

Android 中文合集（126+8 篇）

Android 中文翻译组

关于

Android 中文翻译组是一个非盈利性质的开源组织，聚一批开发人员、大学生、研究生等 Android 爱好者，利用业余时间对 Android 相关的 API 及开发者指南等进行翻译，至今已超过 200 人报名参与，欢迎更多朋友加入，联系 Mail: over140@gmail.com，关于翻译组的更多介绍，请看[这里](#)。

本合集包含 126 章节 API 和 8 章开发者指南。

章节

命名空间	完成章节数(126+8)
android	3
android.accessibilityservice	1
android.bluetooth	8
android.content	1
android.media	2
android.net	2
android.os	2
android.view	12
android.view.inputmethod	5
android.widget	90
Dev Guide	8

名单

本合集参与章节翻译名单：[移动云](#) [文斌](#)、[深夜未眠](#)、[xiaoQLu](#)、[gansc23](#)、[Atomic](#)、[Aman](#)、[Android Club SYSU](#)、[cnmahj](#)、[cofice](#)、[HalZhang](#)、[henly.zhang](#)、[jiahuibin](#)、[Kun](#)、[loveshirui](#)、[madgoat](#)、[pengyouhong](#)、[Tina](#)、[wallace2010](#)、[0_1](#)、[凌云健笔](#)、[逝憶流緣](#)、[天涯明月刀](#)、[Haiya](#)蝴蝶、桂仁、唐明、颖哥儿、[思考的狼](#)、[德罗德](#)、首当其冲、CN 七号、麦子、[獨鶡躑躅](#)、我是谁、一昕、[六必治](#)、[农民伯伯](#)。

支持

本翻译组得到以下社区支持：

[eoeAndroid](#) 为翻译组合集发布特别提供置顶支持。

[eoe Android 开发者门户](#)拥有国内第一款 Android 软件商店:优亿市场(eoeMarket)和中国最早、最专业、最大的 Android 开发者社区。海量、全面、优质的 Android 学习资料、互助共享的 eoeAndroid 开发者服务，让 Android 的开发者、Android 爱好者在 eoeAndroid 中迅速成长，从而汇聚了超过 16 万的 Android 开发者。eoeAndroid 为广大 Android 开发者奠定了坚实的技术基础。

CMD100 为翻译组开辟独立专区。

中国手机开发者联盟(CMD)是 CFT 旗下的一个 Android 开发者技术交流网站。该网站于 2010 年成立,成立至今,开辟了很多国内 Android 开发网站史上的先河,首当其冲的是**免费发布软件到 Google Market 的接口**、这也是网站最给力的特色功能。是目前国内**唯一真正能上传 Android 应用到 Google Market 的软件平台**。开发者上传应用不收取任何费用、只需提供有效信息进行身份认证即可。免去信用卡激活等复杂环节。软件收益通过支付宝进行方便快捷的结算。其次还有官方 Android 开发创意,官方完整版学视频,互动式软件销售平台,CMD100 科技直播间,程序美工直通车等多个原创项目。

中国手机开发者联盟标识为 CMD100,全称是 China Mobile Developer .其后的一百,是暗指旗下平台的注册会员能够在兼职的同时,每天都有 100 美元的收入。这也是网站创始人 Kevin Huang 一直以来追求的目标。让中国程序员赚程序员该赚的钱。让贴着 Made In China 标签的软件直接打入欧美市场、与国际接轨。

移动社区 为翻译组开辟专区并提供学分等奖励。

中国移动开发者社区是由中国移动牵头开发和维护的一个大型的、综合性的资讯社区,为各类移动开发者提供从开发、测试到应用发布等环节所需的一系列业内外资讯、交流平台和自我展示空间。移动世界,精彩无限!我们期待您的加盟。

感谢各社区的大力支持!

招募

文档翻译员, 要求:

1. 有耐心, 这是一场持久战, 需要大家的坚持。
2. 有态度, 认真对待每一篇译稿。
3. 会英语, 至少在翻译工具的帮助下能读懂英文原文。

[急]审核员, 要求:

1. 脾气好, 审稿过程中需要和组员沟通, 需要好脾气来沟通。
2. 技术好, 工作经验 2 年以上, Android 经验半年以上, 对技术有自己的理解。
3. 英语好, 英语 6 级以上, 有相关翻译经验更佳。
4. 原则上每周能审稿至少 1 篇。

共享

无论你是个人还是团队,不管是否加入我们,如果翻译 Android 官方相关文章,请与我们分享进度,把你翻译的章节发邮箱到 over140@gmail.com ,以免重复翻译。我们的进度:[这里](#) (以“进度_”开头的 Excel 文件)。

计划

计划下一步不再出 chm 格式的合集,而是用类似 chm 界面风格、支持自动更新内容的客户端程序,敬请期待,欢迎给我们提建议。

Dev Guide

[What is Android?](#)

[aapt](#)

[adb](#)

[AIDL](#)

[Application Fundamentals](#)

[monkeyrunner](#)

[Other Tools](#)

[App Install Location](#)

What is Android?

署名: gansc23

链接: <http://www.cnblogs.com/gansc23>

版本: Android 3.0 r1

声明

本文档转载并整理自: [Android 是什么 \(What is Android\)](#)。

本文翻译也部分参考: [Android 基础: 什么是 Android? \(电脑老师\)](#)

原文

<http://developer.android.com/guide/basics/what-is-android.html>

Android 是什么? (What is Android?)

Android 是一个针对于移动设备的软件栈, 它包括操作系统, 中间件和关键应用程序。

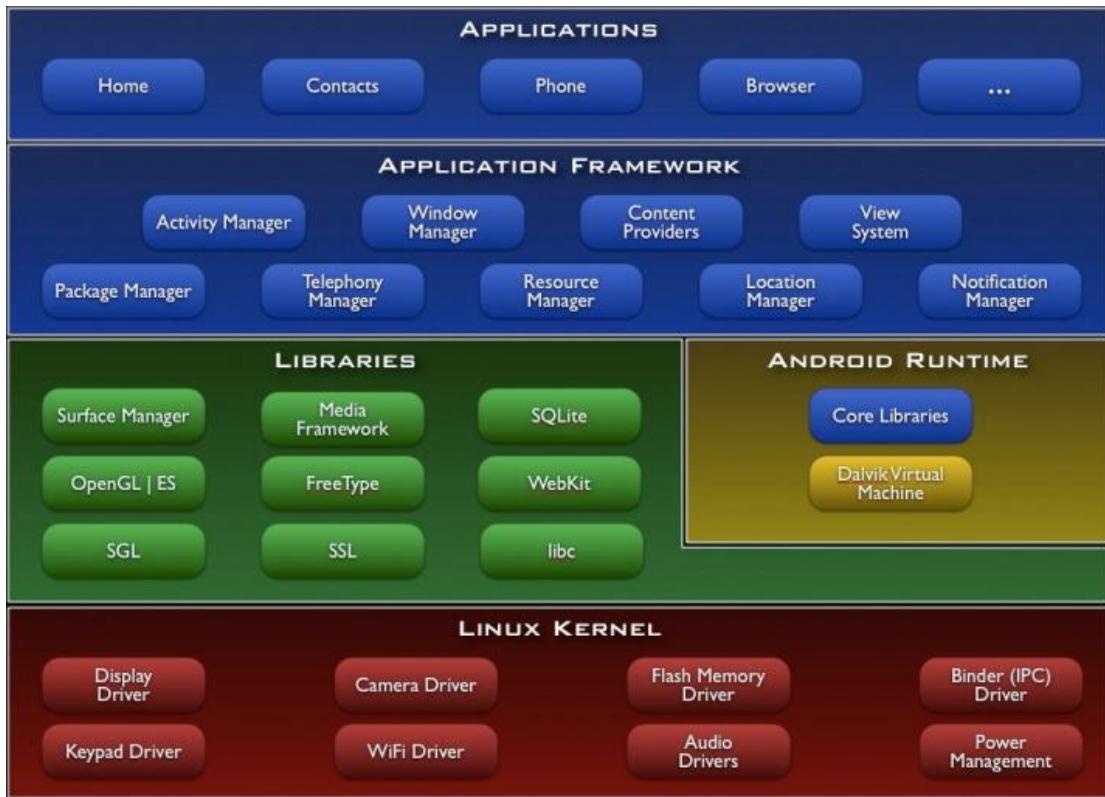
[Android SDK](#) 提供了在 Android 平台使用 Java 编程语言进行开发应用程序的必要的工具和 API。

特性(Features)

- 应用程序框架(**Application framework**) 可重用并可替换的组件
- **Dalvik 虚拟机(Dalvik virtual machine)** 为移动设备而优化
- 集成浏览器(**Integrated browser**) 基于开源的 WebKit 引擎
- 优化的图形处理(**Optimized graphics**) 以定制的 2D 图形库和基于 OpenGL ES 1.0 规范的 3D 图形(可选的硬件加速)为基础
- **SQLite** 结构化数据存储
- 媒体支持(**Media support**) 通用音频、视频, 还有图像格式(MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM 电话(GSM Telephony)** (依赖硬件)
- **蓝牙(Bluetooth), EDGE, 3G, and WiFi** (依赖硬件)
- 照相机, **GPS**, 指南针, 加速感应器(**Camera, GPS, compass, and accelerometer**) (依赖硬件)
- 丰富的开发环境(**Rich development environment**) 包括设备模拟器, 调试工具, 内存和性能分析工具和用于 Eclipse IDE 的插件

Android 架构(Android Architecture)

下图展示了 Android 操作系统的主要组件。每个组件在下面有更详细的描述。



应用程序(Applciations)

Android 将配备一系列核心应用程序，包括电子邮件客户端，短信程序，日历，地图，浏览器，通讯录等。所有的应用程序都是使用 Java 编程语言。

应用框架(Application Framework)

通过提供一个开放的开发平台，Android 提供给开发者建立极其丰富和创新应用的能力。开发者自由地享有硬件设备的优势，访问本地信息，运行后台服务，设置警示，向状态栏添加通知等。

开发人员可以使用和核心应用程序使用的完全相同的 API 框架。应用程序架构的设计的目的是为了简化组件的重用；任何应用程序都可以发布它的功能，其他的应用程序可能会利用到这些功能（需遵守框架提供的安全约束）。依靠同样的机制，组件可以被用户所取代。

底层的所有的应用程序是一组服务和子系统，包括：

- 一组丰富并且可扩展的 view，这组 view 可以被用来构建一个应用程序，包括列表，表格，文本框，按钮，甚至可嵌入的 Web 浏览器。
- 一组 Content Providers，可以使应用程序访问其它应用程序的数据（比如通讯录），或者共享自己的数据。
- 一个资源管理器(Resource Manager)，提供对非代码资源的访问，比如本地化的字符串，图形和布局文件。
- 一个通知管理(Notification Manager)，可以使所有的应用程序在状态栏显示定制的提醒。
- 一个 Activity 管理(Activity Manager)，它管理的应用程序的生命周期，并且提供了一个通用的后台切换栈。

如需了解一个应用程序的详细信息和具体示例，请参考 Notepad 代码示例([Notepad](#))

[Tutorial](#))。

库(Libraries)

Android 包括了一套 C/C++库，这套库被 Android 系统的各个组件使用。通过 Android 的应用框架，这些功能被开放给开发者。其中的一些核心库如下：

- **系统 C 库(System C library)** – 一个继承自 BSD 的标准 C 系统实现 (libc)，被调整成面向基于 linux 的嵌入式设备。
- **媒体库(Media Libraries)** – 基于 PacketVideo 的 OpenCore；该库支持回放和录制许多流行的音频和视频格式，以及静态图像文件，包括 MPEG4, H.264, MP3, AAC, AMR, JPG 和 PNG 格式。
- **Surface 管理器(Surface Manager)** – 管理显示子系统，并能无缝地组合多个应用的 2D 和 3D 图像层。
- **LibWebCore** – 一个流行的 Web 浏览器引擎，它对 Android 浏览器和嵌入式 Web 视图具有良好的支持。
- **SGL** – 底层的 2D 图形引擎。
- **FreeType** – 位图和矢量字体渲染。
- **SQLite** – 所有的应用程序使用并且强大而轻量级的关系型数据库引擎。

运行时(Android Runtime)

Android 包括一个核心库的集合，她们提供了 Java 编程语言的核心库中的绝大多数功能。

每一个 Android 应用都在她自己的进程中运行，该进程也属于某个 Dalvik 虚拟机的实例。 Dalvik 被设计成能让设备高效地运行多个虚拟机。 Dalvik 虚拟机执行的是.dex 结尾的 Dalvik 可执行文件格式，该格式被优化为最小内存使用。虚拟机是基于寄存器的，并且运行那些 Java 编程语言所编译的类，这些类被内置的 dx 工具转换为.dex 格式。

Dalvik 虚拟机依赖 Linux 内核来提供底层的功能，比如线程和低级内存管理。

Linux 内核(Linux Kernel)

Android 依赖 Linux 2.6 来提供核心系统服务，比如安全、存储管理、进程管理、网络栈和驱动模型。该内核同时扮演着介于硬件和软件栈的其余部分之间的一个抽象层。

aapt

译者署名: 移动云 文斌

译者链接: <http://blog.csdn.net/caowenbin>

版本: Android 2.3 r1

原文

<http://developer.android.com/guide/developing/tools/aapt.html>

使用 aapt

aapt 是标准的 Android 辅助打包工具，位于 SDK 的 tools/文件夹下。该工具允许查看、创建或更新 Zip 兼容格式 (zip,jar,apk) 的文档，并且能将资源编译到二进制格式的包中。

通常不需要直接使用 aapt 工具，IDE 插件和编译脚本能利用它打包 apk 文件来合成应用程序。

详细的用法可以打开命令行终端，到 tools 文件夹下运行如下命令：

Linux 或 Mac OS X:

```
./aapt
```

Windows:

```
aapt.exe
```

adb

译者署名: 移动云 文斌

译者链接: <http://blog.csdn.net/caowenbin>

版本: Android 2.3 r1

原文

<http://developer.android.com/guide/developing/tools/adb.html>

Android Debug Bridge

Android 调试桥接器, 简称 adb, 是用于管理模拟器或真机状态的万能工具, 采用了客户端-服务器模型, 包括三个部分:

- 客户端部分, 运行在开发用的电脑上, 可以在命令行中运行 adb 命令来调用该客户端, 像 ADB 插件和 DDMS 这样的 Android 工具也可以调用 adb 客户端。
- 服务端部分, 是运行在开发用电脑上的后台进程, 用于管理客户端与运行在模拟器或真机的守护进程通信。
- 守护进程部分, 运行于模拟器或手机的后台。

当启动 adb 客户端时, 客户端首先检测 adb 服务端进程是否运行, 如果没有运行, 则启动服务端。当服务端启动时, 它会绑定到本地的 TCP5037 端口, 并且监听从 adb 客户端发来的命令——所有的 adb 客户端都使用 5037 端口与 adb 服务端通信。

接下来服务端与所有正在运行的模拟器或手机连接。它通过扫描 5555-5585 之间的奇数号端口来搜索模拟器或手机, 一旦发现 adb 守护进程, 就通过此端口进行连接。需要说明的是, 每一个模拟器或手机使用一对有序的端口, 偶数号端口用于控制台连接, 奇数号端口用于 adb 连接, 例如:

Emulator 1, console: 5554

Emulator 1, adb: 5555

Emulator 2, console: 5556

Emulator 2, adb: 5557 ...

即如果模拟器与 adb 在 5555 端口连接, 则其与控制台的连接就是 5554 端口。

当服务端与所有的模拟器建立连接之后, 就可以使用 adb 命令来控制或者访问了。因为服务端管理着连接并且可以接收到从多个 adb 客户端的命令, 所以可以从任何一个客户端或脚本来控制任何模拟器或手机设备。

下文介绍了可以用来管理模拟器或手机的这些 adb 命令。如果是在 Eclipse 并且安装了 ADT 插件的环境下开发 Android 应用程序, 就不需要从命令行使用 adb 了, ADT 插件已经提供了透明的集成。不过, 还是可以在调试等需要的时候直接使用 adb。

使用 adb 命令

从开发用电脑的命令行或脚本文件中使用 adb 命令的用法是:

```
adb [-d|-e|-s <serialNumber>] <command>
```

当使用的时候, 程序会调用 adb 客户端。因为 adb 客户端不需要关联到任何模拟器, 所以如果有多个模拟器或手机正在运行, 就需要使用-d 参数指定要操作的是哪一个, 更多关于这些选项参数的使用可以参见 [Directing Commands to a Specific Emulator/Device Instance](#)。

查询模拟器或手机状态

了解 adb 服务端连接的模拟器或手机可以帮助更好的使用 adb 命令, 这可以通过 devices 命令列举出来:

```
adb devices
```

执行结果是 adb 为每一个设备输出以下状态信息:

- 序列号(serialNumber) — 由 adb 创建的使用控制台端口号的用于唯一标识一个模拟器或手机设备的字符串, 格式是 <设备类型>-<端口号>, 例如: `emulator-5554`
- 状态(state) — 连接状态, 其值是:
 - `offline` — 未连接或未响应
 - `device` — 已经连接到服务商。注意这个状态并不表示 Android 系统已经完全启动起来, 系统启动的过程中已经可以连接 adb, 但这个状态是正常的可操作状态。

每一个设备的输出形如:

```
[serialNumber] [state]
```

下面是 devices 命令和其执行结果:

```
$ adb devices
List of devices attached
emulator-5554 device
emulator-5556 device
emulator-5558 device
```

如果没有模拟器或手机在运行, 该状态返回的是 `no device`。

操作指定的模拟器或手机

如果有多个模拟器或手机正在运行, 当使用 adb 命令的时候就需要指定目标设备, 这可以通过使用-s 选项参数实现, 用法是:

```
adb -s <serialNumber> <command>
```

即可以在 adb 命令中使用序列号指定特定的目标, 前文已经提到的 devices 命令可以实现查询设备的序列号信息。

例如:

```
adb -s emulator-5556 install helloworld.apk
```

需要注意的是, 如果使用了 `-s` 而没有指定设备的话, adb 会报错。

安装应用程序

可以使用 adb 从开发用电脑中复制应用程序并且安装到模拟器或手机上, 使用 `install` 命令即可, 在这个命令中, 必须指定待安装的 apk 文件的路径:

```
adb install <path_to_apk>
```

关于创建可安装的应用的更多信息，请参见 [Android Asset Packaging Tool \(aapt\)](#).

注意，如果使用了安装有 ADT 插件的 Eclipse 开发环境，就不需要直接使用 adb 或 aapt 命令来安装应用程序了，ADT 插件可以自动完成这些操作。

转发端口

可以使用 forward 命令转发端口 — 将特定端口上的请求转发到模拟器或手机的不同端口上。下例是从 6100 端口转到 7100 端口：

```
adb forward tcp:6100 tcp:7100
```

也可以使用 UNIX 命名的 socket 标识：

```
adb forward tcp:6100 local:logd
```

与模拟器或手机传输文件

可以使用 adb 的 pull 和 push 命令从模拟器或手机中复制文件，或者将文件复制到模拟器或手机中。与 install 命令不同，它仅能复制.apk 文件到特定的位置，pull 和 push 命令可以复制任意文件夹和文件到模拟器或手机的任何位置。

从模拟器或手机中复制一个文件或文件夹（递归的）使用：

```
adb pull <remote> <local>
```

复制一个文件或文件夹（递归的）到模拟器或手机中使用：

```
adb push <local> <remote>
```

在这个命令中<local>和<remote>引用的是文件或文件夹的路径，在开发用电脑上的是 local，在模拟器或手机上的是 remote。

例如：

```
adb push foo.txt /sdcard/foo.txt
```

adb 命令列表

下表列出了所有 adb 支持的命令及其说明：

类别	命令	说明	备注
----	----	----	----

可选项	<code>-d</code>	命令仅对 USB 设备有效	如果有多个 USB 设备就会返回错误
	<code>-e</code>	命令仅对运行中的模拟器有效	如果有多个运行中的模拟器就会返回错误
	<code>-s <serialNumber></code>	命令仅对 adb 关联的特定序列号的模拟器或手机有效(例如 "emulator-5556").	如果不指定设备就会返回错误
一般项	<code>devices</code>	输出所有关联的模拟器或手机设备列表	参见 Querying for Emulator/Device Instances 以获得更多信息。
	<code>help</code>	输出 adb 支持的命令	
	<code>version</code>	输出 adb 的版本号	
调试项	<code>logcat [<option>] [<filter-specs>]</code>	在屏幕上输出日志信息	
	<code>bugreport</code>	为报告 bug, 在屏幕上输出 <code>dumpsyst</code> , <code>dumpstate</code> 和 <code>logcat</code> 数据	
	<code>jdwp</code>	输出有效的 JDWP 进程信息	可以使用 <code>forward jdwp:<pid></code> 转换端口以连接到指定的 JDWP 进程, 例如: <code>adb forward tcp:8000 jdwp:472</code> <code>jdb -attach localhost:8000</code>
数据项	<code>install <path-to-apk></code>	安装应用程序 (用完整路径指定.apk 文件)	
	<code>pull <remote> <local></code>	从开发机 COPY 指定的文件到模拟器或手机	
	<code>push <local> <remote></code>	从模拟器或手机 COPY 文件到开发机	
端口和网络项	<code>forward <local> <remote></code>	从本地端口转换连接到模拟器或手机的指定端口	端口可以使用以下格式表示: <ul style="list-style-type: none"> ● <code>tcp:<portnum></code> ● <code>local:<UNIX domain socket name></code> ● <code>dev:<character device name></code> ● <code>jdwp:<pid></code>
	<code>ppp <tty> [parm]...</code>	通过 USB 运行 PPP <ul style="list-style-type: none"> ● <code><tty></code> — PPP 流中的 tty。例如:<code>/dev/omap_csmi_tty1</code>。 	

		<ul style="list-style-type: none"> ● [parm]... — 0 到多个 PPP/PPPD 选项，例如 <code>defaultroute</code>, <code>local</code>, <code>notty</code> 等等。 <p>注意不用自动启动 PPP 连接</p>	
脚本项	<code>get-serialno</code>	输出 adb 对象的序列号	参见 Querying for Emulator/Device Instances 以获得更多信息。
	<code>get-state</code>	输出 adb 设备的状态	
	<code>wait-for-device</code>	阻塞执行直到设备已经连接，即设备状态是 <code>device</code> .	可以在其他命令前加上此项，那样的话 adb 就会等到模拟器或手机设备已经连接才会执行命令，例如: <code>adb wait-for-device shell getprop</code> 注意该命令并不等待系统完全启动，因此不能追加需要在系统完全启动才能执行的命令，例如 <code>install</code> 命令需要 Android 包管理器支持，但它必须在系统完全启动后才有效。下面的命令 <code>adb wait-for-device install <app>.apk</code> 会在模拟器或手机与 adb 发生连接后就执行 <code>install</code> ，但系统还没有完全启动，所以会引起错误。
服务端项	<code>start-server</code>	检测 adb 服务进程是否启动，如果没启动则启动它。	
	<code>kill-server</code>	终止服务端进程	
Shell	<code>shell</code>	在目标模拟器或手机上启动远程 SHELL	参见 Issuing Shell Commands 以获得更多信息。
	<code>shell [<shellCommand>]</code>	在目标模拟器或手机上执行 <code>shellCommand</code> 然后退出远程 SHELL	

执行 Shell 命令

Adb 提供了 `shell` 来在模拟器或手机上运行各种各样的命令，这些命令的二进制形式存储在这个路径中：

```
/system/bin/...
```

无论是否进入 adb 远程 shell，都可以使用 shell 命令来执行。

在未进入远程 shell 的情况下可以按上述格式执行单条命令：

```
adb [-d|-e|-s {<serialNumber>}] shell <shellCommand>
```

启动远程 shell 使用下面的格式：

```
adb [-d|-e|-s {<serialNumber>}] shell
```

退出远程 shell 时使用 `CTRL+D` 或 `exit` 终止会话。

以下是可以使用的 shell 命令的更多信息。

从远程 shell 检查 sqlite3 数据库

通过远程 shell，可以使用 `sqlite3` 命令行程序来管理由应用程序创建的 SQLite 数据库。`sqlite3` 工具包含很多有用的命令，例如 `.dump` 用于输出表格的内容，`.schema` 用于为已经存在的表输出 SQL CREATE 语句。并且该工具也提供了联机执行 SQLite 命令的能力。

使用 `sqlite3` 时，向前面描述的那样进入模拟器的远程 shell，然后使用 `sqlite3` 命令。也可以在调用 `sqlite3` 时指定数据库的全路径。SQLite3 数据库存储在

`/data/data/<package_name>/databases/` 路径下。

示例：

```
$ adb -s emulator-5554 shell
# sqlite3
/data/data/com.example.google.rss.rssexample/databases/rssitems.db
SQLite version 3.3.12
Enter ".help" for instructions
.... enter commands, then quit...
sqlite> .exit
```

一旦运行了 `sqlite3`，就可以使用 `sqlite3` 命令，退出并返回远程 shell 可以使用 `exit` 或 `CTRL+D`。

使用 Monkey 进行 UI 或应用程序测试

Monkey 是运行于模拟器或手机上的一个程序，通过生成伪随机的大量的系统级的用户事件流来模拟操作，包括单击、触摸、手势等。从而为正在开发中的应用程序通过随机响应进行压力测试。

最简单使用 monkey 的方式是通过下面的命令行，它可以运行指定的应用程序并向其发送 500 个伪随机事件。

```
$ adb shell monkey -v -p your.package.name 500
```

关于 monkey 更多的选项及详细信息，请参见 [UI/Application Exerciser Monkey](#)。

其他 Shell 命令

下表列出了很多有效的 adb shell 命令，完整的列表可以通过启动模拟器并且使用 adb -help 命令获取。

```
adb shell ls /system/bin
```

帮助对于大部分命令是有效的。

Shell 命令	描述	备注
<code>dumpsys</code>	在屏幕上显示系统数据	
<code>dumpstate</code>	将状态输出到文件	
<code>logcat [<option>]... [<filter-spec>]...</code>	输出日志信息	The Dalvik Debug Monitor Service (DDMS) 工具提供了更易于使用的智能的调试环境。
<code>dmesg</code>	在屏幕上输出核心调试信息	
<code>start</code>	启动或重新启动模拟器或手机	
<code>stop</code>	停止模拟器或手机	

使用 logcat 查看日志

Android 日志系统提供了从众多应用程序和系统程序中收集和查看调试信息的机制，这些信息被收集到一系统循环缓冲区中，可以 logcat 命令查看和过滤。

使用 logcat 命令

查看和跟踪系统日志缓冲区的命令 `logcat` 的一般用法是：

```
[adb] logcat [<option>] ... [<filter-spec>] ...
```

下文介绍过滤器和命令选项，详细内容可参见 [Listing of logcat Command Options](#)。

可以在开发机中通过远程 shell 的方式使用 `logcat` 命令查看日志输出：

```
$ adb logcat
```

如果是在远程 shell 中可直接使用命令：

```
# logcat
```

过滤日志输出

每一条日志消息都有一个标记和优先级与其关联。

- 标记是一个简短的字符串，用于标识原始消息的来源（例如“View”来源于显示系统）。
- 优先级是下面的字符，顺序是从低到高：
 - `V` — 明细（最低优先级）
 - `D` — 调试
 - `I` — 信息
 - `W` — 警告
 - `E` — 错误
 - `F` — 严重错误
 - `S` — 无记载（最高优先级，没有什么会被记载）

通过运行 `logcat`，可以获得一个系统中使用的标记和优先级的列表，观察列表的前两列，给出的格式是`<priority>/<tag>`。

这里是一个日志输出的消息，优先级是“`I`”，标记是“`ActivityManager`”：

```
I/ActivityManager( 585): Starting activity: Intent
{ action=android.intent.action... }
```

如果想要减少输出的内容，可以加上过滤器表达式进行限制，过滤器可以限制系统只输出感兴趣的标记-优先级组合。

过滤器表达式的格式是 `tag:priority...`，其中 `tag` 是标记，`priority` 是最小的优先级，该标记标识的所有大于等于指定优先级的消息被写入日志。也可以在一个过滤器表达式中提供多个这样的过滤，它们之间用空格隔开。

下面给出的例子是仅输出标记为“`ActivityManager`”并且优先级大于等于“`Info`”和标记为“`MyApp`”并且优先级大于等于“`Debug`”的日志：

```
adb logcat ActivityManager:I MyApp:D *:S
```

上述表达式最后的 `*:S` 用于设置所有标记的日志优先级为 `S`，这样可以确保仅有标记为“View”（译者注：应该为 `ActivityManager`，原文可能是笔误）和“`MyApp`”的日志被输出，使用 `*:S` 是可以确保输出符合指定的过滤器设置的一种推荐的方式，这样过滤器就成为了日志输出的“白名单”。

下面的表达是显示所有优先级大于等于“warning”的日志：

```
adb logcat *:W
```

如果在开发用电脑上运行 `logcat`（相对于运行远程 `shell` 而言），也可以通过 `ANDROID_LOG_TAGS` 环境变量设置默认的过滤器表达式：

```
export ANDROID_LOG_TAGS="ActivityManager:I MyApp:D *:S"
```

需要注意的是，如果是在远程 `shell` 或是使用 `adb shell logcat` 命令运行 `logcat`，`ANDROID_LOG_TAGS` 不会导出到模拟器或手机设备上。

控制日志格式

日志消息在标记和优先级之外还有很多元数据字段，这些字段可以通过修改输出格式来控制输出结果，`-v` 选项加上下面列出的内容可以控制输出字段：

- `brief` — 显示优先级/标记和原始进程的 PID（默认格式）
- `process` — 仅显示进程 PID
- `tag` — 仅显示优先级/标记
- `thread` — 仅显示进程：线程和优先级/标记
- `raw` — 显示原始的日志信息，没有其他的元数据字段
- `time` — 显示日期，调用时间，优先级/标记，PID
- `long` — 显示所有的元数据字段并且用空行分隔消息内容

可以使用 `-v` 启动 `logcat` 来控制日志格式：

```
[adb] logcat [-v <format>]
```

例如使用 `thread` 输出格式：

```
adb logcat -v thread
```

注意只能在 `-v` 选项中指定一种格式。

Viewing Alternative Log Buffers

Android 日志系统为日志消息保持了多个循环缓冲区，而且不是所有的消息都被发送到默认缓冲区，要想查看这些附加的缓冲区，可以使用 `-b` 选项，以下是可以指定的缓冲区：

- `radio` — 查看包含在无线/电话相关的缓冲区消息
- `events` — 查看事件相关的消息
- `main` — 查看主缓冲区 (默认缓冲区)

`-b` 选项的用法:

```
[adb] logcat [-b <buffer>]
```

例如查看 `radio` 缓冲区:

```
adb logcat -b radio
```

查看 `stdout` 和 `stderr`

默认的, Android 系统发送 `stdout` 和 `stderr` (`System.out` 和 `System.err`) 输出到 `/dev/null`。在 Dalvik VM 进程, 可以将输出复制到日志文件, 在这种情况下, 系统使用 `stdout` 和 `stderr` 标记写入日志, 优先级是 `I`。

要想使用这种方式获得输出, 需要停止运行中的模拟器或手机, 然后使用命令 `setprop` 来允许输出重定位, 示例如下:

```
$ adb shell stop
$ adb shell setprop log.redirect-stdio true
$ adb shell start
```

系统会保留这一设置直到模拟器或手机退出, 也可以在设备中增加 `/data/local.prop` 以使得这一设备成为默认配置。

Logcat 命令选项列表

选项	描述
<code>-b <buffer></code>	加载不同的缓冲区日志, 例如 <code>event</code> 或 <code>radio</code> 。 <code>main</code> 缓冲区是默认项, 参见 Viewing Alternative Log Buffers .
<code>-c</code>	清空 (刷新) 所有的日志并且退出
<code>-d</code>	在屏幕上输出日志并退出
<code>-f <filename></code>	将日志输出到文件 <code><filename></code> , 默认输出是 <code>stdout</code> .
<code>-g</code>	输出日志的大小
<code>-n <count></code>	设置最大的循环数据 <code><count></code> , 默认是 4, 需要 <code>-r</code> 选项
<code>-r <kbytes></code>	每 <code><kbytes></code> 循环日志文件, 默认是 16, 需要 <code>-f</code> 选项
<code>-s</code>	设置默认的过滤器为无输出

<code>-v <format></code>	设置输出格式，默认的是 <code>brief</code> ，支持的格式列表参见 Controlling Log Output Format.
--------------------------------	---

停止 adb 服务

在某些情况下，可能需要终止然后重启服务端进程，例如 `adb` 不响应命令的时候，可以通过重启解决问题。

使用 `kill-server` 可以终止服务端，然后使用其他的 `adb` 命令重启。

Android Interface Definition Language(AIDL)

译者署名: 移动云 文斌

译者链接: <http://blog.csdn.net/caowenbin>

版本: Android 2.3 r1

原文

<http://developer.android.com/guide/developing/tools/aidl.html> (注意: 3.0 r1 以后移到 Appendix 下)

使用 AIDL 设计远程接口 (Designing a Remote Interface Using AIDL)

由于每个应用程序都运行在自己的进程空间，并且可以从应用程序 UI 运行另一个服务进程，而且经常会在不同的进程间传递对象。在 Android 平台，一个进程通常不能访问另一个进程的内存空间，所以要想对话，需要将对象分解成操作系统可以理解的基本单元，并且有序的通过进程边界。

通过代码来实现这个数据传输过程是冗长乏味的，Android 提供了 AIDL 工具来处理这项工作。

AIDL (Android Interface Definition Language) 是一种 IDL 语言，用于生成可以在 Android 设备上两个进程之间进行进程间通信(IPC)的代码。如果在一个进程中（例如 Activity）要调用另一个进程中（例如 Service）对象的操作，就可以使用 AIDL 生成可序列化的参数。

AIDL IPC 机制是面向接口的，像 COM 或 Corba 一样，但是更加轻量级。它是使用代理类在客户端和实现端传递数据。

使用 AIDL 实现 IPC(Implementing IPC Using AIDL)

使用 AIDL 实现 IPC 服务的步骤是：

1. 创建.aidl 文件-该文件 (YourInterface.aidl) 定义了客户端可用的方法和数据的接口。
2. 在 makefile 文件中加入.aidl 文件- (Eclipse 中的 ADT 插件提供管理功能) Android 包括名为 AIDL 的编译器，位于 tools/文件夹。
3. 实现接口-AIDL 编译器从 AIDL 接口文件中利用 Java 语言创建接口，该接口有一个继承的命名为 Stub 的内部抽象类（并且实现了一些 IPC 调用的附加方法），要做的就是创建一个继承于 YourInterface.Stub 的类并且实现在.aidl 文件中声明的方法。
4. 向客户端公开接口-如果是编写服务，应该继承 Service 并且重载 Service.onBind(Intent) 以返回实现了接口的对象实例

创建.aidl 文件(Create an .aidl File)

AIDL 使用简单的语法来声明接口，描述其方法以及方法的参数和返回值。这些参数和返回值可以是任何类型，甚至是其他 AIDL 生成的接口。重要的是必须导入所有非内置类型，哪怕是这些类型是在与接口相同的包中。下面是 AIDL 能支持的数据类型：

- Java 编程语言的主要类型 (int, boolean 等) — 不需要 import 语句。
- 以下的类 (不需要 import 语句):
 - **String**
 - **List** - 列表中的所有元素必须是在此列出的类型，包括其他 AIDL 生成的接口和可打包类型。List 可以像一般的类（例如 List<String>）那样使用，另一边接收的具体类一般是一个 ArrayList，这些方法会使用 List 接口。

- **Map** - Map 中的所有元素必须是在此列出的类型，包括其他 AIDL 生成的接口和可打包类型。一般的 maps（例如 Map<String, Integer>）不被支持，另一边接收的具体类一般是一个 HashMap，这些方法会使用 Map 接口。
 - **CharSequence** - 该类是被 TextView 和其他控件对象使用的字符序列。
 - 通常引用方式传递的其他 AIDL 生成的接口，必须要 import 语句声明
 - 实现了 Parcelable protocol 以及按值传递的自定义类，必须要 import 语句声明。
- 以下是基本的 AIDL 语法：

```
// My AIDL file, named SomeClass.aidl
// Note that standard comment syntax is respected.
// Comments before the import or package statements are not bubbled up
// to the generated interface, but comments above interface/method/field
// declarations are added to the generated interface.

// Include your fully-qualified package statement.
package com.android.sample;

// See the list above for which classes need
// import statements (hint--most of them)
import com.android.sample.IAtmService;

// Declare the interface.
interface IBankAccountService {

    // Methods can take 0 or more parameters, and
    // return a value or void.
    int getAccountBalance();
    void setOwnerNames(in List<String> names);

    // Methods can even take other AIDL-defined parameters.
    BankAccount createAccount(in String name, int startingDeposit, in IAtmService atmService);

    // All non-Java primitive parameters (e.g., int, bool, etc) require
    // a directional tag indicating which way the data will go. Available
    // values are in, out, inout. (Primitives are in by default, and cannot be otherwise).
    // Limit the direction to what is truly needed, because marshalling parameters
    // is expensive.
    int getCustomerList(in String branch, out String[] customerList);
}
```

实现接口(Implementing the Interface)

AIDL 生成了与.aidl 文件同名的接口，如果使用 Eclipse 插件，AIDL 会做为编译过程的一部分自动运行（不需要先运行 AIDL 再编译项目），如果没有插件，就要先运行 AIDL。

生成的接口包含一个名为 Stub 的抽象的内部类，该类声明了所有.aidl 中描述的方法，Stub 还定义了少量的辅助方法，尤其是 asInterface()，通过它或以获得 IBinder（当 applicationContext.bindService() 成功调用时传递到客户端的 onServiceConnected()）并且返回用于调用 IPC 方法的接口实例，更多细节参见 [Calling an IPC Method](#)。

要实现自己的接口，就从 YourInterface.Stub 类继承，然后实现相关的方法（可以创建.aidl 文件然后实现 stub 方法而不用在中间编译，Android 编译过程会在.java 文件之前处理.aidl 文件）。

这个例子实现了对 IRemoteService 接口的调用，这里使用了匿名对象并且只有一个 getPid() 接口。

```
// No need to import IRemoteService if it's in the same project.
private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
    public int getPid() {
        return Process.myPid();
    }
}
```

这里是实现接口的几条说明：

- 不会有返回给调用方的异常
- 默认 IPC 调用是同步的。如果已知 IPC 服务端会花费很多毫秒才能完成，那就不要在 Activity 或 View 线程中调用，否则会引起应用程序挂起（Android 可能会显示“应用程序未响应”对话框），可以试着在独立的线程中调用。
- AIDL 接口中只支持方法，不能声明静态成员。

向客户端暴露接口 (Exposing Your Interface to Clients)

在完成了接口的实现后需要向客户端暴露接口了，也就是发布服务，实现的方法是继承 Service，然后实现以 Service.onBind(Intent) 返回一个实现了接口的类对象。下面的代码片断表示了暴露 IRemoteService 接口给客户端的方式。

```
public class RemoteService extends Service {
    ...
    @Override
    public IBinder onBind(Intent intent) {
        // Select the interface to return. If your service only implements
        // a single interface, you can just return it here without checking
        // the Intent.
        if (IRemoteService.class.getName().equals(intent.getAction())) {
            return mBinder;
        }
        if (ISecondary.class.getName().equals(intent.getAction())) {
            return mSecondaryBinder;
        }
        return null;
    }

    /**
     * The IRemoteInterface is defined through IDL
     */
    private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
        public void registerCallback(IRemoteServiceCallback cb) {
            if (cb != null) mCallbacks.register(cb);
        }
        public void unregisterCallback(IRemoteServiceCallback cb) {
            if (cb != null) mCallbacks.unregister(cb);
        }
    };
}
```

```
/**  
 * A secondary interface to the service.  
 */  
private final ISecondary.Stub mSecondaryBinder = new ISecondary.Stub() {  
    public int getPid() {  
        return Process.myPid();  
    }  
    public void basicTypes(int anInt, long aLong, boolean aBoolean,  
        float aFloat, double aDouble, String aString) {  
    }  
};  
}
```

使用可打包接口传递参数 Pass by value Parameters using Parcables

如果有类想要能过 AIDL 在进程之间传递，这一想法是可以实现的，必须确保这个类在 IPC 的两端的有效性，通常的情形是与一个启动的服务通信。

这里列出了使类能够支持 Parcelable 的 4 个步骤：【译者注：原文为 5，但列表为 4 项，疑为作者笔误】

1. 使该类实现 Parcelable 接口。
2. 实现 public void writeToParcel(Parcel out) 方法，以便可以将对象的当前状态写入包装对象中。
3. 增加名为 CREATOR 的构造器到类中，并实现 Parcelable.Creator 接口。
4. 最后，但同样重要的是，创建 AIDL 文件声明这个可打包的类（见下文），如果使用的是自定义的编译过程，那么不要编译此 AIDL 文件，它像 C 语言的头文件一样不需要编译。

AIDL 会使用这些方法的成员序列化和反序列化对象。

这个例子演示了如何让 Rect 类实现 Parcelable 接口。

```
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();
}
}
```

这个是 Rect.aidl 文件。

```
package android.graphics;

// Declare Rect so AIDL can find it and knows that it implements
// the parcelable protocol.
parcelable Rect;
```

序列化 Rect 类的工作相当简单，对可打包的其他类型的数据可以参见 Parcel 类。

警告：不要忘了对从其他进程接收到的数据进行安全检查。在上面的例子中，rect 要从数据包中读取 4 个数值，需要确认无论调用方想要做什么，这些数值都是在可接受的范围之内。想要了解更多的关于保持应用程序安全的内容，可参见 [Security and Permissions](#)。

调用 IPC 方法(Calling an IPC Method)

这里给出了调用远端接口的步骤：

1. 声明.aidl 文件中定义的接口类型的变量。
2. 实现 ServiceConnection
3. 调用 Context.bindService()，传递 ServiceConnection 的实现
4. 在 ServiceConnection.onServiceConnected()方法中会接收到 IBinder 对象，调用

YourInterfaceName.Stub.asInterface(IBinder)将返回值转换为YourInterface类型

5. 调用接口中定义的方法。应该总是捕获连接被打断时抛出的 DeadObjectException 异常，这是远端方法唯一的异常。
6. 调用 Context.unbindService()断开连接

这里是几个调用 IPC 服务的提示：

- 对象是在进程间进行引用计数
- 可以发送匿名对象作为方法参数

以下是演示调用 AIDL 创建的服务，可以在 ApiDemos 项目中获取远程服务的示例。

```
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;  
    mKillButton.setEnabled(false);  
}  
};  
  
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
        // to us that information.
        if (mSecondaryService != null) {
            try {
                int pid = mSecondaryService.getPid();
                // Note that, though this API allows us to request to
                // kill any process based on its PID, the kernel will
                // still impose standard restrictions on which PIDs you
                // are actually able to kill. Typically this means only
                // the process running your application and any additional
                // processes created by that app as shown here; packages
                // sharing a common UID will also be able to kill each
                // other's processes.
                Process.killProcess(pid);
                mCallbackText.setText("Killed service process.");
            } catch (RemoteException ex) {
                // Recover gracefully from the process hosting the
                // server dying.
                // Just for purposes of the sample, put up a notification.
                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 will  
     * NOT be running in our main thread like most other things -- so,  
     * 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);  
        }  
    }  
};  
}
```

Application Fundamentals

署名：译言 biAji

链接：<http://article.yeeyan.org/view/37503/34036>

版本：Android 2.3 r1

整理：农民伯伯

声明

本文档转载并整理自译言：[Android 开发指南 1——应用程序基础](#)。

原文

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

应用程序基础(Application Fundamentals)

Android 应用程序使用 Java 做为开发语言。[aapt](#) 工具把编译后的 Java 代码连同其它应用程序需要的数据和资源文件一起打包到一个 Android 包文件中，这个文件使用.apk 做为扩展名，它是分发应用程序并安装到移动设备的媒介，用户只需下载并安装此文件到他们的设备。单一.apk 文件中的所有代码被认为是一个应用程序。

从很多方面来看，每个 Android 应用程序都存在于它自己的世界之中：

- 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。
- 每个进程都运行于自己的 Java 虚拟机（VM）中。所以应用程序代码实际上与其它应用程序的代码是隔绝的。
- 默认情况下，每个应用程序均被赋予一个唯一的 Linux 用户 ID，并加以权限设置，使得应用程序的文件仅对这个用户、这个应用程序可见。当然，也有其它的方法使得这些文件同样能为别的应用程序所访问。

使两个应用程序共有同一个用户 ID 是可行的，这种情况下他们可以看到彼此的文件。从系统资源维护的角度来看，拥有同一个 ID 的应用程序也将在运行时使用同一个 Linux 进程，以及同一个虚拟机。

应用程序组件(Application Components)

Android 的核心功能之一就是一个应用程序可以使用其它应用程序的元素（如果那个应用程序允许的话）。比如说，如果你的应用程序需要一个图片卷动列表，而另一个应用程序已经开发了一个合用的而又允许别人使用的话，你可以直接调用那个卷动列表来完成工作，而不用自己再开发一个。你的应用程序并没有吸纳 或链接其它应用程序的代码，它只是在有需求的时候启动了其它应用程序的那个功能部分。

为达到这个目的，系统必须在一个应用程序的一部分被需要时启动这个应用程序，并将那个部分的 Java 对象实例化。与在其它系统上的应用程序不同，Android 应用程序没有为应用准备一个单独的程序入口（比如说，没有 main() 方法），而是为系统依照需求实例化提供了基本的组件。共有四种组件类型：

Activities

- Activity 是为用户操作而展示的可视化用户界面。比如说，一个 activity 可以展示一个菜单项列表供用户选择，或者显示一些包含说明的照片。一个短消息应用程序可

以包括一个用于显示做为发送对象的联系人的列表的 `activity`, 一个给选定的联系人写短信的 `activity` 以及翻阅以前的短信和改变设置的 `activity`。尽管它们一起组成了一个内聚的用户界面, 但其中每个 `activity` 都与其它的保持独立。每个都是以 [Activity](#) 类为基类的子类实现。

- 一个应用程序可以只有一个 `activity`, 或者, 如刚才提到的短信应用程序那样, 包含很多个。每个 `activity` 的作用, 以及其数目, 自然取决于应用 程序及其设计。一般情况下, 总有一个应用程序被标记为用户在应用程序启动的时候第一个看到的。从一个 `activity` 转向另一个的方式是靠当前的 `activity` 启动下一个。
- 每个 `activity` 都被给予一个默认的窗口以进行绘制。一般情况下, 这个窗口是满屏的, 但它也可以是一个小的位于其它窗口之上的浮动窗口。一个 `activity` 也可以使用超过一个的窗口——比如, 在 `activity` 运行过程中弹出的一个供用户反应的小对话框, 或是当用户选择了屏幕上特定项目后显 示的必要信息。
- 窗口显示的可视内容是由一系列视图构成的, 这些视图均继承自 [View](#) 基类。每个视图均控制着窗口中一块特定的矩形空 间。父级视图包含并组织它子视图的布局。叶节点视图 (位于视图层次最底端) 在它们控制的矩形中进行绘制, 并对用户对其直接操作做出响应。所以, 视图是 `activity` 与用户进行交互的界面。比如说, 视图可以显示一个小图片, 并在用户指点它的时候产生动作。Android 有很多既定的视图供用户直接使 用, 包括按钮、文本域、卷轴、菜单项、复选框等等。
- 视图层次是由 [Activity.setContentView\(\)](#) 方法放入 `activity` 的窗口之中的。上下文视图是位于视图层次根位置的视图对象。(参见用户界面章节获取关于视图及层次的更多信息。)

服务(Services)

- 服务没有可视化的用户界面, 而是在一段时间内在后台运行。比如说, 一个服务可以在用户做其它事情的时候在后台播放背景音乐、从网络上获取一些数据或者计算一些东西并提供给需要这个运算结果的 `activity` 使用。每个服务都继承自 [Service](#) 基类。
- 一个媒体播放器播放播放列表中的曲目是一个不错的例子。播放器应用程序可能有一个或多个 `activity` 来给用户选择歌曲并进行播放。然而, 音乐播放这个 任务本身不应该为任何 `activity` 所处理, 因为用户期望在他们离开播放器应用程序而开始做别的事情时, 音乐仍在继续播放。为达到这个目的, 媒体播放器 `activity` 应该启用一个运行于后台的服务。而系统将在这个 `activity` 不再显示于屏幕之后, 仍维持音乐播放服务的运行。
- 你可以连接至 (绑定) 一个正在运行的服务 (如果服务没有运行, 则启动之)。连接之后, 你可以通过那个服务暴露出来的接口与服务进行通讯。对于音乐服务来说, 这个接口可以允许用户暂停、回退、停止以及重新开始播放。
- 同如 `activity` 和其它组件一样, 服务运行于应用程序进程的主线程内。所以它不会对其它组件或用户界面有任何干扰, 它们一般会派生一个新线程来进行一些耗时任 务 (比如音乐回放)。参见下述 [进程和线程\(Processes and Threads\)](#)。

广播接收器(Broadcast receivers)

- 广播接收器是一个专注于接收广播通知信息, 并做出对应处理的组件。很多广播是源自于系统代码的——比如, 通知时区改变、电池电量低、拍摄了一张照片或者用户改变了语言选项。应用程序也可以进行广播——比如说, 通知其它应用程序一些 数据下载完成并处于可用状态。

- 应用程序可以拥有任意数量的广播接收器以对所有它感兴趣的通知信息予以响应。所有的接收器均继承自 [BroadcastReceiver](#) 基类。
- 广播接收器没有用户界面。然而，它们可以启动一个 `activity` 来响应它们收到的信息，或者用 [NotificationManager](#) 来通知用户。通知可以用很多种方式来吸引用户的注意力——闪动背灯、震动、播放声音等等。一般来说是在状态栏上放一个持久的图标，用户可以打开它并获取消息。

内容提供者(Content providers)

- 内容提供者将一些特定的应用程序数据供给其它应用程序使用。数据可以存储于文件系统、SQLite 数据库或其它方式。内容提供者继承于 [ContentProvider](#) 基类，为其它应用程序取用和存储它管理的数据实现了一套标准方法。然而，应用程序并不直接调用这些方法，而是使用一个 [ContentResolver](#) 对象，调用它的方法作为替代。[ContentResolver](#) 可以与任意内容提供者进行会话，与其合作来对所有相关交互通信进行管理。
- 参阅独立的[内容提供者 Content Providers](#) 章节获得更多关于使用内容提供者的内容。

每当出现一个需要被特定组件处理的请求时，Android 会确保那个组件的应用程序进程处于运行状态，或在必要的时候启动它。并确保那个相应组件的实例的存在，必要时会创建那个实例。

激活组件 Activating components: intents

当接收到 [ContentResolver](#) 发出的请求后，内容提供者被激活。而其它三种组件——`activity`、服务和广播接收器被一种叫做 `intent` 的异步消息所激活。`intent` 是一个保存着消息内容的 [Intent](#) 对象。对于 `activity` 和服务来说，它指明了请求的操作名称以及作为操作对象的数据的 URI 和其它一些信息。比如说，它可以承载对一个 `activity` 的请求，让它为用户显示一张图片，或者让用户编辑一些文本。而对于广播接收器而言，`Intent` 对象指明了声明的行为。比如，它可以对所有感兴趣的对象声 明照相按钮被按下。

对于每种组件来说，激活的方法是不同的：

- 通过传递一个 `Intent` 对象至 [Context.startActivity\(\)](#) 或 [Activity.startActivityForResult\(\)](#) 以载入(或指定新工作给)一个 `activity`。相应的 `activity` 可以通过调用 [getIntent\(\)](#) 方法来查看激活它的 `intent`。Android 通过调用 `activity` 的 [onNewIntent\(\)](#) 方法来传递给它继发的 `intent`。

一个 `activity` 经常启动了下一个。如果它期望它所启动的那个 `activity` 返回一个结果，它会以调用 [startActivityForResult\(\)](#) 来取代 [startActivity\(\)](#)。比如说，如果它启动了另外一个 `activity` 以使用户挑选一张照片，它也许想知道哪张照片被选中了。结果将会被封装在一个 `Intent` 对象中，并传递给发出调用的 `activity` 的 [onActivityResult\(\)](#) 方法。

- 通过传递一个 `Intent` 对象至 [Context.startService\(\)](#) 将启动一个服务(或给予正在运行的服务以一个新的指令)。Android 调用服务的 [onStart\(\)](#) 方法并将 `Intent` 对象传递给它。

与此类似，一个 `Intent` 可以被调用组件传递给 [Context.bindService\(\)](#) 以获取一个正在运行的目标服务的连接。这个服务会经由 [onBind\(\)](#) 方法的调用获取这个 `Intent` 对象(如果服务尚未启动，[bindService\(\)](#) 会先启动它)。比如说，一个 `activity` 可以连接至前述的音乐回放服务，并提供给用户一个可操作的(用户界面)以对回

放进行控制。这个 activity 可以调用 `bindService()` 来建立连接，然后调用服务中定义的对象来影响回放。

后面一节：[远程方法调用\(Remote procedure calls\)](#)将更详细的阐明如何绑定至服务。

- 应用程序可以凭借将 Intent 对象传递给 `Context.sendBroadcast()`，`Context.sendOrderedBroadcast()`，以及 `Context.sendStickyBroadcast()`和其它类似方法来产生一个广播。Android 会调用所有对此广播有兴趣的广播接收器的 `onReceive()`方法，将 intent 传递给它们。

欲了解更多 intent 消息的信息，请参阅独立章节 [Intent 和 Intent 滤过器\(Intents and Intent Filters\)](#)。

关闭组件(Shutting down components)

内容提供者仅在响应 ContentResolver 提出请求的时候激活。而一个广播接收器仅在响应广播信息的时候激活。所以，没有必要去显式的关闭这些组件。

而 activity 则不同，它提供了用户界面，并与用户进行会话。所以只要会话依然持续，哪怕对话过程暂时停顿，它都会一直保持激活状态。与此相似，服务也会在很长一段时间内保持运行。所以 Android 为关闭 activity 和服务提供了一系列的方法。

- 可以通过调用它的 `finish()`方法来关闭一个 activity。一个 activity 可以通过调用另外一个是 activity (它用 `startActivityForResult()` 启动的) 的 `finishActivity()`方法来关闭它。
- 服务可以通过调用它的 `stopSelf()`方法来停止，或者调用 `Context.stopService()`。

系统也会在组件不再被使用的时候或者 Android 需要为活动组件声明更多内存的时候关闭它。后面的[组件的生命周期](#)一节，将对这种可能及附属情况进行更详细的讨论。

manifest 文件(The manifest file)

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

这个 manifest 文件以 XML 作为结构格式，而且对于所有应用程序，都叫做 `AndroidManifest.xml`。为声明一个应用程序组件，它还会 做很多额外工作，比如指明应用程序所需链接到的库的名称（除了默认的 Android 库之外）以及声明应用程序期望获得的各种权限。

但 manifest 文件的主要功能仍然是向 Android 声明应用程序的组件。举例说明，一个 activity 可以如下声明：

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

[`<activity>`](#)元素的 name 属性指定了实现了这个 activity 的 [Activity](#) 的子类。icon 和 label 属性指向了包含展示给用户的此 activity 的图标和标签的资源文件。

其它组件也以类似的方法声明——[`<service>`](#) 元素用于声明服务，[`<receiver>`](#) 元素用于声明广播接收器，而 [`<provider>`](#) 元素用于声明内容提供者。`manifest` 文件中未进行声明的 activity、服务以及内容提供者将不为系统所见，从而也就不会被运行。然而，广播接收器既可以在 `manifest` 文件中声明，也可以在代码中进行动态的创建，并以调用 [`Context.registerReceiver\(\)`](#) 的方式注册至系统。

欲更多了解如何为你的应用程序构建 `manifest` 文件，请参阅 [AndroidManifest.xml 文件](#) 一章。

Intent 过滤器(Intent filters)

`Intent` 对象可以被显式的指定目标组件。如果进行了这种指定，Android 会找到这个组件（依据 `manifest` 文件中的声明）并激活它。但如果 `Intent` 没有进行显式的指定，Android 就必须为它找到对于 `intent` 来说最合适的组件。这个过程是通过比较 `Intent` 对象和所有可能对象的 `intent` 过滤器完成的。组件的 `intent` 过滤器会告知 Android 它所能处理的 `intent` 类型。如同其它相对于组件很重要的信息一样，这些是在 `manifest` 文件中进行声明的。这里是上面实例的一个扩展，其中加入了针对 `activity` 的两个 `intent` 过滤器声明：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . . >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

示例中的第一个过滤器——`action "android.intent.action.MAIN"` 和类别 `"android.intent.category.LAUNCHER"` 的组合——是通常具有的。它标明了这个 `activity` 将在应用程序加载器中显示，就是用户在设备上看到的可供加载的应用程序列表。换句话说，这个 `activity` 是应用程序的入口，是用户选择运行这个应用程序后所见到的第一个 `activity`。

第二个过滤器声明了这个 `activity` 能被赋予一种特定类型的数据。

组件可以拥有任意数量的 `intent` 过滤器，每个都会声明一系列不同的能力。如果它没有包含任何过滤器，它将只能被显式声明了目标组件名称的 `intent` 激活。

对于在代码中创建并注册的广播接收器来说，`intent` 过滤器将被直接以 [IntentFilter](#) 对象实例化。其它过滤器则在 `manifest` 文件中设置。

欲获得更多 `intent` 过滤器的信息，请参阅独立章节：[Intent 和 Intent 过滤器](#)。

Activity 和任务(Activities and Tasks)

如前所述，一个 `activity` 可以启动另外一个，甚至包括与它不处于同一应用程序之中的。举个例子说，假设你想让用户看到某个地方的街道地图。而已经存在一个具有此功能的 `activity` 了，那么你的 `activity` 所需要做的工作就是把请求信息放到一个 `Intent` 对象里面，并把它传递给 `startActivity()`。于是地图浏览器就会显示那个地图。而当用户按下 `BACK` 键的时候，你的 `activity` 又会再一次的显示在屏幕上。

对于用户来说，这看起来就像是地图浏览器是你 `activity` 所在的应用程序中的一个组成部分，其实它是在另外一个应用程序中定义，并运行在那个应用程序的进程之中的。Android 将这两个 `activity` 放在同一个任务中 来维持一个完整的用户体验。简单的说，任务就是用户所体验到的“应用程序”。它是安排在一个堆栈中的一组相关的 `activity`。堆栈中的根 `activity` 就是启动了这整个任务的那个——一般情况下，它就是用户在应用程序加载器中所选择的。而堆栈最上方的 `activity` 则是当前运行的—— 用户直接对其进行操作的。当一个 `activity` 启动另外-一个的时候，新的 `activity` 就被压入堆栈，并成为当前运行的 `activity`。而前一个 `activity` 仍保持在堆栈之中。当用户按下 `BACK` 键的时候，当前 `activity` 出栈，而前一个恢复为当前运行的 `activity`。

堆栈中保存的其实是对象，所以如果发生了诸如需要多个地图浏览器的情况，就会使得一个任务中出现多个同一 `Activity` 子类的实例同时存在，堆栈会为每个实例单独开辟一个入口。堆栈中的 `Activity` 永远不会重排，只会压入或弹出。

任务其实就是 `activity` 的堆栈，而不是 `manifest` 文件中的一个类或者元素。所以你无法撇开 `activity` 而为一个任务设置一个值。而事实上 整个任务使用的值是在根 `activity` 中设置的。比如说，下一节我们会谈及“任务的 `affinity`”，从 `affinity` 中读出的值将会设置到任务的 根 `activity` 之中。

任务中的所有 `activity` 是作为一个整体进行移动的。整个的任务（即 `activity` 堆栈）可以移到前台，或退至后台。举个例子说，比如当前任务在堆 栈中存有四个 `activity`——三个在当前 `activity` 之下。当用户按下 `HOME` 键的时候，回到了应用程序加载器，然后选择了一个新的应用程序（也 就是一个新任务）。则当前任务遁入后台，而新任务的根 `activity` 显示出来。然后，过了一小会儿，用户再次回到了应用程序加载器而又选择了前一个应用程序（上一个任务）。于是那个任务，带着它堆栈中所有的四个 `activity`，再一次的到了前台。当用户按下 `BACK` 键的时候，屏幕不会显示出用户刚才离开的 `activity`（上一个任务的根 `activity`）。取而代之，当前任务的堆栈中最上面的 `activity` 被弹出，而同一任务中的上一个 `activity` 显示了出来。

上述的种种即是 `activity` 和任务的默认行为模式。但是有一些方法可以改变所有这一切。`activity` 和任务的联系、任务中 `activity` 的行为 方式都被启动那个 `activity` 的 `Intent` 对象中设置的一系列标记和 `manifest` 文件中那个 `activity` 中的`<activity>`元素的系列属性之间的交互所控制。无论是请求发出者和回应者在这里都拥有话语权。

我们刚才所说的这些关键 `Intent` 标记如下：

```
FLAG_ACTIVITY_NEW_TASK  
FLAG_ACTIVITY_CLEAR_TOP  
FLAG_ACTIVITY_RESET_TASK_IF_NEEDED  
FLAG_ACTIVITY_SINGLE_TOP
```

而关键的`<activity>`属性是：

```
taskAffinity  
launchMode
```

```
allowTaskReparenting  
clearTaskOnLaunch  
alwaysRetainTaskState  
finishOnTaskLaunch
```

接下来的一节会描述这些标记以及属性的作用，它们是如何互相影响的，以及控制它们的使用时必须考虑到的因素。

任务共用性和新任务 Affinities and new tasks

默认情况下，一个应用程序中的 activity 相互之间会有一种 Affinity——也就是说，它们首选都归属于一个任务。然而，可以在<activity>元素中把每个 activity 的 taskAffinity 属性设置为一个独立的 affinity。于是在不同的应用程序中定义的 activity 可以享有同一个 affinity，或者在同一个应用程序中定义的 activity 有着不同的 affinity。affinity 在两种情况下生效：当加载 activity 的 Intent 对象包含了 FLAG_ACTIVITY_NEW_TASK 标记，或者当 activity 的 allowTaskReparenting 属性设置为“true”。

FLAG_ACTIVITY_NEW_TASK 标记

如前所述，在默认情况下，一个新 activity 被另外一个调用了 startActivity() 方法的 activity 载入了任务之中。并压入了调用者所在的堆栈。然而，如果传递给 startActivity() 的 Intent 对象包含了 FLAG_ACTIVITY_NEW_TASK 标记，系统会为新 activity 安排另外的任务。一般情况下，如同标记所暗示的那样，这会是一个新任务。然而，这并不是必然的。如果已经存在了一个与新 activity 有着同样 affinity 的任务，则 activity 会载入那个任务之中。如果没有，则启用新任务。

allowTaskReparenting 属性

如果一个 activity 将 allowTaskReparenting 属性设置为“true”。它就可以从初始的任务中转移到与其拥有同一个 affinity 并转向前台的任务之中。比如说，一个旅行应用程序中包含的预报所选城市的天气情况的 activity。它与这个应用程序中其它的 activity 拥有同样的 affinity（默认的 affinity）而且允许重定父级。你的另一个 activity 启动了天气预报，于是它就会与这个 activity 共处与同一任务之中。然而，当那个旅行应用程序再次回到前台的时候，这个天气预报 activity 就会被再次安排到原先的任务之中并显示出来。

如果在用户的角度看来，一个.apk 文件中包含了多于一个的“应用程序”，你可能会想要为它们所辖的 activity 安排不一样的 affinity。

加载模式(Launch modes)

<activity>元素的 launchMode 属性可以设置四种不同的加载模式：

```
"standard" (默认模式)  
"singleTop"  
"singleTask"  
"singleInstance"
```

这些模式之间的差异主要体现在四个方面：

- 哪个任务会把持对 intent 做出响应的 activity。对“standard”和“singleTop”模式而言，是产生 intent（并调用 startActivity()）的任务——除非 Intent 对象包含 FLAG_ACTIVITY_NEW_TASK 标记。而在这种情况下，如同上面 Affinities 和新任务一节所述，会是另外一个任务。

相反，对“singleTask”和“singleInstance”模式而言，activity 总是位于任务的根部。正是它们定义了一个任务，所以它们绝不会被载入到其它任务之中。

- **activity 是否可以存在多个实例。**一个“standard”或“singleTop”的 activity 可以被多次初始化。它们可以归属于多个任务，而一个任务也可以拥有同一 activity 的多个实例。

相反，对“singleTask”和“singleInstance”的 activity 被限定于只能有一个实例。

因为这些 activity 都是任务的起源，这种限制意味着在一个设备中同一时间只允许存在一个任务的实例。

- **在实例所在的任务中是否会有别的 activity。**一个“singleInstance”模式的 activity 将会是它所在的任务中唯一的 activity。如果它启动了别的 activity，那个 activity 将会依据它自己的加载模式加载到其它的任务中去——如同在 intent 中设置了 FLAG_ACTIVITY_NEW_TASK 标记一样的效果。在其它方面，“singleInstance”模式的效果与“singleTask”是一样的。

剩下的三种模式允许一个任务中出现多个 activity。“singleTask”模式的 activity 将是任务的根 activity，但它可以启动别的 activity 并将它们置入所在的任务中。

“standard”和“singleTop”activity 则可以在堆栈的任意位置出现。

- **是否要载入新的类实例以处理新的 intent。**对默认的“standard”模式来说，对于每个新 intent 都会创建一个新的实例以进行响应，每个实例仅处理一个 intent。

“singleTop”模式下，如果 activity 位于目的任务堆栈的最上面，则重用目前现存的 activity 来处理新的 intent。如果它不是在堆栈顶部，则不会发生重用。而是创建一个新实例来处理新的 intent 并将其推入堆栈。

举例来说，假设一个任务的堆栈由根 activityA 和 activity B、C 和位于堆栈顶部的 D 组成，即堆栈 A-B-C-D。一个针对 D 类型的 activity 的 intent 抵达的时候，如果 D 是默认的“standard”加载模式，则创建并加载一个新的类实例，于是堆栈变为 A-B-C-D-D。然而，如果 D 的载入模式为“singleTop”，则现有的实例会对新 intent 进行处理（因为它位于堆栈顶部）而堆栈保持 A-B-C-D 的形态。

换言之，如果新抵达的 intent 是针对 B 类型的 activity，则无论 B 的模式是“standard”还是“singleTop”，都会加载一个新的 B 的实例（因为 B 不位于堆栈的顶部），而堆栈的顺序变为 A-B-C-D-B。

如前所述，“singleTask”或“singleInstance”模式的 activity 永远不会有多个实例。所以实例将处理所有新的 intent。一个“singleInstance”模式的 activity 永远保持在堆栈的顶部（因为它是那个堆栈中唯一的一个 activity），所以它一直坚守在处理 intent 的岗位上。然而，对一个“singleTask”模式的 activity 来说，它上面可能有，也可能没有别的 activity 和它处于同一堆栈。在有的情况下，它就不在能够处理 intent 的位置上，则那个 intent 将被舍弃。（即便在 intent 被舍弃的情况下，它的抵达仍将使这个任务切换至前台，并一直保留）

当一个现存的 activity 被要求处理一个新的 intent 的时候，会调用 [onNewIntent\(\)](#) 方法来将 intent 对象传递至 activity。（启动 activity 的原始 intent 对象可以通过调用 [getIntent\(\)](#) 方法获得。）

请注意，当一个新的 activity 实例被创建以处理新的 intent 的时候，用户总可以按下 BACK 键来回到前面的状态（回到前一个 activity）。但当使用现存的 activity 来处理新 intent 的时候，用户是不能靠按下 BACK 键回到当这个新 intent 抵达之前的状态的。

想获得更多关于加载模式的内容，请参阅 [<activity>](#) 元素的描述。

清理堆栈(Clearing the stack)

如果用户离开一个任务很长一段时间，系统会清理该任务中除了根 activity 之外的所有 activity。当用户再次回到这个任务的时候，除了只剩下初始化 activity 尚存之外，其余都跟用户上次离开它的时候一样。这样做的原因是：在一段时间之后，用户再次回到一个任务的时候，他们更期望放弃他们之前的所作所为，做些新的事情。

这些属于默认行为，另外，也存在一些 activity 的属性用以控制并改变这些行为：

[alwaysRetainTaskState](#) 属性

如果一个任务的根 activity 中此属性设置为“true”，则上述默认行为不会发生。任务将在很长的一段时间内保留它堆栈内的所有 activity。

[clearTaskOnLaunch](#) 属性

如果一个任务的根 activity 中此属性设置为“true”，则每当用户离开这个任务和返回它的时候，堆栈都会被清空至只留下 rootactivity。换句话说，这是 [alwaysRetainTaskState](#) 的另一个极端。哪怕仅是过了一小会儿，用户回到任务时，也是见到它的初始状态。

[finishOnTaskLaunch](#) 属性

这个属性与 [clearTaskOnLaunch](#) 属性相似，但它仅作用于单个的 activity，而不是整个的 task。而且它可以使任意 activity 都被清理，甚至根 activity 也不例外。当它设置为“true”的时候，此 activity 仅做为任务的一部分存在于当前回话中，一旦用户离开并再次回到这个任务，此 activity 将不复存在。

此外，还有别的方式从堆栈中移除一个 activity。如果一个 intent 对象包含 [FLAG_ACTIVITY_CLEAR_TOP](#) 标记，而且目标任务的堆栈中已经存在了一个能够响应此 intent 的 activity 类型的实例。则这个实例之上的所有 activity 都将被清理以使它位于堆栈的顶部来对 intent 做出响应。如果此时指定的 activity 的加载模式为“standard”，则它本身也会从堆栈中移除，并加载一个新的实例来处理到来的 intent。这是因为加载模式为“standard”的 activity 总会创建一个新实例来处理新的 intent。

[FLAG_ACTIVITY_CLEAR_TOP](#) 与 [FLAG_ACTIVITY_NEW_TASK](#) 经常合并使用。这时，这些标记提供了一种定位其它任务中现存的 activity 并将它们置于可以对 intent 做出响应的位置的方法。

启动任务(Starting tasks)

当一个 activity 被指定一个“`android.intent.action.MAIN`”做为动作，以及“`android.intent.category.LAUNCHER`”做为类别的 intent 过滤器之后（在前述 [intent 过滤器](#) 一节中已经有了这个示例），它就被设置为一个任务的入口点。这样的过滤器设置会在应用程序加载器中为此 activity 显示一个图标和标签，以供用户加载任务或加载之后在任意时间回到这个任务。

第二个能力相当重要：用户必须可以离开一个任务，并在一段时间后返回它。出于这个考虑，加载模式被设定为“singleTask”和“singleInstance”的 activity 总是会初始化一个新任务，这样的 activity 仅能用于指定了一个 `MAIN` 和 `LAUNCHER` 过滤器的情况下。我们来举例说明如果没指定过滤器的情况下会发生的事情：一个 intent 加载了一个“singleTask”的 activity，初始化了一个新任务，用户在这个任务中花费了一些时间来完成工作。然后用户按下了 `HOME` 键。于是任务被要求转至后台并被主屏幕所掩盖。因为它并没有在应用程序加载器中显示图标，这将导致用户无法再返回它。

类似的困境也可由 [FLAG_ACTIVITY_NEW_TASK](#) 标记引起。如果此标记使一个 activity 启动了一个新任务继而用户按下了 `HOME` 键离开了它，则用户必须要有一些方法再次回到这个任务。一些实体（诸如通知管理器）总是在另外的任务中启动新 activity，而不是做为它们自己的一部分，所以它们总是将 [FLAG_ACTIVITY_NEW_TASK](#) 标记包含在 intent 里面并传递

给 `startActivity()`。如果你写了一个能被外部实体使用这个标记调用的 `activity`, 你必须注意要给用户留一个返回这个被外部实体启动的任务的方法。

当你不想让用户再次返回一个 `activity` 的情况下, 可以将 `<activity>` 元素的 `finishOnTaskLaunch` 设置为“`true`”。参见前述[清理堆栈](#)。

进程和线程(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。

进程(Processes)

组件运行所在的进程由 `manifest` 文件所控制。组件元素——`<activity>`, `<service>`, `<receiver>` 和 `<provider>`——都有一个 `process` 属性来指定组件应当运行于哪个进程之内。这些属性可以设置为使每个组件运行于它自己的进程之内, 或一些组件共享一个进程而其余的组件不这么做。它们也可以 设置为令不同应用程序的组件在一个进程中运行——使应用程序的组成部分共享同一个 Linux 用户 ID 并赋以同样的权限。`<application>` 元素也有一个 `process` 属性, 以设定所有组件的默认值。

所有的组件实例都位于特定进程的主线程内, 而对这些组件的系统调用也将由那个线程进行分发。一般不会为每个实例创建线程。因此, 某些方法总是运行在进程的主线程内, 这些方法包括诸如 [View.onKeyDown\(\)](#) 这样报告用户动作以及后面 [组件生命周期](#) 一节所要讨论的生命周期通告的。这意味着组件在被系统调用的时候, 不应该施行长时间的抑或阻塞的操作(诸如网络相关操作或是循环计算), 因为这将阻塞同样位于这个进程的其它组件的运行。你应该如同下面[线程](#)一节所叙述的那样, 为这些长时间操作衍生出一个单独的线程进行处理。

在可用内存不足而又有一个正在为用户进行服务的进程需要更多内存的时候, Android 有时候可能会关闭一个进程。而在这个进程中运行着的应用程序也因此被销毁。当再次出现需要它们进行处理的工作的时候, 会为这些组件重新创建进程。

在决定结束哪个进程的时候, Android 会衡量它们对于用户的相对重要性。比如说, 相对于一个仍有用户可见的 `activity` 的进程, 它更有可能去关闭一个其 `activity` 已经不为用户所见的进程。也可以说, 决定是否关闭一个进程主要依据在那个进程中运行的组件的状态。这些状态将在后续的一节[组件生命周期](#) 中予以说明。

线程(Threads)

尽管你可以把你的应用程序限制于一个单独的进程中, 有时, 你仍然需要衍生出一个线程以处理后台任务。因为用户界面必须非常及时的对用户操作做出响应, 所以, 控管 `activity` 的线程不应用于处理一些诸如网络下载之类的耗时操作。所有不能在瞬间完成的任务都应安排到不同的线程中去。

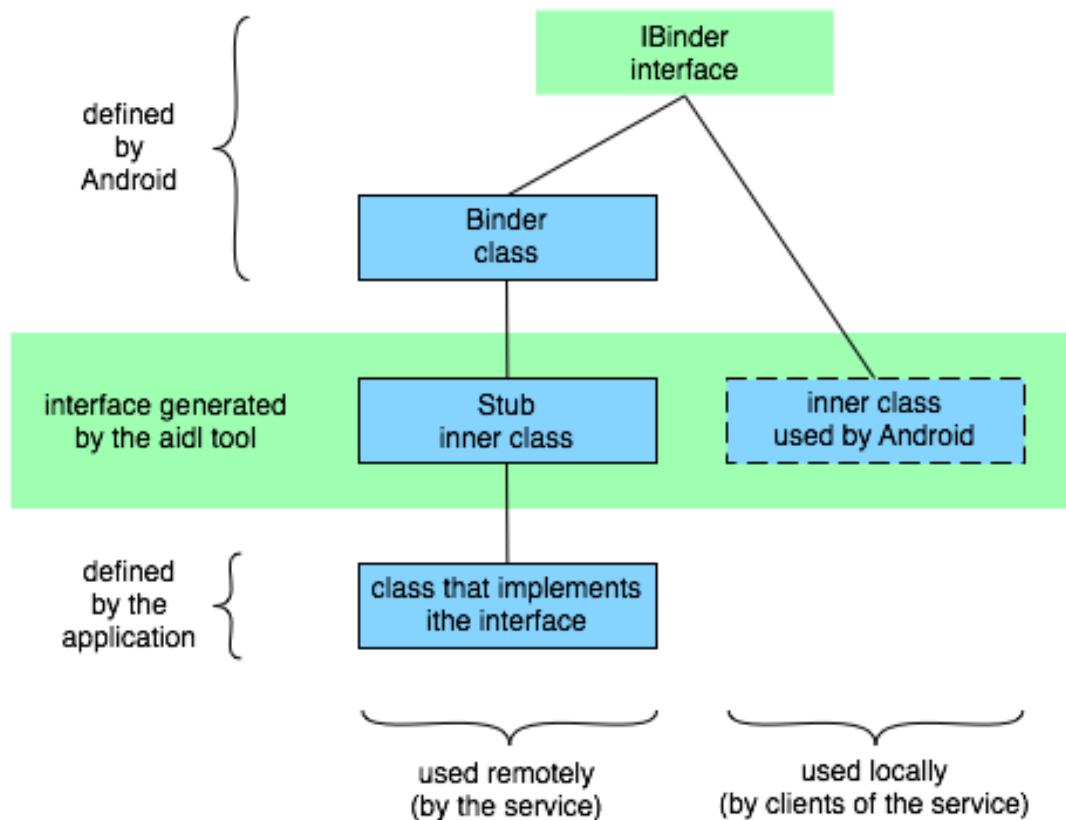
线程在代码中是以标准 Java [Thread](#) 对象创建的。Android 提供了很多便于管理线程的类:[Looper](#) 用于在一个线程中运行一个消息循环, [Handler](#) 用于处理消息, [HandlerThread](#) 用于使用一个消息循环启用一个线程。

远程方法调用(Remote procedure calls)

Android 有一个轻量级的远程方法调用（RPC）机制：即在本地调用一个方法，但在远程（其它的进程中）进行处理，然后将结果返回调用者。这将方法调用及其附属的数据以系统可以理解的方式进行分离，并将其从本地进程和本地地址空间传送至远程过程和远程地址空间，并在那里重新装配并对调用做出反应。返回的结果将以相反的方向进行传递。Android 提供了完成这些工作所需的所有的代码，以使你可以集中精力来实现 RPC 接口本身。

RPC 接口可以只包括方法。即便没有返回值，所有方法仍以同步的方式执行（本地方法阻塞直至远程方法结束）。

简单的说，这套机制是这样工作的：一开始，你用简单的 IDL（界面描绘语言）声明一个你想要实现的 RPC 接口。然后用 [aidl](#) 工具为这个声明生成一个 Java 接口定义，这个定义必须对本地和远程进程都可见。它包含两个内部类，如下图所示：



内部类中有管理实现了你用 IDL 声明的接口的远程方法调用所需要的所有代码。两个内部类均实现了 [IBinder](#) 接口。一个用于系统在本地内部使用，你些的代码可以忽略它；另外一个，我们称为 **Stub**，扩展了 [Binder](#) 类。除了实现了 IPC 调用的内部代码之外，它还包括了你声明的 RPC 接口中的方法的声明。你应该如上图所示的那样写一个 **Stub** 的子类来实现这些方法。

一般情况下，远程过程是被一个服务所管理的（因为服务可以通知系统关于进程以及它连接到别的进程的信息）。它包含着 aidl 工具产生的接口文件和实现了 RPC 方法的 **Stub** 的子类。而客户端只需要包括 aidl 工具产生的接口文件。

下面将说明服务与其客户端之间的连接是如何建立的：

- 服务的客户端（位于本地）应该实现 [onServiceConnected\(\)](#) 和 [onServiceDisconnected\(\)](#) 方法。这样，当至远程服务的连接成功建立或者断开的时候，它们会收到通知。这样它们就可以调用 [bindService\(\)](#) 来设置连接。

- 而服务则应该实现 [onBind\(\)](#) 方法以接受或拒绝连接。这取决于它收到的 intent (intent 将传递给 bindService())。如果接受了连接，它会返回一个 Stub 的子类的实例。
- 如果服务接受了连接，Android 将会调用客户端的 onServiceConnected() 方法，并传递给它一个 IBinder 对象，它是由服务所管理的 Stub 的子类的代理。通过这个代理，客户端可以对远程服务进行调用。

线程安全方法(Thread-safe methods)

在一些情况下，你所实现的方法有可能会被多个线程所调用，所以它们必须被写成线程安全的。

对于我们上一节所讨论的 RPC 机制中的可以被远程调用的方法来说，这是必须首先考虑的。如果针对一个 IBinder 对象中实现的方法的调用源自这个 IBinder 对象所在的进程时，这个方法将会在调用者的线程中执行。然而，如果这个调用源自其它的进程，则这个方法将会在一个线程池中选出的线程中运行，这个线程池由 Android 加以管理，并与 IBinder 存在于同一进程中；这个方法不会在进程的主线程内执行。反过来说，一个服务的 onBind() 方法应为服务进程的主线程所调用，而实现了由 onBind() 返回的对象（比如说，一个实现了 RPC 方法的 Stub 的子类）的方法将为池中的线程所调用。因为服务可以拥有多于一个的客户端，而同一时间，也会有多个池中的线程调用同一个 IBinder 方法。因此 IBinder 方法必须实现为线程安全的。

类似的，一个内容提供者能接受源自其它进程的请求数据。尽管 ContentResolver 和 ContentProvider 类隐藏了交互沟通过程的管理细节，ContentProvider 会由 [query\(\)](#), [insert\(\)](#), [delete\(\)](#), [update\(\)](#) 和 [getType\(\)](#) 方法来相应这些请求，而这些方法也都是由那个内容提供者的进程中所包涵的线程池提供的，而不是进程的主线程本身。所以这些有可能在同一时间被很多线程调用的方法也必须被实现为线程安全的。

组件生命周期(Component Lifecycles)

应用程序组件有其生命周期——由 Android 初始化它们以相应 intent 直到这个实例被摧毁。在此之间，它们有时是激活的有时则相反。或者，如果它是一个 activity，则是可为用户所见或者不能。这一节讨论了 activity、服务以及广播接收器的生命周期，包括它们在生命周期中的状态、在状态之间转变时通知你的方法、以及当这些进程被关闭或实例被摧毁时，这些状态产生的效果。

Activity 生命周期(Activity lifecycle)

一个 activity 主要有三个状态：

- 当在屏幕前台时（位于当前任务堆栈的顶部），它是活跃或运行的状态。它就是相应用户操作的 activity。
- 当它失去焦点但仍然对用户可见时，它处于暂停状态。即是：在它之上有另外一个 activity。这个 activity 也许是透明的，或者未能完全遮蔽全屏，所以被暂停的 activity 仍对用户可见。暂停的 activity 仍然是存活状态（它保留着所有的状态和成员信息并连接至窗口管理器），但当系统处于极低内存的情况下，仍然可以杀死这个 activity。

- 如果它完全被另一个 `activity` 覆盖是，它处于 **停止**状态。它仍然保留所有的状态和成员信息。然而它不在为用户可见，所以它的窗口将被隐藏，如果其它地方需要内存，则系统经常会杀死这个 `activity`。

如果一个 `activity` 处于暂停或停止状态，系统可以通过要求它结束(调用它的 [finish\(\)](#) 方法)或直接杀死它的进程来将它驱出内存。当它再次为用户可见的时候，它只能完全重新启动并恢复至以前的状态。

当一个 `activity` 从这个状态转变到另一个状态时，它被以下列 `protected` 方法所通知：

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```

你可以重载所有这些方法以在状态改变时进行合适的工作。所有的 `activity` 都必须实现 [onCreate\(\)](#) 用以当对象第一次实例化时进行初始化设置。很多 `activity` 会实现 [onPause\(\)](#) 以提交数据变化或准备停止与用户的交互。

调用父类(Calling into the superclass)

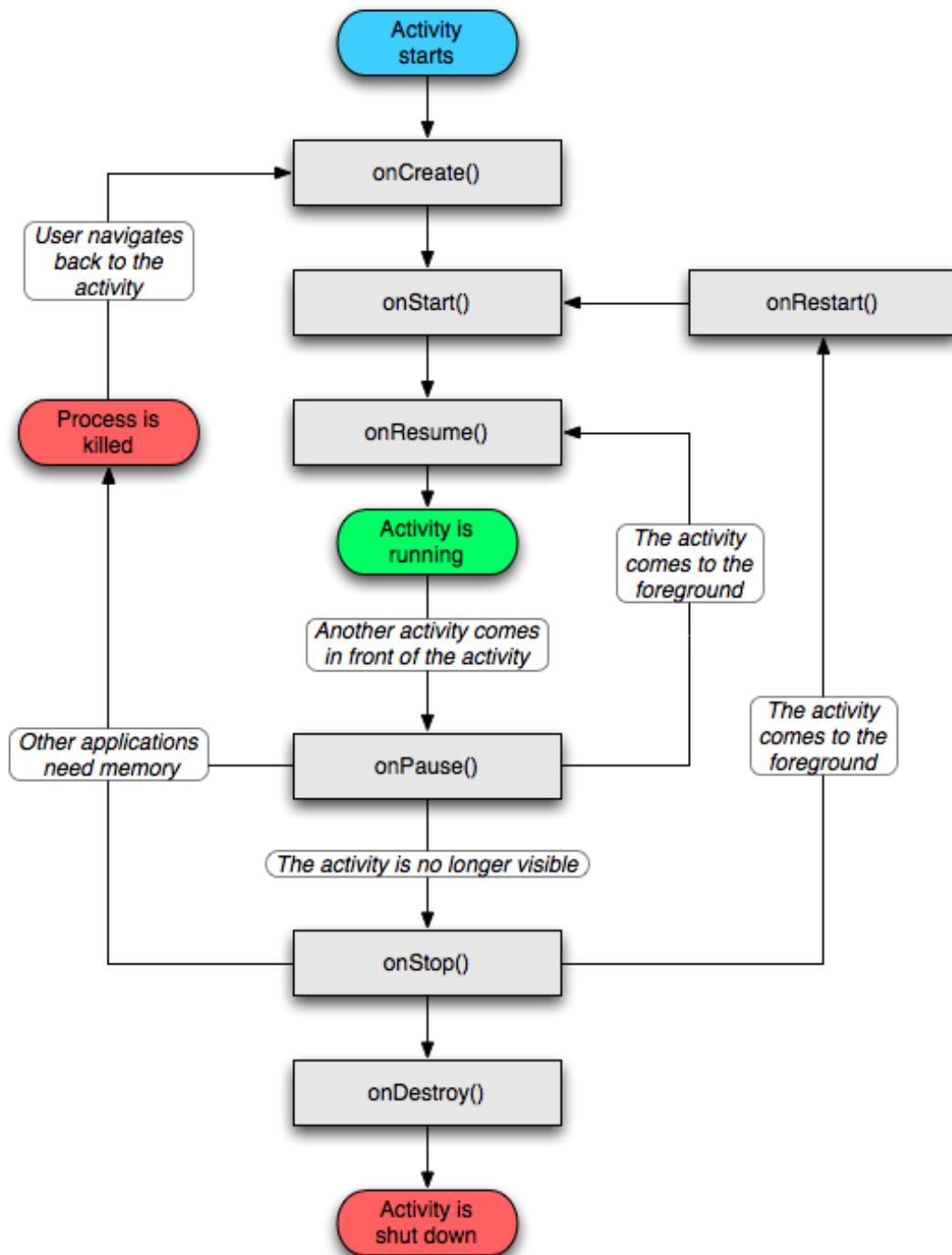
所有 `activity` 生命周期方法的实现都必须先调用其父类的版本。比如说：

```
protected void onPause() {
    super.onPause();
    ...
}
```

总得来说，这七个方法定义了一个 `activity` 完整的生命周期。实现这些方法可以帮助你监察三个嵌套的生命周期循环：

- 一个 `activity` **完整的生命周期** 自第一次调用 `onCreate()`开始，直至调用 `onDestroy()`为止。`activity` 在 `onCreate()`中设置所有“全局”状态以完成初始化，而在 `onDestroy()`中释放所有系统资源。比如说，如果 `activity` 有一个线程在后台运行以从网络上下载数据，它会以 `onCreate()`创建那个线程，而以 `onDestroy()`销毁那个线程。
- 一个 `activity` 的 **可视生命周期**自 `onStart()` 调用开始直到相应的 `onStop()`调用。在此期间，用户可以在屏幕上看到此 `activity`，尽管它也许并不是位于前台或者正在与用户做交互。在这两个方法中，你可以管控用来向用户显示这个 `activity` 的资源。比如说，你可以在 `onStart()` 中注册一个 `BroadcastReceiver` 来监控会影响到你 UI 的改变，而在 `onStop()` 中来取消注册，这时用户是无法看到你的程序显示的内容的。`onStart()` 和 `onStop()` 方法可以随着应用程序是否为用户可见而被多次调用。
- 一个 `activity` 的 **前台生命周期** 自 `onResume()` 调用起，至相应的 `onPause()`调用为止。在此期间，`activity` 位于前台最上面并与用户进行交互。`activity` 会经常在暂停和恢复之间进行状态转换——比如说当设备转入休眠状态或有新的 `activity` 启动时，将调用 `onPause()` 方法。当 `activity` 获得结果或者接收到新的 `intent` 的时候会调用 `onResume()` 方法。因此，在这两个方法中的代码应当是轻量级的。

下图展示了上述循环过程以及 `activity` 在这个过程之中历经的状态改变。着色的椭圆是 `activity` 可以经历的主要状态。矩形框代表了当 `activity` 在状态间发生改变的时候，你进行操作所要实现的回调方法。



下表详细描述了这些方法，并在 activity 的整个生命周期中定位了它们。

方法	描述	是否可被杀死(Killable?)	下一个
onCreate()	在 activity 第一次被创建的时候调用。这里是做所有初始化设置的地方——创建视图、绑定数据至列表等。如果曾经有状态记录（参阅后述 Saving Activity State ），则调用此方法时会传入一个包含着此 activity 以前状态的包对象做为参数。	否	<code>onStart()</code>

	数。 接下来始终遵循调用 <code>onStart()</code> 。		
<code>onRestart()</code>	在 <code>activity</code> 停止后，在再次启动之前被调用。 接下来始终遵循调用 <code>onStart()</code> 。	否	<code>onStart()</code>
<code>onStart()</code>	当 <code>activity</code> 正要变得为用户所见时被调用。 当 <code>activity</code> 转向前台时接下来调用 <code>onResume()</code> ，在 <code>activity</code> 变为隐藏时接下来调用 <code>onStop()</code> 。	否	<code>onResume()</code> 或 <code>onStop()</code>
<code>onResume()</code>	在 <code>activity</code> 开始与用户进行交互之前被调用。此时 <code>activity</code> 位于堆栈顶部，并接受用户输入。 接下来始终遵循调用 <code>onPause()</code> 。	否	<code>onPause()</code>
<code>onPause()</code>	当系统将要启动另一个 <code>activity</code> 时调用。此方法主要用来将未保存的变化进行持久化，停止类似动画这样耗费 CPU 的动作等。这一切动作应该在短时间内完成，因为下一个 <code>activity</code> 必须等到此方法返回后才会继续。 当 <code>activity</code> 重新回到前台时接下来调用 <code>onResume()</code> 。当 <code>activity</code> 变为用户不可见时接下来调用 <code>onStop()</code> 。	是	<code>onResume()</code> 或 <code>onStop()</code>
<code>onStop()</code>	当 <code>activity</code> 不再为用户可见时调用此方法。这可能发生在它被销毁或者另一个 <code>activity</code> （可能是现存的或者是新的）回到运行状态并覆盖了它。 如果 <code>activity</code> 再次回到前台跟用户交互则接下来调用 <code>onRestart()</code> ，如果关闭 <code>activity</code> 则接下来调用 <code>onDestroy()</code> 。	是	<code>onRestart()</code> or <code>onDestroy()</code>
<code>onDestroy()</code>	在 <code>activity</code> 销毁前调用。这是 <code>activity</code> 接收的最后一个调用。这可能发生在 <code>activity</code> 结束（调用了它的 <code>finish()</code> 方法）或者因为系统需要空间所以临时的销毁了此 <code>activity</code> 的实例时。你可以用 <code>isFinishing()</code> 方法来区分这两种情况。	是	无

请注意上表中**可被杀死**一列。它标示了在方法返回后，还没执行 `activity` 的其余代码的任意时间里，系统是否可以杀死包含此 `activity` 的进程。三个方法（`onPause()`、`onStop()` 和 `onDestroy()`）被标记为“是”。`onPause()` 是三个中的第一个，它也是唯一一个在进程被杀死之前必然会调用的方法——`onStop()` 和 `onDestroy()` 有可能不被执行。因此你应该用 `onPause()` 来将所有持久性数据（比如用户的编辑结果）写入存储之中。

在**可被杀死**一列中标记为“否”的方法在它们被调用时将保护 `activity` 所在的进程不会被杀死。所以只有在 `onPause()` 方法返回后到 `onResume()` 方法被调用时，一个 `activity` 才处于可被杀死的状态。在 `onPause()` 再次被调用并返回之前，它不会被系统杀死。

如后面一节[进程和生命周期](#)所述，即使是在这里技术上没有被定义为“可杀死”的 activity 仍然有可能被系统杀死——但这仅会发生在实在没有其它方法的极端情况之下。

保存 activity 状态(Saving activity state)

当系统而不是用户自己出于回收内存的考虑，关闭了一个 activity 之后。用户会期望当他再次回到那个 activity 的时候，它仍保持着上次离开时的样子。

为了获取 activity 被杀死前的状态，你应该为 activity 实现 [onSaveInstanceState\(\)](#) 方法。Android 在 activity 有可能被销毁之前（即 [onPause\(\)](#) 调用之前）会调用此方法。它会将一个以名称-值对方式记录了 activity 动态状态的 [Bundle](#) 对象传递给该方法。当 activity 再次启动时，这个 Bundle 会传递给 [onCreate\(\)](#)方法和随着 [onStart\(\)](#)方法调用的 [onRestoreInstanceState\(\)](#)，所以它们两个都可以恢复捕获的状态。

与 [onPause\(\)](#)或先前讨论的其它方法不同，[onSaveInstanceState\(\)](#) 和 [onRestoreInstanceState\(\)](#) 并不是生命周期方法。它们并不是总会被调用。比如说，Android 会在 activity 易于被系统销毁之前调用 [onSaveInstanceState\(\)](#)，但用户动作（比如按下了 BACK 键）造成的销毁则不调用。在这种情况下，用户没打算再次回到这个 activity，所以没有保存状态的必要。

因为 [onSaveInstanceState\(\)](#)不是总被调用，所以你应该只用它来为 activity 保存一些临时的状态，而不能用来保存持久性数据。而是应该用 [onPause\(\)](#)来达到这个目的。

服务生命周期(Coordinating activities)

服务以两种方式使用：

- 它可以启动并运行，直至有人停止了它或它自己停止。在这种方式下，它以调用 [Context.startService\(\)](#)启动，而以调用 [Context.stopService\(\)](#)结束。它可以调用 [Service.stopSelf\(\)](#) 或 [Service.stopSelfResult\(\)](#)来自己停止。不论调用了多少次 [startService\(\)](#)方法，你只需要调用一次 [stopService\(\)](#)来停止服务。
- 它可以通过自己定义并暴露出来的接口进行程序操作。客户端建立一个到服务对象的连接，并通过那个连接来调用服务。连接以调用 [Context.bindService\(\)](#)方法建立，以调用 [Context.unbindService\(\)](#)关闭。多个客户端可以绑定至同一个服务。如果服务此时还没有加载，[bindService\(\)](#)会先加载它。

这两种模式并不是完全分离的。你可以绑定至一个用 [startService\(\)](#)启动的服务。比如说，一个后台音乐播放服务可以调用 [startService\(\)](#)并传递给它一个包含欲播放的音乐列表的 Intent 对象来启动。不久，当用户想要对播放器进行控制或者查看当前播放曲目的详情时，会启用一个 activity，调用 [bindService\(\)](#)连接到服务来完成操作。在这种情况下，直到绑定连接关闭 [stopService\(\)](#) 才会真正停止一个服务。

与 activity 一样，服务也有一系列你可以实现以用于监控其状态变化的生命周期方法。但相对于 activity 要少一些，只有三个，而且，它们是 public 属性，并非 protected：

```
void onCreate()  
void onStart(Intent intent)  
void onDestroy()
```

倚仗实现这些方法，你监控服务的两个嵌套的生命周期循环：

- 服务的完整生命周期始于调用 [onCreate\(\)](#)而终于 [onDestroy\(\)](#)方法返回。如同 activity 一样，服务在 [onCreate\(\)](#)里面进行它自己的初始化，而在 [onDestroy\(\)](#)里面释放所有资源。比如说，一个音乐回放服务可以在 [onCreate\(\)](#)中创建播放音乐的线程，而在 [onDestroy\(\)](#)中停止这个线程。

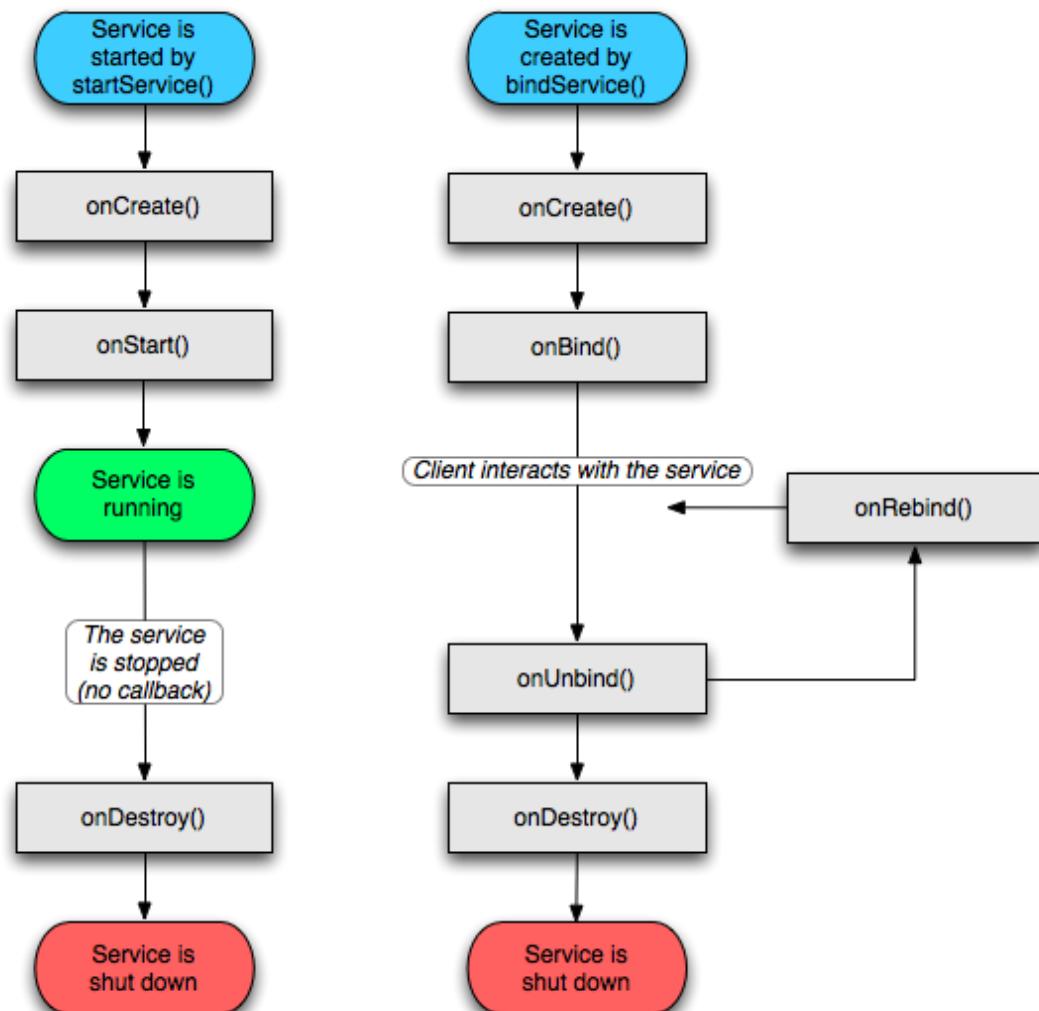
- 服务的活跃生命周期始于调用 [onStart\(\)](#)。这个方法用于处理传递给 `startService()` 的 Intent 对象。音乐服务会打开 Intent 来探明将要播放哪首音乐，并开始播放。
服务停止时没有相应的回调方法——不存在 `onStop()` 方法。
`onCreate()` 和 `onDestroy()` 方法在所有服务中都会被调用，无论它们是由 [Context.startService\(\)](#) 还是由 [Context.bindService\(\)](#) 所启动的。而 `onStart()` 仅会被 `startService()` 所启用的服务调用。

如果一个服务允许别的进程绑定，则它还会有以下额外的回调方法以供实现：

```
IBinder onBind(Intent intent)
boolean onUnbind(Intent intent)
void onRebind(Intent intent)
```

传递给 `bindService` 的 Intent 的对象也会传递给 [onBind\(\)](#) 回调方法，而传递给 `unbindService()` 的 Intent 对象同样传递给 [onUnbind\(\)](#)。如果服务允许绑定，`onBind()` 将返回一个供客户端与服务进行交互的通讯渠道。如果有新的客户端连接至服务，则 `onUnbind()` 方法可以要求调用 [onRebind\(\)](#)。

下图描绘了服务的回调方法。尽管图中对由 `startService` 和 `startService` 方法启动的服务做了区分，但要记住，不论一个服务是怎么启动的，它都可能允许客户端的连接，所以任何服务都可以接受 `onBind()` 和 `onUnbind()` 调用。



广播接收器生命周期(Broadcast receiver lifecycle)

广播接收器只有一个回调方法：

```
void onReceive(Context curContext, Intent broadcastMsg)
```

当广播消息抵达接收器时，Android 调用它的 [onReceive\(\)](#) 方法并将包含消息的 Intent 对象传递给它。广播接收器仅在它执行这个方法时处于活跃状态。当 [onReceive\(\)](#) 返回后，它即为失活状态。

拥有一个活跃状态的广播接收器的进程被保护起来而不会被杀死。但仅拥有失活状态组件的进程则会在其它进程需要它所占有的内存的时候随时被杀掉。

这种方式引出了一个问题：如果响应一个广播信息需要很长的一段时间，我们一般会将其纳入一个衍生的线程中去完成，而不是在主线程内完成它，从而保证用户交互过程的流畅。如果 [onReceive\(\)](#) 衍生了一个线程并且返回，则包涵新线程在内的整个进程都被会判为失活状态（除非进程内的其它应用程序组件仍处于活跃状态），于是它就有可能被杀掉。这个问题的解决方法是令 [onReceive\(\)](#) 启动一个新服务，并用其完成任务，于是系统就会知道进程中仍然在处理着工作。

下一节中，我们会讨论更多进程易误杀的问题。

进程与生命周期(Processes and lifecycles)

Android 系统会尽可能长的延续一个应用程序进程，但在内存过低的时候，仍然会不可避免需要移除旧的进程。为决定保留或移除一个进程，Android 将每个进程都放入一个“重要性层次”中，依据则是它其中运行着的组件及其状态。重要性最低的进程首先被消灭，然后是较低的，依此类推。重要性共分五层，依据重要性列表如下：

1. **前台进程**是用户操作所必须的。当满足如下任一条件时，进程被认为是处于前台的：
 - 它运行着正在与用户交互的 activity (Activity 对象的 [onResume\(\)](#) 方法已被调用)。
 - 一个正在与用户交互的 activity 使用着它提供的一个服务。
 - 它包含着一个正在执行生命周期回调方法([onCreate\(\)](#)、[onStart\(\)](#)或[onDestroy\(\)](#)) 的 [Service](#) 对象。
 - 它包含着一个正在执行 [onReceive\(\)](#) 方法的 [BroadcastReceiver](#) 对象。任一时间下，仅有少数进程会处于前台，仅当内存实在无法供给它们维持同时运行时才会被杀死。一般来说，在这种情况下，设备已然处于使用虚拟内存的状态，必须要杀死一些前台进程以用户界面保持响应。
2. **可视进程**没有前台组件，但仍可被用户在屏幕上所见。当满足如下任一条件时，进程被认为是可视的：
 - 它包含着一个不在前台，但仍然为用户可见的 activity (它的 [onPause\(\)](#) 方法被调用)。这种情况可能出现在以下情况：比如说，前台 activity 是一个对话框，而之前的 activity 位于其下并可以看到。
 - 它包含了一个绑定至一个可视的 activity 的服务。可视进程依然被视为是很重要的，非到不杀死它们便无法维持前台进程运行时，才会被杀死。
3. **服务进程**是由 [startService\(\)](#) 方法启动的服务，它不会变成上述两类。尽管服务进程不会直接为用户所见，但它们一般都在做着用户所关心的事情（比如在后台播放 mp3 或者从网上下载东西）。所以系统会尽量维持它们的运行，除非系统内存不足以维持前台进程和可视进程的运行需要。

4. **背景进程**包含目前不为用户所见的 activity (Activity 对象的 [onStop\(\)](#) 方法已被调用)。这些进程与用户体验没有直接的联系，可以在任意时间被杀死以回收内存供前台进程、可视进程以及服务进程使用。一般来说，会有很多背景进程运行，所以它们一般存放于一个 LRU (最后使用) 列表中以确保最后被用户使用的 activity 最后被杀死。如果一个 activity 正确的实现了生命周期方法，并捕获了正确的状态，则杀死它的进程对用户体验不会有任何不良影响。
5. **空进程**不包含任何活动应用程序组件。这种进程存在的唯一原因是做为缓存以改善组件再次于其中运行时的启动时间。系统经常会杀死这种进程以保持进程缓存和系统内核缓存之间的平衡。

Android 会依据进程中当前活跃组件的重要程度来尽可能高的估量一个进程的级别。比如说，如果一个进程中同时有一个服务和一个可视的 activity，则进程会被判定为可视进程，而不是服务进程。

此外，一个进程的级别可能会由于其它进程依赖于它而升高。一个为其它进程提供服务的进程级别永远高于使用它服务的进程。比如说，如果 A 进程中的内容提供者 为进程 B 中的客户端提供服务，或进程 A 中的服务为进程 B 中的组件所绑定，则 A 进程最低也会被视为与进程 B 拥有同样的重要性。

为运行着一个服务的进程重要级别总高于一个背景 activity。所以一个 activity 以启动一个服务的方式启动一个长时间运行过程比简单的衍生一个线程来进行处理要好。尤其是当处理过程比 activity 本身存在时间要长的情况下。我们以背景音乐播放和上传一个相机拍摄的图片至网站上为例。使用服务则不论 activity 发生何事，都至少可以保证操作拥有“服务进程”的权限。如上一节[广播接收器生命周期](#) 所提到的，这也正是广播接收器使用服务，而不是使用线程来处理耗时任务的原因。

monkeyrunner

署名：译言 biAji

链接：<http://article.yeeyan.org/view/37503/164523>

版本：Android 2.3 r1

整理：农民伯伯

声明

本文档转载并整理自译言：[Monkeyrunner 使用说明](#)

原文

http://developer.android.com/guide/developing/tools/monkeyrunner_concepts.html

monkeyrunner 工具提供了一个 API，使用此 API 写出的程序可以在 Android 代码之外控制 Android 设备和模拟器。通过 monkeyrunner，您可以写出一个 Python 程序去安装一个 Android 应用程序或测试包，运行它，向它发送模拟击键，截取它的用户界面图片，并将截图存储于工作站上。monkeyrunner 工具的主要设计目的是用于测试功能/框架水平上的应用程序和设备，或用于运行单元测试套件，但您当然也可以将其用于其它目的。

monkeyrunner 工具与([UI/Application Exerciser Monkey](#))[用户界面/应用程序测试工具](#)，也称为 monkey 工具并无关联。monkey 工具直接运行在设备或模拟器的 `adb` shell 中，生成用户或系统的伪随机事件流。而 monkeyrunner 工具则是在工作站上通过 API 定义的特定命令和事件控制设备或模拟器。

monkeyrunner 工具为 Android 测试提供了以下特性：

- 多设备控制：monkeyrunner API 可以跨多个设备或模拟器实施测试套件。您可以在同一时间接上所有的设备或一次启动全部模拟器（或统统一起），依据程序依次连接到每一个，然后运行一个或多个测试。您也可以用程序启动一个配置好的模拟器，运行一个或多个测试，然后关闭模拟器。
- 功能测试：monkeyrunner 可以为一个应用自动贯彻一次功能测试。您提供按键或触摸事件的输入数值，然后观察输出结果的截屏。
- 回归测试：monkeyrunner 可以运行某个应用，并将其结果截屏与既定已知正确的结果截屏相比较，以此测试应用的稳定性。
- 可扩展的自动化：由于 monkeyrunner 是一个 API 工具包，您可以基于 Python 模块和程序开发一整套系统，以此来控制 Android 设备。除了使用 monkeyrunner API 之外，您还可以使用标准的 Python `os` 和 `subprocess` 模块来调用如 [Android Debug Bridge](#) 这样的 Android 工具。

您还可以向 monkeyrunner API 中添加您自己的类。我们将在[使用插件扩展 monkeyrunner](#)一节中对此进行详细讨论。

monkeyrunner 工具使用 Jython（使用 Java 编程语言的一种 Python 实现）。Jython 允许 monkeyrunnerAPI 与 Android 框架轻松的进行交互。使用 Jython，您可以使用 Python 语法来获取 API 中的常量、类以及方法。

一个简单的 monkeyrunner 程序实例(A Simple monkeyrunner Program)

以下为一个简单的 monkeyrunner 程序，它将会连接到一个设备，创建一个 `MonkeyDevice` 对象。使用 `MonkeyDevice` 对象，程序将安装一个 Android 应用包，运行其中一个活动，并向其发送按键事件。程序接下来会将结果截图，创建一个 `MonkeyImage` 对象，并使用这个

对象截图将保存至.png 文件。

```
# Imports the monkeyrunner modules used by this program
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice

# Connects to the current device, returning a MonkeyDevice object
device = MonkeyRunner.waitForConnection()

# Installs the Android package. Notice that this method returns a boolean, so you can test
# to see if the installation worked.
device.installPackage('myproject/bin/MyApplication.apk')

# sets a variable with the package's internal name
package = 'com.example.android.myapplication'

# sets a variable with the name of an Activity in the package
activity = 'com.example.android.myapplication.MainActivity'

# sets the name of the component to start
runComponent = package + '/' + activity

# Runs the component
device.startActivity(component=runComponent)

# Presses the Menu button
device.press('KEYCODE_MENU', 'DOWN_AND_UP')

# Takes a screenshot
result = device.takeSnapshot()

# Writes the screenshot to a file
result.writeToFile('myproject/shot1.png', 'png')
```

The monkeyrunner API

monkeyrunnerAPI 于 com.android.monkeyrunner 包中包含三个模块：

- [MonkeyRunner](#): 一个为 monkeyrunner 程序提供工具方法的类。这个类提供了用于连接 monkeyrunner 至设备或模拟器的方法。它还提供了用于创建一个 monkeyrunner 程序的用户界面以及显示内置帮助的方法。
- [MonkeyDevice](#): 表示一个设备或模拟器。这个类提供了安装和卸载程序包、启动一个活动以及发送键盘或触摸事件到应用程序的方法。您也可以用这个类来运行测试包。
- [MonkeyImage](#): 表示一个截图对象。这个类提供了截图、将位图转换成各种格式、比较两个 MonkeyImage 对象以及写图像到文件的方法。

在 python 程序中，您将以 Python 模块的形式使用这些类。monkeyrunner 工具不会自动导入这些模块。您必须使用类似如下的 from 语句：

```
from com.android.monkeyrunner import <module>
```

其中，为您想要导入的类名。您可以在一个 from 语句中导入超过一个模块，其间以逗号分隔。

Running monkeyrunner

您可以直接使用一个代码文件运行 monkeyrunner，抑或在交互式对话中输入 monkeyrunner 语句。不论使用哪种方式，您都需要调用 SDK 目录的 tools 子目录下的 monkeyrunner 命令。如果您提供一个文件名作为运行参数，则 monkeyrunner 将视文件内容

为 Python 程序，并加以运行；否则，它将提供一个交互对话环境。

monkeyrunner 命令的语法为：

```
monkeyrunner -plugin <plugin_jar> <program_filename> <program_options>
```

表 1 阐释了命令的标志和参数。

参数	说明
-plugin <plugin_jar>	(可选) 指定一个内含 monkeyrunner 插件的. jar 文件。欲了解更多关于 monkeyrunner 插件的内容，请参照 (Extending monkeyrunner with plugins) 使用插件扩展 monkeyrunner 。要指定多个文件，包括多次论证。如欲指定超过一个文件，可以多次使用此参数。
<program_filename>	如果您指定此参数，monkeyrunner 将视文件内容为 Python 程序并予以执行。如果此参数未予指定，则开启一个交互式会话。
<program_options>	(可选) <程序文件名> 所指定的程序所需的参数

monkeyrunner 内建帮助(monkeyrunner Built-in Help)

您可以用以下命令来生成 monkeyrunner 的 API 参考：

```
monkeyrunner <format> help.py <outfile>
```

参数说明：

- 可以为 text 或 html，分别代表纯文本和 HTML 输出。
- 指定了输出文件的全路经名称。

使用插件扩展 monkeyrunner(Extending monkeyrunner with Plugins)

您可以用 Java 语言创建新的类，并打包成一个或多个. jar 文件，以此来扩展 monkeyrunnerAPI。您可以使用您自己写的类或者继承现有的类来扩展 monkeyrunnerAPI。您还可以使用此功能来初始化 monkeyrunner 环境。

为了使 monkeyrunner 加载一个插件，您应当如使用如[表 1](#) 中所述的 -plugin 参数来调用 monkeyrunner 命令。

在您编写的插件中，您可以导入或继承位于 com.android.monkeyrunner 包中的几个主要的 monkeyrunner 类：MonkeyDevice、MonkeyImage 和 MonkeyRunner（参见 [monkeyrunnerAPI](#)）。

请注意，插件无法让你访问 Android 的 SDK。您不能导入 com.android.app 等包。这是因为 monkeyrunner 是在框架 API 层次之下与设备或模拟器进行交互的。

插件启动类(The plugin startup class)

用于插件的.jar 文件可以指定一个类，使其在脚本执行之前就实例化。如欲指定这个类，您需要在.jar 文件的 manifest 中添加键 MonkeyRunnerStartupRunner。其值为启动时运行的类的名称。以下代码段显示了如何在一个 ant 构建脚本达到这样的目的：

```
<jar jarfile="myplugin" basedir="${build.dir}">
<manifest>
<attribute name="MonkeyRunnerStartupRunner" value="com.myapp.myplugin"/>
</manifest>
</jar>
```

如欲访问 monkeyrunner 的运行时环境，启动类可以实现 com.google.common.base.Predicate<PythonInterpreter>。例如，用这个类在默认的命名空间中设置一些变量：

```
package com.android.example;

import com.google.common.base.Predicate;
import org.python.util.PythonInterpreter;

public class Main implements Predicate<PythonInterpreter> {
    @Override
    public boolean apply(PythonInterpreter anInterpreter) {
        /*
         * Examples of creating and initializing variables in the monkeyrunner environment's
         * namespace. During execution, the monkeyrunner program can refer to the variables "newtest"
         * and "use_emulator"
         */
        anInterpreter.set("newtest", "enabled");
        anInterpreter.set("use_emulator", 1);

        return true;
    }
}
```

Other Tools

译者署名: 移动云 文斌

译者链接: <http://blog.csdn.net/caowenbin>

版本: Android 2.3 r1

原文

<http://developer.android.com/guide/developing/tools/othertools.html#android>

Other Tools

本文介绍了在开发 Android 应用程序时可以使用的其他工具。

所有的工具都位于 Android SDK 的<code>/tools</code>文件夹下。

android

Android 是一个重要的开发工具, 用于:

- 创建、删除和查看 Android 虚拟设备 (AVDs), 参见 [Android Virtual Devices](#)
- 创建、更新 Android 项目, 参见 [Developing in Other IDEs](#)
- 更新 Android SDK 中的平台、插件和文档, 参见 [Adding SDK Components](#)

如果使用带有 ADT 插件的 Eclipse 作为开发环境, 可以直接在该 IDE 中使用这一工具。在 Eclipse 中创建 Android 项目, 参见 [Developing In Eclipse](#), 在 Eclipse 中更新 SDK, 参见 [Adding SDK Components](#)。

mksdcard

`mksdcard` 工具可以快速创建用于模拟器中的 FAT32 格式的磁盘镜像, 来模拟手机设备中的 SD 卡, 用法是:

```
mksdcard [-l label] <size>[K|M] <file>
```

下表列出了命令先项/参数:

参数	描述
-l	磁盘镜像的卷标
size	要创建的磁盘镜像的大小, 以字节为单位。也可以通过在大小后添加“K”或“M”字母以指定 KB 或 MB 为单位。例如: 1048576K, 1024M.
file	磁盘镜像的路径和文件名

当创建完磁盘镜像以后, 可以在启动模拟器时通过`-sdcard` 参数来加载它, 更多信息可参见 [Android Emulator](#)

```
emulator -sdcard <file>
```

dx

`dx` 工具可以从.class 文件中生成 Android 字节码。该工具将目标文件转换成 Dalvik 可执行格式 (.dex 格式) 文件以供运行, 也可以用可供识别的格式输出类文件的信息和运行单元测试。详细用法可以使用 `dx -help` 来获得帮助。

App Install Location

译者署名: madgoat

译者链接: <http://madgoat.cn>

版本: Android 3.0 r1

原文

<http://developer.android.com/guide/appendix/install-location.html>

自 API Level 8 开始，你可以允许你的应用安装至扩展存储（例如，SD 卡）。这是一个可选功能，你可以在你应用的 manifest 属性 `android:installLocation` 里设定。如果你没设定这个属性，那么你的应用将被安装到内置存储，而且将不允许移动到扩展存储上。

为了允许系统可以在扩展存储上安装你的应用，修改你的 manifest 文件，在 `<manifest>` 元素中包含 `android:installLocation` 属性，设置其值为"preferExternal"或"auto"。例如：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:installLocation="preferExternal"
    ... >
```

如果你定义了 "preferExternal"，意味着你要求你的应用安装至扩展存储，但是系统不能保证应用肯定会安装至扩展存储。如果扩展存储没有空间了，系统将把应用安装到内置存储。用户可以在两个位置之间移动你的应用。

如果你定义了 "auto"，表示你的应用可能会安装在扩展存储，但是对安装位置没有特别的偏好。系统将基于很多因素决定你的应用安装到哪里。用户同样可以将应用在两个位置之间移动。

当你的应用安装在扩展存储上：

- 只要扩展存储已经挂载在设备上，对应用的性能都没有影响。
- .apk 文件保存在扩展存储上，但是所有的用户私有数据，数据库，优化过的 .dex 文件和释放的原生代码都保存在内置存储空间上。
- 存储你应用的唯一容器是被一个随机生成的 KEY 加密存放的，仅仅能被最初安装的设备进行解密操作。因此，安装在 SD 卡上的应用仅仅针对一个设备可以工作。
- 用户可以通过系统设置移动你的应用到内置存储。

警告：当用户启用 USB 大容量存储以共享文件给计算机或者通过系统设置卸载 SD 卡，外置存储从设备卸载并且所有运行在外置存储的应用立刻都被结束。

向后兼容 Backward Compatibility

将你的应用安装至扩展存储的功能是运行 API Level 8（Android 2.2）及以上版本的设备才有效的。使用 API Level 8 之前的版本编译的已存在的应用，将一直安装在内置存储，并且无法移动至扩展存储（即使设备上运行的是 API Level 8 版本的系统）。然而，如果你的应用计划支持低于 8 的 API Level，你可以选择针对 API Level 8 及更高版本支持此特性，并且继续保持与低于 API level 8 的设备兼容。

为了允许安装在扩展存储并且保持与 API Level 8 或更低版本兼容：

- 在 `<manifest>` 元素中，包含值为"auto"或 "preferExternal" 的 `android:installLocation` 属性。
- 继续保持你的 `android:minSdkVersion` 属性不变（小于 8 的值）并且确定你的应用

代码只使用与此 level 保持兼容的 API。

- 为了编译你的应用，更改你的 build target 为 API Level 8。这步操作是必须的，因为旧的 Android 库无法理解 android:installLocation 属性，并且当该属性存在时，也不会编译你的应用。

当你的应用安装到 API Level 低于 8 的设备上时， android:installLocation 属性被忽略，并且应用会被安装至内置存储上。

注意:尽管 XML 标记,例如这个将被之前的平台忽略，但你还是要小心不要使用 API Level 8 中的编程 API，除非你在你的代码中提供向后兼容。关于在应用代码中创建向后兼容的信息，请参考 [Backward Compatibility](#) 这篇文章。

不应当安装在扩展存储的应用

Applications That Should NOT Install on External Storage

当用户启用 USB 大容量存储来给他们的计算机共享文件时(卸载或移除扩展存储)，任何安装在扩展存储上并正在运行的应用都会被结束。实际上此时系统并不知道应用程序的存在，直到大容量存储关闭，或者扩展存储重新挂载到设备上。除了杀死该应用程序使它对用户不可用，它还会使用更严重地方式中断某些类型的应用程序。为了使你的应用始终如你所期望的那样运行，当你使用了下面任何一种特性，那你不应当允许你的应用安装到扩展存储上去，以避免产生当扩展存储被卸载时所导致的后果：

服务 Services

当扩展存储被卸载时，你正在运行的 [Service](#) 将被结束并且不会再重新启动。你可以注册 [ACTION_EXTERNAL_APPLICATIONS_AVAILABLE](#) 广播(broadcast) Intent，当安装在扩展存储上的应用对系统重新有效时，会通知你的应用。在那个时候，你可以重新启动你的 Service。

定时服务 Alarm Services

你注册到 [AlarmManager](#) 的闹钟会被取消。当扩展存储重新挂载时，你必须手工重新注册。

输入法引擎 Input Method Engines

你的输入法([IME](#))将被替换为默认输入法。当扩展存储重新挂载，用户可以打开系统设置以重新启用你的输入法。

壁纸 Live Wallpapers

你正在运行的 [Live Wallpaper](#) 会被替换为默认的。当扩展存储被挂载时，用户可以重新选择 Live Wallpaper。

Live Folders

你的 [Live Folder](#) 将被从 home 屏幕被移除。当扩展存储被挂载上时，用户可以重新添加 Live Folder 到 Home 界面。

应用程序部件 App Widgets

你的 [App Widget](#) 将被从 Home 界面移除，当扩展存储被挂载时，在系统重置 Home 应用之前，用户将无法使用你的 App Widget (通常直到系统重启)。

Account Managers

在扩展存储被挂载之前，你使用 [AccountManager](#) 创建的账户都是不可见的。

Sync Adapters

在扩展存储被挂载之前，你的 [AbstractThreadingSyncAdapter](#) 和所有相关的同步功

能将无法工作。

Device Administrators

你的 [DeviceAdminReceiver](#) 和它的管理能力会被禁止，这会导致设备功能产生无法预料的结果，这种现象会持续到扩展存储重新挂载为止。

Broadcast Receivers listening for "boot completed"

T 系统在扩展存储挂载到设备前发送广播 [ACTION_BOOT_COMPLETED](#)。所以如果你的应用安装到扩展存储上，它拥有也接收不到这个广播。

如果你的应用使用的上面列表中的任何一种特性，那你就应该允许你的应用安装到扩展存储上去。默认情况下，系统将不允许你的应用安装至扩展存储，所以你不需要担心你已存在的应用。然而，如果你不确定你的应用是否永远不会安装到扩展存储上去，那么你可以通过定义 `android:installLocation` 值为 "`internalOnly`" 来确保其安装至内置存储。尽管这不会改变默认的行为，但它明确的指出，你的应用只会被安装在内置存储上并且作为提醒你和其他开发人员已经做出决定。

应当安装在扩展存储的应用

Applications That Should Install on External Storage

简单来说，任何没有使用上一章节功能列表中的应用安装在扩展存储上都是安全的。大型的游戏更是常见的应该允许安装至扩展存储的应用类型，因为游戏当处于非激活状态时，通常不需要提供额外的服务。当扩展存储无效后，游戏进程被结束，这并不会带来明显的影响，当存储重新有效后，用户可以重新启动游戏（假设游戏在正常的 [Activity lifecycle](#) 中保存了状态）。

如果你的应用的 APK 文件大小为几兆(M)，那你就需要认真考虑是否启用应用安装至扩展存储了，这样的话用户可以保留他们的内置存储空间。

android

类

[Manifest](#)

[Manifest.permission](#)

[Manifest.permission_group](#)

Manifest

译者署名: cofice

译者链接: <http://java-cofi.javaeye.com>

版本: Android 2.2 r1

结构

继承关系

public final class Manifest extends Object

java.lang.Object

 android.Manifest

内部类

class Manifest.permission

权限

class Manifest.permission_group

权限组

构造函数

public Manifest ()

构造函数

补充

文章精选

[android Manifest.xml 选项](#)

Manifest.permission

译者署名: cofice

译者链接: <http://java-cofi.javaeye.com>

版本: Android 2.2 r1

结构

继承关系

public static final class Manifest.permission extends Object

java.lang.Object

 android.Manifest.permission

常量

ACCESS_CHECKIN_PROPERTIES	允许在登入数据库的时候读写其中的属性表，并上传改变的值
ACCESS_COARSE_LOCATION	允许应用访问范围(如 WIFI)性的定位
ACCESS_FINE_LOCATION	允许应用访问精确(如 GPS)性的定位
ACCESS_LOCATION_EXTRA_COMMANDS	允许应访问额外的提供定位的指令
ACCESS_MOCK_LOCATION	允许应用创建用于测试的模拟定位提供者
ACCESS_NETWORK_STATE	允许应用访问网络上的信息
ACCESS_SURFACE_FLINGER	允许应用使用低版本视图的特征
ACCESS_WIFI_STATE	允许应用访问关羽 Wi-Fi 网络的信息
ACCOUNT_MANAGER	允许应用进入帐户认证
AUTHENTICATE_ACCOUNTS	允许应用为 ACCOUNT_MANAGER 扮演一个帐户认证系统
BATTERY_STATS	允许应用去统计电源信息
BIND_APPWIDGET	允许应用告诉 AppWidget 哪个应用能够访问该 AppWidget 的数据
BIND_DEVICE_ADMIN	必须通过关机接收者的请求来确保只有系统能够与之交互
BIND_INPUT_METHOD	必须通过 InputMethodService 的请求来确保只有系统能够与之绑定
BIND_WALLPAPER	必须通过 WallpaperService 的请求来确保只有系统能够与之绑定
BLUETOOTH	允许应用去连接蓝牙设备
BLUETOOTH_ADMIN	允许应用找到与之连接的蓝牙设备
BRICK	被请求废止设备(非常危险)
BROADCAST_PACKAGE_REMOVED	允许应用发出一个程序包被移除的广播消息
BROADCAST_SMS	允许应用发出一个收到短信的消息
BROADCAST_STICKY	允许应用发出一个与 intent 相连的消息
BROADCAST_WAP_PUSH	允许应用发出一个收到 WAP PUSH 的广播消息

CALL_PHONE	允许应用启动一个用户确认电话被拨打而不过拨打电话的用户界面的的拨打程序
CALL_PRIVILEGED	允许应用启动一个用户确认电话被拨打而不过拨打电话的用户界面的任意号码的拨打，包括紧急号码.
CAMERA	能够启动照相机设备的请求
CHANGE_COMPONENT_ENABLED_STATE	允许应用去改变一个应用是否是激活状态
CHANGE_CONFIGURATION	允许应用修改当前的配置，如本地设置
CHANGE_NETWORK_STATE	允许应用改变网络的连接状态
CHANGE_WIFI_MULTICAST_STATE	允许应用进入 Wi-Fi 的组播方式
CHANGE_WIFI_STATE	允许应用改变 Wi-Fi 的连接状态
CLEAR_APP_CACHE	允许应用清除所有安装在设备上的应用的缓存
CLEAR_APP_USER_DATA	允许应用清除使用者的信息资料
CONTROL_LOCATION_UPDATES	允许从广播设备来更新或不更新本地的消息
DELETE_CACHE_FILES	允许应用删除掉缓存文件
DELETE_PACKAGES	允许应用删除掉程序包
DEVICE_POWER	允许低权限的访问电源管理项
DIAGNOSTIC	允许应用诊断程序资源
DISABLE_KEYGUARD	允许应用禁用键盘锁
DUMP	允许应用从系统服务中恢复转储的信息
EXPAND_STATUS_BAR	允许应用扩大或缩小状态栏
FACTORY_TEST	如制造商测试的应用一样用终极权限用户运行
FLASHLIGHT	允许访问手电筒
FORCE_BACK	允许应用强制的返回操作而不论是不是最终的 activity
GET_ACCOUNTS	允许应用访问账目服务中的统计清单
GET_PACKAGE_SIZE	允许应用查找出任何程序包使用的空间
GET_TASKS	允许应用找到关于当前或最近运行的任务和在哪些 activities 里运行
GLOBAL_SEARCH	这个权限可以被内容提供者用来允许使用全程搜索他们的数据
HARDWARE_TEST	允许访问硬件及周边设备.
INJECT_EVENTS	允许应用注入用户事件（键盘、触摸）到事件中然后提供给任意的窗口
INSTALL_LOCATION_PROVIDER	允许应用安装一个位置提供商到位置管理器中
INSTALL_PACKAGES	允许应用安装程序包.
INTERNAL_SYSTEM_WINDOW	允许应用打开被部分系统用户接口使用的窗口
INTERNET	允许应用打开网络套接口
KILL_BACKGROUND_PROCESSES	允许应用去呼叫 killBackgroundProcesses(String). 方法
MANAGE_ACCOUNTS	允许应用去管理帐户管理者中的重要清单

MANAGE_APP_TOKENS	允许应用去管理(创建、销毁、顺序)在窗口管理者中的应用
MASTER_CLEAR	
MODIFY_AUDIO_SETTINGS	允许应用修改全局音频设定
MODIFY_PHONE_STATE	允许改变拨打电话的状态-电源等
MOUNT_FORMAT_FILESYSTEMS	允许格式化可移除的存储仓库的文件系统
MOUNT_UNMOUNT_FILESYSTEMS	允许装备或解除可移除的存储仓库的文件系统
PERSISTENT_ACTIVITY	允许应用使它的 activities 更持久稳固
PROCESS_OUTGOING_CALLS	允许应用监督、限定或终止呼出的电话
READ_CALENDAR	允许应用读取用户的日历数据
READ_CONTACTS	允许应用读取用户的联系人数据
READ_FRAME_BUFFER	允许应用抓取屏幕和更多可获得的缓冲数据
READ_HISTORY_BOOKMARKS	允许应用去读取(非写)用户浏览历史和书签
READ_INPUT_STATE	允许应用去的当前键盘和控制的状态
READ_LOGS	允许应用读取低级别的系统日志文件
READ_OWNER_DATA	允许应用读取所有者的数据
READ_PHONE_STATE	允许读取电话的状态
READ_SMS	允许应用读取短信息.
READ_SYNC_SETTINGS	允许应用读取同步的设置
READ_SYNC_STATS	允许应用读取同步的统计数据
REBOOT	重新启动设备的请求
RECEIVE_BOOT_COMPLETED	允许应用接收在系统完成启动后发出的 ACTION_BOOT_COMPLETED 广播信息
RECEIVE_MMS	允许应用去监听多媒体信息并记录和对起进行处理
RECEIVE_SMS	允许应用去监听短消息并记录和对起进行处理
RECEIVE_WAP_PUSH	允许应用监听 WAP push 信息
RECORD_AUDIO	允许应用录音频信息
REORDER_TASKS	允许应用改变任务的关系位置
RESTART_PACKAGES	已废弃使用
SEND_SMS	允许应用发送短消息.
SET_ACTIVITY_WATCHER	允许应用查看和控制 activities 是怎样在系统中运行的
SET_ALWAYS_FINISH	允许应用去控制当 activities 被覆盖后是否是立即接触结束
SET_ANIMATION_SCALE	改变动画的比例因子
SET_DEBUG_APP	设置一个应用为调试模式
SET_ORIENTATION	允许低级别的设置屏幕的方向
SET_PREFERRED_APPLICATIONS	已废弃
SET_PROCESS_LIMIT	允许应用设置可以运行的最大数的应用进程
SET_TIME	允许应用设置系统时间
SET_TIME_ZONE	允许应用设置系统时区时间

SET_WALLPAPER	允许应用设置壁纸
SET_WALLPAPER_HINTS	允许应用设置锁定的壁纸
SIGNAL_PERSISTENT_PROCESSES	允许应用发出一个给所有稳定进程信号的请求
STATUS_BAR	允许应用打开、关闭或使状态栏或图标失去作用
SUBSCRIBED_FEEDS_READ	允许应用访问内容提供者的签署认证
SUBSCRIBED_FEEDS_WRITE	
SYSTEM_ALERT_WINDOW	允许应用使用 TYPE_SYSTEM_ALERT 来打开窗口，并将窗口显示于其他应用的顶端
UPDATE_DEVICE_STATS	允许应用更新设备资料信息
USE_CREDENTIALS	允许应用从管理器得到授权请求
VIBRATE	允许应用访问震动器
WAKE_LOCK	允许使用电源锁定管理以使进程休眠或屏幕变暗
WRITE_APN_SETTINGS	允许应用去写入接入点设置
WRITE_CALENDAR	允许应用写（非读）用户的日历数据
WRITE_CONTACTS	允许应用写（非读）用户的联系人数据
WRITE_EXTERNAL_STORAGE	允许应用写（非读）用户的外部存储器
WRITE_GSERVICES	允许应用修改 Google 服务地图
WRITE_HISTORY_BOOKMARKS	允许应用写（非读）用户的浏览器历史和书签
WRITE_OWNER_DATA	允许应用写（非读）用户的数据
WRITE_SECURE_SETTINGS	允许应用写或读当前系统设置
WRITE_SETTINGS	允许应用写或读系统设置
WRITE_SMS	允许应用写短消息信息
WRITE_SYNC_SETTINGS	允许应用写同步设置

Manifest.permission_group

译者署名: cofice

译者链接: <http://java-cofi.javaeye.com>

版本: Android 2.2 r1

结构

继承关系

public static final class Manifest.permission_group extends Object

java.lang.Object

 android.Manifest.permission_group

常量

ACCOUNTS	直接通过统计管理器访问管理的统计
COST MONEY	可以用来让用户花钱但不需要通过与他们直接牵涉的权限
DEVELOPMENT TOOLS	与开发联盟特征相连的权限组
HARDWARE CONTROLS	被用来提供直接访问硬件设备的权限
LOCATION	用来允许访问用户的当前位置的权限
MESSAGES	用来允许应用发送用户收到的被拦截的信息
NETWORK	用来提供访问网络服务的权限
PERSONAL INFO	用于提供访问用户私人数据如联系人、日历、电子邮件等的权限
PHONE CALLS	用于跟访问和修改拨号状态如截取去话信息、读取和修改电话状态等的权限
STORAGE	与 SD 卡访问相关联的权限组
SYSTEM TOOLS	与系统 API 有关联的权限组

`android.accessibilityservice`

类

[AccessibilityService](#)

`AccessibilityServiceInfo`

AccessibilityService

译者署名: cofice

译者链接: <http://java-cofi.javaeye.com>

版本: Android 2.2 r1

结构

public abstract class AccessibilityService extends Service

```
java.lang.Object
    android.content.Context
        android.content.ContextWrapper
            android.app.Service
                android.accessibilityservice.AccessibilityService
```

类概述

当 AccessibilityEvent 事件被启动后 AccessibilityService 会接回调函数运行于后台, 这些事件指的是在用户接口间的状态转换, 比如, 焦点变化, 按钮被点击等。

一些辅助服务继承于此类并且实现它的抽象方法, 像这样的一个服务和其他服务一样在 `AndroidManifest.xml` 中被声明但它必须被指定操纵 “`android.accessibilityservice.AccessibilityService`” 的意图, 下面的是一段例子:

```
<service android:name=".MyAccessibilityService">
<intent-filter>
<action android:name="android.accessibilityservice.AccessibilityService" />
</intent-filter>
</service>
```

辅助服务的声明周期只能被系统管理, 启动或者停止这个服务必须由明确的用户通过启用或停用设备的设定, 在系统通过呼叫 `onServiceConnected()` 方法与服务绑定后, 这个方法才能被想要执行装载的客户端所重载使用, 一个辅助服务通过呼叫 `setServiceInfo(AccessibilityServiceInfo)` 方法来设定 `AccessibilityServiceInfo` 而配置。你可以在任何时候改变这个服务的配置但最好是在重载方法 `onServiceConnected()` 中来使用。

一个辅助服务可以在特定的包中注册事件以提供特殊的反馈类型并且当最后一个关联的事件被解除的时候发出明确的超时提醒。

通告策略

对于每个回馈类型只有一个辅助服务被通知, 服务登记处按顺序被通知, 因此, 如果有两个服务为同一个包中的同一回馈类型注册那么第一个会被通知, 然而有可能的是, 可以为一个给定的回馈类型去把一个服务注册为默认的, 这样的话如果没有其他的服 务来取代这个事件这个服务就会被呼出使用, 换句话说, 默认的服务不会与其他的服 务竞争并且不管注册的顺序而被通知。

常量

`String SERVICE_INTERFACE`

声明 Intent 必须由这个服务来处理

公共方法

`abstract void onAccessibilityEvent(AccessibilityEvent event)`

参数

`event` 一个事件

`public final IBinder onBind (Intent intent)`

实现返回一个内部的辅助接口的实现，子类不能被重写。

参数

`intent` 与服务相绑定的意图，注意其他任何包含在 Intent 的外部意图将不能在此使用。

返回值

返回一个客户端可以在服务上访问的 IBinder。

`public abstract void onInterrupt ()`

打断辅助回馈内容时呼叫。

受保护方法

`protected void onServiceConnected ()`

这个方法是 AccessibilityService 声明周期的一部分，在系统成功与服务绑定后才被呼叫，如果用来设定 AccessibilityServiceInfo.这个方法更为方便。

android.bluetooth

类

[BluetoothAdapter](#)
[BluetoothClass](#)
[BluetoothClass.Device](#)
[BluetoothClass.Device.Major](#)
[BluetoothClass.Service](#)
[BluetoothDevice](#)
[BluetoothServerSocket](#)
[BluetoothSocket](#)

BluetoothAdapter

译者署名: Android Club SYSU

译者链接: <http://www.android-wiki.net>

版本: Android 2.3 r1

结构

继承关系

public final class BluetoothAdapter extends Object

```
java.lang.Object  
    android.bluetooth.BluetoothAdapter
```

类概述

代表本地的蓝牙适配器设备。BluetoothAdapter 类让用户能执行基本的蓝牙任务。例如: 初始化设备的搜索, 查询可匹配的设备集, 使用一个已知的 MAC 地址来初始化一个 BluetoothDevice 类, 创建一个 BluetoothServerSocket 类以监听其它设备对本机的连接请求等。

为了得到这个代表本地蓝牙适配器的 BluetoothAdapter 类, 调用 getDefaultAdapter() 这一静态方法。这是所有蓝牙动作使用的第一步。当拥有本地适配器以后, 用户可以获得一系列的 BluetoothDevice 对象, 这些对象代表所有拥有 getBondedDevice() 方法的已经匹配的设备; 用 startDiscovery() 方法来开始设备的搜寻; 或者创建一个 BluetoothServerSocket 类, 通过 listenUsingRfcommWithServiceRecord(String, UUID) 方法来监听新来的连接请求。

Note: 大部分方法需要 BLUETOOTH 权限, 一些方法同时需要 [BLUETOOTH_ADMIN](#) 权限。

参见

[BluetoothDevice](#)

[BluetoothServerSocket](#)

常量

String ACTION_DISCOVERY_FINISHED

广播事件: 本地蓝牙适配器已经完成设备的搜寻过程。

需要 [BLUETOOTH](#) 权限接收。

常量值: "android.bluetooth.adapter.action.DISCOVERY_FINISHED"

String ACTION_DISCOVERY_STARTED

广播事件: 本地蓝牙适配器已经开始对远程设备的搜寻过程。

它通常牵涉到一个大概需时 12 秒的查询扫描过程, 紧跟着是一个对每个获取到自身蓝牙名称的新设备的页面扫描。

用户会发现一个把 [ACTION_FOUND](#) 常量通知为远程蓝牙设备的注册。

设备查找是一个重量级过程。当查找正在进行的时候, 用户不能尝试对新的远程蓝牙设备进行连接, 同时存在的连接将获得有限制的带宽以及高等待时间。用户可用 [cancelDiscovery\(\)](#) 类来取消正在执行的查找进程。

需要 [BLUETOOTH](#) 权限接收。

常量值: "android.bluetooth.adapter.action.DISCOVERY_STARTED"

`String ACTION_LOCAL_NAME_CHANGED`

广播活动：本地蓝牙适配器已经更改了它的蓝牙名称。

该名称对远程蓝牙设备是可见的。

它总是包含了一个带有名称的 `EXTRA_LOCAL_NAME` 附加域。

需要 [BLUETOOTH](#) 权限接收。

常量值: "android.bluetooth.adapter.action.LOCAL_NAME_CHANGED"

`String ACTION_REQUEST_DISCOVERABLE`

Activity 活动：显示一个请求被搜寻模式的系统活动。如果蓝牙模块当前未打开，该活动也将请求用户打开蓝牙模块。

被搜寻模式和 [SCAN_MODE_CONNECTABLE_DISCOVERABLE](#) 等价。当远程设备执行查找进程的时候，它允许其发现该蓝牙适配器。

从隐私安全考虑，Android 不会将被搜寻模式设置为默认状态。

该意图的发送者可以选择性地运用 [EXTRA_DISCOVERABLE_DURATION](#) 这个附加域去请求发现设备的持续时间。普遍来说，对于每一请求，默认的持续时间为 120 秒，最大值则可达到 300 秒。

Android 运用 [onActivityResult\(int, int, Intent\)](#) 回收方法来传递该活动结果的通知。被搜寻的时间（以秒为单位）将通过 `resultCode` 值来显示，如果用户拒绝被搜寻，或者设备产生了错误，则通过 [RESULT_CANCELED](#) 值来显示。

每当扫描模式变化的时候，应用程序可以通过 [ACTION_SCAN_MODE_CHANGED](#) 值来监听全局的消息通知。比如，当设备停止被搜寻以后，该消息可以被系统通知给应用程序。

需要 [BLUETOOTH](#) 权限

常量值: "android.bluetooth.adapter.action.REQUEST_DISCOVERABLE"

`String ACTION_REQUEST_ENABLE`

Activity 活动：显示一个允许用户打开蓝牙模块的系统活动。

当蓝牙模块完成打开工作，或者当用户决定不打开蓝牙模块时，系统活动将返回该值。

Android 运用 [onActivityResult\(int, int, Intent\)](#) 回收方法来传递该活动结果的通知。如果蓝牙模块被打开，将通过 `resultCode` 值 [RESULT_OK](#) 来显示；如果用户拒绝该请求，或者设备产生了错误，则通过 [RESULT_CANCELED](#) 值来显示。

每当蓝牙模块被打开或者关闭，应用程序可以通过 [ACTION_STATE_CHANGED](#) 值来监听全局的消息通知。

需要 [BLUETOOTH](#) 权限

常量值: "android.bluetooth.adapter.action.REQUEST_ENABLE"

`String ACTION_SCAN_MODE_CHANGED`

广播活动：指明蓝牙扫描模块或者本地适配器已经发生变化

它总是包含 [EXTRA_SCAN_MODE](#) 和 [EXTRA_PREVIOUS_SCAN_MODE](#)。这两个附加域各自包含了新的和旧的扫描模式。

需要 [BLUETOOTH](#) 权限

常量值: "android.bluetooth.adapter.action.SCAN_MODE_CHANGED"

`String ACTION_STATE_CHANGED`

广播活动：本来的蓝牙适配器的状态已经改变。

例如：蓝牙模块已经被打开或者关闭。

它总是包含 [EXTRA_STATE](#) 和 [EXTRA_PREVIOUS_STATE](#)。这两个附加域各自包含了新的和旧的状态。

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.adapter.action.STATE_CHANGED"

int ERROR

标记该类的错误值。确保和该类中的任意其它整数常量不相等。它为需要一个标记错误值的函数提供了便利。例如：

`Intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAdapter.ERROR)`

常量值: -2147483648 (0x80000000)

String EXTRA_DISCOVERABLE_DURATION

试图在 [ACTION_REQUEST_DISCOVERABLE](#) 常量中作为一个可选的整型附加域，来为短时间内的设备发现请求一个特定的持续时间。默认值为 120 秒，超过 300 秒的请求将被限制。这些值是可以变化的。

常量值: "android.bluetooth.adapter.extra.DISCOVERABLE_DURATION"

String EXTRA_LOCAL_NAME

试图在 [ACTION_LOCAL_NAME_CHANGED](#) 常量中作为一个字符串附加域，来请求本地蓝牙的名称。

常量值: "android.bluetooth.adapter.extra.LOCAL_NAME"

String EXTRA_PREVIOUS_SCAN_MODE

试图在 [ACTION_SCAN_MODE_CHANGED](#) 常量中作为一个整型附加域，来请求以前的扫描模式。可能值有: [SCAN_MODE_NONE](#), [SCAN_MODE_CONNECTABLE](#),

[SCAN_MODE_CONNECTABLE_DISCOVERABLE](#)

常量值: "android.bluetooth.adapter.extra.PREVIOUS_SCAN_MODE"

String EXTRA_PREVIOUS_STATE

试图在 [ACTION_STATE_CHANGED](#) 常量中作为一个整型附加域，来请求以前的供电状态。可能值有: [STATE_OFF](#), [STATE_TURNING_ON](#), [STATE_ON](#), [STATE_TURNING_OFF](#)

常量值: "android.bluetooth.adapter.extra.PREVIOUS_STATE"

String EXTRA_SCAN_MODE

试图在 [ACTION_SCAN_MODE_CHANGED](#) 常量中作为一个整型附加域，来请求当前的扫描模式。可能值有: [SCAN_MODE_NONE](#), [SCAN_MODE_CONNECTABLE](#),

[SCAN_MODE_CONNECTABLE_DISCOVERABLE](#)

常量值: "android.bluetooth.adapter.extra.SCAN_MODE"

String EXTRA_STATE

试图在 [ACTION_STATE_CHANGED](#) 常量中作为一个整型附加域，来请求当前的供电状态。可能值有: [STATE_OFF](#), [STATE_TURNING_ON](#), [STATE_ON](#), [STATE_TURNING_OFF](#)

常量值: "android.bluetooth.adapter.extra.STATE"

`int SCAN_MODE_CONNECTABLE`

指明在本地蓝牙适配器中，查询扫描功能失效，但页面扫描功能有效。因此该设备不能被远程蓝牙设备发现，但如果以前曾经发现过该设备，则远程设备可以对其进行连接。

常量值: 21 (0x00000015)

`int SCAN_MODE_CONNECTABLE_DISCOVERABLE`

指明在本地蓝牙适配器中，查询扫描功能和页面扫描功能都有效。因此该设备既可以被远程蓝牙设备发现，也可以被其连接。

常量值: 23 (0x00000017)

`int SCAN_MODE_NONE`

指明在本地蓝牙适配器中，查询扫描功能和页面扫描功能都失效。因此该设备既不可以被远程蓝牙设备发现，也不可以被其连接。

常量值: 20 (0x00000014)

`int STATE_OFF`

指明本地蓝牙适配器模块已经关闭

常量值: 10 (0x0000000a)

`int STATE_ON`

指明本地蓝牙适配器模块已经打开，并且准备被使用。

常量值: 12 (0x0000000c)

`int STATE_TURNING_OFF`

指明本地蓝牙适配器模块正在关闭。本地客户端可以立刻尝试友好地断开任意外部连接。

常量值: 13 (0x0000000d)

`int STATE_TURNING_ON`

指明本地蓝牙适配器模块正在打开。然而本地客户在尝试使用这个适配器之前需要为 [STATE ON](#) 状态而等待。

常量值: 11 (0x0000000b)

公共方法

`public boolean cancelDiscovery ()`

取消当前的设备发现查找进程

需要 [BLUETOOTH_ADMIN](#) 权限。

因为对蓝牙适配器而言，查找是一个重量级的过程，因此这个方法必须在尝试连接到远程设备前使用用 `connect()` 方法进行调用。发现的过程不会由活动来进行管理，但是它会作为一个系统服务来运行，因此即使它不能直接请求这样的一个查询动作，也必需取消该搜索进程。

如果蓝牙状态不是 [STATE ON](#)，这个 API 将返回 `false`。蓝牙打开后，等待

`ACTION_STATE_CHANGED` 更新成 `STATE_ON`。

返回值

成功则返回 `true`, 有错误则返回 `false`。

`public static boolean checkBluetoothAddress (String address)`

验证譬如"00:43:A8:23:10:F0"之类的蓝牙地址。

字母必须为大写才有效。

参数

`address` 字符串形式的蓝牙模块地址

返回值

地址正确则返回 `true`, 否则返回 `false`。

`public boolean disable ()`

关闭本地蓝牙适配器—不能在没有明确关闭蓝牙的用户动作中使用。

这个方法友好地停止所有的蓝牙连接，停止蓝牙系统服务，以及对所有基础蓝牙硬件进行断电。

没有用户的直接同意，蓝牙永远不能被禁止。这个 `disable()`方法只提供了一个应用，该应用包含了一个改变系统设置的用户界面（例如“电源控制”应用）。

这是一个异步调用方法：该方法将马上获得返回值，用户要通过监听

`ACTION_STATE_CHANGED` 值来获取随后的适配器状态改变的通知。如果该调用 返回 `true` 值，则该适配器状态会立刻从 `STATE_ON` 转向 `STATE_TURNING_OFF`，稍后则会转为 `STATE_OFF` 或者 `STATE_ON`。如果该调用返回 `false`，那么系统已经有一个保护蓝牙适配器被关闭的问题—比如该适配器已经被关闭了。

需要 [BLUETOOTH_ADMIN](#) 权限。

返回值

如果蓝牙适配器的停止进程已经开启则返回 `true`，如果产生错误则返回 `false`。

`public boolean enable ()`

打开本地蓝牙适配器—不能在没有明确打开蓝牙的用户动作中使用。

该方法将为基础的蓝牙硬件供电，并且启动所有的蓝牙系统服务。

没有用户的直接同意，蓝牙永远不能被禁止。如果用户为了创建无线连接而打开了蓝牙模块，则其需要 `ACTION_REQUEST_ENABLE` 值，该值将提出一个请求用户允许以打开蓝牙模块的会话。这个 `enable()`值只提供了一个应用，该应用包含了一个改变系统设置的用户界面（例如“电源控制”应用）。

这是一个异步调用方法：该方法将马上获得返回值，用户要通过监听 `ACTION_STATE_CHANGED` 值来获取随后的适配器状态改变的通知。如果该调用 返回 `true` 值，则该适配器状态会立刻从 `STATE_OFF` 转向 `STATE_TURNING_ON`，稍后则会转为 `STATE_OFF` 或者 `STATE_ON`。如果该调用返回 `false`，那么说明系统已经有一个保护蓝牙适配器被打开的问题—比如飞行模式，或者该适配器已经被打开。

需要 [BLUETOOTH_ADMIN](#) 权限。

返回值

如果蓝牙适配器的打开进程已经开启则返回 `true`，如果产生错误则返回 `false`。

`public String getAddress ()`

返回本地蓝牙适配器的硬件地址

例如： "00:11:22:AA:BB:CC"

需要 [BLUETOOTH](#) 权限。

返回值

字符串形式的蓝牙模块地址

```
public Set<BluetoothDevice> getBondedDevices ()
```

返回已经匹配到本地适配器的 BluetoothDevice 类的对象集合

如果蓝牙状态不是 [STATE_ON](#), 这个 API 将返回 false。蓝牙打开后，等待

[ACTION_STATE_CHANGED](#) 更新成 STATE_ON。

需要 [BLUETOOTH](#) 权限。

返回值

未被修改的 BluetoothDevice 类的对象集合，如果有错误则返回 null。

```
public static synchronized BluetoothAdapter getDefaultAdapter ()
```

获取对默认本地蓝牙适配器的操作权限。

目前 Andoird 只支持一个蓝牙适配器，但是 API 可以被扩展为支持多个适配器。该方法总是返回默认的适配器。

返回值

返回默认的本地适配器，如果蓝牙适配器在该硬件平台上不能被支持，则返回 null。

```
public String getName ()
```

获取本地蓝牙适配器的蓝牙名称

这个名称对于外界蓝牙设备而言是可见的。

需要 [BLUETOOTH](#) 权限。

返回值

该蓝牙适配器名称，如果有错误则返回 null

```
public BluetoothDevice getRemoteDevice (String address)
```

为给予的蓝牙硬件地址获取一个 BluetoothDevice 对象。

合法的蓝牙硬件地址必须为大写，格式类似于"00:11:22:33:AA:BB"。

[checkBluetoothAddress\(String\)](#)方法可以用来验证蓝牙地址的正确性。

BluetoothDevice 类对于合法的硬件地址总会产生返回值，即使这个适配器从未见过该设备。

参数

address 合法的蓝牙 MAC 地址

异常

[IllegalArgumentException](#) 如果地址不合法

```
public int getScanMode ()
```

获取本地蓝牙适配器的当前蓝牙扫描模式

蓝牙扫描模式决定本地适配器可连接并且/或者可被远程蓝牙设备所连接。

可能值有: [SCAN_MODE_NONE](#), [SCAN_MODE_CONNECTABLE](#),

`SCAN_MODE_CONNECTABLE_DISCOVERABLE.`

如果蓝牙状态不是 [STATE_ON](#), 这个 API 将返回 `false`。蓝牙打开后，等待 `ACTION_STATE_CHANGED` 更新成 `STATE_ON`。

需要 [BLUETOOTH](#) 权限。

返回值

扫描模式

```
public int getState ()
```

获取本地蓝牙适配器的当前状态

可能值有 `STATE_OFF`, `STATE_TURNING_ON`, `STATE_ON`, `STATE_TURNING_OFF`.

需要 [BLUETOOTH](#) 类

需要 [BLUETOOTH](#) 权限。

返回值

current state of Bluetooth adapter

```
public boolean isDiscovering ()
```

如果当前蓝牙适配器正处于设备发现查找进程中，则返回真值

设备查找是一个重量级过程。当查找正在进行的时候，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限制的带宽以及高等待时间。用户可用 `cancelDiscovery()`类来取消正在执行的查找进程。

应用程序也可以为 `ACTION_DISCOVERY_STARTED` 或者 `ACTION_DISCOVERY_FINISHED` 进行注册，从而当查找开始或者完成的时候，可以获得通知。

如果蓝牙状态不是 [STATE_ON](#), 这个 API 将返回 `false`。蓝牙打开后，等待 `ACTION_STATE_CHANGED` 更新成 `STATE_ON`。

需要 [BLUETOOTH](#) 权限。

返回值

如果正在查找，则返回 `true`

```
public boolean isEnabled ()
```

如果蓝牙正处于打开状态并可用，则返回真值

与 `getBluetoothState() == STATE_ON` 等价

需要 [BLUETOOTH](#) 权限。

返回值

如果本地适配器已经打开，则返回 `true`

```
public BluetoothServerSocket listenUsingRfcommWithServiceRecord (String name, UUID
```

```
uuid)
```

创建一个正在监听的安全的带有服务记录的无线射频通信（RFCOMM）蓝牙端口。

一个对该端口进行连接的远程设备将被认证，对该端口的通讯将被加密。

使用 `accept()` 方法可以获取从监听 [BluetoothServerSocket](#) 处新来的连接

该系统分配一个未被使用的无线射频通信通道来进行监听。

该系统也将注册一个服务探索协议（SDP）记录，该记录带有一个包含了特定的通用唯一识别码（Universally Unique Identifier，UUID），服务器名称和自动分配通道的本地 SDP 服务。远程蓝牙设备可以用相同的 UUID 来查询自己的 SDP 服务器，并搜寻连接到了哪个

通道上。如果该端口已经关闭，或者如果该应用程序异常退出，则这个 SDP 记录会被移除。

使用 [createRfcommSocketToServiceRecord\(UUID\)](#) 从另一使用相同 [UUID](#) 的设备来连接到这个端口

需要 [BLUETOOTH](#) 权限。

参数

name SDP 记录下的服务器名

uuid SDP 记录下的 UUID

返回值

一个正在监听的无线射频通信蓝牙服务端口

异常

[IOException](#) 产生错误，比如蓝牙设备不可用，或者许可无效，或者通道被占用。

```
public boolean setName (String name)
```

设置蓝牙或者本地蓝牙适配器的昵称。

这个名字对于外界蓝牙设备而言是可见的。

合法的蓝牙名称最多拥有 248 位 UTF-8 字符，但是很多外界设备只能显示前 40 个字符，有些可能只限制前 20 个字符。

如果蓝牙状态不是 [STATE_ON](#)，这个 API 将返回 false。蓝牙打开后，等待

[ACTION_STATE_CHANGED](#) 更新成 [STATE_ON](#)。

需要 [BLUETOOTH_ADMIN](#) 权限。

参数

name 一个合法的蓝牙名称

返回值

如果该名称已被设定，则返回 true，否则返回 false

```
public boolean startDiscovery ()
```

开始对远程设备进行查找的进程

它通常牵涉到一个大概需时 12 秒的查询扫描过程，紧跟着是一个对每个获取到自身蓝牙名称的新设备的页面扫描。

这是一个异步调用方法：该方法将马上获得返回值，注册 [ACTION_DISCOVERY_STARTED](#) 和 [ACTION_DISCOVERY_FINISHED](#) 意图准确地确定该探索是处于开始阶段或者完成阶段。注册 [ACTION_FOUND](#) 以活动远程蓝牙设备 已找到的通知。

设备查找是一个重量级过程。当查找正在进行的时候，用户不能尝试对新的远程蓝牙设备进行连接，同时存在的连接将获得有限制的带宽以及高等待时间。用户可用 [cancelDiscovery\(\)](#) 类来取消正在执行的查找进程。发现的过程不会由活动来进行管理，但是它会作为一个系统服务来运行，因此即使它不能直接请求这样的一个查询动作，也必需取消该搜索进程。

设备搜寻只寻找已经被连接的远程设备。许多蓝牙设备默认不会被搜寻到，并且需要进入到一个特殊的模式当中。

如果蓝牙状态不是 [STATE_ON](#)，这个 API 将返回 false。蓝牙打开后，等待 [ACTION_STATE_CHANGED](#) 更新成 [STATE_ON](#)。

需要 [BLUETOOTH_ADMIN](#) 权限。

返回值

成功返回 true，错误返回 false。

补充

文章精选

[Android 蓝牙开发浅谈](#)

[Android 蓝牙 API 之 BluetoothAdapter 类\(1\)](#)

[Android 蓝牙 API 之 BluetoothAdapter 类\(2\)](#)

[Ophone 平台蓝牙编程基础](#)

BluetoothClass

译者署名： Android Club SYSU

译者链接：<http://www.android-wiki.net>

版本：Android 2.3 r1

结构

继承关系

public final class BluetoothClass extends Object implements Parcelable

java.lang.Object
 android.bluetooth.BluetoothClass

类概述

代表一个描述了设备通用特性和功能的蓝牙类。比如，一个蓝牙类会指定皆如电话、计算机或耳机的通用设备类型，可以提供皆如音频或者电话的服务。

每个蓝牙类都是有 0 个或更多的服务类，以及一个设备类组成。设备类将被分解成主要和较小的设备类部分。

[BluetoothClass](#) 用作一个能粗略描述一个设备（比如关闭用户界面上一个图标的设备）的线索，但当蓝牙服务事实上是被一个设备所支撑的时候，BluetoothClass 的 介绍则不那么可信任。精确的服务搜寻通过 SDP 请求来完成。当运用 [createRfcommSocketToServiceRecord\(UUID\)](#) 和 [listenUsingRfcommWithServiceRecord\(String, UUID\)](#) 来创建 RFCOMM 端口的时候，SDP 请求就会自动执行。

使用 [getBluetoothClass\(\)](#)方法来获取为远程设备所提供的类。

内部类

class BluetoothClass.Device

定义所有设备类的常量

class BluetoothClass.Service

定义所有服务类的常量

公共方法

public int describeContents ()

描述包含在可封装编组的表示中所有特殊对象的种类。

返回值

一个指示被 Parcelabel 所排列的特殊对象类型集合的位掩码。

public boolean equals (Object o)

比较带有特定目标的常量。如果他们相等则标示出来。为了保证其相等，o 必须代表相同的对象，该对象作为这个使用类依赖比较的常量。通常约定，该比较既要可移植又需灵活。

当且仅当 o 是一个作为接收器（使用==操作符来做比较）的精确相同的对象是，这个对象的实现才返回 true 值。子类通常实现 equals(Object) 方法，这样它才会重视这两个对象的类型和状态。

通常约定，对于 equals(Object) 和 hashCode() 方法，如果 equals 对于任意两个对象返回真值，那么 hashCode() 必须对这些对象返回相同的值。这意味着对象的子类通常都覆盖或者都不覆盖这两个方法。

参数

- o 需要对比常量的对象

返回值

如果特定的对象和该对象相等则返回 true，否则返回 false。

public int getDeviceClass ()

返回 [BluetoothClass](#) 中的设备类部分（主要的和较小的）

从函数中返回的值可以和在 [BluetoothClass.Device](#) 中的公共常量做比较，从而确定哪个设备类在这个蓝牙类中是被编码的。

返回值

设备类部分

public int getMajorDeviceClass ()

返回 [BluetoothClass](#) 中设备类的主要部分

从函数中返回的值可以和在 [BluetoothClass.Device.Major](#) 中的公共常量做比较，从而确定哪个主要类在这个蓝牙类中是被编码的。

返回值

主要设备类部分

public boolean hasService (int service)

如果该指定服务类被 [BluetoothClass](#) 所支持，则返回 true

在 [BluetoothClass.Service](#) 中，合法的服务类是公共常量，比如 [AUDIO](#) 类。

参数

service 合法服务类

返回值

如果该服务类可被支持，则返回 true

public int hashCode ()

返回这个对象的整型哈希码。按约定，任意两个在 [equals\(Object\)](#) 中返回 true 的对象必须返回相同的哈希码。这意味着对象的子类通常都覆盖或者都不覆盖这两个方法。

注意：除非同等对比信息发生改变，否则哈希码不随时间改变而改变。

如果你想要实现你自己的哈希码方法，参见 [Writing a correct hashCode method](#)。

返回值

该对象的哈希码

public String toString ()

返回这个对象的字符串，该字符串包含精确且可读的介绍。系统鼓励子类去重写该方法，并且提供了能对该对象的类型和数据进行重视的实现方法。默认的实现方法只是简单地把类名、“@”符号和该对象 hashCode() 方法的 16 进制数连接起来（如下列所示的表达式）：

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

如果你想实现你自己的 `toString` 方法，参见 [Writing a useful `toString` method](#)。

返回值

该对象中一个可被打印的字符串。

public void `writeToParcel` (Parcel out, int flags)

将类的数据写入外部提供的 Parcel 中

参数

`out` 对象需要被写入的 Parcel

`flags` 和对象需要如何被写入有关的附加标志。可能是 0，或者可能是
[PARCELABLE_WRITE_RETURN_VALUE](#)。

BluetoothClass.Device

译者署名： Android Club SYSU

译者链接：<http://www.android-wiki.net>

版本：Android 2.3 r1

结构

继承关系

public final class BluetoothClass.Device extends Object

java.lang.Object

 android.bluetooth.BluetoothClass.Device

类概述

定义所有的设备类常量

每个 [BluetoothClass](#) 对一个带有主要和较小部分的设备类进行编码。

[BluetoothClass.Device](#) 中的常量代表主要和较小的设备类部分（完整的设备类）的组合。

[BluetoothClass.Device.Major](#) 的常量只能代表主要设备类。

参见服务类的常量 [BluetoothClass.Service](#)。

内部类

class [BluetoothClass.Device.Major](#)

定义了所有的主要设备类常量

常量

int [AUDIO_VIDEO_CAMCORDER](#)

int [AUDIO_VIDEO_CAR_AUDIO](#)

int [AUDIO_VIDEO_HANDSFREE](#)

int [AUDIO_VIDEO_HEADPHONES](#)

int [AUDIO_VIDEO_HIFI_AUDIO](#)

int [AUDIO_VIDEO_LOUDSPEAKER](#)

int [AUDIO_VIDEO_MICROPHONE](#)

int [AUDIO_VIDEO_PORTABLE_AUDIO](#)

int [AUDIO_VIDEO_SET_TOP_BOX](#)

int [AUDIO_VIDEO_UNCATEGORIZED](#)

```
int AUDIO_VIDEO_VCR  
  
int AUDIO_VIDEO_VIDEO_CAMERA  
  
int AUDIO_VIDEO_VIDEO_CONFERENCING  
  
int AUDIO_VIDEO_VIDEO_DISPLAY_AND_LOUDSPEAKER  
  
int AUDIO_VIDEO_VIDEO_GAMING_TOY  
  
int AUDIO_VIDEO_VIDEO_MONITOR  
  
int AUDIO_VIDEO_WEARABLE_HEADSET  
  
int COMPUTER_DESKTOP  
  
int COMPUTER_HANDHELD_PC_PDA  
  
int COMPUTER_LAPTOP  
  
int COMPUTER_PALM_SIZE_PC_PDA  
  
int COMPUTER_SERVER  
  
int COMPUTER_UNCATEGORIZED  
  
int COMPUTER_WEARABLE  
  
int HEALTH_BLOOD_PRESSURE  
  
int HEALTH_DATA_DISPLAY  
  
int HEALTH_GLUCOSE  
  
int HEALTH_PULSE_OXIMETER  
  
int HEALTH_PULSE_RATE  
  
int HEALTH_THERMOMETER  
  
int HEALTH_UNCATEGORIZED  
  
int HEALTH_WEIGHING
```

```
int PHONE_CELLULAR  
  
int PHONE_CORDLESS  
  
int PHONE_ISDN  
  
int PHONE_MODEM_OR_GATEWAY  
  
int PHONE_SMART  
  
int PHONE_UNCATEGORIZED  
  
int TOY_CONTROLLER  
  
int TOY_DOLL_ACTION FIGURE  
  
int TOY_GAME  
  
int TOY_ROBOT  
  
int TOY_UNCATEGORIZED  
  
int TOY_VEHICLE  
  
int WEARABLE_GLASSES  
  
int WEARABLE_HELMET  
  
int WEARABLE_JACKET  
  
int WEARABLE_PAGER  
  
int WEARABLE_UNCATEGORIZED  
  
int WEARABLE_WRIST_WATCH
```

BluetoothClass.Device.Major

译者署名: Android Club SYSU

译者链接: <http://www.android-wiki.net>

版本: Android 2.3 r1

结构

继承关系

public static class BluetoothClass.Device.Major extends Object

java.lang.Object

 android.bluetooth.BluetoothClass.Device.Major

类概述

定义所有的主要设备类常量。

常量

int AUDIO_VIDEO

int COMPUTER

int HEALTH

int IMAGING

int MISC

int NETWORKING

int PERIPHERAL

int PHONE

int TOY

int UNCATEGORIZED

int WEARABLE

BluetoothClass.Service

译者署名: Android Club SYSU

译者链接: <http://www.android-wiki.net>

版本: Android 2.3 r1

结构

继承关系

public static final class BluetoothClass.Service extends Object

java.lang.Object
 android.bluetooth.BluetoothClass.Service

类概述

定义所有的服务类常量

任意 [BluetoothClass](#) 由 0 或多个服务类编码。

常量

int AUDIO

int CAPTURE

int INFORMATION

int LIMITED_DISCOVERABILITY

int NETWORKING

int OBJECT_TRANSFER

int POSITIONING

int RENDER

int TELEPHONY

BluetoothDevice

译者署名: Android Club SYSU

译者链接: <http://www.android-wiki.net>

版本: Android 2.3 r1

结构

继承关系

public static class BluetoothDevice extends ViewGroup.LayoutParams

```
java.lang.Object  
    android.view.ViewGroup.LayoutParams  
        android.widget.GalleryLayoutParams
```

类概述

代表一个远程蓝牙设备。让你创建一个带有各自设备的 BluetoothDevice 或者查询其皆如名称、地址、类和连接状态等信息。

对于蓝牙硬件地址而言,这个类仅仅是一个瘦包装器。这个类的对象是不可改变的。这个类上的操作会使用这个用来创建 BluetoothDevice 类的 BluetoothAdapter 类执行在远程蓝牙硬件上。

为了获得 BluetoothDevice,类, 使用 BluetoothAdapter.getRemoteDevice(String)方法去创建一个表示 已知 MAC 地址的设备(用户可以通过带有 BluetoothAdapter 类来完成对设备的查找)或者从一个通过 BluetoothAdapter.getBondedDevices()得到返回值的有联系的设备集合来得到该设备。

注意: 需要 [BLUETOOTH](#) 权限

参见

[BluetoothAdapter](#)

[BluetoothSocket](#)

常量

String ACTION_ACL_CONNECTED

广播活动: 指明一个与远程设备建立的低级别 (ACL) 连接。

总是包含 [EXTRA_DEVICE](#) 附加域

ACL 连接通过 Android 蓝牙栈自动进行管理

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.device.action.ACL_CONNECTED"

String ACTION_ACL_DISCONNECTED

广播活动: 指明一个来自于远程设备的低级别 (ACL) 连接的断开

总是包含 [EXTRA_DEVICE](#) 附加域

ACL 连接通过 Android 蓝牙栈自动进行管理

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.device.action.ACL_DISCONNECTED"

`String ACTION_ACL_DISCONNECT_REQUESTED`

广播活动：指明一个为远程设备提出的低级别（ACL）的断开连接请求，并即将断开连接。

对于友好的断开连接，该常量是有作用的。应用程序可以用它作为暗示去马上中断对远程设备的高级别的连接（RFCOMM,L2CAP,或者其它连接）。

总是包含 [EXTRA_DEVICE](#) 附加域

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.device.action.ACL_DISCONNECT_REQUESTED"

`String ACTION_BOND_STATE_CHANGED`

广播活动：指明一个远程设备的连接状态的改变。比如，当一个设备已经被匹配。

总是包含 `EXTRA_DEVICE`, `EXTRA_BOND_STATE` 和 `EXTRA_PREVIOUS_BOND_STATE`.这些附加域。

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.device.action.BOND_STATE_CHANGED"

`String ACTION_CLASS_CHANGED`

广播活动：一个已经改变的远程设备的蓝牙类。

总是包含 `EXTRA_DEVICE` 和 `EXTRA_BOND_STATE` 这些附加域。

需要 [BLUETOOTH](#) 权限接收

参见

[ERROR\(BluetoothClass\) /{@link BluetoothClass}}](#)

常量值: "android.bluetooth.device.action.CLASS_CHANGED"

`String ACTION_FOUND`

广播活动：发现远程设备

当一个远程设备在查找过程中被发现时，发送该常量值。

总是包含 `EXTRA_DEVICE` 和 `EXTRA_CLASS` 这些附加域。如果可用的话，也可包含 `EXTRA_NAME` 和/或 `EXTRA_RSSI` 这些附加域。

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.device.action.FOUND"

`String ACTION_NAME_CHANGED`

广播活动：指明一个远程设备的昵称第一次找到，或者自从最后一次找到该昵称开始已经改变。

总是包含 `EXTRA_DEVICE` 和 `EXTRA_NAME` 这些附加域

需要 [BLUETOOTH](#) 权限接收

常量值: "android.bluetooth.device.action.NAME_CHANGED"

`int BOND_BONDED`

指明远程设备已经匹配。

一个共享的连接键为了远程设备而存在于本地，因而设备间的通讯可以被认证和加密。

和远程设备的匹配并不意味着设备间已经成功连接。它只意味着匹配过程已经在稍早

之前完成，并且连接键已经存储在本地，准备在下次连接的时候使用。

常量值: 12 (0x0000000c)

`int BOND_BONDING`

指明和远程设备的匹配正在进行中

常量值: 11 (0x0000000b)

`int BOND_NONE`

指明远程设备未被匹配。

不存在为了远程设备而已经共享的连接键，因而设备间的通讯（如果完全被允许）不可被认证和加密。

常量值: 10 (0x0000000a)

`Creator<BluetoothDevice> CREATOR`

`int ERROR`

该类的错误标志值。标记该类的错误值。确保和该类中的任意其它整数常量不相等。它为需要一个标记错误值的函数提供了便利。例如:

`Intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAdapter.ERROR)`

常量值: -2147483648 (0x80000000)

`String EXTRA_BOND_STATE`

作为一个 [ACTION_BOND_STATE_CHANGED](#) 的整型附加域。包含了远程设备的匹配状态。

可能值有: BOND_NONE, BOND_BONDING, BOND_BONDED.

常量值: "android.bluetooth.device.extra.BOND_STATE"

`String EXTRA_CLASS`

作为一个 [ACTION_FOUND](#) 和 [ACTION_CLASS_CHANGED](#) 的 Parcelable [BluetoothClass](#) 附加域。

常量值: "android.bluetooth.device.extra.CLASS"

`String EXTRA_DEVICE`

每次通过该类进行广播时，作为 Parcelable BluetoothDevice 的附加域。它包含了该常量适用的 BluetoothDevice 类。

常量值: "android.bluetooth.device.extra.DEVICE"

`String EXTRA_NAME`

作为 ACTION_NAME_CHANGED 和 ACTION_FOUND 的字符串附加域。它包含了这个蓝牙昵称。

常量值: "android.bluetooth.device.extra.NAME"

`String EXTRA_PREVIOUS_BOND_STATE`

作为 ACTION_BOND_STATE_CHANGED 的整型附加域。包含了远程设备以前的匹配状

态。

可能值有: BOND_NONE, BOND_BONDING, BOND_BONDED.

常量值: "android.bluetooth.device.extra.PREVIOUS_BOND_STATE"

`String EXTRA_RSSI`

作为 ACTION_FOUND 的可选短整型附加域。包含了被蓝牙硬件通知的远程设备的 RSSI(Receive Signal Strength Indication, 接收信号强度指示)值。

常量值: "android.bluetooth.device.extra.RSSI"

公共方法

`public BluetoothSocket createRfcommSocketToServiceRecord (UUID uuid)`

该方法是为了使用带有 `listenUsingRfcommWithServiceRecord(String, UUID)` 方法来进行对等的蓝牙应用而设计的。

使用 `connect()` 初始化这个外界连接。它也将执行一个已给与 UUID 的 SDP 查找，从而确定连接到哪个通道上。

远程设备将被认证，在这个端口上的通讯会被加密。

提示：如果你正试图连接蓝牙串口，那么使用众所周知的 SPP UUID 00001101-0000-1000-8000-00805F9B34FB。但是你如果正试图连接 Android 设备那么请你生成你自己的专有 UUID。

需要 [BLUETOOTH](#) 权限。

参数

`uuid` 查询 RFCOMM 通道的服务记录 UUID

返回值

一个准备好外界连接的 RFCOMM 蓝牙服务端口

异常

`IOException` 出现错误，比如蓝牙模块不可用，或者许可无效。

`public int describeContents ()`

描述了包含在 Parcelable's marshalled representation 中的特殊对象的种类。

返回值

一个指示被 Parcelabel 所排列的特殊对象类型集合的位屏蔽。

`public boolean equals (Object o)`

比较带有特定目标的常量。如果他们相等则标示出来。为了保证其相等，`o` 必须代表相同的对象，该对象作为这个使用类依赖比较的常量。通常约定，该比较既需要可复制、相等和可传递。另外，没有对象引用的时候 `null` 等于 `null`。

默认实现是返回 `true`，仅当 `this == o`。如果你想实现你自己的 `equals` 方法，参见 [Writing a correct equals method](#)。

当且仅当 `o` 是一个作为接收器（使用`==`操作符来做比较）的精确相同的对象是，这个对象的实现才返回 `true` 值。子类通常实现 `equals(Object)` 方法，这样它才会重视这两个对象的类型和状态。

通常约定，对于 `equals(Object)` 和 `hashCode()` 方法，如果 `equals` 对于任意两个对象返回真值，那么 `hashCode()` 必须对这些对象返回相同的值。这意味着对象的子类通常都覆盖或者都不覆盖这两个方法。

参数

o 需要对比常量的对象

返回值

如果指定的对象和该对象相等则返回 true，否则返回 false。

public String getAddress ()

返回该蓝牙设备的硬件地址

例如： "00:11:22:AA:BB:CC".

返回值

字符串类型的蓝牙硬件地址

public BluetoothClass getBluetoothClass ()

获取远程设备的蓝牙类

需要 [BLUETOOTH](#) 权限。

返回值

蓝牙类对象出错时返回空值

public int getBondState ()

获取远程设备的连接状态。

连接状态的可能值有: BOND_NONE, BOND_BONDING, BOND_BONDED.

需要 [BLUETOOTH](#) 权限。

返回值

连接状态。

public String getName ()

获取远程设备的蓝牙昵称。

当执行设备扫描的时候，本地适配器将自动寻找远程名称。该方法只返回来自存储器中该设备的名称。

需要 [BLUETOOTH](#) 权限。

返回值

蓝牙昵称，如果出现问题则返回 null。

public int hashCode ()

返回该对象的一个整型哈希值。通常约定，如果 equals 对于任意两个对象返回真值，那么 hashCode() 必须对这些对象返回相同的值。这意味着对象的子类通常都覆盖或者都不覆盖这两个方法。

注意：除非同等对比信息发生改变，否则哈希码不随时间改变而改变。

如果你想要实现你自己的哈希码方法，参见 [Writing a correct hashCode method](#)。

返回值

该对象的哈希值

public String toString ()

返回该蓝牙设备的字符串表达式。

这是一个蓝牙硬件地址，例如"00:11:22:AA:BB:CC".然而，如果用户明确需要蓝牙硬件

地址以防以后 `toString()` 表达式会改变的话，用户总是需要使用 `getAddress()` 方法。

返回值

该蓝牙设备的字符串表达式。

`public void writeToParcel (Parcel out, int flags)`

将类的数据写入外部提供的 `Parcel` 中

参数

`out` 对象需要被写入的 `Parcel`

`flags` 和对象需要如何被写入有关的附加标志。可能是 0，或者可能是

补充

文章精选

[Android 提高第十二篇之蓝牙传感能用](#)

[Android 提高第十三篇之探秘蓝牙隐藏 API](#)

BluetoothServerSocket

译者署名: Android Club SYSU

译者链接: <http://www.android-wiki.net>

版本: Android 2.3 r1

结构

继承关系

public final class BluetoothServerSocket extends Object implements Closeable

```
java.lang.Object  
    android.bluetooth.BluetoothServerSocket
```

类概述

一个蓝牙监听端口。

蓝牙端口监听接口和 TCP 端口类似: `Socket` 和 `ServerSocket` 类。在服务器端, 使用 `BluetoothServerSocket` 类来创建一个 监听服务端口。当一个连接被 `BluetoothServerSocket` 所接受, 它会返回一个新的 `BluetoothSocket` 来管理该连接。在客户 端, 使用一个单独的 `BluetoothSocket` 类去初始化一个外接连接和管理该连接。

最通常使用的蓝牙端口是 `RFCOMM`, 它是被 Android API 支持的类型。`RFCOMM` 是一个面向连接, 通过蓝牙模块进行的数据流传输方式, 它也被称为串行端口规范 (Serial Port Profile, SPP)。

为了创建一个对准备好的新来的连接去进行监听 `BluetoothServerSocket` 类, 使用 `BluetoothAdapter.listenUsingRfcommWithServiceRecord()` 方法。然后调用 `accept()`方法去监 听该链接的请求。在连接建立之前, 该调用会被阻断, 也就是说, 它将返回一个 `BluetoothSocket` 类去管理该连接。每次获得该类之后, 如果不再需 要接受连接, 最好调用在 `BluetoothServerSocket` 类下的 `close()`方法。关闭 `BluetoothServerSocket` 类不会关 闭这个已经返回的 `BluetoothSocket` 类。

`BluetoothSocket` 类线程安全。特别的, `close()`方法总会马上放弃外界操作并关闭服务器 端口。

注意: 需要 `BLUETOOTH` 权限。

参见

[BluetoothSocket](#)

公共方法

`public BluetoothSocket accept (int timeout)`

阻塞直到超时时间内的连接建立。

在一个成功建立的连接上返回一个已连接的 `BluetoothSocket` 类。

每当该调用返回的时候, 它可以在此调用去接收以后新来的连接。

`close()`方法可以用来放弃从另一线程来的调用。

参数

`timeout` (译者注: 阻塞超时时间)

返回值

已连接的 `BluetoothSocket`

异常

IOException 出现错误，比如该调用被放弃，或者超时。

public BluetoothSocket accept ()

阻塞直到一个连接已经建立。（译者注：默认超时时间设置为-1，见源码）

在一个成功建立的连接上返回一个已连接的 BluetoothSocket 类。

每当该调用返回的时候，它可以在此调用去接收以后新来的连接。

close()方法可以用来放弃从另一线程来的调用。

返回值

已连接的 BluetoothSocket

异常

IOException 出现错误，比如该调用被放弃，或者超时。

public void close ()

马上关闭端口，并释放所有相关的资源。

在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。

关闭 BluetoothServerSocket 不会关闭接受自 accept() 的任意 BluetoothSocket。

异常

IOException

BluetoothSocket

译者署名： Android Club SYSU

译者链接：<http://www.android-wiki.net>

版本：Android 2.3 r1

结构

继承关系

public static class *Gallery.LayoutParams* extends *ViewGroup.LayoutParams*

```
java.lang.Object  
    android.view.ViewGroup.LayoutParams  
        android.widget.Gallery.LayoutParams
```

类概述

已连接或连接到蓝牙套接字(socket)。

蓝牙端口监听接口和 TCP 端口类似： Socket 和 ServerSocket 类。在服务器端，使用 BluetoothServerSocket 类来创建一个 监听服务端口。当一个连接被 BluetoothServerSocket 所接受，它会返回一个新的 BluetoothSocket 来管理该连接。在客户 端，使用一个单独的 BluetoothSocket 类去初始化一个外接连接和管理该连接。

最通常使用的蓝牙端口是 RFCOMM，它是被 Android API 支持的类型。RFCOMM 是一个面向连接，通过蓝牙模块进行的数据流传输方式，它也被称为串行端口规范（Serial Port Profile，SPP）。

为了创建一个 BluetoothSocket 去连接到一个已知设备，使用方法 BluetoothDevice.createRfcommSocketToServiceRecord()。然后调用 connect()方法去尝试一个面向远程设备的连接。这个调用将被阻塞指导一个连接已经建立或者该链接失效。

为了创建一个 BluetoothSocket 作为服务端（或者“主机”），查看 BluetoothServerSocket 文档。

每当该端口连接成功，无论它初始化为客户端，或者被接受作为服务器端，通过 getInputStream()和 getOutputStream()来打开 IO 流，从而获得各自的 InputStream 和 OutputStream 对象

BluetoothSocket 类线程安全。特别的，close()方法总会马上放弃外界操作并关闭服务器端口。

注意：需要 [BLUETOOTH](#) 权限。

参见

[BluetoothServerSocket](#)
[InputStream](#)
[OutputStream](#)

公共方法

public void *close* ()

马上关闭该端口并且释放所有相关的资源。

在其它线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。

异常

IOException

public void connect ()

尝试连接到远程设备。

该方法将阻塞，指导一个连接建立或者失效。如果该方法没有返回异常值，则该端口现在已经建立。

当设备查找正在进行的时候，创建对远程蓝牙设备的新连接不可被尝试。在蓝牙适配器上，设备查找是一个重量级过程，并且肯定会降低一个设备的连接。使用 [cancelDiscovery\(\)](#) 方法去取消一个外界的查询。查询并不由活动所管理，而作为一个系统服务来运行，所以即使它不能直接请求一个查询，应用程序也总会调用 [cancelDiscovery\(\)](#) 方法。

[close\(\)](#) 方法可以用来放弃从另一线程而来的调用。

异常

IOException 一个错误，例如连接失败。

public InputStream getInputStream ()

通过连接的端口获得输入数据流

即使该端口未连接，该输入数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值

输入流

异常

IOException

public OutputStream getOutputStream ()

通过连接的端口获得输出数据流

即使该端口未连接，该输出数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值

输出流

异常

IOException

public BluetoothDevice getRemoteDevice ()

获得该端口正在连接或者已经连接的远程设备。

返回值

远程设备

补充

文章精选

[第四十一讲：Android 蓝牙编程入门](#)

[Android 蓝牙 API 之 BluetoothSocket 类](#)

[Android 蓝牙 API 之 BluetoothSocket 类\(2\)](#)

android.content

类

[SharedPreferences](#)

SharedPreferences

译者署名: madgoat

译者链接: <http://madgoat.cn>

翻译版本: 2010-11-1

结构

继承关系

public interface SharedPreferences

 android.content.SharedPreferences

类概述

用于访问和修改 [getSharedPreferences\(String, int\)](#) 返回偏好设置数据(preference data)的一个接口。对于任何一组特殊的 preferences，所有的客户端共享一个此类单独的实例。

修改 Preferences 必须通过一个 [SharedPreferences.Editor](#) 对象，以确保当他们提交存储数据的操作时，preference 值保持一致的状态。

注意: 当前此类不支持多线程访问。后续将添加。

(译者注: 这里译为“偏好设定”，类似于 ini 文件，用于保存应用程序的属性设置)

参见

[getSharedPreferences\(String, int\)](#)

内部类

interface SharedPreferences.Editor

用于修改 SharedPreferences 对象设定值的接口。

interface SharedPreferences.OnSharedPreferenceChangeListener

接口定义一个用于在偏好设定(shared preference)改变时调用的回调函数。

公共方法

public abstract boolean contains (String key)

判断 preferences 是否包含一个 preference。

参数

key 想要判断的 preference 的名称

返回值

如果 preferences 中存在 preference，则返回 true，否则返回 false。

public abstract SharedPreferences.Editor edit ()

针对 preferences 创建一个新的 Editor 对象，通过它你可以修改 preferences 里的数据，并且原子化的将这些数据提交回 SharedPreferences 对象。(译者注: 原子化——作为一个整体提交，原子性)

注意: 如果你想要在 SharedPreferences 中实时显示，刚通过 Editor 对象进行的修改，那么你必须调用 [commit\(\)](#) 方法。

返回值

返回一个 [SharedPreferences.Editor](#) 的新实例，允许你修改 SharedPreferences 对

象里的值。

public abstract Map<String, ?> getAll ()

取得 preferences 里面的所有值

返回值

返回一个 map，其中包含一列 preferences 中的键值对

异常

空指针异常 (NullPointerException)

public abstract boolean getBoolean (String key, boolean defValue)

从 preferences 中获取一个 boolean 类型的值。

参数

key 获取的 preference 的名称

defValue 当此 preference 不存在时返回的默认值

返回值

如果 preference 存在，则返回 preference 的值，否则返回 defValue。如果使用此名称的 preference 不是一个 boolean 类型，则抛出 ClassCastException。

异常

[ClassCastException](#)

public abstract float getFloat (String key, float defValue)

从 preferences 中获取一个 float 类型的值。

参数

key 获取的 preference 的名称

defValue 当此 preference 不存在时返回的默认值

返回值

如果 preference 存在，则返回 preference 的值，否则返回 defValue。如果使用此名称的 preference 不是一个 float 类型，则抛出 ClassCastException。

异常

[ClassCastException](#)

public abstract int getInt (String key, int defValue)

从 preferences 中获取一个 int 类型的值。

参数

key 获取的 preference 的名称

defValue 当此 preference 不存在时返回的默认值

返回值

如果 preference 存在，则返回 preference 的值，否则返回 defValue。如果使用此名称的 preference 不是一个 int 类型，则抛出 ClassCastException。

异常

[ClassCastException](#)

public abstract long getLong (String key, long defValue)

从 preferences 中获取一个 long 类型的值。

参数

key 获取的 preference 的名称

defValue 当此 preference 不存在时返回的默认值

返回值

如果 preference 存在，则返回 preference 的值，否则返回 defValue。如果使用此名称的 preference 不是一个 long 类型，则抛出 ClassCastException。

异常

[ClassCastException](#)

```
public abstract String getString (String key, String defValue)
```

从 preferences 中获取一个 String 类型的值。

参数

key 获取的 preference 的名称

defValue 当此 preference 不存在时返回的默认值

返回值

如果 preference 存在，则返回 preference 的值，否则返回 defValue。如果使用此名称的 preference 不是一个 String 类型，则抛出 ClassCastException。

异常

[ClassCastException](#)

```
public abstract void registerOnSharedPreferenceChangeListener
```

```
(SharedPreferences.OnSharedPreferenceChangeListener listener)
```

注册一个回调函数，当一个 preference 发生变化时调用。

参数

listener 将会被调用的回调函数

参见

[unregisterOnSharedPreferenceChangeListener\(SharedPreferences.OnSharedPreferenceChangeListener\)](#)

```
public abstract void unregisterOnSharedPreferenceChangeListener
```

```
(SharedPreferences.OnSharedPreferenceChangeListener listener)
```

注销一个之前(注册)的回调函数

参数

listener 要被注销的回调函数

参见

[registerOnSharedPreferenceChangeListener\(SharedPreferences.OnSharedPreferenceChangeListener\)](#)

补充

文章精选

[SharedPreferences](#)

[\[Android 开发者指南\] 第十八讲：Android SharedPreferences 和 File](#)

[SharedPreferences 用法](#)

[Android 程式設計 \(十五\) 使用 SharedPreferences](#)

示例代码

译注： Shared Preferences 保存位置： /data/data/app_name/shared_prefs/*.xml
private boolean flag = false;

```
//取得活动的 Preferences 对象
SharedPreferences settings = getPreferences(Activity.MODE_PRIVATE);
//取得值
flag = settings.getBoolean("flag",false);

//取得活动的 Preferences 对象
SharedPreferences settings = getPreferences(0);
//取得编辑对象
SharedPreferences.Editor editor = settings.edit();
//添加值
editor.putBoolean("true",flag);
//提交保存
editor.commit();
```

android.media

类

[AsyncPlayer](#)

[ThumbnailUtils](#)

AsyncPlayer

译者署名： 农民伯伯

译者链接：<http://over140.cnblogs.com/>

版本：Android 3.0 r1

结构

继承关系

public class AsyncPlayer extends Object

java.lang.Object
 android.media.AsyncPlayer

类概述

播放一个连续(多个)的音频 URLs，但那些任务较重的工作在另外的线程中完成，所以任何预处理或加载的延迟都不阻碍线程调用。

构造函数

public AsyncPlayer (String tag)

构造一个 AsyncPlayer 对象。

参数

tag 用于调试的字符串

公共方法

public void play (Context context, Uri uri, boolean looping, int stream)

开始播放声音。可在某个点上开始播放。这里不保证可能有延迟。在另一个音频文件播放时调用这个方法将导致当前音频停止播放并开始播放新的音频。

参数

context 应用程序上下文

uri 播放的 URI (参见 [setDataSource\(Context, Uri\)](#))

looping 是否无限循环播放声音。 (参见 [setLooping\(boolean\)](#))

stream 音频流(AudioStream)类型 (参见 [setAudioStreamType\(int\)](#)) (译者注：

例如 AudioManager. STREAM_MUSIC)

public void stop ()

停止之前播放的声音。不能在某点上暂停然后接着播放。多次调用没有不良影响。

补充

文章精选

[android 多媒体----AsyncPlayer](#)

[Android 游戏开发之旅 16 异步音乐播放](#)[Android123]

[今天犯了一个愚蠢的错误](#)

ThumbnailUtils

译者署名： 农民伯伯

译者链接：<http://over140.cnblogs.com/>

版本：Android 3.0 r1

结构

继承关系

public class ThumbnailUtils extends Object

java.lang.Object
 android.media.ThumbnailUtils

类概述

为媒体生成常规缩略图。(Android123：该类为Android2.2新增类，可以帮助我们从mediaprovider中获取系统中的视频或图片文件的缩略图)

常量

public static final int OPTIONS_RECYCLE_INPUT

常量用于表示应该回收 [extractThumbnail\(Bitmap, int, int, int\)](#)输入源图片(第一个参数)，除非输出图片就是输入图片。

常量值：2 (0x00000002)

公共方法

public static Bitmap createVideoThumbnail (String filePath, int kind)

创建一张视频的缩略图。如果视频已损坏或者格式不支持可能返回 null。

参数

filePath 视频文件路径

kind 可以为 MINI_KIND 或 MICRO_KIND

public static Bitmap extractThumbnail (Bitmap source, int width, int height, int options)

创建所需尺寸居中缩放的位图。

参数

source 原始位图源

width 目标宽

height 目标高

options 在缩略图抽取时提供的选项

public static Bitmap extractThumbnail (Bitmap source, int width, int height)

创建所需尺寸居中缩放的位图。

参数

source 原始位图源

width 目标宽

height 目标高

补充

文章链接

[ThumbnailUtils - Android2.2 新增类 \(Android123\)](#)

[推荐][Android 缩略图类源代码 \(Android123\)](#)

android.net

类

[TrafficStats](#)

[MailTo](#)

TrafficStats

译者署名： 农民伯伯

译者链接：<http://over140.cnblogs.com/>

版本：Android 2.3 r1

结构

继承关系

public class TrafficStats extends Object

java.lang.Object

 android.net.TrafficStats

类概述

提供网络流量统计的类。这些统计包括通过所有网络接口、mobile 接口和 UID 网络接口的字节发送和接收，网络数据包的发送和接收。

这些统计可能不适用于所有平台。如果本设备不支持统计，[UNSUPPORTED](#) 将被返回。

常量

public static final int UNSUPPORTED

返回值表示该设备不支持统计。

常量值: -1 (0xffffffff)

公共方法

public static long getMobileRxBytes ()

获取通过 Mobile 接口接收到的字节总数（Android123：这里不包含 WiFi）

返回值

字节总数。如果本设备不支持统计，将返回 [UNSUPPORTED](#)。

public static long getMobileRxPackets ()

获取通过 Mobile 接口接收到的数据包总数

返回值

数据包总数。如果本设备不支持统计，将返回 [UNSUPPORTED](#)。

public static long getMobileTxBytes ()

获取通过 Mobile 接口发送的字节总数

返回值

字节总数。如果本设备不支持统计，将返回 [UNSUPPORTED](#)。

public static long getMobileTxPackets ()

获取通过 Mobile 接口发送的数据包总数

返回值

数据包总数。如果本设备不支持统计，将返回 [UNSUPPORTED](#)。

public static long getTotalRxBytes ()

获取通过所有网络接口接收到的字节总数。(Android123: 包含 Mobile 和 WiFi 等)

返回值

字节总数。如果本设备不支持统计, 将返回 [UNSUPPORTED](#)。

public static long getTotalRxPackets ()

获取通过所有网络接口接收到的数据包总数。(Android123: 包含 Mobile 和 WiFi 等)

返回值

数据包总数。如果本设备不支持统计, 将返回 [UNSUPPORTED](#)。

public static long getTotalTxBytes ()

获取通过所有网络接口发送的字节总数。(Android123: 包含 Mobile 和 WiFi 等)

返回值

字节总数。如果本设备不支持统计, 将返回 [UNSUPPORTED](#)。

public static long getTotalTxPackets ()

获取通过所有网络接口发送的数据包总数 (Android123: 包含 Mobile 和 WiFi 等)

返回值

数据包总数。如果本设备不支持统计, 将返回 [UNSUPPORTED](#)。

public static long getUidRxBytes (int uid)

获取通过 UID 网络接口收到的字节数。统计包含所有网络接口。

参数

uid 待检查的进程的 uid

返回值

字节数

参见

[myUid\(\)](#)

public static long getUidTxBytes (int uid)

获取通过 UID 网络接口发送的字节数。统计包含所有网络接口。

参数

uid 待检查的进程的 uid

返回值

字节总数。如果本设备不支持统计, 将返回 [UNSUPPORTED](#)。

参见

[myUid\(\)](#)

补充

文章链接

[Android 流量统计 TrafficStats 类的使用](#)

MailTo

译者署名： 农民伯伯

译者链接：<http://over140.cnblogs.com/>

版本：Android 3.0 r1

结构

继承关系

public class MailTo extends Object

java.lang.Object

 android.net.MailTo

类概述

MailTo URL 解析器。这个类解析到 URL 计划的邮件(a mailto schema URL 参见[这里](#))和可用于解析参数查询。它实现了 RFC 2368 协议。

常量

public static final String MAILTO_SCHEME

常量值："mailto:"

公共方法

public String getBody ()

从一个 Url 中获取邮件的正文内容。如果没有设置正文内容，则返回 null 值。

返回值

返回正文或 null

public String getCc ()

从一个 Url 中获取抄送地址。这可能是多个电子邮件地址，以逗号分隔的空间。如果没有指定抄送地址，则返回 null 值

返回值

逗号分隔的邮件地址或 null

public Map<String, String> getHeaders ()

从一个 Url 中获取邮件头（Android123：比如编码类型，发送时间、IP 等）

返回值

包含解析值的 map

public String getSubject ()

从一个 Url 中获取主题。如果没有指定的主题行，则返回 null 值。

返回值

主题或 null

public String getTo ()

从一个 Url 中获取收信人。这可能是多个电子邮件地址，以逗号分隔的空间。如果没指定，则返回 null 值。

返回值

逗号分隔的电子邮件地址或 null。

public static boolean isMailTo (String url)

测试判断指定字符串是否包含 Email 的 Url。

参数

url 用于测试的字符串

返回值

如果为 true，字符串是包含 Email 的 Url。

public static MailTo parse (String url)

将指定 Url 字符串解析成 MailTo 类。这个解析器实现的 RFC 2368。返回的对象可以用于查询解析的参数。

参数

url 包含 Email 的 Url 字符串

返回值

MailTo 对象

异常

[ParseException](#) 如果该结构并不是包含 Email 的 Url 将抛出此异常。

public String toString ()

返回一个包含关于对象简洁的、大家都可以读懂的字符串。鼓励子类重写此方法，并提供一个考虑到对象的类型和数据的实现。默认的实现价于下面的表达式：

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

如果你打算实现自己 `toString` 方法，参见 [Writing a useful `toString` method](#)。

返回值

表示此对象的可打印输出。

补充

文章链接

[电子邮件解析 android.net.MailTo 类\(本文有参照该链接的翻译\)](#)

android.os

类

[AsyncTask](#)

枚举

[AsyncTask.Status](#)

AsyncTask

译者署名： 0_1

译者链接: <http://dev.10086.cn/blog/?32546>

版本: Android 2.3 r1

结构

继承关系

public abstract class AsyncTask extends Object

java.lang.Object

 android.os.AsyncTask<Params, Progress, Result>

类概述

AsyncTask 能够适当地、简单地用于 UI 线程。 这个类不需要操作线程(Thread)就可以完成后台操作将结果返回 UI。

异步任务的定义是一个在后台线程上运行，其结果是在 UI 线程上发布的计算。 异步任务被定义成三种泛型类型： Params, Progress 和 Result; 和四个步骤： begin , doInBackground, processProgress 和 end。

用法

AysncTask 必须被继承使用。子类至少覆盖一个方法 ([doInBackground\(Params...\)](#))，最终常覆盖另一个([onPostExecute\(Result\)](#))下面是一个子类的例子：

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

一旦创建，一个任务执行起来就非常简单：

```
new DownloadFilesTask().execute(url1, url2, url3);
```

AsyncTask 的泛型类型

这三个类型被用于一个异步任务，如下：

1. Params, 启动任务执行的输入参数
2. Progress, 后台任务执行的百分比
3. Result, 后台计算的结果类型

在一个异步任务里，不是所有的类型总被用。假如一个类型不被使用，可以简单地使用 [Void](#) 类型：

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

4 个步骤

当一个异步任务被执行，任务经过四各步骤：

1. [onPreExecute\(\)](#), 在 UI 线程上调用任务后立即执行。这步通常被用于设置任务，例如在用户界面显示一个进度条。
2. [doInBackground\(Params...\)](#), 后台线程执行 `onPreExecute()` 完后立即调用，这步被用于执行较长时间的后台计算。异步任务的参数也被传到这步。计算的结果必须在这步返回，将传回到上一步。在执行过程中可以调用 [publishProgress\(Progress...\)](#) 来更新任务的进度。
3. [onProgressUpdate\(Progress...\)](#), 一次呼叫 [publishProgress\(Progress...\)](#) 后调用 UI 线程。执行时间是不确定的。这个方法用于当后台计算还在进行时在用户界面显示进度。例如：这个方法可以被用于一个进度条动画或在文本域显示记录。
4. [onPostExecute\(Result\)](#), 当后台计算结束时，调用 UI 线程。后台计算结果作为一个参数传递到这步。

线程规则

有一些线程规则必须去遵守，这个类才会正确的工作：

- 任务实例必须创建在 UI 线程
- [execute\(Params...\)](#) 必须在 UI 线程上调用
- 不要手动调用 [onPreExecute\(\)](#), [onPostExecute\(Result\)](#), [doInBackground\(Params...\)](#), [onProgressUpdate\(Progress...\)](#)
- 这个任务只执行一次（如果执行第二次将会抛出异常）

内部类

```
enum AsyncTask.Status
```

表示任务的当前状态

构造函数

```
public AsyncTask ()
```

创建一个新的异步任务。这个构造函数必须在 UI 线程上调用。

公共方法

```
public final boolean cancel (boolean mayInterruptIfRunning)
```

尝试取消这个任务的执行，如果这个任务已经结束或者已经取消或者不能被取消或者某些其他原因，那么将导致这个操作失败，当调用此方法时，此方法执行成功并且这个任务还没有执行，那么此任务将不再执行。如果任务已经开始，这时执行此操作传入的参数

`mayInterruptIfRunning` 为 `true`, 执行此任务的线程将尝试中断该任务。

参数

`mayInterruptIfRunning` 如果为 `true` 则正在执行的线程将会中断, 如果 `false`, 则会允许正在执行的任务线程执行完毕。

返回值

如果此任务不能取消返回 `false`, 如果已经正常的执行完毕, 返回 `true`

参见

[isCancelled\(\)](#)

[onCancelled\(\)](#)

public final AsyncTask<Params, Progress, Result> execute (Params... params)

用指定的参数来执行此任务, 这个方法将会返回此任务本身, 所以调用者可以拥有此任务的引用。此方法必须在 UI 线程中调用

参数

`params` 任务参数

返回值

`AsyncTask` 的实例

异常

`IllegalStateException` 如果 `getStatus()` 返回的是 `RUNNING` 或者 `FINISHED`

public final Result get ()

等待计算结束并返回结果

返回值

计算结果

异常

`CancellationException` 如果计算取消

`ExecutionException` 如果计算抛出异常

`InterruptedException` 当等待时当前线程抛出异常

public final Result get (long timeout, TimeUnit unit)

等待计算结束并返回结果, 最长等待时间为: `timeOut` (超时时间) .

参数

`timeout` 计算等待超时时间

`unit` 超时的时间单位

返回值

计算结果

异常

`CancellationException` 如果计算取消

`ExecutionException` 如果计算抛出异常

`InterruptedException` 当等待时当前线程抛出异常

`TimeoutException` 等待时间超时

public final AsyncTask.Status getStatus ()

获得任务的当前状态

返回值

当前状态

public final boolean isCancelled ()

如果在任务正常结束之前取消任务成功则返回 true，否则返回 false

返回值

如果任务正常结束之前取消任务成功返回 true。

参见

[cancel\(boolean\)](#)

受保护方法

protected abstract Result doInBackground (Params... params)

覆盖此方法在后台线程执行计算，此方法中的参数是此任务的 [execute\(Params...\)](#) 方法的调用传递的参数，此方法可以调用 [publishProgress\(Progress...\)](#) 在 UI 线程中来更新数据

参数

params 此任务的参数

返回值

返回一个由此任务子类定义的结果 Result

参见

[onPreExecute\(\)](#)

[onPostExecute\(Result\)](#)

[publishProgress\(Progress...\)](#)

protected void onCancelled ()

此方法在 UI 线程中当 [cancel\(boolean\)](#) 被调用后调用

参见

[cancel\(boolean\)](#)

[isCancelled\(\)](#)

protected void onPostExecute (Result result)

此方法在 UI 线程中 [doInBackground\(Params...\)](#)。方法调用之后调用，此方法中的参数的值是 [doInBackground\(Params...\)](#) 的返回值或者当此任务已经被取消或有异常发生时此参数值为空 null

参数

result 由 [doInBackground\(Params...\)](#) 计算出的操作的结果。

参见

[onPreExecute\(\)](#)

[doInBackground\(Params...\)](#)

protected void onPreExecute ()

在方法 [doInBackground\(Params...\)](#) 调用之前调用

参见

[onPostExecute\(Result\)](#)

[doinBackground\(Params...\)](#)

protected void onProgressUpdate (Progress... values)

该方法在 UI 线程中 [publishProgress\(Progress...\)](#)被调用之后调用，该方法中的参数 values 是已经被传递到 [publishProgress\(Progress...\)](#)中的参数

参数

values 进度表示值

参见

[publishProgress\(Progress...\)](#)

[doinBackground\(Params...\)](#)

protected final void publishProgress (Progress... values)

当调用 [doinBackground\(Params...\)](#)在后台执行计算时会调用该方法，每当在 UI 线程中调用此方法时将触发 [onProgressUpdate\(Progress...\)](#)方法的执行

参数

values 将进度值更新到 UI

参见

[onProgressUpdate\(Progress...\)](#)

[doinBackground\(Params...\)](#)

补充

文章精选

[Android AsyncTask 理解](#)

[TabActivity 下在 AsyncTask 中使用 ProgressDialog 存在问题的解决方法](#)

[小心， AsyncTask 不是万能的](#) [blog spot]

AsyncTask.Status

译者署名: 0_1

译者链接: <http://dev.10086.cn/blog/?32546>

版本: Android 2.3 r1

结构

继承关系

public static final enum AsyncTask.Status extends Enum<E extends Enum<E>>

```
java.lang.Object  
  java.lang.Enum<E extends java.lang.Enum<E>>  
    android.os.AsyncTask.Status
```

类概述

标志任务的当前状态，每个状态在任务的生命周期中只会出现一次。

枚举值

public static final AsyncTask.Status FINISHED

标志 [onPostExecute\(Result\)](#)方法已经结束

public static final AsyncTask.Status PENDING

标志任务还没有执行

public static final AsyncTask.Status RUNNING

标志任务正在执行

android.view

接口

[ContextMenu](#)
[ContextMenu.ContextMenuInfo](#)
GestureDetector.OnDoubleTapListener
GestureDetector.OnGestureListener
InputQueue.Callback
KeyEvent.Callback
LayoutInflater.Factory
LayoutInflater.Filter
Menu
MenuItem
MenuItem.OnMenuItemClickListener
ScaleGestureDetector.OnScaleGestureListener
SubMenu
SurfaceHolder
SurfaceHolder.Callback
SurfaceHolder.Callback2
View.OnClickListener
View.OnCreateContextMenuListener
View.OnFocusChangeListener
View.OnKeyListener
View.OnLongClickListener
View.OnTouchListener
ViewGroup.OnHierarchyChangeListener
[ViewManager](#)
[ViewParent](#)
ViewStub.OnInflateListener
ViewTreeObserver.OnGlobalFocusChangeListener
ViewTreeObserver.OnGlobalLayoutListener
ViewTreeObserver.OnPreDrawListener
ViewTreeObserver.OnScrollChangedListener
ViewTreeObserver.OnTouchModeChangeListener
Window.Callback
[WindowManager](#)

类

AbsSavedState
ContextThemeWrapper
Display
FocusFinder
[GestureDetector](#)
GestureDetector.SimpleOnGestureListener

Gravity
HapticFeedbackConstants
InputDevice
InputDevice.MotionRange
InputEvent
InputQueue
KeyCharacterMap
KeyCharacterMap.KeyData
KeyEvent
KeyEvent.DispatcherState
LayoutInflater
[MenuInflater](#)
MotionEvent
MotionEvent.PointerCoords
OrientationEventListener
OrientationListener
[ScaleGestureDetector](#)
ScaleGestureDetector.SimpleOnScaleGestureListener
[SoundEffectConstants](#)
Surface
SurfaceView
TouchDelegate
VelocityTracker
[View](#) (部分)
View.BaseSavedState
View.MeasureSpec
ViewConfiguration
ViewDebug
ViewGroup
ViewGroup.LayoutParams
ViewGroup.MarginLayoutParams
[ViewStub](#)
ViewTreeObserver
Window
 WindowManager.LayoutParams

枚举

ViewDebug.HierarchyTraceType
ViewDebug.RecyclerTraceType

异常

InflateException
Surface.OutOfResourcesException
SurfaceHolder.BadSurfaceTypeException
WindowManager.BadTokenException

ContextMenu

译者署名： Kun
版本： Android 2.3 r1

结构

继承关系

public interface ContextMenu implements Menu

android.view.ContextMenu

类概述

扩展自 [Menu](#) 的上下文菜单提供了修改上下文菜单头(header)的功能。(译者注：当一个视图注册了上下文菜单时，执行一个在该对象上长按(2秒)的动作，将出现一个具有相关功能的浮动菜单。)

上下文菜单不支持菜单项的快捷方式和图标。

当执行长按上下文菜单时，大多数情况会调用 [registerForContextMenu\(View\)](#) 函数和重写执行 [onCreateContextMenu\(ContextMenu, View, ContextMenu.ContextMenuItemInfo\)](#) 函数。(译者注：因为要创建一个上下文菜单，你必须重写这个活动的上下文回调函数 [onCreateContextMenu\(\)](#) 并且通过 [registerForContextMenu\(View\)](#) 为其注册上下文菜单。)

内部类

interface ContextMenu.ContextMenuItemInfo

获得更多关于创建上下文菜单的信息。(译者注：例如：AdapterViews 使用这个类可以精确选择 adapter 的位置来启动上下文菜单。)

公共方法

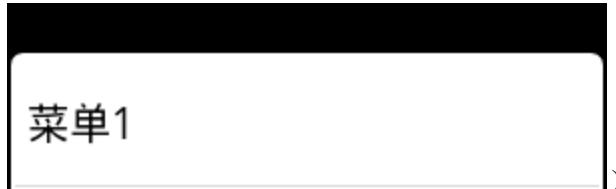
public abstract void clearHeader ()

清除上下文菜单头的信息。(译者注：包括图片和文字信息)



菜单1

Menu.clearHeader(); 后



public abstract ContextMenu setHeaderIcon (Drawable icon)

为上下文菜单头设置图标

参数

icon 你要使用的 [Drawable](#)

返回值

调用你设置修改的上下文菜单

```
public abstract ContextMenu setHeaderIcon (int iconRes)
```

设置上下文菜单头图标为指定的资源 id

参数

iconRes 你要使用的图标资源的目录

(译者注: 把图标放入 res/drawable/ 目录下, R 文件会自动生成对应项。设置方法如 menu.setHeaderIcon(R.drawable.webtext)

这个上下文菜单头是没有设置图标的



菜单1

这个上下文菜单头是设置了图标的



菜单1

)

返回值

调用你设置修改过的上下文菜单

```
public abstract ContextMenu setHeaderTitle (int titleRes)
```

通过资源标识符为上下文菜单头的标题栏设置文字。(译者注: 需要在 res/string 中先设置一段你需要的文字, 如: <string name="titletest">这是一段测试文字</string>

然后通过 R 文件索引到这段文字, menu.setHeaderTitle(R.string.titletest))

参数

titleRes 所需文字资源的索引

返回值

调用你设置修改过的上下文菜单

```
public abstract ContextMenu setHeaderTitle (CharSequence title)
```

设置上下文菜单的标题, 显示在标题栏

参数

title 标题要显示的文字

返回值

调用你设置修改过的上下文菜单

```
public abstract ContextMenu setHeaderView (View view)
```

设置 View 到上下文菜单头上。将替代上下文菜单头的图标和标题（或者替代你之前设置的 headerView）

参数

view 上下文菜单头要使用的 View

返回值

调用你设置修改过的上下文菜单内容

补充

文章精选：

[Android 的上下文菜单： Context Menu](#)

[android 的 ContextMenu](#)

[android 上下文菜单 Context Menu](#)

代码示例：

Test_Contextmenu.java

```
public class Test_Contextmenu extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        TextView txt1 = (TextView) this.findViewById(R.id.txt1);  
        this.registerForContextMenu(txt1);  
        TextView txt2 = (TextView) this.findViewById(R.id.txt2);  
        this.registerForContextMenu(txt2);  
    }  
    // 重写 onCreateContextMenu 用以创建上下文菜单  
    @Override  
    public void onCreateContextMenu(ContextMenu menu, View v,  
        ContextMenuItemInfo menuInfo){  
        super.onCreateContextMenu(menu, v, menuInfo);  
        // 创建 R.id.txt1 的上下文菜单  
        if (v == (TextView) this.findViewById(R.id.txt1)) {  
            menu.setHeaderIcon(R.drawable.icon);  
            menu.setHeaderTitle(R.string.titletest);  
            //menu.clearHeader();  
            // 第一个参数：组ID  
            // 第二个参数：菜单项ID  
            // 第三个参数：顺序号  
            // 第四个参数：菜单项上显示的内容  
            menu.add(1,0,0,"菜单1");  
            menu.add(1,1,1,"菜单2").setCheckable(true); // 增加一个√  
        }  
    }  
}
```

选项

```
    }

    // 创建 R.id.txt2 的上下文菜单（多级）
    else if(v == (TextView) this.findViewById(R.id.txt2)) {

        // ContextMenu.addSubMenu("菜单名称") - 用来添加子菜单。子菜单
        其实就是一个特殊的菜单

        SubMenu sub1 = menu.addSubMenu("父菜单1");
        sub1.setHeaderIcon(R.drawable.folder);
        sub1.add(0, 0, 0, "菜单1");
        sub1.add(0, 1, 1, "菜单2");
        sub1.setGroupCheckable(1, true, true);
        SubMenu sub2 = menu.addSubMenu("父菜单2");
        sub2.setIcon(R.drawable.text);
        sub2.add(1, 0, 0, "菜单3");
        sub2.add(1, 1, 1, "菜单4");
        sub2.setGroupCheckable(1, true, true);
    }
}
```

main.xml

```
<?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"
    >

    <TextView
        android:id="@+id/txt1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请长按触发(txt1)"
        />

    <TextView
        android:id="@+id/txt2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请长按触发(txt2)"
        />

</LinearLayout>
```

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Test_Contextmenu!</string>
    <string name="app_name">Test_Contextmenu</string>
    <string name="titletest">这是一段测试文字</string>
</resources>
```

ContextMenu.ContextMenuInfo

译者署名: Kun

版本: Android 2.3 r1

结构

继承关系

public static interface ContextMenu.ContextMenuInfo

子类及间接子类

间接子类

AdapterView.AdapterContextMenuInfo, ExpandableListView.ExpandableListContextMenuItemInfo

类概述

获得更多关于创建上下文菜单的信息。例如: [AdapterView](#) 使用这个类可以精确选择 adapter 的位置来启动上下文菜单。

ViewManager

译者署名: Aman

译者链接: <http://blog.csdn.net/cocohufei>

版本: Android 2.3 r1

结构

继承关系

public interface ViewManager

android.view.ViewManager

子类及间接子类

间接子类

AbsListView, AbsSpinner, AbsoluteLayout, AdapterView<T extends Adapter>,
AppWidgetHostView, DatePicker, DialerFilter, ExpandableListView, FrameLayout,
Gallery, GestureOverlayView, GridView, HorizontalScrollView, ImageSwitcher,
LinearLayout, ListView, MediaController, RadioGroup, RelativeLayout, ScrollView,
SlidingDrawer, Spinner, TabHost, TabWidget, TableLayout, TableRow, TextSwitcher,
TimePicker, TwoLineList Item, ViewAnimator, ViewFlipper, ViewGroup,
ViewSwitcher, WebView, WindowManager, ZoomControls

类概述

此接口使你可以向一个 Activity 中添加和移除子视图。调用 [Context.getSystemService\(\)](#), 你可以得到该类的一个实例。(译者注: ViewManager 是个接口, 没有任何实现, 抽象类 ViewGroup 对该接口的三个方法进行了具体实现。)

公共方法

public abstract void addView (View view, ViewGroup.LayoutParams params)

(译者注: 增添一个视图对象, 并指定其布局参数

参数

view 制定添加的子视图

params 子视图的布局参数)

public abstract void removeView (View view)

(译者注: 移除指定的视图

参数

view 指定移除的子视图)

public abstract void UpdateViewLayout (View view, ViewGroup.LayoutParams params)

(译者注: 更新一个子视图

参数

view 指定更新的子视图

params 更新时所用的布局参数)

补充

文章精选

[ViewManager 的 Demo](#)

View

农民伯伯

版本: Android 2.2

java.lang.Object
 android.view.View

直接子类:

AnalogClock, ImageView, KeyboardView, ProgressBar, SurfaceView, [TextView](#), ViewGroup, ViewStub

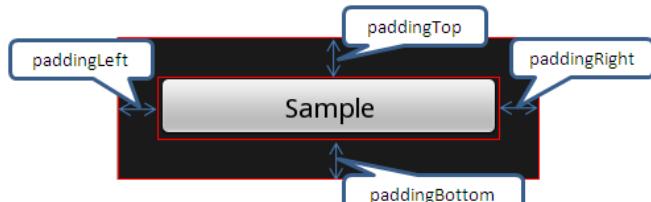
间接子类:

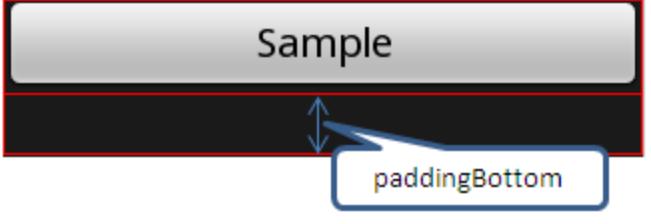
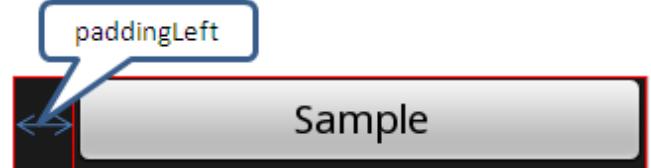
AbsListView, AbsSeekBar, AbsSpinner, AbsoluteLayout, AdapterView<T extends Adapter>, AppWidgetHostView, AutoCompleteTextView, Button, CheckBox, CheckedTextView, Chronometer, CompoundButton, DatePicker, DialerFilter, DigitalClock, [EditText](#), ExpandableListView, ExtractEditText, FrameLayout, GLSurfaceView, Gallery, GestureOverlayView, GridView, HorizontalScrollView, ImageButton, ImageSwitcher, LinearLayout, ListView, MediaController, MultiAutoCompleteTextView, QuickContactBadge, RadioButton, RadioGroup, RatingBar, RelativeLayout, ScrollView, SeekBar, SlidingDrawer, Spinner, TabHost, TabWidget, TableLayout, TableRow, TextSwitcher, TimePicker, ToggleButton, TwoLineListItem, VideoView, ViewAnimator, ViewFlipper, ViewSwitcher, WebView, ZoomButton, ZoomControls

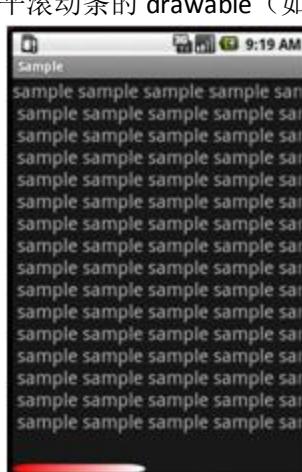
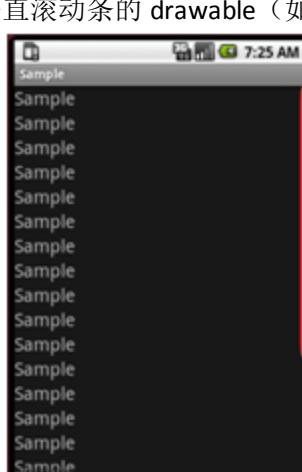
XML 属性

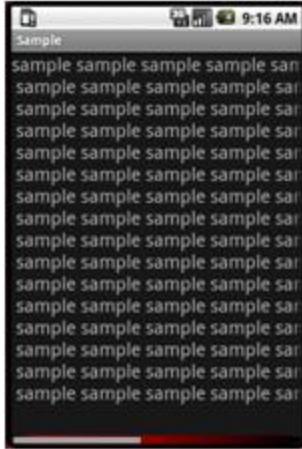
属性名称	描述
android:background	设置背景色/背景图片。可以通过以下两种方法设置背景为透明: "@android:color/transparent"和"@null"。注意 TextView 默认是透明的, 不用写此属性, 但是 Button/ImageButton/ImageView 想透明的话就得写这个属性了。
android:clickable	是否响应点击事件。
android:contentDescription	设置 View 的备注说明, 作为一种辅助功能提供, 为一些没有文字描述的 View 提供说明, 如 ImageButton。这里在界面上不会有效果, 自己在程序中控制, 可临时放一点字符串数据。
android:drawingCacheQuality	设置绘图时半透明质量。有以下值可设置: auto (默认, 由框架决定) /high (高质量, 使用较高的颜色深度, 消耗更多的内存) /low (低质量, 使用较低的颜色深度, 但是用更少的内存)。

	<p>drawingCacheQuality = auto</p> <p>drawingCacheQuality = high</p> <p>drawingCacheQuality = low</p>
android:duplicateParentState	如果设置此属性，将直接从父容器中获取绘图状态（光标，按下等）。见下面代码部分，注意根据目前测试情况仅仅是获取绘图状态，而没有获取事件，也就是你点一下 LinearLayout 时 Button 有被点击的效果，但是不执行点击事件。
android:fadingEdge	设置滚动条时，边框渐变的放向。none（边框颜色不变），horizontal（水平方向颜色变淡），vertical（垂直方向颜色变淡）。参照 fadingEdgeLength 的效果图
android:fadingEdgeLength	设置边框渐变的长度。 
android:fitsSystemWindows	设置布局调整时是否考虑系统窗口（如状态栏）
android:focusable	设置是否获得焦点。若有 requestFocus() 被调用时，后者优先处理。注意在表单中想设置某一个如 EditText 获得焦点，光设置这个是不行的，需要将这个 EditText 前面的 focusable 都设置为 false 才行。在 Touch 模式下获取焦点需要设置 focusableInTouchMode 为 true。
android:focusableInTouchMode	设置在 Touch 模式下 View 是否能取得焦点。
android:hapticFeedbackEnabled	设置触感反馈。（译者注：按软键以及进行某些 UI 交互时振动，暂时不知道用法，大家可以找找 performHapticFeedback 或 HapticFeedback 这个关键字的资料看看。）
android:id	给当前 View 设置一个在当前 layout.xml 中的唯一编号，可以通过调用 View.findViewById() 或 Activity.findViewById() 根据这个编号查找到对应的 View。不同的 layout.xml 之间

	定义相同的 id 不会冲突。格式如”@+id/btnName”
android:isScrollContainer	设置当前 View 为滚动容器。这里没有测试出效果来，ListView/ GridView/ ScrollView 根本就不用设置这个属性，而 EditText 设置 android:scrollbars 也能出滚动条。
	View 在可见的情况下是否保持唤醒状态。
android:keepScreenOn	<p>常在 LinearLayout 使用该属性，但是模拟器这里没有效果。</p> 
android:longClickable	设置是否响应长按事件.
android:minHeight	设置视图最小高度
android:minWidth	设置视图最小宽度度
android:nextFocusDown	设置下方指定视图获得下一个焦点。焦点移动是基于一个在 给定方向 查找最近邻居的算法。如果指定视图不存在，移动焦点时将报运行时错误。可以设置imeOptions= actionDone，这样输入完即跳到下一个焦点。
android:nextFocusLeft	设置左边指定视图获得下一个焦点。
android:nextFocusRight	设置右边指定视图获得下一个焦点。
android:nextFocusUp	设置上方指定视图获得下一个焦点。
android:onClick	点击时从上下文中调用指定的方法。这里指定一个方法名称，一般在 Activity 定义符合如下参数和返回值的函数并将方法名字符串指定为该值即可： <pre>public void onClickButton(View view) android:onClick="onClickButton"</pre>
android:padding	设置上下左右的边距，以像素为单位填充空白。 
android:paddingBottom	设置底部的边距，以像素为单位填充空白。

	
android:paddingLeft	设置左边的边距，以像素为单位填充空白。 
android:paddingRight	设置右边的边距，以像素为单位填充空白。 
android:paddingTop	设置上方的边距，以像素为单位填充空白。 
android:saveEnabled	设置是否在窗口冻结时（如旋转屏幕）保存 View 的数据， 默认为 true，但是前提是你需要设置 id 才能自动保存，参 见 这里 。
android:scrollX	以像素为单位设置水平方向滚动的偏移值，在 GridView 中可看的这个效果。
android:scrollY	以像素为单位设置垂直方向滚动的偏移值
android:scrollbarAlwaysDrawHorizontalTrack	设置是否始终显示垂直滚动条。这里用 ScrollView、ListView 测试均没有效果。
android:scrollbarAlwaysDrawVerticalTrack	设置是否始终显示垂直滚动条。这里用 ScrollView、ListView 测试均没有效果。
android:scrollbarDefaultDelayBeforeFade	设置 N 毫秒后开始淡化，以毫秒为单位。
android:scrollbarFadeDuration	设置滚动条淡出效果（从有到慢慢的变淡直至消失）时间， 以毫秒为单位。Android2.2 中滚动条滚动完之后会消失，再 滚动又会出来，在 1.5、1.6 版本里面会一直显示着。
android:scrollbarSize	设置滚动条的宽度。
android:scrollbarStyle	设置滚动条的风格和位置。设置值：insideOverlay、 insideInset、outsideOverlay、outsideInset。这里没有试出太 多效果，以下依次是 outsideOverlay 与 outsideInset 效果截

	<p>图比较:</p> 
android:scrollbarThumbHorizontal	设置水平滚动条的 drawable (如颜色)。 
android:scrollbarThumbVertical	设置垂直滚动条的 drawable (如颜色)。 
android:scrollbarTrackHorizontal	设置水平滚动条背景 (轨迹) 的色 drawable (如颜色)

	
android:scrollbarTrackVertical I	<p>设置垂直滚动条背景（轨迹）的 drawable 注意直接设置颜色值如” android:color/white ” 将得出很难看的效果，甚至都不理解这个属性了，这里可以参见 ApiDemos 里 res/drawable/ scrollbar_vertical_thumb.xml 和 scrollbar_vertical_track.xml ，设置代码为：</p> <pre>android:scrollbarTrackVertical ="@drawable/scrollbar_vertical_track"</pre> 
android:scrollbars	<p>设置滚动条显示。 none (隐藏), horizontal (水平), vertical (垂直)。见下列代码演示使用该属性让 EditText 内有滚动条。但是其他容器如 LinearLayout 设置了但是没有效果。</p> 

android:soundEffectsEnabled	设置点击或触摸时是否有声音效果
android:tag	设置一个文本标签。可以通过 <code>View.getTag()</code> 或 <code>for with View.findViewWithTag()</code> 检索含有该标签字符串的 <code>View</code> 。但一般最好通过 <code>ID</code> 来查询 <code>View</code> , 因为它的速度更快, 并且允许编译时类型检查。
android:visibility	设置是否显示 <code>View</code> 。设置值: <code>visible</code> (默认值, 显示), <code>invisible</code> (不显示, 但是仍然占用空间), <code>gone</code> (不显示, 不占用空间)

代码:

```

    android:duplicateParentState
        <LinearLayout android:clickable="true"
    android:background="#ff0fff" android:layout_width="100dp"
    android:layout_height="100dp">
        <Button android:duplicateParentState="true"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
    </LinearLayout>
    android:scrollbars
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:minHeight="50dp"
        android:background="@android:drawable/editbox_background"
        android:scrollbars="vertical"
        android:maxLines="4">
    </EditText>

```

ViewStub

译者署名: 唐明

版本: Android 2.2 r1

public final class ViewStub extends View

```
java.lang.Object  
    android.view.View  
        android.view.ViewStub
```

类摘要:

ViewStub 是一个隐藏的, 不占用内存空间的视图对象, 它可以在运行时延迟加载布局资源文件。当 ViewStub 可见, 或者调用 [inflate\(\)](#) 函数时, 才会加载这个布局资源文件。该 ViewStub 在加载视图时在父容器中替换它本身。因此, ViewStub 会一直存在于视图中, 直到调用 [setVisibility\(int\)](#) 或者 [inflate\(\)](#) 为止。ViewStub 的布局参数会随着加载的视图数一同被添加到 ViewStub 父容器。同样, 你也可以通过使用 inflatedId 属性来定义或重命名要加载的视图对象的 Id 值。例如:

```
<ViewStub android:id="@+id/stub"  
        android:inflatedId="@+id/subTree"  
        android:layout="@layout/mySubTree"  
        android:layout_width="120dip"  
        android:layout_height="40dip" />
```

通过"stud" id 可以找到被定义的 ViewStub 对象。加载布局资源文件"mySubTree"后, ViewStub 对象从其父容器中移除。可以通过 id"subTree"找到由布局资源"mySubTree"创建的 View。这个视图对象最后被指定为宽 120dip, 高 40dip。执行加载布局资源文件的推荐方式如下:

```
ViewStub stub = (ViewStub) findViewById(R.id.stub);  
View inflated = stub.inflate();
```

当 inflate() 被调用, 这个 ViewStub 被加载的视图替代并且返回这个视图对象。这使得应用程序不需要额外执行 findViewById() 来获取加载视图的引用。

(译者注: 这个类大概意思是用 ViewStub 类和在 XML 文件里面指定的布局资源文件关联起来, 让布局资源文件在需要使用的时候再加载上去。主要作用是性能优化, 什么时候用什么时候加载, 不用在开始启动的时候一次加载, 既可以加快程序的启动速度, 又可以节省内存资源。)

嵌套类

接口 **ViewStub.OnInflateListener** 一个用于接收 ViewStub 已经成功加载布局资源文件的通知的监听器。

XML 属性

属性名称	相关方法	描述
android:inflatedId	<code>setInflatedId(int)</code>	覆盖待加载视图的 id 值。
android:layout	<code>setLayoutResource(int)</code>	为待加载的资源视图提供一个标识，在 <code>ViewStub</code> 变为可见或获取焦点时使用它。(译者注：要引用的布局资源文件 id)

构造函数

`ViewStub(Context context, int layoutResource)`

创建一个与指定的布局资源文件关联的 `ViewStub` 对象。

参数

`layoutResource` 要加载的布局资源文件的 id 值。

公共方法

`public void draw (Canvas canvas)`

手动在指定的画布绘制这个视图(及所有其子视图)。这个视图必须在调用这个函数之前做好了整体布局。当要自己实现一个视图时，不要重载这个方法；相反，你应该重载 `onDraw(Canvas)` 方法。(译者注：主要用于自定义的视图组件的方法。)

参数

`canvas` 这个画布传到那个已渲染的视图对象。

`public int getInflatedId ()`

返回加载的布局资源文件的 ID，如果加载的布局资源文件的 id 是 `NO_ID`,那么这个加载的 `View` 将保留它原来的 id 值。

相关 XML 属性

`android:inflatedId`

返回值

一个正整数来标识这个要加载的视图或者 `NO_ID` 将保持加载视图原来的 id。

参见

`setInflatedId(int)`

`public int getLayoutResource ()`

返回加载的布局资源文件的 id 值。

相关 XML 属性

`android:layout`

返回值

加载到视图对象的布局资源文件 id 值。

参见

`setLayoutResource(int)`

`setVisibility(int)`

`inflate()`

`public View inflate ()`

加载 `getLayoutResource()` 方法标识的布局资源，并通过加载布局资源替换父容器中它自

己。

返回值

这个已加载的布局资源文件.

public void setInflatedId (int inflatedId)

设置加载视图的 ID。如果这个 id 为 NO_ID，这个加载视图保持它原来的 id 不变。

相关 XML 属性

 android:inflatedId

参数

 inflatedId 一个正整数来标识这个加载视图或者 NO_ID 将保持加载视图原来的 id。

参见

 getInflatedId()

public void setLayoutResource (int layoutResource)

设置待加载的布局资源文件，当 ViewStub 被设置为 visible 或 invisible 或调用 inflate() 时使用。这个在加载布局资源文件时创建的视图用来在父容器中替换它自己。

相关 XML 属性

 android:layout

参数

 layoutResource 一个有效的布局资源文件 id 值（不等于 0）。

参见

 getLayoutResource()

 setVisibility(int)

 inflate()

public void setOnInflateListener (ViewStub.OnInflateListener inflateListener)

设置成功加载布局资源文件后事件通知的监听器。

参数

 inflateListener 该 OnInflateListener 在成功加载后得到事件通知。

参见

 ViewStub.OnInflateListener

public void setVisibility (int visibility)

当可见性设置为 VISIBLE 或 INVISIBLE，inflate() 将被调用，并且加载视图资源在父容器中替换 ViewStub。

参数

 visibility 设置为 VISIBLE（显示），INVISIBLE（隐藏），或 GONE（完全隐藏，不暂用布局位置）。

参见

 inflate()

受保护方法

protected void dispatchDraw (Canvas canvas)

调用这个函数去绘制这个控件的子视图。可以通过派生类重写在绘制子类之前获取控制（但是是在他自己的视图已经被绘制完之后）

参数

canvas 这个画布传到那个已渲染的视图对象。

```
protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)
```

测量这个视图以确定其内容的高度和宽度。通过 `measure(int, int)` 来调用这个方法，并且应该由子类重写以提高内容测量的效率和精确度。

约定：当该方法被重写时，你必须调用 `setMeasuredDimension(int, int)` 来存储已测量视图的高度和宽度。否则将通过 `measure(int, int)` 抛出一个 `IllegalStateException` 异常。调用父类的 `onMeasure(int, int)` 方法是一个有效的办法。

父类的实现是以背景大小为默认大小，除非 `MeasureSpec`（测量细则）允许更大的背景。为了更好测量内容子类应该重写 `onMeasure(int, int)`。

如果这个方法被重写，子类有责任确保测量它的高度和宽度至少是视图的最小宽度和高度（`getSuggestedMinimumHeight()` 和 `getSuggestedMinimumWidth()`）。

参数

`widthMeasureSpec` 由于父类有横向空间要求，参见 `View.MeasureSpec`。

`heightMeasureSpec` 由于父类有纵向空间要求，参见 `View.MeasureSpec`。

ViewTreeObserver

译者署名：首当其冲
版本：Android 3.0 r1

结构

继承关系

public final class ViewTreeObserver extends Object

java.lang.Object
 android.view.ViewTreeObserver

类概述

用于注册监听的视图树观察者(observer)，在视图树种全局事件改变时得到通知。这个全局事件不仅还包括整个树的布局，从绘画过程开始，触摸模式的改变等。ViewTreeObserver 不能够被应用程序实例化，因为它是由视图提供，参照 [getViewTreeObserver\(\)](#)以查看更多信息。

内部类

interface ViewTreeObserver.OnGlobalFocusChangeListener

当在一个视图树中的焦点状态发生改变时，所要调用的回调函数的接口类

interface ViewTreeObserver.OnGlobalLayoutListener

当在一个视图树中全局布局发生改变或者视图树中的某个视图的可视状态发生改变时，所要调用的回调函数的接口类

interface ViewTreeObserver.OnPreDrawListener

当一个视图树将要绘制时，所要调用的回调函数的接口类

interface ViewTreeObserver.OnScrollChangedListener

当一个视图树中的一些组件发生滚动时，所要调用的回调函数的接口类

interface ViewTreeObserver.OnTouchModeChangeListener

当一个视图树的触摸模式发生改变时，所要调用的回调函数的接口类

公共方法

**public void addOnGlobalFocusChangeListener
(ViewTreeObserver.OnGlobalFocusChangeListener listener)**

注册一个回调函数，当在一个视图树中的焦点状态发生改变时调用这个回调函数。

参数

listener 将要被添加的回调函数

异常

IllegalStateException 如果 isAlive() 返回 false

```
public void addOnGlobalLayoutListener (ViewTreeObserver.OnGlobalLayoutListener  
listener)
```

注册一个回调函数，当在一个视图树中全局布局发生改变或者视图树中的某个视图的可视状态发生改变时调用这个回调函数。

参数

listener 将要被添加的回调函数

异常

 IllegalStateException 如果 isAlive() 返回 false

```
public void addOnPreDrawListener (ViewTreeObserver.OnPreDrawListener listener)
```

注册一个回调函数，当一个视图树将要绘制时调用这个回调函数。

参数

listener 将要被添加的回调函数

异常

 IllegalStateException 如果 isAlive() 返回 false

```
public void addOnScrollChangedListener (ViewTreeObserver.OnScrollChangedListener  
listener)
```

注册一个回调函数，当一个视图发生滚动时调用这个回调函数。

参数

listener 将要被添加的回调函数

异常

 IllegalStateException 如果 isAlive() 返回 false

```
public void addOnTouchModeChangeListener (ViewTreeObserver.OnTouchModeChangeListener  
listener)
```

注册一个回调函数，当一个触摸模式发生改变时调用这个回调函数。

参数

listener 将要被添加的回调函数

异常

 IllegalStateException 如果 isAlive() 返回 false

```
public final void dispatchOnGlobalLayout ()
```

当整个布局发生改变时通知相应的注册监听器。如果你强制对视图布局或者在一个没有附加到一个窗口的视图的层次结构或者在 GONE 状态下，它可以被手动的调用

```
public final boolean dispatchOnPreDraw ()
```

当一个视图树将要绘制时通知相应的注册监听器。如果这个监听器返回 true，则这个绘制将被取消并重新计划。如果你强制对视图布局或者在一个没有附加到一个窗口的视图的层次结构或者在一个 GONE 状态下，它可以被手动的调用

返回值

当前绘制能够取消并重新计划则返回 true，否则返回 false。

```
public boolean isAlive ()
```

指示当前的 ViewTreeObserver 是否可用(alive)。当 observer 不可用时，任何方法的调用（除了这个方法）都将抛出一个异常。如果一个应用程序保持和 ViewTreeObserver 一个历时较长的引用，它应该总是需要在调用别的方法之前去检测这个方法的返回值。

返回值

但这个对象可用则返回 true，否则返回 false

```
public void removeGlobalOnLayoutListener (ViewTreeObserver.OnGlobalLayoutListener victim)
```

移除之前已经注册的全局布局回调函数。

参数

victim 将要被移除的回调函数

异常

IllegalStateException 如果 isAlive() 返回 false

```
public void removeOnGlobalFocusChangeListener (ViewTreeObserver.OnGlobalFocusChangeListener victim)
```

移除之前已经注册的焦点改变回调函数。

参数

victim 将要被移除的回调函数

异常

IllegalStateException 如果 isAlive() 返回 false

```
public void removeOnPreDrawListener (ViewTreeObserver.OnPreDrawListener victim)
```

移除之前已经注册的预绘制回调函数。

参数

victim 将要被移除的回调函数

异常

IllegalStateException 如果 isAlive() 返回 false

```
public void removeOnScrollChangedListener (ViewTreeObserver.OnScrollChangedListener victim)
```

移除之前已经注册的滚动改变回调函数。

参数

victim 将要被移除的回调函数

异常

IllegalStateException 如果 isAlive() 返回 false

```
public void removeOnTouchModeChangeListener (ViewTreeObserver.OnTouchModeChangeListener victim)
```

移除之前已经注册的触摸模式改变回调函数

参数

victim 将要被移除的回调函数

异常

IllegalStateException 如果 isAlive() 返回 false

补充

文章链接

[Android 的选择及文字颜色](#)

ViewParent

译者署名: 逝憶流緣

译者链接: <http://t.qq.com/pansonphy>

版本: Android 3.0 r1

结构

继承关系

public interface ViewParent

android.view.ViewParent

间接子类

AbsListView, AbsSpinner, AbsoluteLayout, AdapterView<T extends Adapter>, AppWidgetHostView, DatePicker, DialerFilter, ExpandableListView, FrameLayout, Gallery, GestureOverlayView, GridView, HorizontalScrollView, ImageSwitcher, LinearLayout, ListView, MediaController, RadioGroup, RelativeLayout, ScrollView, SlidingDrawer, Spinner, TabHost, TabWidget, TableLayout, TableRow, TextSwitcher, TimePicker, TwoLineListItem, ViewAnimator, ViewFlipper, ViewGroup, ViewSwitcher, WebView, ZoomControls

类概述

定义了一些作为 View 父类, 它所具有的功能 (译者注: 也可以理解为方法)。当一个 View 与父类交互时, 就可以用到这些 API 了。(译者注: Android 中子控件维系一个 ViewParent 对象, 该对象象征着整个控件树的管理者, 子控件产生影响整个控件树的事件时, 会通知到 ViewParent, ViewParent 会将其转换成一个自顶向下的事件, 分发下去。参照[这里](#))

公共方法

public abstract void bringChildToFront (View child)

把该视图置于其他所有子视图之上。(译者注: 如在 FrameLayout 中切换被叠放的视图)

参数

child 需要改变顺序的子视图

public abstract void childDrawableStateChanged (View child)

当子视图的 drawable 状态发生变化的时候, 调用该方法。

参数

child drawable 状态发生变化的子视图

public abstract void clearChildFocus (View child)

当子视图失去焦点的时候调用该方法。

参数

child 失去焦点的子视图

public abstract void createContextMenu (ContextMenu menu)

如果该指定的 ContextMenu 需要增加菜单，则会由它的父类去填充（同时会向上递归）。

参数

menu 需要填充的菜单

public abstract View focusSearch (View v, int direction)

在指定的方向找到最近的 View 来切换焦点。

参数

v 当前视图

direction FOCUS_UP, FOCUS_DOWN, FOCUS_LEFT, 与 FOCUS_RIGHT 中取值

public abstract void focusableViewAvailable (View v)

通知父类一个新的并且能够取得焦点的子视图可以使用了。用于处理从没有可取得焦点的视图到出现第一个可以取得焦点的视图的转变。

参数

v 新出现的可以取得焦点的视图

public abstract boolean getChildVisibleRect (View child, Rect r, Point offset)

public abstract ViewParent getParent ()

如果存在父视图，则返回该视图；否则返回 NULL。

返回

如果不存在父视图，则返回 null

public abstract void invalidateChild (View child, Rect r)

重绘当前 child 指定的范围。

参数

child 当前视图

r child 中无效的范围区域

public abstract ViewParent invalidateChildInParent (int[] location, Rect r)

重绘当前 child 的全部或者一部分指定范围。location 是长度为 2 的整形数组，两个数分别为 child 左边和上边的坐标。如果父视图中指定的区域 r 是无效的，则返回该 ViewParent 的父视图。如果父视图中指定区域是有效的或者父视图不存在，则返回 null。如果方法返回对象不为空，则需要把 location 数组中的数值更新为返回的 ViewParent 的左边和上边的坐标值。

参数

location 长度为 2 的整形数组，数据为需要失效的 child 左边和上边的坐标值

r child 中无效的范围区域

返回

ViewParent，如果不存在父视图，则返回 null

public abstract boolean isLayoutRequested ()

返回该视图父类是否有 layout 控件被请求。

返回

true 需要, false 不需要

public abstract void recomputeViewAttributes (View child)

通知父类和子类所有的 View 属性需要重新生成。

参数

child 改变属性的 View

public abstract void requestChildFocus (View child , View focused)

当需要转换子类焦点时调用。

参数

child 该 ViewParent 需要取得焦点的视图。该视图包含当前聚焦视图。但事实上也不一定会获得焦点
focused child 的一个有焦点的子视图

public abstract boolean requestChildRectangleOnScreen (View child, Rect rectangle, boolean immediate)

当该 child 视图需要显示在屏幕特定位置时调用。ViewGroup 如果需要重写该方法，可以遵循以下几点：

- child 必须是该 group 的直接子类
- rectangle 要是 child 中的坐标

ViewGroup 要重写该方法，要坚持几下几点：

- 如果 rectangle 规定的区域已经是可见的，那么该方法将什么都不会改变
- 只有在 rectangle 区域可见时，该视图才会有滚动条

参数

child 以起请求的直接子视图
rectangle 需要显示到屏幕上的区域范围
immediate true 禁止有滚动 false 则有

返回值

处理请求操作后是否有滚动

public abstract void requestDisallowInterceptTouchEvent (boolean disallowIntercept)

让父类不用 [onInterceptTouchEvent\(MotionEvent\)](#) 来拦截触屏事件。

该父类需要把该方法传递给它的父类。同时也要服从触屏的请求（也就是说，只有在按上 Up 或者 clear 后才能清除该标识）。

参数

disallowIntercept true 表示 child 不让父类拦截触屏事件

public abstract void requestLayout ()

当父视图的一个 child 的 Layout 控件失效时调用。该方法将会重新请求一个 Layout 控件。

public abstract void requestTransparentRegion (View child)

当一个 child 希望视图层去收集透明区域并报告给窗口排序服务时调用。例如 SurfaceView 可以用这个接口来提高接口性能。如果在当前层次没有视图，没有必要用该方法优化，否则有可能会轻微影响该层的性能。

参数

child 要求透明区域进行处理的视图

```
public abstract boolean showContextMenuForChild (View originalView)
```

显示该视图或者其祖先类的上下文菜单。

大多数情况下，子类不需要重写该方法。但是，当该子类被直接加到窗口管理器上时（例如：[addView\(View, android.view.ViewGroup.LayoutParams\)](#)），就会重写该方法，并显示上下文菜单。

参数

originalView 需要显示上下文菜单的视图

返回

显示上下文菜单时返回 true

```
public abstract ActionMode startActionModeForChild (View originalView,  
ActionMode.Callback callback) Since: API Level 11
```

为指定视图启动一个操作模式。

大多数情况下，一个子类并不需要重新此类。但是，如果子类是直接添加到窗口管理器（例如，[addView\(View, android.view.ViewGroup.LayoutParams\)](#)），那么应重写此方法并启动操作模式。（译者注：关于 ActionMode 参见[这里](#)）

参数

originalView 操作模式首页调用的源视图

callback 处理操作模式生命周期的回调函数

返回值

如果新的操作模式已经启动，返回该操作模式，否则返回 null。

补充

文章精选

[推荐][深入 Android 【六】 —— 界面构造](#)

[Android UI Event Listener](#)

WindowManager

译者署名：逝憶流緣

译者链接：<http://t.qq.com/pansonphy>

版本：Android 2.3 r1

结构

继承关系

public interface WindowManager extends android.view.ViewManager

 android.view.WindowManager

类概述

该接口用于与窗口管理器交互。通过

Context.getSystemService(Context.WINDOW_SERVICE) 可以获取到
WindowManager 的实例。（译者注：如：WindowManager wm =
(WindowManager)context.getSystemService(Context.WINDOW_SERVICE);）

参见

[getSystemService\(String\)](#)

[WINDOW_SERVICE](#)

内部类

public static class WindowManager.LayoutParams

（译者注：继承自 android.view.ViewGroup.LayoutParams）

public static class WindowManager.BadTokenException

添加 view 时，如果该 view 的 WindowManager.LayoutParams 的令牌(token)无效，则会
抛出该异常

公共方法

public abstract Display getDefaultDisplay()

获取默认的显示对象

返回值

默认的 Display 对象

public abstract void removeViewImmediate (View view)

是 [removeView\(View\)](#) 的一个特殊扩展，在方法返回前能够立即调用该视图层次的
[View.onDetachedFromWindow\(\)](#) 方法。不适用于一般的程序；如果您要正确无误的使用它，
那您就需要格外小心了。

参数

view 需要移除的视图

补充

文章链接

WindowManagerDemo

示例代码(来自文章链接的代码)

```
public class WindowManagerDemo extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        TextView textView = (TextView)  
findViewByIId(R.id.label);  
        WindowManager windowManager = (WindowManager)  
getSystemService(Context.WINDOW_SERVICE);  
  
        // print the current window's width and height on the  
title, eg: 320*480  
        setTitle(windowManager.getDefaultDisplay().getWidth() + "*"  
+  
windowManager.getDefaultDisplay().getHeight());  
        textView.setText("See the Title");  
    }  
}
```

GestureDetector

译者署名: Haiya 胡蝶

版本: Android 2.3 r1

结构

继承关系

`public class GestureDetector extends Object`

`java.lang.Object`

`android.view.GestureDetector`

类概述

通过系统提供的 [MotionEvent](#) 来监测各种手势和（触摸）事件。当一个指定的手势事件发生时，[GestureDetector.OnGestureListener](#) 回调函数将通告用户。这个类仅仅处理由触摸引发的 [MotionEvent](#)（不能处理由轨迹球引发的事件）。要使用这个类需执行以下操作：

- 为你的 [View](#) 建立一个 `GestureDetector` 实例。
- 在 `View` 的 `onTouchEvent(MotionEvent)` 方法里确保调用（`GestureDetector` 的）`onTouchEvent(MotionEvent)` 方法。当相关事件发生时，定义在回调函数里的方法将被执行。

嵌套类

`interface GestureDetector.OnDoubleTapListener`

双击和轻击([confirmed single-tap](#))事件的监听器。（译者注： [confirmed single-tap](#) 是用户快速点一下触摸屏所引发的动作。区分下面两种情况：

1. 手指按下，停留 0.2 秒（估计值）以上再抬起时，不算 [confirmed single-tap](#)
2. 快速点击屏幕两次不会引发两次 [confirmed single-tap](#) 事件，而是引发一次 [DoubleTap](#) 事件。）

`interface GestureDetector.OnGestureListener`

在有手势动作发生时，通知的监听器

`class GestureDetector.SimpleOnGestureListener`

当只需要监听部分手势时，用于扩展的便捷类

构造函数

`public GestureDetector(GestureDetector.OnGestureListener listener, Handler handler)`

已弃用，替代方法：

`GestureDetector(android.content.Context,`

`android.view.GestureDetector.OnGestureListener, android.os.Handler)`

通过提供的监听器来创建一个 `GestureDetector`，这个构造函数只能用于非 UI 线程（因为它允许指定一个 `handler`）。

参数

`listener` 用于触发所有回调函数的监听器，不能为空

`handler` 需要使用到的 `handler`

异常

[NullPointerException](#) `Listener` 或者 `handler` 为空时

```
public GestureDetector (GestureDetector.OnGestureListener listener)
```

已弃用，替代方法：

```
GestureDetector(android.content.Context,  
    android.view.GestureDetector.OnGestureListener)
```

通过提供的监听器来创建一个 GestureDetector。你只能于 UI 线程里使用这个构造函数
(这是通常的情况)

参数

listener 用于触发所有回调函数的监听器，不能为空

异常

[NullPointerException](#) 如果 Listener 为空

参见

[Handler\(\)](#)

```
public GestureDetector (Context context, GestureDetector.OnGestureListener listener)
```

通过提供的监听器来创建一个 GestureDetector。(通常情况下) 你只能于 UI 线程里使用
这个构造函数

参数

context 应用程序上下文

listener 用于触发所有回调函数的监听器，不能为空

异常

[NullPointerException](#) 如果 Listener 为空

参见

[Handler\(\)](#)

```
public GestureDetector (Context context, GestureDetector.OnGestureListener listener,  
Handler handler)
```

通过提供的监听器来创建一个 GestureDetector。(通常情况下) 你只能于 UI 线程里使用
这个构造函数

参数

context 应用程序上下文

listener 用于触发所有回调函数的监听器，不能为空

handler 需要使用到的 handler

异常

[NullPointerException](#) 如果 Listener 为空

参见

[Handler\(\)](#)

```
public GestureDetector (Context context, GestureDetector.OnGestureListener listener,  
Handler handler, boolean ignoreMultitouch)
```

通过提供的监听器来创建一个 GestureDetector。(通常情况下) 你只能于 UI 线程里使用
这个构造函数

参数

context 应用程序上下文

`listener` 用于触发所有回调函数的监听器，不能为空
`handler` 需要使用到的 `handler`
`ignoreMultitouch` 是否忽视多点触控（译者注：仅适用于 2.2 以上的 android 版本，如果没设置这个参数（即使用的是上一个构造函数），则会忽视多点触控，由于网上都没有在 `GestureDetector` 传入 `handler` 的例子，所以我也不明白这里的 `handler` 的具体用意。如果有需要深入理解的朋友，请参照 `GestureDetector` 的源码
异常
[NullPointerException](#) 如果 Listener 为空
参见
[Handler\(\)](#)

公共方法

`public boolean isLongpressEnabled ()`

返回值

如果允许长按事件，则返回 `true`，否则为 `false`

`public boolean onTouchEvent (MotionEvent ev)`

分析指定的动作事件，如何满足条件，就触发在 [GestureDetector.OnGestureListener](#) 中提供的回调函数

参数

`ev` 当前的触摸事件（译者注：如 `MotionEvent_DOWN`, `MotionEvent_UP`）

返回值

如果 [GestureDetector.OnGestureListener](#) 消耗了这个事件，则返回 `true`，否则返回 `false`

`public void setIsLongpressEnabled (boolean isLongpressEnabled)`

设置是否启用长按。如果启用长按，当用户按下并保持按下状态时，将收到一个长按事件，同时不再接收其它事件；如果禁用长按，当用户按下并保持按下状态然后再移动手指时，将会接收到 scroll 事件。长按默认为启用。

参数

`isLongpressEnabled` 是否启用接收长按事件

`public void setOnDoubleTapListener (GestureDetector.OnDoubleTapListener onDoubleTapListener)`

设置双击及其相关手势的监听器

参数

`onDoubleTapListener` 触发所有回调函数的监听器，或者设为 `null` 以停止监听双击的手势

补充

文章链接

[GestureDetector 手势识别类 - 进阶篇](#)

[android GestureDetector 使用](#)

[GestureDetector 和 SimpleOnGestureListener 的使用教程](#)

MenuInflater

译者署名: 獨鋼躊躇

译者链接: <http://www.cnblogs.com/mxgsa/>

版本: Android 2.3 r1

结构

继承关系

`public class MenuInflater extends Object`

`java.lang.Object`

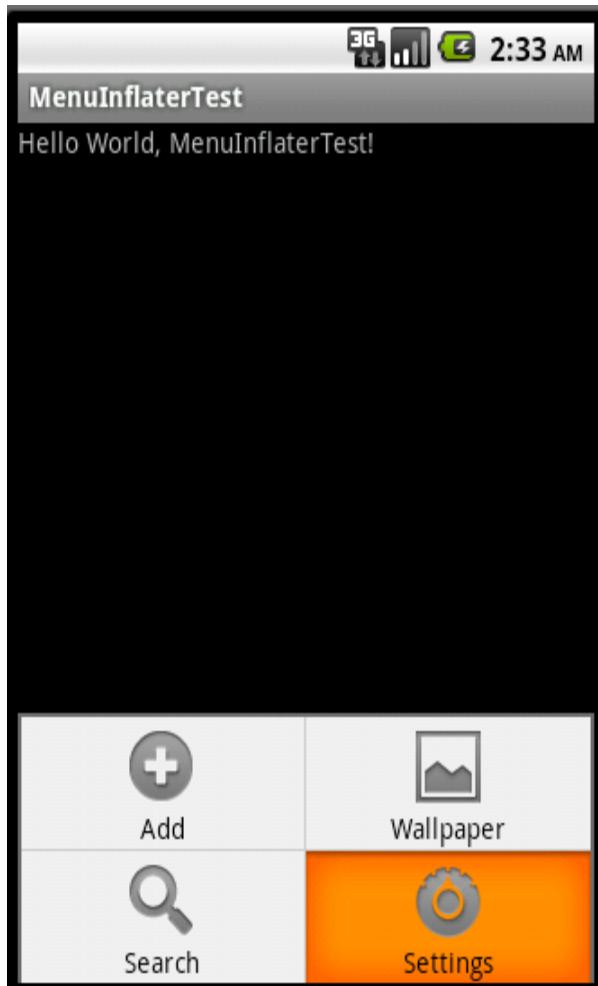
`android.view.MenuInflater`

子类及间接子类

直接子类

`TabActivity`

概述



这个类是用来实例化菜单 XML 文件成菜单对象。

由于性能的原因,由于程序创建时候就加载一些预处理 XML 文件, Menu 过多就造成很重的负担。因此,这是目前无法在运行时使用多于一个 XmlPullParser 的 xml 文件去使用 MenuInflater, 它只能使用一个 XmlPullParser 返回的编译过的资源 (R.某些文件)

构造函数

```
public MenuInflater (Context context)
```

构造填充(inflater)一个菜单

参见

[getMenuInflater\(\)](#)

公共方法

```
public void inflate (int menuRes, Menu menu)
```

菜单层次从一个指定的 xml 资源去填充，如果有错误会抛掷 [InflateException](#)。

参数

menuRes 要加载 XML 布局文件中的资源 ID (例如

R.menu.main_activity)

menu 要填充的菜单，这些项目和子菜单就被添加到要填充菜单中

补充

文章精选

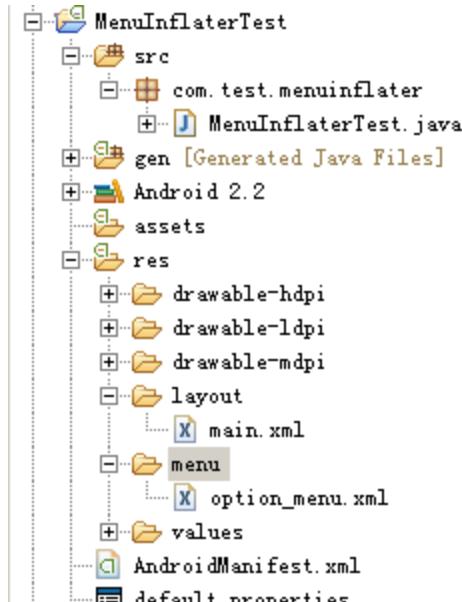
[MenuInflater Android 菜单从 xml 创建方法](#)

[Android 中 MenuInflater 实例](#)

[Android 中 MenuInflater 的使用\(布局定义菜单\)](#)

示例代码

新建一个 android2.2 的项目，项目文件列表



MenuInflaterTest.java

```
public class MenuInflaterTest extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
}
```

```
public boolean onCreateOptionsMenu(Menu menu) {
    // 获取当前的菜单
    MenuInflater inflater = getMenuInflater();
    // 填充菜单
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}

/**
 * 对菜单点击事件处理
 */
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_add:
            break;
        case R.id.menu_wallaper:
            break;
        case R.id.menu_search:
            break;
        case R.id.menu_setting:
            showSettings();
            break;
    }
    return super.onOptionsItemSelected(item);
}

/**
 * 显示设置选项
 */
private void showSettings() {
    Intent settings = new Intent
        (android.provider.Settings.ACTION_SETTINGS);
    settings.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
        | Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED);
    startActivity(settings);
}
}

Main.xml
<?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"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>

Option_menu.xml
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_add"
        android:title="Add"
        android:icon="@android:drawable/ic_menu_add"/>
    <item android:id="@+id/menu_wallpaper"
        android:title="Wallpaper"
        android:icon="@android:drawable/ic_menu_gallery"/>
    <item android:id="@+id/menu_search"
        android:title="Search"
        android:icon="@android:drawable/ic_search_category_default"
        />
    <item android:id="@+id/menu_setting"
        android:title="Settings"
        android:icon="@android:drawable/ic_menu_preferences"/>
</menu>
```

ScaleGestureDetector

译者署名：一昕

翻译时间：2010-12-05

版本:Android 2.3 r1

结构

继承关系

public class ScaleGestureDetector extends Object

java.lang.Object
 android.view.ScaleGestureDetector

类概述

根据接收的 MotionEvent，侦测由多个触点（多点触控）引发的变形手势。callback 方法 ScaleGestureDetector.OnScaleGestureListener 会在特定手势事件发生时通知用户。该类仅能和 Touch 事件引发的 MotionEvent 配合使用。使用该类需要

- 为你的 View 创建 ScaleGestureDetector 实例
- 确保在 onTouchEvent(MotionEvent)方法中调用 onScaleGestureListener.onTouchEvent (MotionEvent). [译者注：前者为该类的 onTouchEvent 方法，后者为 View 的 onTouchEvent 方法。] 在事件发生时，定义在 callback 中的方法会被调用。

(译者注：ScaleGestureDetector 为 Android2.2 新增的类，允许 Views 可以通过提供的 MotionEvents 检测和处理包括多点触摸在内的手勢变化信息。)

内部类

interface ScaleGestureDetector.OnScaleGestureListener

手势发生时接收通知的监听器

class ScaleGestureDetector.SimpleOnScaleGestureListener

一个方便使用的类。若仅想监听一部分尺寸伸缩事件，可继承该类。

公共构造方法

public	ScaleGestureDetector	(Context	context,
	ScaleGestureDetector.OnScaleGestureListener listener)		

构造函数

公共方法

public float getCurrentSpan ()

返回手势过程中，组成该手势的两个触点的当前距离。

返回值

以像素为单位的触点距离。

public long getEventTime ()

返回事件被捕捉时的时间。

返回值

以毫秒为单位的事件时间。

public float getFocusX ()

返回当前手势焦点的 X 坐标。 如果手势正在进行中，焦点位于组成手势的两个触点之间。 如果手势正在结束，焦点为仍留在屏幕上的触点的位置。 若 `isInProgress()` 返回 `false`，该方法的返回值未定义。

返回值

返回焦点的 X 坐标值，以像素为单位。

public float getFocusY ()

返回当前手势焦点的 Y 坐标。 如果手势正在进行中，焦点位于组成手势的两个触点之间。 如果手势正在结束，焦点为仍留在屏幕上的触点的位置。 若 `isInProgress()` 返回 `false`，该方法的返回值未定义。

返回值

返回焦点的 Y 坐标值，以像素为单位。

public float getPreviousSpan ()

返回手势过程中，组成该手势的两个触点的前一次距离。

返回值

两点的前一次距离，以像素为单位。

public float getScaleFactor ()

返回从前一个伸缩事件至当前伸缩事件的伸缩比率。该值定义为 `(getCurrentSpan() / getPreviousSpan())`。

返回值

当前伸缩比率.

public long getTimeDelta ()

返回前一次接收到的伸缩事件距当前伸缩事件的时间差，以毫秒为单位。

返回值

从前一次伸缩事件起始的时间差，以毫秒为单位。

public boolean isInProgress ()

如果手势处于进行过程中，返回 `true`.

返回值

如果手势处于进行过程中，返回 `true`。否则返回 `false`。

补充

文章精选

[android touchexample](#) （中文）

[Making Sense of Multitouch](#) (android-developers.blogspot.com)

SoundEffectConstants

译者署名：凌云健笔

译者链接：<http://www.cnblogs.com/lijian2010/>

版本：Android 2.3 r1

结构

继承关系

public class SoundEffectConstants extends Object

java.lang.Object
 android.view.SoundEffectConstants

类概述

[playSoundEffect\(int\)](#) 所需的播放声效设定常量。

常量

public static final int CLICK

(译者注：单击事件) 常量，取值 0 (0x00000000)

public static final int NAVIGATION_DOWN

(译者注：View.FOCUS_DOWN) 常量，取值 4 (0x00000004)

public static final int NAVIGATION_LEFT

(译者注：View.FOCUS_LEFT) 常量，取值 1 (0x00000001)

public static final int NAVIGATION_RIGHT

(译者注：View.FOCUS_RIGHT) 常量，取值 3 (0x00000003)

public static final int NAVIGATION_UP

(译者注：View.FOCUS_UP) 常量，取值 2 (0x00000002)

公共方法

public static int getContantForFocusDirection(int direction)

针对不同声音方向，获得相应的发声常数。

参数

direction 参数取值为以下常量之一：[FOCUS_UP](#), [FOCUS_DOWN](#), [FOCUS_LEFT](#),
[FOCUS_RIGHT](#), [FOCUS_FORWARD](#), [FOCUS_BACKWARD](#).

返回值

合适的发声常数

补充

文章链接

[触感反馈和声音反馈的效果实现](#)

android.view.inputmethod

接口

[InputConnection](#)

[InputMethod](#)

InputMethod.SessionCallback

[InputMethodSession](#)

InputMethodSession.EventCallback

类

[BaseInputConnection](#)

CompletionInfo

EditorInfo

ExtractedText

ExtractedTextRequest

InputBinding

InputConnectionWrapper

InputMethodInfo

[InputMethodManager](#)

InputConnection

译者署名: 六必治

译者链接: <http://www.cnblogs.com/zcmky/>

版本: Android 2.3 r1

结构

继承关系

public interface InputConnection

android.view.inputmethod.InputConnection

子类及间接子类

间接子类

[BaseInputConnection](#), [InputConnectionWrapper](#)

类概述

`InputConnection` 接口是接收输入的应用程序与 [InputMethod](#) 间的通讯通道。它可以完成以下功能，如读取光标周围的文本，向文本框提交文本，向应用程序提交原始按键事件。

[BaseInputConnection](#) 的子类应实现这一接口。

常量

public static final int GET_EXTRACTED_TEXT_MONITOR

标志，用在 [getExtractedText\(ExtractedTextRequest, int\)](#) 中，表示提取的文本变化时你想接收到更新。

常量值: 1 (0x00000001)

public static final int GET_TEXT_WITH_STYLES

标志，用在 [getTextAfterCursor\(int, int\)](#) 和 [getTextBeforeCursor\(int, int\)](#) 中，表明返回的文本中包含样式。不设置时，你将仅接收原始文本。设置时，你将收到复合的 CharSequence，包括文本和样式段。

常量值: 1 (0x00000001)

公共方法

public abstract boolean beginBatchEdit ()

通知编辑器你将开始批量编辑操作。编辑器尽量避免向你发送状态更新，直到调用 [endBatchEdit\(\)](#) 为止。

public abstract boolean clearMetaKeyStates (int states)

在指定的输入连接中清除指定的元键(meta key)按下状态。

参数

states 清除的状态，可以是 `KeyEvent.getMetaState()` 中的一位或多位结果。

返回值

成功返回 `true`，当连接无效时返回 `false`。

```
public abstract boolean commitCompletion (CompletionInfo text)
```

提交用户的选择，选择先前向 [InputMethodSession.displayCompletions\(\)](#) 提交的选项中的一个。其结果就像用户从实际 UI 中做出一样。

参数

text 提交的结果。

返回值

成功返回 true，当连接无效时返回 false。

```
public abstract boolean commitText (CharSequence text, int newCursorPosition)
```

向文本框提交文本并设置新的光标位置。之前设置的正编辑文字将自动删除。

参数

text 提交的文本。

newCursorPosition 文本范围内新的光标位置。如果大于 0，从提交文本末尾 -1 处计起；<= 0，提交文本开始处计起。所以值为 1 时，光标将定位在你刚刚插入文本之后。注意你不能光标定位在提交文本中，因为编辑器可以修改你提供的文本，所以不必将光标定位在哪。

返回值

成功返回 true，当连接无效时返回 false。

```
public abstract boolean deleteSurroundingText (int leftLength, int rightLength)
```

删除当前光标前的 leftLength 个字符，并删除当前光标后的 rightLength 个字符，不包联想输入(composing)的文字。

参数

leftLength 删除的当前光标之前字符个数。

rightLength 删除的当前光标之后字符个数。

返回值

成功返回 true，当连接无效时返回 false。

```
public abstract boolean endBatchEdit ()
```

调用 [endBatchEdit\(\)](#) 方法通知编辑器之前开始的批量编辑已完成。

```
public abstract boolean finishComposingText ()
```

强制结束文本编辑器，无论联想输入(composing text)是否激活。文本保持不变，移除任何与此文本的编辑样式或其他状态。光标保持不变。

```
public abstract int getCursorCapsMode (int reqModes)
```

取得当前光标位置的文本的大小写状态。参见 [TextUtils.getCapsMode](#) 取得更多信息。此方法在输入连接(connection)无效（如线程冲突）或客户端等待时间过长（等待几秒返回）时可能会失败。上述情况时返回 0。

参数

reqModes 依据 [TextUtils.getCapsMode](#) 的定义取得期望的状态。通过已定义的常数，你可以轻易地传递 [TextBoxAttribute.contentType](#) 到当前。

返回值

返回当前有效的大小写状态。

public abstract ExtractedText getExtractedText (ExtractedTextRequest request, int flags)

获取当前输入连接的编辑器中的当前文本，并监视是否有变化。函数返回当前文本，当文本变化时输入连接可选择性向输入法发送更新。

此方法在输入联接无效时（如线程冲突）或客户端等待时间过长（等待几秒返回）时可能会失败。上述情况时返回 `null` 值。

参数

`request` 描述文本如何返回

`flags` 控制客户端的附加选项，0 或 [GET_EXTRACTED_TEXT_MONITOR](#)

返回值

返回一个 `ExtractedText` 对象描述文本视窗的状态，及所包含的提取文本。

public abstract CharSequence getSelectedText (int flags)

如果有的话取得所选的文本。

此方法在输入连接无效时（如线程冲突）或客户端等待时间过长（等待几秒返回）时可能会失败。上述情况时返回 `null` 值。

参数

`flags` 提供附加选项控制，控制文本如何返回。可为 0 或

[GET_TEXT_WITH_STYLES](#)

返回值

如果有的话返回当前选取文本，如果没有文本被选中返回 `null`。

public abstract CharSequence getTextAfterCursor (int n, int flags)

取得当前光标位置后的 `n` 个字符文本。

此方法在输入连接无效时（如线程冲突）或客户端等待时间过长（等待几秒返回）时可能会失败。上述情况时返回 `null` 值。

参数

`n` 期望的文本长度

`flags` 提供附加选项控制，控制文本如何返回。可为 0 或

[GET_TEXT_WITH_STYLES](#)

返回值

返回当前光标后的文本，返回的文本长度可能小于 `n`

public abstract CharSequence getTextBeforeCursor (int n, int flags)

取得当前光标位置前的 `n` 个字符文本。

本此方法在输入连接无效（如线程冲突）或客户端等待时间过长（等待几秒返回）时可能会失败。上述情况时返回 `null` 值。

参数

`n` 期望的文本长度

`flags` 提供附加选项控制，控制文本如何返回。可为 0 或

[GET_TEXT_WITH_STYLES](#)

返回值

返回当前光标前的文本，返回的文本长度可能小于 `n`

```
public abstract boolean performContextMenuAction (int id)
```

在区域中执行调用上下文菜单动作，其 id 可能下列之一：[selectAll](#), [startSelectingText](#), [stopSelectingText](#), [cut](#), [copy](#), [paste](#), [copyUrl](#), 或 [switchInputMethod](#)。

```
public abstract boolean performEditorAction (int editorAction)
```

让编辑器执行一个它可以完成的操作。

参数

editorAction 必须是动作常量 `EditorInfo.editorType` 中的一个，如 `EditorInfo.EDITOR_ACTION_GO`。

返回值

成功返回 `true`, 如输入连接无效返回 `false`。

```
public abstract boolean performPrivateCommand (String action, Bundle data)
```

API 从输入法向所连接的编辑器发送私有命令。这可用于提供仅用于特定输入法及其客户端功能的特定域（domain-specific）。注意，因为 `InputConnection` 协议是异步的，你无法取回结果或知道客户端是否懂得命令；你可能使用 `EditorInfo` 来确定客户端是否支持某一命令。

参数

action 要执行的命令名称。必须是作用域名，前缀你自己的包名，这样不同的开发者就不会建立让人冲突的命令。

data 命令中的数据

返回值

当命令发送后返回 `true`（无论相关的编辑是否理解它），如输入连接无效返回 `false`。

```
public abstract boolean reportFullscreenMode (boolean enabled)
```

由 IME 调用，通知客户端将在全屏与普通模式间切换。它在 `InputMethodService` 的标准实现中被调用。

```
public abstract boolean sendKeyEvent (KeyEvent event)
```

向当前输入连接所附着的进程发送按键事件。事件像普通按键事件一样由当前焦点，通常是提供 `InputConnection` 的视图。但由于协议的异步性这一点并不总是这样，焦点可能在事件收到时发生改变。

本方法可用于向应用程序发送按键事件。如屏幕键盘可以用这一方法模拟硬件键盘。标准键盘有三种：数字（12 键），预测键盘（20 键）和字母（QWERTY）。你可以通过事件的设备码（device id）确定键盘类型。

在你向本 API 发送的所有按键事件中，你可能希望设置 [KeyEvent.FLAG_SOFT_KEYBOARD](#) 标志，但这一标志不可设置。

参数

event 按键事件

返回值

成功返回 `true`，当输入连接无效返回 `false`。

参见

[KeyEvent](#)
[NUMERIC](#)
[PREDICTIVE](#)
[ALPHA](#)

public abstract boolean setComposingRegion (int start, int end)

将特定区域设为正在编辑文本。以前设置的正在编辑文本自动移除。文本使用默认正在编辑文本样式。(审核注：“composing text”的翻译还需要后期用例子来推测其含义。)

参数

start 正在编辑文本开始的位置

end 正在编辑文本结束的位置。

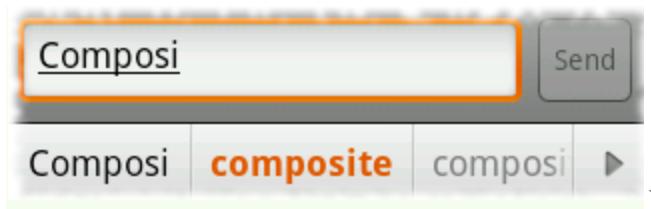
返回值

成功返回 true，当输入联接无效时返回 false。

public abstract boolean setComposingText (CharSequence text, int newCursorPosition)

将当前的光标旁正在联想文本(composing text)替代为给定文本，并设置新光标位置。以前设置的正在编辑文本自动移除。(译者注：

*InputConnection ic = getCurrentInputConnection();
ic.setComposingText("Composi", 1);*



参数

text 如必要正在编辑文本有样式。如文本没有附带样式对象，正在编辑文本将应用默认样式。见如何在文本上附加样式{#link android.text.Spanned}。{#link android.text.SpannableString}和{#link android.text.SpannableStringBuilder}是两种界面实现方式。

newCursorPosition 文本范围内新的光标位置。如果大于 0，从提交文本末尾 -1 处计起；<= 0，提交文本开始处计起。所以值为 1 时，光标将定位于你刚刚插入文本之后。注意你不能光标定位于提交文本中，因为编辑器可以修改你提供的文本，所以不必将光标定位在哪。

返回值

成功返回 true，当输入联接无效时返回 false。

public abstract boolean setSelection (int start, int end)

设置文本编辑器的选定文本。设置到当前光标时，开始与结束取相同值。

返回值

成功返回 true，当输入联接无效时返回 false。

补充

文章链接

[android sdk 中 softkeyboard 的自己解析\(4\)](#)

开源项目

[wifikeyboard](#)

InputMethod

译者署名: 六必治

译者链接: <http://www.cnblogs.com/zcmky/>

版本: Android 2.3 r1

结构

继承关系

public interface InputMethod extends Object

android.view.inputmethod.InputMethod

子类及间接子类

间接子类

[AbstractInputMethodService.AbstractInputMethodImpl](#),

[InputMethodService.InputMethodImpl](#)

类概述

`InputMethod` 接口代表了输入法，它可生成按键事件，生成文本，如数字，`email` 地址，`CJK` 字符，其它语言字符等等。在处理输入事件时，将文本返回至需要文本输入的应用程序。[InputMethodManager](#) 可得到更多关于架构的信息。

应用程序通常不使用这个接口本身，而是依靠 `TextView` 和 `EditText` 提供的标准交互。

输入法实现通常为 [InputMethodService](#) 及其子类的派生。在实现输入法时，包含它的服务控件必须提供 [SERVICE_META_DATA](#) 元数据字段，该元数据字段联接至一包含输入法细节的 XML 资源。所有输入法也必定要求客户端包含 [BIND_INPUT_METHOD](#) 以便与服务控件交互。如果不这样，系统将无法使用输入法，因其无法确认是否完整。

`InputMethod` 接口实际上分为两部分：接口是输入法的最高级接口，提供所有的访问，只有系统能访问（需要 `BIND_INPUT_METHOD` 权限）。另外调用方法 [createSession\(android.view.inputmethod.InputMethod.SessionCallback\)](#) 可实例化 [InputMethodSession](#) 副接口，用于与客户端通讯。

内部类

interface [InputMethod.SessionCallback](#)

常量

public static final String SERVICE_INTERFACE

接口名字，实现输入法的服务应说明它支持输入法，也就是它将用于意向过滤器 (`intent filter`)。服务还需要 [BIND_INPUT_METHOD](#) 权限，这样应用程序不会滥用它。

常量值: "android.view.InputMethod"

public static final String SERVICE_META_DATA

输入法通过此名字发布其自身信息。此元数据必须引用一个包含 [input-method](#) 标签的 XML 资源。

常量值: "android.view.im"

```
public static final int SHOW_EXPLICIT
```

用于 [showSoftInput\(int, ResultReceiver\)](#) 的标志：它表示用户显式地要求其（软键盘）显示。如果没有设置，系统决定可能是一个好主意，显示输入法在用户界面上的导航操作。

常量值: 1 (0x00000001)

```
public static final int SHOW_FORCED
```

标志用于 [showSoftInput\(int, ResultReceiver\)](#)：表明用户强制其（软键盘）显示。如设置，输入法保持可见直至用户在 UI 上取消。

常量值: 2 (0x00000002)

公共方法

```
public abstract void attachToken (IBinder token)
```

输入法创建后首先被调用，它提供一个与系统服务会话的唯一令牌。它需要通过服务识别输入法从而验证其操作。令牌不能传递给应用程序，因其取得了应用程序不应得到的特殊权限。

注意：为避免恶意客户端伤害，你应只接收第一个令牌。其后可能来自客户端。

```
public abstract void bindInput (InputBinding binding)
```

将输入法与新的应用程序环境绑定，以便稍后启动、停止输入处理。通常在应用程序第一次启用输入法时调用此方法。

参数

binding 与输入法绑定的应用程序窗口信息。

参见

[InputBinding](#)

[unbindInput\(\)](#)

```
public abstract void createSession (InputMethod.SessionCallback callback)
```

创建一个新的 InputMethodSession，它可处理客户应用程序与输入法的交互。你可以随后用 [revokeSession\(InputMethodSession\)](#) 销毁会话，这样就不会有任何客户端使用它。

参数

callback 新创建会话调用的接口。

```
public abstract void hideSoftInput (int flags, ResultReceiver resultReceiver)
```

将输入法的软键盘(soft input)部分对用户隐藏。

参数

flags 显示要求的附加信息。当前总是 0。

resultReceiver 向要求显示的客户端通知结果。其结果可能为

InputMethodManager.RESULT_UNCHANGED_SHOWN,

InputMethodManager.RESULT_UNCHANGED_HIDDEN, InputMethodManager.RESULT_SHOWN, 或 InputMethodManager.RESULT_HIDDEN

```
public abstract void restartInput (InputConnection inputConnection, EditorInfo attribute)
```

输入法需重置时调用此方法。

通常输入焦点从一个文本框移至另一个时调用此方法。

参数

`inputConnection` 可选，确定与文本框通讯的输入通讯通道；如为空，你使用通常绑定的输入通讯通道。

`attribute` 文本框（通常是 `EditText`）需要输入的属性

参见

[EditorInfo](#)

```
public abstract void revokeSession (InputMethodSession session)
```

关闭并销毁先前由 [createSession\(android.view.inputmethod.InputMethod.SessionCallback\)](#) 创建的会话。调用后，会话不再有效，对其的调用将失败。

参数

`session` 先前由 `SessionCallback.sessionCreated()` 提供的 `InputMethodSession` 将被吊销。

```
public abstract void setSessionEnabled (InputMethodSession session, boolean enabled).
```

控制某特定输入法会话是否激活。

参数

`session` 先前由 `SessionCallback.sessionCreated()` 提供的 `InputMethodSession` 将改变。

```
public abstract void showSoftInput (int flags, ResultReceiver resultReceiver)
```

将输入法的软键盘(soft input)部分对用户显示。

参数

`flags` 提供显示要求的附加信息。当前为 0 或设置 `SHOW_EXPLICIT` 位。

`resultReceiver` 向要求显示的客户端通知结果。其结果可能为

`InputMethodManager.RESULT_UNCHANGED_SHOWN,`

`InputMethodManager.RESULT_UNCHANGED_HIDDEN`,`InputMethodManager.RESULT_SHOWN`, 或 `InputMethodManager.RESULT_HIDDEN`

```
public abstract void startInput (InputConnection inputConnection, EditorInfo info)
```

应用程序开始接收文本，输入法准备好为应用程序处理接收事件并返回文本时调用本方法。

参数

`inputConnection` 可选，确定与文本框通讯的输入通讯通道；如为空，你使用通常绑定的输入通讯通道。

`info` 需要输入的文本框（通常是 `EditText`）信息。

参见

[EditorInfo](#)

```
public abstract void unbindInput ()
```

解除与应用程序的绑定，先前由 [bindInput\(InputBinding\)](#) 设定的信息对当前输入法无效时调用。

通常在应用程序变为非前台调用。

补充

文章精选

[在 Android 中创建一种新的输入法 \(Creating an Input Method\)](#)

[Android input method panel control](#)

[通过一个 SoftKeyboard 例子 \(可以看看博客里的其他几篇文章\)](#)

[Android IMF 的一处瑕疵](#)

[Android IMF 输入法总结](#)

[为 Android 平台开发一个输入法](#)

[Android IMF 学习笔记一](#)

[Android Framework 系列之 IMF \(一\)](#)

[Android Framework 系列之 IMF \(二\)](#)

[Android Framework 系列之 IMF \(三\)](#)

InputMethodSession

译者署名: 六必治

译者链接: <http://www.cnblogs.com/zcmky/>

版本: Android 2.3 r1

结构

继承关系

public interface InputMethodSession

 android.view.inputmethod. InputMethodSession

子类及间接子类

 间接子类

[AbstractInputMethodService.AbstractInputMethodSessionImpl](#), [InputMethodService.InputMethodSessionImpl](#)

类概述

`InputMethodSession` 接口提供给每个客户端的 [InputMethod](#) 可安全地暴露在应用程序中。

应用程序通常不自己使用本接口，而是依靠 [TextView](#) 和 [EditText](#) 提供的标准交互。

内部类

interface [InputMethodSession.EventCallback](#)

公共方法

public abstract void appPrivateCommand (String action, Bundle data)

执行由应用程序发往输入法的私有命令。它可用于提供专属域功能，仅专属于特定的输入法及其客户端

参数

 action 执行命名的名称。它必须是一个域名称，如前缀你自己的包名称，这样不同的开发者就不产生命令冲突。

 data 随命令包括的任何数据。

public abstract void dispatchKeyEvent (int seq, KeyEvent event,

InputMethodSession.EventCallback callback)

当按键按下时调用此方法。完成事件时，本方法的实现方法必须调用 *callback* 回调函数并返回结果。

如果输入法处理此事件返回 `true`，否则返回 `false`，调用者（如应用程序）将处理事件。

参数

 event 按键事件

返回值

 输入法是否处理本事件。

参见

[ERROR\(/#dispatchKeyUp\)](#)

[KeyEvent](#)

```
public abstract void dispatchTrackballEvent (int seq, MotionEvent event,  
InputMethodSession.EventCallback callback)
```

发生轨迹球事件时调用此方法。

如果输入法处理此事件返回 `true`, 否则返回 `false`, 调用者 (如应用程序) 将处理事件。

参数

`event` 移动事件

返回值

输入法是否处理本事件。

参见

[MotionEvent](#)

```
public abstract void displayCompletions (CompletionInfo[] completions)
```

由文本编辑器调用自动补全, 完成后通知输入法补全有效。可被用于输入法向用户显示备选文本以备插入。

参数

`completions` 补充文本数组有效, 并以最佳结果开头。如果数组为空, 存在的补充将被移除。

```
public abstract void finishInput ()
```

当应用程序要停止接收文字输入时调用此方法。

```
public abstract void toggleSoftInput (int showFlags, int hideFlags)
```

切换软键盘(soft input)窗口。应用程序能切换软键盘(soft input)窗口

参数

`showFlags` 提供附加的操作标志。可能为 0 或设置 `SHOW_IMPLICIT`, `SHOW_FORCED` 位。

`hideFlags` 提供附加的操作标志。可能为 0 或设置 `HIDE_IMPLICIT_ONLY`, `HIDE_NOT_ALWAYS` 位。

```
public abstract void updateCursor (Rect newCursor)
```

当目标输入域的光标位置在自身窗口内移动时调用此方法。本方法通常不被调用, 但输入法要求更新时可被调用。

参数

`newCursor` 当前输入区域窗口坐标系下光标显示的矩阵区域。

```
public abstract void updateExtractedText (int token, ExtractedText text)
```

当内容变化时文本编辑器调用此方法, 通知其新提取文本。本方法仅在输入法调用 [InputConnection.getExtractedText\(\)](#) 并带有通报更新选项时才被调用。

参数

`token` 输入法提供了分辨要求的令牌。

text 新的提取文本。

```
public abstract void updateSelection (int oldSelStart, int oldSelEnd, int newSelStart, int  
newSelEnd, int candidatesStart, int candidatesEnd)
```

当目标输入域的光标或选定变化时，调用此方法。

参数

oldSelStart 之前的选择开始位置相对于光标的文本偏移。
oldSelEnd 之前的选择结束位置相对于光标的文本偏移。
newSelStart 新的选择初始位置相对于光标的文本偏移。
newSelEnd 新的选择结束位置相对于光标的文本偏移。
candidatesStart 当前坐标系下文本文本开始位置的文本偏移。
candidatesEnd 当前坐标系下文本文本结束位置的文本偏移。

补充

文章链接

[android 键盘的啟動關閉、顯示、隱藏及其 bug](#)

BaseInputConnection

译者署名：六必治

译者链接：<http://www.cnblogs.com/zcmky/>

版本：Android 2.3 r1

结构

继承关系

public class BaseInputConnection extends Object implements InputConnection

java.lang.Object

 android.view.inputmethod.BaseInputConnection

类概述

`InputConnection` 接口实现的基类，注意大多数行为提供的是 `Editable` 联接。本类的实现一定要实现 [getEditable\(\)](#) 提供对它们自己的可编辑对象的访问。

公共方法

public boolean beginBatchEdit ()

缺省实现什么也不做。

public boolean clearMetaKeyStates (int states)

缺省实现用 [MetaKeyKeyListener.clearMetaKeyState\(long, int\)](#) 来清除状态。

参数

states 要清除的状态，每个 [KeyEvent.getMetaState\(\)](#) 中可能是 1 或个状态。

返回值

成功返回 `true`，输入连接无效时返回 `false`。

public boolean commitCompletion (CompletionInfo text)

缺省实现什么也不做。

参数

text 提交的完成。

返回值

成功返回 `true`，输入连接无效时返回 `false`。

public boolean commitText (CharSequence text, int newCursorPosition)

缺省实现将用给出的文本替代正在构建的文本。另外在虚拟状态时，将发送包含新文本的按键事件并清空可编辑控件的缓存。

参数

text 提交的文本。

newCursorPosition 文本范围内新光标位置。如`> 0`，从文本末尾`-1` 起计算；如`<= 0`，从文本起始处计算。所以是 `1` 时总是在刚插入文本之后。注意这意味着你不能将位置定义在文本中，因为编辑器可以修改你提交的文本，所以无必要将光标定位在文本中。

返回值

成功返回 true，输入连接无效时返回 false。

```
public boolean deleteSurroundingText (int leftLength, int rightLength)
```

缺省实现将删除当前光标位置附近的可编辑文本。

参数

leftLength 删除文本在当前光标位置前的字符数

rightLength 删除文本在当前光标位置后的字符数。

返回值

成功返回 true，输入连接无效时返回 false。.

```
public boolean endBatchEdit ()
```

缺省实现什么也不做。

```
public boolean finishComposingText ()
```

缺省实现将用给定的文本替代正在构建的文本。另外在虚拟状态时，将发送包含新文本的按键事件并清空可编辑控件的缓存。

```
public static int getComposingSpanEnd (Spannable text)
```

```
public static int getComposingSpanStart (Spannable text)
```

```
public int getCursorCapsMode (int reqModes)
```

缺省实现时，由 `TextUtils.getCapsMode` 返回当前可编辑文本的选定文本的光标大小写状态，虚拟状态时总是返回 0。

参数

reqModes 依据 [TextUtils.getCapsMode](#) 的定义取得期望的状态。通过已定义的常数，你可以轻易地传递 [TextBoxAttribute.contentType](#) 到当前。

返回值

返回当前有效的大小写状态。

```
public Editable getEditable ()
```

返回编辑操作的目标。缺省的实现是返回自己的虚拟可编辑控件，子类须重载一个可编辑控件并提供给它自己。

```
public ExtractedText getExtractedText (ExtractedTextRequest request, int flags)
```

缺省实现总是返回空。

参数

request 描述如何返回文本。

flags 额外的客户端控制选项，0 或 [GET_EXTRACTED_TEXT_MONITOR](#)。

返回值

返回 `ExtractedText` 对象描述文本视窗的状态并包含提取文本的本身。

```
public CharSequence getSelectedText (int flags)
```

缺省实现是返回当前选择文本，没有选取则为空。

参数

`flags` 额外选项控制文本如何返回，可能为 0 或 [GET_TEXT_WITH_STYLES](#)。

返回值

返回当前选择文本，没有选取则为空。

```
public CharSequence getTextAfterCursor (int length, int flags)
```

缺省的实现是返回缓存中当前光标位置后给定数量的文本。

参数

`length` 期望的文本长度。

`flags` 如何返回文本的额外控制。可能为 0 或 [GET_TEXT_WITH_STYLES](#)。

返回值

返回光标位置后的文本，返回文本的长度可能小于 `n`。

```
public CharSequence getTextBeforeCursor (int length, int flags)
```

缺省的实现是返回缓存中当前光标位置前给定数量的文本。

参数

`length` 期望的文本长度。

`flags` 如何返回文本的额外控制。可能为 0 或 [GET_TEXT_WITH_STYLES](#)。

返回值

返回光标位置前的文本，返回文本的长度可能小于 `n`。

```
public boolean performContextMenuAction (int id)
```

缺省实现是什么也不做。

```
public boolean performEditorAction (int actionCode)
```

缺省实现将其发送至返回键(enter key)。

参数

`actionCode` 必须是 [EditorInfo.editorType](#) 中一常量之一，如 [EditorInfo.EDITOR_ACTION_GO](#)。

返回值

成功返回 `true`，输入连接无效时返回 `false`。

```
public boolean performPrivateCommand (String action, Bundle data)
```

缺省实现是什么也不做。

参数

`action` 执行的命令名称。必须是作用域的名称（译者注：带包名，如“com.test.Command”），如以你自己的包名称前缀，这样不同的开发者就不会创建产生冲突的命令。

`data` 命令中的数据。

返回值

命令发送返回 `true`（无论相关的编辑器是否理解它），如输入连接无效返回 `false`。

```
public static final void removeComposingSpans (Spannable text)
```

```
public boolean reportFullscreenMode (boolean enabled)
```

以当前的全屏模式更新 InputMethodManager。

```
public boolean sendKeyEvent (KeyEvent event)
```

提供窗口附属输入连接视窗的发送按键事件的标准实现。

参数

event 按键事件。

返回值

成功返回 true，输入连接无效时返回 false。

```
public boolean setComposingRegion (int start, int end)
```

将特定区域标记为正在构建文本。任何以前设置的正在构建文本自动清除。正在构建文本应用缺省样式。

参数

start 文本中正在构建区域开始位置。

end 文本中正在构建区域结束位置。

返回值

成功返回 true，输入连接无效时返回 false。

```
public static void setComposingSpans (Spannable text)
```

```
public boolean setComposingText (CharSequence text, int newCursorPosition).
```

缺省实现是将给定文本置入可编辑文本，替代任何正在构建文本。新文本标记为正在构建文本并应用构建样式。

参数

text 正在构建文本，如必要应用样式。如文本无附属样式，将正在构建文本应用缺省样式。参见 how to attach style object to the text。{#link android.text.SpannableString} 和{#link android.text.SpannableStringBuilder}是实现的接口。

newCursorPosition 文本范围内新光标位置。如>0，从文本末尾-1起计算；如<=0，从文本起始处计算。所以是 1 时总是在刚插入文本之后。注意这意味着你不能将位置定义在文本中，因为编辑器可以修改你提交的文本，所以无必要将光标定位在文本中。

返回值

成功返回 true，输入连接无效时返回 false。

```
public boolean setSelection (int start, int end)
```

缺省实现是在当前可编辑文本中改变选取的位置。

返回值

成功返回 true，输入连接无效时返回 false。

补充

文章链接

[Android 全屏绘制](#)

[全屏手写输入的笔迹获取](#)

[手写输入法实现过程中的问题](#)

InputMethodManager

译者署名: 六必治

译者链接: <http://www.cnblogs.com/zcmky/>

版本: Android 3.0 r1

结构

继承关系

public final class InputMethodManager extends Object

java.lang.Object

 android.view.inputmethod.InputMethodManager

类概述

整个输入法框架（IMF）结构的核心 API，应用程序之间进行调度和当前输入法交互。你可以用 [Context.getSystemService\(\)](#) 取得这一接口的实例。

架构总述(Architecture Overview)

输入法框架（IMF）共有三个主要部分：

- **输入法管理器**，管理各部分的交互。它是一个客户端 API，存在于各个应用程序的 context 中，用来沟通管理所有进程间交互的全局系统服务。
- **输入法(IME)**，实现一个允许用户生成文本的独立交互模块。系统绑定一个当前的输入法。使其创建和生成，决定输入法何时隐藏或者显示它的 UI。同一时间只能有一个 IME 运行。
- **客户应用程序**，通过输入法管理器控制输入焦点和 IME 的状态。一次只能有一个客户端使用 IME。

应用程序(Applications)

大多数情况下，使用标准 `TextView` 或其子类的应用程序只要做少量工作就可以让软键盘(soft input methods)正常工作。你要注意的是：

- 正确设置你的可编辑文本视图 [inputType](#)，这样输入法有足够的上下文帮助用户向其输入文本。
- 当输入法显示时妥善处理隐藏屏幕空间。理想的情况下应用程序应处理窗口变小，如必要你可以利用系统执行平移窗口。你应在活动（activity）中设置 [windowSoftInputMode](#) 属性或调整创建窗口的相应值，以便系统决定平移或缩放（它会尝试自动调整但可能出错）。
- 你还可以通过相同的 [windowSoftInputMode](#) 控制窗口的首选软键盘状态（打开、关闭等）。

通过 API 可以与 IMF 及其 IME 交互实现更细致的控制，如显示或输入区域，用户选取某输入法等。

当你编写自己的文本编辑器的时候，你要实现 [onCreateInputConnection\(EditorInfo\)](#) 以返回你的 [InputConnection](#) 的接口实例，用来允许 IME 和你的文本编辑域来交互。

输入法(Input Methods)

一个输入法（IME）是 [Service](#)（服务）的实现，通常继承自 [InputMethodService](#)。IME 提供核心的 [InputMethod](#) 接口，尽管提供 [InputMethod](#) 通常是由 [InputMethodService](#) 来处理，而 IME 的实现只需要处理更高层的 API。

更多信息参阅 [InputMethodService](#)。

安全(Security)

输入法关系到许多安全问题，因其基本不受约束地驱动 UI 并监视用户输入。Android 输入法框架还允许调度第三方 IME，所必须小心以限制他们的选择和相互作用。

以下是 IMF 背后安全架构的要点：

- 只允许系统访问经 [BIND_INPUT_METHOD](#) 权限许可访问 IME 的 [InputMethod](#) 接口。通过绑定到要求这个权限的服务来强制实现这一点。所以系统可以保证没有不被信任的客户端在它的控制之外访问到当前的输入法。
- IMF 中可能有许多客户进程，但在同一时间只有一个激活的。未激活客户端不能与 IMF 核心交互通过下述机制实现。
- 输入法客户端只可访问 [InputMethodSession](#) 接口。每个客户端创建一个接口实例，只有与激活客户相关联的会话的调用才会被 IME 处理。这点通过普通 IME 的 [AbstractInputMethodService](#) 执行，必须由 IME 显式的处理，而 IME 正是 [InputMethodSession](#) 的自定义实现。
- 只有激活的 [InputConnection](#) 接受操作。IMF 通知每个客户进程是否激活，IMF 忽略非激活进程对当前输入联接的调用。这确保了当前 IME 只将事件和文本编辑交付用户可见焦点的 UI。
- 一个 IME 永远不能在屏幕关闭时与 [InputConnection](#) 交互。这是通过当屏幕关闭时所有客户端无效，并防止不良 IME 在用户无法关注其行为时驱动 UI。
- 客户应用程序可以要求系统选取一个新的 IME，但不能编程式选择之一。这是避免恶意程序在用户导航到其它应用程序时，选择自己的 IME 并保持运行。也就是说，IME 可以编程式的要求系统选择其它 IME，因它完全控制了用户输入。
- 用户在可切换至一个新的 IME 前必须显式的启用它，以确认系统了解它，使其处于可用状态。

常量

`public static final int HIDE_IMPLICIT_ONLY`

[hideSoftInputFromWindow\(IBinder, int\)](#) 中的标志，表示如果用户未显式地显示软键盘窗口，则隐藏窗口。

常量值: 1 (0x00000001)

`public static final int HIDE_NOT_ALWAYS`

[hideSoftInputFromWindow\(IBinder, int\)](#) 中的标志，表示软键盘窗口总是隐藏，除非开始时以 [SHOW_FORCED](#) 显示。

常量值: 2 (0x00000002)

`public static final int RESULT_HIDDEN`

[showSoftInput\(View, int, ResultReceiver\)](#) 和 [hideSoftInputFromWindow\(IBinder, int, ResultReceiver\)](#) 中 [ResultReceiver](#) 结果代码标志：软键盘窗口从显示切换到隐藏时的状态。

常量值: 3 (0x00000003)

`public static final int RESULT_SHOWN`

[showSoftInput\(View, int, ResultReceiver\)](#) 和 [hideSoftInputFromWindow\(IBinder, int, ResultReceiver\)](#) 中 [ResultReceiver](#) 结果代码标志：软键盘窗口从隐藏切换到显示时的状态。
常量值: 2 (0x00000002)

public static final int RESULT_UNCHANGED_HIDDEN
[showSoftInput\(View, int, ResultReceiver\)](#) 和 [hideSoftInputFromWindow\(IBinder, int, ResultReceiver\)](#) 中 [ResultReceiver](#) 结果代码标志：软键盘窗口不变保持隐藏时的状态。
常量值: 1 (0x00000001)

public static final int RESULT_UNCHANGED_SHOWN
[showSoftInput\(View, int, ResultReceiver\)](#) 和 [hideSoftInputFromWindow\(IBinder, int, ResultReceiver\)](#) 中 [ResultReceiver](#) 结果代码标志：软键盘窗口不变保持显示时的状态。
常量值: 0 (0x00000000)

public static final int SHOW_FORCED
[showSoftInput\(View, int\)](#) 标志，表示用户强制打开输入法（如长按菜单键），一直保持打开直至只有显式关闭。
常量值: 2 (0x00000002)

public static final int SHOW_IMPLICIT
[showSoftInput\(View, int\)](#) 标志，表示隐式显示输入窗口，非用户直接要求。窗口可能不显示。
常量值: 1 (0x00000001)

公共方法

public void displayCompletions (View view, CompletionInfo[] completions)
(译者注：输入法自动完成)

public InputMethodSubtype getCurrentInputMethodSubtype ()
(译者注：获取当前输入法类型？)

public List<InputMethodInfo> getEnabledInputMethodList ()
(译者注：获取已启用输入法列表？)

public List<InputMethodSubtype> getEnabledInputMethodSubtypeList (InputMethodInfo imi, boolean allowsImplicitlySelectedSubtypes)

public List<InputMethodInfo> getInputMethodList ()
(译者注：获取输入法列表)

public Map<InputMethodInfo, List<InputMethodSubtype>> getShortcutInputMethodsAndSubtypes ()

```
public void hideSoftInputFromInputMethod (IBinder token, int flags)
```

关闭/隐藏输入法软键盘区域，用户不再看到或与其交互。只能由当前激活输入法调用，因需令牌(token)验证。

参数

token 在输入法启动时提供令牌验证，验证后可对其进行操作。

flags 提供额外的操作标志。当前可以为 0 或 [HIDE_IMPLICIT_ONLY](#),
[HIDE_NOT_ALWAYS](#) 等位设置。

```
public boolean hideSoftInputFromWindow (IBinder windowToken, int flags)
```

[hideSoftInputFromWindow\(IBinder, int, ResultReceiver\)](#)的无返回值版：从窗口上下文中确定当前接收输入的窗口，隐藏其输入法窗口

参数

windowToken 由窗口请求 [View.getWindowToken\(\)](#) 返回得到的令牌(token)。

flags 提供额外的操作标志。当前可以为 0 或 [HIDE_IMPLICIT_ONLY](#) 位设置。

```
public boolean hideSoftInputFromWindow (IBinder windowToken, int flags, ResultReceiver  
resultReceiver)
```

从窗口上下文中确定当前接收输入的窗口，要求隐藏其软键盘窗口。它可由用户调用并得到结果而不仅仅是显式要求输入法窗口隐藏。

参数

windowToken 由窗口请求 [View.getWindowToken\(\)](#) 返回得到的令牌(token)。

flags 提供额外的操作标志。当前可以为 0 或 [HIDE_IMPLICIT_ONLY](#) 位设置。

resultReceiver 如不为空，当 IME 处理请求告诉你完成时调用。你收到的结果码可以是 [RESULT_UNCHANGED_SHOWN](#), [RESULT_UNCHANGED_HIDDEN](#),
[RESULT_SHOWN](#), 或 [RESULT_HIDDEN](#)。

```
public void hideStatusIcon (IBinder imeToken)
```

(译者注：隐藏状态栏图标？)

```
public boolean isAcceptingText ()
```

当前服务视图接受全文编辑返回真。没有输入法联接为 false，这时其只能处理原始按键事件。

```
public boolean isActive (View view)
```

视图为当前输入的激活视图时返回真。

```
public boolean isActive ()
```

输入法中的任意视图激活时返回真。

```
public boolean isFullscreenMode ()
```

判断相关输入法是否以全屏模式运行。全屏时，完全覆盖你的 UI 时，返回真，否则返回假。

public boolean isWatchingCursor (View view)

如当前输入法要看到输入编辑者的光标位置时返回真。

public void restartInput (View view)

如有输入法联接至视图，重启输入以显示新的内容。可在以下情况时调用此方法：视图的文字导致输入法外观变化或有按键输入流，如应用程序调用 `TextView.setText()` 时。

参数

`view` 文字发生变化的视图。

public void sendAppPrivateCommand (View view, String action, Bundle data)

对当前输入法调用 [InputMethodSession.appPrivateCommand\(\)](#)。

参数

`view` 可选的发送命令的视图，如你要发送命令而不考虑视图附加到输入法，此项可以为空。

`action` 执行的命令名称。必须是作用域的名称，如前缀包名称，这样不同的开发者就不会创建冲突的命令。

`data` 命令中包含的任何数据。

public boolean setCurrentInputMethodSubtype (InputMethodSubtype subtype)

(译者注：此方法为 3.0 中新增的方法)

public void setInputMethod (IBinder token, String id)

强制切换到新输入法部件。只能由持有 `token` 的应用程序(application)或服务(service) 调用当前激活输入法。

参数

`token` 在输入法启动时提供令牌验证，验证后可对其进行操作。

`id` 切换到新输入法的唯一标识。

public void setInputMethodAndSubtype (IBinder token, String id, InputMethodSubtype subtype)

强制切换到一个新的输入法和指定的类型。只能由持有 `token` 的应用程序(application)或服务(service) 调用当前激活输入法。(译者注：此方法为 3.0 中新增的方法)

参数

`token` 在输入法启动时提供令牌验证，验证后可对其进行操作。

`id` 切换到新输入法的唯一标识。

`subtype` 切换到新输入法的新类型。

public void showInputMethodAndSubtypeEnabler (String topId)

(译者注：此方法为 3.0 中新增的方法)

public void showInputMethodPicker ()

(译者注：显示输入法菜单列表)

public boolean showSoftInput (View view, int flags, ResultReceiver resultReceiver)

如需要，显式要求当前输入法的软键盘区域向用户显示。当用户与视图交互，用户表示要开始执行输入操作时，可以调用此方法。

参数

- `view` 当前焦点视图，可接受软键盘输入。
- `flags` 提供额外的操作标志。当前可以是 0 或 [SHOW_IMPLICIT](#) 位设置。
- `resultReceiver` 如不为空，当 IME 处理请求告诉你完成时调用。你收到的结果码可以是 [RESULT_UNCHANGED_SHOWN](#), [RESULT_UNCHANGED_HIDDEN](#), [RESULT_SHOWN](#), 或 [RESULT_HIDDEN](#)。

```
public boolean showSoftInput (View view, int flags)
```

[showSoftInput\(View, int, ResultReceiver\)](#) 的无返回值版：如需要，显式要求当前输入法的软键盘区域向用户显示。

参数

- `view` 当前焦点视图，可接受软键盘输入。
- `flags` 提供额外的操作标志。当前可以是 0 或 [SHOW_IMPLICIT](#) 位设置。

```
public void showSoftInputFromInputMethod (IBinder token, int flags)
```

显示输入法的软键盘区域，这样用户可以看到输入法窗口并能与其交互。只能由当前激活输入法调用，因需令牌(token)验证。

参数

- `token` 在输入法启动时提供令牌验证，验证后可对其进行操作。
- `flags` 提供额外的操作标志。可以是 0 或 [SHOW_IMPLICIT](#), [SHOW_FORCED](#) 位设置。

```
public void showStatusIcon (IBinder imeToken, String packageName, int iconId)
```

(译者注：显示状态栏图标？)

```
public boolean switchToLastInputMethod (IBinder imeToken)
```

```
public void toggleSoftInput (int showFlags, int hideFlags)
```

(译者注：切换软键盘)

```
public void toggleSoftInputFromWindow (IBinder windowToken, int showFlags, int hideFlags)
```

本方法切换输入法的窗口显示。如输入窗口已显示，它隐藏。如无输入窗口则显示。

参数

- `windowToken` 由窗口请求 [View.getWindowToken\(\)](#) 返回得到的令牌(token)。
- `showFlags` 提供额外的操作标志。当前可以为 0 或 [HIDE_IMPLICIT_ONLY](#) 位设置。
- `hideFlags` 提供额外的操作标志。可以是 0 或 [HIDE_IMPLICIT_ONLY](#), [HIDE_NOT_ALWAYS](#) 位设置。

```
public void updateCursor (View view, int left, int top, int right, int bottom)
```

返回窗口的当前光标位置。

`public void updateExtractedText (View view, int token, ExtractedText text)`

(译者注：当内容变化时文本编辑器调用此方法，通知其新提取文本。)

`public void updateSelection (View view, int selStart, int selEnd, int candidatesStart, int`

`candidatesEnd)`

返回当前选择区域。

补充

文章链接

[android 软键盘 InputMethodManager](#)

[Android 输入法框的梳理](#)

[推荐][android 输入法框架简介](#)

[开源拼音输入法社区的大融合（libpinyin）](#)

[android 中文输入法实现](#)

android.widget

接口

AbsListView.OnScrollListener
AbsListView.RecyclerListener
Adapter
[AdapterView.OnItemClickListener](#)
[AdapterView.OnItemLongClickListener](#)
[AdapterView.OnItemSelectedListener](#)
AutoCompleteTextView.Validator
[Checkable](#)
[Chronometer.OnChronometerTickListener](#)
[CompoundButton.OnCheckedChangeListener](#)
[DatePicker.OnDateChangedListener](#)
[ExpandableListAdapter](#)
[ExpandableListView.OnChildClickListener](#)
[ExpandableListView.OnGroupClickListener](#)
[ExpandableListView.OnGroupCollapseListener](#)
[ExpandableListView.OnGroupExpandListener](#)
[Filter.FilterListener](#)
[Filterable](#)
FilterQueryProvider
HeterogeneousExpandableList
[ListAdapter](#)
MediaController.MediaPlayerControl
[MultiAutoCompleteTextView.Tokenizer](#)
PopupWindow.OnDismissListener
RadioGroup.OnCheckedChangeListener
[RatingBar.OnRatingBarChangeListener](#)
SectionIndexer
[SeekBar.OnSeekBarChangeListener](#)
[SimpleAdapter.ViewBinder](#)
[SimpleCursorAdapter.CursorToStringConverter](#)
[SimpleCursorAdapter.ViewBinder](#)
[SimpleCursorTreeAdapter.ViewBinder](#)
[SlidingDrawer.OnDrawerCloseListener](#)
[SlidingDrawer.OnDrawerOpenListener](#)
[SlidingDrawer.OnDrawerScrollListener](#)
[SpinnerAdapter](#)
TabHost.OnTabChangeListener
TabHost.TabContentFactory
TextView.OnEditorActionListener

[TimePicker.OnTimeChangedListener](#)
[ViewSwitcher.ViewFactory](#)
WrapperListAdapter
[ZoomButtonsController.OnZoomListener](#)

类

AbsListView
AbsListView.LayoutParams
[AbsoluteLayout](#)
[AbsoluteLayout.LayoutParams](#)
[AbsSeekBar](#)
[AbsSpinner](#)
[AdapterView<T extends Adapter>](#)
[AdapterView.AdapterContextMenuInfo](#)
AlphabetIndexer
[AnalogClock](#)
ArrayAdapter<T>
AutoCompleteTextView
[BaseAdapter](#)
[BaseExpandableListAdapter](#)
[Button](#)
[CheckBox](#)
[CheckedTextView](#)
[Chronometer](#)
[CompoundButton](#)
[CursorAdapter](#)
[CursorTreeAdapter](#)
[DatePicker](#)
DialerFilter
[DigitalClock](#)
[EditText](#) (部分)
ExpandableListView
ExpandableListView.ExpandableListContextMenuInfo
[Filter](#)
[Filter.FilterResults](#)
FrameLayout
FrameLayout.LayoutParams
[Gallery](#)
[Gallery.LayoutParams](#)
[GridView](#)
HeaderViewListAdapter
[HorizontalScrollView](#)
[ImageButton](#)
[ImageSwitcher](#)

[ImageView](#)
[LinearLayout](#)
[LinearLayout.LayoutParams](#)
[ListView](#)
[ListView.FixedViewInfo](#)
[MediaController](#)
[MultiAutoCompleteTextView](#)
[MultiAutoCompleteTextView.CommaTokenizer](#)
[OverScroller](#)
[PopupWindow](#)
[ProgressBar](#)
[QuickContactBadge](#)
[RadioButton](#)
[RadioGroup](#)
[RadioGroup.LayoutParams](#)
[RatingBar](#)
[RelativeLayout](#)
[RelativeLayout.LayoutParams](#)
[RemoteViews](#)
[ResourceCursorAdapter](#)
[ResourceCursorTreeAdapter](#)
[Scroller](#)
[ScrollView](#)
[SeekBar](#)
[SimpleAdapter](#)
[SimpleCursorAdapter](#)
[SimpleCursorTreeAdapter](#)
[SimpleExpandableListAdapter](#)
[SlidingDrawer](#)
[Spinner](#)
[TabHost](#)
[TabHost.TabSpec](#)
[TableLayout](#)
[TableLayout.LayoutParams](#)
[TableRow](#)
[TableRow.LayoutParams](#)
[TabWidget](#)
[TextSwitcher](#)
[TextView](#) (部分)
[TextView.SavedState](#)
[TimePicker](#)
[Toast](#)
[ToggleButton](#)
[TwoLineListItem](#)

`VideoView`
[`ViewAnimator`](#)
[`ViewFlipper`](#)
[`ViewSwitcher`](#)
[`ZoomButton`](#)
[`ZoomButtonsController`](#)
[`ZoomControls`](#)

枚举

`ImageView.ScaleType`
`TextView.BufferType`

异常

`RemoteViews.ActionException`

AbsoluteLayout

译者署名: madgoat

译者链接: <http://madgoat.cn/>

版本: Android 2.2 r1

public class *AbsoluteLayout* extends *ViewGroup*

java.lang.Object

[android.view.View](#)

 android.view.ViewGroup

 android.widget.AbsoluteLayout

直接子类

[WebView](#)

此类不赞成使用。

推荐使用 [FrameLayout](#), [RelativeLayout](#) 或者定制的 layout 代替。

概述



让你指定子元素的 xy 精确坐标的布局。绝对布局缺乏灵活性，在没有绝对定位的情况下相比其他类型的布局更难维护。

公共方法

public ViewGroup.LayoutParams generateLayoutParams (AttributeSet attrs)

返回一组新的基于所支持的属性集的布局参数

参数

 attrs 构建 layout 布局参数的属性集合

返回值

 一个 ViewGroup.LayoutParams 的实例或者它的一个子类

受保护方法

protected ViewGroup.LayoutParams generateLayoutParams (ViewGroup.LayoutParams p)

返回一组合法的受支持的布局参数。当一个 ViewGroup 传递一个布局参数没有通过 checkLayoutParams(android.view.ViewGroup.LayoutParams) 检测的视图时，此方法被调用。此方法会返回一组新的适合当前 ViewGroup 的布局参数，可能从指定的一组布局参数中复制适当的属性。

参数

p 被转换成一组适合当前 ViewGroup 的布局参数

返回值

an instance of ViewGroup.LayoutParams or one of its descendants

一个 ViewGroup.LayoutParams 的实例或者其中的一个子节点

protected boolean checkLayoutParams (ViewGroup.LayoutParams p)

(译者注：检测是不是 AbsoluteLayout.LayoutParams 的实例，见源码：

```
@Override  
protected boolean checkLayoutParams(ViewGroup.LayoutParams p) {  
    return p instanceof AbsoluteLayout.LayoutParams;  
}
```

protected ViewGroup.LayoutParams generateDefaultLayoutParams ()

返回一组宽度为 WRAP_CONTENT,高度为 WRAP_CONTENT,坐标是 (0, 0) 的布局参数

返回值

一组默认的布局参数或 null 值

protected void onLayout (boolean changed, int l, int t, int r, int b)

在此视图 view 给他的每一个子元素分配大小和位置时调用。派生类可以重写此方法并且重新安排他们子类的布局。

参数

changed 这是当前视图 view 的一个新的大小或位置

- l 相对于父节点的左边位置
- t 相对于父节点的顶点位置
- r 相对于父节点的右边位置
- b 相对于父节点的底部位置

protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)

测量视图以确定其内容宽度和高度。此方法被 measure(int, int) 调用。需要被子类重写以提供对其内容准确高效的测量。

约定：当重写此方法时，你必须调用 setMeasuredDimension(int, int) 来保存当前视图 view 的宽度和高度。不成功调用此方法将会导致一个 IllegalStateException 异常，是由 measure(int, int) 抛出。所以调用父类的 onMeasure(int, int) 方法是必须的。

父类的实现是以背景大小为默认大小，除非 MeasureSpec (测量细则) 允许更大的背景。子类可以重写 onMeasure(int, int) 以对其内容提供更佳的尺寸。

如果此方法被重写，那么子类的责任是确认测量高度和测量宽度要大于视图 view 的最小宽度和最小高度 (getSuggestedMinimumHeight() and getSuggestedMinimumWidth())，使用这两个方法可以取得最小宽度和最小高度。

参数

widthMeasureSpec	强加于父节点的横向空间要求。要求是使用 View.MeasureSpec 进行编码
heightMeasureSpec	强加于父节点的纵向空间要求。要求是使用 View.MeasureSpec 进行编码。

补充

文件链接

[Android 的几种布局方式](#)
[第六讲：用户界面 View（二）](#)
[如何动态改变 AbsoluteLayout 布局中其它布局的坐标](#)

示例代码

```
<AbsoluteLayout  
    android:id="@+id/AbsoluteLayout01" android:layout_height="wrap_content"  
    android:layout_width="fill_parent" >  
    <TextView  
        android:text="TextView01" android:id="@+id/TextView01"  
        android:layout_height="wrap_content" android:layout_y="10px"  
        android:layout_width="wrap_content" android:layout_x="110px">  
    </TextView>  
</AbsoluteLayout>
```

AbsoluteLayout.LayoutParams

译者署名：绵白糖
版本：Android 2.2 r1

public static class AbsoluteLayout.LayoutParams extends ViewGroup.LayoutParams

```
java.lang.Object
    android.view.ViewGroup.LayoutParams
        android.widget.AbsoluteLayout.LayoutParams
```

类概述

每个子元素布局信息与绝对布局相关联。参见绝对布局属性中该类所支持的子视图属性列表。(译者注: `AbsoluteLayout` 的这种“绝对”定位的布局方式和我们非常熟悉的 Windows 编程中的 `Left` 和 `Top` 设置 UI 元素的位置是基本一致的。)

字段

public int x

在 View Group 内部子元素中的 X 水平位置。

public int y

在 View Group 内部子元素中的 Y 垂直位置。

构造函数

public AbsoluteLayout.LayoutParams(int width, int height, int x, int y)

创建一个新的具有指定宽度、高度和位置的布局参数。

参数:

width MATCH_PARENT, WRAP_CONTENT 或者固定大小的像素
height MATCH_PARENT, WRAP_CONTENT 或者固定大小的像素
x 子元素的 X 位置
y 子元素的 Y 位置

public AbsoluteLayout.LayoutParams(Context c, AttributeSet attrs)

创建一组新的布局参数，通过上下文提取的相关属性值设置。XML 属性映射到这个布局参数设置如下：

- layout_x: 子元素的 X 位置
- layout_y: 子元素的 Y 位置
- 所有来自 [ViewGroup.LayoutParams](#) 的 XML 属性

参数:

c 上下文环境。

attrs 从属性设置中提取布局参数值。

public AbsoluteLayout.LayoutParams(ViewGroup.LayoutParams source)

(译者注：根据 `ViewGroup.LayoutParams` 实例化布局参数，从源码可以看出：

```
public LayoutParams(ViewGroup.LayoutParams source) {  
    super(source);  
}
```

公共方法

`public String debug (String output)`

返回设置的布局参数的字符串表示形式。

参数

`output` 用于内部表示的预置字符串

返回值

返回如下格式字符串：输出 + "ViewGroup.LayoutParams={ width=WIDTH,
height=HEIGHT }"

AbsSeekBar

译者署名: madgoat

译者链接: <http://madgoat.cn/>

版本: Android 2.2 r1

public abstract class AbsSeekBar extends ProgressBar

java.lang.Object

[android.view.View](#)

 android.widget.ProgressBar

 android.widget.AbsSeekBar

概述



此类为抽象类。供拖动条 SeekBar 和评分条 RatingBar 继承。

公共方法

public int getKeyProgressIncrement ()

返回方向键改变后的进度值

默认情况下此值是根据最大值而得出的。

返回值

当用户按下方向键后减少或增加之后的进度值。这个进度值是正数。

public int getThumbOffset ()

参见

[setThumbOffset\(int\)](#)

public boolean onKeyDown (int keyCode, KeyEvent event)

默认实现 [KeyEvent.Callback.onKeyMultiple\(\)](#): 如果视图已启用并且可点击, 当 KEYCODE_DPAD_CENTER 或者 KEYCODE_ENTER 被释放时, 执行按下 (Down) 此视图的操作。

参数

keyCode KeyEvent 中用于表示按键被按下的识别码

event 按键操作中定义的 KeyEvent 对象

返回值

假如你已经处理了当前事件, 返回 true。假如你想继续让下一个事件接收者 (receiver) 处理, 则返回 false;

public boolean onTouchEvent (MotionEvent event)

实现这个方法来处理触摸屏幕引发的事件。

参数

event 动作事件

返回值

如果事件已经处理返回 True, 否则返回 false。

public void setKeyProgressIncrement (int increment)

设置使用方向键更改进度时每次的增加值（注：设置此值后，使用按键修改进度值时，每次增加或减少 increment 大小的幅度）

参数

increment The amount to increment or decrement when the user presses the arrow keys.

increment 当用户按下指示键时增加或减少的值

public synchronized void setMax (int max)

设置进度条的范围，从 0 到 max

参数

max 进度条的最大值

public void setThumb (Drawable thumb)

设置可绘制对象 thumb 为 SeekBar 中显示的进度表的结束位置的图案（注：例如下图）



如果 thumb 是一个有效的可绘制对象（例如不是 null 值），那么需要设置 thumb 的偏移量为他的一半宽度（参见 setThumbOffset(int)）

参数

thumb 可绘制对象

public void setThumbOffset (int thumbOffset)

设置 thumb 的偏移量允许 thumb 扩展超出轨道的范围

参数

thumbOffset 以像素为单位的偏移量

（注：例如下图，两图同样取 SeekBar 的最大值）

默认 Offset 为 thumb 的一半时：



setThumbOffset(0)时：



受保护方法

protected boolean verifyDrawable (Drawable who)

如果你的视图子类显示他自己的可视化对象，他将要重写此方法并且为了显示可绘制返回 true。此操作允许进行绘制时有动画效果。

确认当重写从方法时，需调用父类相应方法。（注：即记得调用 super.verifyDrawable(who)）

参数

who 需判断的可绘制对象 (Drawable)。如果是你要显示的对象，返回 True，

否则返回调用父类的结果。

返回值

如果可绘制对象（**Drawable**）已经在视图中显示，返回 **True** 否则返回 **false**。
并且此处不允许使用动画。

AbsSpinner

译者署名：思考的狼

译者链接：<http://blog.163.com/sikaodelang@126/>

版本：Android 2.2 r1

`public abstract class AbsSpinner extends AdapterView<T extends Adapter>`

`java.lang.Object`

[android.view.View](#)

`android.view.ViewGroup`

`android.widget.AdapterView<T extends android.widget.Adapter>`

`android.widget.AbsSpinner`

直接子类

`Gallery, Spinner`

概述

下拉列表的基类。Sdk 可能不再使用到该类（译者注：我认为是过时了）

XML 属性

属性名称	描述
<code>android:entries</code>	引用一个数据源填充 Spinner。对于静态内容，这是一种比编程填充 Spinner 更简单的方式。

公共方法

`public SpinnerAdapter getAdapter ()`

返回与当前部件相关联的适配器

返回值

该适配器用于提供视图内容

`public int getCount ()`

返回值

与此相关的适配器 AdapterView 所拥有的项目数量。（这个数据项，可能比显示的视图数据量还大。）

`public View getSelectedView ()`

返回值

这个视图对应当前选择的项，或者如果不选择视图则为 null

`public void onRestoreInstanceState (Parcelable state)`

允许视图重新应用以前通过 `onSaveInstanceState()`生成代表内部的状态。这个函数决不能以一个空值状态被调用。

参数

`state` 返回以前调用 `onSaveInstanceState()` 保存下来的状态。

`public Parcelable onSaveInstanceState ()`

允许视图生成一个代表内部的状态，以后可用于创建一个与之相同的新的实例。这个状态只可包含那些暂时的或不能被重建的信息。例如，你无法将你目前的位置保存在屏幕上，因为当有一个新的对象出现在这个视图层次上就会自动重新计算。

以下有些实例：当光标停留在一个文本视区上（但是通常来说不能是实文档，因为那个是已储存在内容提供者或者其它长久储存器中），当前所选的项位于一个列表视图中

返回值

返回一个 `Parcelable` 对象包含了当前视图动态状态，或者返回 `null` 如果没有保存，默认返回值为 `null`。

`public int pointToPosition (int x, int y)`

映射到列表中的一个坐标（译者注：根据这个坐标可以确定点击的是哪一个 `item`，看[这里](#)）

参数

`x` 局部坐标 X（译者注：列表范围内的坐标 X）

`y` 局部坐标 Y（译者注：列表范围内的坐标 Y）

返回值

返回这个位置包含的指定点（译者注：返回坐标(item)在列表中的顺序），如果这 2 个点不相交返回 `INVALID_POSITION`

`public void requestLayout ()`

重写以防止布局视图时出现大量布局要求。（译者注：这个方法通常在视图认为它自己不再合适它当前的边界的情况下被调用）

`public void setAdapter (SpinnerAdapter adapter)`

该适配器用于提供数据支持这个 `Spinner`。他还提供了改变基于他们选择的相对位置 的选定项

参数

`adapter` 该 `SpinnerAdapter` 用于下拉列表

`public void setSelection (int position, boolean animate)`

直接跳到数据适配器中指定项

`public void setSelection (int position)`

设置当前选中项，为了支持可访问的子类重写此方法，必须首先调用父类的方法。

参数

`position` 索引，被选中的数据项(从 0 开始)

受保护方法

`protected ViewGroup.LayoutParams generateDefaultLayoutParams ()`

返回默认设置的布局参数。这些参数是在请求传递给 `addView(View)`还没有布局的时候已经设置好了的。如果是返回 `null`，则从 `addView` 抛出一个异常

返回值

参数的默认布局或 null

AdapterView

译者署名: cnmahj

译者链接: <http://android.toolib.net/blog>

版本: Android 2.3 r1

结构

继承关系

public abstract class *AdapterView*<T extends *Adapter*> extends *ViewGroup*

java.lang.Object

 android.view.View

 android.view.ViewGroup

 android.widget.AdapterView<T extends android.widget.Adapter>

子类及间接子类

直接子类

[AbsListView](#), [AbsSpinner](#)

间接子类

[ExpandableListView](#), [Gallery](#), [GridView](#), [ListView](#), [Spinner](#)

类概述

AdapterView 是内容由 [Adapter](#) 来决定的视图类。

参见 [ListView](#)、[GridView](#)、[Spinner](#) 和 [Gallery](#) 等常见子类。

常量

public static final int INVALID_POSITION

代表无效的位置。有效值的范围是 0 到当前适配器项目数减 1 。

常量值: -1 (0xffffffff)

public static final long INVALID_ROW_ID

代表空或者无效的行 ID。

常量值: -9223372036854775808 (0x8000000000000000)

public static final int ITEM_VIEW_TYPE_HEADER_OR_FOOTER

当条项是列表头或列表尾时，调用 [getItemViewType\(int\)](#) 函数的返回值。

常量值: -2 (0xfffffff)

public static final int ITEM_VIEW_TYPE_IGNORE

当适配器禁止条项的视图再利用时，调用 [getItemViewType\(int\)](#) 函数的返回值。

常量值: -1 (0xffffffff)

公有构造函数

```
public AdapterView (Context context)
```

构造函数

```
public AdapterView (Context context, AttributeSet attrs)
```

构造函数

```
public AdapterView (Context context, AttributeSet attrs, int defStyle)
```

构造函数

公有方法

```
public void addView (View child)
```

该类不支持该方法，如果调用将抛出 [UnsupportedOperationException](#) 异常。

参数

child 忽略。

抛出

[UnsupportedOperationException](#) 调用该方法时

```
public void addView (View child, int index)
```

该类不支持该方法，如果调用将抛出 [UnsupportedOperationException](#) 异常。

参数

child 忽略。

index 忽略。

抛出

[UnsupportedOperationException](#) 调用该方法时

```
public void addView (View child, int index, ViewGroup.LayoutParams params)
```

该类不支持该方法，如果调用将抛出 [UnsupportedOperationException](#) 异常。

参数

child 忽略。

index 忽略。

params 忽略。

抛出

[UnsupportedOperationException](#) 调用该方法时

```
public void addView (View child, ViewGroup.LayoutParams params)
```

该类不支持该方法，如果调用将抛出 [UnsupportedOperationException](#) 异常。

参数

child 忽略。

params 忽略。

抛出

[UnsupportedOperationException](#) 调用该方法时

```
public boolean dispatchPopulateAccessibilityEvent (AccessibilityEvent event)
```

分发 [AccessibilityEvent](#) 事件到 [该视图](#) 的子视图中。

参数

event 事件。

返回值

如果事件分发完成，返回真。

public abstract T `getAdapter ()`

返回当前与该小部件关联的适配器。

返回值

用于提供视图内容的适配器。

public int `getCount ()`

返回值

与 AdapterView 相关联的适配器的条目数量。(该值是数据条目的数量，可能大于可见的视图的数量。)

public View `getEmptyView ()`

当前适配器无内容时，AdapterView 会显示特殊的空视图。空视图用于告诉用户，该 AdapterView 没有数据。

返回值

适配器为空时显示的视图。

public int `getFirstVisiblePosition ()`

返回显示在屏幕上的第一个元素在适配器中所处的位置。

返回值

在适配器数据集中的位置。

public Object `getItemAtPosition (int position)`

取得列表中指定位置的数据。

参数

position 要取得数据的位置。

返回值

列表中指定位置的数据。

public long `getItemIdAtPosition (int position)`

(译者注：关于此方法的分析，参见[这里](#)。)

public int `getLastVisiblePosition ()`

返回显示在屏幕上的最后一个元素在适配器中所处的位置。

返回值

在适配器数据集中的位置。

public final AdapterView.OnItemClickListener `getOnItemClickListener ()`

返回值

点击 AdapterView 中的条目时执行的回调函数；没有设置时返回空。

```
public final AdapterView.OnItemLongClickListener getOnItemLongClickListener ()
```

返回值

取得长按 AdapterView 中的条目时执行的回调函数的监听器；未设置则返回空。

```
public final AdapterView.OnItemSelectedListener getOnItemSelectedListener ()
```

```
public int getPositionForView (View view)
```

取得适配器项目对应的视图或其子视图在适配器的数据中所处的位置。

参数

view 适配器条目或其后代的视图。调用时该项目在 AdapterView 中必须可见。

返回值

视图在适配器数据集中的位置；如果视图不在数据列表中或当前不可见，则返回 [INVALID_POSITION](#)。

```
public Object getSelectedItem ()
```

返回值

当前选中条目对应的数据；无选中条目时返回空。

```
public long getSelectedItemId ()
```

返回值

当前选中条目相应的 ID；无选中条目则返回 INVALID_ROW_ID。

```
public int getSelectedItemPosition ()
```

返回当前选中项目在适配器数据中的位置。

返回值

返回从零开始的位置（索引）信息，没有选择条目时返回 INVALID_POSITION。

```
public abstract View getSelectedView ()
```

返回值

当前选中条目对应的视图；无选中条目时返回空。

```
public boolean performItemClick (View view, int position, long id)
```

如果定义了 OnItemClickListener 则调用它。

参数

view AdapterView 中被点击的视图。

position 视图在适配器中的索引。

id 点击的条目的行 ID。

返回值

如果成功调用了定义的 OnItemClickListener 则返回真；否则返回假。

```
public void removeAllViews ()
```

该类不支持该方法，如果调用将抛出 `UnsupportedOperationException` 异常。

抛出

[UnsupportedOperationException](#) 调用该方法时

public void removeView (View child)

该类不支持该方法，如果调用将抛出 `UnsupportedOperationException` 异常。

参数

child 忽略。

抛出

[UnsupportedOperationException](#) 调用该方法时

public void removeViewAt (int index)

该类不支持该方法，如果调用将抛出 `UnsupportedOperationException` 异常。

参数

index 忽略。

抛出

[UnsupportedOperationException](#) 调用该方法时

public abstract void setAdapter (T adapter)

设置用于为该小部件的视图提供用于显示的数据的适配器。

参数

adapter 用于创建视图内容的适配器。

public void setEmptyView (View emptyView)

设置适配器内容为空时显示的视图。

public void setFocusable (boolean focusable)

设置该视图是否可以获取焦点。 设为假时，可以确保在触控模式中该视图不能得到焦点。

参数

focusable 设为真时，该视图可以得到焦点。

public void setFocusableInTouchMode (boolean focusable)

设置在触控模式下该视图是否可以获取焦点。 设为真时，可以保证视图可以得到焦点。

参数

focusable 设为真时，该视图在触控模式下可以得到焦点。

public void setOnClickListener (View.OnClickListener l)

注册点击该视图时执行的回调函数。如果该视图不可点击，会将其改为可以点击的状态。

参数

| 事件发生时运行的回调函数。

public void setOnItemClickListener (AdapterView.OnItemClickListener listener)

注册单击 AdapterView 中的条目时执行的回调函数。

参数

listener 将要调用的回调。

```
public void setOnItemLongClickListener (AdapterView.OnItemLongClickListener listener)
```

注册长按 AdapterView 中的条目时执行的回调函数。

参数

listener 事件发生时运行的回调函数。

```
public void setOnItemSelectedListener (AdapterView.OnItemSelectedListener listener)
```

注册选中 AdapterView 中的条目时执行的回调函数。

参数

listener 事件发生时运行的回调函数。

```
public abstract void setSelection (int position)
```

设置当前选择条目。为了支持无障碍功能，重写该方法的子类必须首先调用父类的该方法。

参数

position 选择的数据条目的索引（从零开始）。

保护方法

```
protected boolean canAnimate ()
```

指示视图组是否能够在首次布局后为其子视图提供动画效果的显示。

返回值

如果子视图可以使用动画效果则返回真，否则返回假。

```
protected void dispatchRestoreInstanceState (SparseArray<Parcelable> container)
```

为了防止适配器生成的视图被解冻而重写。

参数

container 保存有之前存储的状态信息的 SparseArray。

```
protected void dispatchSaveInstanceState (SparseArray<Parcelable> container)
```

为了防止适配器生成的视图被冻结而重写。

参数

container 保存视图状态的 SparseArray。

```
protected void onLayout (boolean changed, int left, int top, int right, int bottom)
```

该视图设置其子视图的大小及位置时调用。派生类可以重写此方法，并为其子类布局。

参数

changed 是否为视图设置了新的大小和位置。

left 相对于父视图的左侧的位置。

top 相对于父视图的顶部的位置。

right 相对于父视图的右侧的位置。

bottom 相对于父视图的底部的位置。

补充

补充说明

本文由 [Android 中文在线文档](#) 转换而成。

文章精选

[小胖's blog](#)

[Android SDK 中文文档计划 \(14\) 使用 AdapterView 来绑定数据](#)

AdapterView.AdapterContextMenuInfo

译者署名: cnmahj

译者链接: <http://android.toolib.net/blog>

版本: Android 2.3 r1

结构

继承关系

```
public static class AdapterContextMenuInfo extends Object  
    implements ContextMenu.ContextMenuItemInfo
```

```
java.lang.Object  
    android.widget.AdapterView.AdapterContextMenuInfo
```

类概述

当显示 AdapterView 的上下文菜单时, 为 [onCreateContextMenu\(ContextMenu, View, ContextMenuItemInfo\)](#) 回调函数提供的额外的菜单信息。

字段

`public long id`

用于显示上下文菜单的子视图的行 ID。

`public int position`

用于显示上下文菜单的子视图在适配器中的位置。

`public View targetView`

用于显示上下文菜单的子视图。也是 AdapterView 的子视图之一。

公有构造函数

`public AdapterView.AdapterContextMenuInfo (View targetView, int position, long id)`

构造函数

补充

本文由 [Android 中文在线文档](#) 转换而成。

AdapterView.OnItemLongClickListener

译者署名: cnmahj

译者链接: <http://android.toolib.net/blog>

版本: Android 2.3 r1

结构

继承关系

public static interface AdapterView.OnItemLongClickListener

类概述

定义了当长按视图中的项目时调用的回调函数的接口。

公有方法

```
public abstract boolean onItemLongClick (AdapterView<?> parent, View view, int position,  
long id)
```

当按下视图中的项目并保持按下状态（长按）时执行的回调函数。 实现时如果需要访问与选中条目关联的数据，可以调用 `getItemAtPosition(position)`。

参数

parent 发生点击事件的 `AbsListView`。
view `AbsListView` 中被点击的视图。
position 视图在一览中的位置（索引）。
id 被点击条目的行 ID。

返回值

如果回调函数处理了长按事件，返回真；否则返回假。

补充

本文由 [Android 中文在线文档](#) 转换而成。

AdapterView.OnItemSelectedListener

译者署名: cnmahj

译者链接: <http://android.toolib.net/blog>

版本: Android 2.3 r1

结构

继承关系

public static interface AdapterView.OnItemSelectedListener

类概述

定义了当选中视图中的项目时调用的回调函数的接口。

公有方法

public abstract void onItemSelected (AdapterView<?> parent, View view, int position, long id)

当选中视图中的项目时执行的回调函数。实现时如果需要访问与选中条目关联的数据，可以调用 `getItemAtPosition(position)`。

参数

parent 发生选中事件的 AbsListView。

view AbsListView 中被选中的视图。

position 视图在一览中的位置（索引）。

id 被点击条目的行 ID。

public abstract void onNothingSelected (AdapterView<?> parent)

当视图中的处于选中状态的条目全部消失时执行的回调函数。启动触控功能或适配器为空都可能导致选中条目消失。

参数

parent 没有任何选中条目的 AdapterView。

补充

本文由 [Android 中文在线文档](#) 转换而成。

AdapterView.OnItemClickListener

译者署名： 麦子

译者链接： 0mellisa0@gmail.com

版本： Android 2.3 r1

结构

继承关系

public static interface AdapterView.OnItemClickListener

子类及间接子类

直接子类

CharacterPickerDialog, PreferenceScreen

类概述

这个接口定义了当 AdapterView 中一元素被点击时，一个回调函数被调用。

公共方法

public abstract void onItemClick (AdapterView<?> parent, View view, int position, long id)

当 AdapterView 中一元素被点击时，回调方法被调用。

如果需要访问与被选项相关的数据，执行程序可以调用 `getItemAtPosition(position)`。

参数

parent 发生点击动作的 AdapterView。

view 在 AdapterView 中被点击的视图(它是由 adapter 提供的一个视图)。

position 视图在 adapter 中的位置。

id 被点击元素的行 id。

AnalogClock

农民伯伯

版本: Android 2.2 r1

public class AnalogClock extends View

java.lang.Object

[android.view.View](#)

 android.widget.AnalogClock

概述



这个控件是一个带有时针和分针的模拟时钟。

受保护方法

protected void onAttachedToWindow ()

视图（AnalogClock）附在窗体时调用。在调用时，视图在窗体表面并开始绘制。注意保证这个方法在 `onDraw(Canvas)` 以前被调用，但是它可以在首次 `onDraw` 以前任何时间调用——包括 `onMeasure(int, int)` 之前或之后。

protected void onDetachedFromWindow ()

视图（DigitalClock）从窗体上分离（移除）时调用，同时窗体表面不再绘制视图。

protected void onDraw (Canvas canvas)

实现你自己的绘制。

参数

`canvas` 在画布上绘制背景

protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)

测量这个视图以确定其内容的高度和宽度。通过 `measure(int, int)` 来调用这个方法，并且应该由子类重写以提高内容测量的效率和精确度。

约定：当该方法被重写时，你必须调用 `setMeasuredDimension(int, int)` 来存储已测量视图的高度和宽度。否则将通过 `measure(int, int)` 抛出一个 `IllegalStateException` 异常。调用父类的 `onMeasure(int, int)` 方法是一个有效的办法。

父类的实现是以背景大小为默认大小，除非 `MeasureSpec`（测量细则）允许更大的背景。为了更好测量内容子类应该重写 `onMeasure(int, int)`。

如果这个方法被重写，子类有责任确保测量它的高度和宽度至少是视图的最小宽度和高度（`getSuggestedMinimumHeight()` 和 `getSuggestedMinimumWidth()`）。

参数

<code>widthMeasureSpec</code>	由于父类有横向空间要求，参见 <code>View.MeasureSpec</code> 。
<code>heightMeasureSpec</code>	由于父类有纵向空间要求，参见 <code>View.MeasureSpec</code> 。

`protected void onSizeChanged (int w, int h, int oldw, int oldh)`

布局期间当视图的大小发生改变时调用。如果只是添加到视图，调用时显示的是旧值 0。

（译者注：也就是添加到视图时，`oldw` 和 `oldh` 返回的是 0）

参数

<code>w</code>	视图当前宽度
<code>h</code>	视图当前高度
<code>oldw</code>	视图以前的宽度
<code>oldh</code>	视图以前的高度

BaseAdapter

译者署名： 德罗德

译者链接：<http://sparkrico.javaeye.com>

翻译时间： 2010-11-03

版本： Android 2.2 r1

结构

继承关系

```
public abstract class BaseAdapter  
    extends Object implements ListAdapter, SpinnerAdapter
```

```
java.lang.Object  
    android.widget.BaseAdapter
```

子类及间接子类

直接子类

ArrayAdapter<T>, CursorAdapter, SimpleAdapter

间接子类

ResourceCursorAdapter, SimpleCursorAdapter

类概述

用于 ListView(实现指定的 ListAdapter 接口)和 Spinner(实现指定的 SpinnerAdapter 接口)的共同实现一个公共基类适配器。

公共方法

```
public abstract boolean areAllItemsEnabled ()
```

在 ListAdapter 中所有的项目都是可用的？如果是，则代表所有的项目都是可选择，可用鼠标点击的。

返回值

如果所有项目是可用的返回真

```
public abstract View getDropDownView (int position, View convertView, ViewGroup parent)
```

获得一个在指定位置上显示下拉弹出数据的视图。

参数

position 想得到项目视图的索引

convertView 如果可能旧有的视图重新使用。注解：在使用之前应该检查这个视图不是空的并且类型合适。如果转换视图显示正确的数据是不可能的，这个方法能够创建一个新的视图

parent 视图最终将依附的父对象。

返回值

一个对应指定位置的数据的视图。

```
public int getItemViewType (int position)
```

获取通过 `getView` 为指定项目创建的视图的类型。

参数

`position` 在 `adapter` 数据里我们知道视图类型的项目的位置

返回值

一个整形的视图类型的描述。如果一个视图通过 `getView(int, View, ViewGroup)` 方法转换成另一个视图，则两个视图将共享同一类型。注意：整形必须在 0 和 `getViewTypeCount() - 1` 之间。`IGNORE_ITEM_VIEW_TYPE` 也可以返回。

public int getViewTypeCount ()

返回通过 `getView(int, View, ViewGroup)` 创建的视图的类型数量。每一个类型表示一组通过 `getView(int, View, ViewGroup)` 方法转换过的视图。如果 `adapter` 针对所有项目返回相同的视图类型，这个方法返回 1。

这个方法仅仅当 `adapter` 设置在 `AdapterView` 时调用。

返回值

通过这个 `adapter` 创建的视图类型的数量

public boolean hasStableIds ()

表明是否项目 ID 时对基础数据的变化保持稳定的。

返回值

如果相同的 ID 指相同的对象，返回真

public boolean isEmpty ()

`adapter` 数据项是否等于零

public boolean isEnabled (int position)

如果指定的位置不是一个隔离项目（隔离项目是一个不可选择，不可用鼠标点击的项目）则返回真。如果位置是无效的，其结果将是不确定的。在这种情况下一个

`ArrayIndexOutOfBoundsException`(越界)异常将抛出

参数

`position` 项目的索引

返回值

如果这个项目不是一个隔离项目则返回真。

public void notifyDataSetChanged ()

通知附属的视图基础数据已经改变，视图应该自动刷新。

public void notifyDataSetChanged ()

监控数据的 `observer` 不再有效

public void registerDataSetObserver ([DataSetObserver](#) observer)

注册一个用于 `adapter` 的 `observer`(观察者:监控数据发生改变时被调用)

参数

`observer` 当数据发生改变时得到通知的对象

```
public void unregisterDataSetObserver (DataSetObserver observer)
```

移除先前通过 registerDataSetObserver(DataSetObserver)方法注册过的 observer(观察者：
监控数据发生改变的类)

参数

observer 移除注册的对象

补充

文章链接

[Android BaseAdapter 例子](#)

[BaseAdapter 的 Bug](#)

[Android 中万能的 BaseAdapter\(Spinner,ListView,GridView\)的使用 !](#)

BaseExpandableListAdapter

译者署名： 天涯明月刀

译者博客：<http://sd6733531.javaeye.com/>

版本：Android 2.3 r1

结构

继承关系

```
public abstract class BaseExpandableListAdapter extends Object  
    implements ExpandableListAdapter, HeterogeneousExpandableList
```

```
java.lang.Object  
    android.widget.BaseExpandableListAdapter
```

子类及间接子类

直接子类

CursorTreeAdapter, SimpleExpandableListAdapter

间接子类

ResourceCursorTreeAdapter, SimpleCursorTreeAdapter

类概述

BaseExpandableListAdapter 是 [ExpandableListAdapter](#) 的抽象基类，从一些数据中提供数据和视图给可折叠列表视图。

所有继承本类的 Adapters 需要保证实现的 [getCombinedChildId\(long, long\)](#) 和 [getCombinedGroupId\(long\)](#) 方法能正确地从 View 组或 View 子元素的 ID 中生成唯一的 ID 号。

(译者注：组元素表示可折叠的列表项，子元素表示列表项展开后看到的多个子元素项。由于可折叠列表单纯寻找组元素和子元素的 ID 不是很方便，因此使用联合 ID 的方式来解决。于是有了 [getCombinedChildId\(\)](#) 和 [getCombinedGroupId\(\)](#) 方法。在 andorid 自带的 ApiDomos 的例子中有这个的代码：App/View/ExpandableList1)。

公共方法

```
public boolean areAllItemsEnabled ()
```

是否启用所有元素。

```
public int getChildType (int groupPosition, int childPosition)
```

获取由 [getChildView\(int, int, boolean, View, ViewGroup\)](#) 方法创建的指定子元素类型。

参数

groupPosition 子元素所在的组位置

childPosition 子元素所在的位置

返回值

0 表示任意一个子元素类型，因此此时应当只声明一种子元素类型。

```
public int getChildTypeCount()
```

获取由 [getChildView\(int, int, boolean, View, ViewGroup\)](#) 创建的所有子元素类型个数。每

种类型表示一个能被 [getChildView\(int, int, boolean, View, ViewGroup\)](#) 转换的(任意组中的)View 集合。如果适配器总是从所有的子元素中返回同一种类型，本方法将返回 1。

本方法将仅仅在 [AdapterView](#) 设置适配器时被调用。

返回值

BaseExpandableListAdapter 默认返回 1。

```
public long getCombinedChildId (long groupId, long childId)
```

若你预见以下默认实现的 IDs 可能出现冲突，请重写本方法。

实现返回一个 long 型：

- 第 0 位:不管 ID 指向的是一个子元素(未设置)还是一个组(已设置)，对于本方法 bit 值为 1。
- 第 1-31 位:小于 31 位的组 ID。
- 第 32-63 位:小于 32 位的子元素 ID。

从列表所有项(组或子项)中获得一个唯一的子 ID 号。可折叠列表要求每个元素(组或子项)在所有的子元素和组中有一个唯一的 ID。本方法负责根据所给的子 ID 号和组 ID 号返回唯一的 ID。此外，若 [hasStableIds\(\)](#) 是 true，那么必须要返回稳定的 ID。

参数

groupId 包含该子元素的组 ID

childID 子元素的 ID 号

返回

列表所有项(组或子项)中唯一的(和可能稳定)的子元素 ID 号。

```
public long getCombinedGroupId (long groupId)
```

若你预见以下默认实现产生 IDs 冲突的话，请重写本方法。

实现返回一个 long 型：

- 第 0 位:不管 ID 指向的是一个子元素(未设置)还是一个组(已设置)，对于本方法 bit 值为 1。
- 第 1-31 位:小于 31 位的组 ID。
- 第 32-63 位:小于 32 位的子元素 ID。

从列表所有项(组或子项)中获得一个唯一的子 ID 号。可折叠列表要求每个元素(组或子项)在所有的子元素和组中有一个唯一的 ID。本方法负责根据所给的子 ID 号和组 ID 号返回唯一的 ID。此外，若 [hasStableIds\(\)](#) 是 true，那么必须要返回稳定的 ID。

参数

groupId 包含该子元素的组 ID

返回

列表所有项(组或子项)中唯一的(和可能稳定)的子元素 ID 号。

```
public int getGroupType (int groupPosition)
```

获得由 [getGroupView\(int, boolean, View, ViewGroup\)](#) 方法创建的组元素类型。为设置的组元素。

参数

groupPosition 应返回类型所在组的位置

返回

0 表示任意组位置，因此此时应当只申明了一种组类型。

```
public int getGroupTypeCount ()
```

返回由 [getGroupView\(int, boolean, View, ViewGroup\)](#)方法创建的组视图类型个数。每个类型表示一个能被 [getGroupView\(int, boolean, View, ViewGroup\)](#)转换的 View 集合。如果适配器总是返回同一种组类型，则此时本方法将返回 1.

本方法将仅当适配器被 AdapterView 设置时调用。

返回

BaseExpandableListAdapter 默认返回 1.

```
public boolean isEmpty ()
```

(译者注：如果适配器没有任何数据，返回真。参见 Adapter)

```
public void notifyDataSetChanged ()
```

(译者注：当后台数据集发生改变时，调用此方法响应数据集的更改。)

参见

[notifyChanged\(\)](#)

```
public void notifyDataSetChangedInvalidated ()
```

(译者注：当后台数据集不被验证的时候，调用此方法。)

参见

[notifyInvalidated\(\)](#)

```
public void onGroupCollapsed (int groupPosition)
```

当组折叠的时候被调用。

参数

groupPosition 要折叠的组所在位置

```
public void onGroupExpanded (int groupPosition)
```

当组展开的时候被调用。

参数

groupPosition 要折叠的组所在位置

```
public void registerDataSetObserver (DataSetObserver observer)
```

(译者注：注册当用该适配器修改数据时调用的观察器。)

```
public void unregisterDataSetObserver (DataSetObserver observer)
```

(译者注：注销之前通过 `registerDataSetObserver(DataSetObserver)` 方法注册到该适配器的观察器。)

补充

文章链接

[Android BaseExpandableListAdapter 教程](#)

Button

农民伯伯

版本: Android 2.2 r1

public class Button extends TextView

```
java.lang.Object  
    android.view.View  
        android.widget.TextView  
            android.widget.Button
```

直接子类

CompoundButton

间接子类

CheckBox, RadioButton, ToggleButton

概述

代表一个按钮部件。用户通过按下按钮，或者点击按钮来执行一个动作。以下是一个按钮在 activity 中典型的应用：

```
public class MyActivity extends Activity {  
    protected void onCreate(Bundle icicle) {  
        super.onCreate(icicle);  
  
        setContentView(R.layout.content_layout_id);  
  
        final Button button = (Button) findViewById(R.id.button_id);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                // Perform action on click  
            }  
        });  
    }  
}
```

然后，你能在 xml 布局中通过 button 的 android:onClick 属性指定一个方法，以替代在 activity 中为 button 设置 OnClickListener。例如：

```
<Button  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:text="@string/self_destruct"  
    android:onClick="selfDestruct" />
```

现在，当用户点击按钮时，Android 系统调用 activity 的 selfDestruct(View)方法。为了正确执行，这个方法必须是 public 并且仅接受一个 View 类型的参数。例如：

```
public void selfDestruct(View view) {  
    // Kabloey  
}
```

方法的 View 参数是被点击部件的引用。

按钮样式

每个按钮的样式默认为系统按钮的背景,不同的设备、不同的平台版本有不同按钮风格。如你不满意默认的按钮样式,想对其定制以符合您应用程序的设计,那么你能用 [state list drawable](#) 替换按钮的背景图片。一个状态列表 drawable 是一个在 XML 中定义的 drawable 资源,根据当前按钮的状态改变其图片。一旦你在 XML 中定义了一个状态列表 drawable,你可以将它应用于你的 android:background 属性。欲了解更多信息和示例,参见 [State List Drawable](#).

实现一个按钮的例子可参见 [Form Stuff tutorial](#)

XML 属性

参见 Button、TextView、View 的 XML 属性。

CheckBox

农民伯伯
版本：Android 2.2 r1

public class CheckBox extends CompoundButton

```
java.lang.Object
    android.view.View
        android.widget.TextView
            android.widget.Button
                android.widget.CompoundButton
                    android.widget.CheckBox
```

概述



复选框是一种有双状态按钮的特殊类型，可以选中或者不选中。如下是一个在 activity 中使用复选框的例子：

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        setContentView(R.layout.content_layout_id);

        final CheckBox checkBox = (CheckBox) findViewById(R.id.checkbox_id);
        if (checkBox.isChecked()) {
            checkBox.setChecked(false);
        }
    }
}
```

CheckedTextView

译者署名: easy

译者博客: <http://blog.sina.com.cn/xjtu yi>

版本: Android 2.2 r1

public class CheckedTextView extends TextView implements Checkable

java.lang.Object

[android.view.View](#)

[android.widget.TextView](#)

android.widget.CheckedTextView

概述



类 CheckedTextView 继承 TextView 并实现 Checkable 接口。当 ListView 的 setSelectionMode 方法并设定为 CHOICE_MODE_SINGLE 或者 CHOICE_MODE_MULTIPLE，而非 CHOICE_MODE_NONE 时，使用此类是很有用的。

公共方法

public boolean dispatchPopulateAccessibilityEvent (AccessibilityEvent event)

在子视图的构建时分派一个辅助事件。(译者注: 通过源码可以看出, 视图构建时设置其选中状态。)

参数

event 事件

返回值

如果事件处理完成, 则返回 true

public boolean isChecked ()

是否选中。

public void setCheckMarkDrawable (Drawable d)

为一个给定的 Drawable 设定检查标记。当 isChecked() 为 true 时则绘制

参数

d 用于检查标记的 Drawable

```
public void setCheckMarkDrawable (int resid)
```

为一个给定的 `Drawable` 设定检查标记，使用它的资源 id 来标识。当 `isChecked()` 为 `true` 时则绘制

参数

`resid` 用于检查标记的 `Drawable`

```
public void setChecked (boolean checked)
```

改变文本视图的选中状态

参数

`checked` 选中文本返回 `true`，未选中返回 `false`

```
public void setPadding (int left, int top, int right, int bottom)
```

设置页边距。视图可能会增加一些必要的空间用于显示滚动条，并依赖滚动条的类型和可见性。因此，设定的值用于回调 `getPaddingLeft()`, `getPaddingTop()`, `getPaddingRight()` 和 `getPaddingBottom()` 时则返回不同的值

参数

`left` 左边距（使用“像素”单位）

`top` 上边距（使用“像素”单位）

`right` 右边距（使用“像素”单位）

`bottom` 下边距（使用“像素”单位）

```
public void toggle ()
```

反转当前视图的选中状态

补充

文章链接

[关于 CheckedTextView 的一些小东西](#)

[Android API 之 CheckedTextView 代码演示](#)

[CheckedTextView 显示问题](#)

示例代码

实现代码参加文章 1 和 2，或者点[这里](#)下载。

Checkable

CN 七号

版本: Android 2.2 r1

public interface Checkable

android.widget.Checkable

间接子类

[CheckBox](#), [CheckedTextView](#), CompoundButton, [RadioButton](#), [ToggleButton](#)

类概述



此接口定义了一个扩展，使得继承它的视图控件具有可选状态属性。

(译者注：凡是继承了此接口的类，便有了 android:checked 属性

```
<CheckBox  
    android:checked="true"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/sCheckBox"  
/>
```

公共方法

```
public abstract boolean isChecked()
```

查询当前视图控件的选中状态。

返回：

返回一个 `boolean` 的值来表示当前视图控件的选中状态，如果当前控件被选中，返回 `true` 值，否则返回 `false` 值。

(译者注：并不一定是控件上显示了对号或者点就是被选中，很可能某个思维不同于常人的程序员会把有对号的定义为未选中而返回一个 `false`。当然这只是可能，按照常理来说还是有对号和点的表示选中 `true`，反之亦然。)

`public abstract void setChecked (boolean checked)`

设置当前视图控件的选中状态。(译者注：

执行前	执行	执行后
任何状态	<code>setChecked(true)</code>	选中
任何状态	<code>setChecked(false)</code>	未选中

参数：

`checked` 指定控件的选中状态 `true` 表示设置为选中，`false` 为未选中。

`public abstract void toggle ()`

此方法用来切换当前视图控件的选中状态。(译者注：即将视图控件的当前状态变为其相反状态

执行前	执行	执行后
选中	<code>toggle()</code>	未选中
未选中	<code>toggle()</code>	选中

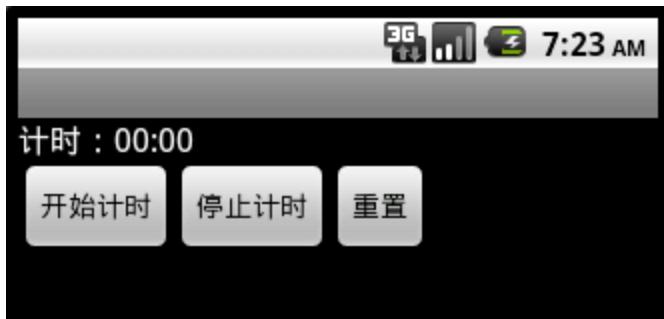
Chronometer

农民伯伯
版本: Android 2.2 r1

`public class Chronometer extends TextView`

```
java.lang.Object
    android.view.View
        android.widget.TextView
            android.widget.Chronometer
```

概述



类实现了一个简单的计时器。

你可以通过 `elapsedRealtime()` 来给它一个基准时间，并从该时间开始计数。如果你不给它基准时间，它将使用你调用 `start()` 时的时间。默认它将显示当前"MM:SS"或 "H:MM:SS"格式的时间，或者你能通过 `setFormat(String)` 设置一个任意字符串来格式化显示计时器显示的时间。

XML 属性

属性名称	描述
<code>android:format</code>	格式化字符串:如果指定, 计时器将根据这个字符串来显示, 替换字符串中第一个“%s”为当前"MM:SS"或 "H:MM:SS"格式的时间显示。如果不指定, 计时器将简单的显示"MM:SS" or "H:MM:SS"格式的时间。(译者注: 如: “This is a Chronometer %s”)

构造函数

`public Chronometer (Context context)`

初始化计时器对象。设置当前时间为基准时间。(译者注: 通过程序动态创建计时器对象)

`public Chronometer (Context context, AttributeSet attrs)`

初始化标准视图布局信息。设置当前时间为基准时间。(译者注: 指通过 XML 来指定一个计时器)

public Chronometer (Context context, AttributeSet attrs, int defStyle)

初始化标准视图布局信息和风格。设置当前时间为基准时间。

公共方法

public long getBase ()

返回先前由 `setBase(long)` 设置的基准时间。

public String getFormat ()

返回先前由 `setFormat(String)` 设置的格式化字符串。

public Chronometer.OnChronometerTickListener getOnChronometerTickListener ()

返回值

返回这个监听器（可能为空）是用于监听计时器变化的事件。

public void setBase (long base)

设置基准时间（译者注：基准时间为真正意义上开始计时的时间，而不是调用 `start` 时时间，比如调用本函数并设置参数 `base` 为 `SystemClock.elapsedRealtime()` 即表示从当前时间开始重新计时）。

参数

`base` 使用 `elapsedRealtime()` 为基准时间

public void setFormat (String format)

设置用于显示的格式化字符串。格式化字符串：如果指定，计时器将根据这个字符串来显示，替换字符串中第一个“%s”为当前“MM:SS”或“H:MM:SS”格式的时间显示。如果这个格式化字符串为空，或者你从未调用过 `setFormat()` 方法，计时器将简单的显示“MM:SS” or “H:MM:SS”格式的时间。（译者注：如：“This is a Chronometer %s”）

参数

`format` 格式化字符串

public void setOnChronometerTickListener (Chronometer.OnChronometerTickListener listener)

设置计时器变化时调用的监听事件。

参数

`listener` The listener.

public void start ()

开始计时。不会影响到由 `setBase(long)` 设置的基准时间，仅显示视图。即使部件不显示，计时器也会通过定时处理消息来工作。为了确保不发生资源泄漏，用户应确保每个 `start()` 方法都有对应的 `stop()` 调用（译者注：有一个 `start` 就有一个 `stop`）。（译者注：`start` 只是显示计时，实际上计时是从基准时间开始的，所以通过 `stop` 停止计时若干秒后再 `start` 时，显示的计时会突然跳到当前显示的计时后的若干秒后继续计时，见此[帖子](#)。）

public void stop ()

停止计时。不会影响到由 `setBase(long)` 设置的基准时间，仅显示视图。这将停止消息发送，有效地释放计时器运行时 `start()` 占用的资源。

受保护方法

`protected void onDetachedFromWindow ()`

视图从窗体上移除时调用，同时窗体表面不再显示视图。

`protected void onWindowVisibilityChanged (int visibility)`

当窗体中视图的可视性（`GONE`, `INVISIBLE`, `VISIBLE`）发生改变时调用。注意它将告诉你你的窗口是否可以被窗口管理器识别，这并不能说明窗口是否被屏幕上的其他窗口遮挡，即使它本身是可见的。

参数

`visibility` 窗口新的可见性

补充

文章链接

[android 中的时间服务 – Chronometer 计时器服务](#)

示例代码

Java 文件

```
public class ChronometerDemo extends Activity {
    private Chronometer cher1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.chronometer);
        cher1 = (Chronometer) findViewById(R.id.cher1);
        cher1.setFormat("计时: %s");
    }
    /**
     * 开始计时
     * @param view
     */
    public void onStart(View view) {
        cher1.start();
    }
    /**
     * 停止计时
     * @param view
     */
    public void onStop(View view) {
        cher1.stop();
    }
    /**
     * 重置
     */
```

```
* @param view
*/
public void onReset(View view) {
    cher1.setBase(SystemClock.elapsedRealtime());
}
}

XML 文件
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <Chronometer android:id="@+id/cher1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Chronometer>
    <LinearLayout android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button android:onClick="onStart" android:text="开始计时"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
        <Button android:onClick="onStop" android:text="停止计时"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
        <Button android:onClick="onReset" android:text="重置"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
    </LinearLayout>
</LinearLayout>
```

Chronometer.OnChronometerTickListener

农民伯伯
版本: Android 2.2 r1

public static interface Chronometer.OnChronometerTickListener

android.widget.Chronometer.OnChronometerTickListener

概述

计时器递增的时候通知这个回调。

公共方法

public abstract void onChronometerTick (Chronometer chronometer)

在计时器变化时通知。

参数

chronometer 计时器对象。

CompoundButton

翻译署名：德罗德

译者博客：sparkrico.javaeye.com

版本：Android 2.2 r1

public abstract class CompoundButton extends Button implements Checkable

```
java.lang.Object  
    android.view.View  
        android.widget.TextView  
            android.widget.Button  
                android.widget.CompoundButton
```

间接子类

CheckBox, RadioButton, ToggleButton

概述

一个带有选中/未选中状态的按钮。当按钮按下或点中时自动改变状态。

公共方法

public boolean dispatchPopulateAccessibilityEvent (AccessibilityEvent event)

在子视图的构建时分派一个辅助事件。（译者注：通过源码可以看出，视图构建时设置其选中状态。）

参数

event 事件

返回值

如果事件全部完成返回 True。

public boolean isChecked ()

（译者注：是否选中）

public void onRestoreInstanceState (Parcelable state)

允许视图重新应用以前通过 `onSaveInstanceState()`生成代表内部的状态。这个函数决不调用一个空的状态。

参数

state 返回以前调用 `onSaveInstanceState()`保存下来的状态。

public Parcelable onSaveInstanceState ()

允许视图生成一个代表内部的状态，以后可用于创建一个与之相同的新的实例。这种状态应该只包含非持久或以后不能够重建的信息。例如，你决不存储你当前在屏幕上的位置，因为这会在视图的层面上重新计算放置一个新的实例。

你可以存储到这里的一些例子：一个文本框中当前光标的位置（但通常不是文字本身，文字通常保存在内容提供者(content provider)或其他持久的储存中），一个列表视图中的当前

选中项。

返回值

返回一个包含视图当前状态的 `Parcelable` 对象，或没有什么状态保存时返回 `null`。默认实现返回 `null`。

```
public boolean performClick ()
```

如果视图定义了 `OnClickListener` 监听器，调用此方法来执行。

返回值

定义了的 `OnClickListener` 被调用返回 `True`，否则返回 `False`

```
public void setButtonDrawable (Drawable d)
```

给按钮背景设置一个可绘制对象（如：图像）

参数

`d` 用作背景的可绘制对象（如：图像）

```
public void setButtonDrawable (int resid)
```

通过资源 `Id` 给按钮背景设置一个图像

参数

`resid` 作为背景图像的资源 `id`

```
public void setChecked (boolean checked)
```

改变按钮的选中状态

参数

`checked true` 选中，`false` 非选中

```
public void setOnCheckedChangeListener (CompoundButton.OnCheckedChangeListener  
listener)
```

注册一个在按钮状态发生改变时执行的回调函数

参数

`listener` 当选中状态改变时调用的函数

```
public void toggle ()
```

改变选中状态为当前状态的逆状态

受保护方法

```
protected void drawableStateChanged ()
```

在视图状态的变化影响到所显示可绘制的状态时调用这个方法。

确保在重载时中调用父类方法

```
protected int[] onCreateDrawableState (int extraSpace)
```

为当前视图生成新的可绘图区状态。这个方式当缓存的图像绘图区状态确定失效时通过视图系统调用。你可以使用 `getDrawableState()` 方法重新取得当前的状态。

参数

`extraSpace` 如果为非零，这是你应该返回的数组在你可以存放你的状态的额

外条目的数量。

返回值

返回一个记录着视图中当前绘图区状态的数组

protected void onDraw (Canvas canvas)

实现你自己的绘制。

参数

canvas 在画布上绘制背景

protected boolean verifyDrawable (Drawable who)

如果你的视图子类显示他自己的可视化对象，他将要重写此方法并且为了显示可绘制返回 **true**。此操作允许进行绘制时有动画效果。

确认当重写从方法时，需调用父类相应方法。

参数

who 需判断的可绘制对象 (**Drawable**)。如果是你要显示的对象，返回 **True**，否则返回调用父类的结果。

返回值

boolean 如果可绘制对象 (**Drawable**) 已经在视图中显示，返回 **True** 否则返回 **false**。并且此处不允许使用动画。

CompoundButton.OnCheckedChangeListener

翻译署名：德罗德

译者博客：sparkrico.javaeye.com

版本：Android 2.2 r1

public static interface CompoundButton.OnCheckedChangeListener

android.widget.CompoundButton.OnCheckedChangeListener

概述

接口定义了一个在按钮的选中状态改变时要调用的回调函数。

公共方法

`public abstract void onCheckedChanged(CompoundButton buttonView, boolean isChecked)`

在按钮选中状态发生改变时被调用

参数

`buttonView` 选中状态发生改变的那个按钮

`isChecked` 按钮新的状态

CursorAdapter

译者署名： 深夜未眠

译者链接：<http://chirs1012f.javaeye.com/>

翻译时间： 2011-1-20

版本： Android 2.3 r1

结构

继承关系

public abstract class CursorAdapter extends BaseAdapter implements Filterable

```
java.lang.Object  
    android.widget.BaseAdapter  
        android.widget.CursorAdapter
```

子类及间接子类

直接子类

ResourceCursorAdapter

间接子类

SimpleCursorAdapter

类概述

通过该类可用 [Cursor](#) 的方式访问数据库，并将查询出来的数据展示到列表视图([ListView](#))部件上。其中游标携带的结果集中必须有列名为“_id”的列，否则这个类无法工作。

构造函数

public CursorAdapter(Context context,Cursor c)

构造函数。每当数据库的数据发生改变时，适配器将调用 `requery()`重新查询以显示最新的数据。

参数

context	应用程序上下文。
c	用来获取数据的游标(Cursor)

public CursorAdapter(Context context,Cursor c, boolean autoRequery)

构造函数。每当数据库的数据发生改变时，适配器将调用 `requery()`重新查询以显示最新的数据。

参数

context	应用程序上下文。
c	用来获取数据的 Cursor
autoRequery	设置为 <code>true</code> 时，每当数据库的数据发生改变时，适配器将调用 <code>requery()</code> 重新查询以显示最新的数据。

公共方法

public abstract void bindView (View view, Context context, Cursor cursor)

重用一个已有的 `view`，使其显示当前 `cursor` 所指向的数据。

参数

view	已存在的视图, 返回之前 newView 方法创建的视图。
context	应用程序上下文
cursor	用于获取数据的 Cursor。Cursor 已经移到正确的位置。

`public void changeCursor (Cursor cursor)`

更改底层的游标为新传入的游标。如果游标已经存在则先关闭这个已存在的游标。

参数

cursor	新 Cursor。
--------	-----------

`public CharSequence convertToString (Cursor cursor)`

将 cursor 转换成 CharSequence。子类应该重写这个方法并转换它们的结果。这个方法的默认实现是：当 cursor 为空时返回一个空串，否则直接返回调用 cursor 的 `toString()`方法。

参数

cursor	将 cursor 转换成 CharSequence 对象。
--------	-------------------------------

返回值

返回表示 CharSequence 的值。

`public int getCount ()`

(译者注：获取适配器中数据的总行数。)

参见

[getCount\(\)](#)

`public Cursor getCursor ()`

返回当前适配器绑定的 Cursor 对象。

返回值

Cursor 对象。

`public View getDropDownView (int position, View convertView, ViewGroup parent)`

获取下拉列表选项指定位置的视图对象

参数

position	视图(View)对象的行索引。
convertView	重用已有的视图(View)对象。备注：在使用前你应该检查一下这个视图对象是否非空并且这个对象的类型是否合适。由此引伸出，如果该对象不能被转换并显示正确的数据，这个方法内部就会重新创建一个合适的视图(View)对象。
parent	不管是转换后的还是重新创建的视图(View)对象，始终都会

依附于 parent 对象上。

返回值

返回视图(View)对象, 该对象显示数据集指定位置上的数据。

`public Filter getFilter ()`

返回一个可以通过一种过滤模式来约束数据的过滤器。

这个方法通常在 Adapter 类实现。

返回值

一个用于约束数据的过滤器

public FilterQueryProvider getFilterQueryProvider ()

返回一个提供过滤的查询过滤器。若过滤器为 null，则不再过滤。

返回值

返回当前过滤器对象，如果不存在返回 null。

参见

[setFilterQueryProvider\(android.widget.FilterQueryProvider\)](#)

[runQueryOnBackgroundThread\(CharSequence\)](#)

public Object getItem (int position)

(译者注：获取数据集中指定位置上的数据项目)

参见

[getItem\(int\)](#)

public long getItemId (int position)

(译者注：获取数据集中的指定位置上的行 id。)

参见

[getItemId\(int\)](#)

public View getView (int position, View convertView, ViewGroup parent)

(译者注：获取一个显示数据集中指定位置数据段视图。可以手动创建视图，或者从 XML 设计文件填充。当视图从 XML 设计文件填充时，父视图（如 GridView，ListView 等）将接受默认的设计参数，除非使用 inflate(int, android.view.ViewGroup, boolean)去指定一个根视图和防止依附于根视图。)

参见

[getView\(int, View, ViewGroup\)](#)

public boolean hasStableIds ()

无论项 ID 代表的基础数据的是否变化都保持不变。

返回值

如果为 TRUE，意味着相同的 ID 始终引用相同的对象。

public View newDropDownView (Context context, Cursor cursor, ViewGroup parent)

生成一个新的下拉视图来保存 cursor 指向的数据

参数

context 应用程序全局信息接口（应用上下文）

cursor 获取数据的游标，它已经移动到正确的位置

parent 与新视图相关联的上级视图

返回值

新创建的视图。

public abstract View newView (Context context, Cursor cursor, ViewGroup parent)

新建一个视图来保存 cursor 指向的数据

参数

- context 应用程序全局信息接口（应用上下文）
- cursor 获取数据的游标，它已经移动到正确的位置
- parent 与新视图相关联的上级视图

返回值

新创建的视图。

```
public Cursor runQueryOnBackgroundThread (CharSequence constraint)
```

执行含指定约束的查询。此查询依赖于适配器的过滤器。查询是由 [FilterQueryProvider](#) 提供。如果没有指定 FilterQueryProvider，当前 cursor 不过滤只返回。该方法会通过 changeCursor(Cursor)方法返回一个 Cursor 对象，并且关闭掉先前的 Cursor 对象。这个方法始终在后台线程执行，而不是在应用程序的主线程（或是 UI 线程）中运行。规定：当参数 constraint 为 null 或为空时，该方法返回原始结果。

参数

- constraint 该查询必须被过滤的约束。

返回值

返回含有新的查询结果的 Cursor。

参考

- [getFilter\(\)](#)
- [getFilterQueryProvider\(\)](#)
- [setFilterQueryProvider\(android.widget.FilterQueryProvider\)](#)

```
public void setFilterQueryProvider (FilterQueryProvider filterQueryProvider)
```

设置一个过滤器，用来过滤当前的 Cursor 对象。当这个适配器需要进行过滤操作时，[runQuery\(CharSequence\)](#)方法被调用。

参数

- filterQueryProvider 过滤器对象，设置为 null 时，就相当于移除了该过滤器。

参考

- [getFilterQueryProvider\(\)](#)
- [runQueryOnBackgroundThread\(CharSequence\)](#)

受保护方法

```
protected void init (Context context, Cursor c, boolean autoRequery)
```

（译者注：供构造函数使用初始化相关参数）

```
protected void onContentChanged ()
```

当 cursor 对象上的 ContentObserver 接收到改变的通知时就会调用该方法，其默认实现提供了自动重新查询方式，但可以被子类重写。

补充

文章精选

- [android CursorAdapter 的监听事件](#)
- [实现基于 Android 的英文电子词典](#)

CursorTreeAdapter

译者署名： 深夜未眠

译者链接：<http://chris1012f.javaeye.com/>

翻译时间： 2011-1-24

版本： Android 3.0 r1

结构

继承关系

```
public abstract class CursorTreeAdapter  
extends BaseExpandableListAdapter implements Filterable
```

```
java.lang.Object  
    android.widget.BaseExpandableListAdapter  
        android.widget.CursorTreeAdapter
```

子类及间接子类

直接子类

ResourceCursorTreeAdapter

间接子类

SimpleCursorTreeAdapter

类概述

通过该适配类可以用一连续的游标(Cursor)对象访问数据库，并将查询出来的数据展示到可伸缩的列表视图([ExpandableListView](#))部件上。顶层游标(Cursor)对象(在构造器中指定)显示全部组，后面的游标(Cursor)对象从 [getChildrenCursor\(Cursor\)](#) 获取并展示子元素组。其中游标携带的结果集中必须有个名为“_id”的列，否则这个类不起任何作用。

构造函数

```
public CursorTreeAdapter (Cursor cursor, Context context)
```

构造函数。每当数据库的数据发生改变时，适配器将调用 [requery\(\)](#)重新查询以显示最新的数据。

参数

cursor 为组(groups)提供数据的游标(Cursor)

context 应用程序上下文。

```
public CursorTreeAdapter (Cursor cursor, Context context, boolean autoRequery)
```

构造函数。

参数

cursor 为组(groups)提供数据的游标(Cursor)

context 应用程序上下文。

autoRequery 设置为 true 时，每当数据库的数据发生改变时，适配器将调用 [requery\(\)](#)重新查询以显示最新的数据。

公共方法

```
public void changeCursor (Cursor cursor)
```

更改相关的游标为新传入的游标。如果游标已经存在则先关闭这个已存在的游标。

参数

cursor 新游标(Cursor)对象。

```
public CharSequence convertToString (Cursor cursor)
```

将 cursor 转换成 CharSequence。子类应该重写这个方法并转换它们的结果。这个方法的默认实现是：当 cursor 为空时返回一个空串，否则直接返回调用 cursor 的 `toString()`方法。

参数

cursor 将 cursor 转换成 CharSequence 对象。

返回值

返回 CharSequence 类型的值（译者注：这个值是通过转换参数 cursor 类型而获得的。）

```
public Cursor getChild (int groupPosition, int childPosition)
```

获取指定组中的指定子元素的相关数据。

参数

groupPosition 组位置（该组内部含有子元素）

childPosition 子元素位置（相对于其它子元素）

返回值

返回指定子元素数据。

```
public long getChildId (int groupPosition, int childPosition)
```

获取指定组中的指定子元素 ID，这个 ID 在组里一定是唯一的。联合 ID（参见 [getCombinedChildId\(long, long\)](#)）在所有条目（所有组和所有元素）中也是唯一的。

参数

groupPosition 组位置（该组内部含有子元素）

childPosition 子元素位置（相对于其它子元素）

返回值

子元素关联 ID。

```
public View getChildView (int groupPosition, int childPosition, boolean isLastChild,  
View convertView, ViewGroup parent)
```

获取一个视图对象，显示指定组中的指定子元素数据。

参数

groupPosition 组位置（该组内部含有子元素）

childPosition 子元素位置（决定返回哪个视图）

isLastChild 子元素是否处于组中的最后一个

convertView 重用已有的视图(View)对象。注意：在使用前你应该检查一下这个视图对象是否非空并且这个对象的类型是否合适。由此引伸出，如果该对象不能被转换并显示正确的数据，这个方法就会调用 [getChildView\(int, int, boolean, View, ViewGroup\)](#) 来创建一个视图(View)对象。

parent 返回的视图(View)对象始终依附于的视图组。

返回值

指定位置上的子元素返回的视图对象

public int getChildrenCount (int groupPosition)

获取指定组中的子元素个数

参数

groupPosition 组位置 (决定返回哪个组的子元素个数)

返回值

返回组中的子元素个数

public Cursor getCursor ()

返回当前适配器绑定的 Cursor 对象。

返回值

Cursor 对象。

public Filter getFilter ()

返回一个可以通过一种过滤模式来约束数据的过滤器。

这个方法通常在 Adapter 类中实现。

返回值

返回一个用来限制数据的过滤器(Filter)对象。

public FilterQueryProvider getFilterQueryProvider ()

返回一个提供过滤的查询过滤器。若过滤器为 null，则不再过滤。

返回值

返回当前过滤器对象，如果不存在返回 null。

参考

[getFilterQueryProvider\(\)](#)

public Cursor getGroup (int groupPosition)

获取指定组中的数据

参数

groupPosition 组位置

返回值

返回组中的数据，也就是该组中的子元素数据。

public int getGroupCount ()

获取组的个数

返回值

组的个数

public long getGroupId (int groupPosition)

获取指定组的 ID，这个组 ID 必须是唯一的。联合 ID(参见 [getCombinedGroupId\(long\)](#))在所有条目(所有组和所有元素)中也是唯一的。

参数

groupPosition 组位置

返回值

返回组 ID

```
public View getGroupView (int groupPosition, boolean isExpanded, View convertView,  
ViewGroup parent)
```

获取显示指定组的视图对象。这个方法仅返回关于组的视图对象，要想获取子元素的视图对象，就需要调用 [getChildView\(int, int, boolean, View, ViewGroup\)](#)。

参数

groupPosition 组位置（决定返回哪个视图）

isExpanded 该组是展开状态还是伸缩状态

convertView 重用已有的视图对象。注意：在使用前你应该检查一下这个视图对象是否非空并且这个对象的类型是否合适。由此引伸出，如果该对象不能被转换并显示正确的数据，这个方法就会调用 [getGroupView\(int, boolean, View, ViewGroup\)](#) 来创建一个视图(View)对象。

parent 返回的视图对象始终依附于的视图组。

返回值

返回指定组的视图对象

```
public boolean hasStableIds ()
```

组和子元素是否持有稳定的 ID，也就是底层数据的改变不会影响到它们。

返回值

返回一个 Boolean 类型的值，如果为 TRUE，意味着相同的 ID 永远引用相同的对象。

```
public boolean isChildSelectable (int groupPosition, int childPosition)
```

是否选中指定位置上的子元素。

参数

groupPosition 组位置（该组内部含有子元素）

childPosition 子元素位置

返回值

是否选中子元素

```
public void notifyDataSetChanged (boolean releaseCursors)
```

通知数据集已改变，该方法还为任何带有缓存功能的游标提供不用释放资源的可选功能。

参数

releaseCursors 是否释放任何带有缓冲功能的游标资源

```
public void notifyDataSetChanged ()
```

该方法内部实现了在每个观察者上面调用 onChanged 事件。每当发现数据集有改变的情况，或者读取到数据的新状态时，就会调用此方法。

```
public void notifyDataSetChanged ()
```

该方法内部实现了在每个观察者上面调用 onInvalidated 事件。每当发现数据集监控有改

变的情况，比如该数据集不再有效，就会调用此方法。

public void onGroupCollapsed (int groupPosition)

当指定组收缩状态的时候调用此方法。

参数

groupPosition 指定组收缩状态

public Cursor runQueryOnBackgroundThread (CharSequence constraint)

执行含指定约束的查询。此查询依赖于适配器的过滤器。查询是由 [FilterQueryProvider](#) 提供。如果没有指定 FilterQueryProvider，当前 cursor 不过滤只返回。该方法会通过 changeCursor(Cursor)方法返回一个 Cursor 对象，并且关闭掉先前的 Cursor 对象。这个方法始终在后台线程执行，而不是在应用程序的主线程（或是 UI 线程）中运行。规定：当参数 constraint 为 null 或为空时，该方法返回原始结果。

参数

constraint 当某个查询需要过滤功能，使用参数 constraint 作为约束条件。

返回值

返回含有新的查询结果的 Cursor。

参考

[getFilter\(\)](#)

[getFilterQueryProvider\(\)](#)

[setFilterQueryProvider\(android.widget.FilterQueryProvider\)](#)

public void setChildrenCursor (int groupPosition, Cursor childrenCursor)

为指定组子元素设置游标(Cursor)，并关闭已存在的游标。这个方法是非常有用的，当进行异步查询操作的时候用来避免 UI 发生阻塞的情况。

参数

groupPosition 组位置(组中的哪个子元素要设置游标对象)

childrenCursor 组子元素的游标(Cursor)

public void setFilterQueryProvider (FilterQueryProvider filterQueryProvider)

设置一个过滤器，用来过滤当前的 Cursor 对象。当这个适配器需要进行过滤操作时，[runQuery\(CharSequence\)](#)方法被调用。

参数

filterQueryProvider 过滤生成器对象，设置为 null 时，就相当于移除了该生成器。

参考

[getFilterQueryProvider\(\)](#)

[runQueryOnBackgroundThread\(CharSequence\)](#)

public void setGroupCursor (Cursor cursor)

设置组游标对象。

参数

cursor 设置组游标对象，并关闭已存在的游标对象。

受保护方法

```
protected abstract void bindChildView (View view, Context context, Cursor cursor, boolean  
isLastChild)
```

用游标(Cursor)的方式将子元素数据绑定在一个已存在的视图(View)对象上。

参数

view	已存在的视图(View)对象，也就是之前 new 出来的。
context	应用程序上下文对象
cursor	获取数据的游标对象，它已经移动到正确的位置
IsLastChild	子元素是否处于组中的最后一个

```
protected abstract void bindGroupView (View view, Context context, Cursor cursor,  
boolean isExpanded)
```

用游标(Cursor)的方式将组数据绑定在一个已存在的视图(View)对象上。

参数

view	已存在的组视图(View)对象，也就是早先 new 出来的。
context	应用程序上下文对象，它已经移动到正确的位置
cursor	获取数据的游标对象
isExpanded	该组是展开状态还是伸缩状态

```
protected abstract Cursor getChildrenCursor (Cursor groupCursor)
```

获取指定组中的子元素游标对象。子类必须实现这个方法，用于在指定组中返回子元素数据。

如果你想用异步查询的方式避免 UI 阻塞的情况发生，可能会返回 null 或是在稍后调用 [setChildrenCursor\(int, Cursor\)](#)。

你有责任在 Activity 生命周期中管理这个游标对象，有一个非常好的思路：使用 [managedQuery\(Uri, String\[\], String, String\[\], String\)](#) 来管理它们。在某些情况下，适配器本身会使游标停止工作，但这个特例不会总是出现，所以我们要确保有效地管理好游标对象。

参数

groupCursor 组游标对象，决定返回哪个组中的子元素游标对象。

返回值

返回指定组中的子元素游标对象或者为 null。

```
protected abstract View newChildView (Context context, Cursor cursor, boolean  
isLastChild, ViewGroup parent)
```

创建一个新的子元素视图并持有指向数据的游标 cursor。

参数

context	应用程序上下文对象
cursor	获取数据的游标对象，它已经移动到正确的位置
IsLastChild	子元素是否处于组中的最后一个
parent	新视图(View)所依附于的父对象。

```
protected abstract View newGroupView (Context context, Cursor cursor, boolean  
isExpanded, ViewGroup parent)
```

创建一个新的组视图并持有组中指向数据的游标 cursor。

参数

context	应用程序上下文对象
cursor	获取数据的游标对象，它已经移动到正确的位置
isExpanded	该组是否展开状态
parent	新视图(View)所依附于的父对象。

补充

文章精选

[说说定制自己的 CursorAdapter](#)

DatePicker

农民伯伯
版本: Android 2.2 r1

`public class DatePicker extends FrameLayout`

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.FrameLayout
                android.widget.DatePicker
```

概述



一个选择年月日的日历布局视图。对于对话框样式的日历视图，参见 [DatePickerDialog](#).

公共方法

`public int getDayOfMonth ()`

获取选择的天数

`public int getMonth ()`

获取选择的月份。(注意: 返回数值为 0..11, 需要自己+1 来显示)

`public int getYear ()`

获取选择的年份

`public void init (int year, int monthOfYear, int dayOfMonth,
DatePicker.OnDateChangedListener onDateChangedListener)`

初始化状态。(译者注: 初始化年月日)

参数

`year` 初始年 (译者注: 注意使用 `new Date()` 初始化年时, 需要+1900, 如下: `date.getYear() + 1900`)

`monthOfYear` 初始月。

`dayOfMonth` 初始日。

`onDateChangedListener` 日期改变时通知用户的事件监听, 可以为空(null)。

`public void setEnabled (boolean enabled)`

设置视图的启用状态。该启用状态随子类的不同而有不同的解释。

参数

`enabled` True if this view is enabled, false otherwise.

设置为 true 表示启动视图，反之禁用。

```
public void updateDate (int year, int monthOfYear, int dayOfMonth)
```

更新日期

受保护方法

```
protected void dispatchRestoreInstanceState (SparseArray<Parcelable> container)
```

重写使我们能够完全控制这小部件的保存或恢复。(译者注：此处直接调用了父类的 *ViewGroup.dispatchThawSelfOnly* 方法)

参数

`container` SparseArray 持有保存以前的状态。The SparseArray which holds previously saved state.

```
protected void onRestoreInstanceState (Parcelable state)
```

允许视图重新应用以前通过 `onSaveInstanceState()` 生成代表内部的状态。这个函数决不调用一个空的状态。

参数

`state` 返回以前调用 `onSaveInstanceState()` 保存下来的状态。

```
protected Parcelable onSaveInstanceState ()
```

允许视图生成一个代表内部的状态，以后可用于创建一个与之相同的新的实例。这种状态应该只包含非持久或以后不能够重建的信息。例如，你决不存储你当前在屏幕上的位置，因为这会在视图的层面上重新计算放置一个新的实例。

你可以存储到这里的一些例子：一个文本框中当前光标的位置（但通常不是文字本身，文字通常保存在内容提供者(content provider)或其他持久的储存中），一个列表视图中的当前选中项。

返回值

返回一个包含视图当前状态的 `Parcelable` 对象，或没有什么状态保存时返回 `null`。默认实现返回 `null`。

DatePicker.OnDateChangedListener

译者署名: cnmahj

译者链接: <http://android.toolib.net/blog>

版本: Android 2.3 r1

结构

继承关系

public static interface DatePicker.OnDateChangedListener

子类及间接子类

直接子类

DatePickerDialog

类概述

表明用户变更了日期的回调函数。

公共方法

```
public abstract void onDateChanged(DatePicker view, int year, int monthOfYear, int dayOfMonth)
```

(译者注: 当用户修改日期 onDateChanged() 将被调用)

参数

view 与监听器关联的视图。

year 用户设置的年。

monthOfYear 用户设置的月份(0–11)，与 [Calendar](#) 兼容。

dayOfMonth 用户设置的日期。

补充

示例代码

```
package com.adakoda.android.datepickerdialogsample;

import java.util.Calendar;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.Toast;

public class DatePickerDialogSampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final Calendar calendar = Calendar.getInstance();
        final int year = calendar.get(Calendar.YEAR);
        final int month = calendar.get(Calendar.MONTH);
        final int day = calendar.get(Calendar.DAY_OF_MONTH);
```

```
final DatePickerDialog datePickerDialog = new DatePickerDialog(
    this,
    new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int
monthOfYear, int dayOfMonth) {
            Toast.makeText(DatePickerDialogSampleActivity.this,
                String.valueOf(year) + "/" +
                String.valueOf(monthOfYear + 1) + "/" +
                String.valueOf(dayOfMonth),
                Toast.LENGTH_SHORT).show();
        }
    },
    year, month, day);
datePickerDialog.show();
}
}
```

DigitalClock

农民伯伯
版本: Android 2.2 r1

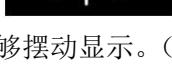
public class DigitalClock extends TextView

```
java.lang.Object  
    android.view.View  
        android.widget.TextView  
            android.widget.DigitalClock
```

概述

20:15:04



像 `AnalogClock` ()，但是是数字的。显示秒。修正：分开显示小时/分钟/秒，均衡的字体不能够摆动显示。（译者注：根据字体按比例显示小时/分钟/秒，无法像 `AnalogClock` 转动显示）

受保护方法

protected void onAttachedToWindow ()

视图 (`DigitalClock`) 附在一个窗体上时调用。在这个点的表面进行绘制。注意这个方法保证在 `onDraw(Canvas)` 以前调用，但是它可以在第一次调用 `onDraw` 以前任何时间调用——包括 `onMeasure(int, int)` 之前或之后。

protected void onDetachedFromWindow ()

视图 (`DigitalClock`) 从窗体上分离 (移除) 时调用。在这个点的表面不再有画面绘制。

EditText

农民伯伯

版本: Android 2.2

```
java.lang.Object  
    android.view.View  
        android.widget.TextView  
            android.widget.EditText
```

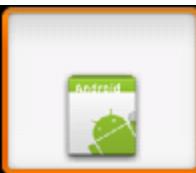
直接子类:

AutoCompleteTextView, ExtractEditText

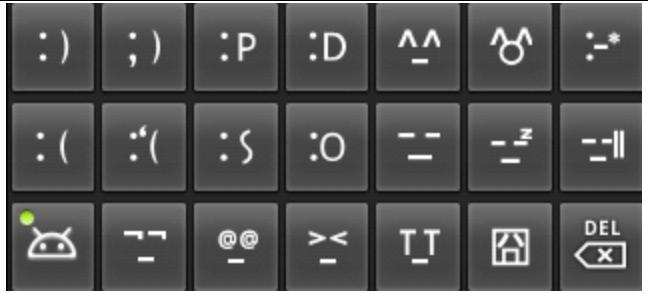
间接子类:

MultiAutoCompleteTextView

继承自 TextView 的 xml 属性说明:

属性名称	描述
android:autoLink	设置是否当文本为 URL 链接/email/电话号码/map 时, 文本显示为可点击的链接。可选值 (none/web/email/phone/map/all)。这里只有在同时设置 text 时才自动识别链接, 后来输入的无法自动识别。
android:autoText	自动拼写帮助。这里单独设置是没有效果的, 可能需要其他输入法辅助才行, 效果参见 视频 。
android:bufferType	指定 getText() 方式取得的文本类别。选项 editable 类似于 StringBuilder 可追加字符, 也就是说 getText 后可调用 append 方法设置文本内容。spannable 则可在给定的字符区域使用样式, 参见 这里 1 、 这里 2 。
android:capitalize	设置英文字母大写类型。设置如下值: sentences 仅第一个字母大写; words 每一个单词首字母大小, 用空格区分单词; characters 每一个英文字母都大写。在模拟器上用 PC 键盘直接输入可以出效果, 但是用软键盘无效果。
android:cursorVisible	设定光标为显示/隐藏, 默认显示。如果设置 false, 即使选中了也不显示光标栏。
android:digits	设置允许输入哪些字符。如 “1234567890.+-*%/\n()”
android:drawableTop	在 text 的正上方输出一个 drawable。在 EditText 中的效果比较搞笑:  ，居然在文本框里, 而且删不了。
android:drawableBottom	在 text 的下方输出一个 drawable(如图片)。如果指定一个颜色的话会把 text 的背景设为该颜色, 并且同时和

	background 使用时覆盖后者。
android:drawableLeft	在 text 的左边输出一个 drawable(如图片)。
android:drawablePadding	设置 text 与 drawable(图片)的间隔，与 drawableLeft、drawableRight、drawableTop、drawableBottom 一起使用，可设置为负数，单独使用没有效果。
android:drawableRight	在 text 的右边输出一个 drawable，如图片。
android:editable	设置是否可编辑。仍然可以获取光标，但是无法输入。
android:editorExtras	指定特定输入法的扩展，如“com.mydomain.im.SOME_FIELD”。源码跟踪至 EditorInfo.extras，暂无相关实现代码。
android:ellipsize	设置当文字过长时，该控件该如何显示。有如下值设置：“start”——省略号显示在开头；“end”——省略号显示在结尾；“middle”——省略号显示在中间；“marquee”——以跑马灯的方式显示(动画横向移动)
android:freezesText	设置保存文本的内容以及光标的位置。参见： 这里 。
android:gravity	设置文本位置，如设置成“center”，文本将居中显示。
android:hint	Text 为空时显示的文字提示信息，可通过 textColorHint 设置提示信息的颜色。
android:imeOptions	设置软键盘的 Enter 键。有如下值可设置：normal, actionUnspecified, actionNone, actionGo, actionSearch, actionSend, actionNext, actionDone, flagNoExtractUi, flagNoAccessoryAction, flagNoEnterAction。可用' '设置多个。这里仅设置显示图标之用，参见文章末尾例子。
android:imeActionId	设置 IME 动作 ID，在 onEditorAction 中捕获判断进行逻辑操作。
android:imeActionButton	设置 IME 动作标签。但是不能保证一定会使用，猜想在输入法扩展的时候应该有用。
android:includeFontPadding	设置文本是否包含顶部和底部额外空白，默认为 true。
android:inputMethod	为文本指定输入法，需要完全限定名(完整的包名)。例如：com.google.android.inputmethod.pinyin，但是这里报错找不到。关于自定义输入法参见 这里 。 sentences 仅第一个字母大写；words 每一个单词首字母大小，用空格区分单词；characters 每一个英文字母都大写
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型。有如下值设置：none、text、textCapCharacters 字母大小、textCapWords 单词首字母大小、textCapSentences 仅第一个字母大小、textAutoCorrect、textAutoComplete 自动完成、textMultiLine 多行输入、textImeMultiLine 输入法多行(如果支持)、textNoSuggestions 不提示、textEmailAddress 电子邮件地址、textEmailSubject 邮件主题、textShortMessage 短信息(会多一个表情按钮出来，点开如下图：



)、textLongMessage 长讯息?、textPersonName 人名、textPostalAddress 地址、textPassword 密码、textVisiblePassword 可见密码、textWebEditText 作为网页表单的文本、textFilte 文本筛选过滤、textPhonetic 拼音输入、numberSigned 有符号数字格式、numberDecimal 可带小数点的浮点格式、phone 电话号码、datetime 时间日期、date 日期、time 时间。部分参考[这里](#)。

android:marqueeRepeatLimit	在 ellipsize 指定 marquee 的情况下，设置重复滚动的次数，当设置为 marquee_forever 时表示无限次。
android:ems	设置 TextView 的宽度为 N 个字符的宽度。参见 TextView 中此属性的截图。
android:maxEms	设置 TextView 的宽度为最长为 N 个字符的宽度。与 ems 同时使用时覆盖 ems 选项。
android:minEms	设置 TextView 的宽度为最短为 N 个字符的宽度。与 ems 同时使用时覆盖 ems 选项。
android:maxLength	限制输入字符数。如设置为 5，那么仅可以输入 5 个汉字/数字/英文字母。
android:lines	设置文本的行数，设置两行就显示两行，即使第二行没有数据。
android:maxLines	设置文本的最大显示行数，与 width 或者 layout_width 结合使用，超出部分自动换行，超出行数将不显示。
android:minLines	设置文本的最小行数，与 lines 类似。
android:linksClickable	设置链接是否点击连接，即使设置了 autoLink。
android:lineSpacingExtra	设置行间距。
android:lineSpacingMultiplier	设置行间距的倍数。如"1.2"
android:numeric	如果被设置，该 TextView 有一个数字输入法。有如下值设置：integer 正整数、signed 带符号整数、decimal 带小数点浮点数。
android:password	以小点"．"显示文本
android:phoneNumber	设置为电话号码的输入方式。
android:privateImeOptions	提供额外的输入法选项(字符串格式)。依据输入法而决定是否提供，如 这里 所见。自定义输入法继承 InputMethodService。 这篇文章 也许有帮助。
android:scrollHorizontally	设置文本超出 TextView 的宽度的情况下，是否出现横拉条。
android:selectAllOnFocus	如果文本是可选择的，让他获取焦点而不是将光标移动为文本的开始位置或者末尾位置。TextView 中设置后无效

	果。
android:shadowColor	指定文本阴影的颜色，需要与 shadowRadius 一起使用。 参见 TextView 中此属性的截图。
android:shadowDx	设置阴影横向坐标开始位置。
android:shadowDy	设置阴影纵向坐标开始位置。
android:shadowRadius	设置阴影的半径。设置为 0.1 就变成字体的颜色了，一般设置为 3.0 的效果比较好。
android:singleLine	设置单行显示。如果和 layout_width 一起使用，当文本不能全部显示时，后面用“...”来表示。如 android:text="test_singleLine" android:singleLine="true" android:layout_width="20dp" 将只显示“t...”。如果不设置 singleLine 或者设置为 false，文本将自动换行
android:text	设置显示文本.
android:textAppearance	设置文字外观。如 “?android:attr/textAppearanceLargeInverse”这里引用的是系统自带的一个外观，? 表示系统是否有这种外观，否则使用默认的外观。可设置的值如下： textAppearanceButton/textAppearanceInverse/textAppearanceLarge/textAppearanceLargeInverse/textAppearanceMedium/textAppearanceMediumInverse/textAppearanceSmall/textAppearanceSmallInverse
android:textColor	设置文本颜色
android:textColorHighlight	被选中文字的底色，默认为蓝色
android:textColorHint	设置提示信息文字的颜色，默认为灰色。与 hint 一起使用。
android:textColorLink	文字链接的颜色.
android:textScaleX	设置文字缩放，默认为 1.0f。参见 TextView 的截图。
android:textSize	设置文字大小，推荐度量单位"sp"，如"15sp"
android:textStyle	设置字形[bold(粗体) 0, italic(斜体) 1, bolditalic(又粗又斜) 2] 可以设置一个或多个，用 “ ” 隔开
android:typeface	设置文本字体，必须是以下常量值之一：normal 0, sans 1, serif 2, monospace(等宽字体) 3]
android:height	设置文本区域的高度，支持度量单位：px(像素)/dp/sp/in/mm(毫米)
android:maxHeight	设置文本区域的最大高度
android:minHeight	设置文本区域的最小高度
android:width	设置文本区域的宽度，支持度量单位：px(像素)/dp/sp/in/mm(毫米)，与 layout_width 的区别看 这里 。
android: maxWidth	设置文本区域的最大宽度
android: minWidth	设置文本区域的最小宽度

android:imeOptions 例子:

```
<EditText android:id="@+id/txtTest" android:imeOptions="actionGo"
    android:layout_width="100dp"
    android:layout_height="wrap_content"></EditText>

((EditText) findViewById(R.id.txtTest)).setOnEditorActionListener(new
TextView.OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId,
        KeyEvent event) {
        if (actionId == EditorInfo.IME_ACTION_GO) {
            Toast.makeText(ActivityResult.this, "你点了Go!", Toast.LENGTH_SHORT).show();
        }
        return false;
    }
});
```

Filter

译者署名: henly.zhang
译者链接: lovecoool@gmail.com
翻译时间: 2010-11-07
版本: Android 2.2 r1

结构

继承关系

public abstract class Filter extends Object

```
java.lang.Object  
    android.widget.Filter
```

类概述

过滤器通过过滤模式来约束数据，通常由实现了 Filterable 接口的子类来生成。

过滤操作是通过调用 filter(CharSequence) 或者 filter(CharSequence, android.widget.FilterListener) 这些异步方法来完成的。以上方法一旦被调用，过滤请求就会被递交到请求队列中等待处理，同时该操作会取消那些之前递交的但是还没有被处理的请求。

构造函数

public Filter ()

创建一个新的异步过滤器。

公共方法

public CharSequence convertResultToString (Object resultValue)

将受过滤的集合对象转换成 CharSequence 文本。所有继承了 Filter 的子类应该重写该方法。该方法的默认实现：如果参数为 null 则返回空字符串或者返回参数的字符串形式。

参数

resultValue 转换成 CharSequence 文本的对象

返回值

CharSequence 文本

public final void filter(CharSequence constraint, Filter.FilterListener listener)

启动一个异步的过滤操作。对该方法的调用会取消之前队列中等待处理的过滤请求并且递交新的过滤请求等待执行。完成过滤操作之后，通知监听器。

参数

constraint 过滤数据的约束条件

listener 监听过滤操作完成之后发出的通知

参见

filter(CharSequence)

performFiltering(CharSequence)

publishResults(CharSequence, android.widget.Filter.FilterResults)

```
public final void filter(CharSequence constraint)
```

启动一个异步的过滤操作。对该方法的调用会取消之前队列中等待处理的过滤请求并且递交新的过滤请求等待执行。

参数

constraint 过滤数据的约束条件

参见

```
filter(CharSequence, android.widget.Filter.FilterListener)
```

受保护方法

```
protected abstract Filter.FilterResults performFiltering (CharSequence constraint)
```

根据约束条件调用一个工作线程过滤数据。子类必须实现该方法来执行过滤操作。过滤结果以 Filter.FilterResults 的形式返回，然后在 UI 线程中通过 publishResults(CharSequence, android.widget.Filter.FilterResults) 方法来发布。

约定：当约束条件为 null 时，原始数据必须被恢复。

参数

constraint 约束条件

返回值

过滤结果

参见

```
filter(CharSequence, android.widget.Filter.FilterListener)
```

```
publishResults(CharSequence, android.widget.Filter.FilterResults)
```

```
Filter.FilterResults
```

```
protected abstract void publishResults (CharSequence constraint, Filter.FilterResults results)
```

通过调用 UI 线程在用户界面发布过滤结果。子类必须实现该方法来显示 performFiltering(CharSequence) 的过滤结果。

参数

constraint 约束条件

results 过滤结果

参见

```
filter(CharSequence, android.widget.Filter.FilterListener)
```

```
performFiltering(CharSequence)
```

```
Filter.FilterResults
```

补充

示例代码

未过滤的数据如下图所示：

TestFilter		
henly	22	
john	23	
lilei	22	

为过滤器设置约束条件（只显示年龄为 22 的用户）：

TestFilter		
henly	22	
lilei	22	

Java: TestFilter.java

```
public class TestFilter extends ListActivity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        ArrayList<HashMap<String, String>> list = new  
            ArrayList <HashMap<String, String>>();  
        HashMap<String, String> map1 = new HashMap<String, String>();  
        HashMap<String, String> map2 = new HashMap<String, String>();  
        HashMap<String, String> map3 = new HashMap<String, String>();  
        map1.put("name", "henly");  
        map1.put("age", "22");  
        map2.put("name", "john");  
        map2.put("age", "23");  
        map3.put("name", "lilei");  
        map3.put("age", "22");  
        list.add(map1);  
        list.add(map2);  
        list.add(map3);  
        SimpleAdapter simpleAdapter = new SimpleAdapter(this, list,  
            R.layout.user, new String[]{"name", "age"}, new int[]{R.id.name, R.id.age});  
        String str = new String("22");  
        CharSequence constraint = str.subSequence(0, str.length());  
        Filter filter = simpleAdapter.getFilter(); //得到一个过滤器  
        filter.filter(constraint); //为该过滤器设置约束条件  
        setListAdapter(simpleAdapter);  
    }  
}
```

```
}
```

XML: main.xml

```
<?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"
    >
    <LinearLayout android:id="@+id/listlinearlayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <ListView android:id="@+id/android:list"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:drawSelectorOnTop="false"
            android:scrollbars="vertical" />
    </LinearLayout>
</LinearLayout>
```

XML: user.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="12dip"
    android:paddingRight="12dip"
    android:paddingTop="1dip"
    android:paddingBottom="1dip"
    >
    <TextView android:id="@+id/name"
        android:layout_width="150dip"
        android:layout_height="30dip"
        android:textSize="12pt"
        />
    <TextView android:id="@+id/age"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="12pt"
        />
</LinearLayout>
```

ExpandableListAdapter

译者署名： 深夜未眠

译者链接：<http://chris1012f.javaeye.com/>

翻译时间： 2011-1-25

版本： Android 3.0 r1

结构

继承关系

public interface ExpandableListAdapter

 android.widget.ExpandableListAdapter

子类及间接子类

 间接子类

 BaseExpandableListAdapter, CursorTreeAdapter, ResourceCursorTreeAdapter,

 SimpleCursorTreeAdapter, SimpleExpandableListAdapter

类概述

这个适配器在 [ExpandableListView](#) 和底层数据之间起到了一个衔接的作用。该接口的实现类提供了访问子元素（以组的形式将它们分类）的数据；同样，也提供了为子元素和组创建相应的视图。

公共方法

public abstract boolean areAllItemsEnabled ()

ExpandableListAdapter 里面的所有条目都可用吗？如果是 yes，就意味着所有条目可以选择和点击了。

返回值

 返回 True 表示所有条目均可用。

参见

[areAllItemsEnabled\(\)](#)

public abstract Cursor getChild (int groupPosition, int childPosition)

获取指定组中的指定子元素数据。

参数

 groupPosition 组位置（该组内部含有子元素）

 childPosition 子元素位置（相对于其它子元素）

返回值

 返回指定子元素数据。

public abstract long getChildId (int groupPosition, int childPosition)

获取指定组中的指定子元素 ID，这个 ID 在组里一定是唯一的。联合 ID（参见 [getCombinedChildId\(long, long\)](#)）在所有条目（所有组和所有元素）中也是唯一的。

参数

 groupPosition 组位置（该组内部含有子元素）

`childPosition` 子元素位置（相对于其它子元素）

返回值

子元素关联 ID。

```
public abstract View getChildView (int groupPosition, int childPosition, boolean isLastChild,
View convertView, ViewGroup parent)
```

获取一个视图对象，显示指定组中的指定子元素数据。

参数

`groupPosition` 组位置（该组内部含有子元素）

`childPosition` 子元素位置（决定返回哪个视图）

`isLastChild` 子元素是否处于组中的最后一个

`convertView` 重用已有的视图(View)对象。注意：在使用前你应该检查一下这个视图对象是否非空并且这个对象的类型是否合适。由此引伸出，如果该对象不能被转换并显示正确的数据，这个方法就会调用 [getChildView\(int, int, boolean, View, ViewGroup\)](#) 来创建一个视图(View)对象。

`parent` 返回的视图(View)对象始终依附于的视图组。

返回值

指定位置上的子元素返回的视图对象

```
public abstract int getChildrenCount (int groupPosition)
```

获取指定组中的子元素个数

参数

`groupPosition` 组位置（决定返回哪个组的子元素个数）

返回值

指定组的子元素个数

```
public abstract long getCombinedChildId (long groupId, long childId)
```

从列表所有项(组或子项)中获得一个唯一的子 ID 号。可折叠列表要求每个元素(组或子项)在所有的子元素和组中有一个唯一的 ID。本方法负责根据所给的子 ID 号和组 ID 号返回唯一的 ID。此外，若 `hasStableIds()` 是 true，那么必须要返回稳定的 ID。

参数

`groupId` 包含该子元素的组 ID

`childId` 子元素的 ID

返回值

列表所有项(组或子项)中唯一的(和可能稳定)的子元素 ID 号。(译者注：ID 理论上是稳定的，不会发生冲突的情况。也就是说，这个列表会有组、子元素，它们的 ID 都是唯一的。)

```
public abstract Cursor getGroup (int groupPosition)
```

获取指定组中的数据

参数

`groupPosition` 组位置

返回值

返回组中的数据，也就是该组中的子元素数据。

```
public abstract int getGroupCount ()
```

获取组的个数

返回值

组的个数

```
public abstract long getGroupId (int groupPosition)
```

获取指定组的 ID，这个组 ID 必须是唯一的。联合 ID(参见 [getCombinedGroupId\(long\)](#))在所有条目(所有组和所有元素)中也是唯一的。

参数

groupPosition 组位置

返回值

返回组相关 ID

```
public abstract View getGroupView (int groupPosition, boolean isExpanded, View convertView, ViewGroup parent)
```

获取显示指定组的视图对象。这个方法仅返回关于组的视图对象，要想获取子元素的视图对象，就需要调用 [getChildView\(int, int, boolean, View, ViewGroup\)](#)。

参数

groupPosition 组位置 (决定返回哪个视图)

isExpanded 该组是展开状态还是伸缩状态

convertView 重用已有的视图对象。注意：在使用前你应该检查一下这个视图对象是否非空并且这个对象的类型是否合适。由此引伸出，如果该对象不能被转换并显示正确的数据，这个方法就会调用 [getGroupView\(int, boolean, View, ViewGroup\)](#) 来创建一个视图(View)对象。

parent 返回的视图对象始终依附于的视图组。

返回值

返回指定组的视图对象

```
public abstract boolean hasStableIds ()
```

组和子元素是否持有稳定的 ID,也就是底层数据的改变不会影响到它们。

返回值

返回一个 Boolean 类型的值，如果为 TRUE，意味着相同的 ID 永远引用相同的对象。

```
public abstract boolean isChildSelectable (int groupPosition, int childPosition)
```

是否选中指定位置上的子元素。

参数

groupPosition 组位置 (该组内部含有这个子元素)

childPosition 子元素位置

返回值

是否选中子元素

```
public abstract boolean isEmpty ()
```

返回值

如果当前适配器不包含任何数据则返回 True。经常用来决定一个空视图是否应该被显示。一个典型的实现将返回表达式 `getCount() == 0` 的结果，但是由于 `getCount()` 包含了头部和尾部，适配器可能需要不同的行为。

参见

[isEmpty\(\)](#)

public abstract void onGroupCollapsed (int groupPosition)

当组收缩状态的时候此方法被调用。

参数

`groupPosition` 收缩状态的组索引

public abstract void onGroupExpanded (int groupPosition)

当组展开状态的时候此方法被调用。

参数

`groupPosition` 展开状态的组位置

public abstract void registerDataSetObserver (DataSetObserver observer)

注册一个观察者(`observer`)，当此适配器数据修改时即调用此观察者。

参数

`observer` 当数据修改时通知调用的对象。

public abstract void unregisterDataSetObserver (DataSetObserver observer)

取消先前通过 `registerDataSetObserver(DataSetObserver)` 方式注册进该适配器中的观察者对象。

参数

`observer` 取消这个观察者的注册

补充

文章精选

[android 可展开（收缩）的列表 ListView\(ExpandableListView\)](#)

[android CursorAdapter 的监听事件](#)

ExpandableListView.OnChildClickListener

译者署名： 情敌贝多芬

译者链接： <http://liubey.javaeye.com/>

版本： Android 3.0 r1

结构

继承关系

public static interface ExpandableListView.OnChildClickListener

java.lang.Object

 android.widget.ExpandableListView.OnChildClickListener

子类及间接子类

 间接子类

[ExpandableListActivity](#)

类概述

这是一个定义了当可折叠列表(expandable list)里的子元素(child)发生点击事件时调用的回调方法的接口。

公共方法

```
public abstract boolean onChildClick (ExpandableListView parent, View v, int groupPosition,  
int childPosition, long id)
```

用当可折叠列表里的子元素(child)被点击的时候被调用的回调方法。

参数

parent	发生点击动作的 ExpandableListView
v	在 expandable list/ListView 中被点击的视图(View)
groupPosition	包含被点击子元素的组(group)在 ExpandableListView 中的位置(索引)
childPosition	被点击子元素(child)在组(group)中的位置
id	被点击子元素(child)的行 ID(索引)

返回值

当点击事件被处理时返回 true

ExpandableListView.OnGroupClickListener

译者署名： 情敌贝多芬

译者链接： <http://liubey.javaeye.com/>

版本： Android 3.0 r1

结构

继承关系

public static interface ExpandableListView.OnGroupClickListener

java.lang.Object

 android.widget.ExpandableListView.OnGroupClickListener

类概述

这是一个定义了当可折叠列表(expandable list)里的组(group)发生点击事件时调用的回调方法的接口。

公共方法

```
public abstract boolean onGroupClick (ExpandableListView parent, View v, int groupPosition,  
long id)
```

用当可折叠列表里的组(group)被点击的时候被调用的回调方法。

参数

parent	发生点击事件的 ExpandableListConnector
v	在 expandable list/ListView 中被点击的视图(View)
groupPosition (索引)	被点击的组(group)在 ExpandableListConnector 中的位置

id	被点击的组(group)的行 ID(索引)
----	-----------------------

返回值

当点击事件被处理的时候返回 true

ExpandableListView.OnGroupCollapseListener

译者署名： 深夜未眠

译者链接: <http://chris1012f.javaeye.com/>

翻译时间： 2011-1-26

版本： Android 3.0 r1

结构

继承关系

public interface ExpandableListView.OnGroupCollapseListener

java.lang.Object

 android.widget.ExpandableListView.OnGroupCollapseListener

子类及间接子类

 间接子类

 ExpandableListActivity

类概述

当收缩某个组时，就会发出通知。

公共方法

public abstract void onGroupCollapse (int groupPosition)

每当收缩当前可伸缩列表中的某个组时，就调用该方法。

参数

groupPosition 组位置，也就是收缩的那个组的位置。

ExpandableListView.OnGroupExpandListener

译者署名： 深夜未眠

译者链接: <http://chris1012f.javaeye.com/>

翻译时间： 2011-1-26

版本： Android 3.0 r1

结构

继承关系

public interface ExpandableListView.OnGroupExpandListener

java.lang.Object

 android.widget.ExpandableListView.OnGroupExpandListener

子类及间接子类

 间接子类

 ExpandableListActivity

类概述

当展开某个组时，就会发出通知。

公共方法

public abstract void onGroupExpand (int groupPosition)

每当展开当前可伸缩列表中的某个组时，就调用该方法。

参数

groupPosition 组位置，也就是展开的那个组的位置。

Filter.FilterListener

译者署名: henly.zhang

译者链接: lovecoool@gmail.com

版本: Android 2.2 r1

结构

继承关系

public static interface Filter.FilterListener

子类及间接子类

直接子类

AbsListView, AutoCompleteTextView, ExpandableListView, GridView, ListView,
MultiAutoCompleteTextView

类概述

监听器用于接收过滤操作完成之后发出的通知。

公共方法

public abstract void onFilterComplete (int count)

过滤操作结束的通知

参数

count 被过滤数据的数量

Filter.FilterResults

译者署名: henly.zhang

译者链接: lovecoool@gmail.com

版本: Android 2.2 r1

结构

继承关系

protected static class Filter.FilterResults

java.lang.Object

 android.widget.Filter.FilterResults

类概述

持有过滤操作完成之后的数据。该数据包括过滤操作之后的数据的值以及数量。

字段

public int count

包含过滤操作之后的数据的数量

public Object values

包含过滤操作之后的数据的值

构造函数

public Filter.FilterResults ()

构造一个 FilterResults 对象。

Filterable

译者署名： 德罗德

译者链接：<http://sparkrico.javaeye.com>

翻译时间： 2010-11-03

版本： Android 2.2 r1

结构

继承关系

public interface Filterable

android.widget.Filterable

子类及间接子类

直接子类

ArrayAdapter<T>, CursorAdapter, CursorTreeAdapter, HeaderViewListAdapter,
ResourceCursorAdapter, ResourceCursorTreeAdapter, SimpleAdapter,
SimpleCursorAdapter, SimpleCursorTreeAdapter

类概述

定义一个可过滤的行为。一个可过滤的类可以通过一个过滤器筛选它的数据。过滤类通常由适配器（Adapter）实现。

公共方法

public abstract Filter getFilter ()

返回一个可以通过一种过滤模式来约束数据的过滤器。

这个方法通常被 Adapter 类实现。

返回值

一个用于约束数据的过滤器

补充

文章链接

[自动匹配的联系人多选框（Filterable 用法）](#)

[如何在 ListView 上加上按键过滤的功能](#)

Gallery

译者署名： henly.zhang

译者链接：<http://www.blogjava.net/zlh320321>

版本：Android 2.3 r1

结构

继承关系

```
public class Gallery extends AbsSpinner
    implements GestureDetector.OnGestureListener
```

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.AdapterView<T extends android.widget.Adapter>
                android.widget.AbsSpinner
                    android.widget.Gallery
```

类概述



一个锁定中心条目并且拥有水平滚动列表的视图。

Gallery（画廊）使用 [Theme.galleryItemBackground](#) 作为 Gallery（画廊）适配器中的各视图的默认参数。如果你没有设置，你就需要调整一些 Gallery（画廊）的属性，比如间距。

Gallery（画廊）中的视图应该使用 `Gallery.LayoutParams` 作为它们的布局参数类型。

参见 [Gallery tutorial](#)。

内部类

```
class Gallery.LayoutParams
```

Gallery(画廊)扩展了 LayoutParams，以此提供可以容纳当前的转换信息和先前的位置转换信息的场所。

XML 属性

属性名称	描述																																							
android:animationDuration	设置布局变化时动画的转换所需的时间(毫秒级)。仅在动画开始时计时。该值必须是整数，比如：100。																																							
android:gravity	指定在对象的 X 和 Y 轴上如何放置内容。指定一下常量中的一个或多个(使用“ ”分割)																																							
	<table border="1"><thead><tr><th>Constant</th><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>top</td><td>0x30</td><td>紧靠容器顶端，不改变其大小</td></tr><tr><td>bottom</td><td>0x50</td><td>紧靠容器底部，不改变其大小</td></tr><tr><td>left</td><td>0x03</td><td>紧靠容器左侧，不改变其大小</td></tr><tr><td>right</td><td>0x05</td><td>紧靠容器右侧，不改变其大小</td></tr><tr><td>center_vertical</td><td>0x10</td><td>垂直居中，不改变其大小</td></tr><tr><td>fill_vertical</td><td>0x70</td><td>垂直方向上拉伸至充满容器</td></tr><tr><td>center_horizontal</td><td>0x01</td><td>水平居中，不改变其大小</td></tr><tr><td>fill_horizontal</td><td>0x07</td><td>水平方向上拉伸使其充满容器</td></tr><tr><td>center</td><td>0x11</td><td>居中对齐，不改变其大小</td></tr><tr><td>fill</td><td>0x77</td><td>在水平和垂直方向上拉伸，使其充满容器</td></tr><tr><td>clip_vertical</td><td>0x80</td><td>垂直剪切(当对象边缘超出容器的时候，将上下边缘超出的部分剪切掉)</td></tr><tr><td>clip_horizontal</td><td>0x08</td><td>水平剪切(当对象边缘超出容器的时候，将左右边缘超出的部分剪切掉)</td></tr></tbody></table>	Constant	Value	Description	top	0x30	紧靠容器顶端，不改变其大小	bottom	0x50	紧靠容器底部，不改变其大小	left	0x03	紧靠容器左侧，不改变其大小	right	0x05	紧靠容器右侧，不改变其大小	center_vertical	0x10	垂直居中，不改变其大小	fill_vertical	0x70	垂直方向上拉伸至充满容器	center_horizontal	0x01	水平居中，不改变其大小	fill_horizontal	0x07	水平方向上拉伸使其充满容器	center	0x11	居中对齐，不改变其大小	fill	0x77	在水平和垂直方向上拉伸，使其充满容器	clip_vertical	0x80	垂直剪切(当对象边缘超出容器的时候，将上下边缘超出的部分剪切掉)	clip_horizontal	0x08	水平剪切(当对象边缘超出容器的时候，将左右边缘超出的部分剪切掉)
Constant	Value	Description																																						
top	0x30	紧靠容器顶端，不改变其大小																																						
bottom	0x50	紧靠容器底部，不改变其大小																																						
left	0x03	紧靠容器左侧，不改变其大小																																						
right	0x05	紧靠容器右侧，不改变其大小																																						
center_vertical	0x10	垂直居中，不改变其大小																																						
fill_vertical	0x70	垂直方向上拉伸至充满容器																																						
center_horizontal	0x01	水平居中，不改变其大小																																						
fill_horizontal	0x07	水平方向上拉伸使其充满容器																																						
center	0x11	居中对齐，不改变其大小																																						
fill	0x77	在水平和垂直方向上拉伸，使其充满容器																																						
clip_vertical	0x80	垂直剪切(当对象边缘超出容器的时候，将上下边缘超出的部分剪切掉)																																						
clip_horizontal	0x08	水平剪切(当对象边缘超出容器的时候，将左右边缘超出的部分剪切掉)																																						
android:spacing	(译者注：设置图片之间的间距)																																							
android:unselectedAlpha	设置未选中的条目的透明度(Alpha)。该值必须是 float 类型，比如：“1.2”。																																							

公共方法

```
public boolean dispatchKeyEvent(KeyEvent event)
```

在焦点路径上分发按钮事件到下一个视图。该路径从视图树的顶端遍历到当前获得焦点的视图。如果当前视图已获得焦点，就分发给自身。否则，就分发到下一个节点的焦点路径上。该方法监听任何按钮事件。

参数

event 被分发的按钮事件

返回值

boolean 时间被处理返回 true，否则 false

```
public void dispatchSetSelected (boolean selected)
```

分发 setSelected 给视图的子类。

参数

selected 新选中的状态

```
public ViewGroup.LayoutParams generateLayoutParams (AttributeSet attrs)
```

返回一个新的已设置属性集合的布局参数。

参数

attrs 用于生成布局参数的属性集合

返回值

一个 ViewGroup.LayoutParams 实例或者它的子类

```
public boolean onDown (MotionEvent e)
```

当轻击和按下手势事件发生时通知该方法。任何按下事件都会直接触发该方法。所有其他的事件都要先于该方法。

参数

e 按下动作事件

```
public boolean onFling (MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)
```

当初始化的按下动作事件和松开动作事件匹配时通知 fling（译者注：快滑，用户按下触摸屏、快速移动后松开）事件。该动作的速度通过计算 X 和 Y 轴上每秒移动多少像素得来。

参数

e1 导致开始 fling 的按下动作事件。

e2 触发当前 onFling 方法的移动动作事件

velocityX 测量 fling 沿 X 轴上的速度，像素/每秒

velocityY 测量 fling 沿 Y 轴上的速度，像素/每秒

返回值

如果该事件被消耗返回 true，否则 false。

```
public boolean onKeyDown (int keyCode, KeyEvent event)
```

处理左，右和点击事件

参数

keyCode 代表按下按钮的按键码，来自 [KeyEvent](#)。

event 定义按钮动作的 KeyEvent 对象。

返回值

如果已经处理了按钮事件，则返回 true。如果你想让下一个事件接收者处理，

就返回 false

参见

[onKeyDown\(int, KeyEvent\)](#)

public boolean onKeyUp (int keyCode, KeyEvent event)

[KeyEvent.Callback.onKeyMultiple\(\)](#) 方法的默认实现：当 [KEYCODE_DPAD_CENTER](#) 或者 [KEYCODE_ENTER](#) 被释放时，执行点击视图操作。

参数

keyCode 代表按下按钮的按键码，来自 [KeyEvent](#)。

event 定义按钮动作的 [KeyEvent](#) 对象。

返回值

如果已经处理了按钮事件，则返回 true。如果你想让下一个事件接收者处理，就返回 false

public void onLongPress (MotionEvent e)

[MotionEvent](#) 初始化并按下触发长按并通知本方法。

参数

e 导致开始长按的初始按下动作事件。

public boolean onScroll (MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)

当初始按下动作事件和当前移动动作事件导致滚动时通知本方法。为了方便提供了 X 和 Y 轴上的距离。

监听屏幕滚动事件。为了方便提供了 X 和 Y 轴上的距离。

参数

e1 导致滚动开始按下的动作事件。

e2 触发当前 [onScroll](#) 方法的移动动作事件。

distanceX 距离是自上一次调用 [onScroll](#) 方法在 X 轴上的距离。不是 e1 和 e2 之间的距离。

distanceY 距离是自上一次调用 [onScroll](#) 方法在 Y 轴上的距离。不是 e1 和 e2 之间的距离。

返回值

如果该事件被消耗返回 true 否则 false

public void onShowPress (MotionEvent e)

用户已经执行按下动作还没有执行移动或者弹起动作。该事件常通过高亮一个元素来向用户提供一个视觉反馈即用户的操作已经被辨识了。

参数

e 按下动作事件

public boolean onSingleTapUp (MotionEvent e)

在轻击动作和 up 动作事件触发时通知本方法。（译者注：点击屏幕上的某项的执行流程有两种情况，一种是时间很短，一种时间稍长：时间很短：

[onDown-->onSingleTapUp-->onSingleTapConfirmed](#)，见[这里 1](#)，[这里 2](#)。）

参数

e 完成开始轻击的 up 动作事件

返回值

如果该事件被消耗返回 true 否则 false

public boolean onTouchEvent (MotionEvent event)

实现该方法来处理触摸屏动作事件

参数

event 动作事件

返回值

如果该事件被消耗返回 true 否则 false

public void setAnimationDuration (int animationDurationMillis)

设置当子视图改变位置时动画转换时间。仅限于动画开始时生效。

参数

animationDurationMillis 动画转换时间（毫秒级）

public void setCallbackDuringFling (boolean shouldCallback)

当 flinged 时是否回调每一个 [getOnItemSelectedListener\(\)](#)。如果设为 false，只回调最终选中的项。如果为 true，则所有的项都会回调。

参数

shouldCallback 设置抛滑的过程中是否回调

public void setGravity (int gravity)

描述子视图的对齐方式。

public void setSpacing (int spacing)

设置 Gallery 中项的间距

参数

spacing Gallery 中项的间距，以像素为单位

public void setUnselectedAlpha (float unselectedAlpha)

设置 Gallery 中未选中项的透明度(alpha)值。

参数

unselectedAlpha 未选中项的透明度(alpha)值

public boolean showContextMenu ()

显示该视图上下文菜单。

返回值

上下文菜单是否显示。

public boolean showContextMenuForChild (View originalView)

为指定的视图或者其父类显示上下文菜单。

大部分情况下，子类不需要重写该方法。但是，如果子类被直接添加到窗口管理器（例

如: `addView(View.android.view.ViewGroup.LayoutParams)`), 此时就需要重写来显示上下文菜单

参数

`originalView` 上下文菜单初次调用的源视图

返回值

如果上下文菜单被显示了 则返回 `true`。

受保护方法

`protected int computeHorizontalScrollExtent ()`

在水平范围内计算滚动条滑块的滚动范围。该值用来计算滚动条滑块的长度。

该范围可以使用任意的单位但是必须跟 [computeHorizontalScrollRange\(\)](#) 和 [computeHorizontalScrollOffset\(\)](#) 的单位保持一致。

默认范围是视图的宽度。

返回值

滚动条滑块的水平滚动范围

`protected int computeHorizontalScrollOffset ()`

在水平范围内计算滚动条滑块的偏移量。该值用来计算水平滑块的位置。

该范围可以使用任意的单位但是必须跟 [computeHorizontalScrollRange\(\)](#) 和 [computeHorizontalScrollExtent\(\)](#) 的单位保持一致。

默认偏移量是视图的偏移量。

返回值

滚动条滑块的水平偏移量。

`protected int computeHorizontalScrollRange ()`

计算滚动条水平方向上的滚动范围。

该范围可以使用任意的单位但是必须跟 [computeHorizontalScrollExtent\(\)](#) 和 [computeHorizontalScrollOffset\(\)](#) 的单位保持一致。

返回值

水平滚动条代表的滑动总范围。

`protected void dispatchSetPressed (boolean pressed)`

分发 `setPressed` 到 `View` 的子类。

参数

`pressed` 新按下的状态

`protected ViewGroup.LayoutParams generateDefaultLayoutParams ()`

返回默认的布局参数。当 `View` 作为参数传递给 [addView\(View\)](#) 而没有布局参数时就会请求这些参数。如果返回 `null`, 则 `addView` 会抛出异常。

返回值

默认的布局参数或 `null`

`protected ViewGroup.LayoutParams generateLayoutParams (ViewGroup.LayoutParams p)`

返回一组合法的受支持的布局参数。当把 `ViewGroup` 传递给 `View` 而该 `View` 的布局参

数并没有通过 `checkLayoutParams(android.view.ViewGroup.LayoutParams)` 的测试时，就会调用该方法。该方法应该返回一组适合该 `ViewGroup` 的新的布局参数，该过程可能需要从指定的一组布局参数中复制相关的属性。

参数

`p` 被转换成适合该 `ViewGroup` 的一组参数。

返回值

返回一个 `ViewGroup.LayoutParams` 的实例或者一个它的子类。

```
protected int getChildDrawingOrder (int childCount, int i)
```

返回迭代的绘制子类索引。如果你想改变子类的绘制顺序就要重写该方法。默认返回 `i` 值。

提示：为了能够调用该方法，你必须首先调用 [setChildrenDrawingOrderEnabled\(boolean\)](#) 来允许子类排序。

参数

`childCount` 子类个数

`i` 当前迭代顺序

返回值

绘制该迭代子类的索引

```
protected boolean getChildStaticTransformation (View child, Transformation t)
```

(译者注：`setStaticTransformationsEnabled` 这个属性设成 `true` 的时候每次 `viewGroup`(看 `Gallery` 的源码就可以看到它是从 `ViewGroup` 间接继承过来的)在重新画它的 `child` 的时候都会促发 `getChildStaticTransformation` 这个函数。[这里 1](#)、[这里 2](#))

```
protected ContextMenu.ContextMenuItemInfo getContextMenuInfo ()
```

`Views` 如果有额外的信息跟上下文菜单有联系的话就需要实现该方法。返回的结果被用作回调方法 `onCreateContextMenu(ContextMenu, View, ContextMenuItemInfo)` 的参数。

返回值

显示上下文菜单的条目的额外信息。这些信息将会改变 `View` 不同的子类

```
protected void onFocusChanged (boolean gainFocus, int direction, Rect previouslyFocusedRect)
```

当该视图的焦点状态发生改变时将会调用视图系统。当导向的方向触发焦点事件时，方向和先前获得焦点的矩形提供焦点事件的来源。当用户重写该方法，必须调用父类方法来触发标准的焦点处理事件。

参数

`gainFocus` 如果 `View` 获得焦点为 `true`，否则 `false`

`direction` 当调用 `requestFocus()` 方法来给该视图焦点时焦点的移动方向。

该值:`FOCUS_UP`, `FOCUS_DOWN`, `FOCUS_LEFT` 或 `FOCUS_RIGHT`。该参数不常用，通常使用它的默认值。

`previouslyFocusedRect` 该视图坐标系统中先前获得焦点的视图的矩形。如果适用，这将获得焦点事件来源的更细致的信息（除了方向以外）。否则为 `null`。

```
protected void onLayout (boolean changed, int l, int t, int r, int b)
```

当视图为每一个子类分配大小和位置时从布局中调用该方法。有子类的派生类应该重写该方法在子类中调用布局。

参数

changed 该视图新的大小和位置。

l 相对父容器的左侧位置

t 相对父容器的顶部位置

r 相对父容器的右侧位置

b 相对父容器的底部位置

补充

文章精选

[Android 开发——使用 Gallery 实现“多级联动”](#)

[android 图片拖动效果\(Gallery\)](#)

示例代码

MyGallery.java

```
public class MyGallery extends Activity {
    /** Called when the activity is first created. */
    private Gallery gallery;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        gallery = (Gallery) findViewById(R.id.gallery);
        gallery.setAdapter(new ImageAdapter(this)); //设置图片适配器
        //设置监听器
        gallery.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
                Toast.makeText(MyGallery.this, "点击了第" + arg2 + "张图片", Toast.LENGTH_LONG).show();
            }
        });
    }

    class ImageAdapter extends BaseAdapter{
        private Context context;
        //图片源数组
        private Integer[] imageInteger={R.drawable.gallery_photo_1,
```

```

        R.drawable.gallery_photo_2,
        R.drawable.gallery_photo_3,
        R.drawable.gallery_photo_4,
        R.drawable.gallery_photo_5,
        R.drawable.gallery_photo_6,
        R.drawable.gallery_photo_7,
        R.drawable.gallery_photo_8
    };
}

public ImageAdapter(Context c) {
    context = c;
}

@Override
public int getCount() {
    return imageInteger.length;
}

@Override
public Object getItem(int position) {
    return position;
}

@Override
public long getItemId(int position) {
    // TODO Auto-generated method stub
    return position;
}

@Override
public View getView(int position, View convertView,
ViewGroup parent) {
    ImageView imageView = new ImageView(context);
    imageView.setImageResource(imageInteger[position]);
    imageView.setScaleType(ImageView.ScaleType.FIT_XY);
    imageView.setLayoutParams(new
    Gallery.LayoutParams(136, 88));
    return imageView;
}
}

main.xml
<?xml version="1.0" encoding="utf-8"?>
<Gallery
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gallery"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:background="?android:galleryItemBackground"
/>

```

Gallery.LayoutParams

译者署名：我是谁

译者链接：<http://blog.sina.com.cn/u/1744311365>

翻译时间：2010.11.4

版本：Android 2.3 r1

结构

继承关系

public static class Gallery.LayoutParams extends ViewGroup.LayoutParams

```
java.lang.Object  
    android.view.ViewGroup.LayoutParams  
        android.widget.Gallery.LayoutParams
```

类概述

Gallery(画廊)扩展了 LayoutParams，以此提供可以容纳当前的转换信息和先前的位置转换信息的场所。

补充

示例代码

```
1. public View getView(int position, View convertView, ViewGroup parent)  
2. {  
3.     ImageView imageView = new ImageView(mContext);  
4.     // 设置当前图像的图像（position 为当前图像列表的位置）  
5.     imageView.setImageResource(resIds[position]);  
6.     imageView.setScaleType(ImageView.ScaleType.FIT_XY);  
7.     imageView.setLayoutParams(new Gallery.LayoutParams(163, 106));  
8.     // 设置 Gallery 组件的背景风格  
9.     imageView.setBackgroundResource(mGalleryItemBackground);  
10.    return imageView;  
11. }
```

GridView

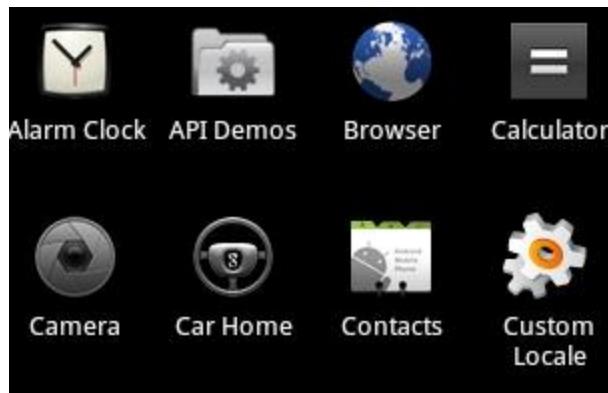
译者署名: 0_1
版本: Android 2.2 r1

public final class GridView extends AbsListView

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.AdapterView<T extends android.widget.Adapter>
                android.widget.AbsListView
                    android.widget.GridView
```

类概述

一个在平面上可显示多个条目的可滚动的视图组件，该组件中的条目通过一个ListAdapter 和该组件进行关联。比如 android 手机中显示的应用：



比如实现九宫格图，用 GridView 是首选，也是最简单的。

构造函数

```
public GridView (Context context)
创建一个默认属性的 GridView 实例

public GridView (Context context, AttributeSet attrs)
创建一个带有 attrs 属性的 GridView 实例

public GridView (Context context, AttributeSet attrs, int defStyle)
创建一个带有 attrs 属性，并且指定其默认样式的 GridView 实例
```

XML 属性

属性名称	描述
android:columnWidth	设置列的宽度。关联的方法为: setColumnWidth(int)
android:gravity	设置此组件中的内容在组件中的位置。可选的值有: top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical 可以多选，用“ ”分开。关联方

	法: <code>setGravity (int gravity)</code>
<code>android:horizontalSpacing</code>	两列之间的间距。关联方法: <code>setHorizontalSpacing(int)</code>
<code>android:numColumns</code>	列数。关联方法: <code>setNumColumns(int)</code>
<code>android:stretchMode</code>	缩放模式。关联方法: <code>setStretchMode(int)</code>
<code>android:verticalSpacing</code>	两行之间的间距。关联方法: <code>setVerticalSpacing(int)</code>

公共方法

`public ListAdapter getAdapter ()`

获得与此组件相关的适配器..

返回值

`ListAdapter` 适配器实例

`public int getStretchMode ()`

获得 `GridView` 的缩放模式..

`public boolean onKeyDown (int keyCode, KeyEvent event)`

默认由 `KeyEvent.Callback.onKeyMultiple ()` 实现, 如果视图是可用的并且是可点击的, 那么传入 `KEYCODE_DPAD_CENTER` 或 `KEYCODE_ENTER` 值是执行的是按下视图操作。

参数

`keyCode` 一个表示按下操作的键值.

`event` 表示按钮事件的对象.

返回值

如果你认为已经完成事件处理, 不想让下一个处理器来处理此事件, 则返回 `true`, 否则返回 `false`.

`public boolean onKeyMultiple (int keyCode, int repeatCount, KeyEvent event)`

默认由 `KeyEvent.Callback.onKeyMultiple()` 实现, 总是返回 `false` (不处理此事件)。

参数

`keyCode` 键值.

`repeatCount` 该动作发生的次数.

`event` 事件对象.

返回值

如果你认为已经完成事件处理, 不想让下一个处理器来处理此事件, 则返回 `true`, 否则返回 `false`.

`public boolean onKeyUp (int keyCode, KeyEvent event)`

默认由 `KeyEvent.Callback.onKeyMultiple ()` 实现, 如果视图是可用的并且是可点击的, 那么传入 `KEYCODE_DPAD_CENTER` 或 `KEYCODE_ENTER` 值是执行的是点击视图操作。

参数

`keyCode` 键值.

`event` 事件对象.

返回值

如果你认为已经完成事件处理,不想让下一个处理器来处理此事件, 则返回 true, 否则返回 false。

public void setAdapter (ListAdapter adapter)

设置 GridView 的数据。

参数

adapter 为 grid 提供数据的适配器

public void setColumnWidth (int columnWidth)

设置 GridView 的列宽.

参数

columnWidth 列的宽度, 以像素为单位

public void setGravity (int gravity)

设置控件内容的位置, 默认值为: Gravity.LEFT.

参数

gravity 位置值

public void setHorizontalSpacing (int horizontalSpacing)

设置列间距.

参数

horizontalSpacing 列间距值

public void setNumColumns (int numColumns)

设置 grid 的列数

参数

numColumns 列数值.

public void setSelection (int position)

设置选中的条目.

参数

position 数据条目在列表中的索引值 (从 0 开始), 如果在可触摸的模式下, 在该索引值下的条目将不会被选中, 但是该索引值仍然指向该条目。

public void setStretchMode (int stretchMode)

设置 grid 中的条目以什么缩放模式去填充空间。.

参数

stretchMode 可选值 : NO_STRETCH , STRETCH_SPACING , STRETCH_SPACING_UNIFORM, 或 STRETCH_COLUMN_WIDTH

public void setVerticalSpacing (int verticalSpacing)

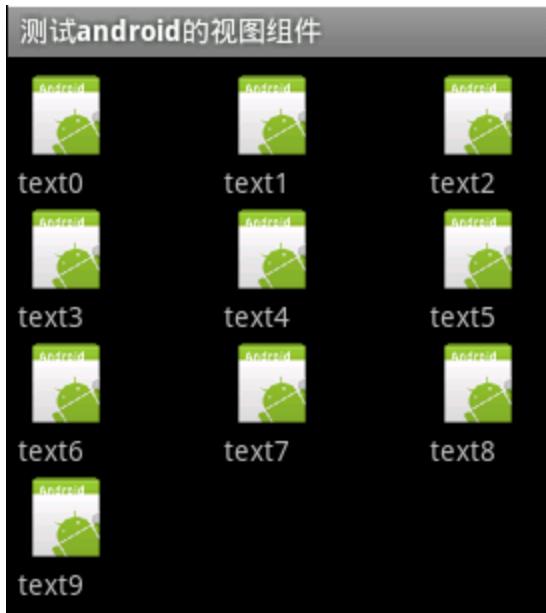
设置行间距.

参数

verticalSpacing 间距值，以像素为单位..

示例：

下面给出一个小例子，先看效果：



代码：

1、 GridView01.java

```
public class GridView01 extends Activity {
    private GridView gridview;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gridview);

        //准备要添加的数据条目

        List<Map<String, Object>> items = new
        ArrayList<Map<String, Object>>();
        for (int i = 0; i < 10; i++) {
            Map<String, Object> item = new HashMap<String,
            Object>();
            item.put("imageItem", R.drawable.icon);
            item.put("textItem", "text" + i);
            items.add(item);
        }

        //实例化一个适配器

        SimpleAdapter adapter = new SimpleAdapter(this, items,
        R.layout.grid_item, new String[]{"imageItem", "textItem"},
```

```

new int[] {R.id.image_item, R.id.text_item});  

//获得GridView实例  

gridview = (GridView) findViewById(R.id.mygridview);  

//gridview.setNumColumns(3); //可以在xml中设置  

//gridview.setGravity(Gravity.CENTER); //同上  

//将GridView和数据适配器关联  

gridview.setAdapter(adapter);  

}

```

2、 gridview.xml

```

<?xml version="1.0" encoding="utf-8"?>  

<LinearLayout  

    xmlns:android="http://schemas.android.com/apk/res/android"  

        android:layout_width="wrap_content"  

        android:layout_height="wrap_content">  

    <GridView android:id="@+id/mygridview"  

        android:numColumns="3"  

        android:gravity="center_horizontal"  

        android:layout_width="wrap_content"  

        android:layout_height="wrap_content">  

    </GridView>  

</LinearLayout>

```

3、 grid_item.xml

```

<?xml version="1.0" encoding="utf-8"?>  

<RelativeLayout android:id="@+id/RelativeLayout01"  

    android:layout_width="fill_parent"  

    android:layout_height="fill_parent"  

    xmlns:android="http://schemas.android.com/apk/res/android">  

    <ImageView android:id="@+id/image_item"  

        android:layout_width="wrap_content"  

        android:layout_height="wrap_content">  

    </ImageView>  

    <TextView android:id="@+id/text_item"  

        android:layout_below="@+id/image_item"  

        android:layout_height="wrap_content"  

        android:layout_width="wrap_content">  

    </TextView>  

</RelativeLayout>

```

第一次翻译文档，翻译的不太好，请大家多多包涵。

RadioGroup

译者署名：首当其冲
版本：Android 2.2 r1

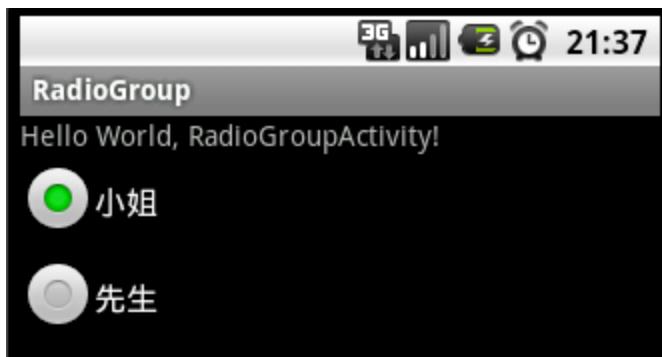
结构

继承关系

public class RadioGroup extends LinearLayout

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.LinearLayout
                android.widget.RadioGroup
```

类概述



这个类用于创建一组按钮之间相互排斥的单选按钮组，在同一个单选按钮组中勾选一个按钮则会取消该组中其它已经勾选的按钮的选中状态。

初始状态下，所有的单选按钮都未勾选，虽然不能取消一个特定的单选按钮的勾选状态，但可以通过单选按钮组去消除它的勾选状态，根据 XML 布局文件中的单选按钮的唯一 ID 去标识指定的选择信息。

内部类

Class **RadioGroup.LayoutParams**

当 WRAP_CONTENT（包裹内容）的子组件在 XML 文件中没有指定相应的宽度和高度的话，可以使用的布局参数默认设置的默认设置

Interface **RadioGroup.OnCheckedChangeListener**

当单选按钮组中的单选按钮的勾选状态发生改变时，所要调用的回调函数的接口类

公共方法

public void addView (View child, int index, ViewGroup.LayoutParams params)

使用指定的布局参数添加一个子视图

参数

child 所要添加的子视图

index 将要添加子视图的位置
params 所要添加的子视图的布局参数

public void check (int id)

如果传递-1 作为指定的选择标识符来清除单选按钮组的勾选状态，相当于调用 clearCheck() 操作

参数

id 该组中所要勾选的单选按钮的唯一标识符 (id)

参见

[getCheckedRadioButtonId\(\)](#)

[clearCheck\(\)](#)

public void clearCheck ()

清除当前的选择状态，当选择状态被清除，则单选按钮组里面的所有单选按钮将取消勾选状态，[getCheckedRadioButtonId\(\)](#)将返回 null

参见

[check\(int\)](#)

[getCheckedRadioButtonId\(\)](#)

public RadioGroup.LayoutParams generateLayoutParams (AttributeSet attrs)

基于提供的属性集合返回一个新的布局参数集合

参数

attrs 用于生成布局参数的属性

返回值

返回一个 [ViewGroup.LayoutParams](#) 或其子类的实例

public int getCheckedRadioButtonId ()

返回该单选按钮组中所选择的单选按钮的标识 ID，如果没有勾选则返回-1

返回值

返回该单选按钮组中所选择的单选按钮的标识 ID

参见

[check\(int\)](#)

[clearCheck\(\)](#)

public void setOnCheckedChangeListener (RadioGroup.OnCheckedChangeListener listener)

注册一个当该单选按钮组中的单选按钮勾选状态发生改变时所要调用的回调函数

参数

listener 当单选按钮勾选状态发生改变时所要调用的回调函数

public void setOnHierarchyChangeListener (ViewGroup.OnHierarchyChangeListener listener)

注册一个当子内容添加到该视图或者从该视图中移除时所要调用的回调函数

参数

listener 当层次结构发生改变时所要调用的回调函数

受保护方法

`protected LinearLayout.LayoutParams generateDefaultLayoutParams ()`

当布局为垂直方向时，将返回一个宽度为“填充父元素”(MATCH_PARENT)，高度为“包裹内容”的布局参数集合，如果为水平方向时，将返回宽度为“包裹内容”，高度为“填充父元素”的布局参数集合

(match_parent 即为 fill_parent,public static final int FILL_PARENT/MATCH_PARENT = -1)

返回值

返回一个默认的布局参数集合

`protected void onFinishInflate ()`

当视图从 XML 中加载，且相应的子视图被添加之后，调用该方法，

即使子类重写了该方法，应该确保去调用父类的方法（通常放在方法在第一句），这样才能完成相应的调用参数

返回值

返回一个默认的布局参数集合

补充

文章链接

[Android 基础教程\(七\)之----单选项框 RadioGroup 的综合应用](#)

[Android UI 设计 RadioGroup 单选按钮用法](#)

[Android 小项目之--猜名字有奖！RadioButton 和 RadioGroup 应用（附源码）](#)

示例代码

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="fill_parent">
<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/girl"
    android:text="小姐"/>
<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boy"
    android:text="先生"/>
</RadioGroup>
</LinearLayout>
```

ImageView

农民伯伯

版本: Android 2.2

```
java.lang.Object  
    android.view.View  
        android.widget.ImageView
```

直接子类:

ImageButton, QuickContactBadge

间接子类:

ZoomButton

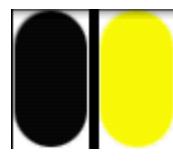
类概述:

显示任意图像，例如图标。ImageView 类可以加载各种来源的图片（如资源或图片库），需要计算图像的尺寸，以便它可以在其他布局中使用，并提供例如缩放和着色（渲染）各种显示选项。

XML 属性

属性名称	描述											
android:adjustViewBounds	是否保持宽高比。需要与 maxWidth、MaxHeight 一起使用，否则单独使用没有效果。											
android:cropToPadding	是否截取指定区域用空白代替。单独设置无效果，需要与 scrollY 一起使用，效果如下，实现代码见代码部分： 											
android:maxHeight	设置 View 的最大高度，单独使用无效，需要与 setAdjustViewBounds 一起使用。如果想设置图片固定大小，又想保持图片宽高比，需要如下设置： 1) 设置 setAdjustViewBounds 为 true; 2) 设置 maxWidth、MaxHeight; 3) 设置设置 layout_width 和 layout_height 为 wrap_content。											
android:maxWidth	设置 View 的最大宽度。同上。											
android:scaleType	设置图片的填充方式。 <table border="1"><tr><td>matrix</td><td>0</td><td>用矩阵来绘图</td><td></td></tr><tr><td>fitXY</td><td>1</td><td>拉伸图片(不按比例) 以填充 View 的宽高 </td><td>layout_height: 30px</td></tr></table>				matrix	0	用矩阵来绘图		fitXY	1	拉伸图片(不按比例) 以填充 View 的宽高 	layout_height: 30px
matrix	0	用矩阵来绘图										
fitXY	1	拉伸图片(不按比例) 以填充 View 的宽高 	layout_height: 30px									

	fitStart	2	按比例拉伸图片，拉伸后图片的高度为 View 的高度，且显示在 View 的左边 	layout_width: 120px
	fitCenter	3	按比例拉伸图片，拉伸后图片的高度为 View 的高度，且显示在 View 的中间 	
	fitEnd	4	按比例拉伸图片，拉伸后图片的高度为 View 的高度，且显示在 View 的右边 	
	center	5	按原图大小显示图片，但图片宽高大于 View 的宽高时，截图图片中间部分显示 	layout_height: 60px layout_width: 80px padding: 10px
	centerCrop	6	按比例放大原图直至等于某边 View 的宽高显示。 	
	centerInside	7	当原图宽高或等于 View 的宽高时，按原图大小居中显示；反之将原图缩放至 View 的宽高居中显示。 	
android:src	设置 View 的 drawable(如图片，也可以是颜色，但是需要指定 View 的大小)			
android:tint	将图片渲染成指定的颜色。见下图：			



左边为原图，右边为设置后的效果，见后面代码。

代码

```
android:tint
<ImageView android:background="@android:color/white"
    android:src="@drawable	btn_mode_switch_bg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ImageView>
<ImageView android:layout_marginLeft="5dp"
    android:background="@android:color/white" android:tint="#ffff00"
    android:src="@drawable	btn_mode_switch_bg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ImageView>

cropToPadding
<ImageView android:background="@android:color/white"
    android:scrollY="-10px" android:cropToPadding="true"
    android:src="@drawable	btn_mode_switch_bg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ImageView>
<ImageView android:background="@android:color/white"
    android:scrollY="10px" android:cropToPadding="true"
    android:src="@drawable	btn_mode_switch_bg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ImageView>
<ImageView android:paddingTop="10px"
    android:background="@android:color/white" android:scrollY="10px"
    android:cropToPadding="true"
    android:src="@drawable	btn_mode_switch_bg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ImageView>
<ImageView android:paddingTop="10px"
    android:background="@android:color/white" android:scrollY="10px"
    android:cropToPadding="false"
    android:src="@drawable	btn_mode_switch_bg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ImageView>
```

HorizontalScrollView

译者署名: Tina
版本: Android 2.3 r1

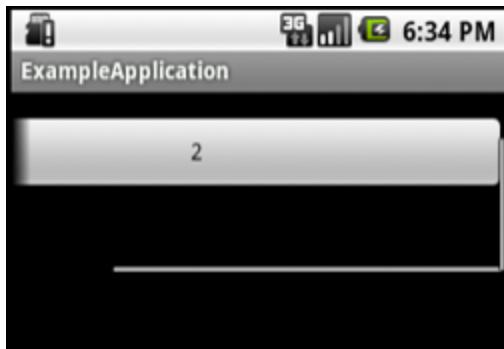
结构

继承关系

`public class HorizontalScrollView extends FrameLayout`

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.FrameLayout
                android.widget.HorizontalScrollView
```

类概述



用于布局的容器，可以放置让用户使用滚动条查看的视图层次结构，允许视图结构比手机的屏幕大。HorizontalScrollView 是一种 [FrameLayout](#)（框架布局），其子项被滚动查看时是整体移动的，并且子项本身可以是一个有复杂层次结构的布局管理器。一个常见的应用是子项在水平方向中，用户可以滚动显示顶层水平排列的子项(items)。

HorizontalScrollView 不可以和 ListView 同时用，因为 ListView 有自己的滚动条设置。最重要的是，如果在需要显示很大的 list 的情况下，两者同时用则会使 ListView 在一些重要的优化上失效。出现这种失效的原因在于，HorizontalScrollView 会强迫 ListView 用 HorizontalScrollView 本身提供的空间容器（infinite container）来显示完整的列表。

类似的情况，[TextView](#) 也有自己的滚动条，所以不需要 ScrollView。但这两者是可以同时使用的，使用的结果会是在一个更大的容器里显示文本视图。

HorizontalScrollView 只支持水平方向的滚动显示。

公共方法

`public void addView (View child)`

添加一个子视图。若这个子视图没有被设置布局参数，则使用 ViewGroup 的缺省参数。

参数

`child` 要添加的子视图

`public void addView (View child, int index)`

添加一个子视图。若这个子视图没有被设置布局参数，则使用 ViewGroup 的缺省参数。

参数

- child 要添加的子视图
- index 子视图要加入的位置

public void addView (View child, int index, ViewGroup.LayoutParams params)

添加一个带有指定布局参数的子视图。

参数

- child 要添加的子视图
- index 子视图要加入的位置
- params 子视图的布局参数

public void addView (View child, ViewGroup.LayoutParams params)

添加一个带有指定布局参数的子视图。

参数

- child 要添加的子视图
- params 子视图的布局参数

public boolean arrowScroll (int direction)

响应点击左右箭头时对滚动条的处理。

参数

- direction The direction corresponding to the arrow key that was pressed 箭头按键所表示的方向

返回值

若此事件成功完成，则返回 true；否则返回 false。

public void computeScroll ()

被父视图调用，用于必要时候对其子视图的值（mScrollX 和 mScrollY）进行更新。典型的情况如：父视图中某个子视图使用一个 [Scroller](#) 对象来实现滚动操作，会使得此方法被调用。

public boolean dispatchKeyEvent (KeyEvent event)

按照可以获得焦点的顺序（从视图树的顶端到当前获得焦点的视图），分派一个按键事件给下一个视图。若此视图为焦点视图，事件将会分派给它自己。否则它将按照顺序，分派给下一个节点。此方法同时触动所有按键监听器。

参数

- event 被分派的事件.

返回值

若事件被处理，则返回 true；否则为 false

public void draw (Canvas canvas)

手动绘制视图（及其子视图）到指定的画布(Canvas)。这个视图必须在调用这个函数之前做好了整体布局。。当实现一个视图时，不需要继承这个方法；相反，你应该实现 [onDraw\(Canvas\)](#) 方法。

参数

canvas 绘制视图的画布

public boolean executeKeyEvent (KeyEvent event)

需要通过按键事件来实现滚动操作时，可以调用此方法。效果类似于由视图树型结构分派事件。

参数

event 需要执行的事件

返回值

若事件被处理，则返回 true；否则为 false

public void fling (int velocityX)

滚动视图的 fling 手势。

参数

velocityX 方向的初始速率。正值表示手指/光标向屏幕右边滑动，而内容相对向左滚动。

public boolean fullScroll (int direction)

处理按下"home/end"快捷键之后的滚动响应。此方法会将视图移左或移右，同时将焦点赋予移动后可视的最左或最右的组件。如果没有任何组件适合得到焦点，此 scrollview 将收回焦点。

参数

direction 滚动方向：FOCUS_LEFT 表示向视图的左边移动，FOCUS_RIGHT 表示向视图的右边移动

返回值

若此方法消耗(consumed)了按键事件则返回 true，否则返回 false。

public int getMaxScrollAmount ()

返回值

按左右箭头时视图可以滚动的最大值。

public boolean isFillViewport ()

表示此 ScrollView 的内容是否被拉伸以适应视口（viewport）的大小。

返回值

若内容填充了视口则返回 true，否则返回 false。

public boolean isSmoothScrollingEnabled ()

返回值

按箭头方向滚动时，是否显示滚动的平滑效果。

public boolean onInterceptTouchEvent (MotionEvent ev)

使用此方法可以拦截所有触摸屏动作引发的事件。这意味着你可以监视分派给子项的事件，并且可以拿到任何当前手势的所有权。

使用此方法需谨慎。因为它与 View.onTouchEvent(MotionEvent)有相当复杂的交互影响。

这两者都必须同时正确地实现。事件将按以下顺序来被方法接收：

1. 接收到 down 事件
2. 事件将被视图组的一个子视图处理，或者被传递给自己的 `onTouchEvent()` 方法处理；这意味着你必须实现 `onTouchEvent()`，并且返回 `true`，这样才可以接着接受到其他的手势（而不是寻求一个父视图来处理它）。`onTouchEvent()` 返回 `true` 后，你将不再接受到 `onInterceptTouchEvent()` 的任何事件，同时所有对触摸动作的处理必须像往常一样在 `onTouchEvent()` 中进行。
3. 如果返回 `false`，则接下来的每个事件（所有的 up 事件，包含最后一个 up）将会首先被传递到这里，然后到目标对象 `view` 的 `onTouchEvent()`。
4. 如果返回 `true`，你将不会接收到以下任何事件：目标 `view` 将会接收到相同的事件，但是带着 `ACTION_CANCEL` 的动作。所有在此之后的事件将会被传递到你的 `onTouchEvent()` 方法中，并且不再在这里出现。

参数

`ev` 沿着树型结构往下分派的动作事件

返回值

若将动作事件从子视图中截获并通过 `onTouchEvent()` 将他们分派给当前 `ViewGroup`，则返回 `true`。当前目标将收到一个 `ACTION_CANCEL` 事件，并且不再会有其他消息被传递到这里。

public boolean onTouchEvent (MotionEvent ev)

此方法用于处理触摸屏的动作事件。

参数

`ev` 动作事件

返回值

若事件被成功处理，则返回 `true`；否则返回 `false`

public boolean pageScroll (int direction)

处理按下"page up/down"快捷键之后的滚动响应。此方法会将视图往左或往右滚动一个页面的距离，同时将焦点赋予移动后可视的最左或最右的组件。如果没有任何组件适合得到焦点，此 `scrollview` 将收回焦点。

参数

`direction` 滚动方向：`FOCUS_LEFT` 表示向视图的左边移动一个页面
`FOCUS_RIGHT` 表示向视图的右边移动一个页面

返回值

若此方法处理(`consumed`)了按键事件则返回 `true`，否则返回 `false`。

public void requestChildFocus (View child, View focused)

当父视图的一个子视图要获得焦点时，调用此方法。

参数

`child` 要获得焦点的子视图。此视图将包含焦点视图，但其本身不必为焦点。
`focused` 事实上拥有焦点的子视图的下层视图。

public boolean requestChildRectangleOnScreen (View child, Rect rectangle, boolean immediate)

当组里的某个子视图需要被定位在屏幕的某个矩形范围时，调用此方法。重载此方法的 `ViewGroup` 可确认以下几点：

- * 子项目将是组里的直系子项
- * 矩形将在子项目的坐标体系中

重载此方法的 `ViewGroup` 应该支持以下几点：

- * 若矩形已经是可见的，则没有东西会改变
- * 为使矩形区域全部可见，视图将可以被滚动显示

参数

`child` 发出请求的子视图

`rectangle` 子项目坐标系内的矩形，即此子项目希望在屏幕上的定位

`immediate` 设为 `true`，则禁止动画和平滑移动滚动条

返回值

进行了滚动操作的这个组（`group`），是否处理此操作

`public void requestLayout ()`

当出现使视图布局失效的改变时，调用此方法。它将规划一个视图树的布局路径。

`public void scrollTo (int x, int y)`

设置视图滚动后的位置。这将引起 `onScrollChanged(int,int,int,int)` 的调用，同时使此视图失效。

此版本同时将滚动锁定于子视图的范围。

参数

`x` 要滚动到的 x 位置

`y` 要滚动到的 y 位置

`public void setFillViewport (boolean fillViewport)`

设置此滚动视图是否将内容宽度拉伸来适应视口（`viewport`）。

参数

`fillViewport` 设置为 `true` 表示将拉伸内容宽度；否则会设置为 `false`。

`public void setOverScrollMode (int mode)`

为视图设置 over-scroll 模式。有效的 over-scroll 模式有 [OVER_SCROLL_ALWAYS](#)（缺省值），[OVER_SCROLL_IF_CONTENT_SCROLLS](#)（只允许当视图内容过大时，进行 over-scrolling）和 [OVER_SCROLL_NEVER](#)。只有当视图可以滚动时，此项设置才起作用。

参数

`mode` 视图的新 over-scroll 模式值

`public void setSmoothScrollingEnabled (boolean smoothScrollingEnabled)`

设置是否呈现按下箭头后的平滑滚动效果（动画效果）。

参数

`smoothScrollingEnabled` 设置是否呈现平滑滚动效果

`public final void smoothScrollBy (int dx, int dy)`

类似 [scrollBy\(int, int\)](#)，但是呈现平滑滚动，而非瞬间滚动（译者注：瞬间滚动——指不

显示滚动过程，直接显示滚动后达到的位置)。

参数

- dx 要滚动的 X 轴像素差值 (译者注：横向像素差值)
- dy 要滚动的 Y 轴像素差值 (译者注：纵向像素差值)

public final void smoothScrollTo (int x, int y)

类似 [scrollTo\(int, int\)](#)，但是呈现平滑滚动，而不是瞬间滚动。

参数

- x 滚动要到达位置的 X 轴值
- y 滚动要到达位置的 Y 轴值

受保护方法

protected int computeHorizontalScrollOffset ()

计算水平方向滚动条的滑块的偏移值。此值用来计算滚动时滑块的位置。

偏移值的范围可以以任何单位表示，但必须与 `computeHorizontalScrollRange()` 和 `computeHorizontalScrollExtent()` 的单位一致。

缺省的偏移值为视图滚动的偏移差值。

返回值

滚动条滑块在水平方向上的偏移值

protected int computeHorizontalScrollRange ()

`scroll view` 的可滚动水平范围是所有子视图的宽度总合。

返回值

水平滚动条表示的全部水平滚动范围

protected int computeScrollDeltaToGetChildRectOnScreen (Rect rect)

计算 X 方向滚动的总合，以便在屏幕上显示子视图的完整矩形 (或者，若矩形宽度超过屏幕宽度，至少要填满第一个屏幕大小)。

参数

- rect 矩形

返回值

滚动差值

protected float getLeftFadingEdgeStrength ()

返回左渐变边缘的强度或密集度。强度的值介于 0.0 (无渐变) 到 1.0 (全渐变) 之间。缺省实现只返回 0.0 或 1.0，而不返回中间值。子类必须重载此方法来给滚动动作提供更平滑的渐变过程。

返回值

左渐变的强度，即介于 0.0f 和 1.0f 之间的浮点值

protected float getRightFadingEdgeStrength ()

返回右渐变边缘的强度或密集度。强度的值介于 0.0 (无渐变) 到 1.0 (全渐变) 之间。缺省实现只返回 0.0 或 1.0，而不返回中间值。子类必须重载此方法来给滚动动作提供更平滑的渐变过程。

返回值

右渐变的强度，即介于 0.0f 和 1.0f 之间的浮点值

```
protected void measureChild (View child, int parentWidthMeasureSpec, int  
parentHeightMeasureSpec)
```

要求子视图测量自身，需要将视图的 MeasureSpec 和其附加内容同时考虑在内。

getChildMeasureSpec 在其中承担了重要角色，它计算出 MeasureSpec，并传递给子视图。

参数

child 要测量的子视图
parentWidthMeasureSpec 此视图的宽度要求
parentHeightMeasureSpec 此视图的高度要求

```
protected void measureChildWithMargins (View child, int parentWidthMeasureSpec, int  
widthUsed, int parentHeightMeasureSpec, int heightUsed)
```

要求子视图测量自身，需要将视图的 MeasureSpec、附加内容和边缘部分同时考虑在内。子项必须有 MarginLayoutParams（边缘布局参数）。getChildMeasureSpec 在其中承担了重要角色，它计算出 MeasureSpec，并传递给子视图。

参数

child 要测量的子视图
parentWidthMeasureSpec 此视图的宽度要求
widthUsed 被父视图（也可能是其他子视图）占用的横向额外空间
parentHeightMeasureSpec 此视图的高度要求
heightUsed 被父视图（也可能是其他子视图）占用的纵向额外空间

```
protected void onLayout (boolean changed, int l, int t, int r, int b)
```

当此视图要给每个子视图赋值大小和位置时，layout 会调用此方法。子项的派生类应当重载此方法，并且调用各个子项的 layout。

参数

changed 此视图有新的大小或位置
l 左边界位置，相对于父视图
t 上边界位置，相对于父视图
r 右边界位置，相对于父视图
b 下边界位置，相对于父视图

```
protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)
```

调用此方法来确定本身和所包含内容的大小（宽度和高度）。此方法被 measure(int,int) 唤起，而且必须被子类重载以得到所包含内容的确切大小。

注意：当重载此方法时，必须调用 setMeasureDimension(int,int) 来保存 View 的大小。如果没有做到，将会引发一个 measure(int,int) 抛出的 IllegalStateException（非法状态错误）。超类 onMeasure(int,int) 可以被调用。

编写基类的确认大小的方法，缺省情况下是根据其背景大小来确认，除非 MeasureSpec 允许有更大的高度或宽度。子类必须重载 onMeasure(int,int) 以得到对其内容大小的更准确的测量。

若此方法被重载，它的子类需要确保其高度和宽度至少达到 View 所规定的最小值（可

通过 `getSuggestedMinimumHeight()` 和 `getSuggestedMinimumWidth()` 得到)。

参数

- | | |
|--------------------------------|---|
| <code>widthMeasureSpec</code> | 受上一层大小影响下的对水平空间的要求。可参看
<code>View.MeasureSpec</code> 。 |
| <code>heightMeasureSpec</code> | 受上一层大小影响下的对垂直空间的要求。可参看
<code>View.MeasureSpec</code> 。 |

`protected void onOverScrolled (int scrollX, int scrollY, boolean clampedX, boolean clampedY)`

被 `overScrollBy(int, int, int, int, int, int, int, int, boolean)` 调用，来对一个 over-scroll 操作的结果进行响应。

参数

- | | |
|-----------------------|---|
| <code>scrollX</code> | 新的 X 滚动像素值 |
| <code>scrollY</code> | 新的 Y 滚动像素值 |
| <code>clampedX</code> | 当 <code>scrollX</code> 被 over-scroll 的边界限制时，值为 true |
| <code>clampedY</code> | 当 <code>scrollY</code> 被 over-scroll 的边界限制时，值为 true |

`protected boolean onRequestFocusInDescendants (int direction, Rect`

`previouslyFocusedRect)`

当在某个 scroll view 的子视图中寻找焦点时，需要小心不能让屏幕之外的组件得到焦点。这比缺省 ViewGroup 的实现代价更高，否则此行为被设为缺省。

参数

- | | |
|------------------------------------|---|
| <code>direction</code> | 值可以为 <code>FOCUS_UP</code> , <code>FOCUS_DOWN</code> , <code>FOCUS_LEFT</code> 或 <code>FOCUS_RIGHT</code> |
| <code>previouslyFocusedRect</code> | 能够给出一个较好的提示的矩形（当前视图的坐标系统）表示焦点从哪里得来。如果没有提示则为 null。 |

返回值

是否取到了焦点。

`protected void onSizeChanged (int w, int h, int oldw, int oldh)`

当 View 的大小改变时此方法被调用。如果 View 是刚刚被加入，则视之前的值为 0。

参数

- | | |
|-------------------|----------------|
| <code>w</code> | View 的当前宽度 |
| <code>h</code> | View 的当前高度 |
| <code>oldw</code> | View 大小改变之前的宽度 |
| <code>oldh</code> | View 大小改变之前的高度 |

补充

文章精选

[Android HorizontalScrollView Example](#)

ImageButton

农民伯伯

版本: Android 2.2

```
java.lang.Object  
    android.view.View  
        android.widget.ImageView  
            android.widget.ImageButton
```

直接子类:

ZoomButton

类摘要:

显示一个可以被用户点击的图片按钮， 默认情况下， ImageButton 看起来像一个普通的按钮，在不同状态（如按下）下改变背景颜色。按钮的图片可用通过<ImageButton> XML 元素的 android:src 属性或 setImageResource(int)方法指定。

要删除按钮的背景， 可以定义自己的背景图片或设置背景为透明。（注：请看



原图



和图片按钮， 默认图片周围有按钮的背景，选中之后为黄色）

为了表示不同的按钮状态（焦点，选择等），你可以为各种状态定义不同的图片。例如，定义蓝色图片为默认图片，黄色图片为获取时焦点时显示的图片，黄色图片为按钮被按下时显示的图片。一个简单的方法可以做到这点——通过 XML 的"selector."配置，如下：

```
<?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:drawable="@drawable/button_normal" /> <!-- default -->  
</selector>
```

保存上面的 XML 到 `res/drawable/`文件夹下（注：注意文件名大小写！），将该文件名作为一个参数设置到 ImageButton 的 android:src 属性（注：如 xml 文件名为 myselector.xml，那么这里设置为"="@drawable/myselector"，设置 android:background 也是可以的，但效果不太一样）。Android 根据按钮的状态改变会自动的去 XML 中查找相应的图片以显示。

`<item>`元素的顺序很重要，因为是根据这个顺序判断是否适用于当前按钮状态，这也是为什么正常（默认）状态指定的图片放在最后，是因为它只会在 `pressed` 和 `focused` 都判断失败之后才会被采用。（注：例如按钮被按下时是同时获得焦点的，但是获得焦点并不一定按了按钮，所以这里会按顺序查找，找到合适的就不往下找了。这里按钮被点击了，那么第一个将被选中，且不再在后面查找其他状态。）

参见 [Form Stuff tutorial](#)。

继承自父类的方法

`public void setAlpha (int alpha)`

设置 ImageButton 图片的透明度（注意不是背景图片的）。效果如图：



参数

`alpha` 透明值 0~255, 0 为完全透明, 255 为完全不透明

ImageSwitcher

翻译人: wallace2010

译者博客: <http://blog.csdn.net/springiscoming2008>

版本: Android 2.2 r1

public class ImageSwitcher extends ViewSwitcher

java.lang.Object

[android.view.View](#)

 android.view.ViewGroup

 android.widget.FrameLayout

 android.widget.ViewAnimator

 android.widget.ViewSwitcher

 android.widget.ImageSwitcher

类概述



(译者注:ImageSwitcher 是 Android 中控制图片展示效果的一个控件,如:幻灯片效果...,颇有感觉啊, 做相册一绝。)

公共方法

```
public void setImageDrawable (Drawable drawable)
```

绘制图片

```
public void setImageResource (int resid)
```

设置图片资源库

```
public void setImageURI (Uri uri)
```

设置图片地址

补充

文章链接

[Android ImageSwitcher](#)

[Image Switcher View | Android Developer Tutorial](#)

示例代码（本文代码转载自[这里](#)）

java 文件

```
public class mainactivity extends Activity implements  
OnItemSelectedListener, ViewFactory {  
  
private ImageSwitcher is;  
private Gallery gallery;  
  
private Integer[] mThumbIds = { R.drawable.b, R.drawable.c,  
R.drawable.d, R.drawable.f, R.drawable.g,  
};  
  
private Integer[] mImageIds = { R.drawable.b, R.drawable.c,  
R.drawable.d, R.drawable.f, R.drawable.g, };  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState);  
requestWindowFeature(Window.FEATURE_NO_TITLE);  
setContentView(R.layout.main);  
  
is = (ImageSwitcher) findViewById(R.id.switcher);  
is.setFactory(this);  
  
is.setInAnimation(AnimationUtils.loadAnimation(this,  
android.R.anim.fade_in));  
is.setOutAnimation(AnimationUtils.loadAnimation(this,  
android.R.anim.fade_out));  
  
gallery = (Gallery) findViewById(R.id.gallery);
```

```
    gallery.setAdapter(new ImageAdapter(this));
    gallery.setOnItemSelectedListener(this);
}

@Override
public View makeView() {
    ImageView i = new ImageView(this);
    i.setBackgroundColor(0xFF000000);
    i.setScaleType(ImageView.ScaleType.FIT_CENTER);
    i.setLayoutParams(new ImageSwitcher.LayoutParams(
        LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
    return i;
}

public class ImageAdapter extends BaseAdapter {
    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return position;
    }

    public long getItemId(int position) {
        return position;
    }

    public View getView(int position, View convertView, ViewGroup parent)
    {
        ImageView i = new ImageView(mContext);

        i.setImageResource(mThumbIds[position]);
        i.setAdjustViewBounds(true);
        i.setLayoutParams(new Gallery.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
        i.setBackgroundDrawable(R.drawable.e);
        return i;
    }

    private Context mContext;
```

```
    }

@Override
public void onItemSelected(AdapterView<?> parent, View view, int
position,
long id) {
    is.setImageResource(mImageIds[position]);
}

}

@Override
public void onNothingSelected(AdapterView<?> parent) {
}

}

xml 文件
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageSwitcher android:id="@+id/switcher"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
    />

    <Gallery android:id="@+id/gallery"
        android:background="#55000000"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"

        android:gravity="center_vertical"
        android:spacing="16dp"
    />
</RelativeLayout>
```

ListAdapter

译者署名： 德罗德

译者链接：<http://sparkrico.javaeye.com>

翻译时间： 2010-11-03

版本： Android 2.2 r1

结构

继承关系

public interface ListAdapter extends Adapter

 android.widget.ListAdapter

子类及间接子类

 直接子类

 ArrayAdapter<T>, BaseAdapter, CursorAdapter, HeaderViewListAdapter,
 ResourceCursorAdapter, SimpleAdapter, SimpleCursorAdapter, WrapperListAdapter

类概述

扩展 Adapter 是在 ListView 与数据之间的一座桥梁。通常数据来自于游标，但不是必须的。ListView 可以显示包含在 ListAdapter 里的所有数据。



公共方法

public abstract boolean areAllItemsEnabled ()

在 ListAdapter 中所有的项目都是可用的？如果是，则代表所有的项目都是可选择，可用鼠标点击的。

返回值

 如果所有项目是可用的返回真

public abstract boolean isEnabled (int position)

如果指定的位置不是一个隔离(separator)项目（隔离项目是一个不可选择，不可用鼠标点击的项目）则返回真。如果位置是无效的，其结果将是不确定的。在这种情况下一个 ArrayIndexOutOfBoundsException(越界)异常将抛出。

参数

position 项目的索引

返回值

如果这个项目不是一个隔离(separator)项目则返回真。

补充

文章精选

[android adapter 的体系](#)

ListView

翻译人: Tina

版本: Android 2.2 r1

翻译时间: 2010 年 11 月 13 日

public class ListView extends AbsListView

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.AdapterView<T extends android.widget.Adapter>  
                android.widget.AbsListView  
                    android.widget.ListView
```

直接子类

ExpandableListView (使用竖滚动条查看的两级列表视图)

类概述



通过竖滚动条查看的列表视图。ListAdapter 里包含的内容和此视图相关联。参见 [List View tutorial](#)。

内部类

class ListView.FixedViewInfo

表示一个列表中的固定视图，如放在最顶部的页眉和最底部的页脚

XML 属性

属性名称	描述
------	----

android:choiceMode	规定此 ListView 所使用的选择模式。缺省状态下，list 没有选择模式。 属性值必须设置为下列常量之一： none，值为 0，表示无选择模式；singleChoice，值为 1，表示最多可以有一项被选中；multipleChoice，值为 2，表示可以多项被选中。 可参看全局属性资源符号 choiceMode。
android:divider	规定 List 项目之间用某个图形或颜色来分隔。可以用 "@[+][package:]type:name" 或者 "?[package:]type:name"（主题属性）的形式来指向某个已有资源；也可以用 "#rgb"， "#argb"， "#rrggbb" 或者 "#aarrggbb" 的格式来表示某个颜色。 可参看全局属性资源符号 divider。
android:dividerHeight	分隔符的高度。若没有指明高度，则用此分隔符固有的高度。必须为带单位的浮点数，如 "14.5sp"。 可用的单位如 px (pixel 像素)， dp (density-independent pixels 与密集度无关的像素)， sp (scaled pixels based on preferred font size 基于字体大小的固定比例的像素)， in (inches 英寸)， mm (millimeters 毫米)。 可以用 "@[package:]type:name" 或者 "?[package:]type:name"（主题属性）的形式来指向某个包含此类型值的资源。 可参看全局属性资源符号 dividerHeight。
android:entries	引用一个将使用在此 ListView 里的数组。若数组是固定的，使用此属性将比在程序中写入更为简单。 必须以 "@[+][package:]type:name" 或者 "?[package:]type:name" 的形式来指向某个资源。 可参看全局属性资源符号 entries。
android:footerDividersEnabled	设成 false 时，此 ListView 将不会在页脚视图前画分隔符。此属性缺省值为 true。 属性值必须设置为 true 或 false。 可以用 "@[package:]type:name" 或者 "?[package:]type:name"（主题属性）的形式来指向某个包含此类型值的资源。 可参看全局属性资源符号 footerDividersEnabled。
android:headerDividersEnabled	设成 false 时，此 ListView 将不会在页眉视图后画分隔符。此属性缺省值为 true。 属性值必须设置为 true 或 false。 可以用 "@[package:]type:name" 或者 "?[package:]type:name"（主题属性）的形式来指向某个包含此类型值的资源。 可参看全局属性资源符号 headerDividersEnabled。

常量

`Int CHOICE_MODE_MULTIPLE`

(常量值为 2) 列表允许同时选取多项

`Int CHOICE_MODE_NONE`

(常量值为 0) 普通列表，不指明选取模式

`Int CHOICE_MODE_SINGLE`

(常量值为 1) 列表只允许选取最多一项

公共方法

`public void addFooterView (View v)`

加一个固定显示于 `list` 底部的视图。如果此方法被调用超过一次，所加的几个视图将按照它们加入的顺序排列。加入的视图可取得焦点。

注意：在调用 `setAdapter` 之前调用此方法。这样的话，可以利用点击光标来收起有 `header view` 和 `footer view` 的 `ListView`。

参数

`v` 要加的视图

`public void addFooterView (View v, Object data, boolean isSelectable)`

加一个固定显示于 `list` 底部的视图。如果此方法被调用超过一次，所加的几个视图将按照它们加入的顺序排列。加入的视图可取得焦点。

注意：在调用 `setAdapter` 之前调用此方法。这样的话，可以利用点击光标来收起有 `header view` 和 `footer view` 的 `ListView`。

参数

`v` 要加的视图

`data` 和此视图关联的数据

`isSelectable` 设为 `true` 则表示 `footer view` 可以被选中

`public void addHeaderView (View v)`

加一个固定显示于 `list` 顶部的视图。如果此方法被调用超过一次，所加的几个视图将按照它们加入的顺序排列。加入的视图可取得焦点。

注意：在调用 `setAdapter` 之前调用此方法。这样的话，可以利用点击光标来收起有 `header view` 和 `footer view` 的 `ListView`。

参数

`v` 要加的视图

`public void addHeaderView (View v, Object data, boolean isSelectable)`

加一个固定显示于 `list` 顶部的视图。如果此方法被调用超过一次，所加的几个视图将按照它们加入的顺序排列。加入的视图可取得焦点。

注意：在调用 `setAdapter` 之前调用此方法。这样的话，可以利用点击光标来收起有 `header view` 和 `footer view` 的 `ListView`。

参数

`v` 要加的视图

`data` 和此视图关联的数据

`isSelectable` 表示此 `header view` 可选与否

```
public void clearChoices ()
```

取消之前设置的任何选择

```
public boolean dispatchKeyEvent (KeyEvent event)
```

按照可以获得焦点的顺序（从视图树的顶端到当前获得焦点的视图），分派一个按键事件给下一个视图。若此视图有焦点，事件将会分派给它自己。否则它将按照顺序，分派给下一个节点。此方法同时触动所有按键监听器。

参数

event 被分派的事件

返回

若事件被处理，则返回 true；否则为 false

```
public boolean dispatchPopulateAccessibilityEvent (AccessibilityEvent event)
```

在视图的子项目被构建时，分派一个辅助事件。

参数

event 事件

返回

若事件全部完成，则返回 true

```
public ListAdapter getAdapter ()
```

返回 ListView 当前用的适配器。返回的适配器不可以和传给 setAdapter(ListAdapter) 的参数一样，但是可以是 WrapperListAdapter。

返回

当前用来显示 ListView 中数据的适配器

参见

[setAdapter\(ListAdapter\)](#)

```
public long[] getCheckItemIds ()
```

此方法已经过时了。使用 [getCheckedItemIds\(\)](#) 代替。

返回被选中项目的索引集合。只有当选择模式没有被设置为 CHOICE_MODE_NONE 时才有效。

```
public long[] getCheckedItemIds ()
```

返回被选中项目的索引集合。只有当选择模式没有被设置为 CHOICE_MODE_NONE，并且适配器有稳定的 ID (`hasStableIds()==true`) 时，结果才有效。

返回

一个新的数组，包含列表中每个被选中的索引 (id)

```
public int getCheckedItemPosition ()
```

返回当前被选中的项目。只有当选择模式已被设置为 CHOICE_MODE_SINGLE 时，结果才有效。

返回

返回当前被选中的项目的索引；若没有项目被选中，则返回 INVALID_POSITION

参见

`setChoiceMode(int)`

`public SparseBooleanArray getCheckedItemPositions ()`

返回当前被选中的项目集合。只有当选择模式没有被设置为 `CHOICE_MODE_NONE` 时，结果才有效。

返回

类型为 `SparseBooleanArray` 的值，其中，对每一个索引所代表的项目，若被选中，则返回 `true`；当选择模式被设置为 `CHOICE_MODE_NONE` 时，返回 `null`。

`public int getChoiceMode ()`

返回

返回当前的选择模式

参见

`setChoiceMode(int)`

`public Drawable getDivider ()`

返回

返回当前画在列表元素之间，作为分隔符的图形

`public int getDividerHeight ()`

返回

返回分隔符的高度

`public int getFooterViewsCount ()`

返回

列表中的页脚视图数量；缺省实现时，数量为 0

`public int getHeaderViewsCount ()`

返回

列表中的页眉视图数量；缺省实现时，数量为 0

`public boolean getItemsCanFocus ()`

返回

`ListAdapter` 所生成的视图是否可以包含能取得焦点的项目

`public int getMaxScrollAmount ()`

返回

The maximum amount a list view will scroll in response to an arrow event.

响应箭头事件时，列表视图可以滚动的最大值。（译者注：此处翻译待改进，恐怕需要仔细查看源代码才能明白其含义，也可以用 `Google Code` 搜索相关的代码）

`public boolean isItemChecked (int position)`

对于由 `position` 指定的项目，返回其是否被选中。只有当选择模式已被设置为

CHOICE_MODE_SINGLE 或 CHOICE_MODE_MULTIPLE 时，结果才有效。

参数

position 要返回选中状态的项目

返回

返回项目的选中状态；若选择模式无效，则返回 false

```
public boolean onKeyDown (int keyCode, KeyEvent event)
```

KeyEvent.Callback.onKeyMultiple()的缺省实现：若视图被激活并且可以被点击，当出现 KEYCODE_DPAD_CENTER 和 KEYCODE_ENTER 代表的行为时，做点击该视图的动作。

参数

keyCode 表示按某个按键的按键代号，参见 KeyEvent

event 定义按键动作的按键事件对象

返回

若事件被成功处理，则返回 true；若想要下一个接收器处理该事件，则返回 false

```
public boolean onKeyMultiple (int keyCode, int repeatCount, KeyEvent event)
```

KeyEvent.Callback.onKeyMultiple()的缺省实现：总是返回 false（不处理该事件）。

参数

keyCode 表示按某个按键的按键代号，参见 KeyEvent

repeatCount 实现动作的次数

event 定义按键动作的按键事件对象

返回

若事件被成功处理，则返回 true；若想要下一个接收器处理该事件，则返回 false

```
public boolean onKeyUp (int keyCode, KeyEvent event)
```

KeyEvent.Callback.onKeyMultiple()的缺省实现：当出现 KEYCODE_DPAD_CENTER 和 KEYCODE_ENTER 代表的行为时，做点击该视图的动作。

参数

keyCode 表示按某个按键的按键代号，参见 KeyEvent

event 定义按键动作的按键事件对象

返回

若事件被成功处理，则返回 true；若想要下一个接收器处理该事件，则返回 false

```
public void onRestoreInstanceState (Parcelable state)
```

重新创建并显示一个视图，此视图拥有之前 onSaveInstanceState() 保存的内部状态。当 state 为 null 时，此方法不会被调用。

参数

state 之前 onSaveInstanceState() 保存的状态

```
public Parcelable onSaveInstanceState ()
```

保存视图的内部状态，用于以后创建新的拥有同样状态的实例。可保存的状态只包含非持久性的，或者可重新组建的信息。比如，永远不可能保存你当前在屏幕上的位置，因为当新的实例被放置于视图层次体系中时，位置会被重新计算。

一些可以被保存的状态：文本视图（但是通常不是指文本本身，因为文本是被保存在内

容提供商或其他持久性的储存体中) 中当前的光标位置; 列表视图中当前的选中项。

返回

返回一个包含视图当前动态状态的接口方法对象; 若没有东西被保存, 则返回 null。缺省情况下返回 null。

```
public boolean onTouchEvent (MotionEvent ev)
```

此方法用于处理触摸屏的动作事件。

参数

ev 动作事件

返回

若事件被成功处理, 则返回 true; 否则返回 false

```
public boolean performItemClick (View view, int position, long id)
```

调用定义好的 OnItemClickListener。

参数

view AdapterView 中被点击到的视图

position 视图在适配器中的索引

id 被点击到的项目的行 id

返回

若有定义好的 OnItemClickListener 被成功调用, 则返回 true; 否则返回 false

```
public boolean removeFooterView (View v)
```

删除之前加入的某个页脚视图。

参数

v 要删除的视图

返回

若视图被成功删除, 则返回 true; 若此视图不是页脚视图, 则返回 false

```
public boolean removeHeaderView (View v)
```

删除之前加入的某个页眉视图。

参数

v 要删除的视图

返回

若视图被成功删除, 则返回 true; 若此视图不是页眉视图, 则返回 false

```
public boolean requestChildRectangleOnScreen (View child, Rect rect, boolean immediate)
```

当组里的某个子项需要被定位在屏幕的某个矩形范围时, 调用此方法。

重载此方法的 ViewGroup 可确认以下几点:

- 子项目将是组里的直系子项
- 矩形将在子项目的坐标体系中

重载此方法的 ViewGroup 必须保证以下几点:

- 若矩形已经是可见的, 则没有东西会改变
- 为使矩形区域全部可见, 视图将可以被滚动显示

参数

child 发出请求的子项目
rect 子项目坐标系内的矩形，即此子项目希望在屏幕上的定位
immediate 设为 **true**，则禁止动画和缓释移动滚动条
返回
这个可滚动显示的组，是否接受请求

public void setAdapter (ListAdapter adapter)

设置 ListView 背后的数据。根据 ListView 目前使用的特性，**adapter** 可能被 **WrapperListAdapter** 收起。例如：加页眉和/或页脚会使 **adapter** 被收起。

参数

adapter 负责维护列表背后的数据，以及生成视图来显示数据里的项目

参见

getAdapter()

public void setCacheColorHint (int color)

当 **color** 的值不为 0 时，此值表示的颜色将提示使用者，列表正在一片单色不透明的背景上被画出。

参数

color 背景色

public void setChoiceMode (int choiceMode)

设置 List 的选择模式。缺省情况下，列表没有选择模式（即值为 **CHOICE_MODE_NONE**）。

参数

choiceMode 值可为 **CHOICE_MODE_NONE**, **CHOICE_MODE_SINGLE** 和 **CHOICE_MODE_MULTIPLE** 中的一种

public void setDivider (Drawable divider)

设置将画在列表中每个项目之间的图形。如果图形没有已设定好的高度，则必须同时调用 **setDividerHeight(int)**。

参数

divider 将用作分隔符的图形

public void setDividerHeight (int height)

设置分隔符（画在列表中每个项目之间）的高度。调用此方法将覆盖由 **setDivider(Drawable)** 设置的高度。

参数

height 分隔符的新高度，单位为像素

public void setFooterDividersEnabled (boolean footerDividersEnabled)

设置可以或者不可以为页脚视图画上分隔符。

参数

headerDividersEnabled 设为 **true**，表明可以画；设为 **false** 则不可以

参见

setHeaderDividerEnabled(boolean)

`addFooterView(android.view.View)`

`public void setHeaderDividersEnabled (boolean headerDividersEnabled)`

设置可以或者不可以为页眉视图画上分隔符。

参数

`headerDividersEnabled` 设为 true, 表明可以画; 设为 false 则不可以

参见

`setFooterDividerEnabled(boolean)`

`addHeaderView(android.view.View)`

`public void setItemChecked (int position, boolean value)`

设置 position 所指定项目的选择状态。只有选择模式为 CHOICE_MODE_SINGLE 或者 CHOICE_MODE_MULTIPLE 时, 此设置才有效。

参数

`position` 需要改变选择状态的项目的索引

`value` 新的选择状态

`public void setItemsCanFocus (boolean itemsCanFocus)`

表明在由 ListAdapter 创建的视图中, 可包含能获得焦点的项目。

参数

`itemsCanFocus` 若项目能获得焦点, 则设为 true; 否则为 false

`public void setSelection (int position)`

选中 position 指定的项目。若为触摸模式, 则指定项目不会被选中, 但位置变化一样。若 position 的值小于 0, 则 position 为 0 的项目将被选中。

参数

`position` 需要选中的项目的索引 (从 0 开始)

`public void setSelectionAfterHeaderView ()`

选中页眉视图下的第一个列表项目。

`public void setSelectionFromTop (int position, int y)`

选中 position 指定的项目, 并将所选项置于距离 ListView 顶端 y 像素的位置 (若为触摸模式, 则指定项目不会被选中, 但位置变化一样)。

参数

`position` 需要选中的项目的索引 (从 0 开始)

`y` 距离 ListView (包括间隙) 顶端的位置

受保护方法

`protected boolean canAnimate ()`

表示此视图组是否可以在第一次被布局后, 仍可以动态调整其子项。

返回

若可以则为 true, 否则为 false

protected void dispatchDraw (Canvas canvas)

调用此方法来绘出子视图。可被衍生类重写，以便在其子项被画出之前取得控制权。

参数

canvas 绘出 View 所用的 canvas (画布?)

protected View findViewTraversal (int id)

参数

id 要找的 View 的 id

返回值

有此 id 的 View，若没有找到则为 null

protected View findViewWithTagTraversal (Object tag)

参数

tag 要找的 View 的标签

返回值

有此标签的 View，若没有找到则为 null

protected void layoutChildren ()

子类必须重写此方法来布局其子项。

protected void onFinishInflate ()

当 View 以及所有子项从 XML 中导入时被调用，是导入的最后一步。即使子类重写 onFinishInflate，也必须保证有调用超方法，这样，方法才会被调用。

protected void onFocusChanged (boolean gainFocus, int direction, Rect

previouslyFocusedRect)

当 View 的焦点改变时被调用。重写时，要确保超类的直接调用，这样取得焦点的方式才是标准的。

参数

gainFocus 若 View 有焦点，则为 True；否则为 False。

direction 当 requestFocus() 被调用时，方向焦点被移动。其值可为 FOCUS_UP, FOCUS_DOWN, FOCUS_LEFT 或 FOCUS_RIGHT。在使用缺省条件的情况下，direction 并不总是可用。

previouslyFocusedRect 之前得到焦点的 View 的坐标系统所构成的矩形。如果可用，这个将被当成精确信息（表明焦点从何而来以及从何方向而来）来传递；否则将传递 null。

protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)

View 调用此方法来确定本身和所包含内容的大小。此方法被 measure(int,int) 唤起，而且必须被子类重写以得到所包含内容的确切大小。

注意：当重写此方法时，必须调用 setMeasureDimension(int,int) 来保存 View 的大小。如果没有做到，将会引发一个 measure(int,int) 抛出的 IllegalStateException（非法状态错误）。超类 onMeasure(int,int) 可以被调用。

编写基类的确认大小的方法，缺省情况下是根据其背景大小来确认，除非 MeasureSpec

允许有更大的高度或宽度。子类必须重写 `onMeasure(int,int)` 以得到对其内容大小的更准确的测量。

若此方法被重写，它的子类需要确保其高度和宽度至少达到 `View` 所规定的最小值（可通过 `getSuggestedMinimumHeight()` 和 `getSuggestedMinimumWidth()` 得到）。

参数

`widthMeasureSpec` 受上一层大小影响下的对水平空间的要求。可参看

`View.MeasureSpec`。

`heightMeasureSpec` 受上一层大小影响下的对垂直空间的要求。可参看

`View.MeasureSpec`。

`protected void onSizeChanged (int w, int h, int oldw, int oldh)`

当 `View` 的大小改变时此方法被调用。如果 `View` 是刚刚被加入，则视之前的值为 0。

参数

`w` `View` 的当前宽度

`h` `View` 的当前高度

`oldw` `View` 大小改变之前的宽度

`oldh` `View` 大小改变之前的高度

补充

参考链接

[Android 入门第六篇之 ListView \(一\)](#)

[android ListView 详解](#)

[android 异步加载 ListView 中的图片](#)

[Google I/O 2010 - The world of ListView](#)

[Android: 显示 SD 卡文件列表](#)

[Android: 带图标的 ListView 实现](#)

[ListView 和 getView 的原理+如何在 ListView 中放置多个 item](#)

MediaController

译者署名： 唐明
版本： Android 2.3 r1

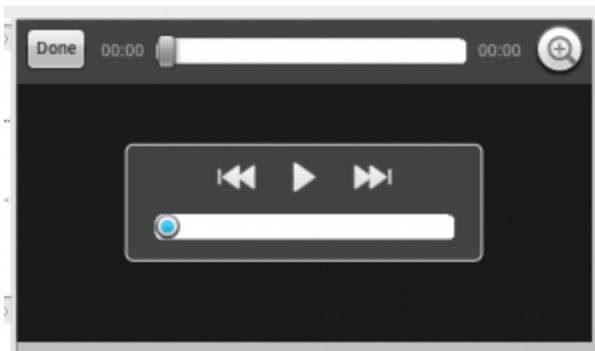
结构

继承关系

public class MediaController extends FrameLayout

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.FrameLayout
                android.widget.MediaController
```

类概述



一个包含媒体播放器(MediaPlayer)控件的视图。包含了一些典型的按钮，像"播放(Play)/暂停(Pause)"，"倒带(Rewind)"，"快进(Fast Forward)"与进度滑动器(progress slider)。它管理媒体播放器(MediaController)的状态以保持控件的同步。

通过编程来实例化使用这个类。这个媒体控制器将创建一个具有默认设置的控件，并把它们放到一个窗口里漂浮在你的应用程序上。具体来说，这些控件会漂浮在通过 `setAnchorView()`指定的视图上。如果这个窗口空闲 3 秒那么它将消失，直到用户触摸这个视图的时候重现。

当媒体控制器是在一个 XML 布局资源文件中创建的时候，像 `show()` 和 `hide()` 这些函数是无效的。媒体播放器将根据这些规则去显示和隐藏：

- 在调用 `setPrevNextListeners()` 函数之前，"previous" 和 "next" 按钮都是隐藏的。
- 如果 `setPrevNextListeners()` 函数被调用但传入的监听器参数是 `null`，那么 "previous" 和 "next" 按钮是可见的但是处于禁用状态。
- "rewind" 和 "fastforward" 按钮是显示的，如果不需要可以使用构造函数 `MediaController(Context, boolean)` 将 `boolean` 设置为 `false`。

公共方法

public boolean dispatchKeyEvent (KeyEvent event)

在焦点路径上分发按钮事件到下一个视图。该路径从视图树的顶端遍历到当前获得焦点的视图。如果当前视图已获得焦点，就分发给自身。否则，就分发到下一个节点的焦点

路径上。这个方法也可以激发任何一个按键消息监听器。

参数

event 被分发的事件

返回值

如果这个事件被处理了返回 true，否则返回 false。

public void hide ()

从屏幕中移除控制器。

public boolean isShowing ()

(译者注：判断媒体控制器是否处于可见状态。)

public void onFinishInflate ()

XML 文件加载视图完成时调用。这个函数在加载的最后阶段被调用，所有的子视图已经被添加。

即使子类重写了 onFinishInflate 方法，也应该始终确保调用父类方法，以便我们调用。

public boolean onTouchEvent (MotionEvent event)

实现这个方法来处理触摸屏幕引发的事件。

参数

event 动作事件

返回值

如果这个事件被处理了返回 true，否则返回 false。

public boolean onTrackballEvent (MotionEvent ev)

实现这个方法处理轨迹球的动作事件，轨迹球相对运动的最后一个事件能用

[MotionEvent.getX\(\)](#) 和 [MotionEvent.getY\(\)](#) 函数获取。这些都是标准化的，用 1 表示用户按下一个 DPAD 按键。（因此他们将经常使用小数值表示，为轨迹球提供更多的细微运动信息）（译者注：DPAD 按键事件：`KeyEvent.KEYCODE_DPAD_CENTER`（居中）、`KeyEvent.KEYCODE_DPAD_DOWN`（向下）、`KeyEvent.KEYCODE_DPAD_LEFT`（向左）、`KeyEvent.KEYCODE_DPAD_RIGHT`（向右）、`KeyEvent.KEYCODE_DPAD_UP`（向上）作比较。分别表示居中、下移、左移、右移、上移的操作。相关链接：[onTrackBallEvent 方法简介](#)）

参数

ev 动作事件

返回值

如果这个事件被处理了返回 true，否则返回 false。

public void setAnchorView (View view)

设置这个控制器绑定(anchor/锚)到一个视图上。例如可以是一个 VideoView 对象，或者是你的 activity 的主视图。

参数

view 将视图来绑定控制器时可见

public void setEnabled (boolean enabled)

设置视图对象的有效状态。这也可以改变子类的有效状态。

参数

enabled 如果要让这个视图对象可用就设置为 true，否则设置为 false。

public void setMediaPlayer (MediaController.MediaPlayerControl player)

(译者注：把这个媒体控制器设置到 VideoView 对象上。)

public void setPrevNextListeners (View.OnClickListener next, View.OnClickListener prev)

(译者注：设置"previous"和 "next"按钮的监听器函数。)

public void show (int timeout)

在屏幕上显示这个控制器。它将在闲置'超时 (timeout)'毫秒到达后自动消失。

参数

timeout 这个参数以毫秒为单位。如果设置为 0 将一直显示到调用 hide() 函数为止。

public void show ()

在屏幕上显示这个控制器。它将在 3 秒以后自动消失。

补充

文章精选

[推荐][\[Android 学习指南\]Android 多媒体\(Media\)](#)

[调用 android api 播放视频](#)

[Customize android VideoView \(ii\)](#)

MultiAutoCompleteTextView

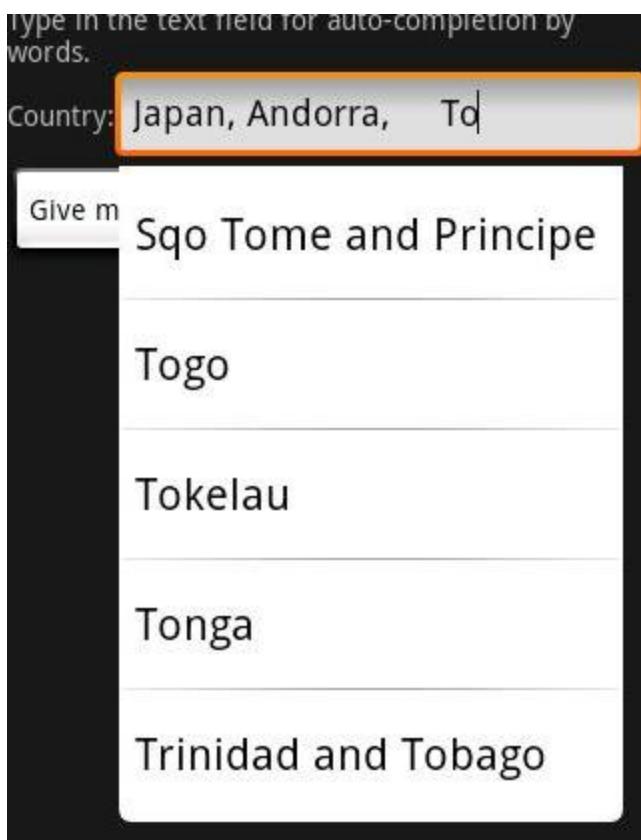
颖哥儿

版本: Android 2.2 r1

`public class MultiAutoCompleteTextView extends AutoCompleteTextView`

```
java.lang.Object  
    android.view.View  
        android.widget.TextView  
            android.widget.EditText  
                android.widget.AutoCompleteTextView  
                    android.widget.MultiAutoCompleteTextView
```

概述



一个继承自 [AutoCompleteTextView](#) 的可编辑的文本视图，能够对用户键入的文本进行有效地扩充提示，而不需要用户输入整个内容。（用户输入一部分内容，剩下的部分系统就会给予提示）。

用户必须提供一个 [MultiAutoCompleteTextView.Tokenizer](#) 以用来区分不同的子串。

下面的代码片段展示了如何创建一个文本视图，这个视图用来对用户输入的国家名称进行有效地补充提示。

```

public class CountriesActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.autocomplete_7);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_dropdown_item_1line, COUNTRIES);
        MultiAutoCompleteTextView textView = (MultiAutoCompleteTextView) findViewById(R.id.edit);
        textView.setAdapter(adapter);
        textView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
    }

    private static final String[] COUNTRIES = new String[] {
        "Belgium", "France", "Italy", "Germany", "Spain"
    };
}

```

公共方法

public boolean enoughToFilter ()

此方法并不是检验什么时候文本的总长度超过了预定的值，而是在仅当从函数 `findTokenStart()` 到 `getSelectionEnd()` 函数得到的文本长度为 0 或者超过了预定值的时候才起作用。（译者注：当文本长度超过阈值时过滤）

public void performValidation ()

此子类方法并不是用来确定整个文本的有效性，而是用来确定文本中的单个符号的有效性。空标记将被移除。

public void setTokenizer (MultiAutoCompleteTextView.Tokenizer t)

设置用来决定用户正在输入文本的范围的分词组件。

受保护方法

protected void performFiltering (CharSequence text, int keyCode)

此方法并不过滤整个编辑框的内容，只是过滤从函数 `findTokenStart()` 到函数 `getSelectionEnd()` 获得的长度为 0 或者超过了预定的值的文本内容。

参数

`text` 指定过滤模式

`keyCode` 插入到编辑框中的最后一个字符；当字符（文本）是通过软键盘输入的时候，小心此字符的值可能为 `NULL`。

protected void performFiltering (CharSequence text, int start, int end, int keyCode)

启动对下拉式列表内容的过滤。过滤模式为编辑框中指定的范围。子类可覆盖此方法，以便于采用一个不同的模式。

protected void replaceText (CharSequence text)

通过替换从函数 `findTokenStart()` 到函数 `getSelectionEnd()` 得到的文本文本范围以及传递给函数 `terminateToken()` 的文版返回的结果来实现文本的输入。另外，文本的替换区域将会被标记为自动文本区，这样如果用户直接按 `DEL` 键，执行过程就会停止。子类可覆盖此方法来实现许多不同的文本插入工作。

参数

text 下拉列表里的选中项

补充

相关文章链接

[Android 控件之 AutoCompleteTextView、MultiAutoCompleteTextView 探究](#)

[AutoCompleteTextView 和 MultiAutoCompleteTextView](#)

[Auto Complete Text](#)

MultiAutoCompleteTextView.CommaTokenizer

颖哥儿
版本: Android 2.2 r1

```
public static class MultiAutoCompleteTextView.CommaTokenizer
    extends Object
    implements MultiAutoCompleteTextView.Tokenizer
```

```
java.lang.Object
    android.widget.MultiAutoCompleteTextView.CommaTokenizer
```

概述

这个简易的组件可以用于一些列表中，这些列表包含被逗号以及一个或数个空格断开的项目。

公共方法

```
public int findTokenEnd (CharSequence text, int cursor)
```

返回标记在文本中的结束位置，其偏移量大小为 cursor

```
public int findTokenStart (CharSequence text, int cursor)
```

返回标记在文本中的开始位置，其偏移量大小为 cursor

```
public CharSequence terminateToken (CharSequence text)
```

返回文本内容，如果有必要的话，此函数会确认文本是否以终结符结束（比如以空格或者逗号结尾）

MultiAutoCompleteTextView.Tokenizer

颖哥儿

版本：Android 2.2 r1

public static interface MultiAutoCompleteTextView.Tokenizer

android.widget.MultiAutoCompleteTextView.Tokenizer

间接子类

MultiAutoCompleteTextView.CommaTokenizer, Rfc822Tokenizer

概述

公共方法

public abstract int findTokenEnd (CharSequence text, int cursor)

返回标记在文本中的结束位置，其偏移量大小为 cursor

public abstract int findTokenStart (CharSequence text, int cursor)

返回标记在文本中的开始位置，其偏移量大小为 cursor

public abstract CharSequence terminateToken (CharSequence text)

返回文本内容，如果有必要的话，此函数会确认文本是否以终结符结束（比如以空格或者逗号结尾）

QuickContactBadge

农民伯伯

版本: Android 2.2

```
public class QuickContactBadge  
    extends ImageView  
    implements View.OnClickListener  
  
java.lang.Object  
    android.view.View  
        android.widget.ImageView  
            android.widget.QuickContactBadge
```

使用截图



在 andorid 自带的 ApiDomos 的例子中有这个的代码: App/Activity/QuickContacksDemo。注意需要 android.permission.READ_CONTACTS 权限，并且联系人里面有数据，并且联系人需要有手机号码，不然出来是一个空的（看代码可知）。

类概述

控件常用于显示一个图片与标准的联系人快捷标示和点击行为。

公共方法

```
public void assignContactFromEmail (String emailAddress, boolean lazyLookup)
```

指定联系人的电子邮箱地址。(注: 它会先搜索这个号码, 如果没有会提醒你是否添加到联系人, 参见[文章 1](#))

参数

emailAddress 联系人的电子邮箱地址

lazyLookup 如果设置为 true, 将不会立即查找这个邮箱地址, 直到 View 被点击时。(注: 是否延迟匹配电子邮件)

```
public void assignContactFromPhone (String phoneNumber, boolean lazyLookup)
```

为联系人指定一个电话号码。(注: 参见[文章 1](#))

参数

phoneNumber 联系人的电话号码

lazyLookup 如果设置为 true, 将不会立即查找这个电话号码, 直到 View 被点击时。

```
public void assignContactUri (Uri contactUri)
```

指定和 QuickContactBadge 关联的联系人 URI。注意，这里只是显示 QuickContact 窗口，并不为你绑定联系人图片。

参数

contactUri CONTENT_URI 或 CONTENT_LOOKUP_URI 其中一种风格的 URI.

```
public void onClick (View v)
```

当 View 被点击时调用。

参数

v 被点击的 View.

```
public void setExcludeMimes (String[] excludeMimes)
```

设置一组要排除不显示的 MIMI 类型列表。例如，可以隐藏 Contacts.CONTENT_ITEM_TYPE 类型的图标。(注：如果像如下设置：

```
setExcludeMimes(new String[] { Contacts.CONTENT_ITEM_TYPE })
```

即隐藏了上面截图的第二个，仅显示电话和短信两个图标)

```
public void setMode (int size)
```

设置 QuickContact 的窗口模式。如下选项： MODE_SMALL、MODE_MEDIUM、MODE_LARGE。(注：默认为 QuickContact.MODE_MEDIUM，设置为 MODE_LARGE 时会同时显示联系人名称)

补充

文章精选

[Android Quick Tip: Using the Quick Contact Badge](#)

[Contacts 模块中的 QuickContacts](#)

[Android 自定义泡泡效果](#)

[推荐][devoquickaction](#)(自定义 QuickContactBadge)

RadioButton

农民伯伯

版本: Android 2.2 r1

`public class RadioButton extends CompoundButton`

```
java.lang.Object  
    android.view.View  
        android.widget.TextView  
            android.widget.Button  
                android.widget.CompoundButton  
                    android.widget.RadioButton
```

概述



单选按钮是一种双状态的按钮，可以选择或不选中。在单选按钮没有被选中时，用户能够按下或点击来选中它。但是，与复选框相反，用户一旦选中就不能够取消选中（译者注：可以通过代码来控制，界面上点击的效果是一旦选中之后就不能取消选中了）。

多个单选按钮通常与 RadioGroup 同时使用。当一个单选组（RadioGroup）包含几个单选按钮时，选中其中一个的同时将取消其它选中的单选按钮。（译者注：示例参见[这里](#)）

公共方法

`public void toggle ()`

将单选按钮更改为与当前选中状态相反的状态。

如果这个单选按钮已经选中，这个方法将不切换单选按钮。（译者注：请看源码：

```
@Override  
public void toggle() {  
    // we override to prevent toggle when the radio is already  
    // checked (as opposed to check boxes widgets)  
    if (!isChecked()) {  
        super.toggle();  
    }  
}
```

RatingBar

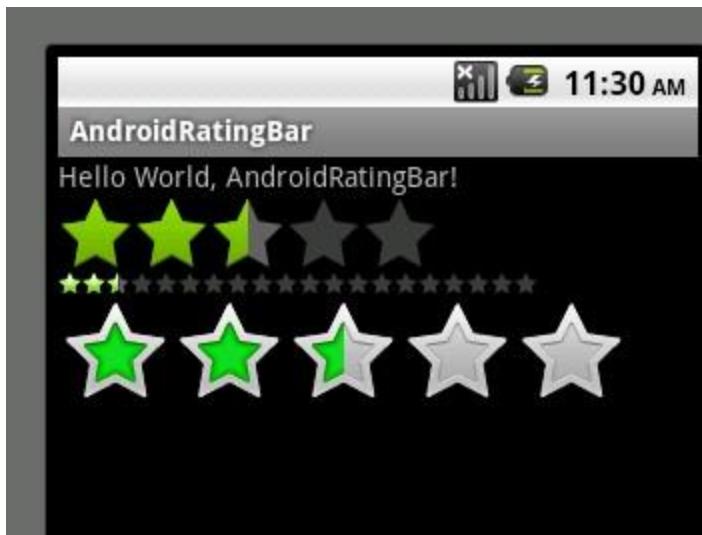
翻译人: wallace2010/ madgoat
译者博客: <http://madgoat.cn/>、
<http://blog.csdn.net/springiscoming2008>

版本: Android 2.2 r1

public class RatingBar extends AbsSeekBar

```
java.lang.Object
    android.view.View
        android.widget.ProgressBar
            android.widget.AbsSeekBar
                android.widget.RatingBar
```

类概述



RatingBar 是基于 SeekBar 和 ProgressBar 的扩展，用星型来显示等级评定。使用 RatingBar 的默认大小时，用户可以触摸/拖动或使用键来设置评分，它有两种样式(小风格用 `ratingBarStyleSmall`, 大风格用 `ratingBarStyleIndicator`)，其中大的只适合指示，不适合于用户交互。

当使用可以支持用户交互的 RatingBar 时，无论将控件(widgets)放在它的左边还是右边都是不合适的。

只有当布局的宽被设置为 `wrap_content` 时，设置的星星数量(通过函数 `setNumStars(int)` 或者在 XML 的布局文件中定义) 将显示出来 (如果设置为另一种布局宽的话，后果无法预知)。

次级进度一般不应该被修改，因为他仅仅是被当作星型部分内部的填充背景。

参见 [Form Stuff tutorial](#).

嵌套类

接口: `RatingBar.OnRatingBarChangeListener`

一个回调函数，当星级进度改变时修改客户端的星级。

XML 属性

属性名称	描述
android:isIndicator	RatingBar 是否是一个指示器（用户无法进行更改）
android:numStars	显示的星型数量，必须是一个整形值，像“100”。
android:rating	默认的评分，必须是浮点类型，像“1.2”。
android:stepSize	评分的步长，必须是浮点类型，像“1.2”。

公共方法

`public int getNumStars ()`

返回显示的星型数量

返回值

显示的星型数量

`public RatingBar.OnRatingBarChangeListener getOnRatingBarChangeListener ()`

返回值

监听器（可能为空）监听评分改变事件

`public float getRating ()`

获取当前的评分（填充的星型的数量）

返回值

当前的评分

`public float getStepSize ()`

获取评分条的步长

返回值

The step size.

步长

`public boolean isIndicator ()`

返回值

判断当前的评分条是否仅仅是一个指示器（注：即能否被修改）

`public void setIndicator (boolean isIndicator)`

设置当前的评分条是否仅仅是一个指示器（这样用户就不能进行修改操作了）

参数

isIndicator Bool 值，是否是一个指示器

`public synchronized void setMax (int max)`

设置评分等级的范围，从 0 到 max

参数

max 评分条最大范围。

`public void setNumStars (int numStars)`

设置显示的星型的数量。为了能够正常显示它们，建议将当前 `widget` 的布局宽度设置为 `wrap content`

参数

`numStars` 星型的数量

```
public void setOnRatingBarChangeListener (RatingBar.OnRatingBarChangeListener listener)
```

设置当评分等级发生改变时回调的监听器

参数

`listener` 监听器

```
public void setRating (float rating)
```

设置分数（星型的数量）

参数

`rating` 设置的分数

```
public void setStepSize (float stepSize)
```

设置当前评分条的步长（step size）

参数

`stepSize` 评分条的步进。例如：如果想要半个星星，它的值为 0.5。

受保护方法

```
protected synchronized void onMeasure (int widthMeasureSpec, int heightMeasureSpec)
```

权衡 `view` 和 `content` 来决定它的宽度和高度的整齐。它被 `measure(int, int)` 调用 并且应该被子类所覆盖，以便提供准确高效的布局测量。

规定：当覆盖这个方法的时候，你必须调用 `setMeasuredDimension(int, int)` 以便存储精确的视图的宽和高。如果不这样做的话将触发 `IllegalStateException` 异常，被函数 `measure(int, int)` 抛出。调用父类 `onMeasure(int, int)` 是合理的。

尺寸的基本类的实现默认是背景大小，除非通过 `MeasureSpec` 允许大的尺寸。子类应该覆盖 `onMeasure(int, int)` 以便提供更好的布局大小。

如果这个方法被覆盖，子类应该负责确保标准的宽和高至少是视图的最小宽度和高度的值（分别为 `getSuggestedMinimumHeight()` 和 `getSuggestedMinimumWidth()` 两方法）。

参数

`widthMeasureSpec` 受主窗口支配的水平空间要求。这个需求通过 `View.MeasureSpec` 进行编码。

`heightMeasureSpec` 受主窗口支配的垂直空间要求。这个需求通过 `View.MeasureSpec` 进行编码。

补充

文章链接

[Android 控件之 RatingBar 评分条](#)

[Android 更换 RatingBar 图片](#)

[\[Android 学习指南\]RatingBar 评分条](#)

示例代码 ([代码转载自 Android 手机开发者论坛](#))

```
AndroidRatingBar.java

public class AndroidRatingBar extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final RatingBar ratingBar_Small =
(RatingBar)findViewById(R.id.ratingbar_Small);
        final RatingBar ratingBar_Indicator =
(RatingBar)findViewById(R.id.ratingbar_Indicator);
        final RatingBar ratingBar_default =
(RatingBar)findViewById(R.id.ratingbar_default);

        ratingBar_default.setOnRatingBarChangeListener(new
RatingBar.OnRatingBarChangeListener() {

    public void onRatingChanged(RatingBar ratingBar, float rating,
        boolean fromUser) {
        ratingBar_Small.setRating(rating);
        ratingBar_Indicator.setRating(rating);
        Toast.makeText(AndroidRatingBar.this,
"rating:"+String.valueOf(rating),
        Toast.LENGTH_LONG).show();
    } });
    }

    main.xml
<?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"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
    <RatingBar
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    style="?android:attr/ratingBarStyleIndicator"
    android:id="@+id/ratingbar_Indicator"
/>
<RatingBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="?android:attr/ratingBarStyleSmall"
    android:id="@+id/ratingbar_Small"
    android:numStars="20"
/>
<RatingBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="?android:attr/ratingBarStyle"
    android:id="@+id/ratingbar_default"
/>
</LinearLayout>
```

RatingBar.OnRatingBarChangeListener

译者: madgoat

博客: <http://madgoat.cn/>

版本: Android 2.2 r1

public static interface RatingBar.OnRatingBarChangeListener

android.widget. RatingBar.OnRatingBarChangeListener

概述

当评分等级改变时通知客户端的回调函数。这包括用户通过手势、方向键或轨迹球触发的改变，以及编程触发的改变。

公共方法

public abstract void onRatingChanged (RatingBar ratingBar, float rating, boolean fromUser)

通知评分等级已经被修改。客户端可以使用 `fromUser` 参数区分用户触发的改变还是编程触发的改变。当用户拖拽时，将不会连续不断的被调用，仅仅当用户最终离开触摸结束评分时调用。

参数

`ratingBar` 评分修改的 RatingBar

`rating` 当前评分分数。取值范围为 0 到星型的数量。

`fromUser` 如果评分改变是由用户触摸手势或方向键轨迹球移动触发的，则

返回 true

整理人: 农民伯伯

RelativeLayout

译者署名: Atomic

译者链接: <http://iapk.com>

版本: Android 2.3 r1

结构

继承关系

public class RelativeLayout extends ViewGroup

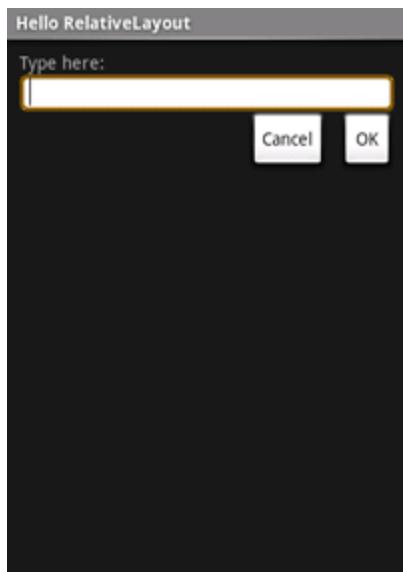
```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.RelativeLayout
```

子类及间接子类

直接子类

DialerFilter, TwoLineListItem

类概述



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或者和容器间的相对位置来进行定位。

注意，不能在 RelativeLayout 容器本身和他的子元素之间产生循环依赖，比如说，不能将 RelativeLayout 的高设置成为 [WRAP_CONTENT](#) 的时候将子元素的高设置成为 [ALIGN_PARENT_BOTTOM](#)。

(译者注：这点额外要注意，当然这也很好理解，如果容器是不定高的，那么子元素当然无法对齐容器的底边^_^)。

可以参考官方 SDK 中的 RelativeLayout 教学贴 [RelativeLayout tutorial](#)

常量

数据类型	常量名称	描述
int	ABOVE	定义将元素的底边对齐另一个元素的顶边
int	ALIGN_BASELINE	定义将元素基线的对齐另一个元素的基线
int	ALIGN_BOTTOM	定义将元素底边的对齐另一个元素的底边
int	ALIGN_LEFT	定义将元素的左边对齐另一个元素的左边
int	ALIGN_PARENT_BOTTOM	定义将元素的底边对齐其父容器(RelativeLayout)的底边
int	ALIGN_PARENT_LEFT	定义将元素的左边对齐其父容器(RelativeLayout)的左边
int	ALIGN_PARENT_RIGHT	定义将元素的右边对齐其父容器(RelativeLayout)的右边
int	ALIGN_PARENT_TOP	定义将元素的顶边对齐其父容器(RelativeLayout)的顶边
int	ALIGN_RIGHT	定义将元素的右边对齐另一个元素的右边
int	ALIGN_TOP	定义将元素的顶边对齐另一个元素的顶边
int	BELOW	定义将元素的顶边对齐另一个元素的底边
int	CENTER_HORIZONTAL	定义让元素在容器(RelativeLayout)内水平居中
int	CENTER_IN_PARENT	定义让元素位于容器(RelativeLayout)的中心
int	CENTER_VERTICAL	定义让元素在容器(RelativeLayout)内垂直居中
int	LEFT_OF	定义将元素的右边对齐另一个元素的左边
int	RIGHT_OF	定义将元素的左边对齐另一个元素的右边
int	TRUE	

内部类

Class **RelativeLayout.LayoutParams**

和 RelativeLayout 相关联的布局信息.

XML 属性

属性名称	描述
android:gravity	设置容器中的内容该如何定位。对象本身的 x 和 y 轴。
android:ignoreGravity	设置容器中的哪个子元素会不受 Gravity 的影响。 (译者注：接收参数：子元素的 Id, 如果设置成为 0，则全部子元素都会受到影响)

公共方法

public boolean dispatchPopulateAccessibilityEvent (AccessibilityEvent event)

分发 [AccessibilityEvent](#) 事件到该 [View](#) 的子视图中。

参数

event 事件本身.

返回值

如果事件分发完成，返回真。

```
public RelativeLayout.LayoutParams generateLayoutParams (AttributeSet attrs)
```

返回一个新的已设置属性集合的布局参数。

参数

attrs 构建 layout 布局参数的属性集合.

返回值

一个 RelativeLayout.LayoutParams 的实例或者他的一个子类

```
public int getBaseline ()
```

返回窗口空间的文本基准线到其顶边界的偏移量。如果这个部件不支持基准线对齐，这个方法返回-1。

返回值

基准线的偏移量，如果不支持基准线对齐则返回-1

```
public void requestLayout ()
```

当容器中的某个视图发生改变，会影响整个布局时，就调用这个方法，他会改变整个视图树的布局。

```
public void setGravity (int gravity)
```

描述子视图该如何摆放，默认值是 Gravity.LEFT | Gravity.TOP(译者注：说明 RelativeLayout 的元素如果不设置位置，会从左上角开始堆叠)。

参数

gravity 位置值，可以参考 [Gravity](#)

相关的 XML 属性

android:gravity

参见

[setHorizontalGravity\(int\)](#)

[setVerticalGravity\(int\)](#)

```
public void setHorizontalGravity (int horizontalGravity)
```

```
public void setVerticalGravity (int verticalGravity))
```

```
public void setIgnoreGravity (int viewId)
```

定义容器中的那个 View 会忽略容器设置的 Gravity，如果容器设置的 Gravity 是 Gravity.LEFT | Gravity.TOP 时，定义这个属性没有任何效果。

参数

viewId 将要忽略容器 Gravity 的视图的 id, 设置为 0 则不会有任何 View 忽略 Gravity.

相关的 XML 属性

android:ignoreGravity

参见

[setGravity\(int\)](#)

受保护方法

protected boolean checkLayoutParams (ViewGroup.LayoutParams p)

(译者注：检测是不是 AbsoluteLayout.LayoutParams 的实例)

protected ViewGroup.LayoutParams generateDefaultLayoutParams ()

返回一组宽度为 WRAP_CONTENT,高度为 WRAP_CONTENT,坐标是 (0, 0) 的布局参数。

返回值

一组默认的布局参数或 null 值。

**protected ViewGroup.LayoutParams generateDefaultLayoutParams
(ViewGroup.LayoutParams p)**

返回一组合法的受支持的布局参数。当一个 ViewGroup 传递一个布局参数没有通过 checkLayoutParams(android.view.ViewGroup.LayoutParams)检测的视图时，此方法被调用。此方法会返回一组新的适合当前 ViewGroup 的布局参数，可能从指定的一组布局参数中复制适当的属性。

参数

p 被转换成一组适合当前 ViewGroup 的布局参数.

返回值

一个 ViewGroup.LayoutParams 的实例或者其中的一个子节点

protected void onLayout (boolean changed, int l, int t, int r, int b)

在此视图 view 给他的每一个子元素分配大小和位置时调用。派生类可以重写此方法并且重新安排他们子类的布局。

参数

changed 这是当前视图 view 的一个新的大小或位置

l 相对于父节点的左边位置

t 相对于父节点的顶点位置

r 相对于父节点的右边位置

b 相对于父节点的底部位置

public void onMeasure (int widthMeasureSpec, int heightMeasureSpec)

测量视图以确定其内容宽度和高度。此方法被 measure(int, int)调用。需要被子类重写以提供对其内容准确高效的测量。

约定：当重写此方法时，你必须调用 setMeasuredDimension(int, int)来保存当前视图 view 的宽度和高度。不成功调用此方法将会导致一个 IllegalStateException 异常，是由 measure(int, int)抛出。所以调用父类的 onMeasure(int, int)方法是必须的。

父类的实现是以背景大小为默认大小，除非 MeasureSpec (测量细则) 允许更大的背景。子类可以重写 onMeasure(int,int)以对其内容提供更佳的尺寸。

如果此方法被重写，那么子类的责任是确认测量高度和测量宽度要大于视图 view 的最小宽度和最小高度 (getSuggestedMinimumHeight() and getSuggestedMinimumWidth())，使用这两个方法可以取得最小宽度和最小高度。

参数

widthMeasureSpec 强加于父节点的横向空间要求。要求是使用 View.MeasureSpec 进行编码。

heightMeasureSpec 强加于父节点的纵向空间要求。要求是使用

View.MeasureSpec 进行编码。

补充

文章精选

[Android 学习指南] [相对布局 RelativeLayout](#)

[Android RelativeLayout 属性](#)

[RelativeLayout 排版 免寫程式](#)[blogspot]

示例代码

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

ResourceCursorAdapter

译者署名： HalZhang

译者链接：<http://www.cnblogs.com/halzhang>

版本：Android 2.3 r1

结构

继承关系

public abstract class ResourceCursorAdapter extends CursorAdapter

java.lang.Object

 android.widget.BaseAdapter

 android.widget.CursorAdapter

 android.widget.ResourceCursorAdapter

子类及间接子类

直接子类

 SimpleCursorAdapter

类概述

这是一个简单的适配器，通过指定一个定义了视图 UI 的 XML 文件来创建视图。

构造函数

public ResourceCursorAdapter (Context context, int layout, Cursor c)

构造函数

参数

 Context 与 ListView 相关的正在运行的 SimpleListItemFactory 上下文

 layout 一个定义了列表项视图的布局文件资源 ID，这个布局同时定义列表项视图和下拉视图，直到你重写它们。

 c 获取数据的游标

public ResourceCursorAdapter (Context context,int layout, Cursor c, boolean autoRequery)

构造函数

参数

 Context 与 ListView 相关的正在运行的 SimpleListItemFactory 上下文

 layout 一个定义了列表项视图的布局文件资源 ID，这个布局同时定义列表项视图和下拉视图，直到你重写它们。

 c 获取数据的游标

 autoRequery 如果此参数为 true，当适配器的数据发生变化时，适配器会调用游标的 requery()方法，使最新的数据始终显示。

公共方法

public View newDropDownView (Context context, Cursor cursor, ViewGroup parent)

生成一个新的下拉视图来控制游标指向的数据

参数

- context** 应用程序全局信息接口（应用上下文）
- cursor** 获取数据的游标，它已经移动到正确的位置
- parent** 与新视图相关联的上级视图

返回值

新创建的视图

```
public View newView (Context context, Cursor cursor, ViewGroup parent)
```

根据指定的 xml 文件创建视图

参数

- context** 应用程序全局信息接口（应用上下文）
- cursor** 获取数据的游标，它已经移动到正确的位置
- parent** 与新视图相关联的上级视图

返回值

新创建的视图

参见

[newView\(android.content.Context, android.database.Cursor, ViewGroup\)](#)

```
public void setDropDownViewResource (int dropDownLayout)
```

设置下拉视图相应的布局资源

参数

- dropDownLayout** 用于创建下拉视图的布局资源

```
public void setViewResource (int layout)
```

设置列表项视图相应的布局资源

参数

- layout** 用于创建列表项视图的布局资源

补充

文章精选

[ListActivity 简介](#)

代码示例（ApiDemos\src\com\example\android\apis\app\QuickContactsDemo.java）

Java:

```
public class QuickContactsDemo extends ListActivity {  
    static final String[] CONTACTS_SUMMARY_PROJECTION = new String[] {  
        Contacts._ID, // 0  
        Contacts.DISPLAY_NAME, // 1  
        Contacts.STARRED, // 2  
        Contacts.TIMES_CONTACTED, // 3  
        Contacts.CONTACT_PRESENCE, // 4  
        Contacts.PHOTO_ID, // 5  
        Contacts.LOOKUP_KEY, // 6  
        Contacts.HAS_PHONE_NUMBER, // 7  
    };  
}
```

```

static final int SUMMARY_ID_COLUMN_INDEX = 0;
static final int SUMMARY_NAME_COLUMN_INDEX = 1;
static final int SUMMARY_STARRED_COLUMN_INDEX = 2;
static final int SUMMARY_TIMES_CONTACTED_COLUMN_INDEX = 3;
static final int SUMMARY_PRESENCE_STATUS_COLUMN_INDEX = 4;
static final int SUMMARY_PHOTO_ID_COLUMN_INDEX = 5;
static final int SUMMARY_LOOKUP_KEY = 6;
static final int SUMMARY_HAS_PHONE_COLUMN_INDEX = 7;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String select = "(" + Contacts.DISPLAY_NAME + " NOTNULL) AND (
        + Contacts.HAS_PHONE_NUMBER + "=1) AND (
        + Contacts.DISPLAY_NAME + " != " ))";
    Cursor c =
        getContentResolver().query(Contacts.CONTENT_URI,
CONTACTS_SUMMARY_PROJECTION, select,
        null, Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
    startManagingCursor(c);
    ContactListItemAdapter adapter = new ContactListItemAdapter(this,
R.layout.quick_contacts, c);
    setListAdapter(adapter);

}

private final class ContactListItemAdapter extends ResourceCursorAdapter {
    public ContactListItemAdapter(Context context, int layout, Cursor c) {
        super(context, layout, c);
    }

    @Override
    public void bindView(View view, Context context, Cursor cursor) {
        final ContactListItemCache cache = (ContactListItemCache) view.getTag();
        TextView nameView = cache.nameView;
        QuickContactBadge photoView = cache.photoView;
        // Set the name
        cursor.copyStringToBuffer(SUMMARY_NAME_COLUMN_INDEX,
cache.nameBuffer);
        int size = cache.nameBuffer.sizeCopied;
        cache.nameView.setText(cache.nameBuffer.data, 0, size);
        final long contactId = cursor.getLong(SUMMARY_ID_COLUMN_INDEX);
    }
}

```

```
        final String lookupKey = cursor.getString(SUMMARY_LOOKUP_KEY);
        cache.photoView.assignContactUri(Contacts.getLookupUri(contactId,
lookupKey));
    }

    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent) {
        View view = super.newView(context, cursor, parent);
        ContactListItemCache cache = new ContactListItemCache();
        cache.nameView = (TextView) view.findViewById(R.id.name);
        cache.photoView = (QuickContactBadge) view.findViewById(R.id.badge);
        view.setTag(cache);

        return view;
    }
}

final static class ContactListItemCache {
    public TextView nameView;
    public QuickContactBadge photoView;
    public CharArrayBuffer nameBuffer = new CharArrayBuffer(128);
}
}
```

Scroller

译者署名：pengyouhong
版本：Android 2.3 r1

public class Scroller extends Object

java.lang.Object
 android.widget.Scrolle

类概述

这个类封装了滚动操作。滚动的持续时间可以通过构造函数传递，并且可以指定滚动动作的持续的最长时间。经过这段时间，滚动会自动定位到最终位置，并且通过 `computeScrollOffset()` 会得到的返回值为 `false`，表明滚动动作已经结束。

构造函数

public Scroller (Context context)

使用缺省的持续时间和动画插入器创建一个 Scroller。（译者注：interpolator 这里翻译为动画插入器，见[这里](#)。）

public Scroller (Context context, Interpolator interpolator)

根据指定的动画插入器创建一个 Scroller，如果指定的动画插入器为空，则会使用缺省的动画插入器（粘滞 viscous）创建。

公共方法

public void abortAnimation ()

停止动画。与 `forceFinished(boolean)` 相反，Scroller 滚动到最终 x 与 y 位置时中止动画。

参见

[forceFinished\(boolean\)](#)

public boolean computeScrollOffset ()

当想要知道新的位置时，调用此函数。如果返回 `true`，表示动画还没有结束。位置改变以提供一个新的位置。

public void extendDuration (int extend)

延长滚动动画时间。此函数允许当使用 `setFinalX(int)` 或 `setFinalY(int)` 时，滚动动作持续更长时间并且滚动更长距离。

参数

extend 滚动事件延长的时间，以毫秒为单位

参见

[setFinalX\(int\)](#)

[setFinalY\(int\)](#)

public void fling (int startX, int startY, int velocityX, int velocityY, int minX, int maxX, int minY,

int maxY()

在 fling (译者注：快滑，用户按下触摸屏、快速移动后松开) 手势基础上开始滚动。
滚动的距离取决于 fling 的初速度。

参数

startX 滚动起始点 X 坐标
startY 滚动起始点 Y 坐标
velocityX 当滑动屏幕时 X 方向初速度，以每秒像素数计算
velocityY 当滑动屏幕时 Y 方向初速度，以每秒像素数计算
minX X 方向的最小值，scroller 不会滚过此点。
maxX X 方向的最大值，scroller 不会滚过此点。
minY Y 方向的最小值，scroller 不会滚过此点。
maxY Y 方向的最大值，scroller 不会滚过此点。

public final void forceFinished (boolean finished)

强制终止的字段到特定值。（译者注：立即停止滚动？）

参数

finished 新的结束值

public final int getCurrX ()

返回当前滚动 X 方向的偏移

返回值

距离原点 X 方向的绝对值

public final int getCurrY ()

返回当前滚动 Y 方向的偏移

返回值

距离原点 Y 方向的绝对值

public final int getDuration ()

返回滚动事件的持续时间，以毫秒计算。

返回值

滚动持续的毫秒数

public final int getFinalX ()

返回滚动结束位置。仅针对“fling”手势有效

返回值

最终位置 X 方向距离原点的绝对距离

public final int getFinalY ()

返回滚动结束位置。仅针对“fling”操作有效

返回值

最终位置 Y 方向距离原点的绝对距离

public final int getStartX ()

返回滚动起始点的 X 方向的偏移

返回值

起始点在 X 方向距离原点的绝对距离

public final int getStartX ()

返回滚动起始点的 Y 方向的偏移

返回值

起始点在 Y 方向距离原点的绝对距离

public final boolean isFinished ()

返回 scroller 是否已完成滚动。

返回值

停止滚动返回 true，否则返回 false

public void setFinalX (int newX)

设置 scroller 的 X 方向终止位置

参数

newX 新位置在 X 方向距离原点的绝对偏移。

参见

[extendDuration\(int\)](#)

[setFinalY\(int\)](#)

public void setFinalY (int newY)

设置 scroller 的 Y 方向终止位置

参数

newY 新位置在 Y 方向距离原点的绝对偏移。

参见

[extendDuration\(int\)](#)

[setFinalY\(int\)](#)

public void startScroll (int startX, int startY, int dx, int dy)

以提供的起始点和将要滑动的距离开始滚动。滚动会使用缺省值 250ms 作为持续时间。

参数

startX 水平方向滚动的偏移值，以像素为单位。负值表明滚动将向左滚动

startY 垂直方向滚动的偏移值，以像素为单位。负值表明滚动将向上滚动

dx 水平方向滑动的距离，负值会使滚动向左滚动

dy 垂直方向滑动的距离，负值会使滚动向上滚动

public void startScroll (int startX, int startY, int dx, int dy, int duration)

以提供的起始点和将要滑动的距离开始滚动。

参数

startX 水平方向滚动的偏移值，以像素为单位。负值表明滚动将向左滚动

startY 垂直方向滚动的偏移值，以像素为单位。负值表明滚动将向上滚动

dx 水平方向滑动的距离，负值会使滚动向左滚动

dy 垂直方向滑动的距离，负值会使滚动向上滚动
duration 滚动持续时间，以毫秒计。

public int timePassed ()

返回自滚动开始经过的时间

返回值

经过时间以毫秒为单位

补充

文章精选

[Scroller 粗浅理解](#)

[ScrollView – scrolling TextView for Android](#)

示例代码

创建工程 MyScroler，或者将下类名“MyScroler”改为自己创建的工程，将下面代码直接覆盖生成的. java 文件运行即可

```
package my.Scroller;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.Scroller;

public class MyScroler extends Activity {
    /** Called when the activity is first created. */
    LinearLayout lay1, lay2, lay;
    private Scroller mScroller;
    private boolean s1, s2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mScroller = new Scroller(this);
        lay1 = new LinearLayout(this) {
            @Override
            public void computeScroll() {
                if (mScroller.computeScrollOffset()) {
                    scrollTo(mScroller.getCurrX(), 0);
                    postInvalidate();
                }
            }
        };
        lay2 = new LinearLayout(this) {
```

```

@Override
public void computeScroll() {
    if (mScroller.computeScrollOffset()) {
        // mScrollX = mScroller.getCurrX();
        scrollTo(mScroller.getCurrX(), 0);
        postInvalidate();
    }
}
};

lay1.setBackgroundColor(this.getResources().getColor(android.R.co
lor.darker_gray));

lay2.setBackgroundColor(this.getResources().getColor(android.R.co
lor.white));
    lay = new LinearLayout(this);
    lay.setOrientation(LinearLayout.VERTICAL);
    LinearLayout.LayoutParams p0 = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,L
inearLayout.LayoutParams.FILL_PARENT);
    this.setContentView(lay, p0);

    LinearLayout.LayoutParams p1 = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,L
inearLayout.LayoutParams.FILL_PARENT);
    p1.weight=1;
    lay.addView(lay1,p1);
    LinearLayout.LayoutParams p2 = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,L
inearLayout.LayoutParams.FILL_PARENT);
    p2.weight=1;
    lay.addView(lay2,p2);
    Button tx = new Button(this);
    Button tx2 = new Button(this);
    tx.setText("Button1");
    tx2.setText("Button2");
    tx.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if(!s1){
                mScroller.startScroll(0, 0, 5, 10, 10);
                s1 = true;
            }else{
                mScroller.startScroll(0, 0, -50, -10,10);
            }
        }
    });
}

```

```
        s1 = false;
    }
}

} );
tx2.setOnClickListener(new OnClickListener(){
    @Override
    public void onClick(View v) {
        if(!s2){
            mScroller.startScroll(0, 0, 5, 20,10);
            s2=true;
        }else{
            mScroller.startScroll(20, 20, -50, -20,10);
            s2=false;
        }
    }
});
lay1.addView(tx);
lay2.addView(tx2);
}
}
```

ScrollView

译者署名: pengyouhong
版本: Android 2.3 r1

结构

继承关系

`public class ScrollView extends FrameLayout`

```
java.lang.Object
  android.view.View
    android.view.ViewGroup
      android.widget.FrameLayout
        android.widget.ScrollView
```

类概述

一种可供用户滚动的层次结构布局容器，允许显示比实际多的内容。ScrollView 是一种 [FrameLayout](#)，意味需要在其上放置有自己滚动内容的子元素。子元素可以是一个复杂的对象的布局管理器。通常用的子元素是垂直方向的 [LinearLayout](#)，显示在最上层的垂直方向可以让用户滚动的箭头。

[TextView](#) 类也有自己的滚动功能，所以不需要使用 ScrollView，但是只有两个结合使用，才能保证显示较多内容时候的效率。但只有两者结合使用才可以实现在一个较大的容器中一个文本视图效果。

ScrollView 只支持垂直方向的滚动。

构造函数

`public ScrollView (Context context)`

创建一个默认属性的 ScrollView 实例。

`public ScrollView (Context context, AttributeSet attrs)`

创建一个带有 attrs 属性的 ScrollView 实例。

`public ScrollView (Context context, AttributeSet attrs, int defStyle)`

创建一个带有 attrs 属性，并且指定其默认样式的 ScrollView 实例。

公共方法

`public void addView (View child)`

添加子视图。如果事先没有给子视图设置 layout 参数，会采用当前 ViewGroup 的默认参数来设置子视图。

参数

`child` 所添加的子视图

`public void addView (View child, int index)`

添加子视图。如果事先没有给子视图设置 layout 参数，会采用当前 ViewGroup 的默认

参数来设置子视图。

参数

- child** 所添加的子视图
- index** 添加子视图的位置

```
public void addView (View child, int index, ViewGroup.LayoutParams params)
```

根据指定的 `layout` 参数添加子视图

参数

- child** 所添加的子视图
- index** 添加子视图的位置
- params** 为子视图设置的 `layout` 参数

```
public void addView (View child, ViewGroup.LayoutParams params)
```

根据指定的 `layout` 参数添加子视图。

参数

- child** 所添加的子视图
- params** 为子视图设置的 `layout` 参数

```
public boolean arrowScroll (int direction)
```

响应点击上下箭头时对滚动条滚动的处理。

参数

- direction** 按下的箭头所对应的方向

返回值

如果我们处理（消耗）了此事件返回 `true`，否则返回 `false`。

```
public void computeScroll ()
```

被父视图调用，用于必要时候对其子视图的值 (`mScrollX` 和 `mScrollY`) 进行更新。典型的情况如：父视图中某个子视图使用一个 [Scroller](#) 对象来实现滚动操作，会使得此方法被调用。

```
public boolean dispatchKeyEvent (KeyEvent event)
```

发送一个 `key` 事件给当前焦点路径的下一个视图。此焦点路径从视图树的顶层执行直到当前焦点视图。如果此视图为焦点视图，将为自己发送。否则，会为当前焦点路径的下一个节点发送。此方法也会激起一个 `key` 监听器。

参数

- event** 发送的 `key` 事件

返回值

事件被处理返回 `true`，否则返回 `false`。

```
public void draw (Canvas canvas)
```

手动绘制视图（及其子视图）到指定的画布 (Canvas)。这个视图必须在调用这个函数之前做好了整体布局。当实现一个视图时，不需要继承这个方法；相反，你应该实现 [onDraw\(Canvas\)](#) 方法。

参数

canvas 绘制视图的画布

public boolean executeKeyEvent (KeyEvent event)

当接收到 key 事件时，用户可以调用此函数来使滚动视图执行滚动，类似于处理由视图体系发送的事件。

参数

event 需要执行 key 的事件

返回值

事件被处理返回 true，否则返回 false。

public void fling (int velocityY)

滚动视图的滑动（fling）手势。（译者注：[如何监听 android 的屏幕滑动停止事件](#)）

参数

velocityY Y 方向的初始速率。正值表示手指/光标向屏幕下方滑动，而内容将向上滚动。

public boolean fullScroll (int direction)

对响应“home/end”短按时响应滚动处理。此方法将视图滚动到顶部或者底部，并且将焦点置于新的可视区域的最顶部/最底部组件。若没有适合的组件做焦点，当前的 ScrollView 会收回焦点。

参数

direction 滚动方向：[FOCUS_UP](#) 表示视图向上滚动；[FOCUS_DOWN](#) 表示视图向下滚动

返回值

若 key 事件被消耗(consumed)返回 true，其他情况返回 false。

public int getMaxScrollAmount ()

返回值

当前滚动视图响应箭头事件能够滚动的最大数。

public boolean isFillViewport ()

指示当前 ScrollView 的内容是否被拉伸以填充视图可视范围（译者注：viewport 可视范围，参见[决定 Scrollviewer 里面 Control 的可视范围](#)）。

返回值

内容填充视图返回 true，否则返回 false。

public boolean isSmoothScrollingEnabled ()

返回值

按箭头方向滚动时，是否显示滚动的平滑效果。

public boolean onInterceptTouchEvent (MotionEvent ev)

实现此方法是为了拦截所有触摸屏幕时的运动事件。可以像处理发送给子视图的事件一样去监视这些事件，并且获取当前手势在任意点的 ownership

使用此方法时候需要注意，因为它与 [View.onTouchEvent\(MotionEvent\)](#) 有相当复杂的交

互，并且前提需要正确执行 [View.onTouchEvent\(MotionEvent\)](#)。事件将按照如下顺序接收到：

1. 收到 down 事件
2. Down 事件或者由视图组的一个子视图处理，或者被用户自己的 `onTouchEvent()` 方法处理；此处理意味你应该执行 `onTouchEvent()` 时返回 `true`，这样才能继续看到剩下的手势（取代找一个父视图处理）。如果 `onTouchEvent()` 返回 `true` 时，你不会收到 `onInterceptTouchEvent()` 的任何事件并且所有对触摸的处理必须在 `onTouchEvent()` 中发生。
3. 如果此方法返回 `false`，接下来的事件（`up to and including the final up`）将最先被传递至此，然后是目标的 `onTouchEvent()`。
4. 如果返回 `true`，将不会收到以下任何事件：目标 `view` 将收到同样的事件但是会伴随 [ACTION_CANCEL](#)，并且所有的更进一步的事件将会传递到你自己的 `onTouchEvent()` 方法中而不会再在这里出现。

参数

`ev` 体系向下发送的动作事件

返回值

如果将运动事件从子视图中截获并且通过 `onTouchEvent()` 发送到当前 `ViewGroup`，返回 `true`。当前目标将会收到 `ACTION_CANCEL` 事件，并且不再会有其他消息传递到此。

(译者注：[onInterceptTouchEvent 和 onTouchEvent 调用时序](#))

public boolean onTouchEvent (MotionEvent ev)

执行此方法为了处理触摸屏幕的运动事件。

参数

`ev` 运动事件

返回值

事件被处理返回 `true`，其它返回 `false`。

public boolean pageScroll (int direction)

响应短按“page up/ down”时候对滚动的处理。此方法将向上或者向下滚动一屏，并且将焦点置于新可视区域的最上/最下。如果没有适合的 `component` 作为焦点，当前 `scrollView` 将收回焦点。

参数

`direction` 滚动方向：[FOCUS_UP](#) 表示向上翻一页，[FOCUS_DOWN](#) 表示向下翻一页。

返回值

此 key 事件被消耗(`consumed`)返回 `true`，其他返回 `false`。

public void requestChildFocus (View child, View focused)

当父视图的一个子视图的要获得焦点时，调用此方法。

参数

`child` 要获得焦点的父视图的子视图。此视图包含了焦点视图。如果没有特殊需求，此视图实际上就是焦点视图。

`focused` 子视图的子孙视图并且此子孙视图是真正的焦点视图

```
public boolean requestChildRectangleOnScreen (View child, Rect rectangle, boolean immediate)
```

当组里的某个子视图需要被定位在屏幕的某个矩形范围时，调用此方法。重载此方法的 [ViewGroup](#) 可确认以下几点：

- * 子项目将是组里的直系子项
- * 矩形将在子项目的坐标体系中

重载此方法的 [ViewGroup](#) 应该支持以下几点：

- * 若矩形已经是可见的，则没有东西会改变
- * 为使矩形区域全部可见，视图将可以被滚动显示

参数

child	发出请求的子视图
rectangle	子项目坐标系内的矩形，即此子项目希望在屏幕上的定位
immediate	设为 true，则禁止动画和平滑移动滚动条
返回值	进行了滚动操作的这个组（group），是否处理此操作。

```
public void requestLayout ()
```

当有改变引起当前视图重新布局时，调用此函数。它将规划一个视图树的 layout 路径。

```
public void scrollTo (int x, int y)
```

设置当前视图滚动到的位置。此函数会引起对 [onScrollChanged\(int, int, int, int\)](#) 函数的调用并且会让视图更新。

当前版本取消了在子视图中的滚动。

参数

x	滚动到的 X 位置
y	滚动到的 Y 位置

```
public void setFillViewport (boolean fillViewport)
```

设置当前滚动视图是否将内容高度拉伸以填充 **视图可视范围**（译者注：viewport 可视范围，参见[决定 Scrollviewer 里面 Control 的可视范围](#)）。

参数

fillViewport 设置为 true 表示拉伸内容高度来适应视口边界；其他设为 false。

```
public void setOverScrollMode (int mode)
```

为视图设置 over-scroll 模式。有效的 over-scroll 模式有 [OVER_SCROLL_ALWAYS](#)（缺省值），[OVER_SCROLL_IF_CONTENT_SCROLLS](#)（只允许当视图内容大过容器时，进行 over-scrolling）和 [OVER_SCROLL_NEVER](#)。只有当视图可以滚动时，此项设置才起作用。

（译者注：这个函数是 2.3 r1 中新增的，API Level 9。关于 over-scroll 这里译为弹性滚动，即，参见帖子：[类似 iPhone 的弹性 ListView 滚动](#)）

参数

mode The new over-scroll mode for this view.

```
public void setSmoothScrollingEnabled (boolean smoothScrollingEnabled)
```

用来设置箭头滚动是否可以引发视图滚动。

参数

`smoothScrollingEnabled` 设置箭头滚动是否可以引起内容的滚动的 bool 值

`public final void smoothScrollBy (int dx, int dy)`

类似于 [scrollBy\(int, int\)](#)，但是滚动时候是平缓的而不是立即滚动到某处。

参数

`dx` 在 X 方向滚动的像素数

`dy` 在 Y 方向滚动的像素数

`public final void smoothScrollTo (int x, int y)`

类似于 [scrollTo\(int, int\)](#)，但是滚动时候是平缓的而不是立即滚动到某处。

参数

`x` 要滚动到位置的 X 坐标

`y` 要滚动到位置的 Y 坐标

受保护方法

`protected int computeScrollDeltaToGetChildRectOnScreen (Rect rect)`

计算 X 方向滚动的总合，以便在屏幕上显示子视图的完整矩形（或者，若矩形宽度超过屏幕宽度，至少要填满第一个屏幕大小）。

参数

`rect` 矩形

返回值

滚动差值

`protected int computeVerticalScrollOffset ()`

计算垂直方向滚动条的滑块的偏移。此值用来计算滚动条轨迹的滑块的位置。

范围可以以任意单位表示，但是必须与 [computeVerticalScrollRange\(\)](#) 和 [computeVerticalScrollExtent\(\)](#) 的单位一致。

缺省的偏移是在当前视图滚动的偏移。

返回值

滚动条的滑块垂直方向的偏移。

`protected int computeVerticalScrollRange ()`

滚动视图的可滚动范围是所有子元素的高度。

返回值

由垂直方向滚动条代表的所有垂直范围，缺省的范围是当前视图的画图高度。

`protected float getBottomFadingEdgeStrength ()`

返回滚动底部的能见度。能见度的值的范围是 0.0（没有消失）到 1.0（完全消失）之间。缺省的执行返回值为 0.0 或者 1.0，而不是他们中间的某个值。滚动时子类需要重载这个方法来提供一个平缓的渐隐的实现。

返回值

滚动底部能见度，值的范围在浮点数 0.0f 到 1.0f 之间。

`protected float getTopFadingEdgeStrength ()`

返回滚动顶部的能见度。能见度的值的范围是 0.0 (没有消失) 到 1.0 (完全消失) 之间。缺省的执行返回值为 0.0 或者 1.0，而不是他们中间的某个值。滚动时子类需要重载这个方法来提供一个平缓的渐隐的实现。

返回值

滚动顶部能见度，值的范围在浮点数 0.0f 到 1.0f 之间。

`protected void measureChild (View child, int parentWidthMeasureSpec, int parentHeightMeasureSpec)`

要求当前视图的一个子视图测量自己，同时兼顾到当前视图的 MeasureSpec 的要求和它的空白。子视图必须有 MarginLayoutParams。比较复杂的工作是在 getChildMeasureSpec 中完成的。

参数

`child` 需要自己测量的子视图

`parentWidthMeasureSpec` 当前视图要求的宽度

`parentHeightMeasureSpec` 当前视图要求的宽度

`protected void measureChildWithMargins (View child, int parentWidthMeasureSpec, int widthUsed, int parentHeightMeasureSpec, int heightUsed)`

要求当前视图的一个子视图测量自己，同时兼顾到当前视图的 MeasureSpec 的要求和它的空白和边界。子视图必须有 MarginLayoutParams。比较复杂的工作是在 getChildMeasureSpec 中完成的。

参数

`child` 需要测量的子视图

`parentWidthMeasureSpec` 当前视图要求的宽度

`widthUsed` 水平方向上由父视图使用的空白 (也可能是视图的其他子视图使用的)

`parentHeightMeasureSpec` 当前视图要求的宽度

`heightUsed` 垂直方向上由父视图使用的空白 (也可能是视图的其他子视图使用的)

`protected void onLayout (boolean changed, int l, int t, int r, int b)`

当前视图需要为子视图分配大小和位置时候调用，子类继承必须要重载此方法并调用自己子视图的 layout 函数。

参数

`changed` 当前视图的新的大小或者位置

`l` 相对父视图，左边界位置

`t` 相对父视图，上边界位置

`r` 相对父视图，右边界位置

`b` 相对父视图，下边界位置

`protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec)`

测量视图以确定其内容宽度和高度。此方法被 measure(int, int) 调用。需要被子类重写以

提供对其内容准确高效的测量。

约定：当重写此方法时，你必须调用 `setMeasuredDimension(int, int)` 来保存当前视图 view 的宽度和高度。不成功调用此方法将会导致一个 `IllegalStateException` 异常，是由 `measure(int, int)` 抛出。所以调用父类的 `onMeasure(int, int)` 方法是必须的。

父类的实现是以背景大小为默认大小，除非 `MeasureSpec`（测量细则）允许更大的背景。子类可以重写 `onMeasure(int, int)` 以对其内容提供更佳的尺寸。

如果此方法被重写，那么子类的责任是确认测量高度和测量宽度要大于视图 view 的最小宽度和最小高度（`getSuggestedMinimumHeight()` 和 `getSuggestedMinimumWidth()`），使用这两个方法可以取得最小宽度和最小高度。

参数

`widthMeasureSpec` 受主窗口支配的水平空间要求。这个需求通过 `View.MeasureSpec`.进行编码。

`heightMeasureSpec` 受主窗口支配的垂直空间要求。这个需求通过 `View.MeasureSpec`.进行编码。

`protected void onOverScrolled (int scrollX, int scrollY, boolean clampedX, boolean clampedY)`
被 [overScrollBy\(int, int, int, int, int, int, int, boolean\)](#) 调用，来对一个 over-scroll 操作的结果进行响应。（译者注：这个函数是 2.3 r1 中新增的，API Level 9）

参数

`scrollX` 新的 X 滚动像素值

`scrollY` 新的 Y 滚动像素值

`clampedX` 当 `scrollX` 被 over-scroll 的边界限制时，值为 true

`clampedY` 当 `scrollY` 被 over-scroll 的边界限制时，值为 true

`protected boolean onRequestFocusInDescendants (int direction, Rect previouslyFocusedRect)`

当在滚动视图的子视图中查找焦点视图时，需要注意不要将焦点设置在滚动出屏幕外的控件上。此方法会比执行缺省的 [ViewGroup](#) 代价高，否则此行为也会设置为缺省

参数

`direction` 指定下列常量之一： `FOCUS_UP`, `FOCUS_DOWN`, `FOCUS_LEFT`, `FOCUS_RIGHT`

`previouslyFocusedRect` 能够给出一个较好的提示的矩形（当前视图的坐标系统）表示焦点从哪里得来。如果没有提示为 `null`。

返回值

是否取得了焦点

`protected void onSizeChanged (int w, int h, int oldw, int oldh)`

布局期间当视图的大小发生改变时调用。如果只是添加到视图，调用时显示的是旧值 0。（译者注：也就是添加到视图时，`oldw` 和 `oldh` 返回的是 0）。

参数

`w` 视图当前宽度

`h` 视图当前高度

`oldw` 视图改变前的宽度

`oldh` 视图改变前的高度

补充

文章精选

[Android ApiDemos/ScrollView2 添加自动滚动和智能焦点切换](#)

[Android 学习指南][使用 ScrollView 实现滚动效果](#)

[Android 中 ScrollView 与 ListView 共用问题的解决方案](#)

SeekBar

译者：madgoat

博客：<http://madgoat.cn/>

2010-10-22

版本：Android 2.2 r1

public class SeekBar extends AbsSeekBar

java.lang.Object

[android.view.View](#)

 android.widget.ProgressBar

 android.widget.AbsSeekBar

 android.widget.SeekBar

概述



SeekBar 是 ProgressBar 的扩展，在其基础上增加了一个可滑动的滑片(注：就是那个可拖动的图标)。用户可以触摸滑片并向左或向右拖动，再或者可以使用方向键都可以设置当前的进度等级。不建议把可以获取焦点的 widget 放在 SeekBar 的左边或右边。

SeekBar 可以附加一个 [SeekBar.OnSeekBarChangeListener](#) 以获得用户操作的通知。

内部类

接口 [SeekBar.OnSeekBarChangeListener](#)

一个回调函数用来当进度等级发生改变时通知客户端

XML 属性

属性名称	描述
android:thumb	SeekBar 上绘制的 thumb (可拖动的那个图标)

公共方法

public void setOnSeekBarChangeListener (SeekBar.OnSeekBarChangeListener l)

设置一个监听器以接受 seekbar 进度改变时的通知。同时提供用户在 SeekBar 上开始和停止触摸手势时的通知。

参数

| SeekBar 的通知监听对象

参见

[SeekBar.OnSeekBarChangeListener](#)

补充

文章链接

在 android 里做一个竖着的 seekbar

<http://blog.csdn.net/saintswordsmen/archive/2010/01/23/5248233.aspx>

Android UI 设计 SeekBar 拖动条用法

<http://www.pocketdigi.com/20100813/36.html>

例子

```
Java:  
import android.app.Activity;  
import android.os.Bundle;  
import android.widget.SeekBar;  
import android.widget.Toast;  
import android.widget.SeekBar.OnSeekBarChangeListener;  
  
/**  
 * @author madgoat.fan  
 *  
 */  
public class SeekBarDemo extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        // TODO Auto-generated method stub  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.seekbardemo);  
  
        final SeekBar seekBar1 = (SeekBar) this.findViewById(R.id.seekBar1);  
        seekBar1.setOnSeekBarChangeListener(new OnSeekBarChangeListener()  
    }  
  
    @Override  
    public void onStopTrackingTouch(SeekBar seekBar) {  
        // TODO Auto-generated method stub  
  
    }  
  
    @Override  
    public void onStartTrackingTouch(SeekBar seekBar) {  
        // TODO Auto-generated method stub  
  
    }  
  
    @Override  
    public void onProgressChanged(SeekBar seekBar, int progress,  
        boolean fromUser) {  
        // TODO Auto-generated method stub  
    }
```

```
        Toast.makeText(SeekBarDemo.this,
                      String.valueOf(seekBar1.getProgress()),
                      Toast.LENGTH_SHORT).show();
    }
});
```

}

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"
>
    <SeekBar android:id="@+id/seekBar1" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:max="100"
    ></SeekBar>
</LinearLayout>
```

整理人：农民伯伯

SeekBar.OnSeekBarChangeListener

译者：madgoat

博客：<http://madgoat.cn/>

2010-10-22

版本：Android 2.2 r1

public static interface SeekBar.OnSeekBarChangeListener

android.widget.SeekBar.OnSeekBarChangeListener

概述

当进度改变后用于通知客户端的回调函数。这包括用户通过手势、方向键或轨迹球触发的改变，以及编程触发的改变。

公共方法

public abstract void onProgressChanged (SeekBar seekBar, int progress, boolean fromUser)

通知进度已经被修改。客户端可以使用 fromUser 参数区分用户触发的改变还是编程触发的改变。

参数

seekBar 当前被修改进度的 SeekBar

progress 当前的进度值。此值的取值范围为 0 到 max 之间。Max 为用户通过 setMax(int) 设置的值，默认为 100

fromUser 如果是用户触发的改变则返回 True

public abstract void onStartTrackingTouch (SeekBar seekBar)

通知用户已经开始一个触摸拖动手势。客户端可能需要使用这个来禁用 seekbar 的滑动功能。

参数

seekBar 触摸的 SeekBar

public abstract void onStopTrackingTouch (SeekBar seekBar)

通知用户触摸手势已经结束。客户端可能需要使用这个来启用 seekbar 的滑动功能。

参数

seekBar 触摸的 SeekBar

整理人：农民伯伯

SimpleAdapter

译者署名： 德罗德

译者链接：<http://sparkrico.javaeye.com>

翻译时间： 2010-11-03

版本： Android 2.2 r1

结构

继承关系

public interface SpinnerAdapter extends Adapter

```
java.lang.Object  
    android.widget.BaseAdapter  
        android.widget.SpinnerAdapter
```

类概述

这是一个简单的适配器，可以将静态数据映射到 XML 文件中定义好的视图。你可以指定数据支持的列表如 ArrayList 组成的 Map。在 ArrayList 中的每个条目对应 List 中的一行。Maps 包含每行数据。你可以指定一个定义了被用于显示行的视图 XML 文件，通过关键字映射到指定的视图。绑定数据到视图分两个阶段，首先，如果一个 SimpleAdapter.ViewBinder 是有效的，`setViewValue(android.view.View, Object, String)` 将被调用。如果返回值是真，绑定完成了。如果返回值为假，下面的视图将按以下顺序去处理：

- 一个实现了 Checkable 的视图（例如 CheckBox），期望绑定值是一个布尔类型。
- TextView 期望绑定值是一个字符串类型，通过调用 `setViewText(Textview, String)` 绑定。
- ImageView 期望绑定值是一个资源 id 或者一个字符串，通过调用 `setViewImage(ImageView, int)` 或 `setViewImage(ImageView, String)`。

如果没有一个合适的绑定发生将会抛出 IllegalStateException。

构造函数

```
public SimpleAdapter (Context context, List<? extends Map<String, ?>> data, int  
resource, String[] from, int[] to)
```

构造函数

参数

context 关联 SimpleAdapter 运行着的视图的上下文。

data 一个 Map 的列表。在列表中的每个条目对应列表中的一行，应该包含所有在 from 中指定的条目

resource 一个定义列表项目的视图布局的资源唯一标识。布局文件将至少应包含哪些在 to 中定义了的名称。

from 一个将被添加到 Map 上关联每一个项目的列名称的列表

to 应该在参数 from 显示列的视图。这些应该全是 TextView。在列表中最初的 N 视图是从参数 from 中最初的 N 列获取的值。

公共方法

```
public int getCount ()
```

获取数据集中记录总行数

```
public View getDropDownView (int position, View convertView, ViewGroup parent)
```

获得一个在指定位置上显示下拉弹出数据的视图。

参数

position 想得到项目视图的索引

convertView 如果可能旧有的视图重新使用。注解：在使用之前应该检查这个视图不是空的并且类型合适。如果转换视图显示正确的数据是不可能的，这个方法能够创建一个新的视图

parent 视图最终将依附的父对象

返回值

一个对应指定位置的数据的视图。

```
public Filter getFilter ()
```

返回一个可以通过一种过滤模式来约束数据的过滤器。

这个方法通常被 Adapter 类实现。

返回值

一个用于约束数据的过滤器

```
public abstract Object getItem (int position)
```

获取数据集中指定位置上的数据项目

参数

position 在 adapter 中我们想得到项目的位置。

返回值

指定位置上的数据。

```
public abstract long getItemId (int position)
```

获取数据集中指定位置上的行 ID。

参数

position 在 adapter 中我们想得到的行 ID 的项目的位置。

返回值

指定位置上的数据。

```
public abstract View getView (int position, View convertView, ViewGroup parent)
```

获取一个显示数据集中指定位置数据段视图。可以手动创建视图，或者从 XML 设计文件填充。当视图从 XML 设计文件填充时，父视图（如 GridView, ListView 等）将接受默认的设计参数，除非使用 inflate(int, android.view.ViewGroup, boolean) 去指定一个根视图和防止依附于根视图。

参数

position 我们想要的在 adapter 中的数据项目的位置

convertView 如果可能旧有的视图重新使用。注解：在使用之前应该检查这个视图不是空的并且类型合适。如果转换视图显示正确的数据是不可能的，这个方法能够创建一个新的视图

parent 视图最终将依附的父对象。

返回值

一个在指定位置上相应的数据的视图。

public SimpleAdapter.ViewBinder getViewBinder ()

返回被用来绑定数据到视图的 SimpleAdapter.ViewBinder。

返回值

一个 ViewBinder，如果 binder 不存在则返回 null

参见

[setViewBinder \(android.widget.SimpleAdapter.ViewBinder\)](#)

public void setDropDownViewResource (int resource)

设置创建下拉视图的布局资源

参数

resource 定义下拉视图的布局资源

参见

[getDropDownView \(int, android.view.View, android.view.ViewGroup\)](#)

public void setViewBinder (SimpleAdapter.ViewBinder viewBinder)

设置 binder 用于绑定数据到视图

参数

viewBinder 用于绑定数据到视图的 binder 可以设置为 null，可用于删除存在的 binder

参见

[getViewBinder \(\)](#)

public void setViewImage (ImageView v, int value)

调用 bindView 去给 ImageView 设置图像，但只有当 ViewBinder 不存在或者如果存在的 ViewBinder 无法处理绑定到一个 ImageView 时才调用。如果提供的数据是一个整形时，[setViewImage\(ImageView, String\)](#)方法将被本方法替代

参数

v 接收图像的 ImageView

value 从数据集获取数据到值

参见

[setViewImage \(ImageView, String\)](#)

public void setViewImage (ImageView v, String value)

调用 bindView 去给 ImageView 设置图像，但只有当 ViewBinder 不存在或者如果存在的 ViewBinder 无法处理绑定到一个 ImageView 时才调用。默认的，这个值被作为一个图像资源来对待。如果这个值作为一个图像的 Uri 来使用。如果提供的数据不是一个整形时，[setViewImage\(ImageView, int\)](#)方法将被本方法替代

参数

v 接收图像的 ImageView

value 从数据集获取数据到值

参见

[setViewImage \(ImageView, int\)](#)

public void setViewText (TextView v, String text)

调用 bindView 去给 TextView 设置文本，但只有当 ViewBinder 不存在或者如果存在的 ViewBinder 无法处理绑定到一个 TextView 时才调用

参数

v 将接收文本的 TextView

text 被设置到 TextView 的文本

补充

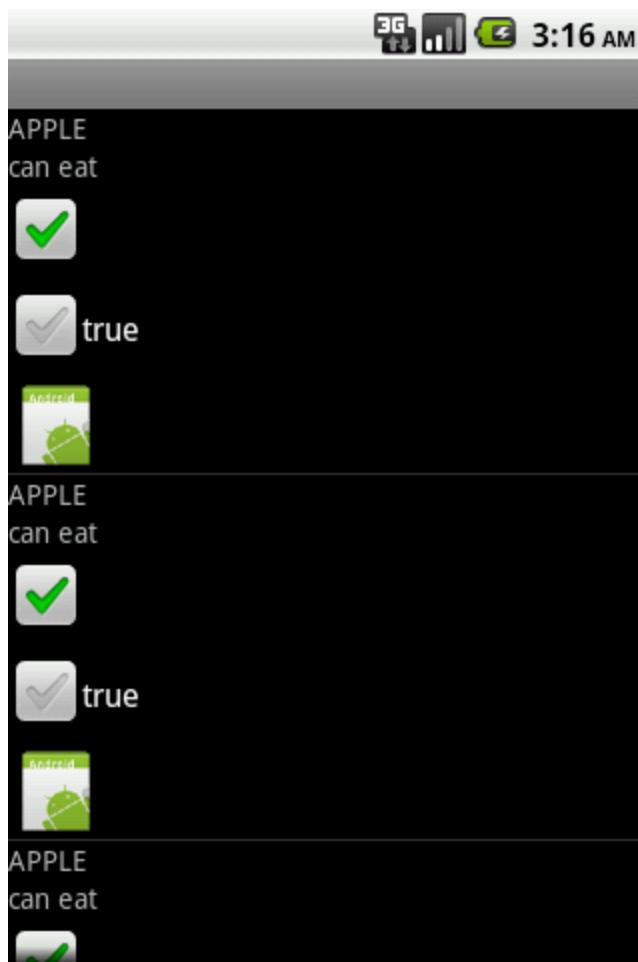
文章精选

[简约而不简单——Android SimpleAdapter](#)

[android listview 组件之 ArrayAdapter, SimpleAdapter](#)

[Android 用 simpleAdapter 来直接显示 BMP 图片 \(有 ViewBinder 用法\)](#)

示例代码



```
private ListView lv;
private List<Map<String, String>> data;
private SimpleAdapter sAdapter;
```

```
lv = (ListView) findViewById(R.id.listview);
data = new ArrayList<Map<String, Object>>();
for (int i = 0; i < 10; i++) {
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("title", "APPLE");
    map.put("content", "can eat");
    map.put("check", true);
    map.put("check1", true);
    map.put("image", R.drawable.icon);
    data.add(map);
}
sAdapter = new SimpleAdapter(getApplicationContext(),
        data,
        R.layout.item,
        new
String[] {"title", "content", "check", "check1", "image"},
        new
int[]{R.id.title, R.id.content, R.id.check, R.id.check1,
        R.id.image});
lv.setAdapter(sAdapter);
```

SimpleAdapter.ViewBinder

译者署名： 德罗德

译者链接：<http://sparkrico.javaeye.com>

翻译时间：2010-11-25

结构

继承关系

public static interface SimpleAdapter.ViewBinder

android.widget.SimpleAdapter.ViewBinder

类概述

SimpleAdapter的外部数据(external clients)可以使用这个类将值绑定到视图。你应该用这个类绑定值到那些不能直接通过SimpleAdapter支持的视图，或者改变通过SimpleAdapter支持绑定的方法的视图。

(译者注：如果SimpleAdapter设置了ViewBinder则使用Viewbinder的绑定规则，否则使用SimpleAdapter的默认绑定规则)

公共方法

public abstract boolean setViewValue (View view, Object data, String textRepresentation)

绑定指定数据到指定的视图。当通过调用 ViewBinder 绑定数据时，此方法必须返回真。如果这个方法返回假，SimpleAdapter 将尝试通过其内部默认的方法绑定数据。

参数

view 要绑定数据的视图

data 要绑定到视图的数据

textRepresentation 一个表示所支持数据的安全的字符串：结果是 data.toString() 或者空字符串，但不会是 null。

返回值

如果数据绑定到视图返回真，否则返回假。

补充

文章链接

[SimpleAdapter.ViewBinder 这个接口对于 ListActivity 有什么效果](#)

[Adapter 应用总结](#)

SimpleCursorAdapter

译者署名： 深夜未眠

译者链接：<http://chris1012f.javaeye.com/>

版本：Android 3.0 r1

结构

继承关系

public class SimpleCursorAdapter extends ResourceCursorAdapter

```
java.lang.Object  
    android.widget.BaseAdapter  
        android.widget.CursorAdapter  
            android.widget.ResourceCursorAdapter  
                android.widget.SimpleCursorAdapter
```

类概述

这是一个用起来很方便的适配器类，它主要将 Cursor 与 TextView 或 ImageView 进行映射。比如，你想设定要展示三列，那么当做好绑定之后，视图就会展示你设定好的那些列；当然了，视图的外观是定义在 XML 文件里面的，你只需用这个类与视图做好绑定就可以了。与视图绑定有两个阶段。第一阶段：如果 [SimpleCursorAdapter.ViewBinder](#) 可用，将会调用 [setViewValue\(android.view.View, android.database.Cursor, int\)](#) 方法。该方法返回 true 就说明绑定成功，否则返回 false，这就到了第二阶段，SimpleCursorAdapter 内部开始自行绑定，过程是这样的，若绑定到 TextView 上，调用 [setViewText\(\)](#)；若绑定到 ImageView 上，调用 [setViewImage\(\)](#)；如果视图不是 TextView 或 ImageView 则抛出 IllegalStateException 异常。当使用带有过滤器的适配器时，例如，在 APIDemo 中有个 AutoCompleteTextView 的例子，我们能使用 [SimpleCursorAdapter.CursorToStringConverter](#) 和接口 [FilterQueryProvider](#) 来控制过滤过程。更多信息请参考 [convertToString\(android.database.Cursor\)](#) 和 [runQueryOnBackgroundThread\(CharSequence\)](#)。

内部类

public interface SimpleCursorAdapter.ViewBinder

这个内部接口可以在外部通过 SimpleCursorAdapter.ViewBinder 的方式进行 Cursor 与 View 的绑定。

public interface SimpleCursorAdapter.CursorToStringConverter

这个内部接口可以在外部通过 SimpleCursorAdapter.CursorToStringConverter 的方式 定义怎样将 Cursor 转换成字符串。

构造函数

public SimpleCursorAdapter (Context context, int layout, Cursor c, String[] from, int[] to)

构造函数启动自动重新查询(auto-requery)。

这个构造器已被标记为弃用(@Deprecated)。

该方法不推荐使用，Cursor 查询操作是执行在应用程序的 UI 线程当中，那么会导致无响应的情况。另一种方式是使用 LoaderManager 和 CursorLoader 来进行。

(译者注：3.0 已不推荐使用该构造方法)

参数

`context` 应用程序上下文，具体来说就是 `ListView` 所在的上下文当中。

`layout` 布局文件的资源定位标识符，也就是说标识了 `ListView` 中的 `item`。那么这个布局文件至少包含了参数 “`to`” 中的传进来值。

`c` 数据库游标，如果游标不可用则为 `null`。

`from` 列名字列表，表示着你要绑定到 UI 上的列。如果游标不可用则为 `null`。

`to` 展示参数 “`from`” 中的列，也就是说 `ListView` 中的视图显示的是参数 “`from`” 的列值，这些视图应该都是 `TextView`。如果游标不可用则为 `null`。

```
public SimpleCursorAdapter (Context context, int layout, Cursor c, String[] from, int[] to,int flags)
```

该适配器类标准的构造函数。(译者注：3.0 新添的构造方法)

参数

`context` 应用程序上下文，具体来说就是 `ListView` 所在的上下文当中。

`layout` 布局文件的资源定位标识符，也就是说标识了 `ListView` 中的 `item`。那么这个布局文件至少包含了参数 “`to`” 中的传进来值。

`c` 数据库游标，如果游标不可用则为 `null`。

`from` 列名字列表，表示着你要绑定到 UI 上的列。如果游标不可用则为 `null`。

`to` 展示参数 “`from`” 中的列，也就是说 `ListView` 中的视图显示的是参数 “`from`” 的列值，这些视图应该都是 `TextView`。如果游标不可用则为 `null`。

`flags` 这个标志用来决定该适配器的行为。(译者注：Android3.0 推荐我们传递 `CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER`。设置标志用来添加一个监听器，监听着参数 `cursor` 的数据是否有更变。)

公共方法

```
public void bindView (View view, Context context, Cursor cursor)
```

绑定所有构造函数中的参数 `from` (字段名) 一一绑定到参数 `to` (视图资源 ID)。与视图绑定有两个阶段。第一阶段：如果 [SimpleCursorAdapter.ViewBinder](#) 可用，将会调用 [setViewValue\(android.view.View, android.database.Cursor, int\)](#) 方法。该方法返回 `true` 就说明绑定成功，否则返回 `false`，这就到了第二阶段，`SimpleCursorAdapter` 内部开始自行绑定，过程是这样的，若绑定到 `TextView` 上，调用 `setText()`;若绑定到 `ImageView` 上，调用 `setImage()`;如果视图不是 `TextView` 或 `ImageView` 则抛出 `IllegalStateException` 异常。

参数

`view` 已存在的视图(`View`)对象，也就是早先 `new` 出来的。

`context` 应用程序上下文。

`cursor` 数据库游标。该游标已经移动到指定位置上。

异常

`IllegalStateException` 如果绑定的视图中不是 `TextView` 或是 `ImageView` 则会抛出这个异常。

参见

[bindView\(android.view.View, android.content.Context, android.database.Cursor\)](#)
[getViewBinder\(\)](#)
[setViewBinder\(android.widget.SimpleCursorAdapter.ViewBinder\)](#)

[setViewImage\(ImageView, String\)](#)

[setViewText\(TextView, String\)](#)

public void changeCursorAndColumns(Cursor c, String[] from, int[] to)

同时更改 Cursor 与 View 的映射关系。

参数

c 数据库游标，如果游标不可用则为 null。

from 列名字列表，表示着你要绑定到 UI 上的列。如果游标不可用则为 null。

to 展示参数“from”中的列，也就是说 ListView 中的视图显示的是参数“from”的列值，这些视图应该都是 TextView。如果游标不可用则为 null。

public CharSequence convertToString (Cursor cursor)

通过 CursorToStringConverter 接口实现并返回一个 CharSequence 类型的值，以表示指定的 Cursor。如果没有设置 CursorToStringConverter，那么就会用另外的方式进行转换。如果列数为-1，或者 cursor 为 null 返回空串，否则返回 cursor.toString()。

参数

cursor 转换为 CharSequence 的数据库游标。

返回值

返回一个不为 null 的 CharSequence 类型来表示参数 cursor。

public SimpleCursorAdapter.CursorToStringConverter getCursorToStringConverter ()

返回自定义的 SimpleCursorAdapter.CursorToStringConverter 的实现。

返回值

如果没有设置 SimpleCursorAdapter.CursorToStringConverter，则为 null。

参考

[setCursorToStringConverter\(android.widget.SimpleCursorAdapter.CursorToStringConverter\)](#)

[getStringConversionColumn\(\)](#)

[setStringConversionColumn\(int\)](#)

[convertToString\(android.database.Cursor\)](#)

public int getStringConversionColumn ()

返回转换成 String 类型的列位置。

返回值

返回列位置，如果没有则返回-1。

参考

[convertToString\(android.database.Cursor\)](#)

[setStringConversionColumn\(int\)](#)

[setCursorToStringConverter\(android.widget.SimpleCursorAdapter.CursorToStringConverter\)](#)

[getCursorToStringConverter\(\)](#)

public SimpleCursorAdapter.ViewBinder getViewBinder ()

返回 SimpleCursorAdapter.ViewBinder 引用，这个 ViewBinder 用来将数据绑定到视图上

的。

返回值

如果 ViewBinder 不存在，则返回 null。

参考

[bindView\(android.view.View, android.content.Context, android.database.Cursor\)](#)
[setViewBinder\(android.widget.SimpleCursorAdapter.ViewBinder\)](#)

```
public void setCursorToStringConverter (SimpleCursorAdapter.CursorToStringConverter cursorToStringConverter)
```

设置 String 转换器。

参数

cursorToStringConverter String 转换器，设置为 null 就意味着移除。

参考

[setCursorToStringConverter\(android.widget.SimpleCursorAdapter.CursorToStringConverter\)](#)

[getStringConversionColumn\(\)](#)

[setStringConversionColumn\(int\)](#)

[convertToString\(android.database.Cursor\)](#)

```
public void setStringConversionColumn (int stringConversionColumn)
```

设置 Cursor 中的列要转换成 String 类型的位置。不过仅当未设置 CursorToStringConverter 时，这个列才会进行默认转换。

参数

stringConversionColumn 列位置，如果参数为 -1，则使用默认转换机制。

参考

[convertToString\(android.database.Cursor\)](#)

[getStringConversionColumn\(\)](#)

[setCursorToStringConverter\(android.widget.SimpleCursorAdapter.CursorToStringConverter\)](#)

[getCursorToStringConverter\(\)](#)

```
public void setViewBinder (SimpleCursorAdapter.ViewBinder viewBinder)
```

设置视图绑定器。

参数

viewBinder 视图绑定器，可用为 null 删除现有的绑定器。

参考

[bindView\(android.view.View, android.content.Context, android.database.Cursor\)](#)
[getViewBinder\(\)](#)

```
public void setViewImage (ImageView v, String value)
```

仅当 ViewBinder 不存在或是当 ViewBinder 不为 ImageView 绑定时（也就是 `setViewValue()` 返回 `false`），则这个方法会被 `bindView()` 调用，以便为 ImageView 设置图片。默认情况下，参数 `value` 作为图片资源 ID 来看待，否则，会视为图片的 Uri。另外还可以通过过滤器来获得更灵活的设置。

参数

- v 图片控件引用
- value 图片资源 ID，是从 Cursor 获取到的。

public void setViewText (TextView v, String text)

仅当 ViewBinder 不存在或是当 ViewBinder 不为 TextView 绑定时(也就是 `setViewValue()` 返回 `false`)，则这个方法会被 `bindView()` 调用，以便为 TextView 设置文本。可用重写适配器从数据库中检索过滤字符串。

参数

- v 文本控件引用
- value 为文本控件设置的文本信息（译者注：是从 Cursor 获取到的）。

public Cursor swapCursor (Cursor c)

交换两个 Cursor 的列以及它们的数据，并最终返回的还是旧的 Cursor。不同于 [changeCursor\(Cursor\)](#) 的是，旧的 Cursor 非但不会关闭，而且还会返回出去。（译者注：3.0 新添的方法）

参数

- c 新的 Cursor 对象。

返回值

返回旧的 Cursor 引用，如果参数 cursor 不可用，则返回 null。如果参数 cursor 与原来的 Cursor 引用相同，那么也返回 null。

补充

文章精选

[ArrayAdapter 和 SimpleCursorAdapter 例子](#)

[Android API : SimpleCursorAdapter\(\)](#)

SimpleCursorAdapter.CursorToStringConverter

译者署名： 深夜未眠

译者链接：<http://chris1012f.javaeye.com/>

版本：Android 3.0 r1

结构

继承关系

public static interface SimpleCursorAdapter.CursorToStringConverter

java.lang.Object

 android.widget.SimpleCursorAdapter.CursorToStringConverter

类概述

这个内部接口可以在外部通过 SimpleCursorAdapter.CursorToStringConverter 的方式定义怎样将 Cursor 转换成字符串。

参见

[convertToString\(android.database.Cursor\)](#)

公共方法

public abstract CharSequence convertToString (Cursor cursor)

返回 CharSequence 类型的值，用来表示参数 cursor。

参数

cursor 需要转换成 CharSequence 类型的数据库游标。

返回值

返回代表参数 cursor 的非空字符串。

SimpleCursorAdapter.ViewBinder

译者署名： 深夜未眠

译者链接：<http://chris1012f.javaeye.com/>

版本：Android 3.0 r1

结构

继承关系

public static interface SimpleCursorAdapter.ViewBinder

java.lang.Object

 android.widget.SimpleCursorAdapter.ViewBinder

类概述

这个内部接口可以在外部通过 SimpleCursorAdapter.ViewBinder 的方式进行 Cursor 与 View 的绑定。Android 推荐我们采用这种方式进行绑定操作，而不是沿用 SimpleCursorAdapter 内部的方式。

参见

[bindView\(android.view.View, android.content.Context, android.database.Cursor\)](#)

[setViewImage\(ImageView, String\)](#)

[setViewText\(TextView, String\)](#)

公共方法

public abstract boolean setViewValue (View view, Cursor cursor, int columnIndex)

将指定的列数据绑定到指定的视图上。当 ViewBinder 处理时，这个方法必须返回 true。如果这个方法返回 false， SimpleCursorAdapter 将用自己的方式进行绑定操作。

参数

view 被绑定的视图。

cursor 数据库游标，绑定数据从它这里获取

columnIndex 列位置，能够在数据库游标中寻找到。

返回值

返回 true 意味着数据与视图已经绑定上，否则为未绑定上。

SimpleCursorTreeAdapter

译者署名： 深夜未眠

译者链接：<http://chris1012f.javaeye.com/>

翻译时间： 2011-3-7

版本： Android 3.0 r1

结构

继承关系

```
public abstract class SimpleCursorAdapter extends  
    ResourceCursorAdapter  
java.lang.Object  
    android.widget.BaseExpandableListAdapter  
        android.widget.CursorTreeAdapter  
            android.widget.ResourceCursorAdapter  
                android.widget.SimpleCursorAdapter
```

类概述

这是一个用起来很方便的适配器类，它主要将 Cursor 与在 XML 文件中定义的 TextView 或 ImageView 进行映射。比如，你想设定要展示三列，那么当做好绑定之后，视图就会展示你设定好的那些列；当然了，视图的外观是定义在 XML 文件里面的，你只需用这个类与视图做好绑定就可以了。（译者注：Android 推荐我们尽可能的将组视图和子视图分离开，也就是说不要把整体定义在一个布局文件当中。）与视图绑定有两个阶段。第一阶段：如果使用 [SimpleCursorAdapter.ViewBinder](#) 时，那么就会调用 [setViewValue\(android.view.View, android.database.Cursor, int\)](#) 方法。该方法返回 true 就说明绑定成功，否则返回 false，这就到了第二阶段，SimpleCursorAdapter 内部开始自行绑定，过程是这样的，若绑定到 TextView 上，调用 [setViewText\(Textview, String\)](#)；若绑定到 ImageView 上，调用 [setViewImage\(Imageview, String\)](#)，如果视图不是 TextView 或 ImageView 则抛出 IllegalStateException 异常。

内部类

```
public interface SimpleCursorAdapter.ViewBinder
```

这个内部接口可以在外部通过 SimpleCursorAdapter.ViewBinder 的方式进行 Cursor 与 View 的绑定。

构造函数

```
public SimpleCursorAdapter (Context context, Cursor cursor, int collapsedGroupLayout,  
int expandedGroupLayout, String[] groupFrom, int[] groupTo, int childLayout, int lastChildLayout,  
String[] childFrom, int[] childTo)
```

构造函数。

参数

context 上下文，多指 ExpandableListView 的上下文

cursor 数据库游标

collapsedGroupLayout 布局资源文件标识 ID，其定义的是收缩时的
ExpandableListView 布局样式，并且内部至少要包含参数 “groupTo”
中指定的视图 ID。

`expandedGroupLayout` 布局资源文件标识 ID, 其定义的是展开时的 `ExpandableListView` 布局样式, 并且内部至少要包含参数 “`groupTo`” 中指定的视图 ID。

`groupFrom` 列名列表, 显示 `ExpandableListView` 的组节点。

`groupTo` 展示参数 “`groupFrom`” 中的列, 也就是说 `ExpandableListView` 中的视图显示的是参数 “`groupFrom`” 的列值, 它们应该都是 `TextView` 或是 `ImageView`。

`childLayout` 布局资源文件标识 ID, 其定义的是子视图的布局样式 (不包括最后一个子视图), 内部至少要包含参数 “`childTo`” 中指定的视图 ID。

`lastChildLayout` 布局资源文件标识 ID, 其定义的是最后一个子视图的布局样式, 内部至少要包含参数 “`childTo`” 中指定的视图 ID。

`lastChildLayout` 布局资源文件标识 ID, 其定义的是最后一个子视图的布局样式, 内部至少要包含参数 “`childTo`” 中指定的视图 ID。`lastChildLayout` 布局资源文件标识 ID, 其定义的是最后一个子视图的布局样式, 内部至少要包含参数 “`childTo`” 中指定的视图 ID。

`childFrom` 列名列表, 显示 `ExpandableListView` 的子节点。

`childTo` 展示参数 “`childFrom`” 中的列, 也就是说 `ExpandableListView` 中的视图显示的是参数 “`childFrom`” 的列值, 它们应该都是 `TextView` 或是 `ImageView`。

```
public SimpleCursorTreeAdapter (Context context, Cursor cursor, int collapsedGroupLayout,  
int expandedGroupLayout, String[] groupFrom, int[] groupTo, int childLayout, String[] childFrom,  
int[] childTo)
```

构造函数。(译者注: 该构造函数只是少了一个参数 `lastChildLayout`)

参数

`context` 上下文, 多指 `ExpandableListView` 的上下文

`cursor` 数据库游标

`collapsedGroupLayout` 布局资源文件标识 ID, 其定义的是收缩时的 `ExpandableListView` 布局样式, 并且内部至少要包含参数 “`groupTo`” 中指定的视图 ID。

`expandedGroupLayout` 布局资源文件标识 ID, 其定义的是展开时的 `ExpandableListView` 布局样式, 并且内部至少要包含参数 “`groupTo`” 中指定的视图 ID。

`groupFrom` 列名列表, 显示 `ExpandableListView` 的组节点。

`groupTo` 展示参数 “`groupFrom`” 中的列, 也就是说 `ExpandableListView` 中的视图显示的是参数 “`groupFrom`” 的列值, 它们应该都是 `TextView` 或是 `ImageView`。

`childLayout` 布局资源文件标识 ID, 其定义的是子视图的布局样式 (不包括最后一个子视图), 内部至少要包含参数 “`childTo`” 中指定的视图 ID。

`lastChildLayout` 布局资源文件标识 ID, 其定义的是最后一个子视图的布局样式, 内部至少要包含参数 “`childTo`” 中指定的视图 ID。

`childFrom` 列名列表, 显示 `ExpandableListView` 的子节点。

`childTo` 展示参数 “`childFrom`” 中的列, 也就是说 `ExpandableListView` 中的视图显示的是参数 “`childFrom`” 的列值, 它们应该都是 `TextView`

或是 ImageView。

```
public SimpleCursorTreeAdapter (Context context, Cursor cursor, int groupLayout, String[]  
groupFrom, int[] groupTo, int childLayout, String[] childFrom, int[] childTo)
```

构造函数。

参数

context 上下文，多指 ExpandableListView 的上下文

cursor 数据库游标

groupLayout 显示组元素的资源文件。该资源文件定义了如何显示组元素。该布局文件必须至少包括 groupTo 中所定义的 View。(即 groupTo 中的 View id 数组必须都在该布局文件中找到)

groupFrom 列名列表，显示 ExpandableListView 的组节点。

groupTo 展示参数 “groupFrom” 中的列，也就是说 ExpandableListView 中的视图显示的是参数 “groupFrom” 的列值，它们应该都是 TextView 或是 ImageView。

childLayout 布局资源文件标识 ID，其定义的是子视图的布局样式 (不包括最后一个子视图)，内部至少要包含参数 “childTo” 中指定的视图 ID。

lastChildLayout 布局资源文件标识 ID，其定义的是最后一个子视图的布局样式，内部至少要包含参数 “childTo” 中指定的视图 ID。

childFrom 列名列表，显示 ExpandableListView 的子节点。

childTo 展示参数 “childFrom” 中的列，也就是说 ExpandableListView 中的视图显示的是参数 “childFrom” 的列值，它们应该都是 TextView 或是 ImageView。

公共方法

```
public SimpleCursorAdapter.ViewBinder getViewBinder ()
```

返回 [SimpleCursorTreeAdapter.ViewBinder](#) 引用，这个 ViewBinder 用来将数据绑定到视图上的。

返回值

如果 ViewBinder 不存在，则返回 null。

参考

[setViewBinder\(android.widget.SimpleCursorTreeAdapter.ViewBinder\)](#)

```
public void setViewBinder (SimpleCursorTreeAdapter.ViewBinder viewBinder)
```

. 设置视图绑定器。

参数

viewBinder 视图绑定器。可以设置为 null 来删除已经存在的绑定器。

参考

[getViewBinder\(\)](#)

```
public void setViewText (TextView v, String text)
```

仅当 ViewBinder 不存在或是当 ViewBinder 不为 TextView 绑定时(也就是 `setViewValue()` 返回 false)，则这个方法会被 `bindView()` 调用，以便为 TextView 设置文本。可重写适配器从数据库中检索过滤字符串。

参数

v 文本控件引用

value 为文本控件设置的文本信息（译者注：是从 Cursor 获取到的）。

受保护方法

```
protected void bindChildView (View view, Context context, Cursor cursor, boolean  
isExpanded)
```

通过参数 cursor 将数据绑定到已有的子视图上。

参数

view 已有视图，返回之前调用 newChildView 创建的视图。

context 应用程序上下文

cursor 用于获取数据的 Cursor。Cursor 已经移到正确的位置。

isLastChild 子元素是否处于组中的最后一个

```
protected void bindGroupView (View view, Context context, Cursor cursor, boolean  
isExpanded)
```

通过参数 cursor 将数据绑定到已有组视图上。

参数

view 已有组视图，返回之前调用 newGroupView 创建的视图。

context 应用程序上下文

cursor 用于获取数据的 Cursor。Cursor 已经移到正确的位置。

isExpanded 组视图是否呈展开状态

```
protected void setViewImage (ImageView v, String value)
```

这个方法会被 bindView() 调用，以便为 ImageView 设置图片。默认情况下，参数 value 作为图片资源 ID 来看待，否则会视为图片的 Uri。另外还可以通过过滤器来获得更灵活的设置。

参数

v 图片控件引用

value 图片资源 ID，是从 Cursor 获取到的。

补充

文章精选

[android 播放器（music player）源码分析 2](#)

SimpleCursorTreeAdapter.ViewBinder

译者署名： 深夜未眠

译者链接: <http://chris1012f.javaeye.com/>

翻译时间： 2011-3-3

版本： Android 3.0 r1

结构

继承关系

public static interface SimpleCursorTreeAdapter.ViewBinder

java.lang.Object

 android.widget.SimpleCursorTreeAdapter.ViewBinder

类概述

这个内部接口可以在外部通过 SimpleCursorTreeAdapter.ViewBinder 的方式进行 Cursor 与 View 的绑定。Android 推荐我们采用这种方式进行绑定操作，而不是沿用 SimpleCursorTreeAdapter 内部的方式。

参见

[setViewImage\(ImageView, String\)](#)

[setViewText\(TextView, String\)](#)

公共方法

public abstract boolean setViewValue (View view, Cursor cursor, int columnIndex)

将指定的列数据绑定到指定的视图上。当 ViewBinder 处理绑定时，这个方法必须返回 true；否则 SimpleAdapter 将尝试通过其内部默认的方法绑定数据。

参数

view 被绑定的视图。

cursor 数据库游标，绑定数据从它这里获取

columnIndex 列位置，能够在数据库游标中寻找到。

返回值

返回 true 意味着数据与视图已经绑定上，否则为未绑定上。

SimpleExpandableListAdapter

译者署名：天涯明月刀

译者博客：<http://sd6733531.javaeye.com/>

版本：Android 2.3 r1

结构

继承关系

public abstract class SimpleExpandableListAdapter extends BaseExpandableListAdapter

```
java.lang.Object  
    android.widget.BaseExpandableListAdapter  
        android.widget.SimpleExpandableListAdapter
```

类概述

一个使用 Map 存储组元素和子元素的静态数据，使用 XML 资源文件定义组元素和子元素 View 如何显示的简单适配器。你可以区分指定组元素返回的数据是 List<Map> 类型。在 ArrayList 中的每个实体对应 ExpandableList 中的一个组。实体中的 Map 列表包含了组下的每行数据。你也可以指定一个 XML 文件来定义用于在组元素的 View 的显示。此时 Map 中的键值将与指定的 View(XML 中定义的 view id)对应起来。子元素的处理情况类似。注意当可折叠的深度若不只 1 层，返回的数据将被指定为 List 类型。首个 List 对应子元素中所代表的组，第二个 List 对应孙子元素在子元素组中的位置。Map 亦将支持这样的特殊元素。(子元素嵌套组元素的情况)

(译者注：ExpandableList 支持深度大于 1 的情况。在 andorid 自带的 ApiDomos 的例子中有这个的代码：App/View/ExpandableList3)。

构造函数

```
public SimpleExpandableListAdapter (Context context, List<? extends Map<String, ?>>  
groupData, int groupLayout, String[] groupFrom, int[]  
groupTo, List<? extends List<? extends Map<String, ?>>> childData, int  
childLayout, String[] childFrom, int[] childTo)
```

构造函数

参数

context 与 [SimpleExpandableListAdapter](#) 关联的 [ExpandableListView](#) 的上
下文。

groupData 一个 Maps 列表(List)。集合中的每个字典项与可折叠列表中的每
个组元素一致。字典项提供了组元素包含的所有数据，并包含所有在"groupFrom"中指
定的记录。

groupLayout 显示组元素的资源文件。该资源文件定义了如何显示组元素。该
布局文件必须至少包括 groupTo 中所定义的 View。(即 groupTo 中的 View id 数组必须都在该
布局文件中找到)

groupFrom 一个键值列表。对应与组相关联的 Map 中的键值。

groupTo 组 View 应当显示 groupFrom 参数中的所有列数据。这些数据应

当都用 `TextView` 来显示。列表中的前 N 个 `View` 从前 N 个 `groupFrom` 参数获得列元素的数据。

`childData` 一个 `Map` 列表的列表。外部列表中的每个实体对应一个组(按照组的位置编号)。在内部列表的每个实体对应某个组的子元素(按照子元素的位置编号)。该 `Map` 对应了子元素的数据。(按照 `childFrom` 数组中的值编号)。该 `Map` 包含了每个子元素的数据，并且应当包括所有在 `childFrom` 中指定的实体。

`childLayout` 显示子元素的资源文件。该资源文件定义了如何显示子元素。布局文件至少应该包括所有在 `childTo` 中定义的 `View`。(即 `childTo` 中的 `view id` 数组必须都在该布局文件中找到)

`childFrom` 定义显示子元素的列名。该列名与 `childData` 中的子元素属性(字典键值)对应。

`childTo` 子 `View` 应当显示 `childFrom` 参数中的所有列数据。这些数据应当都用 `TextView` 来显示。列表中的前 N 个 `View` 从前 N 个 `childFrom` 参数获得列元素的数据。

```
public SimpleExpandableListAdapter (Context context, List<? extends Map<String, ?>>
groupData, int expandedGroupLayout, int collapsedGroupLayout, String[] groupFrom, int[]
groupTo, List<? extends List<? extends Map<String, ?>>> childData, int
childLayout, String[] childFrom, int[] childTo)
```

构造函数。

参数

`context` 与 `SimpleExpandableListAdapter` 关联的 `ExpandableListView` 的上下文。

`groupData` 一个 `Maps` 列表(`List`)。集合中的每个字典项与可折叠列表中的每个组元素一致。字典项提供了组元素包含的所有数据，并包含所有在" `groupFrom`"中指定的记录。

`expandedGroupLayout` 定义组展开时的 `View` 的 XML 资源布局。该布局文件应当至少包括所有在 `groupTo` 中所定义的 `View`。(即 `groupTo` 中的 `View id` 数组必须都在该布局文件中找到)

`collapsedGroupLayout` 定义组折叠时的 `View` 的 XML 资源布局。该布局文件应当至少包括所有在 `groupTo` 中所定义的 `View`。(即 `groupTo` 中的 `View id` 数组必须都在该布局文件中找到)

`groupFrom` 一个键值列表。对应与组相关联的 `Map` 中的键值。

`groupTo` 组 `View` 应当显示 `groupFrom` 参数中的所有列数据。这些数据应当都用 `TextView` 来显示。列表中的前 N 个 `View` 从前 N 个 `groupFrom` 参数获得列元素的数据。

`childData` 一个 `Map` 列表的列表。外部列表中的每个实体对应一个组(按照组的位置编号)。在内部列表的每个实体对应某个组的子元素(按照子元素的位置编号)。该 `Map` 对应了子元素的数据。(按照 `childFrom` 数组中的值编号)。该 `Map` 包含了每个子元素的数据，并且应当包括所有在 `childFrom` 中指定的实体。

`childLayout` 显示子元素的资源文件。该资源文件定义了如何显示子元素。布局文件至少应该包括所有在 `childTo` 中定义的 `View`。(即 `childTo` 中的 `view id` 数组必须都在该布局文件中找到)

`childFrom` 定义显示子元素的列名。该列名与 `childData` 中的子元素属性(字典键值)对应。

childTo 子 View 应当显示 childFrom 参数中的所有列数据。这些数据应当都用 TextView 来显示。列表中的前 N 个 View 从前 N 个 childFrom 参数获得列元素的数据。

```
public SimpleExpandableListAdapter (Context context, List<? extends Map<String, ?>>
groupData, int expandedGroupLayout, int collapsedGroupLayout, String[] groupFrom, int[]
groupTo, List<? extends List<? extends Map<String, ?>>> childData, int childLayout, int
lastChildLayout, String[] childFrom, int[] childTo)
```

构造函数。

参数

context 与 SimpleExpandableListAdapter 关联的 ExpandableListView 的上下文。

groupData 一个 Maps 列表(List)。集合中的每个字典项与可折叠列表中的每个组元素一致。字典项提供了组元素包含的所有数据，并包含所有在"groupFrom"中指定的记录。

expandedGroupLayout 定义组展开时的 View 的 XML 资源布局。该布局文件应当至少包括所有在 groupTo 中所定义的 View。(即 groupTo 中的 View id 数组必须都在该布局文件中找到)

collapsedGroupLayout 定义组折叠时的 View 的 XML 资源布局。该布局文件应当至少包括所有在 groupTo 中所定义的 View。(即 groupTo 中的 View id 数组必须都在该布局文件中找到)

groupFrom 一个键值列表。对应与组相关联的 Map 中的键值。

groupTo 组 View 应当显示 groupFrom 参数中的所有列数据。这些数据应当都用 TextView 来显示。列表中的前 N 个 View 从前 N 个 groupFrom 参数获得列元素的数据。

childData 一个 Map 列表的列表。外部列表中的每个实体对应一个组(按照组的位置编号)。在内部列表的每个实体对应某个组的子元素(按照子元素的位置编号)。该 Map 对应了子元素的数据。(按照 childFrom 数组中的值编号)。该 Map 包含了每个子元素的数据，并且应当包括所有在 childFrom 中指定的实体。

childLayout 显示子元素的资源文件。该资源文件定义了如何显示子元素。布局文件至少应该包括所有在 childTo 中定义的 View。(即 childTo 中的 view id 数组必须都在该布局文件中找到)

lastChildLayout 定义每组中最后一个子元素的 View 资源布局情况。该布局文件应当至少包括所有在 childTo 中所定义的 View。(即 childTo 中的 View id 数组必须都在该布局文件中找到)

childFrom 定义显示子元素的列名。该列名与 childData 中的子元素属性(字典键值)对应。

childTo 子 View 应当显示 childFrom 参数中的所有列数据。这些数据应当都用 TextView 来显示。列表中的前 N 个 View 从前 N 个 childFrom 参数获得列元素的数据。

公共方法

```
public Object getChild (int groupPosition, int childPosition)
```

```
public long getChildId (int groupPosition, int childPosition)
```

(译者注：获取与在给定组给予孩子相关的数据。)

```
public View getChildView (int groupPosition, int childPosition, boolean  
isLastChild, View convertView, ViewGroup parent)
```

(译者注：获取子项)

```
public int getChildrenCount (int groupPosition)
```

(译者注：返回在指定 Group 的 Child 数目。)

```
public Object getGroup (int groupPosition)
```

```
public int getGroupCount ()
```

(译者注：返回 Group 的总数目。)

```
public long getGroupId (int groupPosition)
```

```
public View getGroupView (int groupPosition, boolean  
isExpanded, View convertView, ViewGroup parent)
```

(译者注：获取父项)

```
public boolean hasStableIds ()
```

```
public boolean isChildSelectable (int groupPosition, int childPosition)
```

(译者注：是否让 child 获得焦点)

```
public View newChildView (boolean isLastChild, ViewGroup parent)
```

新建一个子元素 View。

参数

isLastChild 该 child 是否是组中的最后一个元素

parent 新的 View 的最终父亲。

返回

新的子元素 View 对象。

```
public View newGroupView (boolean isExpanded, ViewGroup parent)
```

新建一个组元素 View。

参数

isExpanded 该组元素是否当前处于折叠状态

parent 新 View 的最终父亲。

返回

新的组元素 View 对象。

补充

文章精选

ExpandableListActivity 和 SimpleExpandableListAdapter 的使用方法

[Expandable lists](#)[blogspot]

示例代码：

```
List<Map<String, String>> groupData = new ArrayList<Map<String, String>>();
List<List<Map<String, String>>> childData = new
ArrayList<List<Map<String, String>>>();
for (int i = 0; i < 20; i++) {
    Map<String, String> curGroupMap = new HashMap<String,
String>();
    groupData.add(curGroupMap);
    curGroupMap.put(NAME, "Group " + i);
    curGroupMap.put(IS_EVEN, (i % 2 == 0) ? "This group is even" :
"This group is odd");

    List<Map<String, String>> children = new ArrayList<Map<String,
String>>();
    for (int j = 0; j < 15; j++) {
        Map<String, String> curChildMap = new HashMap<String,
String>();
        children.add(curChildMap);
        curChildMap.put(NAME, "Child " + j);
        curChildMap.put(IS_EVEN, (j % 2 == 0) ? "This child is even" :
"This child is odd");
    }
    childData.add(children);
}

// Set up our adapter
mAdapter = new SimpleExpandableListAdapter(
    this,
    groupData,
    android.R.layout.simple_expandable_list_item_1,
    new String[] { NAME, IS_EVEN },
    new int[] { android.R.id.text1, android.R.id.text2 },
    childData,
    android.R.layout.simple_expandable_list_item_2,
    new String[] { NAME, IS_EVEN },
    new int[] { android.R.id.text1, android.R.id.text2 }
);
```

SlidingDrawer

译者署名: xiaoQLu

译者链接: <http://www.cnblogs.com/xiaoQLu>

版本: Android 3.0 r1

结构

继承关系

public class SlidingDrawer extends ViewGroup

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.SlidingDrawer
```

类概述

SlidingDrawer (滑动式抽屉) 隐藏屏外的内容，并允许用户拖拽一个 handle 以显示隐藏的内容。SlidingDrawer 可以在垂直或者水平使用。它由两个子视图组成：一个是用户拖拽的 handle (柄)，另一个是随着拖动变化的 content (内容)。SlidingDrawer 应当作为内部布局的覆盖来使用，也就是说 SlidingDrawer 内部应该使用 FrameLayout 或 RelativeLayout 布局。SlidingDrawer 的大小决定了其内容显示时所占空间的大小，所以它的尺寸一般定义为 match_parent。在 XML 布局中 SlidingDrawer 必须指定 handle 和 content 的 id:

```
<SlidingDrawer  
    android:id="@+id/drawer"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
  
    android:handle="@+id/handle"  
    android:content="@+id/content">  
  
<ImageView  
    android:id="@+id/handle"  
    android:layout_width="88dip"  
    android:layout_height="44dip" />  
  
<GridView  
    android:id="@+id/content"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />  
  
</SlidingDrawer>
```

内部类

interface **SlidingDrawer.OnDrawerCloseListener**

当 drawer (抽屉) 关闭时调用

interface **SlidingDrawer.OnDrawerOpenListener**

当 drawer (抽屉) 打开时调用

interface **SlidingDrawer.OnDrawerScrollListener**

当 drawer (抽屉) 滑动 (滚动) 时调用

XML 属性

属性名称	描述									
android:allowSingleTap	指示是否可通过单击 handle 打开或关闭 (如果是 false, 刚用户必须通过拖动, 滑动或者使用轨迹球, 来打开/关闭抽屉。) 默认的是 true。									
android:animateOnClick	指示当用户点击 handle 的时候, 抽屉是否以动画的形式打开或关闭。默认的是 true。									
android:bottomOffset	Handle 距离 SlidingDrawer 底部的额外距离									
android:content	标识 SlidingDrawer 的内容									
android:handle	标识 SlidingDrawer 的 handle (译者注: 如按钮)									
android:orientation	SlidingDrawer 的方向。必须是下面的一个值: <table border="1"><thead><tr><th>常量</th><th>值</th><th>描述</th></tr></thead><tbody><tr><td>horizontal</td><td>0</td><td>水平方向对齐</td></tr><tr><td>vertical</td><td>1</td><td>竖直方向对齐</td></tr></tbody></table>	常量	值	描述	horizontal	0	水平方向对齐	vertical	1	竖直方向对齐
常量	值	描述								
horizontal	0	水平方向对齐								
vertical	1	竖直方向对齐								
android:topOffset	Handle 距离 SlidingDrawer 顶部的额外距离									

常量

public static final int **ORIENTATION_HORIZONTAL**

(译者注: 水平方向对齐)

常量值: 0 (0x00000000)

public static final int **ORIENTATION_VERTICAL**

(译者注: 垂直方向对齐)

常量值: 1 (0x00000001)

构造函数

public **SlidingDrawer** (Context context, AttributeSet attrs)

用 xml 中设置的属性来创建一个新的 SlidingDrawe

参数

context 上下文

attrs XML 中定义的属性

public SlidingDrawer (Context context, AttributeSet attrs, int defStyle)

用 xml 中设置的属性来创建一个新的 SlidingDrawe

参数

context 上下文
attrs XML 中定义的属性
defStyle 要应用到这个组件上的样式

公共方法

public void animateClose ()

动画效果关闭抽屉。

参见

[close\(\)](#)
[open\(\)](#)
[animateOpen\(\)](#)
[animateToggle\(\)](#)
[toggle\(\)](#)

public void animateOpen ()

动画效果打开抽屉。

参见

[close\(\)](#)
[open\(\)](#)
[animateOpen\(\)](#)
[animateToggle\(\)](#)
[toggle\(\)](#)

public void animateToggle ()

在打开和关闭抽屉之间动画切换

参见

[close\(\)](#)
[open\(\)](#)
[animateOpen\(\)](#)
[animateToggle\(\)](#)
[toggle\(\)](#)

public void close ()

立即关闭抽屉

参见

[toggle\(\)](#)
[open\(\)](#)
[animateClose\(\)](#)

public View getContent ()

返回抽屉的内容(content)

返回值

返回在抽屉内容的视图，它在 XML 中是用 “content” id 标识的

```
public View getHandle ()
```

返回抽屉的 handle

返回值

返回在抽屉 handle 的视图，它在 XML 中是用 “handle” id 标识的

```
public boolean isMoving ()
```

抽屉是否在滚动或滑动。

返回值

如果在滚动或滑动，返回 true，否则返回 false

```
public boolean isOpened ()
```

当前抽屉是否被完全打开

返回值

如果是打开的，返回 true，否则返回 false。

```
public void lock ()
```

锁定 SlidingDrawer，忽略触摸事件

参见

[unlock\(\)](#)

```
public boolean onInterceptTouchEvent (MotionEvent event)
```

实现这个方法可以拦截所有的触屏事件，它在事件被传到子类之前拦截，并获得当前手势的所有权。

使用这个方法时要注意，因为它与 [View.onTouchEvent\(MotionEvent\)](#)有一个相当复杂的交互，使用它需要用正确的方法来实现。事件会按照下列顺序接受：

1. down 事件会被首先传到本方法中。
2. 这个 down 事件会被当前 viewgroup 的 onTouchEvent()方法或者其各个子视图处理，也就是说你应该实现 onTouchEvent()方法并返回 true，你会继续看到剩下事件的传递（而不是找一个 parent view 处理它）。同样的，从 onTouchEvent()中返回 true，你不会在 onInterceptTouchEvent()中接受到任何接下来的事件，并且所有的事件都会被 onTouchEvent()处理。
3. 如果当前方法返回 false，所有接下来的事件（截止到最后包含注册的事件）首先都会被继续传到这里，然后一起传递给目标的 onTouchEvent()方法。截至及包括最后注册。
4. 如果在这里返回 true，将不会收到以下任何事件：目标 view 将收到同样的事件但是是伴随 [ACTION_CANCEL](#) 事件，并且所有的更进一步的事件将会传递到你自己的 onTouchEvent()方法中而不会再在这里出现。

（译者著：这里实在是太麻烦了，很不好懂，我自己看着都头晕，在网上找到一篇很不错的总结，才知道是怎么回事，这里总结一下，记住这个原则你就会很清楚了：

1、onInterceptTouchEvent()是用于处理事件（类似于预处理，当然也可以不处理）并改变事件的传递方向，也就是决定是否允许 Touch 事件继续向下（子控件）传递，一旦返回 True（代表事件在当前的 viewGroup 中会被处理），则向下传递之路被截断（所有子控件将没有机会参与 Touch 事件），同时把事件传递给当前的控件的 on

`TouchEvent()`处理；如果返回 `false`，则把事件交给子控件的 `onInterceptTouchEvent()` 处理

2、`onTouchEvent()`用于处理事件，返回值决定当前控件是否消费（consume）了这个事件，也就是说在当前控件在处理完 `Touch` 事件后，是否还允许 `Touch` 事件继续向上（父控件）传递，一旦返回 `True`，则父控件不用操心自己来处理 `Touch` 事件。

相关文章[这里 1](#)、[这里 2](#)）

参数

`event` 分层次的动作事件

返回值

如果将运动事件从子视图中截获并且通过 `onTouchEvent()`发送到当前 `ViewGroup`，返回 `true`。当前目标将会收到 `ACTION_CANCEL` 事件，并且不再会有其他消息传递到此。

```
public boolean onTouchEvent (MotionEvent event)
```

实现触摸屏幕事件的方法

参数

`event` 当前事件

返回值

如果事件被处理就返回 `true`，否则返回 `false`

```
public void open ()
```

立即打开抽屉

参见

[toggle\(\)](#)

[close\(\)](#)

[animateOpen\(\)](#)

```
public void setOnDrawerCloseListener(SlidingDrawer.OnDrawerCloseListener  
onDrawerCloseListener)
```

给抽屉的关闭事件绑定监听器

参数

`onDrawerCloseListener` 抽屉关闭时的监听器

```
public void setOnDrawerOpenListener(SlidingDrawer.OnDrawerOpenListener  
onDrawerOpenListener)
```

给抽屉的打开事件绑定监听器

参数

`onDrawerOpenListener` 抽屉打开时的鉴别器

```
public void setOnDrawerScrollListener (SlidingDrawer.OnDrawerScrollListener  
onDrawerScrollListener)
```

给抽屉的滚动(收缩)事件绑定监听器，轻滑(fling)也被当作一个滚动(收缩)事件，同时它可以触发抽屉关闭或者打开事件。

参数

`onDrawerScrollListener` 当滚动(收缩)开始或者停止时通知的监听器

`public void toggle ()`

在抽屉打开或关闭状态之间切换。事件会立即产生。

参见

[close\(\)](#)

[open\(\)](#)

[animateOpen\(\)](#)

[animateToggle\(\)](#)

[toggle\(\)](#)

`public void unlock ()`

解锁 SlidingDrawer 使触摸事件能被处理

参见

[lock\(\)](#)

补充

文章精选

[Android 高手进阶教程\(二\)之 ---Android Launcher 抽屉类 SlidingDrawer 的使用](#)

[SlidingDrawer 抽屉类{Android 学习指南}](#)

MotionEvent 事件在 `onInterceptTouchEvent()`、`onTouchEvent()` 中的传递顺序

http://hi.baidu.com/j_fo/blog/item/7321c91324203437dc54017d.html

http://www.cnblogs.com/rocky_yi/archive/2011/01/21/1941522.html#

示例代码

SlidingDrawer 的 XML 文件示例

```
<SlidingDrawer
    android:id="@+id/drawer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:handle="@+id/handle"
    android:content="@+id/content">

    <ImageView
        android:id="@+id/handle"
        android:layout_width="88dip"
        android:layout_height="44dip" />

    <GridView
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</SlidingDrawer>
```

SlidingDrawer.OnDrawerCloseListener

译者署名: xiaoQLu

译者链接: <http://www.cnblogs.com/xiaoQLu>

版本: Android 3.0 r1

结构

继承关系

public static interface SlidingDrawer.OnDrawerCloseListener

android.widget.SlidingDrawer.OnDrawerCloseListener

类概述

当抽屉被关闭时调用的回调。

公共方法

public abstract void onDrawerClosed()

当抽屉被完全关闭时调用

补充

文章精选

[Android 控件之 SlidingDrawer（滑动式抽屉）详解与实例](#)

SlidingDrawer.OnDrawerOpenListener

译者署名: xiaoQLu

译者链接: <http://www.cnblogs.com/xiaoQLu>

版本: Android 3.0 r1

结构

继承关系

public static interface SlidingDrawer.OnDrawerOpenListener

android.widget.SlidingDrawer.OnDrawerOpenListener

类概述

当抽屉被打开时调用

公共方法

public abstract void onDrawerOpened()

当抽屉被完全打开时调用

SlidingDrawer.OnDrawerScrollListener

译者署名: xiaoQLu

译者链接: <http://www.cnblogs.com/xiaoQLu>

版本: Android 3.0 r1

结构

继承关系

public static interface SlidingDrawer.OnDrawerScrollListener

android.widget.SlidingDrawer.OnDrawerScrollListener

类概述

当抽屉被滚动时调用

公共方法

public abstract void onScrollEnded ()

当用户停止拖曳/滑动抽屉的 handle 时调用

public abstract void onScrollStarted ()

当用户开始拖曳/滑动抽屉的 handle 时调用

Spinner

译者署名：思考的狼

译者链接：<http://blog.163.com/sikaodelang@126/>

版本：Android 2.2 r1

public final class Spinner extends AbsSpinner

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.AdapterView<T extends android.widget.Adapter>  
                android.widget.AbsSpinner  
                    android.widget.Spinner
```

类概述



下拉列表（Spinner）是一个每次只能选择所有项中一项的部件。它的项来自于与之相关的适配器中。

XML 属性

属性名称	描述
android:prompt	该提示在下拉列表对话框显示时显示。（译者注：对话框的标题：）



公共方法

`public int getBaseline()`

返回这个控件文本基线的偏移量。如果这个控件不支持基线对齐，那么方法返回-1。

返回值

返回控件基线左边边界位置，不支持时返回-1

(译者注：这个类不知道干什么用，只找到下面的代码

```
public class AndroidBamboo extends Activity
{
    public void onCreate( Bundle savedInstanceState )
    {
        super.onCreate ( savedInstanceState );
        Spinner spinner = new Spinner ( this );
        spinner.setPrompt( "500" );
        String [ ] items = { "bam", "boo", "lab", "code",
"programming", "framework", "android" };
        ArrayAdapter array_adapter = new ArrayAdapter
<String> ( this, android.R.layout.simple_spinner_item,
items );
        array_adapter.setDropDownViewResource
( android.R.layout.simple_spinner_dropdown_item );
        spinner.setAdapter ( array_adapter );
        int baseline = spinner.getBaseline ( );
        setContentView ( spinner );
    }
}
```

`public CharSequence getPrompt()`

返回值

当对话框弹出的时候显示的提示（译者注：获得弹出视图上的标题字）

`public void onClick(DialogInterface dialog, int which)`

当点击弹出框中的项时这个方法将被调用。

参数

dialog 点击弹出的对话框
which 点击按钮(如: `Button`)或者点击位置

public Boolean performClick()

如果它被定义就调用此视图的 `OnClickListener` (译者注: 例如可以在加载时默认弹出下拉列表)。

返回值

为 `True` 一个指定的 `OnClickListener` 被调用, 为 `false` 时不被调用。

public void setOnItemClickListener(AdapterView.OnItemClickListener l)

`Spinner` 不支持 `item` 的点击事件, 调用此方法将引发异常。

参数

`l` 这个监听将被忽略

public void setPromptId(CharSequence prompt)

设置对话框弹出的时候显示的提示 (译者注: 设置弹出视图上的标题字)

参数

`prompt` 设置的提示

public void setPromptId(int promptId)

设置对话框弹出的时候显示的提示 (译者注: 设置弹出视图上的标题字)

参数

`prompted` 当对话框显示是显示这个资源 `id` 所代表的提示。

受保护的方法

protected void onDetachedFromWindow ()

当这个视图从屏幕上卸载时候被调用。在这一点上不再绘制视图。

protected void onLayout (boolean changed, int l, int t, int r, int b)

当 `View` 要为所有子对象分配大小和位置时, 调用此方法。派生类与子项们应该重载这个方法和调用布局每一个子项。

参数

`changed` 这是这个视图的一个新的大小或位置

`l` 相对父空间的左位置

`t` 相对父空间的顶端位置

`r` 相对父空间的右端位置

`b` 相对父空间的底部位置

参见

[Creates and positions all views](#)

补充

文章链接

[Android 学习指南——Spinner 下拉列表](#)

[android 自定义 Spinner 下拉菜单\(下拉列表框\)样式](#)

示例代码

Java 代码

```
public class SpinnerActivity extends Activity {  
  
    /** Called when the activity is first created. */  
  
    String mes = "Wolf";  
  
    private static final String[] m_arr = {"第一组", "第二组", "第三组"};  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.main);  
  
        Spinner s1 = (Spinner) findViewById(R.id.Spinner01);  
  
        s1.setPrompt("请选择颜色");  
  
        //ArrayAdapter adapter = ArrayAdapter.createFromResource(this,  
        R.array.spinnercolor, android.R.layout.simple_spinner_item);  
  
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
        android.R.layout.simple_spinner_item, m_arr);  
  
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
  
        s1.setAdapter(adapter);  
  
        s1.setSelection(1, true);  
        //s1.setPromptId(CONTEXT_INCLUDE_CODE);  
  
  
        int baseline = s1.getBaseline();  
        s1.performClick();  
  
        s1.setOnItemSelectedListener(new  
        Spinner.OnItemSelectedListener() {  
  
            public void onItemSelected(AdapterView<?> arg0, View arg1, int  
            arg2, long arg3) {
```

```
        dispToast("选择的是"+m_arr[arg2]);  
  
        arg0.setVisibility(View.VISIBLE);  
  
    }  
  
    public void onNothingSelected(AdapterView<?> arg0) {  
  
        //  
  
    }  
  
});  
  
Toast.makeText(this, s1.getPrompt() ,Toast.LENGTH_LONG).show();  
  
}  
  
public void dispToast(String str){  
  
    Toast.makeText(this, str, Toast.LENGTH_SHORT).show();  
  
}  
  
}
```

XML 文件

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="hello">Hello World, SpinnerActivity!</string>  
    <string name="app_name">Spinner测试</string>  
    <string-array name="spinnercolor">  
        <item>红色</item>  
        <item>绿色</item>  
        <item>白色</item>  
        <item>橙色</item>  
    </string-array>  
    <string name="planet_prompt">Please Choose a Item! </string>  
</resources>
```

SpinnerAdapter

译者署名： 德罗德

译者链接：<http://sparkrico.javaeye.com>

翻译时间： 2010-11-03

版本： Android 2.2 r1

结构

继承关系

public interface SpinnerAdapter extends Adapter

 android.widget.SpinnerAdapter

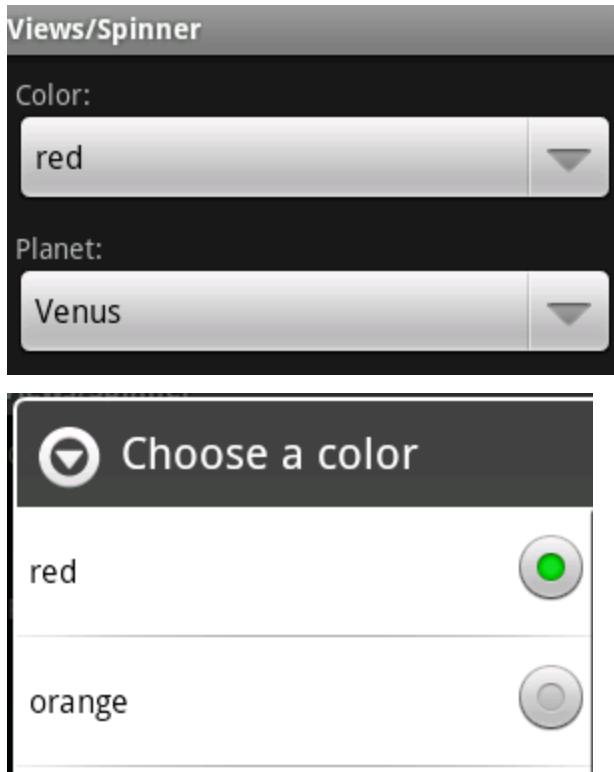
子类及间接子类

 直接子类

 ArrayAdapter<T>, BaseAdapter, CursorAdapter, ResourceCursorAdapter,

 SimpleAdapter, SimpleCursorAdapter

类概述



扩展自 `Adapter` 的适配器是在 `Spinner` 与数据之间的一座桥梁。一个 `Spinner Adapter` 允许定义两个不同的视图：一是在 `Spinner` 上显示数据，另一个是当 `Spinner` 按下时在下拉列表里显示数据。

公共方法

```
public abstract View getDropDownView (int position, View convertView,  
ViewGroup parent)
```

获得一个在指定位置上显示下拉弹出数据的视图。

参数

position 项目视图的索引

convertView 如果可能旧有的视图重新使用。注解：在使用之前应该检查这个视图不是空的并且类型合适。如果转换视图显示正确的数据是不可能的，这个方法能够创建一个新的视图

parent 视图最终将依附的父对象

返回值

一个对应指定位置的数据的视图

TabHost

译者署名: madgoat

译者链接: <http://madgoat.cn>

翻译时间: 2010-11-25

版本: Android 2.3 r1

结构

继承关系

```
public class TabHost extends FrameLayout implements  
ViewTreeObserver.OnTouchModeChangeListener
```

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.FrameLayout  
                android.widget.TabHost
```

类概述



提供选项卡（Tab 页）的窗口视图容器。此对象包含两个子对象：一组是用户可以选择指定 Tab 页的标签；另一组是 FrameLayout 用来显示该 Tab 页的内容。个别元素通常控制使用这个容器对象，而不是设置在子元素本身的值。

（译者注：即使使用的是单个元素，也最好把它放到容器对象 ViewGroup 里）

内部类

interface TabHost.OnTabChangeListener

接口定义了当选项卡更改时被调用的回调函数

interface TabHost.TabContentFactory

当某一选项卡被选中时生成选项卡的内容

class TabHost.TabSpec

单独的选项卡，每个选项卡都有一个选项卡指示符，内容和 tag 标签，以便于记录。

公共方法

public void addTab (TabHost.TabSpec tabSpec)

新增一个选项卡

参数

`tabSpec` 指定怎样创建指示符和内容.

public void clearAllTabs ()

从 tab widget 中移除所有关联到当前 tab host 的选项卡

public boolean dispatchKeyEvent (KeyEvent event)

分发按键事件到焦点传递路线上的下一视图。焦点传递路线从视图树的顶层开始一直到当前获取焦点的视图停止。如果此视图已经获取焦点,将分发给它自身。否则,将分发到焦点传递路线的下一节点。此方法会触发任何一个按键监听器.

(译者注: 关于 focus path,可以参考以下地址:

<http://blog.csdn.net/maxleng/archive/2010/05/04557758.aspx>)

参数

`event` 分发的按键事件

返回值

如果事件已经处理则返回 true,否则返回 false.

public void dispatchWindowFocusChanged (boolean hasFocus)

当窗口包含的此视图获取或丢失焦点时触发此方法.ViewGroups 应该重写以路由到他的子元素

参数

`hasFocus` 如果窗口包含的此 view 依获取焦点,返回 true,否则返回 false.

public int getCurrentTab ()

(译者注: 获取当前选项卡的 id)

public String getCurrentTabTag ()

(译者注: 当前选项卡的 Tag 标签内容)

public View getCurrentTabView ()

(译者注: 获取当前选项卡的视图 view)

public View getCurrentView ()

(译者注: 获取当前的视图 view)

public FrameLayout getTabContentView ()

获取保存 tab 内容的 FrameLayout

public TabWidget getTabWidget ()

(译者注: 根据系统规定的 id: tabs 来找到 TabWidget, 并返回, 注意, 这里的 ID 必须是 tabs。源代码中表示如下:

```
private TabWidget mTabWidget;  
mTabWidget=(TabWidget) findViewById(com.android.internal.R.id.tabs);
```

```
public TabHost.TabSpec newTabSpec (String tag)
```

获取一个新的 TabHost.TabSpec，并关联到当前 tab host

参数

tag 所需的选项卡标签(tag)

```
public void onTouchModeChanged (boolean isInTouchMode)
```

当触摸模式发生改变时调用的回调函数.

参数

isInTouchMode 如果视图结构当前处于触摸模式,返回 true,否则返回 false.

```
public void setCurrentTab (int index)
```

(译者注: 设置当前的选项卡

参数

Index 为当前选项卡的索引。)

```
public void setCurrentTabByTag (String tag)
```

(译者注: 根据选项卡的 Tab 标签来设置当前的选项卡

参数

tag 想要被设置为当前选项卡的 tag 标签值。)

```
public void setOnTabChangedListener (TabHost.OnTabChangeListener l)
```

注册一个回调函数，当选项卡中的任何一个 tab 的选中状态发生改变时调用。

(译者注: setCurrentTab(index)时会触发调用)

参数

l 将运行的回调函数

```
public void setup ()
```

如果使用 `findViewById()` 加载 TabHost，那么在新增一个选项卡 tab 之前，需要调用 `setup()`。然而，当你在一个 TabActivity 里使用 `getTabHost()` 获取 TabHost，你就不再需要调用 `setup()` 了。(译者注: 实现 tab 窗口的两种方法: 继承 activity 时，使用 `findViewById()` 查找 TabHost，然后调用 `setup()`; 继承 TabActivity，通过 `getTabHost()` 查找，此时不用调用 `setup()`) 例子:

```
mTabHost = (TabHost) findViewById(R.id.tabhost);
mTabHost.setup();
mTabHost.addTab(TAB_TAG_1, "Hello, world!", "Tab 1");
```

```
public void setup (LocalActivityManager activityGroup)
```

如果你使用 `setContent(android.content.Intent)`，那么当 `activityGroup` 用于加载本地 `activity` 之时，必须调用此方法。如果你拓展（继承）TabActivity 将自动调用 `setup()` 方法。

参数

activityGroup 用来为选项卡内容加载 activities 的 activityGroup

受保护方法

`protected void onAttachedToWindow ()`

当视图附加到窗口上时被调用。在这个点的表面进行绘制。注意此函数确保在 `onDraw(Canvas)` 之前调用，然而它可能在第一次执行 `onDraw` 之前的任何时间被调用——包括的 `onMeasure(int,int)` 的之前或之后。

`protected void onDetachedFromWindow ()`

当视图从窗口分离时被调用。在这个点的表面不再有画面绘制。

补充

文章链接

[史上最全的 Android 的 Tab 与 TabHost 讲解](#)

[Android UI 设计 Tab TabHost 标签页的使用](#)

[Android 控件之 TabHost Tab 页](#)

[动态 Tab 页](#)

示例代码



Java:

```
public class Tabs1 extends TabActivity {
```

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    TabHost tabHost = getTabHost();  
  
    LayoutInflator.from(this).inflate(R.layout.tabs1, tabHost.getTabContentView(),  
true);  
  
    tabHost.addTab(tabHost.newTabSpec("tab1")  
        .setIndicator("tab1")  
        .setContent(R.id.view1));  
    tabHost.addTab(tabHost.newTabSpec("tab3")  
        .setIndicator("tab2")  
        .setContent(R.id.view2));  
    tabHost.addTab(tabHost.newTabSpec("tab3")  
        .setIndicator("tab3")  
        .setContent(R.id.view3));  
}  
}
```

XML:

```
<?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">  
  
    <TextView android:id="@+id/view1"  
        android:background="@drawable/blue"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:text="@string/tabs_1_tab_1"/>  
    <TextView android:id="@+id/view2"  
        android:background="@drawable/red"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:text="@string/tabs_1_tab_2"/>  
    <TextView android:id="@+id/view3"  
        android:background="@drawable/green"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:text="@string/tabs_1_tab_3"/>  
</FrameLayout>
```

详细参见 apidemo/view/tabs1

TabHost.TabSpec

译者署名: madgoat

译者链接: <http://madgoat.cn>

翻译时间: 2010-11-25

版本: Android 2.3 r1

结构

继承关系

public class TabHost.TabSpec extends Object

java.lang.Object

 android.widget.TabHost.TabSpec

类概述

每个选项卡都有一个选项卡指示符, 内容和 tag 标签用于跟踪。这种生成器可以帮助从这些选项中做出选择。针对选项卡的指示符, 你可以选择:

- 1) 设置一个标题
- 2) 设置一个标题和图标

针对选项卡的内容, 你可以选择:

- 1) 视图的 ID
- 2) TabHost.TabContentFactory 创建的视图内容
- 3) 加载 Activity 的 Intent 对象

公共方法

public String getTag ()

(译者注: 获取 tag 标签字符串)

public TabHost.TabSpec setContent (int viewId)

为选项卡的内容指定视图的 ID

(译注: 即设定选项卡内容的视图)

public TabHost.TabSpec setContent (Intent intent)

指定一个加载 activity 的 Intent 对象作为选项卡内容

public TabHost.TabSpec setContent (TabHost.TabContentFactory contentFactory)

指定一个 TabHost.TabContentFactory 用于创建选项卡的内容

public TabHost.TabSpec setIndicator (CharSequence label)

指定一个标签作为选项卡指示符

public TabHost.TabSpec setIndicator (View view)

指定一个视图作为选项卡指示符

```
public TabHost.TabSpec setIndicator (CharSequence label, Drawable icon)
```

为选项卡指示符指定一个标签和图标

补充

文章链接

[TabHost 中的 TabSpec 的标题能不能缩小啊？](#)

[android 的 TabActivity](#)

TextView

农民伯伯

版本: Android 2.2

```
java.lang.Object  
    android.view.View  
        android.widget.TextView
```

直接子类:

Button, CheckedTextView, Chronometer, DigitalClock, EditText

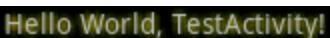
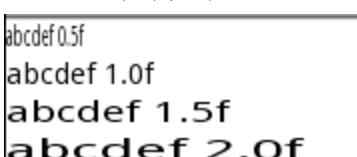
间接子类:

AutoCompleteTextView, CheckBox, CompoundButton, ExtractEditText,
MultiAutoCompleteTextView, RadioButton, ToggleButton

XML 属性

属性名称	描述
android:autoLink	设置是否当文本为 URL 链接/email/电话号码/map 时, 文本显示为可点击的链接。可选值 (none/web/email/phone/map/all)
android:autoText	如果设置, 将自动执行输入值的拼写纠正。此处无效果, 在显示输入法并输入的时候起作用。
android:bufferType	指定 getText() 方式取得的文本类别。选项 editable 类似于 StringBuilder 可追加字符, 也就是说 getText 后可调用 append 方法设置文本内容。spannable 则可在给定的字符区域使用样式, 参见 这里 1 、 这里 2 。
android:capitalize	设置英文字母大写类型。此处无效果, 需要弹出输入法才能看得到, 参见 EditText 此属性说明。
android:cursorVisible	设定光标为显示/隐藏, 默认显示。
android:digits	设置允许输入哪些字符。如 “1234567890.+-*%\\n()”
android:drawableBottom	在 text 的下方输出一个 drawable, 如图片。如果指定一个颜色的话会把 text 的背景设为该颜色, 并且同时和 background 使用时覆盖后者。
android:drawableLeft	在 text 的左边输出一个 drawable, 如图片。
android:drawablePadding	设置 text 与 drawable(图片)的间隔, 与 drawableLeft、drawableRight、drawableTop、drawableBottom 一起使用, 可设置为负数, 单独使用没有效果。
android:drawableRight	在 text 的右边输出一个 drawable, 如图片。
android:drawableTop	在 text 的正上方输出一个 drawable, 如图片。
android:editable	设置是否可编辑。这里无效果, 参见 EditText。
android:editorExtras	设置文本的额外的输入数据。在 EditText 再讨论。
android:ellipsize	设置当文字过长时, 该控件该如何显示。有如下值设

	置：“start”——省略号显示在开头；“end”——省略号显示在结尾；“middle”——省略号显示在中间；“marquee”——以 <u>跑马灯</u> 的方式显示(动画横向移动)
android:freezesText	设置保存文本的内容以及光标的位置。参见： 这里 。
android:gravity	设置文本位置，如设置成“center”，文本将居中显示。
android:hint	Text 为空时显示的文字提示信息，可通过 <code>textColorHint</code> 设置提示信息的颜色。此属性在 <code>EditView</code> 中使用，但是这里也可以用。
android:imeOptions	附加功能，设置右下角 IME 动作与编辑框相关的动作，如 <code>actionDone</code> 右下角将显示一个“完成”，而不设置默认是一个回车符号。这个在 <code>EditView</code> 中再详细说明，此处无用。
android:imeActionId	设置 IME 动作 ID。在 <code>EditView</code> 再做说明，可以先看这篇帖子： 这里 。
android:imeActionButton	设置 IME 动作标签。在 <code>EditView</code> 再做说明。
android:includeFontPadding	设置文本是否包含顶部和底部额外空白，默认为 true。
android:inputMethod	为文本指定输入法，需要完全限定名（完整的包名）。例如：com.google.android.inputmethod.pinyin，但是这里报错找不到。
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型。 在 <code>EditView</code> 中再详细说明，这里无效果。
android:marqueeRepeatLimit	在 <code>ellipsize</code> 指定 <code>marquee</code> 的情况下，设置重复滚动的次数，当设置为 <code>marquee_forever</code> 时表示无限次。
android:ems	设置 <code>TextView</code> 的宽度为 N 个字符的宽度。这里测试为一个汉字字符宽度，如图： 
android:maxEms	设置 <code>TextView</code> 的宽度为最长为 N 个字符的宽度。与 <code>ems</code> 同时使用时覆盖 <code>ems</code> 选项。
android:minEms	设置 <code>TextView</code> 的宽度为最短为 N 个字符的宽度。与 <code>ems</code> 同时使用时覆盖 <code>ems</code> 选项。
android:maxLength	限制显示的文本长度，超出部分不显示。
android:lines	设置文本的行数，设置两行就显示两行，即使第二行没有数据。
android:maxLines	设置文本的最大显示行数，与 <code>width</code> 或者 <code>layout_width</code> 结合使用，超出部分自动换行，超出行数将不显示。
android:minLines	设置文本的最小行数，与 <code>lines</code> 类似。
android:linksClickable	设置链接是否点击连接，即使设置了 <code>autoLink</code> 。
android:lineSpacingExtra	设置行间距。
android:lineSpacingMultiplier	设置行间距的倍数。如“1.2”
android:numeric	如果被设置，该 <code>TextView</code> 有一个数字输入法。此处无用，

	设置后唯一效果是 TextView 有点击效果，此属性在 EditText 将详细说明。
android:password	以小点“.”显示文本
android:phoneNumber	设置为电话号码的输入方式。
android:privateImeOptions	设置输入法选项，此处无用，在 EditText 将进一步讨论。
android:scrollHorizontalBy	设置文本超出 TextView 的宽度的情况下，是否出现横拉条。
android:selectAllOnFocus	如果文本是可选择的，让他获取焦点而不是将光标移动为文本的开始位置或者末尾位置。TextView 中设置后无效果。
android:shadowColor	指定文本阴影的颜色，需要与 shadowRadius 一起使用。 效果： 
android:shadowDx	设置阴影横向坐标开始位置。
android:shadowDy	设置阴影纵向坐标开始位置。
android:shadowRadius	设置阴影的半径。设置为 0.1 就变成字体的颜色了，一般设置为 3.0 的效果比较好。
android:singleLine	设置单行显示。如果和 layout_width 一起使用，当文本不能全部显示时，后面用“...”来表示。如 android:text="test_singleLine" android:singleLine="true" android:layout_width="20dp" 将只显示“t...”。如果不设置 singleLine 或者设置为 false，文本将自动换行
android:text	设置显示文本。
android:textAppearance	设置文字外观。如 “?android:attr/textAppearanceLargeInverse”这里引用的是系统自带的一个外观，? 表示系统是否有这种外观，否则使用默认的外观。可设置的值如下： textAppearanceButton/textAppearanceInverse/textAppearanceLarge/textAppearanceLargeInverse/textAppearanceMedium/textAppearanceMediumInverse/textAppearanceSmall/textAppearanceSmallInverse
android:textColor	设置文本颜色
android:textColorHighlight	被选中文字的底色，默认为蓝色
android:textColorHint	设置提示信息文字的颜色，默认为灰色。与 hint 一起使用。
android:textColorLink	文字链接的颜色。
android:textScaleX	设置文字缩放，默认为 1.0f。分别设置 0.5f/1.0f/1.5f/2.0f 效果如下： 
android:textSize	设置文字大小，推荐度量单位“sp”，如“15sp”

android:textStyle	设置字形[bold(粗体) 0, italic(斜体) 1, bolditalic(又粗又斜) 2] 可以设置一个或多个, 用“ ”隔开
android:typeface	设置文本字体, 必须是以下常量值之一: normal 0, sans 1, serif 2, monospace(等宽字体) 3]
android:height	设置文本区域的高度, 支持度量单位: px(像素)/dp/sp/in/mm(毫米)
android:maxHeight	设置文本区域的最大高度
android:minHeight	设置文本区域的最小高度
android:width	设置文本区域的宽度, 支持度量单位: px(像素)/dp/sp/in/mm(毫米), 与 layout_width 的区别看 这里 。
android:maxWidth	设置文本区域的最大宽度
android:minWidth	设置文本区域的最小宽度

TimePicker

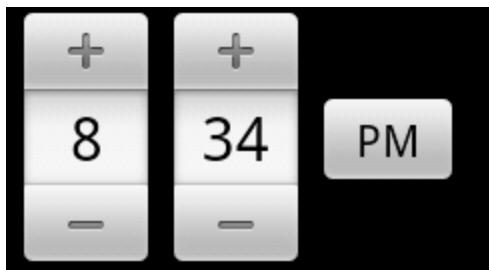
翻译人：桂仁

版本：Android 2.2 r1

public class TimePicker extends FrameLayout

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.FrameLayout  
                android.widget.TimePicker
```

概述



用于选择一天中时间的视图，支持 24 小时及上午/下午模式。小时，分钟及上午/下午（如果可用）都可以用垂直滚动条来控制。用键盘来输入小时。两个数的小时数可以通过输入两个数字来实现，例如在一定时间内输入 ‘1’ 和 ‘2’ 即选择了 12 点。分钟能显示输入的单个数字。在 AM/PM 模式下，用户可以输入'a', 'A"或 'p', 'P'来选取。对于对话框视图，参见 [TimePickerDialog](#)。

公共方法

public int getBaseline ()

返回窗口空间的文本基准线到其顶边界的偏移量。如果这个部件不支持基准线对齐，这个方法返回-1。

返回值

基准线的偏移量，如果不支持基准线对齐则返回-1。

public Integer getCurrentHour ()

获取当前时间的小时部分。

返回值

当前小时（0-23）

public Integer getCurrentMinute ()

获取当前时间的分钟部分。

返回值

当前分钟。

```
public boolean is24HourView ()
```

获取当前系统设置是否是 24 小时制。

返回值

如果是 24 小时制返回 true，否则返回 false。

```
public void setCurrentHour (Integer currentHour)
```

设置当前小时。

```
public void setCurrentMinute (Integer currentMinute)
```

设置当前分钟 (0-59)。

```
public void setEnabled (boolean enabled)
```

设置可用的视图状态。可用的视图状态的解释在子类中改变。

参数

enabled 如果可用为 true，否则为 false。

```
public void setIs24HourView (Boolean is24HourView)
```

设置是 24 小时还是上午/下午制。

参数

is24HourView True 表示 24 小时制. False 表示上午/下午制.

```
public void setOnTimeChangedListener (TimePicker.OnTimeChangedListener
```

```
onTimeChangedListener)
```

设置时间调整事件的回调函数。

参数

onTimeChangedListener 回调函数，不能为空。

受保护方法

```
protected void onRestoreInstanceState (Parcelable state)
```

允许一个视图回复到之前用 `onSaveInstanceState()` 保存的状态，`state` 参数不能为空。

参数

state 之前调用 `onSaveInstanceState()` 返回的状态。

```
protected Parcelable onSaveInstanceState ()
```

用来允许一个视图保存当前的内部状态，之后可以创建新的实例应用相同的状态。状态信息不能包含常量或在之后重新构造。例如，你永远不能保存在屏幕上的当前位置，因为当创建一个新的视图时，它将会被放置到它的层次结构中，它的位置会被重新计算。

你可以存储到这里的一些例子：一个文本框中当前光标的位置（但通常不是文字本身，文字通常保存在内容提供者(content provider)或其他持久的储存中），一个列表视图中的当前选中项。

返回值

返回一个包含视图当前状态的 `Parcelable` 对象，或没有什么状态保存时返回

null。默认实现返回 null。

补充

文章链接

[\[示例代码\]Hello, TimePicker](#)

[\[示例代码\]日期選擇器\(DatePicker\) 和時間選擇器\(TimePicker\)](#)

[Android TimePicker DatePicker 简单说明](#)

TimePicker.OnTimeChangedListener

翻译人：桂仁

版本：Android 2.2 r1

public static interface TimePicker.OnTimeChangedListener

android.widget.TimePicker.OnTimeChangedListener

间接子类

[TimePickerDialog](#)

概述

调整时间事件的回调接口。

公共方法

`public abstract void onTimeChanged (TimePicker view, int hourOfDay, int minute)`

参数

`view` 与监听相关的视图。

`hourOfDay` 当前小时

`minute` 当前分钟

Toast

译者署名: cnmahj、 jiahuibin
译者链接: <http://p.toolib.com/step>
翻译时间: 2010 年 11 月 2 日
版本: Android 2.2 r1

类结构

public class Toast extends Object

java.lang.Object
 android.widget.Toast

概述



Toast 是一种提供给用户简洁信息的视图。Toast 类帮助你创建和显示该信息。

该视图已浮于应用程序之上的形式呈现给用户。因为它并不获得焦点，即使用户正在输入什么也不会受到影响。它的目标是尽可能已不显眼的方式，使用户看到你提供的信息。有两个例子就是音量控制和设置信息保存成功。

使用该类最简单的方法就是调用一个静态方法，让他来构造你需要的一切并返回一个新的 Toast 对象。

常量

int LENGTH_LONG

持续显示视图或文本提示较长时间。该时间长度可定制。

参见

[setDuration\(int\)](#)

int LENGTH_SHORT

持续显示视图或文本提示较短时间。该时间长度可定制。该值为默认值。

参见

[setDuration\(int\)](#)

构造函数

public Toast (Context context)

构造一个空的 Toast 对象。在调用 `show()` 之前，必须先调用 `setView(View)`。

(译者注：只有使用 `setView(View)` 的时候，才使用 `new Toast(Context content)` 来得到 Toast 对象，否则必须用 `makeText()` 方法来创建 toast 对象，并且这种方式获得 Toast 对象不能使用 `setText()` 方法。)

参数

`context` 使用的上下文。通常是你的 Application 或 Activity 对象。

公共方法

`public int cancel ()`

如果视图已经显示则将其关闭，还没有显示则不再显示。一般不需要调用该方法。正常情况下，视图会在超过存续期间后消失。

`public int getDuration ()`

返回存续期间

请参阅

[setDuration \(int\)](#)

`public int getGravity ()`

取得提示信息在屏幕上显示的位置。

请参阅

[Gravity](#)

[setGravity \(\)](#)

`public float getHorizontalMargin ()`

返回横向栏外空白。

`public float getVerticalMargin ()`

返回纵向栏外空白。

`public View getView ()`

返回 `View` 对象。

请参阅

[setView \(View\)](#)

`public int getXOffset ()`

返回相对于参照位置的横向偏移像素量。

```
Toast msg = Toast.makeText(Main.this, "Message", Toast.LENGTH_LONG);
msg.setGravity(Gravity.CENTER, msg.getXOffset() / 2, msg.getYOffset() / 2);
msg.show();
```

`public int getYOffset ()`

返回相对于参照位置的纵向偏移像素量。

`public static Toast makeText (Context context, int resId, int duration)`

生成一个从资源中取得的包含文本视图的标准 `Toast` 对象。

参数

`context` 使用的上下文。通常是你的 [Application](#) 或 [Activity](#) 对象。

resId 要使用的字符串资源 ID，可以是已格式化文本。

duration 该信息的存续期间。值为 [LENGTH_SHORT](#) 或 [LENGTH_LONG](#)

异常

当资源未找到时抛异常 [Resources.NotFoundException](#)

```
public static Toast makeText (Context context, CharSequence text, int duration)
```

生成一个包含文本视图的标准 [Toast](#) 对象。

参数

context 使用的上下文。通常是你自己的 [Application](#) 或 [Activity](#) 对象。

resId 要显示的文本，可以是已格式化文本。

duration 该信息的存续期间。值为 [LENGTH_SHORT](#) 或 [LENGTH_LONG](#)

```
public void setDuration (int duration)
```

设置存续期间。

请参阅

[LENGTH_SHORT](#)

[LENGTH_LONG](#)

```
public void setGravity (int gravity, int xOffset, int yOffset)
```

设置提示信息在屏幕上的显示位置。

(译者注：自定义 [Toast](#) 的显示位置，例如 `toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0)`可以把 [Toast](#) 定位在左上角。[Toast](#) 提示的位置 `xOffset`:大于 0 向右移，小于 0 向左移

请参阅

[Gravity](#)

[getGravity\(\)](#)

```
public void setMargin (float horizontalMargin, float verticalMargin)
```

设置视图的栏外空白。

参数

horizontalMargin 容器的边缘与提示信息的横向空白(与容器宽度的比)。

verticalMargin 容器的边缘与提示信息的纵向空白(与容器高度的比)。

```
public void setText (int resId)
```

更新之前通过 `makeText()` 方法生成的 [Toast](#) 对象的文本内容。

参数

resId 为 [Toast](#) 指定的新的字符串资源 ID。

```
public void setText (CharSequence s)
```

更新之前通过 `makeText()` 方法生成的 [Toast](#) 对象的文本内容。

参数

s 为 [Toast](#) 指定的新的文本。

```
public void setView (View view)
```

设置要显示的 View 。

(译者注：注意这个方法可以显示自定义的 toast 视图，可以包含图像，文字等等。是比较常用的方法。)

请参阅

[getView\(\)](#)

public void show ()

按照指定的存续期间显示提示信息。

补充

文章链接

[让 Toast 一直显示的解决方法](#)

[通知 Toast 详细用法（显示 view）](#)

[Android 一种信息提示机制：Toast](#)

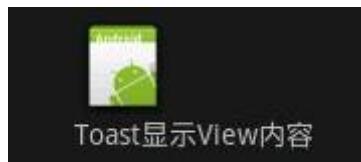
[\[推荐\] android Toast 大全（五种情形）建立属于你自己的 toast](#)

示例代码

示例一：使用图片的 Toast

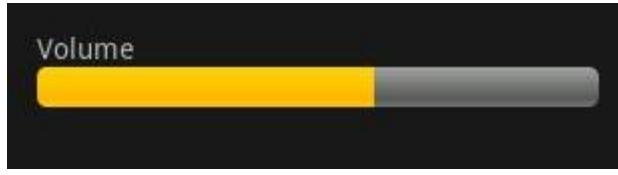
```
Toast toast = new Toast(this);
ImageView view = new ImageView(this);
view.setImageResource(R.drawable.icon);
toast.setView(view);
toast.show();
```

示例二：带文字带图片 Toast



```
private void showToast() {
    // 1 创建 Toast
    Toast toast = Toast.makeText(this, "图文显示", Toast.LENGTH_LONG);
    // 2 创建 Layout，并设置为水平布局
    LinearLayout mLayout = new LinearLayout(this);
    mLayout.setOrientation(LinearLayout.HORIZONTAL);
    ImageView mImage = new ImageView(this); // 用于显示图像的 ImageView
    mImage.setImageResource(R.drawable.icon);
    View toastView = toast.getView(); // 获取显示文字的 Toast View
    mLayout.addView(mImage); // 添加到 Layout
    mLayout.addView(toastView);
    // 3 关键，设置 Toast 显示的 View(上面生成的 Layout).
    toast.setView(mLayout);
    toast.show();
}
```

示例三：综合 Toast 例子



Main.xml

```
<?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"
    >
    <Button android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Toast显示View"
    />
    <Button android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Toast直接输出"
    />
    <Button android:id="@+id/button3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="VolumeToast应用"
    />
</LinearLayout>
```

Toast.xml

```
<?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"
    >
    <ImageView android:src="@drawable/toast"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
    <TextView android:id="@+id/tv1"
        android:text=""
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
</LinearLayout>
Volumetoast.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="280dp"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Volume"
    />
<ProgressBar
    android:id="@+id/progress"
    android:layout_width="280dp"
    android:layout_height="wrap_content"
    android:progress="50"
    android:max="100"
    style="?android:attr/progressBarStyleHorizontal"
    />
</LinearLayout>
```

Java 代码文件

```
public class toasttest extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button1=(Button)findViewById(R.id.button1);
        button1.setOnClickListener(bt1lis);
        Button button2=(Button)findViewById(R.id.button2);
        button2.setOnClickListener(bt2lis);
        Button button3=(Button)findViewById(R.id.button3);
        button3.setOnClickListener(bt3lis);
    }
    OnClickListener bt1lis=new OnClickListener(){
```

```
    @Override
    public void onClick(View v) {
        showToast();
    }

};

OnClickListener bt2lis=new OnClickListener(){
    @Override
    public void onClick(View v) {
        Toast.makeText(toasttest.this,"直接输出测试",
        Toast.LENGTH_LONG).show();
    }

};

OnClickListener bt3lis=new OnClickListener(){
    @Override
    public void onClick(View v) {
        showvolumeToast();
    }

};

public void showToast(){
    LayoutInflator
    li=(LayoutInflator)SystemService(Context.LAYOUT_INFLATER_SERVICE);
    View view=li.inflate(R.layout.toast,null);
    //把布局文件 toast.xml 转换成一个 view
    Toast toast=new Toast(this);
    toast.setView(view);
    //载入 view,即显示 toast.xml 的内容
    TextView tv=(TextView)view.findViewById(R.id.tv1);
    tv.setText("Toast 显示 View 内容");
    //修改 TextView 里的内容
    toast.setDuration(Toast.LENGTH_SHORT);
    //设置显示时间, 长时间 Toast.LENGTH_LONG, 短时间为
    Toast.LENGTH_SHORT,不可以自己编辑
    toast.show();
}

public void showvolumeToast() {
    // TODO Auto-generated method stub
    LayoutInflator
    li=(LayoutInflator)SystemService(Context.LAYOUT_INFLATER_SERVICE);
    View volumeView=li.inflate(R.layout.volumetoast,null);
```

```
((ProgressBar)volumeView.findViewById(R.id.progress)).setProgress(((ProgressBar)
volumeView.findViewById(R.id.progress)).getProgress() + 10);
    //这里还有点问题，就是 progress 的进度条不动，因为我们主要是想
给大家展示
    //Toast 的用法，这里就不深究了
    Toast volumeToast= new Toast(this);
    volumeToast.setGravity(Gravity.BOTTOM, 0, 100);
    volumeToast.setView(volumeView);
    volumeToast.setDuration(Toast.LENGTH_SHORT);
    volumeToast.show();
}
}
```

备注：我们的代码没有重复概述中的通过代码布局 toast 实现方法，我们是通过布局文件转换成 view 视图来实现复杂 toast，这是两种常用的方法，大家可以根据具体情况进行选择。

下载

/Files/over140/2010/11/demo_Toast.rar

不足之处

现象：当点击一个按钮 可以显示一个四秒的 toast，但是我要是连点两下就是 8 秒 三下十二秒

解决办法：只用一个 Toast，自己设置 toast.setText(), setDuration(); 之后无论多少次.show()都能马上更新显示，一会就消失了

TextSwitcher

译者署名: madgoat

译者链接: <http://madgoat.cn/>

版本: Android 2.2 r1

public class TextSwitcher extends ViewSwitcher

java.lang.Object

[android.view.View](#)

 android.view.ViewGroup

 android.widget.FrameLayout

 android.widget.ViewAnimator

 android.widget.ViewSwitcher

 android.widget.TextSwitcher

概述



ViewSwitcher 仅仅包含子类型 TextView。TextSwitcher 被用来使屏幕上的 label 产生动画效果。每当 setText(CharSequence)被调用时，TextSwitcher 使用动画方式将当前的文字内容消失并显示新的文字内容。

构造函数

public TextSwitcher (Context context)

创建一个新的空 TextSwitcher

参数

context 应用程序上下文

public TextSwitcher (Context context, AttributeSet attrs)

使用提供的 context 和 attributes 来创建一个空的 TextSwitcher

参数

context 应用程序环境

attrs 属性集合

公共方法

public void addView (View child, int index, ViewGroup.LayoutParams params)

根据指定的布局参数新增一个子视图

参数

- child 新增的子视图
- index 新增子视图的位置
- params 新增子视图的布局参数

抛出异常

IllegalArgumentException 当子视图不是一个 TextView 实例时

public void setCurrentText (CharSequence text)

设置当前显示的文本视图的文字内容。非动画方式显示。

参数

- text 需要显示的新文本内容

public void setText (CharSequence text)

设置下一视图的文本内容并切换到下一视图。可以动画的退出当前文本内容，显示下一文本内容。

参数

- text 需要显示的新文本内容

示例

摘自 API Demos->View->TextSwitcher

Java:

```
/*
 * Uses a TextSwitcher.
 */
public class TextSwitcher1 extends Activity implements ViewSwitcher.ViewFactory,
    View.OnClickListener {

    private TextSwitcher mSwitcher;
    private int mCounter = 0;

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

        setContentView(R.layout.text_switcher_1);

        mSwitcher = (TextSwitcher) findViewById(R.id.switcher);
        mSwitcher.setFactory(this);

        Animation in = AnimationUtils.loadAnimation(this,
            android.R.anim.fade_in);
        Animation out = AnimationUtils.loadAnimation(this,
            android.R.anim.fade_out);
```

```

        mSwitcher.setInAnimation(in);
        mSwitcher.setOutAnimation(out);

        Button nextButton = (Button) findViewById(R.id.next);
        nextButton.setOnClickListener(this);

        updateCounter();
    }

    public void onClick(View v) {
        mCounter++;
        updateCounter();
    }

    private void updateCounter() {
        mSwitcher.setText(String.valueOf(mCounter));
    }

    public View makeView() {
        TextView t = new TextView(this);
        t.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL);
        t.setTextSize(36);
        return t;
    }
}

```

Xml:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button android:id="@+id/next"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_switcher_1_next_text" />
    <TextSwitcher android:id="@+id/switcher"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>

```

ToggleButton

农民伯伯

版本: Android 2.2 r1

*public class **ToggleButton** extends **CompoundButton***

java.lang.Object

[android.view.View](#)

[android.widget.TextView](#)

[android.widget.Button](#)

[android.widget.CompoundButton](#)

[android.widget.ToggleButton](#)

概述



通过一个带有亮度指示同时默认文本为“ON”或“OFF”的按钮显示选中/未选中状态。

XML 属性

属性名称	描述
android:disabledAlpha	设置按钮在禁用时透明度。 disabledAlpha:未指定 disabledAlpha:0 disabledAlpha:0.1 disabledAlpha:0.5 disabledAlpha:1.0 disabledAlpha:1.1 disabledAlpha:2.0
android:textOff	未选中时按钮的文本
android:textOn	选中时按钮的文本

公共方法

```
public CharSequence getTextOff()
```

返回按钮未选中时的文本。

返回值

文本

```
public CharSequence getTextOn()
```

返回按钮选中时的文本。

返回值

文本

```
public void setBackgroundDrawable (Drawable d)
```

设置指定的可绘制（译者注：如图片）为背景，或删除背景。如果让背景有边距，这个视图的边距就是背景的边距。然而，当背景被删除时，这个视图的边距不能被触摸。如果需要设置边距，请使用方法 `setPadding(int, int, int, int)`。

（译者注：如果设置删除背景整个就不显示了，此外设置背景后选中和被选中的图片也



不显示了，如下图：

```
ToggleButton rb = (ToggleButton) findViewById(R.id.tb);
rb.setPadding(5, 0, 10, 0);
rb.setBackgroundDrawable(getResources().getDrawable(R.drawable.icon));
)
```

参数

d 设置可绘制（译者注：如图片）为背景，或设置为空删除背景。

```
public void setChecked (boolean checked)
```

改变按钮的选中状态。

参数

checked true 让按钮选中，false 让按钮不选中

```
public void setTextOff (CharSequence textOff)
```

设置按钮未选中时显示的文本。

参数

textOff 文本

```
public void setTextOn (CharSequence textOn)
```

设置按钮选中时显示的文本。

参数

textOn 文本

受保护方法

```
protected void drawableStateChanged ()
```

在视图状态的变化影响到所显示可绘制的状态时调用这个方法。

确保在覆盖时中调用父类方法（译者注：super.drawableStateChanged()）。

```
protected void onFinishInflate ()
```

XML 文件加载视图完成时调用。这个函数在加载的最后阶段被调用，所有的子视图已经被添加。

即使子类重写了 onFinishInflate 方法，也应该始终确保调用父类方法（译者注：super.onFinishInflate()），使系统能够调用。

TwoLineListItem

译者署名:loveshirui

译者链接: <http://heji.javaeye.com>

版本: Android 2.2 r1

public class TwoLineListItem extends RelativeLayout

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.RelativeLayout  
                android.widget.TwoLineListItem
```

概述



这个布局是用在 ListView 中的，有两个子 View。每一项有两个 ID 值为 text1 和 text2 的 TextView 的元素。也有一个 ID 是 selectIcon 的元素，这个元素可以是任何 View 的子类（最典型的就是 ImageView）。支持标准的布局资源文件（不包含所选择的 icon），但是你可以为这个对象自己定义 XML 布局文件。

XML 属性

属性名称	描述
android:mode	布局模式: oneLine 一行 collapsing 折叠 twoLine 两行

公共方法

Public TextView getText1 ()

返回 ID 为 Text1 的 TextView 对象

返回值

ID 为 Text1 的 TextView 对象

Public TextView getText2 ()

返回 ID 为 Text1 的 TextView 对象

返回值

ID 为 Text1 的 TextView 对象

受保护方法

```
protected void onFinishInflate ()
```

初始化 XML 布局。

ViewAnimator

译者署名： madgoat

译者链接：<http://madgoat.cn>

版本：Android 2.3 r1

public class ViewAnimator extends FrameLayout

java.lang.Object

[android.view.View](#)

 android.view.ViewGroup

 android.widget.FrameLayout

 android.widget.ViewAnimator

已知直接子类

ViewFlipper, ViewSwitcher

已知间接子类

ImageSwitcher, TextSwitcher

概述

[FrameLayout](#) 容器的基类，当进行视图切换时显示动画效果。

(译者注：此类不常用，常用其直接子类 ViewFlipper, ViewSwitcher 或间接子类 ImageSwitcher, TextSwitcher)

XML 属性

属性名称	描述
android:inAnimation	(译者注：设置 View 进入屏幕时候使用的动画)
android:outAnimation	(译者注：设置 View 离开屏幕时候使用的动画)

公共方法

public void addView (View child, int index, ViewGroup.LayoutParams params)

添加一个具有指定布局参数子视图。

参数

child 添加的子视图

index 添加的子视图所在的位置

params 设置子视图的布局参数

public int getBaseline ()

返回 widget 的文本基线到 widget 上边界的偏移量，如果当前 widget 不支持基线对齐，此方法返回-1

返回值

在 widget 界限内基线的偏移量，如果不支持基线对齐，返回-1

public View getCurrentView ()

返回与当前显示的子元素相应的视图

 返回值

 当前显示的视图

 参见

[getDisplayedChild\(\)](#)

public int getDisplayedChild ()

返回当前显示的子视图的索引

public Animation getInAnimation ()

返回被用来显示视图进入屏幕的动画

 返回值

 一个动画对象 animation, 如果没有设置的话, 返回一个 null

 参见

[setInAnimation\(android.view.animation.Animation\)](#)

[setInAnimation\(android.content.Context, int\)](#)

public Animation getOutAnimation ()

返回被用来显示视图离开屏幕的动画

 返回值

 一个动画对象 animation, 如果没有设置的话, 返回一个 null

 参见

[setOutAnimation\(android.view.animation.Animation\)](#)

[setOutAnimation\(android.content.Context, int\)](#)

public void removeAllViews ()

调用此方法从 ViewGroup 中移除所有的子视图

public void removeViewAt (int index)

移除组(group)中指定位置上的视图

 参数

 index 需要移除的视图在组中的位置

public void removeViewInLayout (View view)

在布局时移除一个视图。当你需要在 onLayout()中移除一个视图时, 此方法很有用。

 参数

 view 要从组中移除的视图

public void removeViews (int start, int count)

从组中移除指定范围的视图

 参数

 start 要移除的多个视图在组中的开始位置

 count 移除视图的数量

```
public void removeViewsInLayout (int start, int count)
```

在布局时移除一组视图。当你需要在 `onLayout()` 中移除很多视图时，此方法很有用。

参数

start 移除的多个视图在组中第一个视图的索引

count 移除视图的数量

```
public void setAnimateFirstView (boolean animate)
```

设置当前视图在首次加载时是否动画显示。

参数

animate 在第一次显示当前视图时，是否动画显示

```
public void setDisplayedChild (int whichChild)
```

设置哪个子视图将被显示出来

参数

whichChild 将要显示的子视图的索引

```
public void setInAnimation (Animation inAnimation)
```

设置视图进入屏幕时使用的动画。

参数

inAnimation 视图进入屏幕时使用的动画

参见

[getInAnimation\(\)](#)

[setInAnimation\(android.content.Context, int\)](#)

```
public void setInAnimation (Context context, int resourceId)
```

设置视图进入屏幕时使用的动画。

参数

context 上下文

resourceID 动画的资源 id

参见

[getInAnimation\(\)](#)

[setInAnimation\(android.view.animation.Animation\)](#)

```
public void setOutAnimation (Animation outAnimation)
```

设置视图退出屏幕时使用的动画

参数

outAnimation 当视图退出屏幕时开始的动画

参见

[getOutAnimation\(\)](#)

[setOutAnimation\(android.content.Context, int\)](#)

```
public void setOutAnimation (Context context, int resourceId)
```

设置视图退出屏幕时使用的动画

参数

context 上下文
resourceID 动画的资源 id

参见

[getOutAnimation\(\)](#)
[setOutAnimation\(android.view.animation.Animation\)](#)

public void showNext ()

手动显示下一个子视图

public void showPrevious ()

手动显示上一个子视图

补充

文章精选

[Android-自定义切换视图一](#)

[Ophone 2D UI 动画教程之二——屏幕切换效果实现](#)

ViewFlipper

译者署名: ivanlee
版本: Android 2.2 r1

结构

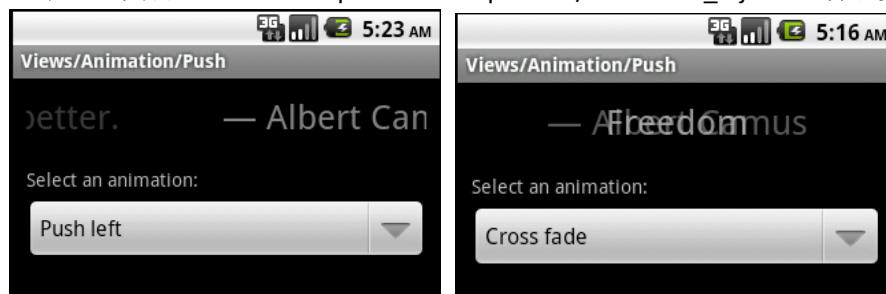
继承关系

`public class ViewFlipper extends ViewAnimator`

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.FrameLayout
                android.widget.ViewAnimator
                    android.widget.ViewFlipper
```

类概述

被添加到 ViewFlipper 中的两个或两个以上的视图之间将执行一个简单的 ViewAnimator 动画。一次仅能显示一个子视图。如果需要，可以设置间隔时间使子视图像幻灯片一样自动显示。(译者注: com.example.android.apis.view/Animation_2.java 包含该类示例程序)



XML 属性

属性名称	描述
<code>android:autoStart</code>	当设为 true 时，自动启动动画 此时必须是一个布尔值，属性值为 true 或 false (对应于全局资源属性 R.attr.autoStart)
<code>android:flipInterval</code>	显示下一个视图的时间间隔

公共方法

`public bool isAutoStart ()`

如果视图显示到窗口上时会自动调用 `startFlipping()` 方法，则返回 true

`public bool isFlipping()`

如果子视图正在切换，则返回 true

`public bool setAutoStart (bool autoStart)`

设置视图显示到窗口上时是否会自动调用 `startFlipping()` 方法

```
public bool setFlipInterval (int milliseconds)
```

视图间切换的时间间隔

参数

milliseconds 毫秒数

```
public bool startFlipping ()
```

开始在子视图间定时循环切换

```
public bool stopFlipping ()
```

停止切换

补充

文章精选

[ViewFlipper 的使用](#)

[android 开发 ViewFlipper 触摸动画](#)

[Android 固定物件的進出動畫 \(入場/出場\) - ViewFlipper](#) (blogspot)

[android 手势翻页效果](#)

ViewSwitcher

译者署名： ivanlee
版本： Android 2.3 r1

结构

继承关系

public class ViewSwitcher extends ViewAnimator

```
java.lang.Object
    android.view.View
        android.view.ViewGroup
            android.widget.FrameLayout
                android.widget.ViewAnimator
                    android.widget.ViewSwitcher
```

已知直接子类

ImageSwitcher, TextSwitcher

类概述

在两个视图间转换时显示动画，有一个可以创建这些视图的工厂类。你可以用工厂来创建这些视图，也可以自己创建。一个 ViewSwitcher 只允许包含两个子视图，且一次仅能显示一个。

（译者注：与 ViewFlipper 类相似，但该类不常用，常用其两个子类 ImageSwitcher：转换图片时增加动画效果；TextSwitcher：转换文字时增加动画效果；其实例见 apidemos 中 ImageSwitcher 实例和 TextSwitcher 实例）

内部类

interface ViewSwitcher.ViewFactory

在一个 ViewSwitcher 里创建视图

构造函数

public ViewSwitcher (Context context)

构造一个新的空的视图转换器(ViewSwitcher)。

参数

context 应用环境（译者注：应用程序上下文）

public ViewSwitcher (Context context, AttributeSet attrs)

构造一个指定上下文、属性集合的空的视图转换器(ViewSwitcher)。

参数

context 应用环境（译者注：应用程序上下文）

attrs 属性集合

公共方法

```
public void addView(View child, int index, ViewGroup.LayoutParams params)
```

添加一个指定布局参数的子视图

参数

child 添加的子视图

index 添加的子视图的索引

params 子视图的布局参数

异常

IllegalStateException 如果切换器中已经包含了两个视图时。

```
public View getNextView ()
```

返回下一个要显示的视图

返回

视图切换之后将要显示出的下一个视图

```
public void reset ()
```

重置视图转换器(ViewSwitcher) 来隐藏所有存在的视图，并使转换器达到一次动画都还没有播放的状态。

```
public void setFactory (ViewSwitcher.ViewFactory factory)
```

设置用来生成将在视图转换器中切换的两个视图的工厂。也可以调用两次

addView(android.view.View, int, android.view.ViewGroup.LayoutParams)来替代使用工厂的方法。

参数

factory 用来生成转换器内容的视图工厂

补充

文章精选

[android UI ViewSwitcher 的使用（续二）](#)

[Android: Don't Overlook ViewSwitcher](#)

[Using a ViewSwitcher in your Android xml layouts](#)

ViewSwitcher.ViewFactory

译者署名: ivanlee

版本: Android 2.3 r1

结构

继承关系

public static interface ViewSwitcher.ViewFactory

android.widget.ViewSwitcher.ViewFactory

类概述

在视图转换器(ViewSwitcher)中创建视图

(译者注: ImageSwitcher, TextSwitcher 常实现此接口来生成视图)

公共方法

public abstract View makeView ()

创建一个用于添加到视图转换器(ViewSwitcher)中的新视图

返回值

a View 一个视图

ZoomButtonsController

译者署名: 獨鈞躊躇

译者链接: <http://www.cnblogs.com/mxgsa/>

版本: Android 2.2 r1

结构

继承关系

public class ZoomButtonsController extends View implements View.OnTouchListener

java.lang.Object
 android.widget.ZoomButtonsController

类概述

ZoomButtonsController 处理缩放控件的显示和隐藏并且定位其在相关父视图的位置。他也可以做为缩放控件的容器，允许在缩放控制窗口里面显示一些附加的按钮。

通常情况下，客户端在按下或者移动显示容器需要调用 `setVisible(true)`方法(不需要调用 `setVisible(false)`，来隐藏空间，因为当时间超时，它会自动隐藏)，同时，当拥有者不能再进一步缩放的时候，客户端应该调 `setZoomInEnabled(boolean)` 和 `setZoomOutEnabled(boolean)` 来及时更新。

如果你需要和自定义视图搭配使用，请在 [onDetachedFromWindow\(\)](#) 方法中调用 [setVisible\(false\)](#)。

构造函数

public ZoomButtonsController (View ownerView)

ZoomButtonsController 的构造函数

参数

`ownerView` 被缩放控件进行缩放的可视控件， 缩放控件显示将和可视控件保持一致

公共方法

public ViewGroup getContainer ()

获取缩放控件的父容器。

客户端可以增加其他的可视控件，和缩放控件一起放到这个容器中

返回

缩放控件的容器，它的布局将和它子控件的布局保持一致

public View getZoomControls ()

获取缩放控件的视图

返回

缩放控件视图

public boolean isAutoDismissed ()

获取缩放控件是否显示后自动关闭

返回

缩放控件是否显示后自动关闭

public boolean isVisible ()

缩放控件是否对用户可视

返回

true 或者 false

public void setAutoDismiss (boolean autoDismiss)

设置缩放控件是否显示后自动关闭

参数

AutoDismiss true 或者 false

public void setFocusable (boolean focusable)

设置缩放控件是否获取焦点，如果控件获取焦点，就是可以使用轨迹球和方向键进行操作，否则，只能触摸进行操作

参数

focusable True 和 false

public void setOnZoomListener (ZoomButtonsController.OnZoomListener listener)

设置 ZoomButtonsController.OnZoomListener 倾听接收回调进行缩放

参数

listener 介绍是否缩放的监听器

public void setVisible (boolean visible)

设置缩放控件是否对用户可视

参数

visible 缩放控件是否对用户可视 (true 或者 false)

public void setZoomInEnabled (boolean enabled)

是否允许空间放大

参数

enabled 是否允许空间放大 (true 或者 false)

public void setZoomOutEnabled (boolean enabled)

是否允许控件缩小

参数

enabled 是否允许控件缩小 (true 或者 false)

public void setZoomSpeed (long speed)

设置用户操作缩放按钮到缩放回调的延迟时间

参数

speed 以毫秒为单位的缩放回调的间隔时间

补充

文章精选

[Android Google map 使用心得](#)

[googlemap 加载多个 overlay 内存溢出](#)

示例代码

```
//初始化一个 ZoomButtonsController
ZoomButtonsController zoomctrl = mapView.getZoomButtonsController();
//设置可获取焦点
zoomctrl.setFocusable(true);
//设置允许放大
zoomctrl.setZoomInEnabled(true);
//设置允许缩小
zoomctrl.setZoomOutEnabled(true);
//设置缩放的速度
zoomctrl.setZoomSpeed(1000);
//自动隐藏关闭
zoomctrl.setAutoDismissed(false);
//设置控件可视
zoomctrl.setVisible(true);
//设置缩放事件的监视器
zoomctrl.setOnZoomListener(new ZoomButtonsController.OnZoomListener() {
    //缩放
    public void onZoom(boolean zoomIn) {
        // TODO Auto-generated method stub
    }

    //显示状态改变
    public void onVisibilityChanged(boolean visible) {
        // TODO Auto-generated method stub
    }
});
```

ZoomButtonsController.OnZoomListener

译者署名: 獨鋼躊躇

译者链接: <http://www.cnblogs.com/mxgsa/>

版本: Android 2.2 r1

结构

继承关系

public static interface ZoomButtonsController.OnZoomListener

android.widget.ZoomButtonsController.OnZoomListener

概述

当用户执行了互动或者触发了一些行为的接口, 例如缩放。

```
// 设置缩放事件的监视器
zoomctrl.setOnZoomListener(new ZoomButtonsController.OnZoomListener() {
    // 缩放
    public void onZoom(boolean zoomIn) {
        // TODO Auto-generated method stub
    }

    // 显示状态改变
    public void onVisibilityChanged(boolean visible) {
        // TODO Auto-generated method stub
    }
});
```

公共方法

public abstract void onVisibilityChanged (boolean visible)

当缩放控件的显示发生变化时调用

参数

visible 缩放控件是否可见

public abstract void onZoom (boolean zoomIn)

当该控件的拥有者需要被缩放时候调用

参数

zoomIn 缩放的方向: true 是放大, false 是缩小

ZoomButton

农民伯伯

版本: Android 2.2

```
public class ZoomButton extends ImageButton implements View.OnLongClickListener
```

java.lang.Object

 android.view.View

 android.widget.ImageView

 android.widget.ImageButton

 android.widget.ZoomButton

概述



缩放按钮，实际上是很普通的两个按钮，可以简单的理解为一个 ImageButton 加上一张缩放的图标。如下代码：

```
<ZoomButton android:id="@+id/zoomIn"
    android:background="@android:drawable/btn_plus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<ZoomButton
    android:id="@+id/zoomOut"
    android:background="@android:drawable/btn_minus"
    android:layout_width="wrap_content"
    android:layout height="wrap content" />
```

公共方法

```
public boolean dispatchUnhandledMove (View focused, int direction)
```

对于获得焦点的 View，这个方法是捕获箭头事件最后的机会。这就是在获取焦点的 View 没有在内部处理、系统在要求的方向也不能找到一个新的 View 让其获得焦点时调用。

参数

focused 当前焦点 View

direction 焦点移动的方向。其中之一:FOCUS_UP, FOCUS_DOWN, FOCUS_LEFT 和 FOCUS_RIGHT.

返回值

如果为 true，将清除这个 View 未处理的事件。

(注：从源码中可看出 ZoomButton 覆盖了父类的该方法，在 super 之前调用了一下 clearFocus，如下代码：

```
@Override  
public boolean dispatchUnhandledMove(View focused, int direction) {  
    clearFocus();  
    return super.dispatchUnhandledMove(focused, direction);  
}
```

`public boolean onKeyDown (int keyCode, KeyEvent event)`

默认实现至 `KeyEvent.Callback.onKeyMultiple()` : 当点击执行时 `KEYCODE_DPAD_CENTER` 或 `KEYCODE_ENTER` 被释放。

参数

`keyCode` 按下按钮代表的键值, 属于 `KeyEvent` (注: `KeyEvent` 的静态属性)。

`event` 该 `KeyEvent` 对象, 定义按钮动作

返回值

如果您处理这一事件中, 返回 `true`。如果你想允许事件被下一个接收器处理, 返回 `false`。

`public boolean onLongClick (View v)`

当一个 `View` 被长按时调用。

参数

`v` 被长按的 `View`。

返回值

如果返回 `true`, 这个回调在长按时被执行了, 反之返回 `false`.

`public boolean onTouchEvent (MotionEvent event)`

实现这个方法处理触摸屏移动事件。

参数

`event` 该移动事件.

返回值

如果是 `true`, 该事件是触摸, 反之返回 `false`。

`public void setEnabled (boolean enabled)`

设置这个 `View` 启用状态。

参数

`enabled` 设置 `true` 表示启用, 反之表示禁用。

`public void setZoomSpeed (long speed)`

(注: 单独使用无实际意义, 可以用来存放临时数据)

ZoomControls

译者署名: jiahuibin

联系译者: huibin.jia@gmail.com

2010-10-22

版本: Android 2.2 r1

一、类结构

public class ZoomControls extends LinearLayout

```
java.lang.Object  
    android.view.View  
        android.view.ViewGroup  
            android.widget.LinearLayout  
                android.widget.ZoomControls
```

二、概述



ZoomControl 显示一个简单的设置来控制缩放并回调已注册的事件。

三、公共方法

public boolean hasFocus ()

如果这个视图获得了焦点就返回真。

返回值

如果这个视图获得了焦点就返回真。

public void hide ()

这个方法可以将 zoomControl 视图隐藏起来，不显示。

public boolean onTouchEvent (MotionEvent event)

这个方法处理触摸屏移动事件。

参数

event 该移动事件.

返回值

如果是 true, 该事件是触摸, 反之返回 false。

public void setIsZoomInEnabled (boolean isEnabled)

这个方法可以设置放大按钮是否可用。

参数

isEnabled 如果是 true, 放大按钮可用, 反之不可用 (按钮变成灰色)。

```
public void setIsZoomOutEnabled (boolean isEnabled)
```

这个方法可以设置缩小按钮是否可用。

参数

isEnabled 如果是 true 缩小按钮可用，反之不可用（按钮变成灰色）。

```
public void setOnZoomInClickListener (View.OnClickListener listener)
```

注册放大监听器

参数

Listener 点击放大按钮事件触发的事件。（译者注：为 OnClickListener 的实例，可以复写 onClick 方法，里面为具体的响应动作。我感觉这个方法还有下面的方法是本类的重点，我们可以在里面实现相应的动作，来响应我们的事件。）

```
public void setOnZoomOutClickListener (View.OnClickListener listener)
```

注册缩小监听器

参数

Listener 点击缩小按钮事件触发的事件。（译者注：为 OnClickListener 的实例，可以复写 onClick 方法，里面为具体的响应动作。）

```
public void setZoomSpeed (long speed)
```

设置缩放速度。

参数

Speed 缩放速度

```
public void show ()
```

这个方法和 hide 方法对应，用来显示 ZoomControl。

四、代码

声明：这个代码部分思路来源于网络上一个博客，谨以此来解释，特此声明。

```
package com.jhb.zoomcontroltest;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;
import android.widget.ZoomControls;
```

```
public class ZoomCotroltest extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
```

```

        setTitle("ZoomControls");

        ZoomControls zoomControls = (ZoomControls)
this.findViewById(R.id.zoomControls);
        zoomControls.hide();
        zoomControls.show();
        // setOnZoomInClickListener() - 响应单击放大按钮的事件
        zoomControls.setIsZoomInEnabled(true);
        zoomControls.setIsZoomOutEnabled(true);
        zoomControls.setOnZoomInClickListener(new
OnClickListener() {
            public void onClick(View v) {
                Toast.makeText(ZoomControltest.this, "单击了
放大按钮", Toast.LENGTH_SHORT).show();
            }
        });
        // setOnZoomOutClickListener() - 响应单击缩小按钮的事件
        zoomControls.setOnZoomOutClickListener(new
OnClickListener() {
            public void onClick(View v) {
                Toast.makeText(ZoomControltest.this, "单击了
缩小按钮", Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

它对应的布局文件是下面这样的：

```

<?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">
    <!--
        放大/缩小按钮控件
    -->
    <ZoomControls android:id="@+id/zoomControls"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </ZoomControls>
</LinearLayout>

```

