者API玩。

If a recurring payment fails, such as could happen if the customer's credit card has become invalid, the subscription does not renew. Google Play notifies your app at the end of the active cycle that the purchase state of the subscription is now "Expired". Your app does not need to grant the user further access to the subscription content.

如果一个经常性付款失败,如可能会在客户的信用卡已经成为无效,订阅没有续签。谷歌打通知你的应用程序结束时的活动周期,采购状态的订阅现在“过期”。你的应用程序不需要授予用户进一步访问订阅内容。

As a best practice, we recommend that your app includes business logic to notify your backend servers of subscription purchases, tokens, and any billing errors that may occur. Your backend servers can use the server-side API to query and update your records and follow up with customers directly, if needed.

作为一项最佳实践,我们建议您的应用程序包含业务逻辑来通知你的后端服务器的订阅购买,令牌,以及任何可能出现的错误收费。你的后端服务器可以使用服务器端API查询和更新你的记录和跟踪直接与客户,如果需要的话。

Subscription cancellation

Users can view the status of all of their subscriptions and cancel them if necessary from the My Apps screen in the Play Store app. Currently, the In-app Billing API does not provide support for canceling subscriptions direct from inside the purchasing app, although your app can broadcast an Intent to launch the Play Store app directly to the My Apps screen.

用户可以查看所有的地位的订阅和取消他们如果有必要从我的应用程序屏幕上玩商店应用程序。目前,应用收费API并不提供支持取消订阅直接从国内采购程序,虽然你的应用程序可以通过意图推出玩商店应用程序直接向我的应用程序屏幕。

When the user cancels a subscription, Google Play does not offer a refund for the current billing cycle. Instead, it allows the user to have access to the cancelled subscription until the end of the current billing cycle, at which time it terminates the subscription. For example, if a user purchases a monthly subscription and cancels it on the 15th day of the cycle, Google Play will consider the subscription valid until the end of the 30th day (or other day, depending on the month).

当用户取消订阅,谷歌玩不提供退款,为当前的结算周期。相反,它允许用户访问取消订阅,直到年底当前的结算周期,它将终止订阅。例如,如果一个用户购买每月订阅和取消它的第15天的周期,谷歌玩将考虑订阅有效直到最后30天(或其他天,这取决于月)。

In some cases, the user may contact you directly to request cancellation of a subscription. In this and similar cases, you can use the server-side API to query and directly cancel the user's subscription from your servers.

在某些情况下,用户可能会联系你直接请求取消订阅。在这个和类似的情况下,您

可以使用服务器端API查询和直接取消用户的订阅从您的服务器。

Important: In all cases, you must continue to offer the content that your subscribers have purchased through their subscriptions, for as long any users are able to access it. That is, you must not remove any subscriber's content while any user still has an active subscription to it, even if that subscription will terminate at the end of the current billing cycle. Removing content that a subscriber is entitled to access will result in penalties. Please see the policies document for more information.

重要:在所有的情况下,你必须继续提供内容,您的用户购买了通过他们的订阅,只要任何用户都可以访问它。也就是说,您必须没有删除任何用户的内容,同时任何用户仍然有一个活跃的订阅它,即使该订阅将终止在结束当前的结算周期。删除内容的用户有权访问会导致罚款。请参阅政策文档获得更多信息。

App uninstallation

When the user uninstalls an app that includes purchased subscriptions, the Play Store app will notify the user that there are active subscriptions. If the user chooses to continue with the uninstallation, the app is removed and the subscriptions remain active and recurring billing continues. The user can return to cancel the associated subscriptions at any time in the My Apps screen of the Play Store app. If the user chooses to cancel the uninstallation, the app and subscriptions remain as they were.

当用户卸载一个应用程序,包括购买订阅,这出戏商店应用程序会通知用户,有活跃的订阅。如果用户选择继续uninstalltion,应用 程序被移除和订阅保持活跃和重复计费仍在继续。用户可以回到取消相关的订阅任何时候我的应用程序屏幕上玩商店应用程序。如果用户选择取消卸载,应用程序和 订阅保持他们。

Refunds

As with other in-app products, Google Play does not provide a refund window for subscription purchases. For example, users who purchase an app can ask for a refund from Google Play within a 15-minute window. With subscriptions, Google Play does not provide a refund window, so users will need to contact you directly to request a refund.

与其他应用程序的产品,谷歌玩不提供退款窗口购买订阅。例如,用户购买应用程序可以要求退款从谷歌戏中15分钟的窗口。订阅,谷歌玩不提供退款窗口,所以用户需要你直接联系要求退款。

If you receive requests for refunds, you can use the server-side API to cancel the subscription or verify that it is already cancelled. However, keep in mind that Google Play considers cancelled subscriptions valid until the end of their current billing cycles, so even if you grant a refund and cancel the subscription, the user will still have access to the content.

如果你收到请求退款,您可以使用服务器端API来取消订阅或者检查是否已经取消了。然而,请记住,谷歌玩认为取消订阅有效直到他们当前的账单周期,所以即使你格兰特退款和取消订阅,用户仍可以访问的内容。

Note: Partial refunds for canceled subscriptions are not available at this time.

注:部分退款为取消订阅尚未提供。

Payment processing and policies

In general, the terms of Google Play allow you to sell in-app subscriptions only through the standard payment processor, Google Wallet. For purchases of any subscription products, just as for other in-app products and apps, the transaction fee for subscriptions, just as for other in-app purchases, is the same as the transaction fee for application purchases (30%).

一般来说,谷歌的条款允许你去卖玩应用内订阅只有通过标准的付款处理器,谷歌钱包。对购买任何订阅产品,就像其他应用内的产品和应用程序,交易费给订阅,就像其他应用内购买、相同的交易费给应用程序的购买(30%)。

Apps published on Google Play that are selling subscriptions must use In-app Billing to handle the transaction and may not provide links to a purchase flow outside of the app and Google Play (such as to a web site).

应用程序在谷歌发布玩,出售订阅必须使用应用收费来处理事务,不得提供链接到一个购买流之外的应用程序和谷歌玩(比如一个web站点)。

For complete details about terms and policies, see the policies document.

对于完整的细节条款和政策,看到政策文件。

System requirements for subscriptions

In-app purchases of subscriptions are supported only on devices that meet these minimum requirements:

- Must run Android 2.2 or higher
- Google Play Store app, version 3.5 or higher, must be installed

Google Play 3.5 and later versions include support for the In-app Billing v2 API or higher, which is needed to support handling of subscription products.

应用内购买订阅是只支持的设备,满足这些最低要求:

- 必须运行Android 2.2或更高吗
- 谷歌玩商店应用程序,版本3.5或更高版本,必须安装

谷歌玩3.5和更高版本包括支持应用收费v2 API或更高,它需要支持处理订阅的产品。

Compatibility considerations

As noted in the previous section, support for subscriptions is available only on devices that meet the system requirements. Not all devices will receive or install Google Play 3.5, so not all users who install your apps will have access to the In-app Billing API and subscriptions.

正如在前面的小节中,支持订阅只有在设备,满足系统要求。并非所有设备都将收到或安装谷歌玩3.5,所以并不是所有用户安装您的应用程序可以访问API和订阅应用收费。

If you are targeting older devices that run Android 2.1 or earlier, we recommend that you offer those users an alternative way buy the content that is available through subscriptions. For example, you could create standard in-app products (one-time purchases) that give access to similar content as your subscriptions, possibly for a longer interval such as a year.

如果你的目标是老设备,运行Android 2.1或更早的时候,我们建议您的用户提供一种替代方式购买内容,可以通过订阅。例如,您可以创建标准应用内产品(一次性购买),获得类似的内容作为你的订阅,很可能是一个较长的时间间隔一年等。

Implementing Subscriptions

Subscriptions are a standard In-app Billing product type. If you have already implemented In-app Billing for one-time purchase products, you will find that adding support for subscriptions is straightforward, with minimal impact on your code. If you are new to In-app Billing, you can implement subscriptions using the standard communication model, data structures, and user interactions as for other in-app products.subscriptions.

The full implementation details for In-app Billing are provided outside of this document, starting with the In-app Billing Overview. This document is focused on highlighting implementation details that are specific to subscriptions, along with some strategies for the associated billing and business models.

订阅是一个标准的应用收费产品类型。如果你已经实现了应用收费一次性购买的产品,你会发现,添加支持订阅是非常简单的,对代码的影响微乎其微。如果你是新的应用收费,您可以实现订阅使用标准的通信模型、数据结构,以及用户交互应用内订阅等其他产品。

完整的实现细节提供了应用收费之外的文档开始,应用收费的概述。本文档重点关注强调实现细节是特定于订阅,连同一些战略相关的计费和商业模式。

Sample application

To help you get started with your In-app Billing implementation and subscriptions, an updated version of the In-app Billing sample app is available. You can download the sample app from the Android SDK repository using the Android SDK Manager. For details, see [Downloading the Sample Application](#).

帮助您开始使用你的应用收费实现和订阅,一个更新版本的应用收费样例应用程序使用的是可用的。您可以下载样例应用程序在Android SDK库使用Android SDK经理。有关详细信息,请参见下载示例应用程序。

Application model

With subscriptions, your app uses the standard In-app Billing application model, sending billing requests to the Play Store application over interprocess communication (IPC) and receiving purchase responses from the Play Store app in the form of asynchronous broadcast intents. Your application does not manage any network connections between itself and the Google Play server or

use any special APIs from the Android platform.

Your app also uses the standard In-app Billing components — a billing Service for sending requests, a BroadcastReceiver for receiving the responses, and a security component for verifying that the response was sent by Google Play. Also recommended are a response Handler for processing notifications, errors, and status messages, and an observer for sending callbacks to your application as needed. All of these components and their interactions are described in full in the In-app Billing Overview and related documents.

To initiate different types of billing communication with Google Play, your app will use the standard set of in-app billing requests and receive the same responses. Inside the requests and responses are two new fields described below.

订阅,你的应用程序使用了标准的应用收费应用程序模型,请求发送帐单玩存储应用程序在进程间通信(IPC)和接响应玩商店购买的程序的形式的异步广播意图。你的应用程序没有管理任何网络连接本身和谷歌之间扮演服务器或使用任何特殊的Android平台的api。

你的应用程序还使用了标准的应用收费组件——一个账单服务发送请求,一个用于接响应BroadcastReceiver,和一个安全组件验证响应被谷歌 玩。还建议是一个响应处理程序来处理通知、错误和状态信息,和一个观察者发送回调到您的应用程序所需要的。所有这些组件及其之间的交互中充分描述了应用收 费概述和相关文件。

启动不同类型的计费沟通与谷歌进行游戏的时候,你的应用程序将使用标准的一些应用收费请求和接收相同的反应。在请求和响应是下面描述的两个新字段。

Purchase token

Central to the end-to-end architecture for subscriptions is the purchase token, a string value that uniquely identifies (and associates) a user ID and a subscription ID. Google Play generates the purchase token when the user completes the purchase of a subscription product (and payment is approved by Google Wallet) and then sends it to the purchasing app on the device through the In-app Billing API.

At the conclusion of a PURCHASE_REQUEST message flow, your app can retrieve the purchase token and other transaction details by initiating a GET_PURCHASE_INFORMATION request. The Bundle returned by the call contains an JSON array of order objects. In the order corresponding to the subscription purchase, the token is available in the purchaseToken field.

An example of a JSON order object that includes a subscription purchase token is shown below.

```
{
  "nonce" : 1836535032137741465,
  "orders" :
    [ {
      "notificationId" : "android.test.purchased",
      "orderId" : "transactionId.android.test.purchased",
      "packageName" : "com.example.dungeons",
      "productId" : "android.test.purchased",
      "developerPayload" : "bGoa+V7g/yqDXvKRqq+JTFn4uQZbPiQJo4pf9RzJ",
      "purchaseTime" : 1290114783411,
      "purchaseState" : 0,
      "purchaseToken" : "rojeslcyyiapnqcynkjyyjh" } ]
}
```

中央对端到端架构是购买订阅令牌,一个字符串值,惟一地标识(和同事)用户ID和一个订阅ID。谷歌玩生成采购令牌当用户完成购买一个订阅产品(和付款是通过谷歌钱包),然后将它发送给采购应用程序在设备上通过应用收费API。

完成采购请求消息流时,应用程序可以检索令牌和其他事务细节购买来启动一个获得购买信息的请求。调用返回的包包含一个JSON数组对象的顺序。在订单对应于订阅购置,令牌是可得到的在purchaseToken字段。

一个例子是一个JSON对象的顺序,包括一个订阅购买令牌如下所示。

After receiving a purchase token, your apps can store the token locally or pass it to your backend servers, which can then use it to query the billing status or cancel the subscription remotely. If your app will store the token locally, please read the Security and Design document for best practices for maintaining the security of your data.

在收到购买令牌,您的应用程序能够在本地存储令牌或将它传递给你的后端服务器,然后使用它来查询账单状态或取消订阅远程。如果你的应用程序会在本地存储

令牌,请阅读安全设计文件为最佳实践,对维护安全的数据。

Checking the In-app Billing API version

Subscriptions support is available only in versions of Google Play that support the In-app Billing v2 API (Google Play 3.5 and higher). For your app, an essential first step at launch is to check whether the version of Google Play installed on the device supports the In-app Billing v2 API and subscriptions.

To do this, create a CHECK_BILLING_SUPPORTED request Bundle that includes the required key-value pairs, together with

The API_VERSION key, assigning a value of 2. The BILLING_REQUEST_ITEM_TYPE key, assigning a value of "subs" Send the request using sendBillingRequest(Bundle) and receive the response Bundle. You can extract the response from the BILLING_RESPONSE_RESPONSE_CODE key of the response. RESULT_OK indicates that subscriptions are supported.

The sample app declares constants for the accepted BILLING_REQUEST_ITEM_TYPE values (from Consts.java):

订阅支持只在版本的谷歌玩,支持应用收费v2 API(谷歌玩3.5和更高版本)。为你的应用程序,一个重要的第一步是检查是否在推出的版本上安装谷歌发挥设备支持应用收费v2 API和订阅。

为此,创建一个检查账单支持请求包,其中只包含必需的键-值对,加上

api版本的关键,分配一个值为2。

计费请求项目类型的键,分配一个值“潜艇”

发送请求使用sendBillingRequest(包)和接响应包。你可以提取响应从计费响应响应码键的响应。结果表明,支持可以订阅。

样例应用程序声明常量认可帐单请求项目类型的值(来自Consts.java):

```
// These are the types supported in the IAB v2
public static final String ITEM_TYPE_INAPP = "inapp";
public static final String ITEM_TYPE_SUBSCRIPTION = "subs";
```

It sets up a convenience method for building the request bundle (from `BillingService.java`):

```
protected Bundle makeRequestBundle(String method) {
    Bundle request = new Bundle();
    request.putString(Consts.BILLING_REQUEST_METHOD, method);
    request.putInt(Consts.BILLING_REQUEST_API_VERSION, 2);
    request.putString(Consts.BILLING_REQUEST_PACKAGE_NAME,
getPackageName());
    return request;
}
```

Here's an example of how to test support for In-App Billing v2 and subscriptions (from `BillingService.java`):

```
/**
 * Wrapper class that checks if in-app billing is supported.
 */
class CheckBillingSupported extends BillingRequest {
    public String mProductType = null;
    public CheckBillingSupported() {
        // This object is never created as a side effect of starting
this
        // service so we pass -1 as the startId to indicate that we
should
        // not stop this service after executing this request.
        super(-1);
    }

    public CheckBillingSupported(String type) {
        super(-1);
        mProductType = type;
    }

    @Override
    protected long run() throws RemoteException {
        Bundle request = makeRequestBundle("CHECK_BILLING_SUPPORTED");
        if (mProductType != null) {
            request.putString(Consts.BILLING_REQUEST_ITEM_TYPE,
mProductType);
        }
        Bundle response = mService.sendBillingRequest(request);
        int responseCode =
response.getInt(Consts.BILLING_RESPONSE_RESPONSE_CODE);
        if (Consts.DEBUG) {
            Log.i(TAG, "CheckBillingSupported response code: " +
                ResponseCode.valueOf(responseCode));
        }
        boolean billingSupported = (responseCode ==
ResponseCode.RESULT_OK.ordinal());
        ResponseHandler.checkBillingSupportedResponse(billingSupported,
mProductType);
        return Consts.BILLING_RESPONSE_INVALID_REQUEST_ID;
    }
}
```

}

Requesting a subscription purchase

Once you've checked the API version as described above and determined that subscriptions are supported, you can present subscription products to the user for purchase. When the user has selected a subscription product and initiated a purchase, your app handles the purchase just as it would for other in-app products — by sending a REQUEST_PURCHASE request. You can then launch Google Play to display the checkout user interface and handle the financial transaction..

一旦你已经检查了API版本如上所述,决定,订阅都受支持,您可以现在订阅产品给用户进行购买。当用户选择了一个订阅产品,并开始了一场购买,你的程序处理购买就像其他应用内产品——通过发送请求购买请求。然后您可以启动谷歌发挥显示结帐的用户界面和处理金融事务。

The REQUEST_PURCHASE includes a Bundle containing the item details, as described in the In-app Billing Overview. For a subscription, the Bundle must also specify:

The ITEM_ID key, with a value that specifies a valid, published subscription product. The ITEM_TYPE key, with a value of "subs" (ITEM_TYPE_SUBSCRIPTION in the sample app). If the request does not specify the subscription's ITEM_TYPE, Google Play attempts to handle the request as a standard in-app purchase (one-time purchase). Google Play synchronously returns a response bundle that includes RESPONSE_CODE, PURCHASE_INTENT, and REQUEST_ID. Your app uses the PURCHASE_INTENT to launch the checkout UI and the message flow proceeds exactly as described in Messaging sequence. 请求购买包括一个包包含项目的细节中所描述的一样,应用收费的概述。为订阅,包还必须指定:

这个ITEM_ID钥匙,一个值,用于指定一个有效的、出版订阅产品。

项目类型的键,值为“潜艇”(项目类型的订阅示例应用程序中的)。如果请求没有指定订阅的项类型,谷歌演出努力处理请求作为一个标准应用内购买(一次性购买)。

谷歌玩同步返回一个响应包,包括响应代码,购买意向,并请求id。你的应用程序使用了购买意向发射检出UI和消息流收益中描述的确切消息序列。

Here's how the sample app initiates a purchase for a subscription, where

mProductType is ITEM_TYPE_SUBSCRIPTION (from BillingService.java).

```
/**
 * Wrapper class that requests a purchase.
 */
class RequestPurchase extends BillingRequest {
    public final String mProductId;
    public final String mDeveloperPayload;
    public final String mProductType;

    ...
    @Override
    protected long run() throws RemoteException {
        Bundle request = makeRequestBundle("REQUEST_PURCHASE");
        request.putString(Consts.BILLING_REQUEST_ITEM_ID, mProductId);
        request.putString(Consts.BILLING_REQUEST_ITEM_TYPE,
mProductType);
        // Note that the developer payload is optional.
        if (mDeveloperPayload != null) {
            request.putString(Consts.BILLING_REQUEST_DEVELOPER_PAYLOAD,
mDeveloperPayload);
        }
        Bundle response = mService.sendBillingRequest(request);
        PendingIntent pendingIntent
            =
response.getParcelable(Consts.BILLING_RESPONSE_PURCHASE_INTENT);
        if (pendingIntent == null) {
            Log.e(TAG, "Error with requestPurchase");
            return Consts.BILLING_RESPONSE_INVALID_REQUEST_ID;
        }

        Intent intent = new Intent();
        ResponseHandler.buyPageIntentResponse(pendingIntent, intent);
        return response.getLong(Consts.BILLING_RESPONSE_REQUEST_ID,
Consts.BILLING_RESPONSE_INVALID_REQUEST_ID);
    }

    @Override
    protected void responseCodeReceived(ResponseCode responseCode) {
        ResponseHandler.responseCodeReceived(BillingService.this,
this, responseCode);
    }
}
```

Restoring transactions

Subscriptions always use the managed by user account purchase type, so that you can restore a record of subscription transactions on the device when needed. When a user installs your app onto a new device, or when the user uninstalls/reinstalls the app on the original device, your app should restore the subscriptions that the user has purchased.

订阅总是使用管理用户帐户购买类型,这样你就可以恢复订阅交易的记录设备上的需要时。当用户安装您的应用程序到一个新的设备,或者当用户卸载/ reinstalls

app原始设备,你的程序应该恢复订阅用户购买了。

The process for restoring subscriptions transactions is the same as described in Messaging sequence. Your app sends a RESTORE_TRANSACTIONS request to Google Play. Google Play sends two broadcast intents as asynchronous responses — a RESPONSE_CODE intent and a PURCHASE_STATE_CHANGED intent.

这个过程对恢复订阅事务描述的一样,在消息传递序列。您的应用程序发送一个恢复交易请求谷歌玩。谷歌玩发送两个广播意图作为异步响应——一个响应代码的意图和采购状态改变的意图。

The PURCHASE_STATE_CHANGED intent contains a notification ID that your app can use to retrieve the purchase details, including the purchase token, by sending a standard GET_PURCHASE_INFORMATION request. The Bundle returned in the call includes an JSON array of order objects corresponding to subscription (and in-app product) purchases that you can restore locally.

购买状态改变的意图包含一个通知你的应用程序可以使用ID来检索购买详细信息,包括购买的令牌,通过发送一个标准得购买信息的请求。Bundle中返回调用包含一个JSON数组的订单对象对应于订阅(和应用内的产品),您可以恢复购买本地。

Your app can store the restored purchase state and other transaction details in the way that best meets your needs. Your app can use it later to check the subscription validity, although please read the Security and Design document for best practices for maintaining the security of your data.

你的应用程序可以存储恢复采购状态和其他交易的细节,最满足你的需要。你的应用程序可以稍后使用它来检查订阅的有效性,但请先阅读安全设计文件为最佳实践,对维护安全的数据。

Checking subscription validity=

Subscriptions are time-bound purchases that require successful billing recurrences over time to remain valid. Your app should check the validity of purchased subscriptions at launch or prior to granting access to subscriber content.

订阅是有时间限制的采购要求成功的账单复发随着时间继续有效。你的应用程序应该检查的有效性,在发射时,购买订阅前或授予访问用户内容。

With In-app Billing, you validate a subscription by keeping track of its purchase state and then checking the state whenever needed. Google Play provides two ways to let you know when the purchase state of a subscription changes:

- In-app Billing Notifications. Google Play pushes a notification to your app to indicate a change in the purchase state of a subscription. Your app can store the most recent purchase state for a given purchase token and then check that state at run time, as needed.
- Google Play Android Developer API. You can use this HTTP-based API to poll Google Play for the current purchase state of a subscription. You can store the purchased state for each purchaseToken on your backend servers. For more information, see Google Play Android Developer API, below.

For most use-cases, especially those where backend servers are already keeping track of subscribed users, implementing a combination of both methods is the recommended approach. A typical implementation might work like this:

与应用收费,您验证订阅通过追踪其购买的状态,然后在需要时检查状态。谷歌发挥提供了两种方法让你知道什么时候购买订阅状态变化:

- 应用收费通知。谷歌玩推着一个通知到您的应用程序来显示一个改变了订阅服务的采购状态。你的应用程序可以存储最近的采购状态对于一个给定的购买令牌,然后在运行时检查状态,因为需要。
- 谷歌的Android开发者API玩。您可以使用这个基于http的API来调查谷歌玩游戏的当前购买的状态订阅。你可以存储为每个purchaseToken购买的状态在你的后端服务器。有关更多信息,请参阅谷歌玩Android开发者API,下面。

对于大多数用例,尤其是在那些后端服务器已经跟踪订阅用户,实现两者的结合方法是推荐使用的方法。一个典型的实现可能会是这样工作的:

When the user successfully purchases a new subscription, your app notifies a backend server, which stores the purchase token, user name, and other information in a secure location. Since your app cannot know the expiration date, your server can poll Google Play to get the expiration and store it with the purchase token and other data. Because your server now knows the

expiration date, it does not need to poll Google Play again until after the expiration date, at which time it can confirm that the subscription was not cancelled. On the client side, your app can continue to update the server whenever the purchase state changes, storing the state locally. If you are using both notifications and the Google Play Android Developer API to validate subscriptions, we recommend the following:

If your app wants to check validity but you can't reach your server (or you don't have a server), use the latest purchase state received by notification. If you have a server and it's reachable, always give preference to the purchase state obtained from your server over the state received in notifications. If necessary, you can also use a RESTORE_TRANSACTIONS request to retrieve a record of all managed and in-app products purchased by the user, which you can then store locally. However, using RESTORE_TRANSACTIONS on a regular basis is not recommended because of performance impacts.

Regardless of the approach you choose, your app should check subscriptions and validity at launch, such as prior to accessing subscriber content, game levels, and so on.

当用户成功地购买一个新的订阅,你的程序会通知一个后端服务器,该商店购买令牌、用户名和其他信息在一个安全的地方。

因为你的应用程序无法知道有效期,服务器可以调查谷歌发挥得到和到期存放在这个令牌和其他数据的购买。

因为你的服务器现在知道的过期日期,它不需要调查谷歌后再玩的过期日期,在这段时间里,它可以确认没有取消订阅。

在客户端,你的应用程序可以继续更新服务器每次购买状态发生变化时,在本地保存状态。

如果您使用的是两个通知和谷歌的Android开发者API来验证订阅,我们推荐以下:

如果您的应用程序要检查有效性但你不能达到你的服务器(或者您没有一个服务器),使用最新的采购状态收到通知。

如果你有一个服务器,它是可获得的,总是优先将采购状态从服务器获得在国家中收到通知。

如果需要,您还可以使用一个恢复交易请求来检索一个记录所有的管理和应用内购买产品的用户,然后可以在本地存储。然而,使用恢复定期交易并不可取,因为性能影响。

不管你选择的方法,你的程序应该检查订阅和有效性发行时,比如访问订阅内容之前,游戏关卡,等等。

Table 1. Summary of purchaseState values for subscription purchases, as received with a PURCHASE_STATE_CHANGED intent.

State	Comments
Purchased successfully	Sent at original purchase only (not at recurring billing cycles).
Cancelled	Sent at original purchase only if the purchase has failed for some reason

Launching your product page to let the user cancel or view subscriptions

In-app Billing does not currently provide an API to let users directly view or cancel subscriptions from within the purchasing app. Instead, users can launch the Play Store app on their devices and go to the My Apps screen to manage subscriptions. In My Apps, users can see a list of their subscriptions organized by application. Tapping one of the subscriptions loads the app's product page, from which users can see active subscriptions and billing status and cancel subscriptions as needed.

To make it easier for users to find and manage their subscriptions from inside your app, we recommend that you offer a "View My Subscriptions" or "Manage Subscriptions" option in your UI that directly loads your app's product page in the Play Store app.

To do this, create an intent with the ACTION_VIEW action and include the market:// URI (rather than the http:// URI) of your app's details page. Here's an example:

应用收费目前并不提供一个API来让用户直接从内部视图或取消订阅的采购程序。相反,用户可以启动这个应用程序在他们的设备上玩商店和去我的应用程序界面管理订阅。在我的应用程序,用户可以看到他们的订阅列表组织应用程序。攻丝的一个订阅加载应用的产品页面,用户可以看到活跃的订阅和记账地位和根据需要取消订阅。

为了方便用户查找和管理他们的订阅应用程序内部,我们建议您提供“查看我的订阅”或“管理订阅”选项在你的UI,直接加载你的应用的产品页面在剧中商店应用程序。

为此,创建一个意图与操作视图操作,包括市场:// URI(而不是网站的URI)您的应用程序的细节页面。这里的一个例子:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("market://details?id=com.example.app"));
startActivity(intent);
```

Recurring billing, cancellation, and changes in purchase state

Google Play notifies your app when the user completes the purchase of a subscription, but the purchase state does not change over time, provided that recurring billing takes place successfully. Google Play does not notify your app of a purchase state change until the subscription expires because of non-payment or user cancellation.

Over the life of a subscription, your app does not need to initiate any recurring billing events — those are all handled by Google Play and they are transparent to your application if billing is successful.

When the user cancels a subscription during an active billing cycle, Google Play does not notify your app immediately of the change in purchase state. Instead, it waits until the end of the active billing cycle and then notifies your app that the purchase state has changed to "Expired".

Similarly, if payment for the next billing cycle fails, Google Play waits until the end of the active billing cycle and then notifies your app at that time that the purchase state has changed to "Expired".

Your app can handle user cancellation and non-payment in the same way,

since both cause a change to the same "Expired" purchase state. Once the purchase state has become "Expired", your app does not need to grant further access to the subscription content. 谷歌打通知你的应用中,当用户完成购买订阅,但采购状态不随时间变化,只要重复计费发生成功。谷歌玩不通知你的应用程序的一个采购状态改变直到订阅过期因为缺钱或用户取消。

一生的订阅,你的应用程序不需要启动任何重复计费事件——所有这些都是由谷歌打球,他们对您的应用程序是透明的,如果计费是成功的。

当用户取消订阅一个活跃的结算周期期间,谷歌玩不立即通知你的应用程序中所发生的变化采购状态。相反,它会等到最后的活跃的结算周期,然后会通知你的应用程序,购买状态更改为“过期”。

类似地,如果支付下次帐单周期失败,谷歌玩会等到最后的活跃的结算周期,然后会通知应用程序当时的采购状态更改为“过期”。

你的应用程序可以处理用户的取消和未付款以同样的方式,因为他们都发生了变化,同样的“过期”采购状态。一旦采购状态已经成为“过期”,你的应用程序不需要格兰特进一步访问订阅内容。

Modifying your app for subscriptions

For subscriptions, you make the same types of modifications to your app as are described in [Modifying your Application Code](#).

Note that, in your UI that lets users view and select subscriptions for purchase, you should add logic to check for purchased subscriptions and validate them. Your UI should not present subscriptions if the user has already purchased them.

为订阅,你犯同样的类型的修改你的应用程序作为描述修改您的应用程序代码。

注意,在你的用户界面,让用户订阅视图并选择购买,您应该添加逻辑来检查购买订阅和验证它们。你的UI应该不会订阅如果用户已经购买了他们。

Administering Subscriptions

To create and manage subscriptions, you use the tools in the Developer Console, just as for other in-app products.

At the Developer Console, you can configure these attributes for each subscription product:

Purchase Type: always set to "subscription" Subscription ID: An identifier for the subscription Publishing State: Unpublished/Published Language: The default language for displaying the subscription Title: The title of the subscription product Description: Details that tell the user about the subscription Price: USD price of subscription per recurrence Recurrence: monthly or yearly Additional currency pricing (can be auto-filled) For details, please see Administering In-app Billing.

创建和管理订阅,你使用工具的开发人员控制台,就像其他应用内的产品。

在开发人员控制台,您可以配置这些属性为每个订阅行为产品:

购买类型:总是设置为“订阅”

订阅ID:一个标识符的认购

发布状态:出版/发表

语言:默认语言显示订阅

标题:标题的订阅的产品

描述:细节,告诉用户关于订阅

价格:美元的订阅价格每复发

复发:每月或每年

额外的货币定价(可以自动填写)

有关详情,请参阅管理应用收费。

Google Play Android Developer API

Google Play offers an HTTP-based API that you can use to remotely query the validity of a specific subscription at any time or cancel a subscription. The API is designed to be used from your backend servers as a way of securely managing subscriptions, as well as extending and integrating subscriptions with other services.

谷歌提供了一个基于**http**的**玩API**,您可以使用远程查询的有效性在任何时间特定订阅或取消订阅。这个**API**是设计用来从你的后端服务器作为一种安全的管理订阅以及扩展和集成订阅和其他服务。

Using the API

To use the API, you must first register a project at the Google APIs Console and receive a Client ID and shared secret that your app will present when calling the Google Play Android Developer API. All calls to the API are authenticated with OAuth 2.0.

Once your app is registered, you can access the API directly, using standard HTTP methods to retrieve and manipulate resources, or you can use the Google APIs Client Libraries, which are extended to support the API.

The Google Play Android Developer API is built on a RESTful design that uses HTTP and JSON, so any standard web stack can send requests and parse the responses. However, if you don't want to send HTTP requests and parse responses manually, you can access the API using the client libraries, which provide better language integration, improved security, and support for making calls that require user authorization.

For more information about the API and how to access it through the Google APIs Client Libraries, see the documentation at:

使用**API**,您必须首先注册一个项目在**谷歌API**控制台和接收客户**ID**和**共享密钥**,您的应用将展示当调用**谷歌Android开发者API**玩。所有**API**调用被验证使用**OAuth 2.0**。

一旦应用程序注册,您可以访问**API**直接使用标准的**HTTP**方法来检索和操作资源,或者您可以使用**谷歌API**客户端库,这是扩展到支持**API**。

谷歌的**Android开发者API**玩是建立在一个**RESTful**设计使用**HTTP**和**JSON**,因此任何标准**web**堆栈可以发送请求和解析响应。然而,如果你不想发送**HTTP**请求和解析响应手动,您可以访问**API**使用的客户端库,它提供更好的语言集成,提高安全性

和支持,用于调用,需要用户授权。

API的更多信息,以及如何使用它通过谷歌API客户端库,请参阅文档:

<https://developers.google.com/android-publisher/v1/>

Quota

Applications using the Google Play Android Developer API are limited to an initial courtesy usage quota of 15000 requests per day (per application). This should provide enough access for normal subscription-validation needs, assuming that you follow the recommendation in this section.

If you need to request a higher limit for your application, please use the “Request more” link in the Google APIs Console. Also, please read the section below on design best practices for minimizing your use of the API.

在应用程序使用谷歌Android开发者API玩被限制到一个初始的礼貌使用配额的每天15000个请求(每个应用程序)。这应该提供足够的访问为正常订阅确认需求,假设您遵循建议在这一节中。

如果你需要要求较高的限制对于您的应用程序,请使用“请求更多”的链接在谷歌api控制台。同时,请阅读下面部分设计最佳实践,以减少使用的API。

Authorization

Calls to the Google Play Android Developer API require authorization. Google uses the OAuth 2.0 protocol to allow authorized applications to access user data. To learn more, see Authorization in the Google Play Android Developer API documentation.

调用谷歌Android开发者API要求授权玩。谷歌使用OAuth 2.0协议允许授权的应用程序访问用户数据。欲了解更多,请看授权在谷歌的Android开发者玩API文档。

Using the API efficiently

Access to the Google Play Android Developer API is regulated to help ensure a high-performance environment for all applications that use it. While you can request a higher daily quota for your application, we highly recommend that you minimize your access using the technique(s) below.

- Store subscription expiry on your servers — your servers should use the Google Play Android Developer API to query the expiration date for new subscription tokens, then store the expiration date locally. This allows you to check the status of subscriptions only at or after the expiration (see below).
- Cache expiration and purchaseState — If your app contacts your backend servers at runtime to verify subscription validity, your server should cache the expiration and purchaseState to ensure the fastest possible response (and best experience) for the user.
- Query for subscription status only at expiration — Once your server has retrieved the expiration date of subscription tokens, it should not query the Google Play servers for the subscription status again until the subscription is reaching or has passed the expiration date. Typically, your servers would run a batch query each day to check the status of expiring subscriptions, then update the database. Note that:

Your servers should not query all subscriptions every day

Your servers should never query subscription status dynamically, based on individual requests from your Android application.

By following those general guidelines, your implementation will offer the best possible performance for users and minimize use of the Google Play Android Developer API.

访问谷歌的**Android开发者API**是监管玩一个高性能环境,以帮助确保对所有使用它的应用程序。虽然您可以要求一个更高的每日配额对于您的应用程序,我们强烈建议你减少你的访问使用技术(s)以下。

- 在您的服务器上存储订阅过期——你的服务器应该使用谷歌**Android开发者API**查询玩的过期日期为新的订阅令牌,然后存储在本地的过期日期。这允许您检查状态的订阅只在或到期后(见下文)。
- 缓存过期和**purchaseState**——如果你的应用程序在运行时接触你的后端服务器来验证订阅效度,你的服务器应该缓存过期和**purchaseState**确保最快的响应(和最好的经验)的用户。
- 查询订阅状态只有在过期——一旦你的服务器检索过期日期的订阅令牌,它不应该查询谷歌服务器玩订阅的状态才能再次订阅达到或已通过的过期日期。

通常,您的服务器将运行一个批处理查询,每天检查状态的过期订阅,然后更新数据库。注意:

你的服务器不应查询所有订阅每一天

你的服务器不应该查询订阅状态动态,基于单个请求从你的Android应用程序。

按照这些通用的指导原则,您的实现将提供最佳的性能,减少用户使用谷歌Android开发者API玩。

来自 "[index.php?title=Subscriptions&oldid=7824](#)"



Security and Design

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：玄月冰灵

原文链

接：

http://developer.android.com/guide/google/play/billing/billing_best_practices.html

目录

[\[隐藏\]](#)

1 安全和设计

- [1.1 安全最佳实践](#)
 - [1.1.1 在服务器上完成签名验证任务](#)
 - [1.1.2 保护你的未加密内容](#)
 - [1.1.3 打乱你的代码](#)
 - [1.1.4 修改所有的样例应用代码](#)
 - [1.1.5 使用安全随机数](#)
 - [1.1.6 采取行动对抗商标和版权侵犯](#)
 - [1.1.7 保护你的Google Play公钥](#)

安全和设计

当你在设计实现你的应用内支付时，确保跟随在这个文档中讨论的安全与设计指南。这些向导是为使用Google Play应用内支付服务推荐的最佳实践。

安全最佳实践

在服务器上完成签名验证任务

如何实践，你应该在一个远程服务器上而不是设备上完成签名验证。在服务器上

实现验证处理使攻击者很难通过逆向工程你的APK文件来破坏验证处理。如果你向一个远程服务器卸载安全处理，确保设备到服务器的握手是安全的。

保护你的未加密内容

要阻止恶意用户重新发布你的未加密内容，不要把它和你的apk文件捆绑，以以下一种方法代替：

- 使用实时服务发送你的内容，例如内容源。通过实时服务发送内容允许你的内容始终为最新的。
- 使用远程服务器发送你的内容。

当你从远程服务器发送内容或使用实时服务，你可以把未加密内容存储在设备内存中或SD卡中。如果存储在SD卡中，确保加密内容且使用设备指定的密钥。

打乱你的代码

你应该打乱你的应用内支付代码使攻击者很难逆向工程你的安全协议和其他应用组件。在最低程度，我们推荐你在代码中使用Proguard打乱工具。

除了运行打乱程序时，我们推荐你使用以下技术打乱应用内代码。

- 内联方法进入其他方法
- 使字符串忙碌代替定义为常量
- 使用JAVA映像访问方法

使用这些技术可以帮助减少你应用的攻击面和帮助最小化攻击来妥协应用内支付的实现。

注释：如果你使用Proguard来打乱代码，你必须在Proguard设置文件中使用以下行： -keep class com.android.vending.billing.**

修改所有的样例应用代码

应用内支付样例代码是公开发行的，任何人都可以下载，这也意味着如果你使用样例代码正如它发布时那样，将使攻击者非常容易逆向工程你的应用。样例代码应用只打算作用一个例子使用。如果你使用样例应用的任何部分，你必须在你发布前修改它，或作为产品应用的一部分释出。

特别的是，攻击者寻找应用的入口和出口，因此对你来说修改你应用的这些和样

例代码相同的部分十分重要。

使用安全随机数

随机数必须不可预测和重复。总是使用一种加密安全随机数生成器（例如SecureRandom）当你生成随机数，这可以帮助减少重复攻击。

也就是，如果你在一个服务器上执行随机数验证，确保你在服务器上生成随机数。

采取行动对抗商标和版权侵犯

如果你看到你的内容在Google Play被重新发布，果断快速采取行动。文件商标侵犯通知和版权侵犯通知。

如果你使用远程服务器发送或管理内容，使你的应用无论何时进入内容验证未加密内容的购买状态，这将允许你必要时撤销使用和减少盗版。

保护你的**Google Play**公钥

要在恶意用户和黑客面前保护你的公钥安全，不要嵌入任何例如文字字符串的代码，一部分在运行时构建字符串或使用位运算（例如：异或一些其他字符串）来隐藏真实的密钥。密钥并不是密码信息，但是如果你不想使黑客或恶意用户很容易就用任何密钥替代公钥的话。

来自 "[index.php?title=Security_and_Design&oldid=9006](#)"



Testing In-app Billing

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址

址:http://docs.eoeandroid.com/guide/google/play/billing/billing_testing.html

译者:十旋转45度

更新时间: 2012年9月12日

目录

[[隐藏](#)]

1 测试应用的内部付费机制

- [1.1 利用静态响应对应用的内部购买行为进行测试](#)
- [1.2 使用自己的商品ID测试应用的内部购买行为](#)

[内部付费机制概述 - Overview of In-app Billing](#)

[内部付费机制的实现 - Implementing In-app Billing](#)

[设计与安全 - Security and Design](#)

[应用程序付费机制管理 - Administering In-app Billing](#)

[内部付费机制参考 - In-app Billing Reference](#)

[下载应用样例](#)

测试应用的内部付费机制

Google Play发布网站提供一系列工具可帮助你在应用程序发布前对其内部付费接口进行测试。利用这些工具可以创建测试帐户并购买测试用的预留商品，购买行为将会给您的应用发送静态付费响应[static billing responses]。

要想测试某应用程序的内部付费机制，你必须先将应用安装到Android设备上，Android模拟器是无法进行内部付费机制测试的。用于测试的设备必须运行Android 1.6或以上（API level 4以上）的标准版本，并已安装最新版的Google Play应用。若设备运行的Google Play应用不是最新版，那你的测试应用将无法给Google Play应用发送付费请求。欲知开发Android应用程序要对设备如何设置，

请参阅设备硬件的使用

下一节介绍，如何设置并使用测试应用程序内部付费机制的工具。

利用静态响应对应用的内部购买行为进行测试

建议先用Google Play发出的静态响应测试应用的内部付费接口。这保证了应用程序能正确处理Google Play的响应，保证了应用程序能够正确地验证签名。

要想用静态响应来进行测试，你要利用带有预留商品ID的物品先创建内部付费请求。每个预留商品ID会返回一条发自Google Play的一条特定静态响应。利用预留商品ID创建的内部付费请求是不会真正扣费的，于是也就无法指定付款方式了。

图1展示了商品ID为 android.test.purchased的预留商品的结账流程。

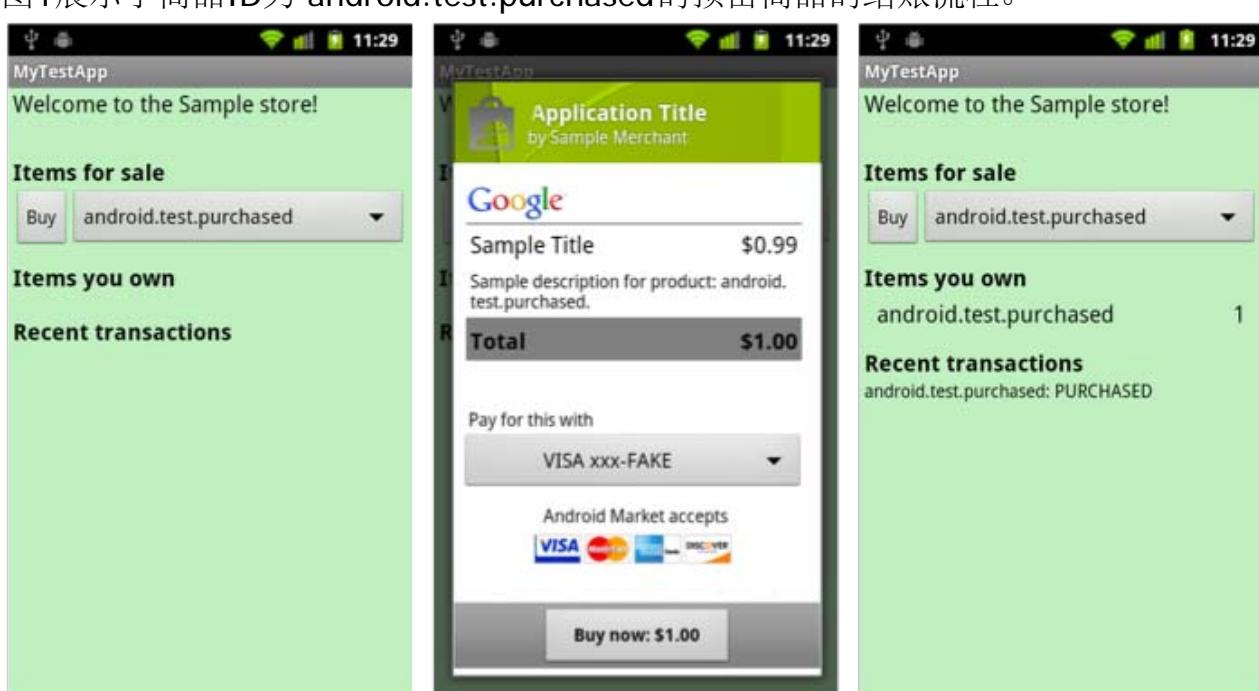


图1. 预留商品 android.test.purchased 的购买流程。

你无需在应用程序中将预留商品罗列出来，Google Play本身保有预留商品ID。此外，你也不需要为了测试静态响应而将应用上载到发布网站上，简单地在设备上安装应用并登录，就可以进行测试。

用于测试应用内部付费静态响应的预留商品ID有4个：

- **android.test.purchased**

当使用本商品ID发出内结算请求，Google Play会按照购买成功的情况对此进行响应。响应包括一个JSON字符串，其中带有虚拟购买信息（如，虚拟订单ID）。在某些情况下，JSON字符串带有签名，这样你就可以利用这些响应对验证签名的接口进行测试。

- **android.test.canceled**

当使用本商品ID发出内结算请求，Google Play会按照取消购买的情况对此进行响应。这种情况可能发生于订购过程出错，如无效的信用卡，或支付前取消用户订单。

- **android.test.refunded**

当使用本商品ID发出内结算请求，Google Play会按照退款的情况对此进行响应。退款必须由你（卖家）发起。当你通过Google Wallet帐户发出一条退款请求，Google Play会给应用发出一条退款消息。这种情况只发生在Google Play得到Google Wallet通知已退款之后。欲了解更多退款相关信息，请参阅[处理IN_APP_NOTIFY消息](#)和[应用程序内结算定价](#)。

- **android.test.item_unavailable**

当使用本商品ID发出内结算请求，Google Play会按照商品清单不存在此购买商品的情况对此进行响应。

某些情况下，预留项目会返回带签名的静态响应，这样你就可以对应用的签名验证进行测试。要想使用特定预留商品ID测试签名验证，你需要创建测试帐户或将你的应用作为非发布用的应用草案上传。表1展示了静态响应带签名的条件。

表1.静态响应带签名的条件

应用程序是否发布过？	应用草案是否上传并且未发布？	运行此应用程序的用户	静态响应签名
没有	没有	任何人	无签名
没有	没有	开发者	签名
有	没有	任何人	无签名
有	没有	开发者	签名
有	没有	测试账号	签名
有	有	任何人	签名

使用预留商品ID创建一条内部付费请求，只需构建一般

的**REQUEST_PURCHASE**请求，但是所用的商品ID不是从应用商品列表上选取的真正商品ID，而是用预留商品ID取代。

利用预留商品ID对应用程序进行测试，请遵循以下步骤：

1，将应用程序安装到**Android**设备上。

测试内部付费用模拟器是无法进行的，必须安装到真机设备上才能进行。
欲了解如何在设备上安装应用程序，请参阅[在设备上运行应用](#)。

2，在设备上登录你的开发者帐户

如果测试用的是预留商品ID，那你也无须使用测试帐号。

3，确保设备运行的是**Google Play**或**MyApps**应用受支持的版本。

若你的设备运行的是**Android 3.0**，那么应用内部付费的实现要求**MyApps**版本5.0.12以上。若设备运行的是**Android**的其他版本，那么内部付费要求**Google Play**应用版本2.3.4以上。欲了解如何检查**Google Play**版本，请参阅[Google Play的更新](#)。

4.运行应用，购买预留商品**ID**。

注：用预留商品**ID**创建的内部购买请求会将正常的**Google Play**商品体系覆盖。当你发出一个预留商品**ID**的请求，其涉及的服务与真实商品环境不具可比性。

使用自己的商品**ID**测试应用的内部购买行为

使用静态响应测试完成，并且应用程序上的签名验证也通过之后，你就可以用真实的购买行为测试应用程序内部购买机制了。测试真实的内部购买实际上就是测试端到端的内部购买，其包含发自**Google Play**的真实响应和用户在您的应用将体验到的真实结账流程。

注：端到端测试无需将应用发布，你只需将应用作为应用草案上传就可以进行测试。

用真实的购买行为测试应用内部付费接口，你需要在**Google Play**发布网站是注册至少一个测试帐号。你不能使用自己的开发者帐户来进行内部购买测试，因为**Google Wallet**不运行购买自己的商品。若你先前没有设置过测试帐户，请参阅[测试帐户的](#)

设置。

只有当商品列表中的商品被发布后，测试帐户才可以对其进行购买。应用程序并不需要发布，但商品商品是需要发布的。

当你使用测试帐户购买物品，测试帐户通过Google Wallet支付，同时你的Google Wallet Merchant帐户会收到付款。

用真实购买行为对应用程序内部付费进行测试，请遵循以下步骤：

1，将应用程序作为一个应用草案上传到发布网站。

进行真实商品ID的端到端测试无需将你的应用程序发布，只需将应用作为应用草案上传。但在上传之前，必须给你的应用程序以发布密钥签名。此外，上传的应用程序版本号必须与你设备上进行测试的应用版本号相匹配。欲了解如何将应用程序上传到Google Play，请参阅[应用程序的上传](#)。

2.给应用程序的商品列表添加物品。

请确认你已将物品发布（应用程序可以是未发布）。请参阅[创建商品列表](#)来学习如何实现。

3.在**Android**设备上安装应用程序。

测试应用程序内部付费时不能用模拟器进行的，你必须在真机设备上安装应用程序才能测试内部付费。

欲了解如何在设备上安装应用程序，请参阅[在设备上运行应用](#)。

4，将某个测试帐户设为设备上的主帐户。

为了进行应用程序内部付费端到端的测试，设备上的主账号必须有为某个Google Play网站上注册的[测试帐户](#)。若设备上的主帐户不是测试帐户，那你必须先恢复出厂设置，然后再登录测试帐户。恢复出厂设置，请遵循以下步骤：

1. 在设备上找到设置**[Settings]**。
2. 点选隐私**[Privacy]**。
3. 点选恢复出厂设置**[Factory data reset]**。
4. 点击重置手机**[Reset phone]**。
5. 手机重置后，一定要在设备启动过程登录测试账号。

，确保你的设备运行的

或

应用是受支持的版本。

若你的设备运行的是Android 3.0，那么应用内部付费的实现要求MyApps版本5.0.12以上。若设备运行的是Android的其他版本，那么内部付费要求Google Play应用版本2.3.4以上。欲了解如何检查Google Play版本，请参阅[Google Play的更新](#)。

6，在应用程序中进行内部购买。

注：更改设备主帐户的唯一方法是执行恢复出厂设置，请确保你登录的主帐户正确。

当应用程序内部付费接口测试完毕后，你就可以将应用程序发布到Google Play了。步骤可以参考[准备](#), [签名](#), 和[发布到Google Play](#)。

来自“[index.php?title=Testing_In-app_Billing&oldid=11033](#)”



Administering In-app Billing

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链接：http://developer.android.com/guide/google/play/billing/billing_admin.html

编辑者：天娘

完成时间：2012.8.10

目录

- [1 应用程序付费机制管理](#)
 - [1.1 建立产品列表](#)
 - [1.2 一次添加一个商品到产品列表](#)
 - [1.3 增加批量商品到产品列表](#)
 - [1.4 选择商品的购买类型](#)
 - [1.5 退费的处理](#)
 - [1.6 设定测试帐户](#)
 - [1.7 在哪里寻求支援](#)

应用程序付费机制管理

应用程序付费机制管理能使你从处理经济业务中解放出来，但是你仍然需要做一些管理任务，包括建立和维持你的产品清单在你的发布站点上，注册测试账户，需要的时候处理退款事件。

你必须有Google发布者帐号来注册测试帐号。同时你必须有Google Wallet merchant帐户来创建产品列表和退款问题。如果你已经有一个发布者帐号在Google Play上，你就能使用你当前的帐号。你不必再去注册一个新的帐号来支持应用程序付费机制管理。如果你没有发布者帐号，你可以注册成为一个GooglePlay开发者并且在GooglePlay [publisher site](#) 上建立一发布者账户。如果你没有Google Wallet merchant的帐号，你可以在[Google Wallet site](#) 上注册一个。

建立产品列表

Google Play发布者网站提供你的每一个发布的应用的产品列表。你要销售的应用程序内商品只能来自这张列表上的商品。每个应用程序都有自己的列表清单，你不能销售你的应用中的商品到其他的产品列表。

你可以通过点击In-APP Products 链接，进入应用程序产品目录，根据你的发布者帐号列出应用程序（见图1）。只要你有Google Wallet merchant帐户并且你的应用程序的manifest包含com.android.vending.BILLING权限。



图1：你可以通过点击In-App Products 链接进入应用程序产品列表

一个产品列表指定的商品是你要销售的应用程序-应用程序的产品，订阅，或者是两者都有。对了每个商品，产品列表包含他们的一些信息，如产品ID，产品的描述，和价格（看图2）。产品列表只存储你的销售的应用程序的元数据。它不存储任何数字的内容。存储和传送应用中的数字内容是你的责任。



图2：应用程序的产品列表

你可以为你已经发布的应用程序和已经上传并且已经存储在Google Play网站上的草稿型程序创建产品列表。然而，你必须要有Google Wallet merchant帐户，并且你的应用程序的manifest文件必须包含com.android.vending,BILLING权限。如果应用程序的manifest文件没有包含这个权限，你能够编辑在产品列表中已经存在的产品，但是你不能添加新的产品到列表中。了解更多的信息关于这个权限，请看[Updating Your Application's Manifest.](#)

另外，应用程序的套件只能有一个产品列表。如果你为一个应用程序创建一个产品列表，并且使用[multiple APK feature](#)来分配超过一个APK给你的应用程序，产品列表接受的是来自各种版本的所有连接。你无法替你的各个版本的APK建立特定的产品列表。

你有两种方法来为你的产品列表添加商品：你可以一次上传一个通过使用In-app Products用户界面（看图3），或者通过CSV文件导入商品（看图2）。如果你的应用只有很少的商品，如果你只是想为了测试而导入很少的商品到产品列表，那么你可以一次添加一个是个很有效的方法。如果你的应用有大量的商品，使用CSV文件添加很有用。

注意：尚未支持批次产品含有的订阅列表上传。

一次添加一个商品到产品列表

使用In-app Products用户界面添加商品，遵循这些步骤：

1. 登录你的发布者帐号。
2. 进入GooglePlay的列表界面，在应用名字的下面，点击In-app Products。
3. 在应用产品列表页，点击添加应用
4. 在创建新应用程序页面（见图3），提供你卖的商品的细节，然后点击保存或者发布按钮



图3：这个创建新应用程序页面，能让你在产品列表中增加商品。

你必须在产品列表中为你的项目输入下面的信息：

- 产品编号

产品编号是唯一的跨应用程序的命名空间。一个产品编号必须以小写字母或是数字开头，只能由小写字母(a-z),数字(0-9),下划线(_)和小数点(.)。产品编号“android.test”保留的，因为所有的产品编号都是以“android.test.”开头的。

另外，当你的产品编号产生后你不能修改的它，并且你不能在次使用这个相同名字的产品编号。

购买类型

购买类型可以管理每个用户帐户，非托管或认购。购买类型设置后，你就不能修改了。欲了解更多信息，请参阅稍后在本文档中选择购买类型的文章

- 发布状态

一个商品的发布状态可以是已经发布或是没有发布。用户在检验的时候可以看到这个商品的发布状态，这个商品的状态被设置成了已发布，那么这个商品一定在Google Play上发布了。

注意：这个是不适用于测试的帐号，如果应用没有发布，商品已经发布，这个商品对于测试帐号是可见的。查看[Testing In-app Billing](#) 了解更多信息。

- 语言

语言设置决定了在付费期间哪种语言能显示商品的标题和商品的描述。一个产品列表会继承默认的语言来自它的父应用程序。你可以增加更多的语言通过点击add language。你也可以选择让商品的标题和描述来自默认的语言自动翻译，通过选择Fill fields with auto translation 检查框（见图4）。如果你没有使用自动翻译功能，那么你必须提供商品标题和描述的版本信息。

- 标题

标题是对商品的简短描述。例如，“睡觉药剂”。标题必须是独特的，跨应用程序的命名空间。每个商品必须有一个标题。用户在付费的时候商品的标题是可见的。为了得到最适宜的外观，标题应该不超过25个字；而且，标题有55个字的长度限制。

- 描述

描述是介绍商品的长字串。例如，“此物品会马上让动物睡着，但无法对生气的小精灵起作用。”。每个商品必须有一个描述。用户在付费的时候描述是可见的。描述有80个字的长度限制。

- 价格

你必须使用你自己国家的汇率来提供一个默认的价格。你也可以使用其他的汇率来计算你的商品价格，但这么做只会在如果那个国家是你应用的主要市场。你可以在Google Play市场的开发人员控制里的编辑应用页面去指定销售的目标国家。如果要指定成其它汇率，你也可以手动的方式输入每个国家汇率而订的价格也可以点击自动填入的方式，让Google Play从你的国家汇率自动转成其他国家的汇率（见图4）。



图4.替每个商品的标题和描述指定其它的汇率和其它的语言。

了解更多信息关于产品编号和产品列表，请看[Creating In-App Product IDs](#)。了解更多的信息关于价格，请看[In-App Billing Pricing](#)。

注意：确保你能使用产品编号的命名空间，在你保存你的产品编号之后，你不能再次使用和修改你的产品编号。

增加批量商品到产品列表

使用CSV文件增加批量商品到产品列表，你首先需要创建你的CSV文件。CSV里的资料呈现方式就如同你在产品使用页面以手动方式新增商品一样。（见一次添加一个商品到产品列表）。CSV文件使用逗号（,）和分号（;）来分隔资料。逗号用来分隔主要资料的值，而分号用来分隔资料。举例，一个CSV文件里的语法应该像下面的这样子：

```
"product_id","publish_state","purchase_type","autotranslate","locale;title;description","autofill","country;price"
```

底下是描述和使用方式：

- 产品ID

这个值如果应用商品使用页面里产品ID设定值。如果产品列表里已经有这个产品ID,而且当你进入CSV你想覆盖商品使用页面里的产品ID的话，这个值就会被你CSV里所指定的新资料取代。覆盖的状况不会发生CSV里没有重复的产品ID的情形下。

- 发布日期

这个值等同于应用商品页面的发布状态设定，一样可以设定成已发布或未发布的状态。

- 支付类型

这个值等同于应用商品页面的购买状态设定值。使用帐号做管理的值是managed_by_android，不受管理的值是managed_by_publisher。

- 自动翻译

这个值等同于应用商品使用界面选择了自动翻译并填入选项的功能，可以是true，也可以是false。

- 地域

这个值等同于应用商品使用界面的语言设定。你必须设定一个预设的地域，而且预设的地域要也要设置在第1位，而且包含标题和描述。

如果你想要将自动翻译设置为预设值，你就必须使用底下的语法规则：

如果自动翻译设置这true，你必须指定预设的地域，预设的标题，预设的描述以及其他地域的使用格式：“true,”default_locale; default_locale_title; default_locale_description; locale_2; locale_3, ...”

如果自动翻译设置成false，你必须指定预设的地域，预设的标题，预设的描述，以及翻译过的标题，描述，并使用底下格式：“false,”default_locale; default_locale_title; default_locale_description; locale_2; locale_2_title; locale_2_description; locale_3; locale_3_title; locale_3_description; ...”

请见表1列出的你可以使用的语言代码和语言值

- 标题

这个值等同于应用商品页面里的标题设定。如果标题包含分号，就必须加上反斜线加以区分（如：“\\”）。

- 描述

这个值等同于应用商品页面里的描述值。如果描述值包含分号，就必须加以反斜线加以区分如（：“\\”）。如果含的是反斜线，就再加上发斜线来区分。如（：“\\\\”）。

- 自动填入

这个值等同于应用商品页面里的自动填入。可以设置为true或false。若要指定不同的国家或价格完全取决于您是否设定自动填入。

如果自定填入被设置为true，你就需要指定一个依你国家汇率设定出来的价格，并使用下面的语法：

"true","default_price_in_home_currency" 如果自动填入被设置为false，你就需要每个国家依不同的汇率来决定价格，并使用底下语法：

```
"false", "home_country; default_price_in_home_currency; country_2;country_2_price;country_3;
country_3_price; ..."
```

- 国家

你指定价格的国家，你可以设定你想要销售的目标国家。国家代码是大写的2个字（ISO国码，像是“US”），这个国码是被ISO 3166-2完善和规范的格式。

- 价格

这个值等同于应用商品使用界面的价格。必须指定为微量单位（micro-units）。若需要将下面的汇率换成微量单位，请乘上1, 000, 000。假如你想要销售的商品价格为美金\$1.99，这个值就是1990000。

表1.地区栏里你可以使用的语言代码

Language	Code	Language	Code
Chinese	zh_TW	Italian	it_IT
Czech	cs_CZ	Japanese	ja_JP
Danish	da_DK	Korean	ko_KR
Duth	nl_NL	Norwegian	no_NO
English	en_US	Polish	pl_PL
French	fr_FR	Portuguese	pt_PT
Finnish	fi_FI	Russian	ru_RU
German	de_DE	Spanish	es_ES

Hebrew	iw_IL	Swedish	sv_SE
Hindi	hi_IN	--	--

为了导入这些指定的元素到你的CSV文件中，做下面的步骤：

1. 登录你的发布者帐号。
2. 在GooglePlay列表面板，在应用名字的下面，点击In-app Products。
3. 在In-app Products 应用列表页面，点击选择文件，选中你的CSV文件。CSV文件必须在你本地的电脑上或者是在连接到你的电脑上的本地硬盘上。
4. 如果你想要覆盖已存在在您的产品列表中的商品，请选择覆盖选项。这个覆盖时机是在当应用产品列表的某些产品ID和CSV文件里面的product_id相同时。所谓的覆盖不会删除存在在产品列表而没有存在在CSV文件的商品。
5. 在In-app Products 列表页面，点击Import from CSV。

你可以使用应用商品列表产品列表的移到CSV功能将已有的产品列表移到CSV中。如果你已经手动添加商品到商品列表，然后你想要开始通过CSV管理你的产品项目时，这个方法会很有用。

选择商品的购买类型

商品的购买类型能控制GooglePlay以何种方式管理商品的购买行为。这里有2中购买类型，分别为：受管理（使用帐号）和不受管理。经常使用帐户管理的商品只能被每位使用者购买一次。当商品在这种状态下，GooglePlay会以每个使用者当基础来替每件商品储存交易资讯，而且会被永久储存。这种商品能让你可以透过发出RESTORE_TRANSACTIONS请求来跟GoolgePlay查询，并且能针对特定的使用者（再次安装应用时）还原该商品为已购买的状态。

如果使用者试图购买一件被管理的商品，但是该商品他早就买过了，GooglePlay就会显示出“商品已经买过了”的错误讯息。这个讯息会在当GooglePlay要在付费页面显示价格和商品描述资讯的付费流程中发生。当使用者没有看到这个错误讯息时，付费页面却消失了，页面返回到你的应用的使用页面，这种情况下，最好的方式我们认为：你的应用根本不应该让使用者看到该错误讯息，示范程序有说明你可以如何做到被管理类型的商品做出商品追踪，并且直接将购买过的商品在应用列表里隐藏。你现在的应用也应该如此。将那些已经销售过的商品反灰或隐藏、甚至设成无法被点选的状态。

如果你现在要销售的是游戏等级或是应用程序的特殊功能，“被使用帐户管理”的类型商品是很有用的。这些商品不会被传送、而且通常需要被还原，无论是使用者再度安装你的应用、或者使用者清楚了他们的装置资料、甚至是在新的装置上安装你的应用时，都很有用。

“不被管理的商品”，他们的交易资讯不会被存储在GooglePlay里。这意味着你无法针对购买商品类型设定为“不被管理”的商品，想GooglePlay做查询交易资讯的动作。此时，这些不被管理商品的交易资讯，管理责任就落在了你的手上了。而且，不被管理的商品可以无限次购买，完全交在你的手上。

如果你销售的是消费型商品，像是燃料或者是魔法法术，“不被管理”的商品类型就变得很有用了。这些商品都是在你的应用里的消耗型商品，而且通常会被购买多次。

退费的处理

应用程序付费机制管理不允许使用者发送退费请求到GooglePlay。应用程序内的商品的退费完全主导在你（APP的开发者）。你可以通过你的Google Checkout merchant帐号来处理退费流程。当你执行了商

品退费，Google Play就会收到从Google Checkout传来的退费通知，接着Google Play就会发送商品退费的讯息给使用者的APP。更多信息请看 [Handling IN_APP_NOTIFY messages](#)和[In-app Billing Pricing](#)这两篇文章。

重要：你不能使用Google Checkout的API来处理退费或取消交易。你必须以手动方式通过Google Checkout merchant 帐号来处理这些事。然而，你仍可以使用Google Checkout API来接收订单资讯。

设定测试帐户

Google Play发布网站能让你可以建立一个或多个测试帐号。一个测试帐号是一个标准的Google帐号，这个帐号是你在发布者站点上注册的测试帐号。测试帐号是被认证的，因此你可以通过上传却未发布的APP来做出购买请求。

你可以使用任何的Google帐户来当测试帐户。如果你想要让更多个人测试你app的应用程序付费机制，而你不需要把你的发布网站的登入帐户给他们来使用时，测试帐户就变得很有用了。如果你想要自己拥有也控制测试帐户，你可以自己建立帐户，并且将这组帐户设成开发者帐户或测试者帐户。

测试帐户有下面三种限制：

- 测试帐户的使用者只能在已经上传到你发布帐户的APP做购买请求。（即使应用程序不需要被发布出去）。
- 测试帐户只能支付在应用产品列表的商品（并且已经发布）。
- 测试帐户的使用者无法存取你的发布帐户，也无法上传应用到你的发布帐户里。

增加测试帐户到你的发布帐户里，遵循下面的步骤：

1.登录你的发布帐户。2.在页面的上方的左侧，你的名字底下，点击Edit profile。3.在测试帐户中，为你想要的注册的测试帐户添加邮箱地址，多个帐户使用逗号隔开。4.点击Save来保存你的内容更改。



图5.这是帐户中编辑个人资料里的授权和应用程序内结账区域，你可以在这里注册测试帐号。

在哪里寻求支援

如果你在操作应用程序付费机制管理时遇到问题和困难，可以联系表格中列出的支援的资源列表（见表2）。依你的需求寻求合适的论坛，你可以更快速地获得你所需要的支援。

表2.Google Play应用程序付费机制管理开发者支持的资源。

支援的形态	开发与测试问题	主题范围
开发和测试问题	Google Groups: android-developers	应用程序付费机制的整合问题、使用者经验、处理回应、模糊程式码、内部进程的沟通、测试环境的设定。

	栈溢出： http://stackoverflow.com/questions/tagged/android	
账单 问题 追踪	Billing project issue tracker	关于应用程序付费机制的Bug和 问题会报

想要了解如何发文到上面的论坛的基本资讯。请见资源栏里的[Developer Forums](#)文章。

来自“[index.php?title=Administering_In-app_Billing&oldid=8498](#)”

In-app Billing Reference

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址:http://docs.eoeandroid.com/guide/google/play/billing/billing_reference.html

译者:十旋转45度

更新时间: 2012年9月13日

目录

[[隐藏](#)]

[1 应用内部付费机制参考](#)

- [1.1 应用程序内部付费的服务器响应码](#)
- [1.2 应用程序内部付费服务的接口](#)
- [1.3 应用程序内部付费广播Intent](#)
- [1.4 验证和取消的HTTP API](#)
- [1.5 应用内部付费API版本](#)

[内部付费机制概述 - Overview of In-app Billing](#)

[内部付费机制的实现 - Implementing In-app Billing](#)

[设计与安全 - Security and Design](#)

[应用程序付费机制管理 - Administering In-app Billing](#)

[程序内付费机制测试 - In-app Billing Reference](#)

[下载应用样例](#)

应用内部付费机制参考

[应用程序内部付费的服务器响应码](#)

下表列出了所有从Google Play发送到你的应用程序的服务器响应码。Google Play将这些响应码作为**com.android.vending.billing.RESPONSE_CODE**广播intent的**response_code**附加信息[extras]异步发送。你的应用程序必须处理这些所有响应码。

表1. Google Play返回的响应码汇总

响应码	值	描述
RESULT_OK	0	表示该请求成功发送至服务器。若 CHECK_BILLING_SUPPORTED 请求返回此响应代码，那么支持付费行为。
RESULT_USER_CANCELED	1	表示用户在付款页面选中的是返回按钮而非购买物品。

RESULT_SERVICE_UNAVAILABLE	2	表示网络连接中断。
RESULT_BILLING_UNAVAILABLE	3	表示应用程序内部付费不可用，原因是指定的 API_VERSION 不为Google Play应用识别，或用户是没有结算在应用程序内部付费的权限（如，用户所处地区禁止在应用程序内部付费行为）
RESULT_ITEM_UNAVAILABLE	4	表示Google Play在应用程序的商品列表上找不到请求的物品。产生这种情况的原因可能是你的 REQUEST_PURCHASE 请求中商品ID拼写错误，或者这个物品并未在应用程序的商品列表中发布。
RESULT_DEVELOPER_ERROR	5	表示应用程序试图创建一条应用程序内部付费请求，但应用程序没有在manifest中声明com.android.vending.BILLING的许可；也可表示应用程序签名不正确，或者你发送的请求不正确，如Bundle key丢失，或使用了无法识别的请求类型。
RESULT_ERROR	6	表示产生了意外的服务器错误。如，你尝试购买自己的物品而触发的错误，这不为Google Wallet允许。

应用程序内部付费服务的接口

以下部分介绍Google Play的应用程序内部付费服务的接口。接口是在**IMarketBillingService.aidl**文件中定义的，该文件可在[应用程序内部付费示例](#)中找到。

接口由一个请求方法**sendBillingRequest()**构成。该方法只需一个**Bundle**参数。Bundle参数包括几个键-值对，在表2中列出：

表2 传给**sendBillingRequest()**请求的Bundle键的描述

键	类型	可能的值	是否必需？	描述
BILLING_REQUEST	String	CHECK_BILLING_SUPPORTED , REQUEST_PURCHASE , GET_PURCHASE_INFORMATION , CONFIRM_NOTIFICATIONS , 或 RESTORE_TRANSACTIONS	是	使用 sendBillingRequest() 创建的付费请求的类型。可能的值在本表后续部分会详细讨论。
API_VERSION	int	“2” [详细信息] “1” [详细信息]	是	您要使用的Google Play应用内部付费服务的版本，当前版本为2。
PACKAGE_NAME	String	有效的包名称。	是	发出请求的应用程序名称。
ITEM_ID	String	任何有效的商品标识。	对 REQUEST_PURCHASE 请求来说是必需的。	创建内部付费请求所涉及商品ID。使用Google Play的应用程序内部付费服务所贩卖的物品在Google Play发布网站必需具有独一无二商品ID。
NONCE	long	任何有效的 long 值。	对 GET_PURCHASE_INFORMATION 和 RESTORE_TRANSACTIONS 请求来说是必需的。	只使用一次的数字。你的应用程序必须用 GET_PURCHASE_INFORMATION 和 RESTORE_TRANSACTIONS 请求生成并发送一个随机数。该随机数用 PURCHASE_STATE_CHANGED 广播intent返回，这样你就可以用这个值来验证Google Play发回的交易响应是否完整。
NOTIFY_IDS	long 值的数组	任何有效的 long 值数组	对 GET_PURCHASE_INFORMATION 和 CONFIRM_NOTIFICATIONS 请求来说是必需的。	通知标识符的一个数组。每次购买状态发生改变时，通知ID就会通过广播intent IN_APP_NOTIFY 发送到你的应用程序。你可以使用该通知来检索的详细信息的购买状态变化。
				开发者指定的字符串，通过 REQUEST_PURCHASE 请求指定。本字段在

DEVELOPER_PAYLOAD	String	任何有效的字符串，长度不超过256字符	否	包含订单交易信息的JSON字符串中返回。用此键可发送订单的附加信息，例如，可以使用此键发送订单的索引键，使用数据库来存储购买信息情况，这是非常有用的。建议不要使用此键来发送数据或内容。
--------------------------	---------------	---------------------	---	--

BILLING_REQUEST键可以有以下值：

- **CHECK_BILLING_SUPPORTED**

本请求用于验证 Google Play 应用程序是否支持内部付费。通常在应用启动的时候发送此请求，用以判断启用还是禁用某些与内部付费相关的UI功能。

- **REQUEST_PURCHASE**

此请求用于给Google Play应用发送购买消息，是进行内部付费的基石。当用户表明其想要购买你应用程序中的某商品时，你就需要发送此请求。接着，Google Play通过用户介绍界面来处理交易。

- **GET_PURCHASE_INFORMATION**

此请求用于检索购买状态变化的详细信息。当购买请求支付成功或者结账时取消交易，或是先前交易退款的情况下，购买状态就会改变。购买状态发生改变的时候，Google Play会通知你的应用，这种情况下你只需要发送此请求就可检索交易信息。

- **CONFIRM_NOTIFICATIONS**

该请求用于确认你的应用程序接收到购买状态变化的详细信息。也就是说，此消息表明你发送了给定通知的**GET_PURCHASE_INFORMATION**请求，并且接收到该通知的购买信息。

- **RESTORE_TRANSACTIONS**

此请求用于检索用户的交易状态以管理购买（更多信息，请参阅[选择购买类型](#)）。只有当你需要获取用户交易状态时才需要发送此消息，通常为应用程序在设备上首次安装或重装的情况。

应用程序的每条内部付费请求会生成同步响应。响应是一个**Bundle**，可以包括以下一个或多个键：

- **RESPONSE_CODE**

此键提供了请求的状态信息和错误信息。

- **PURCHASE_INTENT**

此键提供了[PendingIntent](#)，用于启动结算activity。

- **REQUEST_ID**

此键提供了请求标识，用它来匹配异步请求的响应。

键与不同类型的请求之间有某种匹配联系。表3展示每个请求类型所返回的相应的键。

表3 应用程序内部付费的请求的每种类型对应的Bundle键的说明。

请求类型	键返回	可能的响应码
CHECK_BILLING_SUPPORTED	RESPONSE_CODE	RESULT_OK, RESULT_BILLING_UNAVAILABLE, RESULT_ERROR, RESULT_DEVELOPER_ERROR
REQUEST_PURCHASE	RESPONSE_CODE, PURCHASE_INTENT, REQUEST_ID	RESULT_OK, RESULT_ERROR, RESULT_DEVELOPER_ERROR
GET_PURCHASE_INFORMATION	RESPONSE_CODE, REQUEST_ID	RESULT_OK, RESULT_ERROR, RESULT_DEVELOPER_ERROR
CONFIRM_NOTIFICATIONS	RESPONSE_CODE, REQUEST_ID	RESULT_OK, RESULT_ERROR, RESULT_DEVELOPER_ERROR
RESTORE_TRANSACTIONS	RESPONSE_CODE, REQUEST_ID	RESULT_OK, RESULT_ERROR, RESULT_DEVELOPER_ERROR

应用程序内部付费广播Intent

一下内容是Google Play发送的应用程序内部付费广播intent的说明。这些广播intent将已发生的内部付费行为通知到你的应用程序。你的应用程序必须接入BroadcastReceiver才能接收广播intent，如[应用程序内部付费示例](#)中的BillingReceiver。

com.android.vending.billing.RESPONSE_CODE

此广播intent含有一个Google Play响应码，在你创建完内部付费请求之后发送。服务器响应码可以表示，付费请求成功发送到Google Play，或者付费请求出现了错误。此intent不是用来报告购买状态的变化（如退款或购买信息）。此响应码相关的更多信息，请参阅[Google Play内部结算响应码](#)。示例中赋予此广播intent一个常量ACTION_RESPONSE_CODE。

附加信息[Extras]

long型的请求ID。请求ID定义特定的付费请求，发出请求后Google Play会返回此ID。
response_code **int**型的Google Play服务器响应码。

com.android.vending.billing.IN_APP_NOTIFY

此响应表明，购买状态发生了改变，购买成功，取消或者退款。该响应包含一个或多个通知ID。每个通知ID对应一个特定的服务器端消息，每一个消息包含一个或多个交易的信息。你的应用程序接收到一个IN_APP_NOTIFY广播intent后，你用通知ID发送GET_PURCHASE_INFORMATION请求来检索信息的详细信息。该示例赋予此广播intent一个常量ACTION_NOTIFY。

附加信息[Extras]

notification_id **String**型通知ID，提示购买状态变化。当购买状态发生变化时Google Play会通知你，通知包含一个独一无二的通知ID。使用GET_PURCHASE_INFORMATION请求发送通知ID，可以获取购买状态改变相关的更多详情。

com.android.vending.billing.PURCHASE_STATE_CHANGED

此广播intent包含一个或多个交易的详细信息。交易信息存在于JSON字符串中。JSON字符串带签名，签名是和JSON字符串（未加密）一同发送到你的应用程序的。你的应用程序可以验证这个JSON字符串的签名以确保你的应用内部付费的信息安全。该示例赋予此广播intent一个常量ACTION_PURCHASE_STATE_CHANGED。

附加信息[Extras]

inapp_signed_data **String**型带签名的JSON字符串。
inapp_signature **String**型签名。

注：你的应用程序必须将广播intent和附加信息映射到的常量对应用程序来说必须独一无二。参阅示例中的**Consts.java**文件的做法。

JSON字符串中的字段描述如下表（见表4）：

表4 PURCHASE_STATE_CHANGED intent返回的JSON字段描述。

字段	描述
nonce	只使用一次的数字。你的应用程序生成此随机数，并用 GET_PURCHASE_INFORMATION 请求发送。Google Play将此随机数作为JSON字符串的一部分回传，这样你就可以验证消息的完整性。
notificationId	使用 IN_APP_NOTIFY 广播intent发送的独一无二的标识符。每个 notificationId 对应一条在Google Play服务器待检索的特定消息。你的应用程序通过 GET_PURCHASE_INFORMATION 消息向Google Play回传 notificationId ，这样Google Play就可以判断哪些信息要检索。
orderId	交易的唯一标识符，对应Google Wallet Order ID。
packageName	购买行为发自哪个应用程序包。
productId	物品的商品标识。每个物品都有一个商品ID，你必须在Google Play发布网站的应用商品列表中指定。
purchaseTime	商品的购买时间，以毫秒为单位（1970年1月1日之后）。
purchaseState	订单的购买状态。可能的值为0（已购买），1（取消购买），2（退款），或3（已过期，只用于购买订阅）。
purchaseToken	标识订阅购买给定物品或和用户对的唯一标记。查询订阅有效性时可以使用该标记指定订阅。只在应用程序内部付费API第2版和更高版本得到支持。
developerPayload	开发者指定的字符串，其包含订单相关的补充信息。当创建 REQUEST_PURCHASE 请求时，你可以指定此字段的值。

验证和取消的HTTP API

Google Play提供了基于HTTP的API，你可以在任何时间用它来远程查询特定订阅的有效性或取消订阅。此API被设计用于后端服务器安全地管理订阅，以及用其他服务拓展和整合订阅。参阅[Google Play Android开发者API](#)的更多信息。

应用内部付费API版本

不同版本的API都有自己附加功能。运行时，你的应用可以查询Google Play应用以确定它支持哪个版本的API，以及那些功能可用。通常情况下，Google Play应用更新将支持最新版本的API。版本汇总，请参阅[应用程序内部付费API版本](#)。

后面的部分列出了应用程序内部结算API的支持版本。Bundle对象作为参数传给**sendBillingRequest()**，而使用的 API版本由对象的**API_VERSION**键指定。**sendBillingRequest()**是在**IMarketBillingService.aidl** 文件中定义的，该文件可在应用程序内部付费[示例](#)中找到。更多详情，请参阅[应用程序内部付费服务接口](#)。

在应用程序内结算第2版 2012年5月

- 新增订阅支持。
 - 新增一个字符串值“2”，作为传给**sendBillingRequest()**的Bundle对象的**API_VERSION**键。
 - 新增一个JSON字段，**purchaseToken**，**PURCHASE_STATE_CHANGED** intent返回给**orders**列表。
 - 新增一个**purchaseState**值，3（已过期）**PURCHASE_STATE_CHANGED** intent返回给**orders**列表。该值表示的订阅已过期，不再有效。
 - 需要Google Play（Play store）3.5或更高版本。

在应用程序内结算第1版

2011年3月

- 最初的版本。
- 需要Google Play/Android Market 2.3.4或更高版本。

来自“[index.php?title=In-app_Billing_Reference&oldid=11382](#)”

1个分类: [Android Dev Guide](#)



Application Licensing

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/google/play/licensing/index.html>

编辑者： eoe耗子

更新时间： 2012.08.09

Application Licensing

Google Play提供了一个许可证服务，让用户可以为发布的应用程序执行许可证政策。有了Google Play的许可证，应用程序可以在运行的时候访问Google Play，获取当前用户的许可证状态，然后根据情况决定是否可以使用更多的功能。

使用该服务，可以在应用程序之间应用一个灵活的许可证政策——每个应用程序都可以用最合适的方式执行许可证。必要时，应用程序可以在Google Play中获取的许可证状态的基础上自定义常量。例如，应用程序可以检查许可证状态，然后运用自定义的常量允许用户在特定的时间段内免许可证运行。除此之 外，应用程序还可以限制在某些设备上运行。

许可证服务时控制访问应用程序的保险方式。应用程序检查许可证状态时，Google Play服务器使用一对与发布者帐号关联的唯一密钥签署许可证状态反馈。应用程序在编译的.apk文件中存储该公开密钥，并使用该密钥验证许可证状态反馈。

通过Google Play发布的任意程序都可以使用Google Play的许可证服务，不需要特殊的帐号或者注册。此外，该服务没有使用专门的框架API，意味着用户可以为API level 3以上的任意应用程序添加许可证。

注意: Google Play许可证服务最开始是用于付费应用程序希望验证当前用户是否真的付费了。然而,任何程序(包括免费的应用程序)都可能使用许可证服务下载APK扩展文件。在这种情况下,应用程序发送给许可证服务器的请求,就不是为了验证用户有没有付费,而是验证请求的扩展文件的URL。更多关于下载应用程序的扩展文件的信息,请参考[APK扩展文件指南](#)。

想要了解更多Google Play应用程序的许可证服务,为应用程序添加集成,请参阅下面的文档:

[许可证概览](#)

描述了服务的原理,以及典型的许可证实现

[安装许可证](#)

解释了如何建立Google Play帐号、开发环境和测试环境,用于为应用程序添加许可证。

[为应用程序添加许可证](#)

为向应用程序添加许可证提供了一步步的指导。

[许可证参考](#)

提供了关于许可证库的类和服务反馈码的具体信息。

来自“[index.php?title=Application_Licensing&oldid=9013](#)”



Licensing Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： GloriousOnion

原文链

接：

<http://developer.android.com/guide/google/play/licensing/overview.html>

目录

[[隐藏](#)]

1 许可机制概述

- [1.1 快速预览](#)
- [1.2 在此篇文档中](#)
- [1.3 认证响应是安全可靠的](#)
- [1.4 认证验证库](#)
- [1.5 要求与限制](#)
- [1.6 传统版权保护机制的替代者](#)

许可机制概述

Google Play许可认证属于网络服务，它允许APP通过查询某个可信的Google Play许可服务器来确定该App是否已许可指定用户使用。如果该应用的登记购买记录中存有该用户，则Google Play将许可该用户使

快速预览

- 认证机制用于验证应用程序是否是从Google Play付费购买的
- 应用程序可以控制如何进行证

用。

当向Google Play客户端应用的服务发出许可查询的请求时，验证过程便开始了。Google Play应用随后向许可服务器发出查询请求，并接受服务器返回的查询结果。Google Play应用将结果传给你的应用，根据返回的结果，你可以判断该用户是否可以继续使用你的应用。

注意：如果付费应用以“草稿应用（应用程序未发布）”的形式提交，许可服务器会认为所有用户都是该应用的许可用户（因为此应用连购买都不能实现）。这种例外在测试应用的认证功能实现时必不可少。

为了能够正确识别用户并确定验证状态，认证服务器需要用户接受用户和应用的信息。应用与Google Play客户端通过协作生成必要信息，Google Play客户端会将这些信息发送给认证服务器。为使应用更易添加认证功能，Android SDK提供了可下载并引入你的工程中的库文件，即Android Market Licensing包。可在应用中添加许可验证库（LVL），

书验证

- Google Play上的所有开发者都可免费使用这项服务

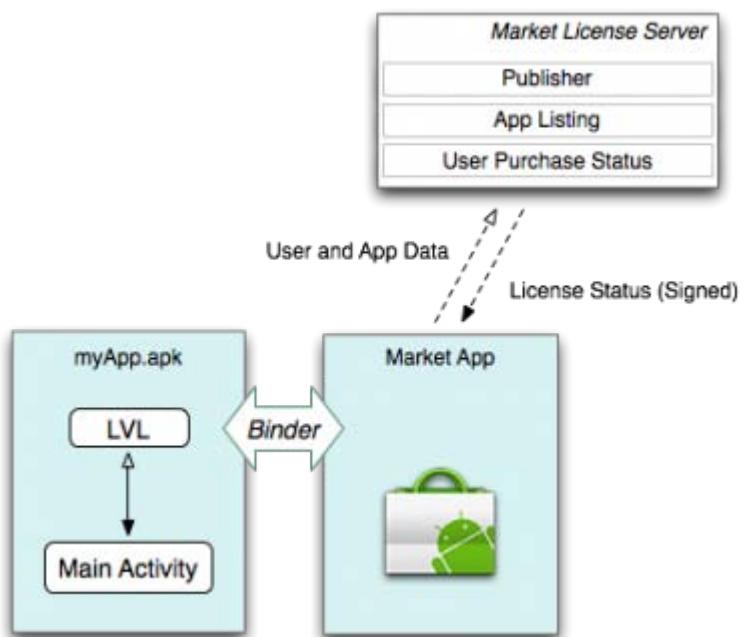
在此篇文档中

[认证响应是安全可靠的](#)

[认证验证库](#)

[要求与限制](#)

[传统版权保护机制的替代者](#)





它可以处理与Google Play许可服务器的所有交互。当加入LVL后，只需调用该对象的处理方法，便可轻而易举地确定当前用户的许可状态。

图1. 应用程序会通过认证验证库 (LVL)

和Google Play客户端程序生成证书认证对象，

用一个函数并实现对状态响应的回调，便可轻而易举地确定当前用户的许

您所开发的应用并不直接向许可服务器发送查询请求，而是通过调用Google Play客户端所提供的远程进程间通信 (remote IPC) 来生成许可查询请求。该许可查询请求包括：

- 程序应提供的：包名、场景（用于验证服务器响应）和回调函数（用于异步返回响应）；
- Google Play客户端收集的设备和用户的必要信息，例如：设备主要使用的Google账号用户名、IMSI及其他。随后，Google Play客户端将会把许可查询请求发送至认证服务器。
- Google Play利用已有的可用信息来评估该查询请求，并试图构建该用户的可信身份及购买记录，通过与购买记录核对比较，将会返回许可查询响应。Google Play使用IPC（跨进程通信）的回调函数将该响应结果传给你的应用程序。

开发者可以设置应用检测证书的时间、频率等，除此之外，对处理响应、验证签名相应信息、强制进行访问控制也都是完全可控的。

请注意，在证书验证过程中，应用不需处理网络连接，也不需使用Android平台所提供的证书验证相关API。

认证响应是安全可靠的

为保证每一次认证查询的完整性，认证响应会使用Google Play服务器与应用之间的专用密钥对进行签名。

认证服务器会为每一个发布者账号生成唯一的密钥对，并把公开密钥存放在个人账户信息页中，你需要把公开密钥从网站上复制到你的代码之中。服务器内部存放着相对应的私有密钥，私有密钥会用来为该账号的认证响应进行签名。当应用接收到结果签名的认证响应后，会先使用已有的公开密钥进行数据验证。该方法可以检测出被篡改或伪造的认证响应。

认证验证库

Android SDK提供可下载的Google Market Licensing包，该认证包中包含证书验证库（LVL）。LVL大大简化了向应用添加认证功能的过程，并可保证功能实现得安全稳定。LVL所提供的内部类可以处理绝大多数标准的认证请求操作，例如通知Google Play客户端生成认证查询请求，以及验证核实认证查询响应。它的外部接口使得开发者可以很容易地在应用中为特定代码制定认证策略，并可按需控制使用权。LVL的关键接口如下：

Policy

根据接收到的认证查询响应或是其他有用信息（例如与应用所关联的后端服务器），开发者的具体实现可控制特定用户的程序使用权。如果需要，该实现能衡量认证查询响应中多个字段并可进行其他限制。

另外，该实现也可处理产生错误结果的认证查询，如网络错误。

LicenseCheckerCallback

根绝Policy对象对认证查询响应的处理结果，可以实现对应用使用权的控制。开发者可以按照需要来完成具体实现，包括在界面显示认证结果，导向应用购买页面等（如果认证不成功）。

为方便开发者熟悉使用Policy，LVL提供了Policy对象的两个完整实现，不加修改或略加修改来满足特定需求就可以很容易地使用它们：

ServerManagedPolicy

可以灵活使用的Policy对象。当用户设备离线时（如用户正在乘坐飞机），它会通过使用认证服务器提供的设置来处理缓存的响应和对应用的使用权。由于对绝大多数程序适用，所以非常推荐使用ServerManagedPolicy。

StrictPolicy

限制性很强的Policy对象，它不会缓存任何认证响应信息，同时，只有当服务器返回认证响应才能够使用应用。

LVL是可下载的Android SDK程序包。包中除了LVL，还有一个示例程序，该示例程序展示了类库应如何与应用程序相结合，以及应用程序应如何处理响应数据、界面交互和错误情况等。 LVL源代码是以Android类库工程的实行提供的，这意味着开发者可以维护一份库代码并在多个应用中共用。Android SDK提供提供完整的测试环境，所以即使您没有一部实体机，也可在发布前通过Android SDK来测试应用的认证功能。

要求与限制

Google Play认证机制是为开发者设计的，开发者使用认证机制可以对发布到Google Play的应用程序进行认证控制。这项服务不能用于那些不是通过Google Play发布的程序及没有Google Play客户端程序的设备。

以下几点是您在为应用实现认证功能时所应注意的：

- 只有安装有Google Play客户端应用或Android 1.5 (API Level 3) 及以上的设备才可以使用该服务；
- 为完成证书检测，与认证服务器的网络连接是必不可少的。当网络连接不可用时，开发者可以通过实现缓存证书的不同动作来控制对应用的使用；
- 应用的认证控制安全性完全基于开发者的具体实现。认证服务本身提供给开发者安全检测证书的手段，但是具体实施和对证书的处理都是由开发者来决定的；
- 对于不提供Google Play的设备而言，具有证书认证功能并不会影响程序的其他功能；
- 如果开发者使用的服务提供[APK扩展文件](#)，就可以为免费软件实现认证控制。

传统版权保护机制的替代者

Google Play认证是一套灵活、安全的程序使用控制机制。它有效的取代了Google Play之前提供的版本保护机制，并且使应用程序有了更广阔的市场潜力。

- Google Play传统的版权保护机制限制之一：应用程序只能安装在提供内部安全存储的设备上。例如，获得Root权限的设备不能下载受版权保护的应用，同时，这些应用也不可以安装在SD卡上；

使用**Google Play**认证机制能够使应用的认证模型转变为证书认证，这种认证模型与开发者定义的认证策略和在**Google Play**上的发布者有关，而不受设备特性影响。基于上述原因，应用可以安装在包括SD卡在内的各种兼容存储设备上，于此同时，开发者也可对应用进行相应的控制。

完全杜绝应用的非授权使用并不现实，但证书认证服务使得开发者可以处理绝大多数需求，同时，兼容各种有锁无锁运行于**Android 1.5**及以上版本的设备。关于如何在应用中添加证书认证功能，请详见[添加认证功能](#)。

来自“[index.php?title=Licensing_Overview&oldid=11184](#)”



Setting Up for Licensing

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文链

接：<http://developer.android.com/guide/google/play/licensing/setting-up.html>

编辑者： eoe耗子

更新时间： 2012.08.06

为应用程序添加许可证之前，需要建立Google Play的发布帐号，开发环境以及用来验证安装的测试帐号。

目录

[[隐藏](#)]

[1 建立发布者帐户](#)

- [1.1 许可证的管理设置](#)

[2 建立开发环境](#)

- [2.1 建立运行时环境](#)
- [2.2 下载LVL](#)
- [2.3 建立许可证认证库](#)
- [2.4 为应用程序添加库工程资源](#)

[3 建立测试环境](#)

- [3.1 设置许可证认证的静态反馈](#)
- [3.2 建立测试帐号](#)
- [3.3 在运行时环境中使用授权帐号登陆](#)

建立发布者帐户

如果您还没有Google Play的发布者帐户，那么就需要使用Google帐户去注

册一个，并同意Google Play发布者网站的服务条款：

<http://play.google.com/apps/publish>

更多信息，请参考[开始发布](#)

如果您已经有Google Play的发布者帐户，那么就可以使用现有的帐户来建立许可证。

拥有了Google Play的发布者帐号，您就可以：

- 获取许可证的公开密钥
- 在发布应用程序之前，测试应用程序是否完成了许可证的安装
- 发布添加了许可证支持的应用程序

许可证的管理设置

在发布者网站上，用户可以管理多个Google Play的许可证。管理操作可以在Licensing面板、Edit Profile页上进行，如下图1所示。该管理功能可以实现：

- 建立多个测试帐号，每个帐号由不同的邮件地址注册。许可证服务器允许用户在模拟器或真机上登录测试帐号，发送许可证认证，接受静态测试反馈。
- 获得对应帐户许可证的公开密钥。当用户需要在应用程序中安装许可证的时候，就必须将该公开密钥拷到应用程序中。
- 用户使用发布者帐号或测试帐号登录，上传应用程序到发布者帐户，当服务器接收到该应用程序的许可证认证请求时，用户可以配置服务器发送的静态测试反馈信息。

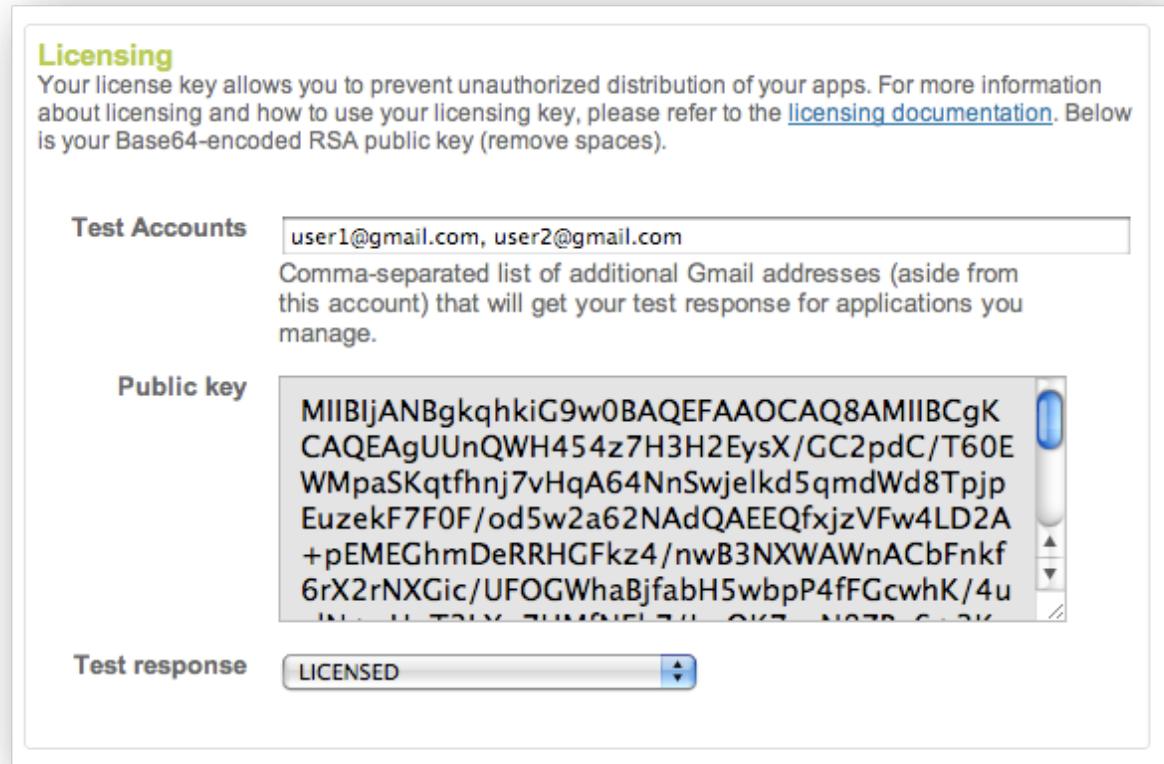


图1. Licensing面板、Edit Profile页中用户的许可证管理设置
更多关于如何使用测试帐号，以及静态测试反馈的信息，请参考下面的[建立测试环境](#)

建立开发环境

为许可证建立开发环境包括以下几个操作：

1. 为开发[建立运行时环境](#)
2. [下载LVL](#)到SDK
3. [建立许可证验证库](#)
4. [在应用程序中导入LVL库工程](#)

下面具体介绍了这些操作，当建立完成之后，就可以[将许可证添加到应用程序](#)

首先，需要建立一个合适的运行时环境，可以运行、调试、测试应用程序许可证认证和执行。

建立运行时环境

正如前文所述，应用程序许可证认证不是直接与许可证服务器连接，而是通过绑定Google Play程序提供的服务，然后初始化一个许可证认证的请求。然后，Google Play服务就会与许可证服务器进行直接的交互，最终将结果反馈到用户的应用程序中。要调试、测试应用程序的许可证，需要建立一个运行时环境，包括必要的 Google Play服务，这样应用程序才能给许可证服务器发送认证请求。

可用的运行时环境有两种：

- 安装Google Play应用程序的android设备
- 运行Google API附加组件，API level 8（第二版）或更高的android模拟器

在真机中运行

使用android真机测试许可证，需要设备满足以下条件：

- 运行android1.5以上（API level 3以上）的兼容版本
- 运行安装Google Play客户端程序的系统映像

如果系统映像中没有安装Google Play，那么应用程序将无法与许可证服务器交互。

关于如何建立开发android应用程序的设备的通用信息，请参考[使用硬件设备](#)

在模拟器中运行

如果用户没有android的真机，可以使用模拟器进行测试。

因为androidSDK提供的平台不包含Google Play，所以用户需要从SDK库中下载Google API附加组件平台（API level 8 以上）。下载完成之后，需要使用该系统映像创建一个虚拟机。

Google API附加组件不包含完整的Google Play客户端。但是，包含了以下内容：

- Google Play后台服务，可以实现`ILicensingService`远程接口，从而允许应用程序通过网络向许可证服务器发送许可证认证请求。
- 一组基本的帐户服务，使用户可以在虚拟机中添加Google帐户，并使用发布者帐号或测试帐号登陆。

发布者帐号或测试帐号登陆，可以让用户在不发布程序的情况下进行测试。更多信息，请参考下面的[授权帐号登陆](#)

通过SDK管理器可以或许多个版本的Google API附加组件，但是只有android2.2以上的版本包含必要的Google Play服务。

要建立一个测试添加许可证程序的模拟器，遵守以下步骤：

1. 运行SDK管理器（在eclipse的**Window**菜单中或执行`<sdk>/tools/android sdk`）
2. 选择并下载需要的**Google API**（必须是android2.2以上）
3. 下载完成之后，打开虚拟机管理器（在eclipse的**Window**菜单中或执行`<sdk>/tools/android avd`）
4. 点击**New**并设置新虚拟机的具体参数。
5. 在弹出的对话框中，给虚拟机取一个有描述意义的名字，在目标菜单中选择对应的**Google API**，作为运行新虚拟机的系统映像。根据需要配置其它参数，然后点击 **Create AVD**，完成虚拟机的创建。SDK工具创建了新的虚拟机配置后，将会在可用的**android**虚拟机列表中显示出来。

如果您对**android**虚拟机及其使用还不熟悉，请参考[管理虚拟机](#)

更新项目配置

建立了上述要求的运行时环境以后（真机或模拟器），确保更新过应用程序工程或按需要创建了脚本，这样编译后的包含许可证的.**.apk**文件才算配置到环境中了。尤其如果在eclipse中配置，确保为运行/调试配置了合适的设备或模拟器。

如果工程是在**android 1.5(API level 3)**以上编译的，那么应用程序的编译配置不需要改变。例如：

- 如果应用程序时在**android 1.5**以上编译的，那么不需要对编译配置进行任何的修改，就可以支持许可证。编译目标已经达到了许可证的最低要求，因此，用户可以继续使用该版本的**android**平台。
- 类似地，如果是在**android 1.5 (API level 3)** 的模拟器上运行**Google API**附加组件 **API 8**，将其作为应用程序的运行时环境，就不需要修改应用程序的编译配置了。

总的来说，为应用程序添加许可证应该对应用程序的编译配置没有什么影响。

下载LVL

许可证验证库（LVL）是一组帮助类的集合，在为应用程序添加许可证的时候，能够大大简化工作。在任何情况下，下载LVL，将其作为安装许可证的基础都是推荐的一种方式。

LVL包可以在android SDK中下载。该包中包括：

- 存储在android库工程内部的LVL资源
- 一个基于LVL库工程的、名为“sample”的实例程序。这个实例介绍了应用程序如何使用库中的帮助类对许可证进行认证和执行。

要下载LVL包到开发环境中，可以使用SDK管理器。运行android SDK管理器，选择**Google Market Licensing**包，如下图2所示。同意服务条款，点击 **Install Selected** 开始下载。

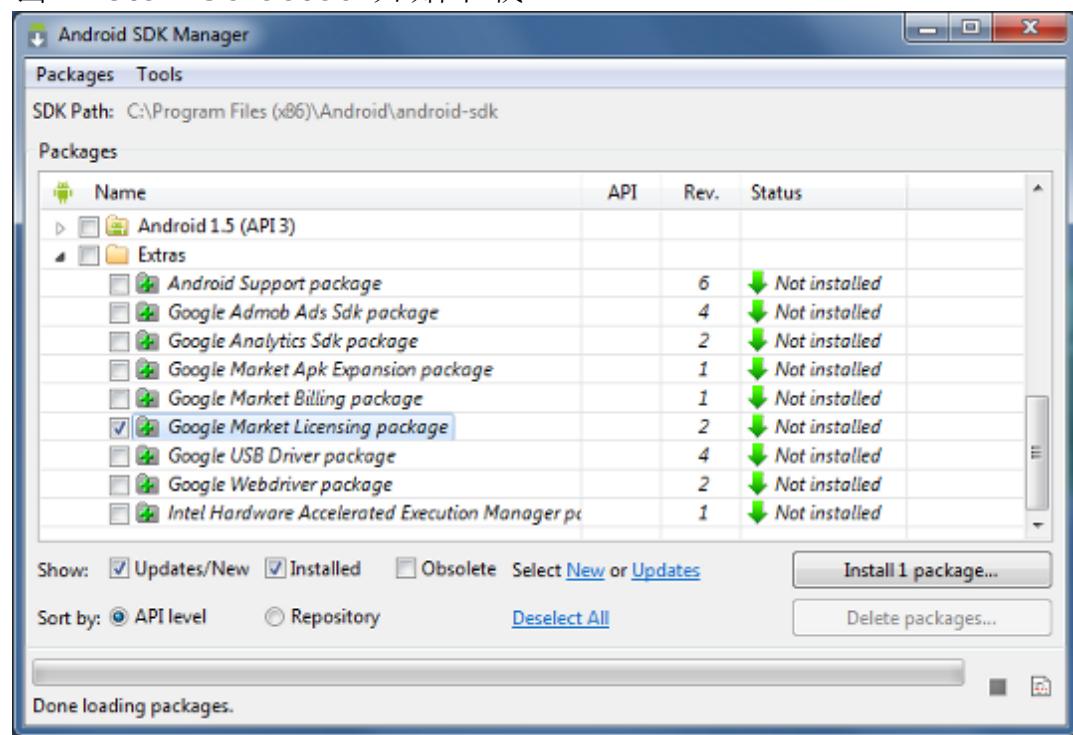


图2 包含LVL及其实例的许可证包

下载完成之后，android SDK管理器将会把LVL库工程和实例程序安装在以下目录：

<sdk>/extras/google/market_licensing/library/ LVL库工程
 <sdk>/extras/google/market_licensing/sample/ 实例程序

如果您仍然不熟悉如何下载包到SDK，请参考文档[配置SDK](#)

建立许可证认证库

下载了LVL之后，需要将其安装到开发环境中，可以使用android库工程，或者直接拷贝（导入）库资源到已有的程序包中。总的来说，将LVL作为库工程来使用时比较推荐的一种方式，因为这样可以在不同的应用程序之间重复使用许可证代码，随着时间的推移，这样维护起来也方便。不过，要注意的是 LVL不是用来单独编译，或作为一个静态的jar包添加到应用程序中的。

移动库资源的位置

有时用户需要在一定程度上自定义LVL资源，因此需要将库资源（整个`<sdk>/market_licensing/library`目录）移动或复制到SDK以外的工作目录中。然后使用新位置中的资源作为工作集。如果用户正在使用源代码管理系统，添加、追踪工作目录下的资源，而不是SDK默认目录下的。

移动库资源很重要，因为将来更新许可证包的时候，SDK会将新文件安装在与原文件相同的位置上。移动工作库文件是一种保险做法，确保用户的工作不会因为下载新版本的LVL被无意中覆盖掉。

创建LVL的库工程

使用LVL的推荐方式是新建一个android库工程。库工程是一种包含android源代码和资源的开发工程。其他android应用程序工程可以引用该库工程，而且，在编译的时候，会将编译的源代码放在.apk文件中。放到授权这个例子中，就意味着只需要在库工程中一次性实现，

然后再不同的应用程序工程中包含库的源代码即可。使用这种方式，可以很方便地维护、并集中管理多个工程的许可证的统一实现。

LVL是一个已经配置好的库工程——只要下载下来了，就直接可以使用了。如果使用安装ADT的eclipse，需要将LVL作为新的开发工程添加到工作空间，就像添加新的应用程序工程一样。

1. 使用新工程引导从已有资源创建新工程。选择LVL的库工程目录（该目录包含库的AndroidManifest.xml文件），作为工程的根目录。
 2. 在创建库工程时，需要的话，可以选择应用程序的名字、包，以及其他属性。
 3. 库的编译目标需要是android 1.5 (API level 3) 以上。
- 创建完成之后，在`project.properties`文件中已经定义工程为库工程，因此不需要再进行额外的设置了。

库工程

LVL是android的库工程，意味着可以在多个应用程序中使用LVL的代码和资源。

如果您对库工程及其使用还不熟悉，请参考[管理工程](#)

更多关于如何创建应用程序工程或在eclipse中使用库工程的信息，请参考[在装有ADT的eclipse中管理应用程序](#)

复制LVL资源到应用程序

除了添加LVL作为库工程之外，还可以直接将库资源拷贝到应用程序中。要实现该功能，直接拷贝（或导入）LVL的library/src/com目录到应用程序的src/目录即可。如果用户使用的是直接将库资源添加到引用程序中的方法，则可以跳过下一部分，开始[为应用程序添加许可证](#)中描述的库的研究。

为应用程序添加库工程资源

如果要将LVL资源当作库工程来使用，就需要在应用程序的工程属性中添加LVL库工程的引用。该引用使编译工具在编译的时候将LVL库工程资源添加到应用程序中。这个添加引用的过程依赖于开发环境。具体如下文所述。使用安装ADT的eclipse开发，应该像上一部分描述的一样，已经添加库工程到工作空间了。如果还没有添加，在继续之前先添加。

然后，打开应用程序的工程属性窗口，如下图所示。选择“Android”属性组，点击添加，然后选择LVL库工程(com_android_vending_licensing)，点击确认。更多信息，请参考[管理安装ADT的eclipse工程](#)

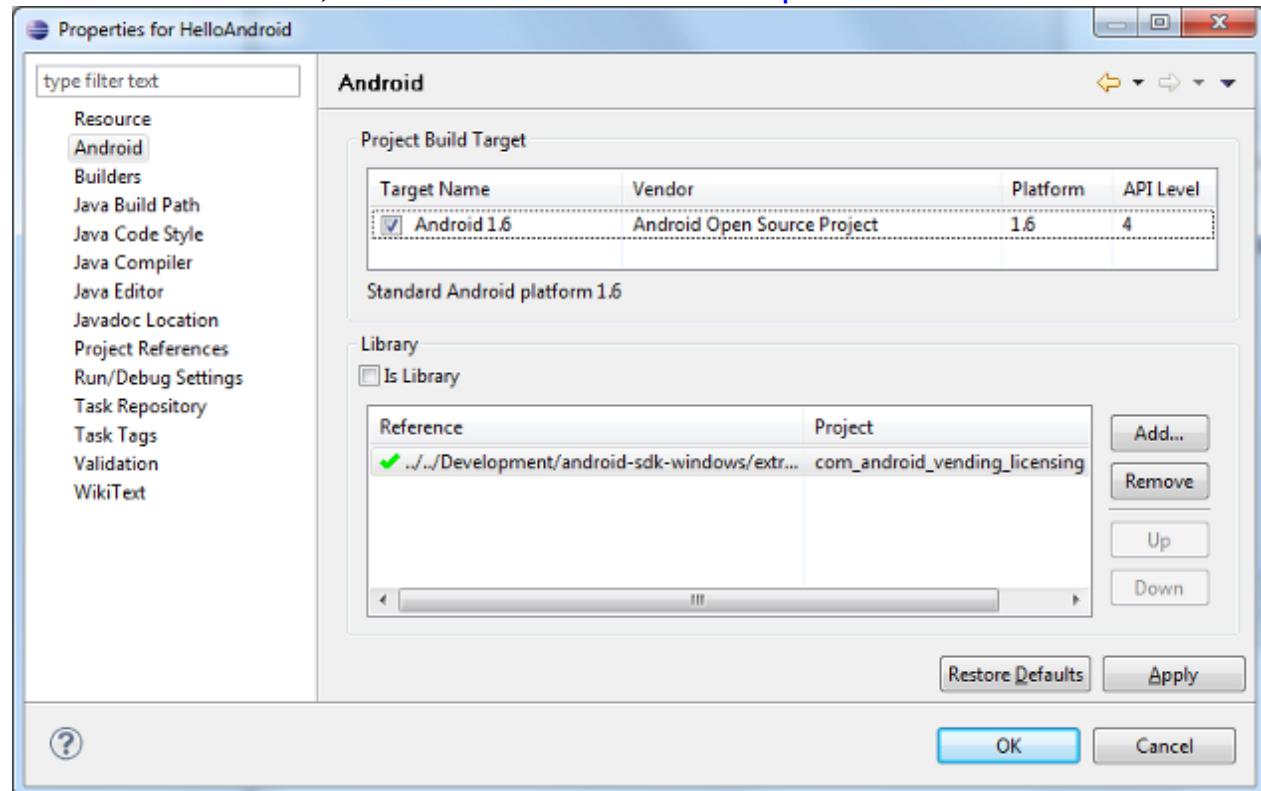


图3. 使用安装ADT的eclipse开发，可以通过应用程序的工程属性添加LVL库工程到应用程序

如果使用SDK命令行工具开发，导航至包含应用程序工程的文件夹，打开`project.properties`文件。在文件中添加一行，指明键`android.library.reference.<n>`，以及库的路径。例如：

```
android.library.reference.1=path/to/library_project
```

或者，可以使用该命令更新工程属性，添加库工程的引用：

```
android update lib-project  
--target <target_ID> \  
--path path/to/my/app_project \  
--library path/to/my/library_project
```

更多如何使用库工程的信息，请参考[建立库工程](#)

建立测试环境

Google Play发布者网站提供应用程序在发布之前测试许可证的配置工具。如果用户正在安装许可证，那么可以利用发布者网站的工具来测试应用程序政策，处理不同的许可证反馈信息和错误情况。

许可证测试环境的主要组件有：

- 在对应的帐号下配置测试反馈，这样用户使用发布者帐号或测试帐号登录后，上传应用程序到发布者帐号，请求服务器进行许可证认证时，就可以收到许可证反馈信息。
- 一组可选的测试帐号，用于当上传的应用程序的进行许可证认证时，接收静态测试反馈（不管应用程序有没有发布）。
- 用户使用发布者帐号或某一个测试帐号登陆之后，会有一个包含Google Play程序或Google API附加组件的应用程序的运行时环境。

正确地建立测试环境的步骤：

1. [设置许可证服务器返回的静态测试反馈](#)
 2. 需要的话[建立测试帐号](#)
 3. [登陆](#)到合适的模拟器或真机，激活许可证认证测试
- 下面给出详细信息。

设置许可证认证的静态反馈

Google Play提供发布者帐户的配置设置，使用户可以重写许可证认证过程，返回一个特定的静态反馈码。这个设置只能够用来测试，而且只应用于使用发布者帐号或注册的测试帐号登陆到模拟器或真机上的用户上传的应用程序的许可证认证。对于其他用户，服务器会根据一般规则进行许可证认证。

要在自己的帐号中设置测试反馈，先使用发布者帐号登陆，点击"Edit Profile"。在该页面，在Licensing面板中找到Test Response菜单，如下图所示。用户可以从全部有效的服务器反馈码中选择合适的，来控制要测试的应用程序的反馈和条件。

总的来说，用户应当确保使用Test Response菜单中每一个可用的反馈码来进行应用程序许可证的测试。关于这些反馈码的信息，请参考[许可证引用中的服务器反馈码](#)。

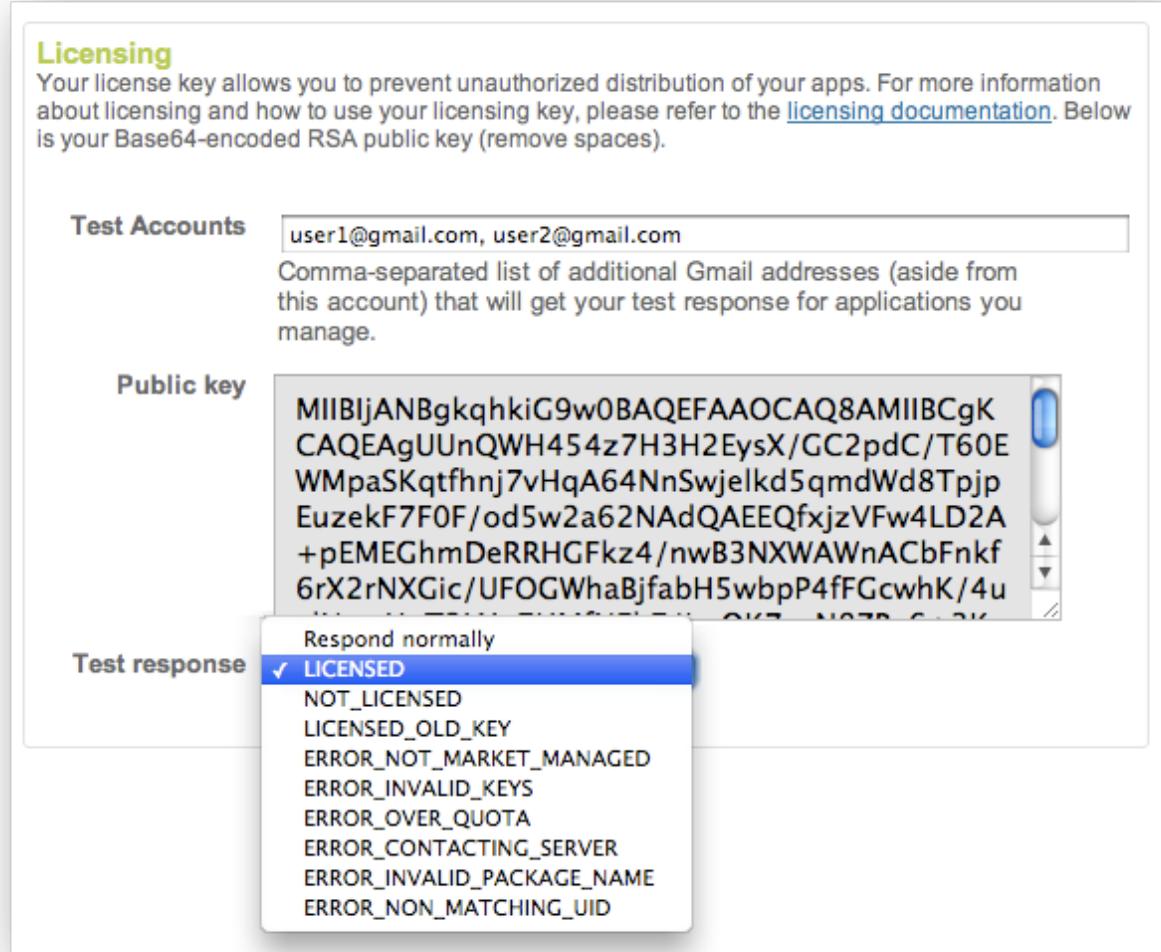


图4. Licensing面板中的Edit Profile页面，展现了测试帐号字段和测试反馈菜单

需要注意的是，设置的测试反馈适用于帐户范围内——即不仅仅适用于单一程序，而是所有跟该发布者帐户有关的应用程序。如果一次测试多个程序，改变测试反馈将会影响接下来的所有许可证认证（假设用户使用发布者帐号或测试帐号登陆）。

成功接收到测试反馈之前，必须登陆安装程序、并正在请求服务器的模拟器或真机。尤其，必须使用已经建好的发布者帐号或某一个测试帐号登陆。更多关于测试帐号的信息，请参考下一部分。

参考[服务器反馈码](#)，查看可用的测试反馈码，以及各自表示的意义。

建立测试帐号

有些时候，用户可能希望让多个开发小组测试最终将要通过发布者帐号发布的应用程序上的许可证，但是，确不想给他们登陆该帐号的权限。为了达到这个需求，Google Play发布者网站允许用户建立一个或多个可选的测试帐号，这个测试帐号可以通过该发布者帐号请求许可证服务器、接收静态测试反馈。

测试帐号是用户注册在发布者帐号上的标准Google帐号。这样的帐号可以接收上传的应用程序的测试反馈。然后，开发者可以使用这些帐号登陆到模拟器或真机，激活已安装程序的许可证认证。服务器接收到测试帐户的用户发出的许可证认证请求，将会返回发布者帐号中设置的静态测试反馈码。

当然，测试帐号在权限上有一定的限制，包括：

- 使用测试帐号的用户只能为已经上传到开发者帐号的应用程序请求许可证服务器
- 使用测试帐号的用户不能上传应用程序到开发者帐号
- 使用测试帐号的用户不能设置开发者帐号的静态测试反馈

下面的表格总结了开发者帐号、测试帐号和其他帐号在功能上的区别。

表1. 测试许可证的不同帐号之间的区别

Account Type	Can check license before upload?	Can receive test response?	Can set test response?
Publisher account	Yes	Yes	Yes

Test account	No	Yes	No
Other	No	No	No

注册发布者帐号的测试帐号

首先，需要注册发布者帐号的每一个测试帐号。如图4中所示，在Licensing面板的Edit Profile页面注册测试帐号。至需要输入以逗号隔开的帐号列表，点击保存即可。

用户可以使用任意Google帐号作为测试帐号。如果希望拥有并控制该测试帐号，可以自己建立一个帐号，然后将权限发放给开发人员和测试人员。

上传应用程序、分配测试帐号

如上所述，测试帐号只能接收上传到发布者帐号的应用程序的静态测试反馈。考虑到这些用户没有权限上传应用程序，作为发布者，应该和这些用户合作，收集要上传的应用，分配应用进行测试。关于收集和分配，任何方便的方法都可以使用。

一旦应用程序上传了，在许可证服务器上可以看见该应用程序，开发人员和测试人员就可以不用上传新版本，也能在本地开发环境中继续修改程序。只有在本地应用程序修改了manifest文件中的versionCode属性时才需要重新上传。

为测试帐号分配公开密钥

许可证认证服务器以一般的方法处理静态测试反馈，包括签署许可证反馈数据、添加额外的参数等的功能。要支持开发人员使用测试帐号完成许可证的测试，而不是使用开发者帐号，就需要为他们分配公开密钥。没有权限登陆到开发者网站美酒没有权限获取公开密钥，没有公开密钥就不能验证许可证反馈。

需要注意的是，如果用户决定为自己的帐号生成一个新的许可证密钥，那么必须要通知所有测试帐号的用户。对测试人员来说，可以将新的密钥安装到应用程序包中，一起分配给用户。对于开发人员来说，需要直接将密钥给他们。

在运行时环境中使用授权帐号登陆

许可证服务用于决定特定的用户能否使用特定的应用程序——在进行许可证

认证的时候，Google Play程序从系统的原始帐号中收集用户的信息，连同应用程序的包名和其他信息一起发送给服务器。但是，如果没有可用的用户信息，许可证认证将不会成功，Google Play程序就会结束请求，返回一个错误。

测试时，要确保应用程序能够成功的访问服务器，必须确保使用以下帐户登陆到真机或模拟器：

- 发布者帐户
- 发布者帐户注册的测试帐号

使用发布者帐号登陆的优势是，使应用程序在上传到发布者帐号之前，就可以接收静态测试反馈。

如果用户是某大型组织的成员，或者是要发布的应用程序的外部人员，那么使用测试帐号更合适些。

要登录到真机或模拟器，遵循以下步骤。推荐的方式是使用原始帐号登陆——然而，如果真机或模拟器中已经存在其他帐号，可以创建一个额外的帐号，然后再使用发布者帐号或测试帐号登陆。

1. 打开 *Settings > Accounts & sync*
2. 选择**Add Account**, 选择添加一个Google帐号
3. 选择**Next** 然后**Sign in**。
4. 输入发布者帐号或发布者帐号注册的测试帐号的用户名和密码。
5. 选择**Sign in**。系统就会使用新的帐号登陆了。

一旦登陆了之后，就可以开始应用程序的许可证认证了（假设已经完成了上述LVL集成）。当应用程序激活许可证认证时，将会收到一个包含在发布者帐号中设置的静态测试反馈的反馈信息。

注意，如果使用的是模拟器，每次重启模拟器、清除数据时都需要重新登陆。

完成了上述步骤之后，继续[为应用程序添加许可证](#)。

来自“[index.php?title=Setting_Up_for_Licensing&oldid=10601](#)”



使用Google帐号登陆模拟器

如果是在模拟器上进行许可证测试，需要使用一个Google帐号登陆。如果没有看到创建Google帐号的选项，那么有可能是该虚拟机是在标准的android系统映像上，而不是在 Google API附加组件，API level 8 (第二版) 以上的系统映像上运行。

更多信息，请参考上面的[建立运行时环境](#)。

Adding Licensing to Your App

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： atearasan

原文链接：<http://developer.android.com/guide/google/play/licensing/adding-licensing.html>

目录

- [1 给你的应用添加签名](#)
 - [1.1 添加签名权限](#)
 - [1.2 实现Policy](#)
 - [1.2.1 自定义Policy向导](#)
 - [1.2.2 ServerManagedPolicy](#)
 - [1.2.3 StrictPolicy](#)
 - [1.3 实现Obfuscator](#)
 - [1.3.1 AESObfuscator](#)
 - [1.4 在Activity中检查签名](#)
 - [1.4.1 签名检查和响应的描述](#)
 - [1.4.2 导入](#)
 - [1.4.3 以内部类的方式实现LicenseCheckerCallback](#)
 - [1.4.4 创建一个从LicenseCheckerCallback 到UI的Handler线程](#)
 - [1.4.5 初始化LicenseChecker和LicenseCheckerCall -back](#)
 - [1.4.6 调用checkAccess\(\)初始化签名检查](#)
 - [1.4.7 为签名认证植入你的密钥](#)
 - [1.4.8 关闭IPC连接的时候调用LicenseChecker的onDestory\(\)方法](#)
 - [1.5 实现DeviceLimiter](#)
 - [1.6 混淆你的代码](#)
 - [1.7 发布签名过的应用](#)
 - [1.7.1 删除Copy Protection](#)
 - [1.8 获得支持](#)

给你的应用添加签名

当你建立一个开发者账号和开发环境之后(见设置签名), 你就可以在License Verification Library(LVL)给你的app添加签名了。

在LVL添加签名认证需要遵循下面这些步骤:

1. 在你的应用的manifest.xml里添加签名权限
2. 实现Policy —— 你可以选择LVL 中提供的完整的实现方案, 或者你自己创建一个
3. 实现Obfuscator, 如果你的Policy需要缓存任何签名认证结果
4. 在你应用的main Activity中添加代码进行签名检查
5. 实现DeviceLimiter(可选, 但是不推荐)

下面描述了这些步骤. 当你完成了这些操作, 你就应用可以成功编译你的应用了, 并按照设定的测试环境开始测试你的应用.

LVL包含了对全套源码完整的概述, 见Summary of LVL Classes and Interfaces

添加签名权限

为了使用Google Play发送签名验证请求到服务器, 你的应用本身必须有相应的权限: com.android.vending.CHECK_LICENSE. 如果你的应用没有进行签名认证权限的声明就尝试进行签名认证, LVL会抛出一个安全异常.

在你的应用里添加签名认证的权限, 需要在<manifest>下创建元素<uses-permission>, 如:

```
<uses-permission android:name="com.android.vending.CHECK_LICENSE">
```

下面是例子:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>
    <!-- Devices >= 3 have version of Google Play that supports licensing. -->
    <uses-sdk android:minSdkVersion="3" />
    <!-- Required permission to check licensing. -->
    <uses-permission android:name="com.android.vending.CHECK_LICENSE" />
    ...
</manifest>
```

注意：你不能在LVL库的项目的manifest中声明CHECK_LICENSE权限，因为SDK Tools不会把它合并到依赖的项目中。因此，你必须在每个项目的manifest中进行权限声明。

实现Policy

Google Play的签名服务本身不确定是否应该让提交的签名通过认证。相反，这个职责交给了你应用中提供的Policy。

Policy是LVL声明的接口，用来根据签名验证结果控制你的应用允许或禁用用户访问。使用LVL，你的应用必须提供Policy的实现。

Policy定了两个方法，allowAccess()、processServerResponse()，当LicenseChecker实例处理来自签名服务器的响应的时候会被调用。它还定义了一个叫做LicenseResponse的枚举类，在调用processServerResponse()的时候把指定的签名验证结果传进去。

- processServerResponse() 在决定是否授权之前让你对从签名验证服务器接收到的响应数据进行预处理。

一个典型的实现是可以从响应数据中提取一些或全部字段并存到本地进行持久化存储，比如通过SharedPreferences 存储，以确保可以跨应用访问，而且不会因为设备关机导致数据丢失。例如，Policy可以在持久性数据容器 中持有签名最后一次验证成功的时间戳、重试次数、有效期等类似的信息，而不是在应用每次启动的时候重新加载这些数据。

当在本地存储响应数据的时候，Policy必须确保数据进行了混淆处理(见“实现Obfuscator”)

- allowAccess()根据任何有效的签名验证响应数据(来自服务器响应或缓存)或其他特定应用程序信息决定是否授权给用户访问你的应用。例如，你实现的allowAccess()可以带入一些额外的条件，如usage或者从后台服务器检索到的数据。在所有的情况下，如果用户被授权使用应用，方法应该只返回true。如果因为网络或系统原因导致签名验证未完成，你可以指定一个重试次数并允许用户暂时访问直到下一次完成验证。

为了简化你给应用添加签名的过程并提供一个设置Policy的说明，LVL提供了两个完整的Policy实现，你可以直接使用而不需要任何修改：

- ServerManagedPolicy，使用服务器默认设置和缓存的响应结果来管理各种网络环境下的访问，
- StrictPolicy，不缓存任何响应结果并且只有服务器返回验证过的响应结果才能访问。

对于大多数应用来说，比较推荐ServerManagedPolicy。ServerManagedPolicy是LVL默认的

方案，并且已经包含在LVL的示例应用中.

自定义**Policy**向导

在你的签名实现中，你可以使用LVL提供的完整的**Policy**实现(**ServerManagedPolicy**或**StrictPolicy**)或者你可以自定义一个**Policy**. 对于任何自定义**Policy**, 这里有几个重要的设计要点需要理解并且在你的实现中使用.

签名验证服务器一般会限制请求次数以防止过度使用服务器资源而导致拒绝服务. 当一个应用超过了请求限制, 服务器会返回503响应码, 作为一个通用的服务器错误告诉你的应用. 这就意味着在限制被重置之前将没有可用的签名验证响应结果提供给用户, 这会无期限的影响应用用户.

如果你准备自定义一个**policy**, 我们推荐**Policy**:

1. 把大部分最近的成功的签名验证结果缓存(并混淆处理) 在本地进行持久化保存
2. 只要缓存的响应结果是有效的, 那么对所有的签名检查都返回缓存的响应结果, 而不是创建一个验证请求. 我们推荐根据服务器额外提供的**VT**设置响应结果的有效期. 更多信息见**Server Response Extras**
3. 如果尝试任何请求都返回错误, 使用一个指数回避这个情况. 记录**Google Play**客户端自动尝试失败的请求, 因为在大多数情况下你的**Policy**不需要再去尝试
4. 提供一个“宽限期限”允许用户在限定的时间或次数内使用你的应用, 在需要重新进行签名验证的时候. 当没有有效的签名校验结果时, 你可以设置一个范围很大的限制值控制在宽限期内的用户允许使用你的应用直到下一次签名检查完成.

依照上面列出的向导设计你的**Policy**是很关键, 因为它可以确保最好的用户体验, 甚至同时让你能在出错的情况下有效的控制你的应用.

记录任何服务器提供的可用的设置帮助你管理有效期和缓存, 重试宽限期等等. 使用服务器提供的设置非常简单并且高度推荐. 可以看看**ServerManagedPolicy**实现的例子. 有关于服务器的提供设置和信息可以见**Server Response Extras**

ServerManagedPolicy

ServerManagedPolicy是LVL提供的完整实现的并推荐使用的**Policy**接口. 这个实现已经作为默认的**Policy**跟LVL的类和服务整合在类库里.

ServerManagedPolicy提供了处理签名和重试响应的所有方法. 它会把所有的响应结果缓存

到本地**SharedPreferences** 文件, 并使用应用的**Obfuscator** 的实现类混淆. 这确保了签名的响应数据是安全的并且不会因为设备关机而丢失.

ServerManagedPolicy 提供了接口的具体实现方

法**processServerResponse()**和**allowAccess()**同时也包括一组支持管理签名声响应数据的方法和类型.

重要的是, **ServerManagedPolicy**的关键特性是使用服务器提供的设置为基础管理应用的签名授权, 而不受各种网络和异常条件的影响. 当一个应用连接到**Google Play**服务器进行签名声认证的时候, 服务器会把设置项作为**key-value**的形式存到某些响应类型的额外的字段中. 例如, 服务器提供了推荐应用签名使用的一些值, 尤其是是有效期、重试宽限期、最大重试次数. **ServerManagedPolicy** 把签名声响应数据中的值传入到**processServerResponse()** 并在**allowAccess()**中校验值. 有关于**ServerManagedPolicy**使用服务器提供的设置项, 见**Server Response Extras**.

为了方便、更好的性能和从**Google Play**服务器上取得更好的签名设置效果, 强烈推荐使用**ServerManagedPolicy**作为你的**Policy**.

如果你担心存在本地**SharedPreferences**中的签名声响应数据的安全, 你可以使用更强大的混淆算法或设计一个更严格的不存储签名数据的**Policy**. LVL包含了一个这样的例子, 更多信息见**StrictPolicy** .

使用**ServerManagedPolicy**, 只要将它导入到你的activity中, 创建一个实例, 当创建**LicenseChecker**的时候传递它的引用进去. 更多信息见**Instantiate LicenseChecker and LicenseCheckerCallback**

StrictPolicy

StrictPolicy是LVL提供的另一个可选的**Policy**的实现接口. 它提供的**Policy**比**ServerManagedPolicy**更为严格, 它不允许用户访问应用除非在访问的时候从服务器接收到标识用户可以访问的签名声响应数据.

StrictPolicy的主要特点是不会把任何签名声响应数据以持久化方式存在本地. 因为不存储数据, 重试请求不会被记录并且缓存的响应数据不会用于完成签名声检查. **Policy**允许访问只能在:

- 从服务器接收到签名声响应数据
- 并且响应数据标识用户验证通过可以访问应用

如果你所关心的是在所有情况下, 没有用户可以访问你的应用除非用户在使用的时候被确认认证过了, 那么使用**StrictPolicy**是比较合适的. 此外, **Policy**比**ServerManagedPolicy**稍微安全点, 自从不在本地缓存数据后, 恶意用户就没有办法篡改本地数据获取应用访问权限.

同时, **Policy**对于普通用户来说是一个挑战, 这意味着他们在没有可用网络的时候无法使用应用. 另一个副作用是自从你的应用不使用缓存后将会发送更多的签名验证请求到服务器.

总的来说, 这个策略代表了在某个程度上的用户是绝对安全并且可控的. 所以用**Policy**之前要仔细考虑下.

使用**StrictPolicy**, 只要将它导入到你的activity中, 创建一个实例, 当创建**LicenseChecker**的时候传递它的引用进去. 更多信息见**Instantiate LicenseChecker and LicenseCheckerCallback**.

实现**Obfuscator**

一个典型的**Policy**实现需要为一个应用保存签名校验数据到持久化存储器, 所以它可以跨应用调用并且不会因为设备关机而丢失数据. 例如, **Policy**可以在持久性数据容器中持有签名最后一次验证成功的时间戳、重试次数、有效期等类似的信息, 而不是在应用每次启动的时候重新加载这些数据. 默认的**Policy**实现在**LVL**的**ServerManagedPolicy**中, 它会把响应数据保存在**SharedPreferences**实例中, 以确保数据是持久化的.

因为**Policy**会使用保存的数据去判断允许还是不允许访问应用, 它必须保证任何数据是安全的并且不能重用或被拥有**root**权限的用户控制. 所以**Policy**必须要在存储之前混淆数据, 对每一个设备使用一个唯一的密钥. 在指定的应用和指定的设备中使用密钥是很关键的, 因为它可以防止在应用和设备上共享混淆过的数据.

LVL帮助应用以一种安全、持久化的方式保存签名校验数据. 首先, 它提供**Obfuscator** 接口让你的应用在存储数据的时候可以选择混淆算法. 在这个基础上, **LVL**提供了帮助类**PreferenceObfuscator**, 负责调用应用的**Obfuscator** 类, 并且读写**SharedPreferences** 实例中混淆的数据.

LVL提供了**Obfuscator** 的完整实现类——**AESObfuscator**, 使用**AES**加密技术混淆数据. 你可以直接在你的应用中使用**AESObfuscator**或者修改它以适应你的需求. 更多信息见一下章节.

AESObfuscator

AESObfuscator是**LVL**提供的并推荐的**Obfuscator** 的实现接口. 该实现已经集成在**LVL**的示例应用中并且在**lib**库中作为默认的**Obfuscator** .

AESObfuscator提供安全混淆实现是在从数据存储器中读写数据的时候通过**AES**加密或解密实现的. **Obfuscator** 种子加密要用到应用提供的三个数据字段:

1. salt – 一个随机的字节(byte)数组用于每一个(单位)的混淆
2. 应用的字符标识, 代表着应用的包名
3. 设备的字符标识, 尽可能来自指定的设备的源数据, 以便于唯一值

使用AESObfuscator, 首先要导入到你的activity. 定义一个私有静态的常量数组(private static final array)用来保存salt字节并初始化生成20个随机字节.

```
...
// Generate 20 random bytes, and put them here.
private static final byte[] SALT = new byte[] {
    -46, 65, 30, -128, -103, -57, 74, -64, 51, 88, -95,
    -45, 77, -117, -36, -113, -11, 32, -64, 89
};
...
```

接下来, 定义一个任何地方都可以使用的变量保存设备的标识符并指定一个值. 例如, 在LVL的例子中查询系统设置是使用android.Settings.Secure.ANDROID_ID, 这个值对于每一个设备来说都是唯一的.

注意, 根据你使用的API, 你的应用可能需要添加访问系统特定信息的权限. 例如, 查询TelephonyManager获取设备的IMEI码或相关数据, 你的应用就需要在manifest里添加权限android.permission.READ_PHONE_STATE

请求新权限唯一的目的就是为了获取特定设备的信息用于你的Obfuscator, 考虑做可能会影响你的应用或在Google Play上被过滤(因为有些权限可能引起SDK构建工具添加关联的<uses-feature>)

最后, 构建AESObfuscator实例, 传入参数salt、应用标识符、设备标识符. 你可以在构建Policy 和LicenseChecker 的时候直接创建. 例如:

```
...
// Construct the LicenseChecker with a Policy.
mChecker = new LicenseChecker(
    this, new ServerManagedPolicy(this,
        new AESObfuscator(SALT, getPackageName(), deviceId)),
    BASE64_PUBLIC_KEY // Your public licensing key.
);
...
```

完整的例子见LVL示例代码的MainActivity.

在**Activity**中检查签名

当你实现了Policy管理你应用的访问, 下一步就是增加签名认证到你的应用, 创建一个验证

请求到服务器如果需要基于响应结果管理你应用的访问. 所有增加签名检查和处理响应结果的工作都在你的**main Activity**里.

增加签名检查和处理响应结果, 你必须:

1. 导入
2. 以内部类的方式实现**LicenseCheckerCallback**
3. 创建一个从**LicenseCheckerCallback** 到UI线程的**Handler**
4. 实例化**LicenseChecker**和**LicenseCheckerCallback**
5. 调用**checkAccess()**初始化签名检查
6. 给签名认证植入你的公钥
7. 关闭IPC连接的时候调用**LicenseChecker**的**onDestory()**

下面描述了这些步骤.

签名检查和响应的描述

在大部分情况下, 你应该把签名检查的工作放在你应用的**main Activity**的**onCreate()**中. 这可以确保用户一加载你的应用马上就会进行签名检查. 有些时候, 你也可以把签名检查放在其他位置. 例如, 如果你的应用包含了多个可以通过其他Intent启动的**Activity**的模块, 你可以把签名检查放在这写**Activity**里面.

签名检查主要包含两个步骤: 一个启动签名检查的方法——在**LVL**里, 这是你实现的**LicenseCheckerCallback**接口. 这个接口定义了两个方法, **allow()** 和 **dontAllow()**, 它们都会被**lib**库根据签名检查的结果触发. 你可以根据你的需求实现任何逻辑, 允许或不允许用户访问你的应用. 注意, 这两个方法不确定是否允许访问——这个由你实现的**Policy**来决定. 当然, 这些方法简单的提供了应用的行为, 怎么允许和禁止(或处理应用的错误).

allow()和**dontAllow()**方法提供了它们响应结果的“理由”(**reason**), 这个是**Policy**中的一个值, **LICENSED**, **NOT_LICENSED**, or **RETRY**. 需要提出来的是, 你应该在**dontAllow()**中处理接收到**RETRY**的情况并且提供一个“重试”的按钮给用户, 这可能会因为服务在请求阶段不可用而发生.



图6 签名检查的流程图

上面的图表说明了签名检查发生的过程:

1. 在应用main Activity里实例化LicenseCheckerCallback和LicenseChecker等对象. 当构造LicenseChecker的时候, 需要传入Context、Policy的实现类、开发者账号的公钥作为签名检查的参数.
2. 然后在LicenseChecker对象里调用checkAccess(). 方法的实现调用Policy判断SharedPreferences里是否有有效的签名校验数据.
 - 如果有, checkAccess()调用allow()
 - 否则, LicenseChecker会初始化一个签名校验请求发送到签名校验服务器

注意:签名校验服务器总是会返回LICENSED 当你执行一个测试应用的签名校验时

3. 接收到响应数据的时候, LicenseChecker会创建一个LicenseValidator检查签名并提取响应数据中的字段, 然后把它们传进你的Policy做进一步的操作.
 - 如果签名有效, Policy会缓存响应结果到SharedPreferences并通过LicenseValidator等下调用LicenseCheckerCallback对象里的allow()
 - 如果签名无效, Policy会通知LicenseValidator调用LicenseCheckerCallback里的dontAllow()
4. 如果发生可恢复的本地或者服务器错误, 例如当前网络无法发送请求, LicenseChecker会传递RETRY为响应结果到Policy的processServerResponse(). 并且allow()和dontAllow()也会接收一个reason参数. allow()的reason值一般是Policy.LICENSED 或Policy.RETRY, dontAllow()的reason值一般是Policy.DONT_ALLOW 或Policy.RETRY或Policy.RETRY. 如果你想给用户显示一个响应值, 那么这些值是非常有用的, 例如提供一个"重试"的按钮当dontAllow()响应Policy.RETRY, 这可能是由于服务不可用
5. 如果应用出现错误, 例如应用尝试以一个无效的包名去检查签名时候, LicenseChecker传递一个错误的响应结果到LicenseCheckerCallback的applicationError()中

注意, 除了在启动的时候进行签名校验并处理结果, 你的应用也需要提供Policy的实现, 如果Policy保存响应结果(例如ServerManagedPolicy), 一个Obfuscator 的实现.

导入

首先, 打开应用的main Activity文件, 从LVL的包中导入LicenseChecker和LicenseCheckerCallback.

```
import com.android.vending.licensing.LicenseChecker; import  
file:///D:/guide/Adding_Licensing_to_Your_App[2015/9/23 19:20:59]
```

如果你需要用LVL提供的默认Policy实现ServerManagedPolicy, 也导入它, 同时导入AESObfuscator. 如果使用自定义的Policy和Obfuscator就不要了.

```
import com.android.vending.licensing.ServerManagedPolicy; import
com.android.vending.licensing.AESObfuscator;
```

以内部类的方式实现LicenseCheckerCallback

LicenseCheckerCallback是LVL提供的处理签名检查结果的接口. 使用LVL进行签名检查, 必须实现LicenseCheckerCallback和它的方法允许或者不允许访问应用.

签名的检查结果总是会调用LicenseCheckerCallback中的一个方法, 基于响应结果的验证, 服务器验证码的本身, 和任何你的Policy提供的额外的处理. 你的应用可以在任何需要的地方实现方法. 一般最好保持方法简单, 限制它们管理UI状态和应用的访问. 如果你想对签名响应结果进行进一步处理, 例如连接后端服务器或使用自定义的约束, 你应该考虑把你的代码合并到Policy, 而不是把它们放在LicenseCheckerCallback的方法里.

在大多数情况下, 你应该把LicenseCheckerCallback的实现作为一个内部类定义在你的main Activity中.

根据需要实现allow()和dontAllow(). 首先, 你可以在方法里只进行简单的结果处理, 例如在对话框里显示签名的检查结果. 这可以帮助你更快的运行你的应用而且有利于调试. 然后, 当你决定你想要其他的行为之后就可以添加更复杂处理.

这里是一些在dontAllow()里处理没有验证结果的建议:

- 如果reason是Policy.RETRY, 显示一个"重试"的对话框给用户, 包含一个按钮可以创建一个新的签名认证请求
- 显示"购买这个应用"的对话框, 包含一个按钮可以引导用户到Google Play上可以购买这个应用的详情页. 有关于设置这种链接的信息, 见 [Linking to Your Products](#)
- 显示一个Toast通知表明这个应用是有限的, 因为它没有认证通过

下面是LVL中实现LicenseCheckerCallback的例子, 在方法中使用对话框显示签名检查结果.

```
private class MyLicenseCheckerCallback implements LicenseCheckerCallback {
    public void allow(int reason) {
        if (isFinishing()) {
            // Don't update UI if Activity is finishing.
            return;
        }
        // Should allow user access.
        displayResult(getString(R.string.allow));
    }

    public void dontAllow(int reason) {
        if (isFinishing()) {
```

```
// Don't update UI if Activity is finishing.  
return;  
}  
displayResult(getString(R.string.dont_allow));  
  
if (reason == Policy.RETRY) {  
    // If the reason received from the policy is RETRY, it was probably  
    // due to a loss of connection with the service, so we should give  
  
    // user a chance to retry. So show a dialog to retry.  
    showDialog(DIALOG_RETRY);  
} else {  
    // Otherwise, the user is not licensed to use this app.  
    // Your response should always inform the user that the application  
    // is not licensed, but your behavior at that point can vary. You  
  
    // provide the user a limited access version of your app or you can  
    // take them to Google Play to purchase the app.  
    showDialog(DIALOG_GOTOMARKET);  
}
```

另外, 你应该实现`applicationError()`, LVL让你的应用处理无法重试的错误. 错误列表见Licensing Reference的Server Response Codes. 你可以在任何你需要的地方实现这个方法. 在大多数情况下, 这个方法应该记录错误并调用`dontAllow()`.

创建一个从 LicenseCheckerCallback 到 UI 的 Handler 线程

在签名检查的时候, LVL会传递一个请求到Google Play, 来处理和签名验证服务器的通信. LVL在异步IPC(使用Binder)上传递请求, 所以实际处理和网络通信不在你应用的同一个线程中. 同样的, 当Google Play接收到结果时, 它在IPC里调用回调方法, 在你应用进程的IPC线程池里轮流执行.

LicenseChecker通过Google Play管理你应用的IPC通信,包括调用发送请求和接收响应的回调函数的方法。LicenseChecker会跟踪打开的签名请求并管理它们的超时设定。

所以它可以适当的处理超时也可以处理传进来的响应，而不会影响你的UI线程，
LicenseChecker实例化的时候会产生一个后台线程。它会在线程里处理所有的签名检查结果，无论是是从服务器接收到的响应结果还是超时错误。LVL最后都会从后台线程调用你的LicenseCheckerCallback。

对你的应用来说，这意味着：

1. 在大多数情况下，你的LicenseCheckerCallback里的方法会被后台线程触发
 2. 这些方法不会修改状态或在UI线程里调用任何处理方法，除非你在UI线程里创建一个Handler并且把你的回调方法传进去。

如果你想让**LicenseCheckerCallback**里的方法更新UI线程,如下所示,在main Activity里

的onCreate()里创建Handler. 在这个例子里, LVL示例应用的LicenseCheckerCallback方法(见上文)调用displayResult()通过Handler的post()更新UI线程.

```
private Handler mHandler;

@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    mHandler = new Handler();
}
```

然后, 在LicenseCheckerCallback方法里, 你可以使用Handler方法传递Runnable或Message对象到Handler对象. 这里是LVL里的例子, 在UI线程里传递Runnable到Handler显示签名状态.

```
private void displayResult(final String result) {
    mHandler.post(new Runnable() {
        public void run() {
            mStatusText.setText(result);
            setProgressBarIndeterminateVisibility(false);
            mCheckLicenseButton.setEnabled(true);
        }
    });
}
```

初始化LicenseChecker和LicenseCheckerCall -back

在main Activity的onCreate()里, 创建LicenseChecker和LicenseCheckerCallback的私有实例对象. 你必须先实例化LicenseCheckerCallback, 因为你调用LicenseChecker的构造函数的时候需要传递LicenseCheckerCallback的引用进去.

当你实例化LicenseChecker的时候, 你需要传入这些参数:

- Context
- 用于签名检查的Policy实例引用. 在大多数情况下, 你可以使用LVL提供的默认Policy实例, ServerManagedPolicy.
- 用于认证的你公共账号密钥的字符

如果你使用ServerManagedPolicy, 你不需要直接访问类, 所以你可以像下面的例子里一样在LicenseChecker的构造函数里初始化它. 注意你在构造ServerManagedPolicy的时候需要传入一个Obfuscator实例的引用.

下面是在Activity的onCreate()里初始化LicenseChecker和LicenseCheckerCallback的例子.

```
public class MainActivity extends Activity {
    ...
}
```

```

private LicenseCheckerCallback mLicenseCheckerCallback;
private LicenseChecker mChecker;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    // Construct the LicenseCheckerCallback. The library calls this when
done.
    mLicenseCheckerCallback = new MyLicenseCheckerCallback();

    // Construct the LicenseChecker with a Policy.
    mChecker = new LicenseChecker(
        this, new ServerManagedPolicy(this,
            new AESObfuscator(SALT, getPackageName(), deviceId)),
        BASE64_PUBLIC_KEY // Your public licensing key.
    );
    ...
}
}

```

注意只有在本地的缓存里有有效的签名校验数据的时候, LicenseChecker才会在UI线程里调用LicenseCheckerCallback的回调方法. 如果发送了签名校验请求到服务器, 回调方法总会被调用, 甚至在网络错误的时候也会调用.

调用**checkAccess()**初始化签名校验

在main Activity里, 增加调用LicenseChecker实例的**checkAccess()**的方法. 在这个方法里, 传入LicenseCheckerCallback实例的引用作为参数. 如果你需要在调用处理任何特殊的UI效果或状态管理, 你可以在**checkAccess()**的封装方法里有效的处理. 下面是LVL里从**doCheck()**封装方法里调用**checkAccess()**的例子:

```

public class MainActivity extends Activity {
    ...
    private LicenseCheckerCallback mLicenseCheckerCallback;
    private LicenseChecker mChecker;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        // Construct the LicenseCheckerCallback. The library calls this when
done.
        mLicenseCheckerCallback = new MyLicenseCheckerCallback();

        // Construct the LicenseChecker with a Policy.
        mChecker = new LicenseChecker(
            this, new ServerManagedPolicy(this,
                new AESObfuscator(SALT, getPackageName(), deviceId)),
            BASE64_PUBLIC_KEY // Your public licensing key.
        );
        ...
    }
}

```

为签名校验植入你的密钥

对每一个开发者账号, Google Play是自动生成只用于签名检查的2048位的RSA公有/私有密钥. 密钥和开发者的账号是唯一关联的, 并且通过这个账号发布的应用都共享这一个密钥. 虽然密钥和开发者账号关联, 但是它和你登录应用(或它的衍生产品)所用的不一样.

Google Play 开发者网站公开用于签名的公共密钥授权给任何开发者, 但是它保持所有用户的私有密钥在一个安全的地方. 当一个应用为你账号发布的应用请求签名认证, 签名认证服务器会标识响应结果是使用的私有密钥. 当LVL接收到响应的时候, 它会使用应用提供的公共密钥去验证响应结果的签名.

要在应用中添加签名, 你必须获得你开发账号的公钥并复制到你的应用. 下面是怎么获取你账号的公钥:

1. 登录**Google Play**开发者网站. 确保你登录账号上的应用已发布(或将发布).
2. 在账号首页, 查找"编辑属性"(Edit profile)链接并点击
3. 在编辑属性(Edit profile)页面, 查找"签名"(Licensing)面板. 用于签名的公钥在文本框"公钥"(Public Key)里.

添加公钥到你的应用, 只要在文本框里复制到你的应用里, 赋值给你**String** 属性**BASE64_PUBLIC_KEY**. 当你复制的时候确保你选择了完整密钥, 不要遗漏了字符.

示例:

```
public class MainActivity extends Activity {
    private static final String BASE64_PUBLIC_KEY = "MIIBIjANBgkqhkiG ... ";
//truncated for this example
    ...
}
```

关闭IPC连接的时候调用**LicenseChecker**的**onDestory()**方法

最后, 让LVL在你的应用Context改变的时候清空, 在Activity的**onDestory()**实现里增加一个调用**LicenseChecker**的**onDestory()**的方法. 这会让**LicenseChecker**适时的关闭任何连接到Google Play 应用的**IllicensingService**的IPC连接, 并且删除任何service和handler的引用.

调用**LicenseChecker**的**onDestory()**失败会导致你的应用在生命周期内出现问题. 例如, 当用户在签名检查的时候改变屏幕方向, 应用的**Context**会被销毁. 如果你的应用不能适时的关闭**LicenseChecker**的IPC连接, 你用的应用会在接收到响应的时候崩溃. 类似的, 如果用户在签名检查的时候退出应用会导致应用在接收到响应的时候崩溃, 除非你在断开连接的时候正确的调用**LicenseChecker**的**onDestory()**.

下面是LVL里的例子, mChecker是LicenseChecker的实例:

```
@Override
protected void onDestroy() {
    super.onDestroy();
    mChecker.onDestroy();
    ...
}
```

如果你继承或修改LicenseChecker, 你可能还需要调用LicenseChecker的finishCheck()清理所有IPC连接.

实现DeviceLimiter

有些时候, 你可能希望你的Policy限制一些实际的设备允许使用单一的签名. 这可以防止用户将签名的应用移动到其他设备上并且在这些设备上通过同一个账号使用应用. 它还可以防止用户通过提供账号关联的签名给其他人共享应用, 他们随后就可以在他们的设备上登录账号并授权给应用.

LVL支持通过DeviceLimiter接口给每台定义了方法allowDeviceAccess()的设备签名. 当LicenseValidator处理来自签名服务器的响应数据时候, 它会调用allowDeviceAccess(), 把响应数据中的用户ID传进去.

如果你不行限定设备, 这不是必需的工作——LicenseChecker类会自动使用默认的实现NullDeviceLimiter. 顾名思义, NullDeviceLimiter是一个"无操作"的类, 它的allowDeviceAccess()方法对所有用户和设备只会简单的返回LICENSED.

有些时候, 你可能希望你的Policy限制一些实际的设备允许使用单一的签名. 这可以防止用户将签名的应用移动到其他设备上并且在这些设备上通过同一个账号使用应用. 它还可以防止用户通过提供账号关联的签名给其他人共享应用, 他们随后就可以在他们的设备上登录账号并授权给应用.

LVL支持通过DeviceLimiter接口给每台定义了方法allowDeviceAccess()的设备签名. 当LicenseValidator处理来自签名服务器的响应数据时候, 它会调用allowDeviceAccess(), 把响应数据中的用户ID传进去.

如果你不行限定设备, 这不是必需的工作——LicenseChecker类会自动使用默认的实现NullDeviceLimiter. 顾名思义, NullDeviceLimiter是一个"无操作"的类, 它的allowDeviceAccess()方法对所有用户和设备只会简单的返回LICENSED.

警告: Per-device 签名对大多数应用不建议使用, 因为:

- 它需要你提供后端服务器来管理用户和设备的映射
- 它可能在不经意间导致用户在另一个设备上无法访问他们合法购买的应用

混淆你的代码

确保你的应用安全, 特别对于使用签名和/或自定义的约束和保护的付费应用来说, 混淆你的应用程序的代码是非常重要的. 适当的混淆你应用的代码可以让恶意用户反编译你的应用程序字节码变得非常困难, 修改它——例如移除签名检查的部分, 然后重新编译.

大多数混淆处理程序对 android 应用都是有效的, 包括 ProGuard 也提供了代码优化的特性. 强烈推荐使用了 Google Play Licensing 的应用使用 ProGuard 或类似的程序混淆你的代码.

发布签名过的应用

你测试完你的签名之后, 就可以准备在 Google Play 上发布你的应用了. 按照正常的步骤准备、登录, 然后发布你的应用.

删除 Copy Protection

当你上传签名过的应用之后, 记得在应用里删除 copy protection 如果它现在在使用. 登录开发者网站进入应用上传的详情页, 检查并删除 copy protection. 在发布(Publishing)选项部分, 确保 Copy Protection 单选按钮选项是“关闭”(Off)状态.

获得支持

如果你在实现或部署发布你的应用的时候有问题或遇到问题, 请使用下面表格中的支持资源列表. 通过指导你查询正确的论坛, 你可以更快速的获得你需要的支持.

表2 Google Play 授权服务的开发者支持资源

支持类型	资源	主体范围
开发和测试问	Google 群组: android-developers	LVL 下载和集成, lib 项目, Policy 问

题

题, 用户体验建议, 处理响应,
Obfuscator, IPC, 测试环境设置

Stack

Overflow: <http://stackoverflow.com/questions/tagged/android>

账号, 发布和部署问题

Google Play 帮助论坛

开发者账号, 签名密钥, 测试账号, 服务器响应, 测试响应, 应用部署和结果

授权常见问题

LVL 问题跟踪

Marketlicensing project 问题跟踪

LVL源码和接口实现的Bug和问题报告

怎么把一般信息发送到上面的群组列表里去, 见Resource标签页的 Developer Forums 文档
 来自 "[index.php?title=Adding_Licensing_to_Your_App&oldid=8463](#)"

Licensing Reference

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：sfshine

原文链接：<http://developer.android.com/guide/google/play/licensing/licensing-reference.html>

目录

[[隐藏](#)]

[1 Licensing Reference-许可参考](#)

- [1.1 LVL Classes and Interfaces-LVL类和接口](#)
- [1.2 Server Response Codes-服务器返回码](#)
- [1.3 Server Response Extras-服务器返回值](#)
 - [1.3.1 License validity period-许可有效阶段](#)
 - [1.3.2 Retry period and maximum retry count-试用时间和最大使用次数](#)

Licensing Reference-许可参考

LVL Classes and Interfaces-LVL类和接口

Table 1 lists all of the source files in the License Verification Library (LVL) available through the Android SDK. All of the files are part of the com.android.vending.licensing package.

表一列出了通过AndroidSDK可以获得的许可认证库(LVL)的源代码文件。这些文件都是com.android.vending.licensing这包的一部分。 Table 1. Summary of LVL library classes and interfaces.

表一：LVL库的类文件和接口

Category 目录	Name 名字	Description 描述
License check and result 许可查看和结果	LicenseChecker	Class that you instantiate (or subclass) to initiate a license check. 一个您实例化或者子类话的类,这个类用来开始一个许可检查。
	LicenseCheckerCallback	Interface that you implement to handle result of the license check. 您可以通过实现这个接口来处理许可检

查的结果。

Policy策略	Policy	Interface that you implement to determine whether to allow access to the application, based on the license response.通过实现这个接口您可以决定是否允许通过许可反馈来访问应用
	ServerManagedPolicy	Default Policy implementation. Uses settings provided by the licensing server to manage local storage of license data, license validity, retry.默认的Policy接口。使用许可服务器提供的设置来管理本地的存储数据，许可合法性和尝试。
	StrictPolicy	Alternative Policy implementation. Enforces licensing based on a direct license response from the server only. No caching or request retry.本地Policy接口。强制许可基于来自服务器的直接的许可响应。不缓存也不请求重试。
Data obfuscation(optional)	Obfuscator	Interface that you implement if you are using a Policy (such as ServerManagedPolicy) that caches license response data in a persistent store. Applies an obfuscation algorithm to encode and decode data being written or read. 如果您使用Policy (比ServerManagedPolicy) , 您将继承这个接口。这个接口把许可响应数据缓存在一个持久存储区里。在被读写的时候，它使用模糊算法来编码和解码。
	AESObfuscator	Default Obfuscator implementation that uses AES encryption/decryption algorithm to obfuscate/unobfuscate data.使用AES加密/解密算法来混淆/反混淆数据的默认算法接口。

Device limitation(optional)	DeviceLimiter	Interface that you implement if you want to restrict use of an application to a specific device. 实现这个接口，您可以限制应用在特定设备上的使用。Called from LicenseValidator. 调用来自LicenseValidator。 Implementing DeviceLimiter is not recommended for most applications because it requires a backend server and may cause the user to lose access to licensed applications, unless designed with care. 对于大多数应用，我们不推荐继承DeviceLimiter接口，它需要一个后台服务器并且它可能导致用户和许可的应用失去联系，除非在设计的时候认真 的考虑了。
	NullDeviceLimiter	Default DeviceLimiter implementation that is a no-op (allows access to all devices).默认的DeviceLimiter接口，这个接口允许访问任何设备。
Library core, no integration needed	ResponseData	Class that holds the fields of a license response.这个类控制一个许可响应的区域
	LicenseValidator	Class that decrypts and verifies a response received from the licensing server.这个类用来解密并验证从许可服务器发来的响应
	ValidationException	Class that indicates errors that occur when validating the integrity of data managed by an Obfuscator.这个类指出在验证Obfuscator管理的数据的完整性时出现的错误
	PreferenceObfuscator	Utility class that writes/reads obfuscated data to the system's SharedPreferences store.这个工具类用于读写混淆数据存储到系统

		的Sharedpreferences
ILicensingService		One-way IPC interface over which a license check request is passed to the Google Play client. 单向的进程间通讯接口，这个接口的许可检查请求来自google play客户端
ILicenseResultListener		One-way IPC callback implementation over which the application receives an asynchronous response from the licensing server. 单向进程间通讯调用的实现，这个实现基于应用收到的来自许可服务器的异步响应

Server Response Codes-服务器返回码

Table 2 lists all of the license response codes supported by the licensing server. In general, an application should handle all of these response codes. By default, the LicenseValidator class in the LVL provides all of the necessary handling of these response codes for you.

表二列出了许可服务器所支持的所有许可返回码。一般的，一个应用应该处理所有的这些返回码。默认的，LVL中的LicenseValidator(许可校验码)类提供所有需要处理的响应代码

Table 2. Summary of response codes returned by the Google Play server in a license response.

表二 google play商店服务器以许可响应的形式返回的响应代码

Response Code 返回码	Description 描述	Signed? 是否签名?	Extras 值	Comments 评价
LICENSED	The application is licensed to the user. The user has purchased the application or the application only exists as a draft. 这个应用被授权给这个用户,这个用户购买了这个	Yes 是	VT, GT, GR	Allow access according to Policy constraints 经过Policy验证后运行使用应用

	应用,或者这个应用仅仅是一个草稿版本(已经上传,但是没有发布).			
LICENSED_OLD_KEY	The application is licensed to the user, but there is an updated application version available that is signed with a different key. 这个应用被授权给这个用户,这个应用有升级版,但是升级版的签名和当前的应用不同	Yes是	VT, GT,GR, UT	Optionally allow access according to Policyconstraints. Can indicate that the key pair used by the installed application version is invalid or compromised. The application can allow access if needed or inform the user that an upgrade is available and limit further use until upgrade. 有选择性的允许使用应用.可以表明已安装的应用所使用的密钥不存在或者已经被损坏.如果需要应用可以被允许使用或者通知用户需要进行升级才能使用
NOT_LICENSED	The application is not licensed to the user. 这个应用没有授权给这个用户	No		Do not allow access.不允许使用该应用
ERROR_CONTACTING_SERVER	The application is not licensed to	No		according toPolicy retry limits.试用次数取

	<p>the user. Local error — the Google Play application was not able to reach the licensing server, possibly because of network availability problems. 本地错误—google play 应用不能连接到许可服务器,可能是因为网络连接问题</p>		决于Policy
ERROR_SERVER_FAILURE	<p>Server error — the server could not load the publisher account's key pair for licensing. 服务器错误—服务器不能获取发布的密钥来进行许可操作</p>	No	<p>Retry the license check according toPolicy retry limits. 根据Policy的使用次数重新进行许可检查</p>
ERROR_INVALID_PACKAGE_NAME	<p>Local error — the application requested a license check for a package that is not installed on the device. 本地错误,这个应用需要一个许可证来对包进行许可检查,但是这个许可证没有安装在设备上</p>	No	<p>Retry the license check according toPolicy retry limits. 根据Policy的使用次数重新进行许可检查 Do not retry the license check. Typically caused by a development error. 不要进行许可证检查,一般是由开发错误引起.</p>

ERROR_NON_MATCHING_UID	Local error — the application requested a license check for a package whose UID (package, user ID pair) does not match that of the requesting application. 本地错误---这个应用需要一个许可证来对安装包进行检查,但是这个许可证的UID(包,用户ID)和当前的请求应用不相吻合.	No	Do not retry the license check. Typically caused by a development error. 不要进行许可检查,一般是由应用开发错误
ERROR_NOT_MARKET_MANAGED	Server error — the application (package name) was not recognized by Google Play. 服务器错误 应用(包名)没有被google play识别.	No	Do not retry the license check. Can indicate that the application was not published through Google Play or that there is an development error in the licensing implementation. 不要进行许可检查,可以表明应用没有在google play上发布或者在开发中许可实现的时候存在错误

Note: As documented in Setting Up The Testing Environment, the response code can be manually overridden for the application developer and any registered test users via the Google Play publisher site.

注意:就像在设置测试环境一张描述的那样,相应码可以手动被google play的注册开发者或者测试者重写.

Additionally, as noted above, applications that are in draft mode (in other words, applications that have been uploaded but have never been published) will return LICENSED for all users, even if not listed as a test user. Since the application has never been offered for download, it is assumed that any users running it must have obtained it from an authorized channel for testing purposes.

除此之外,如上所述,在"草稿"状态的应用(这个应用已经上传,但是从未发布)将会返回LICENSED 给所有的用户,即使这些用户不是列出的测试用户.由于这个应用从来没有提供下载,所以这个应用被视为从一个授权渠道拿来做测试之用的

Server Response Extras-服务器返回值

To assist your application in managing access to the application across the application refund period and provide other information, The licensing server includes several pieces of information in the license responses. Specifically, the service provides recommended values for the application's license validity period, retry grace period, maximum allowable retry count, and other settings. If your application uses APK expansion files, the response also includes the file names, sizes, and URLs. The server appends the settings as key-value pairs in the license response "extras" field. 为了帮助您的应用管理用户在退款期间的使用并提供其他信息,许可服务器包含了几组许可返回信息.具体来说,服务器提供应用许可证有效期期间的建议值, 重试 宽限期, 允许的最大重试次数, 以及其他设置。如果您的应用使用了apk扩充文件,那返回值也包括文件名,文件大小,URL等.服务器在许可响应的"额外"领域作为键值对追加设置项.

Any Policy implementation can extract the extras settings from the license response and use them as needed.

任何Policy类的继承可以从许可返回信息中提取设置的值,在需要的时候使用他们.

The LVL default Policy implementation, ServerManagedPolicy, serves as a working implementation and an illustration of how to obtain, store, and use the settings.

LVL的默认Policy类默认继承 ServerManagedPolicy,实现功能,说明怎么获取,存储,使用这些设置.

Table 3. Summary of license-management settings supplied by the Google Play server in a license response.

表3 google play服务器以一个许可响应的形式提供的许可管理设置

Extra	Description
VT	License validity timestamp. Specifies the date/time at which the current (cached) license response expires and must be rechecked on the licensing server. See the section below about License validity period. 许可有效期

时间戳.指定了当前(或是缓存的)许可响应已经过期,必须去许可服务器上复查. 有关许可证有效期, 请参阅下文

GT	Grace period timestamp. Specifies the end of the period during which a Policy may allow access to the application, even though the response status is RETRY. 宽限期的时间戳.指定在许可响应信号是RETRY的时候允许应用使用的最后日期. The value is managed by the server, however a typical value would be 5 or more days. See the section below about Retry period and maximum retry count.这个值是由服务器管理的,一般这个值是5天或者更多.参阅下面的使用期限和最大使用次数
GR	<p>Maximum retries count. Specifies how many consecutive RETRY license checks the Policy should allow, before denying the user access to the application.</p> <p>最大使用次数,指定在禁止用户使用应用前,Policy可以允许多少次连续的RETRY许可检查信号.</p> <p>The value is managed by the server, however a typical value would be "10" or higher. See the section below about Retry period and maximum retry count.</p> <p>这个值也是由服务器管理,一般是"10"或者更大,参阅下面的使用期限和最大使用次数</p>
UT	<p>Update timestamp. Specifies the day/time when the most recent update to this application was uploaded and published. 更新时间戳.指定应用新版本上传或发布的日期和时间.</p> <p>The server returns this extra only for LICENSED_OLD_KEYS responses, to allow the Policy to determine how much time has elapsed since an update was published with new licensing keys before denying the user access to the application. 服务器只返回LICENSED_OLD_KEYS的响应来允许Policy决定一个含有新的许可信息的更新版本发布后还有多少时间来禁止用户使用旧版本</p>
FILE_URL1orFILE_URL2	The URL for an expansion file (1 is for the main file, 2 is the patch file). Use this to download the file over HTTP. 扩展文件的URL(1代表主文件,2是补丁文件). 使用这个URL来从HTTP上下载文件.

FILE_NAME1orFILE_NAME2	The expansion file's name (1 is for the main file, 2 is the patch file). You must use this name when saving the file on the device. 扩展文件的名字(1是主文件,2是补丁文件).在设备上保存这个文件时,您必须使用这个名字
FILE_SIZE1orFILE_SIZE2	The size of the file in bytes (1 is for the main file, 2 is the patch file). Use this to assist with downloading and to ensure that enough space is available on the device's shared storage location before downloading. 文件的大小,以字节为单位(1是主文件,2是补丁文件).使用他来辅助下载并在下载前确保设备的存储空间足够.

License validity period-许可有效阶段

The Google Play licensing server sets a license validity period for all downloaded applications.

Google play许可服务器对所有的下载了的应用设置了一个有效期.

The period expresses the interval of time over which an application's license status should be considered as unchanging and cacheable by a licensing Policy in the application.

这个阶段在表述时,一个应用许可状态的时间间隔应该作为没有在变化和可以通过应用中的Policy许可证被缓存来考虑.

The licensing server includes the validity period in its response to all license checks, appending an end-of-validity timestamp to the response as an extra under the key VT.

许可服务器在他的对所有许可检查响应中包含了有效期,这个操作通过在响应中追加有效期结束时间戳作为VT键下额外的数值来实现的.

A Policy can extract the VT key value and use it to conditionally allow access to the application without rechecking the license, until the validity period expires.

一个Policy可以把VT的键提取出来然后使用它来在不重新检查许可证的情况下使用应用,直到有效时间结束.

The license validity signals to a licensing Policy when it must recheck the licensing status with the licensing server.

当一个许可证必须使用许可服务器重新检查许可状态的时候, 许可证向一个许可Policy发一个有效性信号

It is not intended to imply whether an application is actually licensed for use.

他不是想要去暗示这个应用是否被许可可以使用.

That is, when an application's license validity period expires, this does not mean that the application

is no longer licensed for use — rather, it indicates only that the Policy must recheck the licensing status with the server.

而是,当一个应用的许可过期了,这不意味着这个应用再也不能被许可使用了—而是,他暗示只有Policy必须重新使用服务器检查许可状态.

It follows that, as long as the license validity period has not expired, it is acceptable for the Policy to cache the initial license status locally and return the cached license status instead of sending a new license check to the server.

只要许可有效期没有过期,他就可以被Policy接受并缓存最初的还没有过期的许可状态到本地,然后返回缓存的许可状态而不是向服务器发送一个新的许可检查.

The licensing server manages the validity period as a means of helping the application properly enforce licensing across the refund period offered by Google Play for paid applications. It sets the validity period based on whether the application was purchased and, if so, how long ago.

Specifically, the server sets a validity period as follows:

作为一种辅助在google play上的付费应用恰当的强制许可的方式,许可服务器会管理该应用的许可有效时间.他判断是否这个应用是付费的,如果是,允许使用多长时间 ,通过这个来设置许可的有效时间.具体来说,服务器设置有效期的方法如下:

- For a paid application, the server sets the initial license validity period so that the license response remains valid for as long as the application is refundable. A licensing Policy in the application may cache the result of the initial license check and does not need to recheck the license until the validity period has expired.
- 对于一个付费的应用,服务器通过设置最初的许可有效期来使许可响应保持有效.只要应用在退还款期.一个在应用中的许可Policy可能缓存最初许可证的检查结果,而不需要重新检查许可直到有效期结束.
- When an application is no longer refundable, the server sets a longer validity period — typically a number of days.
- 当一个应用不再可以退还,服务器会设置一个比较长的有效期,通常是很多天.
- For a free application, the server sets the validity period to a very high value (`long.MAX_VALUE`). This ensures that, provided the Policy has cached the validity timestamp locally, it will not need to recheck the license status of the application in the future.
- 对于一个免费的应用,服务器设置有效期为一个非常大的数值(`long.MAX_VALUE`).这样可以确保Policy已经在本地缓存了一个有效的时间戳,他在将来不需要重新检查许可状态.

The `ServerManagedPolicy` implementation uses the extracted timestamp (`mValidityTimestamp`) as a primary condition for determining whether to recheck the license status with the server before allowing the user access to the application.

`ServerManagedPolicy` (服务管理策略)的实现使用了提取到的时间戳(`mValidityTimestamp`)来作为基本的判断是否需要在允许用户使用该应用之前通过服务器重新检查许可状态的条件

Retry period and maximum retry count-试用时间和最大使用次数

In some cases, system or network conditions can prevent an application's license check from reaching the licensing server, or prevent the server's response from reaching the Google Play client

application. For example, the user might launch an application when there is no cell network or data connection available—such as when on an airplane—or when the network connection is unstable or the cell signal is weak.

在一些情况下,系统或网络条件可能阻止应用通过许可服务器进行许可检查或者阻止来自google play客户端的服务器响应.例如,用户可能在没有基站网络或者数据连接的环境下启动应用—比如在飞机上—或者,基站信号很弱或者网络不稳定.

When network problems prevent or interrupt a license check, the Google Play client notifies the application by returning a RETRY response code to the Policy's processServerResponse() method. In the case of system problems, such as when the application is unable to bind with Google Play's ILicensingService implementation, the LicenseChecker library itself calls the Policy processServerResponse() method with a RETRY response code.

当网络问题阻止或者中断了许可检查,googleplay客户端会通过返回一个RETRY响应码给Policy的processServerResponse()方法来通知应用这个情况.在系统问题情况下,比如当应用不能和google play的 ILicensingService服务绑定, LicenseChecker库会使用RETRY响应码来调用processServerResponse()方法.

In general, the RETRY response code is a signal to the application that an error has occurred that has prevented a license check from completing.

一般的,RETRY响应码一般表明应用存在一个错误导致不能完成许可检查.

The Google Play server helps an application to manage licensing under error conditions by setting a retry "grace period" and a recommended maximum retries count. The server includes these values in all license check responses, appending them as extras under the keys GT and GR.

Google play服务器通过设置一个"宽限期"并设置最多使用次数来帮助应用在存在错误的情况下处理许可.服务器在所有的许可检查响应中包含这些值,他们被作为GT和GR的追加值.

The application Policy can extract the GT and GR extras and use them to conditionally allow access to the application, as follows:

应用的Policy可以提取GT和GR值并使用它们来有条件允许用户使用应用程序:

- For a license check that results in a RETRY response, the Policy should cache the RETRY response code and increment a count of RETRY responses.
- 如果许可检查返回RETRY响应,Policy应该缓存RETRY响应码并记录,并不断增加RETRY的响应次数
- The Policy should allow the user to access the application, provided that either the retry grace period is still active or the maximum retries count has not been reached. The ServerManagedPolicy uses the server-supplied GT and GR values as described above. The example below shows the conditional handling of the retry responses in the allow() method. The count of RETRY responses is maintained in the processServerResponse() method, not shown.

如果试用的"宽限期"还没到或者最大使用次数还没到,那Policy应该允许用户访问应用.

ServerManagedPolicy使用服务器提供的GT和GR值.下面的例子在allow()方法中展示了怎么处理使用时的响应.RETRY的数量 在processServerResponse()方法中保存,在这里没有展示.

```
public boolean allowAccess() {
    long ts = System.currentTimeMillis();
```

```
if (mLastResponse == LicenseResponse.LICENSED) {
    // Check if the LICENSED response occurred within the validity timeout.
    if (ts <= mValidityTimestamp) {
        // Cached LICENSED response is still valid.
        return true;
    }
} else if (mLastResponse == LicenseResponse.RETRY &&
           ts < mLastResponseTime + MILLIS_PER_MINUTE) {
    // Only allow access if we are within the retry period or we haven't used up our
    // max retries.
    return (ts <= mRetryUntil || mRetryCount <= mMaxRetries);
}
return false;
}
```

来自“[index.php?title=Licensing_Reference&oldid=9014](#)”



Google Play Services

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

译:[CuGBabyBeaR](#)

原文:[Google Play Services](#)

完成时间:12.08.06

Google Play 服务

Google Play 服务是一个通过Google Play市场的递送平台，Google Play市场提供了将包括Google+在内的Google产品到Android应用里的整合。Google Play框架由一个运行在设备上的服务，和一个与您的应用打包在一起的lib库客户端组成。

简单的用户验证

您的应用可以通过使用用户已存在设备上的Google账户，而不用冗长的账号验证。只需要用户点几下您就设置完毕了。

[更多...](#)

整合Google+

Google Play服务把整合您的应用与登录Google+，+1按钮和Google+历史变得容易。

[更多...](#)

来自 "[index.php?title=Google_Play_Services&oldid=7430](#)"



Filters on Google Play

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

译:[CuGBabyBeaR](#)

原文:[Filters on Google Play](#)

完成时间:12.08.06

目录

[\[隐藏\]](#)

[1 Google Play过滤器](#)

- [1.1 快速预览](#)
- [1.2 在这篇文档中](#)
- [1.3 其他相关](#)

[2 过滤器是如何在Google Play上工作的](#)

[3 基于Manifest元素的过滤器](#)

[4 高级manifest过滤器](#)

[5 其他过滤器](#)

[6 发布具有不同过滤器的多个APK](#)

Google Play过滤器

当一个用户用安卓设备在Google Play上搜索或者浏览时，结果将会基于应用是否与该设备兼容而被过滤。

例如，如果一个应用需要一个摄像头（在应用的manifest文件中指定），那么Google Play将不会在任何没有摄像头的设备上显示这个应用。

manifest文件中的声明与设备的硬件是否相符不是唯一的过滤方式。过滤同时可能基于用户的国家和运营商、是否插入SIM卡等其他的因素。

Google Play过滤器的改变是与Android平台自身的改变相独立的。此文档将会定期更新，以反映影响Google Play过滤应用方式的任何改变。

快速预览

- Google Play使用过滤器控制着安卓系统设备在用户访问市场时能否获取您的应用。
- 通过比较您在应用的manifest文件中所声明的配置和设备定义的配置，以及其他一些因素来进行过滤。

在这篇文档中

[过滤器是如何在Google Play上工作的](#)

过滤器是如何在Google Play上工作的

Google Play使用如下所述的过滤器控制，以判定是否向正在使用Google Play app浏览或搜索应用的用户显示您的应用。当决定是否显示您的应用时，Google Play检查设备的硬件和软件配置，以及它的网络运营商、位置、和其他参数。然后Google Play会将这些信息与由应用的manifest文件所表示的限制项和依赖项以及发布细节相比较。如果根据过滤器规则应用适合于该设备，Google Play将应用显示给用户。否则，即便用户通过点击直接指向该应用ID的深层链接而向该应用提出明确请求，Google Play也会从搜索结果和浏览分类中将您的应用隐藏。

注意：当用户浏览[Google Play网站](#)时，他们可以看见所有已发布的应用。但是，Google Play网站会将应用的需求与 用户所注册的每一个设备的兼容性 相比较，并且只能允许他们安装与他们的设备所兼容的应用。

您可以对您的应用使用任意可用过滤器的组合。例如，您可以在应用里设置minSdkVersion的要求4，设置 smallScreens="false"，然后上传应用到Google Play时 您可以只将European countries (运营商)设为对象。于是Google Play的过滤器会阻止这个应用显示在任何不满足这三个要求的设备上。

所有的过滤器限制都和应用的版本所联系，并且可以在不同的版本之间改变。例如，如果一个用户安装了您的应用，然后您发布了一个更新使得这个应用对此用户不可见，这个用户不会看见这个可用更新。

基于Manifest元素的过滤器

大多数过滤器是由应用的manifest文件中的元素所触发的，这个manifest文件是[AndroidManifest.xml](#)（尽管不是manifest文件里所有的东西都能触发过滤器）。表1列出了您可以用来触发过滤器的manifest元素，以及解释过滤器是如何作用到这些元素上的。

表1. 触发Google Play过滤器的manifest元素。

manifest元素	过滤器名	工作方式
<code><supports-screens></code>	Screen Size 屏幕大小	应用通过设置 <code><supports-screens></code> 元素的属性，来指定能够支持的屏幕的大小。在应用发

基于Manifest元素的过滤器

高级manifest过滤器

其他过滤器

发布具有不同过滤器的多个APK

其他相关

Android Compatibility

<supports-gl-texture>

<supports-screens>

<uses-configuration>

<uses-feature>

<uses-library>

<uses-permission>

<uses-sdk>

布后，Google Play会基于用户设备的屏幕大小，使用这些属性来判断是否显示给用户。

一般说来，Google Play假定设备上的平台能够将较小的布局适应于较大的屏幕，但是不能讲较大的布局兼容较小的屏幕。因此，如果一个应用声明屏幕尺寸只支持“normal”中等尺寸，Google Play会将这个应用显示在中等和大尺寸屏幕的设备上，但不会显示在小尺寸屏幕的设备上。

如果一个应用不声明[`<supports-screens>`](#)的属性，Google Play会为这些属性套用默认值，默认值基于API版本有所不同。

具体做法是：

- 对于那些将[`android: minSdkVersion`](#)或[`android: targetSdkVersion`](#)设置为3或以下的应用，[`<supports-screens>`](#)元素自身未定义并且没有可用属性。在这种情况下Google Play假定应用是为中等尺寸的屏幕设计的，并且在有中等和大尺寸屏幕的设备上显示这个应用。
- 当[`android: minSdkVersion`](#)或[`android: targetSdkVersion`](#)设置为4或者更高的时候，所有属性的默认值都是“true”。在这种方式下，这个应用被认为是默认支持所有屏幕尺寸。

例1：

Manifest声明<uses-sdk
`android:minSdkVersion="3"`> 并且没有包括[`<supports-screens>`](#)元素。结果如下：Google Play不会向小尺寸屏幕设备的用户显示这个APP，但会向中等和大尺寸屏幕设备的用户显示，除非不满足其他过滤器。

例2：

Manifest声明<uses-sdk
`android:minSdkVersion="3"`
`android:targetSdkVersion="4"`>并且没有包括[`<supports-screens>`](#)元素。结果如

下: Google Play会向使用任何设备的用户显示这个应用，除非不满足其他的过滤器。

例3:

Manifest声明<uses-sdk

android:minSdkVersion="4">并且没有包括<supports-screens>元素。结果如下: Google Play会向所有用户显示这个应用，除非不满足其他的过滤器。

获取更多有关如何声明您的应用所支持屏幕尺寸的信息，请浏览[<supports-screens>](#)和[Supporting Multiple Screens](#)。

[<uses-configuration>](#)

Device Configuration:

keyboard,
navigation, touch
screen

设备配置:

键盘, 轨迹球, 触摸屏

应用可以要求某项硬件特征，Google Play将只在具备此硬件的设备上显示这个应用。

例1:

Manifest包含了<uses-configuration

android:reqFiveWayNav="true" />，然后一个用户在不具备五向导航控制的设备上搜索应用。结果: Google Play不会向此用户显示这个应用。

例2:

Manifest没有包含<uses-configuration>元素。

结果: Google Play会向所有用户显示这个应用，除非不满足其他的过滤器。

更多细节见[<uses-configuration>](#)。

[<uses-feature>](#)

Device Features
(name)

设备特征

应用可以要求设备上需要某一设备特征。这项功能自Android 2.0 (API Level 5) 引入。

例1:

	(特征名)	<p>Manifest包含<uses-feature android:name="android.hardware.sensor.light"/>，一个用户在不具备光线传感器的设备上搜索应用。结果：Google Play不会向该用户显示这个应用</p> <p>例2：</p> <p>Manifest没有包含<uses-feature>元素。结果：Google Play会向所有用户显示这个应用，除非不满足其他的过滤器。</p> <p>完整信息请访问<u><uses-feature></u></p> <p>基于隐含的功能的过滤器：在某些情况下，Google Play将<uses-permission>元素请求的权限解释并折合为<uses-feature>中声明的设备特征。见下文的<uses-permission>。</p>
	<p>OpenGL-ES Version (openGLESVersion)</p> <p>OpenGL-ES版本</p>	<p>应用可以使用<uses-feature android:openGLESVersion="int">属性，要求设备支持一个特定的OpenGL-ES版本。</p> <p>例1：</p> <p>一个应用通过在manifest中多次设置openGLESVersion属性，来请求多个OpenGL-ES版本。结果：Google Play将获取该应用所指定的最高版本。</p> <p>例2：</p> <p>一个应用要求OpenGL-ES版本号1.1，然后一个用户在OpenGL-ES版本2.0的设备上搜索应用。结果：Google Play会向这个用户显示这个应用，除非不满足其他过滤器。如果一个设备报告他支持OpenGL-ES版本X，Google Play会假定它同样支持早于X的其他版本。</p> <p>例3：</p> <p>一个用户在一个没有报告OpenGL-ES版本的设</p>

备（例如Android 1.5或更早版本的设备）上搜索应用。结果：Google Play认为这个设备只支持OpenGL-ES 1.0。Google Play只会向此用户显示没有指定openGLESVersion或者指定OpenGL-ES版本1.0或更低的应用。

例4：

Manifest没有指定openGLESVersion。结果：Google Play会向所有用户显示这个应用，除非不满足其他的过滤器。

访问[<uses-feature>](#)获取更多信息。

<uses-library>

Software Libraries

应用可以要求设备上具有指定的公共库。

例1：

一个应用指定 com.google.android.maps 库，而一个用户使用没有com.google.android.maps库的设备搜索应用。结果：Google Play不会向此用户显示该应用。

例2：

Manifest没有包含<uses-library>元素。结果：Google Play会向所有用户显示这个应用，除非不满足其他的过滤器。

访问[<uses-library>](#)获取更多细节。

<uses-permission>

严格说来，Google Play并不基于<uses-permission>元素进行过滤。但Google Play确实读取了这个元素，用以确定是否有应用所用到的硬件功能，没有在<uses-feature>里被正确的声明。例如，如果一个应用申请了CAMERA权限，但是没有在<uses-feature>声明android.hardware.camera，Google Play认为

这个应用请求了一个摄像头，不应该显示给使用不支持摄像头设备的用户。

一般来说，如果一个应用请求了硬件相关的权限，Google Play会认为这个应用隐式地请求了硬件特征，即使这些硬件特征的请求可能与`<uses-feature>`的声明不相符。然后Google Play会基于这个隐式声明的`<uses-feature>`特征设置过滤。

访问`<uses-library>`元素以获得各个权限隐含的硬件特征列表。

`<uses-sdk>`

Minimum Framework Version
(minSdkVersion)

最小框架版本

应用可以要求一个最低API版本。

例1：

应用的Manifest包含了`<uses-sdk android:minSdkVersion="3">`，并且使用的API是API Level 3。一个使用API Level 2设备的用户搜索应用。结果：Google Play不会将该应用显示给此用户。

例2：

manifest没有声明`minSdkVersion`，并且引入的API是API Level 3。一个用户使用API Level 2的设备搜索应用。结果：Google Play默认`minSdkVersion`是"1"，这个应用适合于Android的所有版本。Google Play会给这个用户显示这个应用，并允许用户下载此应用。然后这个应用在运行的时候崩溃了。

由于您想避免第二种情形，我们建议您始终声明一个`minSdkVersion`。浏览[android:minSdkVersion](#)以获得更多细节。

Maximum

<p>Framework Version (maxSdkVersion)</p> <p>最大框架版本</p>	<p>不建议。Android 2.1 或更晚版本不检查或者不 enforce <code>maxSdkVersion</code> 属性，并且如果在应用的 manifest 里设置了 <code>maxSdkVersion</code>，SDK 将不编译。对于已经编译了 <code>maxSdkVersion</code> 的应用，Google Play 会尊重并将其作为过滤器使用。</p> <p>不推荐声明 <code>maxSdkVersion</code>。浏览 android:maxSdkVersion 以获得更多细节。</p>
---	---

高级manifest过滤器

作为表1中manifest元素的补充，Google Play也会基于表2中一些高级的manifest元素过滤应用。

这些manifest元素和他们所触发的过滤器只适用于一些特定的使用情况。这些是为了那些在应用分配上需求绝对控制的高性能游戏或类似应用而设计的。大部分应用应该绝对用不上这些过滤器。

表2. 适用于Google Play过滤器的高级manifest元素

Manifest 元素	摘要
<p><code><compatible-screens></code></p> <p>如果设备屏幕尺寸和密度（density）不符合<code><compatible-screens></code>元素里的任一屏幕设置（通过一个<code><screen></code>元素声明），Google Play将会过滤该应用。</p> <div style="border-left: 2px solid orange; padding-left: 10px; margin-top: 10px;"> <p>注意：通常情况下，您不需要使用这个manifest元素。使用这个元素会通过排除所有您未列出的所有屏幕尺寸和密度配置，使您的应用显著地减少潜在用户。您应该使用<code><supports-screens></code>这个manifest元素（见上述表1），来为您未说明的，某些非主流的屏幕配置启用屏幕适配模式。</p> </div>	

<supports-gl-texture>

如果该应用支持的一个或者更多GL纹理压缩格式同样被设备支持, Google Play将不会过滤该应用。

其他过滤器

Google Play 使用其他应用的特征去判断是否向一个使用所列出的设备的特定用户显示一个应用, 如下表所述。

表3. 作用于Google Play过滤器的应用和发布特征。

过滤器名	工作方式
Publishing Status 发布状态	<p>只有被发布的应用才会允许在Google Play中搜索和浏览到。</p> <p>即使一个应用是未发布的, 但如果用户可以在他们的, 包括“已购买”、“已安装”或“最近卸载”在内的下载空间中看见它, 他也可以安装这个应用。</p> <p>如果一个应用被挂起, 用户将不能重新安装或者升级它, 即使它出现在用户的Downloads中。</p>
Priced Status 售价状态	并不是所有的用户都能看见付费应用。为了显示付费应用, 这个设备必须要有一个SIM卡, 并且运行在Android 1.1或者更高的环境, 同时必须在付费应用可用的国家 (由SIM卡运营商判断) 内。
Country / Carrier Targeting 针对国家/运营商	<p>当您上传你的应用到Google Play时, 您可以选择一个特定的国家作为目标。这个应用将只会对您所选择的国家 (运营商) 可见, 方法如下:</p> <ul style="list-style-type: none"> 如果运营商是可用的, 那么设备所在的国家是基于运营商来判断的。如果没有运营商可以用来判断, Google Play会试图基于IP判断国家。

	<ul style="list-style-type: none"> 运营商是基于设备的SIM卡决定的（对于GSM设备来讲），而不是当前漫游的运营商。
Native Platform 本地平台	一个包括本地指向特定平台（例如ARM EABI v7 或者 x86）库的应用，是仅在支持该平台的设备上可见的。浏览 What is the Android NDK 来获取NDK相关以及使用本地库的更多细节。
Copy-Protected Applications 有防拷保护的应用	当您设置发布选项时，通过设置拷贝保护为“On”，为您的应用设定防拷保护。Google Play将不会向开发者的设备或者未发布的设备显示有拷贝保护的应用。

发布具有不同过滤器的多个APK

一些特定的Google Play过滤器允许您，为了向不同配置的设备提供不同的APK，而对同一应用发布不同的APK。例如，如果你正在创建一个使用高品质图形assets的视频游戏，你或许想创建两个APK来支持不同的纹理压缩格式。这样你就可以通过只包含该设备设置所需求的纹理，而减少APK文件的大小。Google Play会根据每个设备对您的纹理压缩格式的支持，提供您声明支持的该设备的APK。

目前，只有当每个APK提供不同的基于如下设置的过滤器时，Google Play才会允许您为同一个应用发布多个APK。

- OpenGL texture compression formats OpenGL纹理压缩格式

通过使用[`<supports-gl-texture>`](#) 元素。

- Screen size (and, optionally, screen density) 屏幕尺寸(以及屏幕密度(可选))

通过使用[`<supports-screens>`](#) 或 [`<compatible-screens>`](#) 元素

- API level API版本

通过使用[`<uses-sdk>`](#) 元素。

所有其他的过滤器依然是正常工作的，但只有这三个过滤器能区别同一个应用中Google Play列出的不同APK。例如你不能只是基于设备是否具有摄像头，为一个应用发布多个APK。

注意：为同一个应用发布多个APK 是作为一个高级功能而考虑，大多数应用应该只发布一个设备配置支持面很广的APK。发布多个APK要求您遵循您的过滤器内的具体规则，并额外的注意每个APK的版本代码，确保每个配置正确的更新路径。

如果您需要更多有关如何在Google Play发布多个APK的信息，请阅读[Multiple APK Support](#)。

来自“[index.php?title=Filters_on_Google_Play&oldid=7431](#)”



Multiple APK Support

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：青冥

原文链

接：<http://developer.android.com/guide/google/play/publishing/multiple-apks.html>

目录

- [1 Multiple APK Support\(相同的应用程序有多个APKs支持\)](#)
 - [1.1 在发布前注意的内容](#)
 - [1.1.1 Active APKs](#)
 - [1.1.2 简单模式和高级模式](#)
 - [1.2 Multiple APKs如何工作](#)
 - [1.2.1 支持过滤器](#)
 - [1.2.2 发布multiple APKs的规则](#)
 - [1.3 创建 Multiple APKs](#)
 - [1.3.1 指定版本的代码](#)
 - [1.3.2 定制version code](#)
 - [1.3.3 使用一个版本的代码scheme](#)
 - [1.4 Using a Single APK Instead](#)
 - [1.4.1 支持multiple GL textures](#)
 - [1.4.2 支持多个屏幕](#)
 - [1.4.3 支持多种API的levels](#)

Multiple APK Support(相同的应用

程序有多个APKs支持)

多种apk的支持是一个特点在Google Play，它允许你发布不同的APKs为你的应用匹配不同尺寸的设备。每个APK是您的应用程序的完整和独立的版本，但它们共享同一应用程序在Google Play上市，必须共享相同的包的名称，并签署具有相同的release key。此功能用在您的应用不能通过单一的APK来兼容所有设备的情况。

Android提供了许多的方式，您提供给尽可能多的设备。Android应用程序通常在最兼容的设备上运行，用一个单一的APK，通过提供不同的配置替代资源（例如，不同的布局，为不同的屏幕尺寸）和Android系统的设备在运行时选择适当的资源。然而，在少数情况下，一个单一的APK是不能支持所有设备的配置，因为替代资源的APK文件太大（大于50MB）或其他技术的挑战，阻止了一个APK在所有设备上工作。

虽然我们鼓励您开发和发布一个单一的APK，支持尽可能多的设备配置，但是这样做有时是不可能。为了让您的apk尽可能多的支持不同的设备，Google Play允许你发布多种的应用，在相同的应用列表下面。Google Play 然后根据你在每个APK的manifest文件中声明的版本信息，适配出相对应的apk。

通过发布多个您的版本的apks应用，您可以：

- 不同的OpenGL纹理压缩格式，支持每个的APK。
- 支持APK的每个不同的屏幕配置。
- 每个的APK支持不同的平台版本。

目前，这是唯一的设备的特点，Google Play 支持 发布相同的应用程序有多个APKs支持。

注意：只有当你的APK是太大（大于50MB），你应当使用多个APKs支持不同的设备配置。使用单一的APK，以支持不同的配置的设备始终是最好的做法，因为它使简单的应用程序更新的应用的途径和让用户很明白该怎样操作（也让你的开发变的简单，避免开发和发布的复杂性） Read the section below about Using a Single APK Instead to consider your

options before publishing multiple APKs

在发布前注意的内容

在你发布多个apks为同一个应用在Google Play，你必须理解一些关于怎样发布在Google Play上面的工作。

Active APKs

在你发布你的应用（无论是发布的一个或多个APKs）时，你必须“激活”从你的APK（S）apk文件选项卡上面。当你激活的APK，它将会移动一个激活的APKs列表。这个名单可以让你预览的你即将发布的APK（s）。

如果没有任何错误，任何“active”APK将公布到Google Play，当您单击“Publish”按钮（如果应用程序是未发布的），或当您单击“保存”按钮（如果应用程序已经发布）。

简单模式和高级模式

在Google Play上面提供了两种方式管理你的应用：简单模式和高级模式。你可以通过在Apk上面的选项卡，切换他们。简单的模式是传统的方式发布应用程序，使用一个apk在一次。

在简单模式，只有一个的APK可以激活一次。如果你上传一个新的APK（升级你的应用程序），要想激活升级的应用必须的取消当前的应用（您必须然后单击“保存”发布新的APK）。

高级模式允许你激活并发布多个APKs（每个apk都针对不同设备的配置）。然而，有些在清单中声明的规则的要求，是否你将要被允许激活apk。当您激活APK和违反规则之一，你会收到一条错误或警告消息。如果消息是一个错误，你可以不发布，直到你解决这个问题，如果它是一个警告，你可以发布ActiveAPKs，但您的应用程序是否适用于不同的设备可能有意想不到的后果。这些规则更下面的讨论。

Multiple APKs如何工作

使用Google Play Multiple APKs的概念是，你必须为您的应用程序只是一个entry在Google Play上面，但针对不同的设备可能会下载到一个不同的APK。这意味着：

- 你保持产品细节只有一组（应用程序的描述，图标，截图等）。这也意味着你不能收取不同的价格对不同APKs。
- 所有的用户只能看到您的应用程序的一个合适它设备的版本，所以它们不会混淆，你可能已经出版的“tablets（平板）”或“手机”。
- 所有用户评论是相同的应用程序列表，即使在不同的设备上的是Multiple APKs。
- 如果你发布不同版本的Android（不同的API级别），then when a user's device receives a system update that qualifies them for a different APK you've published,

Google Play updates the user's application to the APK designed for the higher version of Android。任何系统与应用程序相关的数据将被保留（与正常的使用single APK应用程序更新是一样的）。

发布多个APKs相同的应用程序，您必须启用高级模式，在您的应用程序的apk文件“选项卡（如在上一节讨论）”。一旦在高级模式下，你可以上传，激活，然后发布多个相同的应用程序APKs。以下各节描述更多的是它如何工作的。

支持过滤器

收到每个APK是确定Google Play APK的每个舱单文件中的元素指定的过滤器的设备。然而，Google Play允许您发布多个APKs只有当每个的APK使用filter，支持以下的设备特性的variation：

- OpenGL纹理压缩格式

这是基于你的清单文件的<supports-gl-texture>，元素。

例如，开发一个使用OpenGL ES的游戏时，你可以提供一个APK为设备支持ATI的纹理压缩和一个独立的APK支持PowerVR的压缩（等等）的设备。

- 屏幕尺寸（和可选，屏幕密度）

这是基于你的manifest文件的<supports-screens> 或 <compatible-screens>元素。你不应该使用这两种元素在同一时刻，你应该在尽可能的情况下只使用 <supports-screens>。

例如，您可以提供一个APK的支持小，正常大小的屏幕和其他的APK，支持大型和XLARGE屏幕。

注：Android系统，提供了很强大的支持，为一个APK支持所有的屏幕配置。你应该避免，创建多个APKs到支持不同的屏幕上，除非绝对必要，而遵循的指导，支持多个屏幕，使您的应用程序非常灵活，可以适应所有的屏幕配置的一个单一的APK。

注意：默认情况下，如果你不声明屏幕尺寸属性<supports-screens>元素在默认情况下是“真”。然而，由于android:xlargeScreens 属性增加在Android 2.3 (API 9级) 属性，如果你的应用程序不设置任何 android:minSdkVersion or android:targetSdkVersion to "9" 或者更高，Google Play将假设它是false。

注意：你不应该把<supports-screens>和<compatible-screens>一起使用。同时使用增加了机会，你会引入一个错误，因为它们之间的冲突。为帮助决定使用哪一个，请阅读 Distributing to Specific Screens。如果你不能避免同时使用，be aware that for any conflicts in agreement between a given size, "false" will win.

• API Level

这是基于您的manifest文件<uses-sdk>元素。你可以使用android: minSdkVersion和android: maxSdkVersion 属性来指定不同的API级别的支持。

例如，你可以发布你的应用程序一个APK的支持API level 4 - 7 (Android 1.6 - 2.1) ， 使用唯一的API level 4或者更低，如果支持的API level 8及以上 (Android 2.2 +) 使用API 的level为8 或者更低。

注意：

- If you usethis characteristic as the factor to distinguish multiple APKs, then the APKwith a higher android:minSdkVersion value must have a higher android:versionCode value. This is also true if two APKs overlap their devicesupport based on a different supported filter. This ensures that when a devicerceives a system update, Google Play can offer the user an update for yourapplication (because updates are based on an increase in the app version code).This requirement is described further in the section below about Rules for multiple APKs.
- 你应该避免使用android: maxSdkVersion在一般情况下，因为只要你正确使用public API开发您的应用程序，它始终是与未来版本的Android兼容。如果你想发布一个较高的API level不同的APK，你还没有需要到指定的最高版本，假如一个android: minSdkVersion是4，另一个是8，这个设备是8或者更高，那么这个设备将会选择android: minSdkVersion是8的APK。

Othermanifest elements that enable GooglePlay filters—but are not listed above—arestill applied for each APK as usual. However, Google Play does not allow you to publish multiple APKs based on variations of them. Thus, you cannot publishmultiple APKs if the above listed filters are the same for each APK (but theAPKs differ based on other characteristics in the

manifest file). For example, you cannot provide different APKs that differ purely on the <uses-configuration> characteristics

发布**multiple APKs**的规则

在你发布您的应用程序的**multiple APKs**之前，你需要了解以下的规则，关于发布**multiple APKs**如何定义：

- 相同的应用程序发布的所有APKs你必须有相同的包的名称，并使用相同的证书密钥签署。
- 每个的APK 必须有一个不同版本的代码，由指定的 android: versionCode属性。
- 每个的APK 不能完全匹配的配置支持，另外的APK（也就是说，一个apk不能包含支持别的配置的设备）。

That is, each APK must declare slightly different support for at least one of the supported Google Play filters (listed above).

通常情况下，你会区分你的APKs基于一个特定的特性（如支持的纹理压缩格式），因此，每个的APK将申明为不同的设备支持。然而，它确定发布他们的支持略有重叠的多个APKs。当两个APKs做重叠（他们支持一些相同的设备配置），将接收设备属于重叠的范围内具有较高的版本代码（通过定义的APK android: versionCode）。

- 你不能启动一个新的APK取代一个版本的代码较低。例如，假设你有一个active 的屏幕尺寸小的APK - 正常版本代码是0400，然后尝试更换一个版本代码为0300相同屏幕尺寸的APK。这就出现了一个错误，因为这意味着以前的APK的用户将无法更新的应用程序。
- 需要一个较高的APK level 必须有一个更高的版本的代码。

这是true的，只有当：1、APKs不同，仅仅根据支持的API的levels（没有其他支持的过滤器，区分彼此的APKs）2、APKs时使用另一个支持的过滤器为基础，但有该过滤器内的APKs之间的重叠。

这是重要的，因为用户的设备收到来自谷歌的应用程序更新版本的代码只有Google play的APK高于目前在设备上的APK版本。这将确保，如果设备收

到一个系统的更新，然后限定它以较高的API级别的APK安装，设备接收应用程序更新，因为该版本代码增加。

注：该版本的代码增加的大小是无关紧要的，它只是需要在更大的版本，支持更高的API levels。

下面是一些例子：

- 1、If an APK you've uploaded for API levels 4 and above (Android 1.6+) has a version code of 0400, then an APK for API levels 8 and above (Android 2.2+) must be 0401 or greater. In this case, the API level is the only supported filter used, so the version codes must increase in correlation with the API level support for each APK, so that users get an update when they receive a system update.
- 2、If you have one APK that's for API level 4 (and above) and small - large screens, and another APK for API level 8 (and above) and large - xlarge screens, then the version codes must increase. In this case, the API level filter is used to distinguish each APK, but so is the screen size. Because the screen sizes overlap (both APKs support large screens), the version codes must still be in order. This ensures that a large screen device that receives a system update to API level 8 will receive an update for the second APK.
- 3、If you have one APK that's for API level 4 (and above) and small - normal screens, and another APK for API level 8 (and above) and large - xlarge screens, then the version codes do not need to increase in correlation with the API levels. Because there is no overlap within the screen size filter, there are no devices that could potentially move between these two APKs, so there's no need for the version codes to increase from the lower API level to the higher API level.

如果，未能遵守上述规则将会在Google Play上出现错误，要想激活您APKs，直到你解决这个错误，否者您将无法发布您的应用程序。

还有其他您激活您的APKs时可能发生的冲突，但是这将导致警告，而不是错误。警告，可以由以下原因引起：

- 1、当你修改的APK设备的特点，并没有其他APKs支持的??设备，然后属于支持的范围以外的“缩水”的支持。例如，如果目前的APK支持small和normal大小的屏幕，你改变它仅支持small屏幕，那么你已经减少了对设备的支持，在Google Play一些设备将再也看不到你的应用程序。您可以通过增加另外的APK，支持正常大小的屏幕，这样所有设备便支持了，这个问题便得到了解决。
- 2、当有两个或更多APKs之间的“重叠”。例如，如果支持的APK sizes small, normal, and large, 而另外的APK支持large和XLARGE，有重叠，因为两个APKs支持large。如果不解决这个问题，那么large设备都将收到的z这两个APK具有最高的version code。

当发生这种冲突时，你会看到一条警告消息，但仍然可以发布您的应用程序。

创建 Multiple APKs

一旦你决定发布 Multiple APKs，你可能需为每个发布的应用，要创建单独的Android项目，可以适当分开开发的APK。你可以通过简单的复制现有的项目，并给它一个新名称。（另外，你可能会使用 build系统，可以输出为不同的资源 -如textures --根据不同build的配置。）

提示：一种方式，以避免重复您的应用程序代码的大部分是使用 library project。library project持有共享代码和资源，其中可以包括在您的实际应用项目。

相同的应用程序创建多个项目，这是一个好的做法，以确定每个人的名字，表示设备上的APK限制，所以你可以很容易地识别它们。例如“HelloWorld_8”可能是API 8级及以上的设计了一个应用程序的一个好名

注：相同的应用程序发布的所有APKs 必须具有相同的包的名称，并使用相同的证书密钥签署。可以肯定你也了解每个多APKs规则。

指定版本的代码

每个相同的应用程序的APK，必须有一个独特的 `version code`,，通过指定的`android:versionCode`属性。当你发布multiple APKs，一定要小心的指定`version code`,，因为他们每个人都必须是不同的，但在某些情况下，必须或应该被定义在一个特定的顺序，根据配置每个APK的支持。

定制**version code**

通常需要较高的API级别的APK必须有一个更高的版本代码。例如，如果您创建两个APKs以支持不同的API级别，较高的API级别的APK必须有更高版本的代码。这将确保，如果设备receiver系统的更新，然后限定它以较高的API级别安装的APK，用户接收到一个更新的应用程序的通知。如何为更多的信息关于怎样要求apples，请参阅上面有关节多个APKs规则。

你也应该考虑版本代码的顺序如何，可能会影响它的APK用户收到任何因重叠覆盖的不同APKs或未来的变化，你可能使你的APKs。

例如，如果你有不同APKs根据屏幕上 size, such as one for small - normal and one for large - xlarge，但预见到的时候，你会改变的APKs是一个为小型和一个normal - XLARGE，然后你应该做的大版本的代码 - XLARGE的APK是较高的。这样，一个正常大小的设备将得到相应的更新，当你做出改变，因为版本的代码增加从现有的APK新的APK，支持现在的设备。

此外，创建多个不同根据不同的OpenGL texture compression formats的支持AmultiplePKs时，要知道，许多设备支持多种格式。由于设备接收到的APK时有重叠覆盖两APKs的最高版本的代码，你应该之间的APKs order版本的代码，以便，首选的压缩格式的APK具有最高的版本代码。例如，您可

能要执行单独的版本，您的应用程序使用的PVRTC，ATITC，和ETC1压缩格式。如果你喜欢这些格式，在这个确切的顺序，然后PVRTC应当是高版本的代码，接着ETC1应该是最低版本的代码。因此，如果设备支持PVRTC和ETC1，它接收PVRTC的APK，因为它具有最高的版本代码。

使用一个版本的代码**scheme**

为了让不同APKs以更新其版本他人的独立代码（例如，当你修复了一个bug仅为一个apk，并不需要更新所有APKs），你应该使用**scheme** 为您的版本的代码，它提供的计划足够的空间，使彼此之间的APK可以增加一个代码，而不需要增加在其他。你还应该包括你的代码的实际版本的名称（就是，用户可见的分配机`android: versionName`），所以很容易为您关联的版本代码和版本名称。

注意：当您增加的APK版本的代码，Google Play 会提示用户以前的版本更新程序。因此，为了避免不必要的更新，你不应该增加为APKs版本代码。

我们建议用至少7位数的版本代码：interger表示支持的配置，是在更高的序位，the lower order bits的名称（从`android: versionName`）。例如，当应用程序版本的名称是3.1.0版本的代码，API level 4的APK和API level 11的APK，分别为0400310和1100310类似。前两个数字是该API的等级（4和11，分别）保留的中间2位是无论屏幕大小或的GL纹理格式（在这些例子中不使用），个数字为应用程序的版本名称（3.1.0）。图1显示了两个例子，基于两个平台版本（API level），屏幕尺寸size。



图1 一个版本代码的建议方案，使用API级别的前两个数字，第二和第三位的最大和最小的屏幕尺寸（1 - 4表示四个zizes）或者表示 the texture formats，应用程序版本的最后三个数字。

这个版本的代码的计划仅仅是一个建议，你应该如何建立一个模式，它是可变化的，为你的应用程序的evolves。特别是，这个计划并没有提出解决、确定不同的纹理压缩格式的方案。可One option might be to define your own table that specifies a different integer to each of the different compression formats your application supports (for example, 1 might correspond to ETC1 and 2 is ATITC, and so on).

您可以使用任何你想要的方案，但您应该谨慎考虑您的应用程序的未来版本将需要增加他们的版本代码，并可以接收设备如何更新设备配置更改（例如，由于系统更新）时，或当修改配置为了一个或几个APKs的支持。

Using a Single APK Instead

创建多个APKs发布在Google Play上面是个不正常的步骤。在大多数情况下，你应该大多数发布您的应用程序，是一个单一的APK，我们鼓励你这样做。当你遇到一个情况，在一个单一的APK变得很困难，你应该仔细考虑所有的选择，然后才决定发布多multiple APKs。

首先，只发布一个single APK，遍支持所有设备的几个关键的好处：

- 发布和管理您的应用程序更容易。

只有一个APK的担心在任何特定时间，你不太可能成为混淆APK是什么。你也没有保持跟踪多个版本的代码，每个机构APK-只使用一个APK的，你可以简单地增加与每个发行版本的代码完成。

- 你需要管理只有一个的代码库。

虽然你可以使用一个 library project 多个Android项目之间共享代码，它仍然有可能在每个项目中，你会重现一些代码，这可能会变得难以管理，尤其是在解决Bug。

- 您的应用程序能够适应设备配置变化。

通过创建一个单一的APK，其中包含所有的资源为每个设备配置，应用程序可以适应配置在运行时发生的变化。例如，如果用户docks 或其他手机设备

连接到一个更大的屏幕，还有一个改变，这将调用系统配置的变化，以支持更大的屏幕。如果你有不同的屏幕配置在相同的APK的所有资源，那么你的应用程序将为新的interface加载替代资源和优化用户体验。

- App restore across devices just works.

If a user has enabled data backup on his or her current device and then buys a new device that has a different configuration, then when the user's apps are automatically restored during setup, the user receives your application and it runs using the resources optimized for that device. For example, on a new tablet, the user receives your application and it runs with your tablet-optimized resources. This restore process does not work across different APKs, because each APK can potentially have different permissions that the user has not agreed to, so Google Play may not restore the application at all. (If you use multiple APKs, the user receives either the exact same APK if it's compatible or nothing at all and must manually download your application to get the APK designed for the new device.)

以下各节描述了一些其他的选项，在你决定发布multiple APKs支持不同配置的设备。

支持**multiple GL textures**

为了一个apk支持multiple GL textures，您的应用程序应该查询设备支持的GL纹理格式，然后使用appropriate（合适）的资源，或从Web服务器上下载。例如，为了保持你的APK体积小，可以查询不同的GLtextures格式的设备的支持，应用程序启动时第一次，然后只下载你需要该设备的textures。

为了获得最大的性能和兼容性，只要不影响视觉质量，应用程序应该使用ETC1 textures。然而，因为ETC1不能色度的急剧变化，如 line art 和文本（大多数），图像处理，不支持Alpha，它可能不是最好的纹理格式。

单一的APK，你应该尝试使用ETC1 textures和未压缩的textures，只要合理，当使用ETC1不满足使用的需求时，考虑使用的PVRTC，ATITC，或DXTC。

下面是支持纹理压缩格式，从里面例如查询 `GLSurfaceView.Renderer` 的：

```
public void onSurfaceChanged(GL10 gl, int w, int h) {
    String extensions = gl.glGetString(GL10.GL_EXTENSIONS);
    Log.d("ExampleActivity", extensions);
}
```

这将返回一个字符串，它列出了每个支持的压缩格式

支持多个屏幕

除非你的APK文件超过Google Play50MB大小的限制，支持多个屏幕应该始终做一个单一的APK。由于Android 1.6，Android系统的管理大部分各种屏幕尺寸和密度上所需的工作，以支持你的应用在不同设备下运行。

为了进一步优化您的应用程序，在不同屏幕尺寸和密度上运行，你应该提供**alternative resources**，在不同的分辨率和不同屏幕尺寸的不同布局设计，如位图的**drawable**。

欲了解更多有关如何支持多个屏幕上用一个单一的APK信息，请阅读[Supporting Multiple Screens.](#)。

此外，你应该考虑使用兼容包支持库，使您可以添加您的活动设计的**Fragments**，以便支持在更大的屏幕上运行 如**tablets**。

支持多种API的**levels**

假如你想尽可能多的支持android的不同的平台，你应当使用APIs可以用的最低**version**。比如，你的应用可能要求APIs的**version**高于2.1（API level 7），他将要大约支持超过95%的android设备。（。。。。）

通过使用一个支持库兼容性软件包，你也可以使用一些最新的版本（如Android 3.0）的API，同时还支持Android 1.6的低版本。支持库包括**APIFragments, Loaders, and more**。使用**Fragment API**是特别有价值的，这样可以优化你的用户界面，为大型设备，如**tables**。

另外，如果你想使用一些API，只有在新版本的Android 有的api（那些你没有仍然支持的功能），那么你应该考虑使用reflection。通过使用reflection，你可以检查当前设备是否支持某些API。如果API是不可用，您的应用程序可以正常 disable and hide 功能。

只有当运行一个版本，支持他们使用新的API的另一种方法是检查当前设备的API级别。也就是说，你可以查询的这个SDK_INT，和创建不同的代码路径，取决于设备所支持的API级别。例如：

```
if ( android.os.Build.VERSION.SDK_INT >= 11 ) {  
    // Use APIs supported by API level 11 (Android 3.0) and up  
} else {  
    // Do something different to support older versions  
}
```

来自“[index.php?title=Multiple_APK_Support&oldid=8688](#)”

APK Expansion Files

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

APK Expansion Files

原文地址：<http://docs.eoeandroid.com/guide/google/play/expansion-files.html>

翻译： zhycheng

更新： 2012/9/19

APK的扩展文件

谷歌商城最近要求APK文件小于50M。对于大多数的应用程序，这个空间足够应用程序的代码和资源文件。然而有些应用程序需要更多的空间保存高保真图形，多媒体文件和其他的资源文件。之前，如果你的应用程序超过50M，当用户打开你的应用程序时你要自己主办下载其他的资源文件。主机和服务文件下载是昂贵的，而且用户体验经常比理想中要差。为了使这个过程简单和用户喜欢，谷歌商城允许你为你的APK文件添加两个大的扩展文件。

谷歌商城为您的应用程序管理扩展文件和为设备服务不花你的钱。扩展文件保存到设备的共享存储位置(SD卡或usb挂载分区,也被称为“外部”存储),你的应用程序可以访问它们。在大多数设备,谷歌商城同时下载扩展文件和APK文件,当用户第一次打开你的应用程序它就拥有需要的一切,。然而,在某些情况下,当你的应用程序启动的时候您的应用程序必须从谷歌商城下载文件。

来自 "[index.php?title=APK_Expansion_Files&oldid=11712](#)"



Google Cloud Messaging

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Jennifer

原文链接：<http://developer.android.com/guide/google/gcm/index.html>

Google Cloud Messaging

[2]

谷歌云信息 (GCM) 是一种服务，可以让开发者在Android设备上从服务器端发送数据到Android应用程序。这可能是一个告诉Android应用的短消息：在服务端有新的数据更新（例如，朋友上传了一个电影，或者是一个大于4kb的负载数据（像如即时通讯应用程序可以直接消耗的消息）。GCM服务处理所有在排队的信息，发送给目标设备的目标应用程序。

GCM是完全免费的，不管你需要多大的消息，没有配额限制。

了解更多关于GCM，可以加入[android-gcm group](#)，参阅如下文档：

[开始](#)

阅读此文档了解基于GCM的Android应用开发的基础步奏。

[结构概览](#)

阅读此文档，了解GCM的架构与概念。

[应用教程](#)

阅读此文档完成设置运行GCM demo 应用。

进阶主题

阅读此文档深入理解GCM 关键特性。

迁移

如果你是一个C2DM转变到GCM的开发者，阅读此文档。

GCM还为[客户端](#)和[服务端](#)开发提供了帮助库。

来自“[index.php?title=Google_Cloud_Messaging&oldid=8500](#)”

Getting Started

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： Jennifer

分任务原文链接：<http://developer.android.com/guide/google/gcm/gs.html>

目录

- [1 GCM: Getting Started](#)
 - [1.1 Creating a Google API Project](#)
 - [1.2 Enabling the GCM service:](#)
 - [1.3 Obtaining an API Key](#)
 - [1.4 Install the Helper Libraries](#)
 - [1.5 Writing the Android Application](#)
 - [1.6 编写服务端应用](#)

GCM: Getting Started

此文档描述了如何编写一个Android 应用和服务端逻辑，如何使用GCM帮助库（客户和服务）。

Creating a Google API Project

创建一个Google API工程：

1. 打开Google APIs Console页面。
2. 如果还没有创建一个API工程,本页会提示你做如下事情:

[Gcm-create-api-proj.png](#)



注意：如果已经有工程了，你看到的第一页将是Dashboard页，从那你可以通过打开工程下拉菜单（左上角）**Other**

projects > Create. 创建一个新的工程。

3. 点击创建工程，浏览器会变换到类似：

<https://code.google.com/apis/console/#project:4815162342>

4. 注意#project:的值(本例 4815162342)。是工程ID，会在后面作为GCM传递ID。

Enabling the GCM service:

允许GCM 服务：

1. 在Google APIs Console主页，选择Services。
2. 打开 Google Cloud Messaging开关。
3. 在Service页的Terms中，选择各选项。

Obtaining an API Key

获得一个API key:

1. 在Google APIs Console主页,选择API Access。会看到如下屏幕：

Gcm-api-access.png



2. 点击创建新的Server key，一个服务key或者浏览器key应当可用，服务key的优点是允许添加IP地址白名单，屏幕显示：



3. 点击创建：



注意本例的**APIkey**值 (key将显示在此) ，将在后面用到。

注意：如果需要倒换key，点击生成新的key，新的key会被创建，而老的仍然在24小时内有效。如果想让旧的key立即无效(例如，你觉得被破解了)，点击删除key。

Install the Helper Libraries

要实现下面段落的内容，必须首先安装帮助库（参考客户和服务），从SDK Manager，安装 **Extras > Google Cloud Messaging for Android Library**. 在YOUR_SDK_ROOT/extras/google/下创建一个GCM文件夹，包含子目录：gcm-client, gcm-server, samples/gcm-demo-client, samples/gcm-demo-server, 和 samples/gcm-demo-appengine。

注意：如果在SDK Manager没看到**Extras > Google Cloud Messaging for Android Library**，确认你的版本号是20或更高，确保更新后重启SDK Manager。

Writing the Android Application

本段叙述了如何使用GCM来编写Android应用的步奏。

第一步：复制gcm.jar文件到应用的classpath 要写应用，首先从SDK的gcm-client/dist中复制gcm.jar文件到应用的classpath.

第二步：修改应用的Android manifest

1.GCM要求Android2.2或更新版本，所以，如果应用没有GCM便不能工作，则要添加如下行，xx代表目标SDK最新的版本号：

```
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="xx" />
```

2. 声明并使用custom permission，如此只有此应用程序才能接收GCM messages:

```
<permission android:name="my_app_package.permission.C2D_MESSAGE" android:protectionLevel="signature" />
<uses-permission android:name="my_app_package.permission.C2D_MESSAGE" />
```

此permission必须命名为my_app_package.permission.C2D_MESSAGE (my_app_package是应用程序的包名, 定义在manifest tag中) , 不然不会有效。

注意: 此permission 不是要求你的目标应用要达到4.1或更高 (例如: minSdkVersion 16)

3.添加如下permission:

```
<!-- App receives GCM messages. -->
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<!-- GCM connects to Google Services. -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- GCM requires a Google account. -->
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<!-- Keeps the processor from sleeping when a message is received. -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

4.添加如下broadcast receiver:

```
<receiver android:name="com.google.android.gcm.GCMBroadcastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
        <category android:name="my_app_package" />
    </intent-filter>
</receiver>
```

这个broadcast receiver响应处理GCM传给的2个intent(**com.google.android.c2dm.intent.RECEIVE**和**com.google.android.c2dm.intent.REGISTRATION**)，而且应当定义在manifest(而不是programmatically)，如此这些intents能够被应用程序接收到，即使应用程序没在运行。通过设定**com.google.android.c2dm.permission.SEND** permission, 确保只有GCM system framework发送

的intent才发送给receiver，（普通的应用不能处理拥有此类permission的intent）。注意在分类标签里的**android:name**必须是应用的包名（分类标签不要求应用程序的minSdkVersion 是16或更高）。

5. 添加如下的Intent服务：

```
<service android:name=".GCMIntentService" />
```

像在下一步所示，此intent服务会被GCMBroadcastReceiver (GCM库提供) 调用。必须是**com.google.android.gcm.GCMBaseIntentService**的子类，必须包含一个公共构造器，而且应当被命名为my_app_package.GCMIntentService (除非使用GCMBroadcastReceiver的子类来重写方法来命名服务)。

第三步：编写my_app_package.GCMIntentService类

下一个编写my_app_package.GCMIntentService类，重写如下回调方法（被GCMBroadcastReceiver调用）：

.onRegistered(Context context, String regId): 接收到一个注册过的intent后被调用，作为一个参数传递GCM分配的注册ID给设备/相应应用程序。典型地，应当发送regid给服务端以使它能发送消息给设备。

.onUnregistered(Context context, String regId): 当设备从GCM注销注册后被调用，典型地，应当发送regid给服务端以注销设备。

.onMessage(Context context, Intent intent): 当服务端发送消息给GCM调用，GCM传送给设备。如果消息拥有payload, 它的content可以作为intent的extras来获得。
.onError(Context context, String errorId): 当设备试图要注册或注销，但GCM返回一个错误时调用。典型地，什么都不做除了估计错误（返回errorId）试着解决问题。

onRecoverableError(Context context, String errorId): 当设备试图要注册或注销，但是GCM服务不可获得，GCM库会使用指数备份尝试重复操作，除非此方法被重写且返回false。此方法是可选方法，只有当想显示消息给用户或取消重复尝试才应当被重写。

注意：上面的方法运行在intent service的线程，因此任意进行网络调用，不会阻塞UI线程。

第四步：编写应用的主activity

在应用的主activity添加如下重要的成员部分：

```
import com.google.android.gcm.GCMRegistrar;
```

在onCreate()方法中添加：

```
GCMRegistrar.checkDevice(this);
GCMRegistrar.checkManifest(this);
final String regId = GCMRegistrar.getRegistrationId(this);
if (regId.equals("")) {
    GCMRegistrar.register(this, SENDER_ID);
} else {
    Log.v(TAG, "Already registered");
}
```

checkDevice()方法验证支持GCM的设备，如果不支持则抛出异常（例如，如果一个不包含Google APIs的模拟器），类似地，checkManifest()验证应用的menifest是否包含了在编写Android应用所要求的条件（此方法只有当开发应用时才有必要；一旦准备好发布应用，可以删掉此方法）。

一旦完成合理的验证，设备调用GCMRegistrar.register()来注册设备，传递注册时获得的SENDER_ID。但由于当注册intent到来后，GCMRegistrar单独跟踪注册ID，可以先调用GCMRegistrar.getRegistrationId()来确认设备是否已经注册。

注意：有可能设备已经成功注册在GCM，但发送注册ID给服务端失败，这种情况下应该尝试再次发送。参阅[高级进阶主题](#)，获得更多关于如何处理此状况的细节知识。

一旦完成合理的验证，设备调用GCMRegistrar.register()来注册设备，传递注册时获得的SENDER_ID。但由于当注

册intent到来后， GCMRegistrar单独跟踪注册ID，可以先调用CMRegistrar.getRegistrationId()来确认设备是否已经注册。

编写服务端应用

编写服务端应用：

1. 从 SDK的gcm-server/dist路径下复制gcm-server.jar文件到你的服务器classpath
2. 创建一个servlet（或其他服务端机器）供Android应用使用，来传送从GCM获得的注册ID。这个应用可能需要发送其他信息，例如用户的email或者用户名，让服务器可以把注册ID与用户设备匹配。
3. 同样，创建一个servlet来注销注册ID。
4. 当服务器需要发送消息给注册ID，可以使用GCM库里的com.google.android.gcm.server.Sender helper类，例如：

```
import com.google.android.gcm.server.*;  
  
Sender sender = new Sender(myApiKey);  
Message message = new Message.Builder().build();  
MulticastResult result = sender.send(message, devices, 5);
```

上段代码做了如下事情：

使用工程的API key创建一个sender对象。

使用给定的注册ID创建一个消息(消息的builder也有方法来发送所有消息参数，像失效的key和负载数据)。

发送消息，最大尝试5次（防止GCM服务器不可用），保存响应结果。

现在有必要来解析结果，采取如下相应的行动：

如果创建了消息但返回的结果是典型的注册ID，则用此典型注册ID代替当前的注册ID。

如果返回错误结果是NotRegistered，需要移除那个注册ID，因为此应用没有在设备中安装。

如下是处理这两种情况的代码段：

```
if (result.getMessageId() != null) {
    String canonicalRegId = result.getCanonicalRegistrationId();
    if (canonicalRegId != null) {
        // same device has more than one registration ID: update database
    }
} else {
    String error = result.getErrorCodeName();
    if (error.equals(Constants.ERROR_NOT_REGISTERED)) {
        // application has been removed from device - unregister database
    }
}
```

来自“[index.php?title=Getting_Started&oldid=8514](#)”

GCM Architectural Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

GCM结构概述-GCM Architectural Overview

谷歌云消息服务（GCM）是免费提供给用户的，它帮助你把服务端的数据推送到客户端。它可能是一个轻量级的消息，通知有新的数据上传的服务器（例如，朋友上传了一个电影），或者可能是一个包含了4KB的有效载荷数据的消息（所以即时消息应用可以直接消耗消息）。GCM服务处理排队的消息，并传递到目标的Android应用。想知道如何使用GCM，请看[Getting Started](#)

目录

[\[隐藏\]](#)

[1 介绍-Introduction](#)

[2 结构概述-Architectural Overview](#)

- [2.1 生命周期-Lifecycle Flow](#)
- [2.2 用户能看见什么-What Does the User See?](#)

[3 构建使用GCM的应用程序-Writing Android Applications that Use GCM](#)

快速预览

- 一个状态通知允许应用程序通知用户一个事件但不扰乱他们的当前活动
- 你可以把一个intent和通知绑定在一起，这样当用户点击通知选项时系统就可以进行初始化

本文内容

[基础-The Basics](#)

[响应通知-Responding to Notifications](#)

[管理通知-Managing your Notifications](#)

[创建通知-Creating a Notification](#)

[更新通知-Updating the notification](#)

[添加声音-Adding a sound](#)

[添加振动-Adding vibration](#)

[添加闪灯-Adding flashing lights](#)

[更多特性-More features](#)

[自定义通知的布局-Creating a Custom Notification Layout](#)

3.1 创建清单-Creating the Manifest

介绍-Introduction

下面是谷歌的云消息服务的主要特点：

- 它允许第三方应用服务器发送邮件到安卓应用。
- 关于消息的交付和顺序， GCM没有作出保证。
- Android设备上的应用并不需要一直运行，准备接收消息。当消息到达时，系统将通过Intent broadcast（意图广播）唤醒应用，只要应用程序设置适当的广播接收机和权限。
- 它不提供任何内置的用户界面或其他对消息数据的处理方法。 GCM的只是简单地把原始数据传递给Android应用程序，而程序会负责如何处理消息数据。例如，应用程序可能会发布通知，显示自定义的用户界面，或默默地同步数据。
- 它需要可以运行Android2.2或者更高版本而且要装有谷歌商店应用的设备，或者可以运行Android2.2 API的仿真器。然而，你并没有被限制通过谷歌商店来发布程序。
- 它使用现有的谷歌服务的连接。对于前3.0设备，需要用户在他们的移动设备上设置谷歌帐户。运行Android4.0.4或更高版本的设备是不需要谷歌账户的。

结构概述-Architectural Overview

这章节概述了GCM是如何工作的。

此表汇总的GCM涉及的关键术语和概念。它分为这些类别：

- 组件-在GCM中发挥作用的物理实体
- 凭证-用于GCM不同阶段的ID和令牌，以确保各方已经通过验证和消息将被送往正确的地方。

组件

移动设备	运行使用GCM的Android应用的设备。它必须是2.2的Android设备并且已经安装了谷歌商店，而且如果版本低于Android4.0.4，必须至少要有一个谷歌账户登录。另外，作为测试，你可以使用运行android2.2的仿真器。
第三方应用服务	开发人员用以作为实现GCM一部分的应用服务器。第三方服务器通过GCM服务器给设备上的应用发送数据。
GCM服务器	谷歌服务器从第三方服务器上获取数据，并把他们发送到移动设备上

凭证

Sender ID	从API控制台获取的项目ID，在 Getting Started 里有描述。Sender ID被用在 registration process 中来确认安卓应用是否已经被允许发消息给设备。
Application ID	安卓程序注册Application ID，用来获得消息的。安卓程序是通过 manifest 中的包名来区分，确认的。这确保该消息是针对正确的Android应用程序。
Registration ID	由GCM服务器发送给安卓应用，允许应用接收消息。一旦安卓应用拥有了registered ID,就把它发送给第三方应用服务器，服务器用它们来确认哪些设备已经注册了，正准备接收消息。换句话说，registered ID 被绑定在运行在特殊设备上的特殊应用上。
谷歌用户帐号	为了GCM的工作，移动设备必须至少包含一个谷歌帐户，如果设备运行比4.0.4的Android版本较低。
消息发送者验证令牌	保存在第三方应用服务器上的API key，允许服务器访问谷歌服务（Google services）。API key 存在于消息发送请求的头部信息里。

生命周期-Lifecycle Flow

这里是云到设备的消息中所涉及的主要过程: [Enabling GCM](#). 运行在移动设备上的应用，注册用来接收消息

[Sending a message](#). 第三方应用服务器给设备发送消息。

[Receiving a message](#). 应用从GCM服务器上接收消息。

启动google云消息服务-Enabling GCM

下面是当移动设备上的应用去注册来接收消息时发生的事件的序列。

1 第一次应用要使用消息服务时，他会给GCM发送一个注册Intent。

`Intent com.google.android.c2dm.intent.REGISTER`包括发送者ID和Android应用程序的ID。

注意：因为应用第一次运行的时候没有呼叫生命周期里的方法，所以来注册intent被传递给`onCreate()`方法，但这只限于应用还没有注册的情况下。

2 如果注册成功，GCM会广播一

个`com.google.android.c2dm.intent.REGISTER Intent`，这回给应用一个注册ID。

应用会在以后用到这个ID（例如会在`onCreate()`方法里校验是否已经注册）。注意，谷歌可能会定期刷新的注册ID，所以你在设计应用的时候，要认识到`com.google.android.c2dm.intent.REGISTER`可能会被多次调用。您的Android应用程序需要能够作出相应的反应。

3 要完成注册，Android应用程序把注册ID发送到应用程序服务器。服务器要把注册ID存储在数据库中。

注册ID一直持续到Android的应用程序显式注销它，或者Google对它进行刷新。

注意：当用户卸载应用程序，它不是自动在GCM上被注销的。时机是在当GCM发消息给设备并且设备反馈应用已经被删除了。在这是，你的服务器设备标记为未注册的。（服务器会收到一个[NotRegistered](#)的错误）

要注意的是注册ID要想完全在GCM上被删除要花费几分钟的时间。所以这段时间内，第三方服务器发送一个消息，他会获得一个有效的消息ID，即

使消息并没有被发送到设备上。

发送消息-Sending a Message

对于应用程序服务器将消息发送到一个Android应用程序，下面的事情一定要到位：

- 应用必须要有一个注册ID，这样允许它在一个特定的设备上来接受消息
- 第三方服务器已经存储了注册ID
- API key。这是开发者必须已经在应用服务器上为应用准备好的。（有关的讨论，请参看[Role of the 3rd-party Application Server](#)）现在已经准备好给设备发消息了。

下面列出了当应用服务器发消息时会发生的事件序列：

1 应用服务器给GCM发消息

2 Google会给消息排序并存储他们，当设备不在线的时候。

3 当设备在线的时候，Google会把消息发送给他们。

4 在设备上，系统会使用合适的权限通过Intent广播把消息广播给具体的应用。消息会唤醒那个应该接收消息的应用，所以应用不用一直在准备着接受消息。

5 应用会处理消息。如果应用在做不一般的处理，你可能需要抓取[PowerManager.WakeLock](#)并且在服务上做一些处理。Android应用程序可以注销的GCM如果它不再想接收邮件。

接收消息-Receiving a Message

下面是设备上的应用接收消息时触发的事件序列：

1 系统接收送到的消息并从消息中提取键值对。

2 系统通过com.google.android.c2dm.intent.RECEIVE把键值对信息发送给应用。

3 应用通过`com.google.android.c2dm.intent.RECEIVE`提取数据，并加以处理。

用户能看见什么-What Does the User See?

当移动设备用户安装包括GCM的Android应用程序，Google Play Store 会通知他们（GCM）-应用包含GCM。他们必须批准应用有这些（GCM的）特性的使用权。

构建使用**GCM**的应用程序-Writing Android Applications that Use GCM

to do

创建清单-Creating the Manifest

每个android应用都必须有`AndroidManifest.xml`存在于它的根目录。这个文件里面包含很多必要信息。（更多讨论，参考[Android Developers Guide](#)）为了使用GCM,这个文件必须包括如下：

- 使用`com.google.android.c2dm.permission.RECEIVE` 的权限，这样应用就能注册和接收信息了。
- 使用`android.permission.INTERNET` 的权限,这样应用可以发送注册ID到第三方应用服务器了。
- 使用`android.permission.GET_ACCOUNTS`的权限,因为GCM需要一个google帐号（这一点在低于Android 4.0.4版本中是必须的）
- 使用`android.permission.GET_ACCOUNTS`的权限,这样当消息到来时，应用可以被唤醒。
- 使用`applicationPackage + ".permission.C2D_MESSAGE"` 的权限,这样可以防止其他应用注册和使用本应用的消息。权限名称必须完全符合这个模式，否则将Android应用程序将不会收到的消息
- `com.google.android.c2dm.intent.RECEIVE`

和`com.google.android.c2dm.intent.REGISTRATION`的接收器，而且类别要设置为`applicationPackage`。接收器需要`com.google.android.c2dm.SEND`权限，以便只GCM框架可以将消息发送到它。请注意，注册和接收的消息是通过[Intens](#)实现的。

- intent服务用来处理由广播接收器收到的intent
- 如果GCM的特性对应用起着至关重要的作用，请确定设置`* android:minSdkVersion="8"`。这保证应用不会被安装在不适合的设备上。

下面是具体的设置

```
<manifest package="com.example.gcm" ...>
    <uses-sdk android:minSdkVersion="8"
    android:targetSdkVersion="16" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <permission android:name="com.example.gcm.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <uses-permission android:name="com.example.gcm.permission.C2D_MESSAGE" />
    <application ...>
        <receiver
            android:name=".MyBroadcastReceiver"
            android:permission="com.google.android.c2dm.permission.SEND" >
            <intent-filter>
                <action
                    android:name="com.google.android.c2dm.intent.RECEIVE" />
                <action
                    android:name="com.google.android.c2dm.intent.REGISTRATION" />
                <category android:name="com.example.gcm" />
            </intent-filter>
        </receiver>
        <service android:name=".MyIntentService" />
    </application>
</manifest>
```

来自“[index.php?title=GCM_Architectural_Overview&oldid=13300](#)”



GCM Demo Application

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：玄月冰灵

主任务原文链

接：<http://developer.android.com/guide/google/gcm/demo.html>

目录

- [1 GCM样例应用](#)
 - [1.1 要求](#)
 - [1.2 Setting Up GCM](#)
 - [1.3 设置服务器](#)
 - [1.3.1 使用标准网页服务器](#)
 - [1.3.2 使用JAVA的App Engine](#)
 - [1.4 设置设备](#)

GCM样例应用

谷歌云推送（GCM）样例展示了如何在你的安卓应用中使用谷歌云推送框架。向导指引你通过设置来运行样例。

这个样例包含以下部分：

- 一个网页服务器包含一个你可以发送消息的页面。
- 一个安卓应用接收和显示这些消息。

此处的API参考文档是基于样例的帮助库：

- 客户端参考
- 服务端参考

以下部分描述了如何从SDK管理器下载样例代码和帮助库。样例代码和帮助库在开源站点同样有效。

要求

网页服务器：

- Ant1.8（可能早期版本也能运行，但不保证）。
- 以下其中之一：

一个兼容Servlets API版本2.5的运行中的网页服务器，例如Tomcat6或Jetty, 或Java App Engine SDK 版本1.6或更新。

- 使用GCM需要一个已注册的谷歌账号。
- 账号的API密钥。

安卓应用：

- 使用Google API且运行安卓2.2的模拟器（或设备）。
- 使用GCM需要已注册账号的谷歌API工程ID。

Setting Up GCM

Before proceeding with the server and client setup, it's necessary to register a Google account with the Google API Console, enable Google Cloud Messaging in GCM, and obtain an API key from the Google API Console.

For instructions on how to set up GCM, see Getting Started.

设置服务器

本节描述的不同选项设置了一个服务器。

使用标准网页服务器

使用标准的、servlet-compliant 网页服务器来设置服务器。1. 从SDK管理器，安装 Extras > Google Cloud Messaging for Android Library。这会在YOUR_SDK_ROOT/extras/google/ 目录下创建一个包含这些子目录的gcm目录：gcm-client, gcm-server, samples/gcm-demo-client, samples/gcm-demo-server, 和 samples/gcm-demo-appengine。

注释：如果你在SDK管理器中没有看到 Extras > Google Cloud Messaging for Android Library，确保你已经运行了版本20或更高版本。确保在更新后重启SDK管理器。

2. 在文本编辑器中，编辑

samples/gcm-demo-server/WebContent/WEB-INF/classes/api.key

并用获取的API密钥代替存在的文本。3. 在命令行窗口，进入samples/gcm-demo-server 目录。4. 运行ant war 生成服务器的WAR文件：

```
$ ant war
Buildfile:build.xml

init:
  [mkdir] Created dir: build/classes
  [mkdir] Created dir: dist

compile:
  [javac] Compiling 6 source files to build/classes

war:
  [war] Building war: dist/gcm-demo.war

BUILD SUCCESSFUL
Total time: 0 seconds
```

5. 在你运行的服务器上部署dist/gcm-demo.war。例如，如果你使用Jetty，复制demo.war 到安装Jetty的webapps 目录。 6. 在浏览器中打开服务器主页。URL取决于你使用的服务器和你机器的IP地址。但它将会类似这样<http://192.168.1.10:8080/gcm-demo/home>, gcm-demo 是应用的环境，/home 是主servlet的路径。

注释：你可以获得IP在Linux或MacOS上运行ifconfig ，或在Windows上运行ipconfig 。



你的服务器现在准备好了。

使用JAVA的App Engine

使用标准的Java App Engine设置服务器：

1. 从SDK管理器中，安装Extras > Google Cloud Messaging for Android Library。这将在 YOUR_SDK_ROOT/extras/google/ 目录下创建包含这些子目录的gcm 目录： gcm-client, gcm-server, samples/gcm-demo-client, samples/gcm-demo-server, and samples/gcm-demo-appengine. 2. 在文本编辑器中，编辑： samples/gcm-demo-appengine/src/com/google/android/gcm/demo/server/ApiKeyInitializer.java

并用获取的API密钥代替存在的文本。

注释：在那个类中设置的API密钥值将在App Engine中一次性创建持久的实体。如果你部署应用，你可以使用App Engine的Datastore Viewer 在之后修改它。

3. 在命令行窗口，进入samples/gcm-demo-appengine 目录。 4. 开始用ant runserver开发App Engine服务器，使用-Dsdk.dir 指向本地的App Engine SDK，并用-Dserver.host 设置你服务器的主机名和IP地址。

```
$ ant -Dsdk.dir=/opt/google/appengine-java-sdk runserver -Dserver.host=192.168.1.10
Buildfile: gcm-demo-appengine/build.xml

init:
    [mkdir] Created dir: gcm-demo-appengine/dist

copyjars:

compile:

datanucleusenhance:
    [enhance] DataNucleus Enhancer (version 1.1.4) : Enhancement of
classes
    [enhance] DataNucleus Enhancer completed with success for 0
classes. Timings : input=28 ms, enhance=0 ms, total=28 ms. Consult
the log for full details
    [enhance] DataNucleus Enhancer completed and no classes were
enhanced. Consult the log for full details

runserver:
    [java] Jun 15, 2012 8:46:06 PM
com.google.apphosting.utils.jetty.JettyLogger info
    [java] INFO: Logging to JettyLogger(null) via
com.google.apphosting.utils.jetty.JettyLogger
    [java] Jun 15, 2012 8:46:06 PM
com.google.apphosting.utils.config.AppEngineWebXmlReader
readAppEngineWebXml
    [java] INFO: Successfully processed gcm-demo-
appengine/WebContent/WEB-INF/appengine-web.xml
    [java] Jun 15, 2012 8:46:06 PM
com.google.apphosting.utils.config.AbstractConfigXmlReader
readConfigXml
    [java] INFO: Successfully processed gcm-demo-
appengine/WebContent/WEB-INF/web.xml
    [java] Jun 15, 2012 8:46:09 PM
com.google.android.gcm.demo.server.ApiKeyInitializer
contextInitialized
    [java] SEVERE: Created fake key. Please go to App Engine admin
console, change its value to your API Key (the entity type is
'Settings' and its field to be changed is 'ApiKey'), then restart
the server!
    [java] Jun 15, 2012 8:46:09 PM
com.google.appengine.tools.development.DevAppServerImpl start
    [java] INFO: The server is running at http://192.168.1.10:8080/
    [java] Jun 15, 2012 8:46:09 PM
com.google.appengine.tools.development.DevAppServerImpl start
    [java] INFO: The admin console is running at
http://192.168.1.10:8080/_ah/admin
```

5. 在浏览器中打开服务器主页。URL取决于你使用的服务器和你机器的ID地址，但它将会类似这样 <http://192.168.1.10:8080/home>, /home 是主Servlet的路径。

注释：你可以通过在Linux或MacOS中运行 ifconfig 或者在Windows中运行ipconfig来获取IP。



你的服务器现在准备好了。

设置设备

设置设备：

1. 从SDK管理器中，安装Extras > Google Cloud Messaging for Android Library。这将会在包含这些子目录的YOUR_SDK_ROOT/extras/google 目录下创建一个gcm目录： gcm-client, gcm-server, samples/gcm-demo-client, samples/gcm-demo-server, and samples/gcm-demo-appengine。
2. 使用文本编辑器，打开 samples/gcm-demo-client/src/com/google/android/gcm/demo/app/CommonUtilities.java 并为SENDER_ID 和 SERVER_URL 常量设置特征值。例如：

```
static final String SERVER_URL = "http://192.168.1.10:8080/gcm-demo";
static final String SENDER_ID = "4815162342";
```

注意 SERVER_URL 是服务器的URL，应用环境（或只是服务器，如果你使用App Engine），它并不包含斜杠 (/) 。同时注意SENDER_ID 是你在服务器设置步骤中获取的谷歌API项目ID。

3. 在命令行窗口，进入gcm-demo-client 目录。
4. 使用SDK的android 工具生成ant 构造文件：

```
$ android update project --name GCMDemo -p . --target android-16
Updated project.properties
Updated local.properties
Updated file ./build.xml
Updated file ./proguard-project.txt
file:///D:/guide/GCM_Demo_Application[2015/9/23 19:21:21]
```

如果此命令因 android-16 无法识别而失败，尝试不同的目标（至少android-15）

5. 使用ant 创建应用APK文件：

```
$ ant clean debug
Buildfile: build.xml

...
-do-debug:
[zipalign] Running zip align on final apk...
[echo] Debug Package: bin/GCMDemo-debug.apk
[propertyfile] Creating new property file: bin/build.prop
[propertyfile] Updating property file: bin/build.prop
[propertyfile] Updating property file: bin/build.prop
[propertyfile] Updating property file: bin/build.prop

-post-build:

debug:

BUILD SUCCESSFUL
Total time: 3 seconds
```

6. 开启安卓模拟器：

```
$ emulator -avd my_avd
```

这个例子呈现了命名为my_avd 的AVD（安卓虚拟设备）用安卓2.2和谷歌API级别8的预先设置。需要更多关于如何运行一个安卓模拟器的信息，查看安卓开发向导中的管理虚拟设备。

7. 确认谷歌账号被添加入模拟器。尤其不能是其他账号（例如senderId）。如果模拟器运行安卓4.0.4或更新的，这步是可选的，这个版本GCM不要求账号。

8. 在模拟器中安装应用：

```
$ ant installd
Buildfile: gcm-demo-client/build.xml

-set-mode-check:
```

- set - debug - files :

install:

```
[echo] Installing gcm-demo-client/bin/GCMDemo-debug.apk onto  
default emulator or device...  
[exec] 1719 KB/s (47158 bytes in 0.026s)  
[exec] pkg: /data/local/tmp/GCMDemo-debug.apk  
[exec] Success
```

installd:

BUILD SUCCESSFUL
Total time: 3 seconds

9. 在模拟器中，启动GCM样例应用。初始化屏幕会像这样：



注释：发生了什么？当设备从GCM收到一个注册返回intent，它联系服务器注册它自己，使用注册的servlet从GCM返回通过的注册ID，服务器保存注册ID并用它向手机发送消息。

10. 现在，返回你的浏览器刷新页面。它将会显示有一个设备已注册。



11. 点击发送消息。浏览器会显示：



并且在你的模拟器：

注释：发生了什么？当你点击按钮，网页服务器发送了一条消息到你设备的GCM地址（更特别的是，在注册步骤中GCB返回了注册ID）。设备受到消息在主activity中显示。它将发布一条系统通知，即使样例程序没有运行用户也会被通知到。

来自“[index.php?title=GCM_Demo_Application&oldid=8518](#)”

GCM Advanced Topics

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

原文地址：<http://developer.android.com/guide/google/gcm/adv.html>

编辑者： 天娘

完成时间： 2012.8.7

目录

[[隐藏](#)]

1 GCM高级主题

- [1.1 消息的生命周期](#)
- [1.2 节流](#)
- [1.3 保持同步注册状态](#)
- [1.4 使用指数回退自动重试](#)
- [1.5 注销是怎么工作的](#)
- [1.6 发送同步VS有效载荷的消息](#)
- [1.7 设置消息的截至日期](#)
- [1.8 从多个发送人中接受消息](#)

GCM高级主题

这篇文章涵盖GCM的高级主题

消息的生命周期

当第三方服务器发送一个消息给GCM，并接受一个消息返回ID，这并不意味着这个消息已经交付给设备。相反，这仅仅意味着这个消息已经接受交付。在消息接受交付的时候发生什么事取决于很多因素。

最好的情况就是如果设备连接到GCM，屏幕是亮着的，没有任何节流的限制，看（节流），这个消息会正确地交付。

如果这个设备已经连接了但是是闲置的，这个消息仍然会投递成功除非这个`delay_while_idle`标志设置为真。否则，它会被存储在 GCM服务器中直到设备被唤醒。另外，这个`collapse_key`标志也有任务：如果已经有一个消息有相同钥匙（注册ID）存储起来等待交付，旧的消息会被新的消息所取代掉，然而，如果这个`collapse key` 没有设置，两个消息都会存储起来等待以后的交付。

注意：这里有一个关于多少消息能存储而没有崩溃的限制。这个上限一般是100。如果达到了这个上限，所有的消息都会失效。然后当设备在后台运行，它会收到一个特殊的消息指出当前已经达到了上限。应用程序能通过请求完全地同步，来正确地，典型地处理这种情况。

如果设备没有连接到GCM，消息会存储起来直到设备和GCM的连接建立起来（再一次地考虑`collapse key` 原则）。当连接已经建立后，GCM就会交付所有待定的消息给设备，不管`delay_while_idle`标志。如果设备从来没有连接（例如，如果它被格式化了），消息最终会在GCM的存储仓库中超时，丢弃。默认的超时时间是4个星期，除非这个`time_to_live`标志设置了。

注意：当你设置`time_to_live`标志时，你必须也要设置`collapse_key`。否则，这个消息就被当做一次恶意的请求而被拒绝。

最后，当GCM试图交付消息给设备并且应用程序已经被卸载，GCM会立刻丢弃这个消息并且使这个注册ID失效。将来试图发送消息给那个设备会得到一个`NotRegistered`错误。详情查看[How Unregistration Works](#)。

虽然追踪每个消息个体的踪迹是不可能的，但是谷歌API的控制台能统计消息发送给设备，消息崩溃和消息等待交付。

节流

为了阻止滥用（例如发送大量的消息给设备）并且优化网络的有效利用和电池的寿命，GCM工具实现节流消息使用一种象征筒模式。消息被节流在每个应用 程序和每个`collapse key`(包含没有崩溃的消息)。每个应用程序

的collapse key被授予一些最初的token，并且新的token会阶段性地授予。每个token只会对一个发送给设备的消息有效，如果一个应用程序 collapse key耗尽了提供给它的所有有用(token，新的消息会缓冲在待定队列中直到新的token在授权期的时候成为可用。这样在授权的间隔时期压制可能会增加消息交付给应用程序，在很短的时间内发送大量的消息提供消息的延迟，消息在应用程序的collapse key等待队列中可能在下次授权之前交付。

保持同步注册状态

无论何时应用程序接收到包含registration_id的com.google.android.c2dm.intent.REGISTRATION意图，它会保存这个ID以便未来使用，传递给第三方服务器来完成注册，并且追踪服务器是否完成了注册。如果服务器注册失败，它会再次尝试或者注销。

这里有两个细节需要特别小心：

- 应用程序更新
- 备份和还原

当一个应用程序更新时，它应该使其现有注册ID失效，因为它不能保证现有ID能在新版本下工作。因为在程序更新的时候没有调用生命周期方法，所以可以在注册ID存储的时候存储当前应用程序的ID来实现最佳的确认方法。然后当程序开始时，比较当前的程序版本和存储的程序版本是否相同。如果相同，那么就清空存储的注册ID，重新开始注册程序。

相同地，你不应该在应用程序备份的时候保存这个注册ID，因为当程序再次还原的时候这个注册ID可能会失效，这种情况可能会使应用程序在一种无效的状态（那就是，应用程序认为已经注册了，服务器和GCM就不再存储这个注册ID了，这样的话应用程序可能不会得到任何消息）。

规范标识

在服务器方面认为只要应用程序表现正常，每件事都会正常工作。然而，如果在程序中有错误触发同一设备的多个注册，这个可能很难使状态保持一致，你可能会结束这些的信息。

GCM提供了一个名为“规范注册的ID”来轻松恢复这些状态。一个规范的注

册ID被定义在那些请求注册应用程序的ID的最后。这就是规范注册ID,服务器在发送消息应该使用。

如果在之后你试图发送一条消息使用不同的注册ID，GCM会和平常一样处理这个请求，但是它会包含权威注册ID在registration_id响应里面。确保在服务器中存储的注册ID和规范注册ID替换成功，因为最后你停止工作的时候要用到这个规范注册ID。

使用指数回退自动重试

当应用程序接受到一个外部错误被设定为SERVICE_NOT_AVAILABLE的com.google.android.c2dm.intent.REGISTRATION的意图。它会重试失败的操作（注册或是注销）。

最简单的案例，如果你的程序仅仅调用register和GCM，这并不是一个程序的最基本的部分，应用程序能够简单地忽略错误，而在下一次程序开始时再次尝试注册。另外，它可能会使用指数回退重试先前的操作。在指数回退中，每次存在故障，它再次尝试花费的时间可能是先前操作的两倍。如果注册（或是注销）操作是同步的，那么它会被在一个简单的循环中重试。然而，一旦它是异步的，最佳的方法是安排一个即将发生的intent来重试这个操作。下面的步骤描述怎么使用上面的来实现MyIntentService例子：

1. 创建一个随机的token来验证刚开始的重试intent:

```
private static final String TOKEN =
    Long.toHexString(new Random().nextLong())
```

2. 修改handleRegistration()方法以便在适当的时候创建一个即将发生的intent:

```
...
if (error != null) {
    if ("SERVICE_NOT_AVAILABLE".equals(error)) {
        long backoffTimeMs = // get back-off time from shared
preference
        long nextAttempt = SystemClock.elapsedRealtime() +
backoffTimeMs;
        Intent retryIntent = new
Intent("com.example.gcm.intent.RETRY");
        retryIntent.putExtra("token", TOKEN);
        PendingIntent retryPendingIntent =

```

```

        PendingIntent.getBroadcast(context, 0, retryIntent,
0);
        AlarmManager am = (AlarmManager)
            context.getSystemService(Context.ALARM_SERVICE);
        am.set(AlarmManager.ELAPSED_REALTIME, nextAttempt,
retryPendingIntent);
        backoffTimeMs *= 2; // Next retry should wait longer.
        // update back-off time on shared preferences
    } else {
        // Unrecoverable error, log it
        Log.i(TAG, "Received error: " + error);
    }
...

```

这个回退时间存储在shared preference中，这样就确保了能在多个activity运行时跨越。这个intent的名字是什么没有关系，只要相同intent使用下面的步骤。 3.修改onHandleIntent()方法增加一个else if语句为重试的intent:

```

...
} else if (action.equals("com.example.gcm.intent.RETRY")) {
    String token = intent.getStringExtra("token");
    // make sure intent was generated by this class, not by
a malicious app
    if (TOKEN.equals(token)) {
        String registrationId = // get from shared
properties
        if (registrationId != null) {
            // last operation was attempt to unregister; send
UNREGISTER intent again
        } else {
            // last operation was attempt to register; send
REGISTER intent again
        }
    }
...

```

4.创建一个MyReceiver的实例在你的activity中:

```

private final MyBroadcastReceiver mRetryReceiver = new
MyBroadcastReceiver();

```

5.在activity的onCreate()方法中，注册一个实例来接受com.example.gcm.intent.RETRY意图:

```

...
IntentFilter filter = new
IntentFilter("com.example.gcm.intent.RETRY");
filter.addCategory(getPackageName());
registerReceiver(mRetryReceiver, filter);
...

```

注意：你必须动态地创建一个**broadcast receiver**实例，因为定义在**manifest**中只能接受有**com.google.android.c2dm.permission.SEND**权限的**intent**。这个**com.google.android.c2dm.permission.SEND**权限是系统的权限，它不能授予给一般的应用程序。

6. 在activity中的`onDestroy()`方法中，注销这个**broadcast receiver**：

```
unregisterReceiver ( mRetryReceiver );
```

注销是怎么工作的

这里有两种方法从GCM中注销一个设备：手动地和自动地。

一个Android应用程序能够手动地注销它本身通过发布一个**com.google.android.c2dm.intent.UNREGISTER**的**intent**，这个**intent**很有用，它可以在退出系统时注销，在进入系统时注册。查看更多的讨论**Architectural Overview**。这里列出了当一个应用程序注销它自己的时候发生的顺序事件：

1. 应用程序发布一个**com.google.android.c2dm.intent.UNREGISTER**的意图，传送这个注册ID(应用程序应该保存它的注册ID当它接收到正确的**com.google.android.c2dm.intent.REGISTRATION**意图)。
2. 当GCM服务器完成了注销，它会发送一个有**unregistered**外部设置的**com.google.android.c2dm.intent.REGISTRATION**意图。
3. 然后应用程序必须连接第三方服务器以便它能移除注册ID。
4. 应用程序应该清空它的注册ID。

一个应用程序能够自动地注销在它从设备中卸载之后。然而，这个过程可能不会立刻发生，因为Android没有提供卸载的回调函数。下面的情节介绍自动注销的：

1. 在用户卸载程序的最后。

2. 第三方服务器发送消息给GCM服务器。

3. GCM服务器发送消息给设备。

4. GCM客户端接受消息并且查询包管理，如果没有找到包，返回“包没有找到”错误。

5. GCM客户端通知GCM服务器应用程序已经卸载。

6. GCM服务器标记要删除的注册ID

7. 第三方服务器发送消息给GCM。

8. GCM返回一个NotRegistered错误信息给第三方服务器。

9. 第三方服务器删除注册ID。

注意这可能会花费一些时间来完全地从GCM中除去注册ID。这样可能在消息发送到第7步之前会得到一个有效的消息ID作为响应。即使消息没有传递给设备。最终，这个注册ID会移除同时服务器会得到一个NotRegistered错误，没有任何更进一步的行动要求第三方服务器（当一个应用程序开发和测试，这些情节会经常发生）。

发送同步**VS**有效载荷的消息

在GCM中发送的每个消息都有以下的特征：

- 它有一个4096字节的有效载荷限制。
- 一般情况下，它会存储在GCM中4个星期。

但是去除这些相同点，这些消息能表现得非常不同取决于一些特殊的设置。在消息中有个很大的差别就是它们是collapsed(每个新的消息会取代先前的消息)或者不是collapsed(每个独立的消息已经传递)。每个消息在GCM中发送既不是一个同步发送的(collapsible)消息，也不是一个“message with payload”(non-collapsible消息)。这些观点在接下来的部分会有更多的描述。

同步发送消息

一个告诉一个手机应用程序从服务器同步数据的同步发送（**collapsible**）消息经常是一件很高兴的事。例如，假如你有一个邮件应用程序，当用户接受一封新的邮件在服务器中，服务器会通知邮件应用程序有新邮件。这就说明了应用程序同步服务器来接受新邮件。随着新邮件的累积，在应用程序有机会同步之前，服务器可能会多次发送消息给应用程序。但是如果用户已经接受了25封新邮件，这就没有必要继续发送每个“新邮件”的消息，一个就够了。另一个例子是一个关于娱乐的应用程序中更新用户的最新的得分，只有最近的消息是有用的，因此有必要来使每个新消息取代先前的消息。

上面的邮件和娱乐应用程序是你有可能使用GCM `collapse_key`参数的例子。当设备联机的时候一个**collapse key**是一个任意的字符串习惯于倒塌一组消息，因此只有最近的消息能发送给客户端。例如，“新邮件”，“有效的更新”，等等。

GCM允许最多4个不同的**collapse key**在GCM中的服务器中使用，在任何给定的时间。换句话说，GCM服务器能同时存储4中不同的同步发送消息，每个消息都有一个不同的**collapse key**。如果你超过了这个数字GCM只会给你4个**collapse key**，和一些没有授权的。

有效载荷的消息

和同步发送消息不同，每个“有效载荷的消息”（**non-collapsible** 消息）是传递的。这个载荷的消息最多包含4kb。例如，这里是观众正在讨论一项体育赛事在IM应用中的JSON格式的消息：

```
{
  "registration_id" : "APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...",
  "data" : {
    "Nick" : "Mario",
    "Text" : "great match!",
    "Room" : "PortugalVSDenmark",
  },
}
```

一个“有效载荷的消息”不只是“ping”到手机应用程序与服务器以获取数据。以上面的IM应用程序为例，你如果想要发送每个消息，因为每个消息包含

不同的内容。为了构建一个non-collapsible 消息，你只需要把collapsed_key参数除去即可。这样GCM将会发送单独地每条消息。注意发送的顺序是没有保证的。

GCM会存储最多100个non-collapsible 消息，超过这个之后，所有的消息都会被GCM抛弃，并且新的消息会创建还会告诉客户端还有多久回到。消息是通过一个定期的 com.google.android.c2dm.intent.RECEIVE意图，和下面的附加部分：

- Message_type 这个值是个固定的字符串“deleted_messages”。
- Total_deleted 这个值是个关于删除消息数目的字符串。

应用程序应该响应与同步服务器同步恢复丢弃的消息。

我该选择哪种？

如果你的应用程序不需要使用non-collapsible消息，在性能上来讲collapsible消息是个很好的选择，因为它们会给设备的电池造成很少的负担。

设置消息的截至日期

生存时间（TTL）的特征可以让发件人指定使用发送请求time_to_live的参数的消息的最大寿命。此参数值必须是一个从0到2419200秒的时间，它对应于GCM存储和发送消息的最大时限。请求不包含这个字段的最大时限是4周。

以下是此功能的一些可能用途

- 视频聊天来电
- 即将到期的邀请事件
- 日历事件

背景

GCM在消息发送之后经常会立即交付消息。然而，这可能不是经常合适的。例如，设备可能会关掉，脱机，或是其他的不利因素。其他情况，发件人本身可能要求消息不能传递，直至设备变为活动状态，可以使

用**delay_while_idle**标志。最后，GCM的可能故意拖延消息，以防止应用程序消耗过多的资源和影响电池寿命。

当发生这种情况，GCM会存储消息并且只要它可行了就会交付。虽然在大多数情况下，这是很好的，但是也有一些后期的消息可能也永远不会被交付的应用程序。例如，如果消息是一个小的时间内有意义的来电或视频聊天通知，在呼叫前被终止了。或如果该消息是一个事件的邀请，如果事件已经结束后收到，这就没有用了。

指定消息的截至日期的另一个优点是GCM不会扼杀一个0秒**time_to_live**价值的消息。换句话说，GCM尽最大努力保证消息的交付“现在或者是从不”。请记住，不能立即交付。0表示消息**time_to_live**价值将被丢弃。然而，因为这样的邮件永远不会存储，于是就提供了最好延迟发送通知。

这里有一个包含TTL的JSON格式的请求：

```
{
  "collapse_key" : "demo",
  "delay_while_idle" : true,
  "registration_ids" : [ "xyz" ],
  "data" : {
    "key1" : "value1",
    "key2" : "value2",
  },
  "time_to_live" : 3
},
```

从多个发送人中接受消息

GCM允许多方发送消息给同一个应用程序。例如，你的应用程序是一个文章收集器有多个贡献者，当他们发布一篇新文章的时候，你想要他们中的每个人都 能发送消息。这些消息可能包含一个URL链接以便应用程序能下载这篇文章。GCM使你能够让这些贡献者者发送自己的消息不必集中在同一个位置。

为了实现这个可能，你只需要做的使是你的发送人创建一个自己的项目ID。然后当请求注册的时候包含这些ID在发送人的字段中，通过逗号隔开。最后，分配把这些ID分配给你搭挡，然后他们就有能力发送消息给你的应用程序使用他们自己的认证的钥匙。

下面的代码片段指出了这些特征。发送人传递作为额外的意图在一个逗号分隔的列表中。

```
Intent intent = new Intent(GCMConstants.INTENT_TO_GCM_REGISTRATION);
intent.setPackage(GSF_PACKAGE);
intent.putExtra(GCMConstants.EXTRA_APPLICATION_PENDING_INTENT,
    PendingIntent.getBroadcast(context, 0, new Intent(), 0));
String senderIds = "968350041068,652183961211";
intent.putExtra(GCMConstants.EXTRA_SENDER, senderIds);
ontext.startService(intent);
```

注意这里有最多100个发送人的限制。

来自“[index.php?title=GCM_Advanced_Topics&oldid=11134](#)”



Migration

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人：玄月冰灵

分任务原文链

接：<http://developer.android.com/guide/google/gcm/c2dm.html>

目录

[[隐藏](#)]

1 迁移

- [1.1 历史回顾](#)
- [1.2 GCM与C2DM相比有什么不同?](#)
 - [1.2.1 C2DM和GCM之间的关系](#)
- [1.3 迁移你的应用](#)
 - [1.3.1 改变客户端](#)
 - [1.3.2 改变服务器](#)

迁移

安卓云至设备消息（C2DM）已弃用。C2DM服务会在短期内继续维持，但C2DM将不能允许新用户，也不会授予新的配额。强烈鼓励C2DM开发者转移到谷歌云推送（GCM）。GCM是新一代的C2DM。

这个文档是写给将从C2DM转移到GCM的开发者。它描述了GCM和C2DM的区别，解释了如何将存在的C2DM应用转移到GCM。

历史回顾

C2DM启动于2010年，帮助安卓应用从服务器发送数据导他们的应用。服务器可以告诉应用立即联系服务器，获取应用更新货用户数据。C2DM处理所有的消息队列和发送到目标设备上运行的目标应用。

GCM代替C2DM。GCM重点如下：

- 简单实用，无需注册表单。
- 无配额。
- 通过安卓开发控制台可以使GCM和C2DM统计有效。
- 省电。
- 对新API的更多设置。

GCM与C2DM相比有什么不同？

GCM的建立以C2DM为核心基础。这里是它们的不同：

简单API密钥

使用GCM服务，你需要从谷歌API控制台页面获取简单API密钥。更多的信息，请访问 [开始](#)。

发送者ID

在C2DM，发送者ID是一个邮箱地址。在GCM，发送者ID是你从API控制台取得的项目ID，详细请访问 [开始](#)。

JSON格式

GCM HTTP请求支持除无格式文本意外的JSON格式。更新，请访问 [构建概述](#)。

组播消息

在GCM中你可以同时发送同样的消息到多个设备。例如，一个体育应用想要发送一个分数更新给体育迷，立即可以用相同的请求（JSON请求）发送消息给多大1000个注册ID。更多，请访问 [构建概述](#)。

多发送者

多个成员可以用一个共同的注册ID发送消息给相同的引用。更多，请访问 [高级主题](#)。

存活时间消息

想视频聊天或日历应用可以发送存活时间值在0~4周过期的邀请事件。**GCM**将会存储消息知道它们过期。一个消息的存活时间值到0将不会存储在**GCM**服务器，也不会节流。更多，请查看 [高级主题](#)。

消息负载

应用可以使用“消息负载”发送多大4KB的信息。例如，这将对聊天应用非常有用。使用这个特征，简单省略 `collapse_key` 参数，消息将不会崩溃。**GCM**将存储多大100挑消息。如果你超过这个数字，所有的消息会丢弃，但你将收到一条特别的消息。如果一个应用收到这条消息，它需要和服务器同步。更多，请查看 [高级主题](#)。

标准注册ID

C2DM和**GCM**之间的关系

C2DM和**GCM**并不是彼此协作的。例如，你不能发送通知从**GCM**到**C2DM**注册ID，也不能把**C2DM**注册ID当做**GCM**注册ID用。从你的服务端应用，你必须保持跟踪是否是来自**C2DM**或**GCM**的注册ID，并使用适当的终端。

当你从**C2DM**转移到**GCM**，你的服务器需要知道是否被授予一个包含老的**C2DM**发送者或新的**GCM**项目ID的注册ID。这是我们推荐的方法：有新应用版本（使用**GCM**）用注册ID发送一位数据。这位数据告诉你的服务器这个注册ID是**GCM**的。如果你没有获得额外位数据，你标记注册ID是**C2DM**的。你可以完成迁移。

迁移你的应用

这部分说明了如何转移存在的**C2DM**应用到**GCM**。

改变客户端

迁移很简单！只是应用中必须的更改的是当注册新服务时用生成的项目代替过去注册intent的发送者参数邮箱账号。例如：

```
Intent registrationIntent = new Intent( "com.google.android.c2dm.intent.REGISTER" );
// sets the app name in the intent
registrationIntent.putExtra( "app" , PendingIntent.getBroadcast( this ,
0 , new Intent( ) , 0 ) );
registrationIntent.putExtra( "sender" , senderID );
startService( registrationIntent );
```

收到从GCM的响应后，获取的注册ID必须被发送到应用服务器。当这样做了，应用应该指示它发送了一个GCM注册ID，使得服务器可以和存在的C2DM注册区分。

改变服务器

当应用服务器收到一个GCM注册ID，它应该存储并标记了它。发送消息到GCM设备要求一些修改：

- 请求应该被发送到一个新的终端：<https://android.googleapis.com/gcm/send>.
- 请求的授权头应该包含注册期间生成的API密钥。这个密钥代替弃用的客户端登陆授权令牌。

例如：

```
Content-Type:application/json
Authorization:key=AIzaSyB-1uEai2WiUapxCs2Q0GZYzPu7Udno5aA
{
  "registration_id" : "APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...",
  "data" : {
    "Team" : "Portugal",
    "Score" : "3",
    "Player" : "Varela",
  }
},
```

此主题的详细讨论和更多例子，亲查看 构建概述。最后，一旦你应用有足够的用户迁移到新服务，你可能想要利用新的JSON格式请求，提供GCM的完整特征设置的访问。

来自“[index.php?title=Migration&oldid=8519](#)”



Overview

来自eoeAndroid wiki

跳转到： [导航](#), [搜索](#)

负责人： GloriousOnion

原文地址：<http://developer.android.com/guide/webapps/overview.html>

Web App概述

实质上，Android有两种不同的方法发布应用：一是通过客户端应用（通过Android SDK开发并以.apk文件安装到用户设备上），二是通过Web应用（其通过互联网标准进行开发并使用浏览器进行访问，其特点是不会向用户设备安装任何程序）。

图1说明了如何从web浏览器或者安卓应用程序来访问网页。但是不要仅仅作为查看网站的手段来开发安卓应用程序。相反，安卓应用程序嵌入的网页应该针对环境进行设计。甚至可以定义安卓应用程序和网页之间的接口，这一接口允许JavaScript调用安卓应用程序API——向基于web的应用程序提供安卓API。

要为安卓供电设备开发网页，请参阅下列文档：

- [Web应用的目标界面](#)

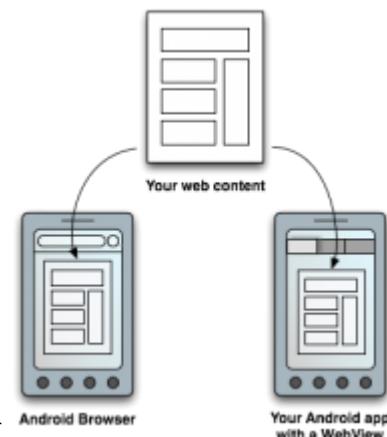


图1. 可以通过传统浏览器或是布局中有WebView的Android应用将你的网络内容提供用户使用。

介绍如何恰当地定义Android设备上Web应用尺寸并支持多种分辨率。如果你正在一个搭建至少对Android设备适用（应为发布到网上的所有东西做出假设）的Web应用，本文档会有不少帮助，而对以移动

设备为目标及正在使用[WebView](#)的读者，本文档则大有裨益。

- [**在WebView中构建Web App**](#)

介绍如何使用[WebView](#)将网页嵌入到Android应用中，并将JavaScript与Android API绑定。

- [**调试Web App**](#)

介绍如何通过JavaScript输出API调试Web App

- [**Web应用程序最佳实践**](#)

为了提供有效web应用程序，您应该遵循一系列做法。

来自“[index.php?title=Overview&oldid=13824](#)”

